Universiteit
Leiden
The Netherlands

DNA expressions : a formal notation for DNA

Vliet, R. van

Citation
Vliet, R. van. (2015, December 10). *DNA expressions : a formal notation for DNA*. *IPA Dissertation Series*. Retrieved from https://hdl.handle.net/1887/37052

Version:	Not Applicable (or Unknown)
License:	Leiden University Non-exclusive license
Downloaded from:	https://hdl.handle.net/1887/37052

Note: To cite this publication please use the final published version (if applicable).

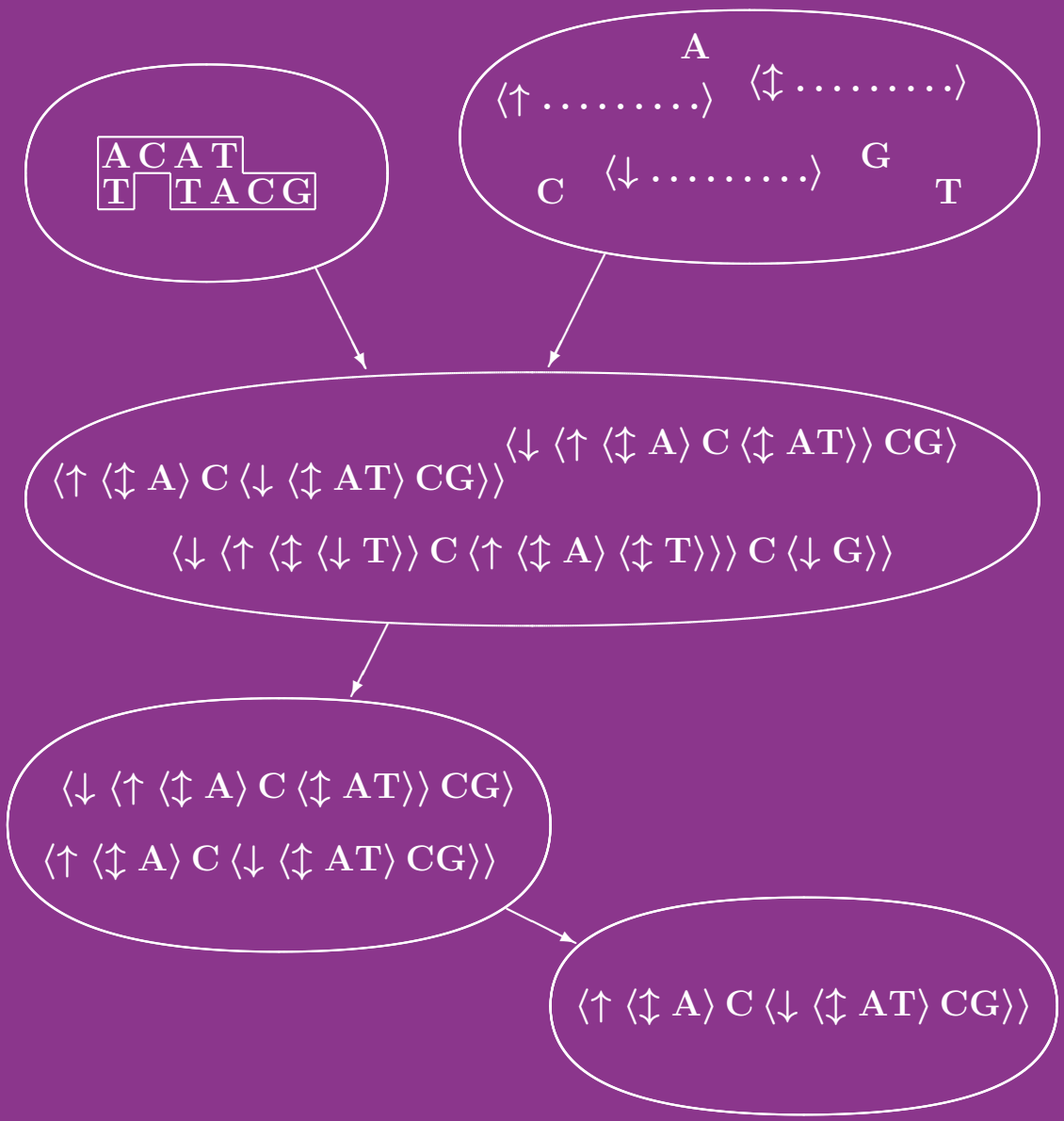
Cover Page

Universiteit Leiden

The handle http://hdl.handle.net/1887/37052 holds various files of this Leiden University dissertation.

Author: Vliet, Rudy van
Title: DNA expressions : a formal notation for DNA
Issue Date: 2015-12-10

DNA Expressions
A Formal Notation for DNA



Rudy van Vliet

DNA Expressions
A Formal Notation for DNA

Rudy van Vliet

The work described in this thesis has been carried out under the auspices of the research school IPA (Institute for Programming research and Algorithmics).

© 2015 Rudy van Vliet, except for the Calvin and Hobbes comic strip
Typeset using LaTeX
Printing: Ridderprint BV
Printed on BalancePure® recycled paper
ISBN: 978-94-6299-254-2
IPA Dissertation Series 2015-23

Despite the effort put into the careful writing of this thesis, it is inevitable that it contains errors. Errors detected can be reported to the author at *rvvliet@liacs.nl*. He will maintain a list of errata at his website on DNA expressions, which is currently to be found at

`http://www.liacs.leidenuniv.nl/~vlietrvan1/dnaexpressions/`

The first report of any indisputable error will be rewarded for with €0.10 and an honourable mention in the list of errata.

DNA Expressions
A Formal Notation for DNA

Proefschrift

ter verkrijging van
de graad van Doctor aan de Universiteit Leiden,
op gezag van Rector Magnificus prof. mr C.J.J.M. Stolker,
volgens besluit van het College voor Promoties
te verdedigen op donderdag 10 december 2015
klokke 12.30 uur

door

Rudy van Vliet
geboren te Alphen aan den Rijn
in 1969

Promotiecommissie

Promotor:	prof. dr J.N. Kok	
Copromotor:	dr H.J. Hoogeboom	
Overige leden:	prof. dr N. Jonoska	(University of South Florida)
	dr R. Brijder	(Universiteit Hasselt)
	prof. dr H.P. Spaink	
	prof. dr T.H.W. Bäck	

© 1992 Watterson.
Dist. by UNIVERSAL UCLICK. All rights reserved.

Contents

1 Introduction **1**
1.1 Background of the thesis 1
1.2 Contribution of the thesis 2
1.3 Set-up of the thesis 4
1.4 Resulting publications 5

2 Preliminaries **9**
2.1 Strings, trees, grammars, relations and complexity 9
2.2 DNA molecules 19
2.3 DNA computing 25
2.3.1 Splicing systems 25
2.3.2 Adleman's experiment 26

I DNA Expressions in General **31**

3 Formal DNA Molecules **33**
3.1 $\mathcal{N}$-words 33
3.2 Definition of formal DNA molecules 33
3.3 Components of a formal DNA molecule 36
3.4 Properties, relations and functions of formal DNA molecules 39

4 DNA Expressions **43**
4.1 Operators and DNA expressions 43
4.2 Brackets, arguments and DNA subexpressions 51
4.3 Recognition of DNA expressions 54
4.4 Computing the semantics of a DNA expression 58
4.5 A context-free grammar for $\mathcal{D}$ 67
4.6 The structure tree of a DNA expression 73
4.7 Equivalent DNA expressions 75

5 Basic Results on DNA Expressions **79**
5.1 Expressible formal DNA molecules 79
5.2 Nick free DNA expressions 82
5.3 Some equivalences 83

II Minimal DNA Expressions **99**

6 The Length of a DNA Expression **101**

6.1 The operators in a DNA expression . . . 101
6.2 Blocks of components of a formal DNA molecule . . . 103
6.3 Lower bounds for the length of a DNA expression . . . 124

7 The Construction of Minimal DNA Expressions 137
7.1 Minimal DNA expressions for a nick free formal DNA molecule . . . 138
7.2 Minimal DNA expressions for a formal DNA molecule with nick letters . . . 171

8 All Minimal DNA Expressions 183
8.1 Reverse construction of a minimal DNA expression . . . 183
8.2 Operator-minimal $\updownarrow$-expressions . . . 200
8.3 Characterization of minimal DNA expressions . . . 204
8.4 The structure tree of a minimal DNA expression . . . 215
8.5 The number of (operator-)minimal DNA expressions . . . 217

9 An Algorithm for Minimality 237
9.1 The algorithm and its correctness . . . 237
9.1.1 The procedure `Make`$\updownarrow$`ExprMinimal` . . . 255
9.1.2 The procedure `Denickify` . . . 262
9.1.3 The procedure `RotateToMinimal` . . . 271
9.2 The algorithm for an example . . . 274
9.3 Detailed implementation and complexity of the algorithm . . . 284
9.4 Decrease of length by the algorithm . . . 302

III Minimal Normal Form 311

10 A Minimal Normal Form for DNA Expressions 313
10.1 Definition of the minimal normal form . . . 314
10.2 Characterization of the minimal normal form . . . 317
10.3 The structure tree of a DNA expression in minimal normal form . . . 324
10.4 Regularity of $\mathcal{D}_{\mathrm{MinNF}}$ . . . 325

11 Algorithms for the Minimal Normal Form 341
11.1 Recursive algorithm for the minimal normal form . . . 341
11.2 Two-step algorithm for the minimal normal form . . . 348
11.3 Implementation and complexity of the algorithm . . . 354

12 Conclusions and Directions for Future Research 367

Samenvatting 369

Over de Auteur 375

Dankwoord 377

Bibliography 379

List of Symbols **383**

Index **385**

Titles in the IPA Dissertation Series since 2009 **393**

Chapter 1

Introduction

This thesis describes DNA expressions, a formal notation for DNA molecules that may contain nicks and gaps. In this chapter, we first sketch the background of this research, the field of DNA computing. We subsequently describe our contribution to the field, and give an outline of the thesis. We finally list the publications that have resulted from this thesis work.

1.1 Background of the thesis

Natural computing is the field of research that, on the one hand, investigates ways of computing inspired by nature, and, on the other hand, analyses computational processes occurring in nature, see [Rozenberg et al., 2012]. Sources of inspiration are, a.o., the organization of neurons in the brain, the operation of cells of organisms in general, and also the crucial role for life of DNA.

Since the discovery of the structure and function of DNA molecules, DNA has been (and still is) intensively studied by biologists and biochemists. In the course of time, also computer scientists became interested in DNA. For example, the evolution of DNA over the generations inspired researchers to develop *evolutionary algorithms*, which is one branch of natural computing.

The probably best-known subbranch of evolutionary algorithms is formed by the *genetic algorithms*. In a genetic algorithm, possible solutions to a problem are encoded as strings (as analogues of DNA molecules). In an iterative process, the algorithm maintains a population of these strings. Every iteration, the strings are evaluated and the 'better' strings in the population are selected to produce a next generation by operations resembling recombination and mutation. This way, good solutions to the problem are obtained, see, e.g., [Holland, 1975] and [Whitley & Sutton, 2012].

Another area where the study of DNA and computer science meet, is *DNA computing*, which is also a branch of natural computing. In this field, it is investigated how DNA molecules themselves can be used to perform computations. That is, instead of mimicking the 'behaviour' of DNA by software on silicon, the DNA molecules serve as the hardware that really do the work. Also models to describe these computations are studied.

The formal study of computational properties of DNA really began when Tom Head [1987] defined formal languages consisting of strings that can be modified by operations based on the way that restriction enzymes process DNA molecules. Theoretical computer scientists explored the generative power and other properties of such languages, see, e.g., [Kari et al., 1996] and [Head et al., 1997].

The interest of the computer science community in the computational potential of DNA was boosted, when Leonard Adleman [1994] demonstrated that (real, physical) DNA molecules can in principle be used to solve computationally hard problems. He performed an experiment in a biolab that solved a small instance of the directed Hamiltonian path problem using DNA, enzymes and standard biomolecular operations.

Since then, research on DNA computing is flourishing. Researchers from various disciplines, ranging from theoretical computer science to molecular biology, investigate the computational power of DNA molecules, both from a theoretical and an experimental point of view. Research groups from all over the world operate in this field, as is illustrated by the contributions to the annual conference on DNA Computing and Molecular Programming. For the latest two editions of this conference, see [Murata & Kobayashi, 2014] and [Phillips & Yin, 2015].

Initially, people even envisioned a universal DNA-based computer, i.e., a machine that takes a program encoded in DNA as an input, and carries out that program using (other) DNA molecules, in the same way that ordinary, electronic computers carry out programs, see, e.g., [Kari, 1997]. Nowadays, the applications of DNA computing and other types of molecular programming that are investigated are more specific. Current topics of interest include, a.o., gene assembly in ciliates, DNA sequence design, self-assembly and nanotechnology, see, e.g., [Ehrenfeucht et al., 2004], [Kari et al., 2005], [Winfree, 2003], [Zhang & Seelig, 2011], and [Chen et al., 2006]. The basic concepts of DNA computing are described in [Păun et al., 1998] and [Kari et al., 2012].

We conclude this section with two remarkable examples of DNA nanotechnology. [Rothemund, 2006] reports on a method (called 'scaffolded DNA origami') to create nanoscale shapes and patterns from DNA. With this method, a long single-stranded DNA molecule (the scaffold) folds into a given shape, when combined with carefully designed short pieces of DNA. Some of the shapes that Rothemund formed in his experiments in the lab were stars, triangles, and smiley faces.

[Gu et al., 2010] describes the operation of a nanoscale assembly line built of DNA. One DNA molecule (the 'walker') traverses a track provided by a second DNA molecule, and on its way, picks up nanoparticles ('cargo') donated by three different DNA-based machines. Each DNA machine carries a specific type of particle. As the machines can be programmed independently either to donate particles or not, the assembly line can be used to produce eight ($= 2^3$) distinct products.

1.2 Contribution of the thesis

Much research in the field of DNA computing concerns questions like what kind of DNA molecules (or other types of molecules) can be constructed, and what these molecules may be used for. As the DNA origami and assembly line from the previous section demonstrate, DNA turns out to have unexpected applications. Less attention is paid in the literature to formal ways to denote DNA molecules. Some examples are [Schroeder & Blattner, 1982], [Boneh et al., 1996] and [Deaton et al., 1999].

An advantage of formal notations over more verbal descriptions, is that the former are shorter and more precise. They do not give rise to ambiguities, e.g., as to which DNA molecules are actually meant. They can be used to describe precisely what computations are carried out with the molecules and what the results of these computations are. This way, the notations may serve as (a first step towards) a formal calculus for the processing of DNA molecules. Having such a calculus could be an advantage for research in areas

such as DNA computing and (parts of) genetic engineering.

Formal grammars to describe DNA and RNA are considered in, among others, [Searls, 1992] and [Rivas & Eddy, 2000]. A useful property of such grammars, is that descriptions of molecules satisfying a grammar may be automatically parsed, during which process errors in the descriptions can be detected. A successful parse yields (some representation of) a derivation of the expression parsed, which may be further interpreted. For example, derivations of an RNA strand in a grammar may be used to predict its secondary structure, i.e., the way the strand is folded. Different derivations for the same strand (if the grammar is ambiguous) may yield different secondary structures, which are indeed observed in reality.

The importance of formal notations is also recognized in other research areas. For example, Laros et al. [2011] conclude that a formalization of the nomenclature for describing human gene variants revealed the full complexity of this nomenclature. The grammars they propose might also help to develop tools to recognize variants at the DNA level.

Let us return to DNA molecules. In order to describe a double-stranded DNA molecule, people often use the standard double-word notation (like $\begin{smallmatrix}\text{ACATG}\\ \text{TGTAC}\end{smallmatrix}$). If we were only concerned with perfectly complementary, double-stranded DNA molecules like this, then there would be an even simpler notation: it would suffice to specify the sequence of nucleotides in one of the strands, assuming a certain orientation of this strand. The other strand would be uniquely determined by Watson-Crick complementarity. For example, a description of the molecule $\begin{smallmatrix}\text{ACATG}\\ \text{TGTAC}\end{smallmatrix}$ might then be: ACATG. DNA molecules may, however, also take other shapes, and it is desirable that non-standard DNA molecules can also be denoted.

In this thesis, we describe a concise and precise notation for DNA molecules, based on the letters A, C, G and T and three operators $\uparrow$, $\downarrow$ and $\updownarrow$. The resulting *DNA expressions* denote *formal DNA molecules* – a formalization of DNA molecules. We do not only account for perfect, double-stranded DNA molecules, but also for single-stranded DNA molecules and for double-stranded DNA molecules containing nicks (missing phosphodiester bonds between adjacent nucleotides in the same strand) and gaps (missing nucleotides in one of the strands).

Our three operators bear some resemblance to the operators used in [Boneh et al., 1996] and [Li, 1999], but their functionality is quite different. The operator $\uparrow$ acts as a kind of ligase for the upper strands: it creates upper strands and connects the upper strands of its arguments. The operator $\downarrow$ is the analogue for lower strands. Finally, $\updownarrow$ fills up the gap(s) in its argument. The effects of the three operators do not perfectly match the effects of existing techniques in real-life DNA synthesis. Yet, the operators are useful to describe certain types of DNA molecules.

In our formal language, different DNA expressions may denote the same formal DNA molecule. Such DNA expressions are called *equivalent*. We examine which DNA expressions are *minimal*, which means that they have the shortest length among all DNA expressions denoting the same formal DNA molecule. Among others, we describe how to construct a minimal DNA expression for a given molecule.

For a given DNA expression E, one may want to find an equivalent, minimal DNA expression, e.g., in order to save space for storing the description of a DNA molecule. A natural way to achieve this consists of two steps: (1) to determine the molecule denoted by E, and (2) to use the constructions mentioned above to obtain a minimal DNA expression for that molecule.

We present a different approach. We describe an efficient algorithm, which *directly* rewrites E into an equivalent, minimal DNA expression. This approach is elegant, because it operates at the level of DNA expressions only, rather than to refer to the DNA molecules they denote. For many DNA molecules, there exist more than one (equivalent) minimal DNA expressions. Depending on the input, the algorithm may yield each of these.

When one wants to decide whether or not two DNA expressions E_1 and E_2 are equivalent, one may determine the DNA molecules that they denote and check if these are the same. Again, we choose a different approach. We define a *normal form*: a set of properties, such that for each DNA expression there is exactly one equivalent DNA expression with these properties. As the DNA expressions that satisfy the normal form are minimal, it is called a *minimal normal form*.

We subsequently describe an algorithm to rewrite an arbitrary DNA expression into the normal form. Now to decide whether or not E_1 and E_2 are equivalent, one determines their normal form versions and then checks if these are the same. Also this algorithm strictly operates at the level of DNA expressions. It does not refer to the DNA molecules denoted.

Recall that the algorithm for rewriting a given DNA expression into an equivalent, minimal expression may produce any minimal DNA expression, depending on the input. Hence, by itself, this algorithm is not sufficient to produce a normal form. However, the algorithm serves as the first step of our algorithm for the minimal normal form.

1.3 Set-up of the thesis

This thesis is organized as follows. Chapter 2 is intended as an introduction to the terminology from theoretical computer science and DNA, for readers that are not familiar with (either of) these fields. In fact, many terms occurring in Sections 1.1 and 1.2 are defined or explained there. The chapter also describes in more detail the contributions to the area of DNA computing by Head and Adleman, mentioned in Section 1.1.

The description of our own research starts in Chapter 3, and consists of three parts. Part I deals with DNA expressions in general. First, Chapter 3 describes a formalization of DNA molecules with nicks and gaps. This is the semantic basis of our notation. In Chapter 4, we define DNA expressions. Among other things, we examine how one can check whether or not a given string is a DNA expression and how one can compute its semantics. We also give a context-free grammar generating the DNA expressions. In Chapter 5, we derive some general results on DNA expressions, e.g., about the molecules that can be denoted by them, and about different DNA expressions that denote (almost) the same molecule.

In Part II, we focus on *minimal* DNA expressions. In Chapter 6, we derive lower bounds on the length of DNA expressions denoting a given molecule. Chapter 7 describes how to construct DNA expressions that actually achieve the lower bounds, and thus are minimal. Different types of molecules are dealt with by different constructions. In Chapter 8, we prove that there do not exist minimal DNA expressions other than those obtained with the constructions described. We also give an elegant characterization of minimal DNA expressions by six syntactic properties, which makes it easy to check whether or not a given DNA expression is minimal. Finally, we compute the number of minimal DNA expressions denoting a given molecule. In Chapter 9, we describe and analyse a recursive algorithm to rewrite an arbitrary DNA expression into an equivalent, minimal DNA expression. The algorithm applies a series of local rearrangements to the

result	brief description
Definition 3.2 (p. 35)	formal DNA molecules
Definition 4.1 (p. 47)	DNA expressions
Theorem 5.5 (p. 81)	expressible formal DNA molecules
Theorem 6.31 (p. 134)	lower bound on length DNA expr.
Theorem 7.5 (p. 138)	minimal $\updownarrow$-expressions
Theorem 7.24 (p. 158)	construction of minimal, nick free $\uparrow$-expressions and $\downarrow$-expressions
Theorem 7.46 (p. 177)	construction of minimal $\uparrow$-expressions (and $\downarrow$-expressions) with nicks
Lemma 8.22 (p. 205), Theorem 8.26 (p. 211)	characterization minimal DNA expr.
Corollary 8.47 (p. 232)	number of minimal DNA expressions
Figure 9.15 (p. 285)	algorithm for minimality
Definition 10.1 (p. 314)	minimal normal form
Lemma 10.6 (p. 317), Theorem 10.8 (p. 322)	characterization minimal normal form
Figure 11.6 (p. 356)	algorithm for minimal normal form

Table 1.1: Overview of main results from the thesis.

input DNA expression, which make sure that step by step, the DNA expression acquires the six properties that characterize minimality, while still denoting the same molecule. We prove that this algorithm is efficient.

The minimal normal form is the subject of Part III. In Chapter 10, we define the normal form. We prove that the DNA expressions in minimal normal form are characterized by five syntactic properties. The language of all normal form DNA expressions turns out to be regular. Chapter 11 is about algorithms to rewrite a given DNA expression into the normal form. First, we propose a recursive set-up, which appears to be inefficient. Therefore, we also describe an alternative, two-step algorithm. This algorithm first makes the DNA expression minimal (using the algorithm from Chapter 9) and then rewrites the resulting minimal DNA expression into the normal form. This second algorithm, which uses the characterization of the normal form by five properties, is efficient.

In Chapter 12 we summarize and discuss the results, draw conclusions from our work and suggest directions for future research.

To facilitate a quick look-up, we list the main results from the thesis also in Table 1.1. The contents of the thesis is schematically summarized in Figure 1.1. The figure can be understood as follows. In order to denote (formal) DNA molecules, we use letters representing the bases, and operators $\uparrow$, $\downarrow$ and $\updownarrow$. The result are DNA expressions. Every expressible formal DNA molecule is denoted by infinitely many DNA expressions. Some of these DNA expressions are shorter than others. We consider the ones with minimal length, the minimal DNA expressions. There may be more than one minimal DNA expression for the same DNA molecule. Only one of these is in (minimal) normal form.

1.4 Resulting publications

We have published the definitions and the main results from this thesis in two technical reports, one conference paper and three journal papers. We list them here:

- R. van Vliet: *Combinatorial Aspects of Minimal DNA Expressions (ext.)*, Technical

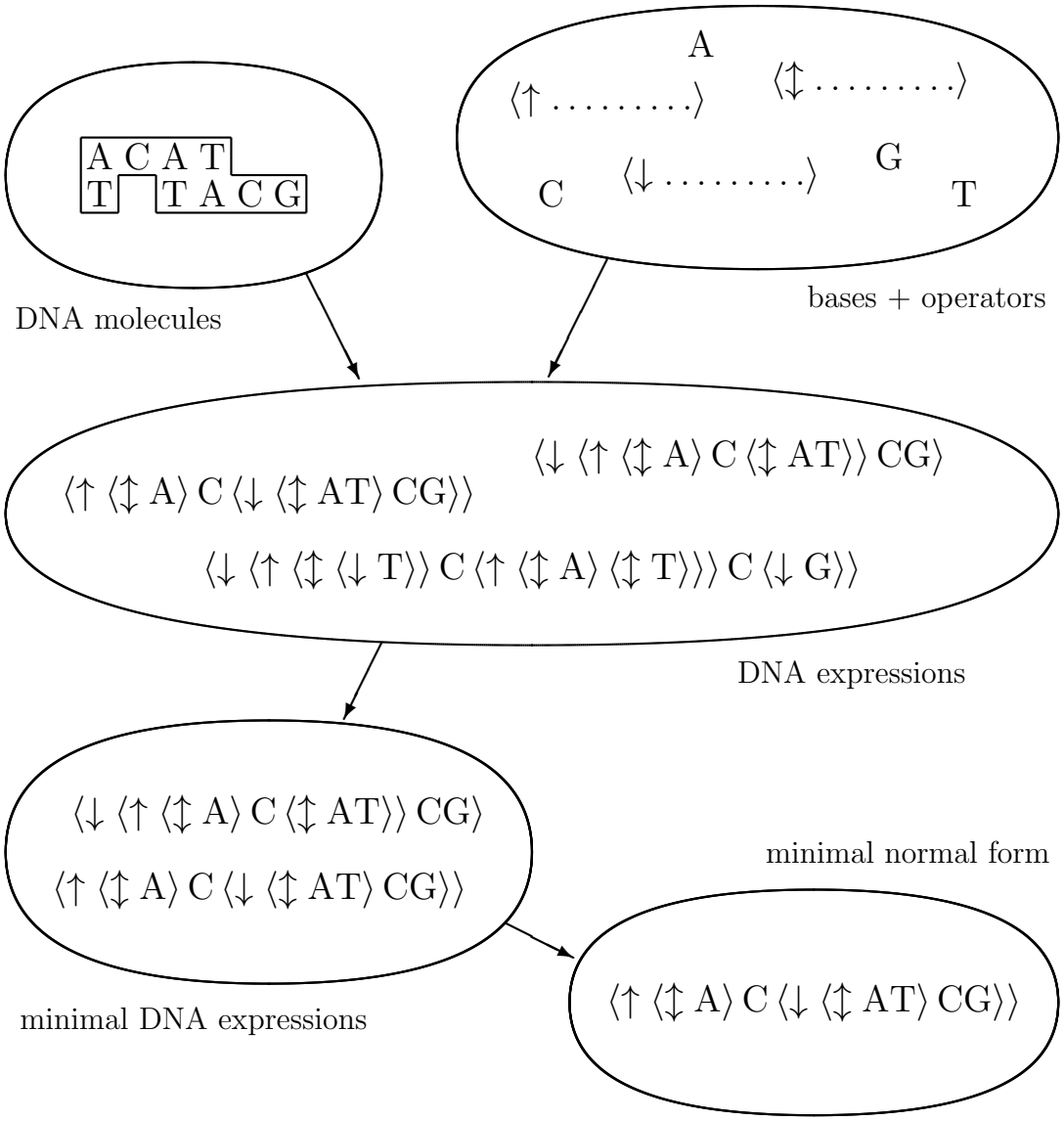


Figure 1.1: Schematic view of the contents of the thesis.

Report 2004-03, Leiden Institute of Advanced Computer Science, Leiden University (2004), see the repository of Leiden University at
`https://openaccess.leidenuniv.nl`

This report contains, a.o., the formal proofs of the results in the conference paper *Combinatorial aspects of minimal DNA expressions* below. Due to space limitations, these could not be included in the paper itself. The report roughly corresponds to Chapters 3–8 of this thesis.

- R. van Vliet, H.J. Hoogeboom, G. Rozenberg: Combinatorial aspects of minimal DNA expressions, *DNA Computing – 10th International Workshop on DNA Computing, DNA10, Milan, Italy, June 7–10, 2004 – Revised Selected Papers*, Lecture Notes in Computer Science **3384** (C. Ferretti, G. Mauri, C. Zandron, eds), Springer (2005), 375–388.

  This paper has been presented at the conference mentioned. It was awarded one of

the two *best student papers awards* from this conference. The paper contains some of the main results from Chapters 3–8 of this thesis.

- R. van Vliet, H.J. Hoogeboom, G. Rozenberg: The construction of minimal DNA expressions, *Natural Computing* **5**(2) (2006), 127–149.

  After DNA10, six of the papers presented at the conference were selected for a special issue of *Natural Computing*. Among these was the above paper *Combinatorial aspects of minimal DNA expressions*. We significantly revised the paper, focusing on the *construction* of minimal DNA expressions. Because of the more limited scope, we could elaborate more on the proof that the resulting DNA expressions are really minimal. These aspects are covered in Chapters 6 and 7 of this thesis.

- R. van Vliet: *All about a Minimal Normal Form for DNA Expressions*, Technical Report 2011-03, Leiden Institute of Advanced Computer Science, Leiden University (2011), see the repository of Leiden University at
  `https://openaccess.leidenuniv.nl`

  This report contains, a.o., more details about the results in the two journal papers *Making DNA expressions minimal* and *A minimal normal form for DNA expressions* below. The report roughly corresponds to Chapters 9–11 of this thesis.

- R. van Vliet, H.J. Hoogeboom: Making DNA expressions minimal, *Fundamenta Informaticae* **123**(2) (2013), 199–226.

  This paper is part 1 of a diptych, which were published together. It contains some of the main results from Chapter 9 of this thesis. Part 2 is the paper *A minimal normal form for DNA expressions* below.

- R. van Vliet, H.J. Hoogeboom: A minimal normal form for DNA expressions, *Fundamenta Informaticae* **123**(2) (2013), 227–243.

  This paper is part 2 of a diptych, which were published together. It contains some of the main results from Chapters 10 and 11 of this thesis. Part 1 is the above paper *Making DNA expressions minimal*.

Chapter 2

Preliminaries

The topic of this thesis is a formal language to describe DNA molecules. As such, it is a combination of theoretical computer science and molecular biology. Therefore, in the description and discussion of the subject, we will frequently use terms and concepts from both fields. Readers with a background in biology may not be familiar with the terminology from computer science and vice versa. In order for this thesis to be understandable to readers with either background, this chapter provides a brief introduction to the two fields.

First, we introduce some terminology and present a few results from computer science, concerning strings, trees, grammars, relations, and algorithmic complexity. Next, we discuss DNA, its structure and some possible deviations from the perfect double-stranded DNA molecule. We finally describe two important contributions to the field of DNA computing, which has emerged at the interface of computer science and biology.

Readers that are familiar with both theoretical computer science and DNA, may skip over this chapter and proceed to Chapter 3. If necessary, they can use the list of symbols and the index at the end of this thesis to find the precise meaning of a symbol or term introduced in the present chapter.

2.1 Strings, trees, grammars, relations and complexity

An *alphabet* is a finite set, the elements of which are called *symbols* or *letters*. A finite sequence of symbols from an alphabet Σ is called a *string* over Σ. For a string $X = x_1x_2 \dots x_r$ over an alphabet Σ, with $x_1, x_2, \dots, x_r \in \Sigma$, the *length* of X is r. In general, we use $|X|$ to denote the length of a string X. The length of the *empty string* λ equals 0.

For a non-empty string $X = x_1x_2 \dots x_r$, we define $L(X) = x_1$ and $R(X) = x_r$. The concatenation of two strings X_1 and X_2 over an alphabet Σ is usually denoted as X_1X_2; sometimes, however, we will explicitly write $X_1 \cdot X_2$. Concatenation is an *associative* operation, which means that $(X_1 \cdot X_2) \cdot X_3 = X_1 \cdot (X_2 \cdot X_3)$ for all strings X_1, X_2, X_3 over Σ. Because of this, the notation $X_1X_2X_3$ (or $X_1 \cdot X_2 \cdot X_3$) is unambiguous.

For a letter a from the alphabet Σ, the number of occurrences of a in a string X is denoted by $\#_a(X)$. Sometimes, we are not so much interested in the number of occurrences of a single letter in a string X, but rather in the total number of occurrences of two different letters a and b in X. This total number is denoted by $\#_{a,b}(X)$.

One particular alphabet that we will introduce in this thesis is $\Sigma = \{\text{A, C, G, T}\}$. If

X = ACATGCAT, then, for example, $|X| = 8$, $L(X) = \text{A}$ and $\#_{\text{A,T}}(X) = 5$.

The set of all strings over an alphabet Σ is denoted by Σ^*, and $\Sigma^+ = \Sigma^* \setminus \{\lambda\}$ (the set of non-empty strings). A *language* over Σ is a subset $\mathcal{K}$ of Σ^*.

Substrings

A *substring* of a string X is a (possibly empty) string X^s such that there are (possibly empty) strings X_1 and X_2 with $X = X_1 X^s X_2$. If $X^s \neq X$, then X^s is a *proper substring* of X. We call the pair (X_1, X_2) an *occurrence* of X^s in X. If $X_1 = \lambda$, then X^s is a *prefix* of X; if $X_2 = \lambda$, then X^s is a *suffix* of X. If a prefix of X is a proper substring of X, then it is also called a *proper prefix*. Analogously, we may have a *proper suffix* of X.

For example, the string X = ACATGCAT has one occurrence of the substring ATGCA and two occurrences of the substring AT. One of the occurrences of AT is (ACATGC, λ), so AT is a (proper) suffix of X.

If (X_1, X_2) and (Y_1, Y_2) are different occurrences of X^s in X, then (X_1, X_2) *precedes* (Y_1, Y_2) if $|X_1| < |Y_1|$. Hence, all occurrences in X of a given string X^s are linearly ordered, and we can talk about the first, second, ... occurrence of X^s in X. Although, formally, an occurrence of a substring X^s in a string X is the pair (X_1, X_2) surrounding X^s in X, the term will also be used to refer to the substring itself, at the position in X determined by (X_1, X_2).

Note that for a string $X = x_1 x_2 \dots x_r$ of length r, the empty string λ has $r + 1$ occurrences: $(\lambda, X), (x_1,\ x_2 \dots x_r), \dots, (x_1 \dots x_{r-1},\ x_r), (X, \lambda)$.

If a string X is the concatenation of k times the same substring X^s, hence $X = \underbrace{X^s \dots X^s}_{k \text{ times}}$, then we may write X as $(X^s)^k$.

Let (Y_1, Y_2) and (Z_1, Z_2) be occurrences in a string X of substrings Y^s and Z^s, respectively. We say that (Y_1, Y_2) and (Z_1, Z_2) are *disjoint*, if either $|Y_1| + |Y^s| \leq |Z_1|$ or $|Z_1| + |Z^s| \leq |Y_1|$. Intuitively, one of the substrings occurs (in its entirety) before the other one.

If the two occurrences are not disjoint, hence if $|Z_1| < |Y_1| + |Y^s|$ and $|Y_1| < |Z_1| + |Z^s|$, then they are said to *intersect*. Note that, according to this formalization of intersection, an occurrence of the empty string λ may intersect with an occurrence of a non-empty string. In this thesis, however, we will not deal with this pathological type of intersections. Occurrences of two non-empty substrings intersect, if and only if the substrings have at least one (occurrence of a) letter in common.

We say that (Y_1, Y_2) *overlaps* with (Z_1, Z_2), if either $|Y_1| < |Z_1| < |Y_1| + |Y^s| < |Z_1| + |Z^s|$ or $|Z_1| < |Y_1| < |Z_1| + |Z^s| < |Y_1| + |Y^s|$. Hence, one of the substrings starts *before* and ends *inside* the other one.

Finally, the occurrence (Y_1, Y_2) of Y^s *contains* (or *includes*) the occurrence (Z_1, Z_2) of Z^s, if $|Y_1| \leq |Z_1|$ and $|Z_1| + |Z^s| \leq |Y_1| + |Y^s|$.

In Figure 2.1, we have schematically depicted the notions of disjointness, intersection, overlap and inclusion.

If it is clear from the context which occurrences of Y^s and Z^s in X are considered, e.g., if these strings occur in X exactly once, then we may also say that the substrings Y^s and Z^s *themselves* are disjoint, intersect or overlap, or that one contains the other.

Note the difference between intersection and overlap. If (occurrences of) two substrings intersect, then either they overlap, or one contains the other, and these two possibilities are mutually exclusive. For example, in the string X = ACATGCAT the (only occurrence of

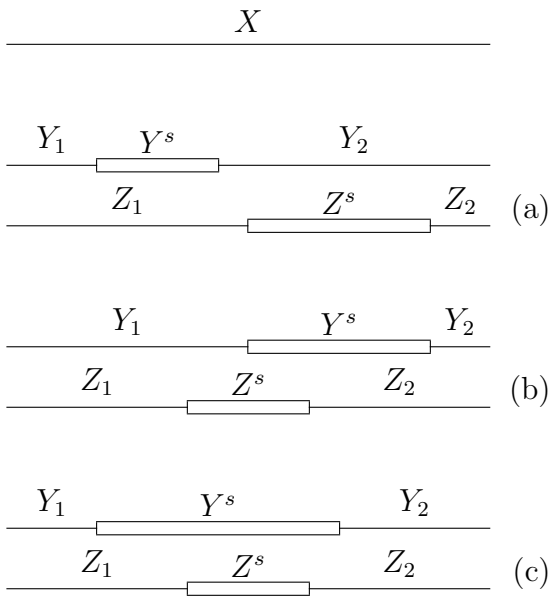


Figure 2.1: Examples of disjoint and intersecting occurrences (Y_1, Y_2) of Y^s and (Z_1, Z_2) of Z^s in a string X. (a) The occurrences are disjoint: $|Y_1| + |Y^s| \leq |Z_1|$. (b) The occurrences overlap: $|Z_1| < |Y_1| < |Z_1| + |Z^s| < |Y_1| + |Y^s|$. (c) The occurrence of Y^s contains the occurrence of Z^s: $|Y_1| \leq |Z_1|$ and $|Z_1| + |Z^s| \leq |Y_1| + |Y^s|$.

the) substring $Y^s = \text{ATGCA}$ intersects with both occurrences of the substring $Z^s = \text{AT}$. It contains the first occurrence of Z^s and it overlaps with the second occurrence of Z^s.

Functions on strings

Let Σ be an alphabet. We can consider the set Σ^* (of strings over Σ) as an algebraic structure, with the concatenation as operation: the concatenation of two strings over Σ is again a string over Σ. In this context, the empty string λ is the *identity* 1_{Σ^*}, i.e., the unique element satisfying $X \cdot 1_{\Sigma^*} = 1_{\Sigma^*} \cdot X = X$ for all $X \in \Sigma^*$.

Let K be a set with an associative operation $\circ$ and identity 1_K. A function h from Σ^* to K is called a *homomorphism*, if $h(X_1X_2) = h(X_1) \circ h(X_2)$ for all $X_1, X_2 \in \Sigma^*$ and $h(1_{\Sigma^*}) = 1_K$. Hence, to specify h if suffices to give its values for the letters from Σ and for the identity $1_{\Sigma^*} = \lambda$.

We have already seen an example of a homomorphism. The length function $|\cdot|$ is a homomorphism from Σ^* to the non-negative integers with addition as the operation. Indeed, $|\lambda| = 0$, which is the identity for addition of numbers.

If a homomorphism h maps the elements of Σ^* into Σ^* (i.e., if $K = \Sigma^*$ and the operation of K is concatenation), then h is called an *endomorphism*.

Rooted trees

A *graph* is a pair (V, E), where V is a set of *nodes* or *vertices* and E is a set of *edges* between the nodes. If the edges are undirected, then the graph itself is called *undirected*. Otherwise, the graph is *directed*. Figure 2.2 shows examples of an undirected graph and a directed graph.

A *tree* is a non-empty, undirected graph such that for all nodes X and Y in the graph,

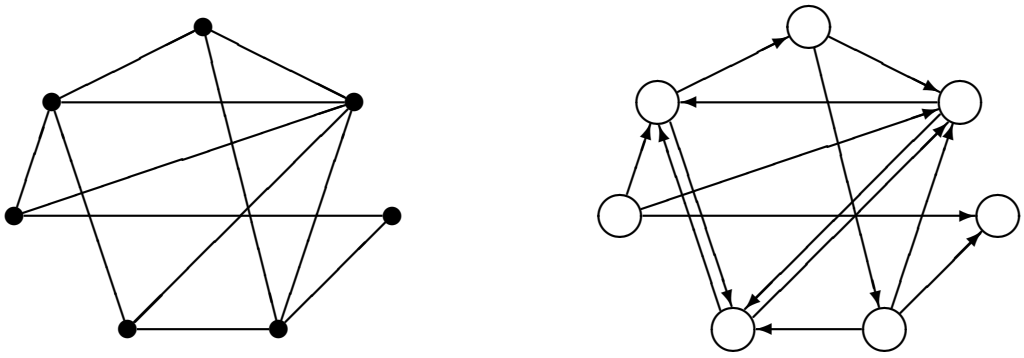


Figure 2.2: Examples of graphs. (a) An undirected graph with seven nodes. (b) A directed graph with seven nodes.

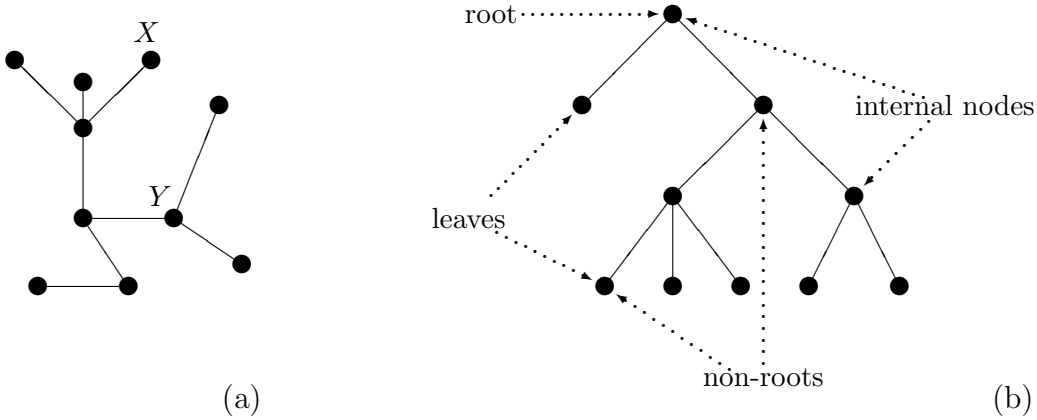


Figure 2.3: Examples of trees. (a) A tree with ten nodes. (b) A rooted tree with ten nodes, in which the root and some non-roots, internal nodes and leaves have been indicated.

there is exactly one simple path between X and Y. In particular, a tree is connected. Figure 2.3(a) shows an example of a tree. The *distance* between two nodes in a tree is the number of edges on the path between the two nodes. For example, the distance between nodes X and Y in the tree from Figure 2.3(a) is 3.

A *rooted tree* is a tree with one designated node, which is called the *root* of the tree. A *non-root* in the tree is a node other than the root of the tree. Let X be a non-root in a rooted tree t. The nodes on the path from the root of the tree to X (including the root, but excluding X) are the *ancestors* of X. The last node on this path is the *parent* of X. X is called a *child* of its parent. All nodes 'below' a node X in the tree, i.e., nodes that X is an ancestor of, are called *descendants* of X. The *subtree rooted in* X is the subtree of t with root X, consisting of X and *all* its descendants, together with the edges connecting these nodes. A *leaf* in a rooted tree is a node without descendants. Nodes that do have descendants are called *internal nodes*. We thus have two ways to partition the nodes in a rooted tree: either in a root and non-roots, or in leaves and internal nodes.

Usually, in a picture of a rooted tree, the root is at the top, its children are one level lower, the children of the children are another level lower, and so on. An example is given in Figure 2.3(b). In this example we have also indicated the root and some of the

non-roots, internal nodes and leaves. Note that the choice of a root implicitly fixes an orientation of the edges in the tree: from the root downwards.

A *level* of a rooted tree is the set of nodes in the tree that are at the same distance from the root of the tree. The root is at level 1, the children of the root are at level 2, and so on. The *height* of a rooted tree is the maximal non-empty level of the tree. Obviously, this maximal level only contains leaves. There may, however, also be leaves at other levels. For example, the height of the tree depicted in Figure 2.3(b) is 4, level 2 contains a leaf and an internal node, and level 4 contains five leaves.

It follows immediately from the definition that the height of a tree can be recursively expressed in the heights of its subtrees:

Lemma 2.1 *Let t be a rooted tree, and let $X_1, \ldots, X_n$ for some $n \geq 0$ be the children of the root of t.*

1. *If $n = 0$ (i.e., if t consists only of a root), then the height of t is 1.*
2. *If $n \geq 1$, then the height of t is equal to*

$$\max_{i=1}^{n} \textit{(height of the subtree of } t \textit{ rooted at } X_i\textit{)} + 1.$$

A rooted tree is *ordered* if for each internal node X, the children of X are linearly ordered ('from left to right'). Finally, an *ordered, rooted, node-labelled tree* is an ordered rooted tree with labels at the nodes.

Grammars

A *grammar* is a formalism that describes how the elements of a language (i.e., the strings) can be derived from a certain initial symbol using rewriting rules. We are in particular interested in context-free grammars and right-linear grammars.

A *context-free grammar* is a 4-tuple $G = (V, \Sigma, P, S)$, where

- V is a finite set of *non-terminal symbols* (or *variables*): symbols that may occur in intermediate strings derived in the grammar, but not in final strings,
- Σ is a finite set of *terminal symbols*: symbols that may occur in intermediate strings *and* final strings derived in the grammar,
- P is a finite set of *productions*: rewriting rules for elements from V,
- $S \in V$ is the *start symbol*.

The sets V and Σ are disjoint. Every production is of the form $A \longrightarrow Z$, where $A \in V$ and $Z \in (V \cup \Sigma)^*$. It indicates that the non-terminal symbol A may be replaced by the string Z over $V \cup \Sigma$.

Let (X_1, X_2) be an occurrence of the non-terminal symbol A in a string X over $V \cup \Sigma$. Hence, $X = X_1 A X_2$ for some $X_1, X_2 \in (V \cup \Sigma)^*$. When we *apply* the production $A \longrightarrow Z$ to this occurrence of A in X, we substitute A in X by Z. The result is the string $X_1 Z X_2$.

A string that can be obtained from the start symbol S by applying zero or more productions from P, is called a *sentential form*. In particular, the string S (containing only the start symbol) is a sentential form. It is the result of applying zero productions.

The *language of* G (or the *language generated by* G) is the set of all sentential forms that only contain terminal symbols, i.e., the set of all strings over Σ that can be obtained from the start symbol S by the application of zero or more[1] productions. We use $\mathcal{L}(G)$ to denote the language of G.

A language $\mathcal{K}$ is called context-free, if there exists a context-free grammar G such that $\mathcal{K} = \mathcal{L}(G)$.

Let X be an arbitrary string over $V \cup \Sigma$. A *derivation* in G of a string Y from X is a sequence of strings starting with X and ending with Y, such that we can obtain a string in the sequence from the previous one by the application of one production from P. If we use $X_0, X_1, \ldots, X_k$ to denote the successive strings (with $X_0 = X$ and $X_k = Y$), then the derivation is conveniently denoted as $X_0 \Longrightarrow X_1 \Longrightarrow \cdots \Longrightarrow X_k$. If the initial string X in the derivation is equal to the start symbol S of the grammar, then we often simply speak of a *derivation of* Y (and do not mention S).

For arbitrary strings X over $V \cup \Sigma$, the language $\mathcal{L}_G(X)$ is the set of all strings over Σ that can be derived in G from X:

$$\mathcal{L}_G(X) = \{Y \in \Sigma^* \mid \text{there exists a derivation in } G \text{ of } Y \text{ from } X\}.$$

If the grammar G is clear from the context, then we will also write $\mathcal{L}(X)$. In particular, $\mathcal{L}(G) = \mathcal{L}_G(S) = \mathcal{L}(S)$.

Example 2.2 Consider the context-free grammar $G = (\{S, A, B\}, \{a, b\}, P, S)$, where

$$\begin{array}{rcll} P = \{S & \longrightarrow & \lambda, \\ S & \longrightarrow & ASB, \\ A & \longrightarrow & a, \\ B & \longrightarrow & b & \}. \end{array}$$

A possible derivation in G is

$$\begin{array}{rcl} S & \Longrightarrow & ASB \\ & \Longrightarrow & AASBB \\ & \Longrightarrow & AASBb \\ & \Longrightarrow & aASBb \\ & \Longrightarrow & aASbb \\ & \Longrightarrow & aaSbb \\ & \Longrightarrow & aabb \end{array} \tag{2.1}$$

In this derivation, we successively applied the second, the second, the fourth, the third, the fourth, the third and the first production from P.

It is not hard to see that $\mathcal{L}(G) = \{a^m b^m \mid m \geq 0\}$. ■

The notation

$$A \longrightarrow Z_1 \mid Z_2 \mid \ldots \mid Z_n$$

is short for the set of productions

$$\begin{array}{rcl} A & \longrightarrow & Z_1, \\ A & \longrightarrow & Z_2, \\ \vdots & \vdots & \vdots \\ A & \longrightarrow & Z_n \end{array}$$

[1] In practice, of course, because $S \notin \Sigma$, we need to apply at least one production to obtain an element of the language of G.

For example, the set of productions from the grammar G in Example 2.2 can be written as

$$\begin{array}{rcll} P = \{S & \longrightarrow & \lambda \mid ASB, \\ A & \longrightarrow & a, \\ B & \longrightarrow & b & \}. \end{array}$$

With this shorter notation for the productions, we may use 'production (i, j)' to refer to the production with the j^{th} right-hand side from line i. In our example, production $(1, 2)$ is the production $S \longrightarrow ASB$.

If a sentential form contains more than one non-terminal symbol, then we can choose which one to expand next. Different choices usually yield different derivations, which may still yield the same final string. If, in each step of a derivation, we expand the leftmost non-terminal symbol, then the derivation is called a *leftmost derivation*. Derivation (2.1) in Example 2.2 is clearly not a leftmost derivation.

Example 2.3 Let G be the context-free grammar from Example 2.2. A leftmost derivation of the string $aabb$ in G is

$$\begin{array}{rcl} S & \Longrightarrow & ASB \\ & \Longrightarrow & aSB \\ & \Longrightarrow & aASBB \\ & \Longrightarrow & aaSBB \\ & \Longrightarrow & aaBB \\ & \Longrightarrow & aabB \\ & \Longrightarrow & aabb \end{array} \tag{2.2}$$

■

The structure of a derivation in a context-free grammar that begins with the start symbol, can be conveniently expressed by means of an ordered, rooted, node-labelled tree, which is called a *derivation tree* or a *parse tree*. To build up the tree, we closely follow the derivation.

We start with only a root, which is labelled by the start symbol S. This corresponds to the first string in the derivation. In each step of the derivation, a production $A \longrightarrow Z$ is applied to a certain occurrence of a non-terminal A in the current string. Let $Z = x_1 \dots x_r$ for some $r \geq 0$ and letters $x_1, \dots, x_r$ from $V \cup \Sigma$. For $i = 1, \dots, r$, we create a node with label x_i. In the special case that $r = 0$, we create one node with label λ. By construction, there already exists a node corresponding to (this occurrence of) the non-terminal A. The new nodes become the children of this node, and are arranged from left to right according to the order of their labels in Z.

The concatenation of the labels of the leaves (in the order of their occurrence from left to right in the tree) is called the *yield* of the derivation tree. By construction, it is equal to the string derived.

Different derivations may have the same derivation tree. In our example grammar G, this is also the case for the two derivations of $aabb$ that we have seen. Figure 2.4(a) shows their common derivation tree. Indeed, the yield of this tree is $aa \cdot \lambda \cdot bb = aabb$. For each derivation tree, however, there is only one *leftmost* derivation.

A context-free grammar G is called *ambiguous*, if there is at least one string $X \in \mathcal{L}(G)$ which is the yield of two (or more) different derivation trees in G, i.e., for which the

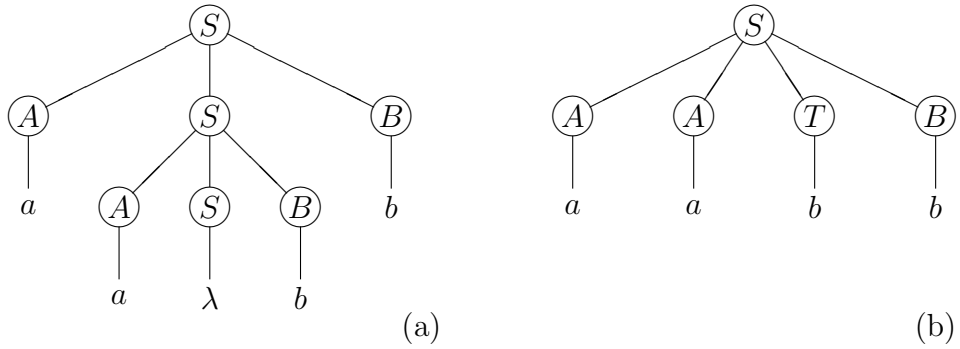


Figure 2.4: Two derivation trees. (a) The derivation tree corresponding to both Derivation (2.1) and Derivation (2.2) of *aabb* in the example context-free grammar G. It is also a derivation tree for *aabb* in the context-free grammar G' from Example 2.4. (b) Another derivation tree for *aabb* in G'.

grammatical structure is not unique. In this case, X also has two (or more) different leftmost derivations in G.

A context-free grammar that is not ambiguous, is *unambiguous*. One can prove that grammar G from Example 2.2 and Example 2.3 is unambiguous. In particular, the tree in Figure 2.4(a) is the *unique* derivation tree of *aabb* in G.

Example 2.4 Consider the context-free grammar $G' = (\{S, T, A, B\}, \{a, b\}, P', S)$, where

$$\begin{aligned} P' = \{ & S \longrightarrow \lambda \mid ASB \mid AATB, \\ & T \longrightarrow ATB \mid b, \\ & A \longrightarrow a, \\ & B \longrightarrow b \qquad \}. \end{aligned}$$

Then the tree from Figure 2.4(a) is also a derivation tree for *aabb* in G'. However, Figure 2.4(b) contains another derivation tree for the same string in G'. Hence, G' is ambiguous. It is not hard to see that $\mathcal{L}(G') = \mathcal{L}(G) = \{a^m b^m \mid m \geq 0\}$. ■

A *right-linear grammar* is a special type of context-free grammar, in which every production is either of the from $A \longrightarrow \lambda$ or of the form $A \longrightarrow aB$ with $A, B \in V$ and $a \in \Sigma$. A language $\mathcal{K}$ is called *regular*, if there exists a right-linear grammar G such that $\mathcal{K} = \mathcal{L}(G)$.

Example 2.5 Consider the right-linear grammar $G = \{\{S, B\}, \{a, b\}, P, S\}$, where

$$\begin{aligned} P = \{ & S \longrightarrow \lambda \mid aB, \\ & B \longrightarrow bS \quad \}. \end{aligned}$$

A possible derivation in G is

$$\begin{aligned} S &\Longrightarrow aB \\ &\Longrightarrow abS \\ &\Longrightarrow abaB \\ &\Longrightarrow ababS \\ &\Longrightarrow ababaB \\ &\Longrightarrow abababS \\ &\Longrightarrow ababab. \end{aligned}$$

It is not hard to see that in this case, $\mathcal{L}(G) = \{(ab)^m \mid m \geq 0\}$. ■

To prove that a given language is regular, one may prove that it is generated by a certain right-linear grammar. Sometimes, however, one can also use a result from formal language theory, stating that a language generated by a context-free grammar with a particular property is regular.

Let G be a context-free grammar, let Σ be the set of terminal symbols in G and let A be a non-terminal symbol in G. We say that A is *self-embedding* if there exist non-empty strings X_1, X_2 over Σ, such that the string X_1AX_2 can be derived from A. Intuitively, we can 'blow up' A by rewriting it into X_1AX_2, rewriting the new occurrence of A into X_1AX_2, and so on.

G itself is called self-embedding, if it contains at least one non-terminal symbol that is self-embedding. In other words: G is not self-embedding, if none of its non-terminal symbols is self-embedding. A right-linear grammar is not self-embedding, because for each production $A \longrightarrow Z$ in such a grammar, the right hand side Z contains at most one non-terminal symbol, which then is the last symbol of Z. Hence, if we can derive a string X_1AX_2 from a non-terminal symbol A, then $X_2 = \lambda$. This observation implies that any regular language can be generated by a grammar that is not self-embedding. As was proved in [Chomsky, 1959], the reverse is also true: a context-free grammar that is not self-embedding generates a regular language. We thus have:

Proposition 2.6 *A language $\mathcal{K}$ is regular, if and only if it can be generated by a context-free grammar that is not self-embedding.*

To prove that a given language is *not* regular, one often uses the *pumping lemma* for regular languages. This lemma describes a property that all regular languages have. If the given language lacks this property, then it cannot be regular.[2]

Proposition 2.7 (Pumping lemma for regular languages). *Let $\mathcal{K}$ be a regular language over an alphabet Σ. There exists an integer $n \geq 1$, such that for each string $x \in \mathcal{K}$ with $|x| \geq n$, there exist three strings u, v, w over Σ, such that*

1. *$x = uvw$, and*
2. *$|uv| \leq n$, and*
3. *$|v| \geq 1$ (i.e., $v \neq \lambda$), and*
4. *for every $i \geq 0$, also the string $uv^iw \in \mathcal{K}$.*

Hence, each string $x \in \mathcal{K}$ that is sufficiently long can be 'pumped' (in particular, the substring v, which is 'not far' from the beginning of x, can be pumped), and the result will still be an element of $\mathcal{K}$. We give an example to explain how the lemma is often applied.

Example 2.8 Let $\mathcal{K}$ be the context-free language from Example 2.2: $\mathcal{K} = \{a^mb^m \mid m \geq 0\}$.

Suppose that $\mathcal{K}$ is regular. By Proposition 2.7, there exists an integer $n \geq 1$, such that each string $x \in \mathcal{K}$ with $|x| \geq n$ can be written as $x = uvw$ and can then be pumped. If we choose $x = a^nb^n$, then by Property (2), the substring v consists of only a's. When we take, e.g., $i = 2$, by Property (3), the number of a's in the string uv^iw becomes larger than the number of b's. This implies that this string is not in $\mathcal{K}$. As this contradicts Property (4), the hypothesis that $\mathcal{K}$ is regular must be false. ■

[2]Unfortunately, the reverse implication does not hold. That is, there exist languages that have the property, but are not regular.

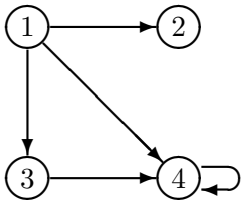


Figure 2.5: Graphical representation of the binary relation R from Example 2.9.

Binary relations

A *binary relation* R on a set X is a subset of $X \times X = \{(x, y) \mid x, y \in X\}$. If $(x, y) \in R$, then we also write xRy; if $(x, y) \notin R$, then we may write $x \not R y$. A binary relation can be naturally depicted as a directed graph $G = (X, R)$, i.e., a graph with the elements of X as nodes and edges determined by R.

Example 2.9 Let $X = \{1, 2, 3, 4\}$. Then $R = \{(1, 2), (1, 3), (1, 4), (3, 4), (4, 4)\}$ is a binary relation on X. This relation has been depicted in Figure 2.5. ■

A binary relation R on X is
- *reflexive* if for every $x \in X$, xRx
- *symmetric* if for every $x, y \in X$, xRy implies yRx
- *antisymmetric* if for every $x, y \in X$, (xRy and yRx) implies $x = y$
- *transitive* if for every $x, y, z \in X$, (xRy and yRz) implies xRz

The relation R from Example 2.9 is antisymmetric and transitive. It is not reflexive and not symmetric.

If a relation R is reflexive, symmetric and transitive, R is called an *equivalence relation*; if R is reflexive, antisymmetric and transitive, we call R a *partial order*.

Given a binary relation R, the set $R^{-1} = \{(y, x) \mid (x, y) \in R\}$ is the *inverse relation* of R. A binary relation R_1 is a *refinement* of a binary relation R_2 if $R_1 \subseteq R_2$, in other words: if xR_1y implies xR_2y. In this case R_2 is called an *extension* of R_1.

Complexity of an algorithm

An *algorithm* is a step-by-step description of an effective method for solving a problem or completing a task. There are, for example, a number of different algorithms for sorting a sequence of numbers. In this thesis, we describe an algorithm to determine the semantics of a DNA expression, and a few algorithms to transform a given DNA expression into another DNA expression with some desired properties. In each of these cases, the input of the algorithm is a DNA expression E, which is in fact just a string over a certain alphabet, satisfying certain conditions.

Algorithms can, a.o., be classified by the amount of time or by the amount of memory space they require, depending on the size of the input. In particular, one is often interested in the *time complexity* (or *space complexity*) of an algorithm, which expresses the rate by which the time (space) requirements grow when the input grows. In our case, the size of the input is the length $|E|$ of the DNA expression E. Hence, growing input means that we consider longer strings E.

For example, an algorithm is said to have *linear* time complexity, if its time requirements are roughly proportional to the size of its input: when the input size (the length

$|E|$) grows with a certain factor, the time required by the algorithm grows with roughly the same factor. In this case, we may also say that this time *is* linear in the input size. An algorithm has *quadratic* time complexity, if its time requirements grow with a factor c^2 when the input size grows with a factor c.

We speak of a *polynomial* time complexity, if the time requirements can be written as a polynomial function of the input size. Both linear time complexity and quadratic time complexity are examples of this. If the time required by an algorithm grows by an exponential function of the input size, the algorithm has an *exponential* time complexity.

In the analysis of complexities, we will also use the *big O notation*. For example, we may say that the time spent in an algorithm for a given DNA expression E *is in* $\mathcal{O}(|E|)$. By this, we mean that this time grows *at most* linearly with the length of E. We thus have an *upper bound* on the time complexity. In this case, in order to conclude that the algorithm really has linear time complexity, we need to prove that $|E|$ also provides a *lower bound* for the complexity.

2.2 DNA molecules

Many properties of organisms are (partly) determined by their genes. Examples for humans are the sex, the colour of the eyes and the sensitivity to certain diseases. The genetic information is stored in *DNA molecules*, and in fact, a gene is a part of a DNA molecule. Copies of an organism's DNA can be found in nearly every cell of the organism. In the cell, a DNA molecule is packaged in a *chromosome*, together with DNA-bound proteins. A human cell contains 23 pairs of chromosomes, where each pair consists of a chromosome inherited from the father and one from the mother.

The structure of the DNA molecule was first described by the scientists James Watson and Francis Crick in [1953]. The model they proposed was confirmed by experiments by, a.o., Maurice Wilkins and Rosalind Franklin. Watson, Crick and Wilkins jointly received the Nobel Prize in Physiology or Medicine in 1962. Franklin died four years before this occasion.

Nucleotides

The acronym DNA stands for *DeoxyriboNucleic Acid*. This name refers to the basic building blocks of the molecule, the *nucleotides*, each of which consists of three components: (i) a *phosphate group* (related to phosphoric *acid*), (ii) the *sugar deoxyribose* and (iii) a *base* or *nucleobase*. Here, the prefix 'nucleo' refers to the place where the molecules were discovered: the nucleus of a cell.

The chemical structure of a nucleotide is depicted in Figure 2.6(a). The subdivision into three components is shown in Figure 2.6(b). The phosphate group is attached to the $5'$-site (the carbon atom numbered $5'$) of the sugar. The base is attached to the $1'$-site. Within the sugar, we also identify a *hydroxyl group* (OH), which is attached to the $3'$-site.

There are four types of bases: *adenine*, *cytosine*, *guanine* and *thymine*, which are abbreviated by A, C, G and T, respectively. The only place where nucleotides can differ from each other is the base. Hence, each nucleotide is characterized by its base. Therefore, the letters A, C, G and T are also used to denote the entire nucleotides.

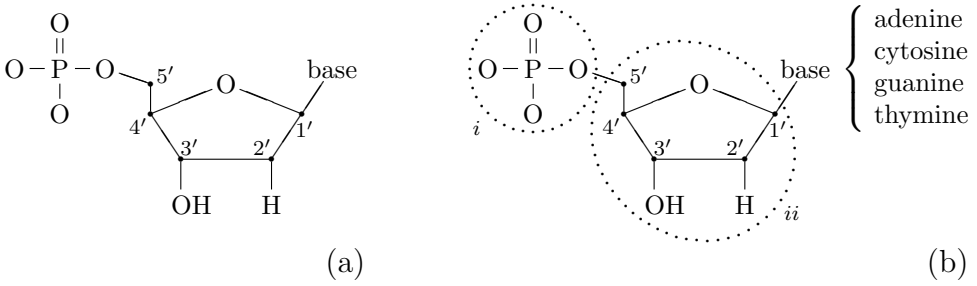


Figure 2.6: (a) Simplified picture of the chemical structure of a nucleotide, with $1'$ through $5'$ numbering carbon atoms. (b) The three components of a nucleotide: the phosphate group (*i*), the sugar (*ii*) and the base.

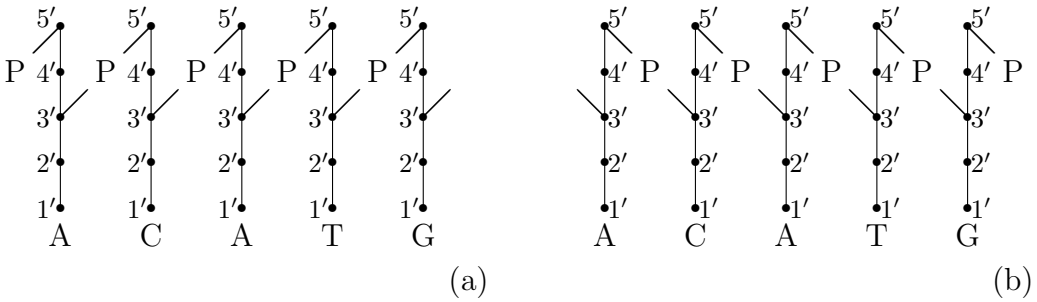


Figure 2.7: Schematical pictures of of two (different) single-stranded DNA molecules: (a) $5'$-ACATG-$3'$; (b) $3'$-ACATG-$5'$.

Connections between nucleotides

Different nucleotides can bind to each other in two ways.

First, the hydroxyl group of one nucleotide can interact with the phosphate group of another nucleotide, yielding the so-called *phosphodiester bond*. This is a strong (covalent) bond. The formation of a phosphodiester bond is facilitated by a *ligase* enzyme.

The resulting molecule has a free (unused) phosphate group at the $5'$-site of one nucleotide and a free hydroxyl group at the $3'$-site of the other nucleotide. These can, in turn, connect to the hydroxyl group or the phosphate group, respectively, of yet another nucleotide. The so obtained chains of nucleotides are called *single-stranded DNA molecules*, or simply *single strands* or *strands*. In biochemistry, this last term is also used to refer to double-stranded DNA molecules (which will be introduced shortly), but we will limit the use of 'strand' to single-stranded DNA molecules.

The end of the strand that has a free phosphate group at its $5'$-site is called the $5'$-end of the strand. The other end then is the $3'$-end of the strand. The chemical properties of the $5'$-end (with its phosphate group) and the $3'$-end (with its hydroxyl group) are very different, and so single strands have a well-defined *orientation*.

Figure 2.7(a) shows a single strand consisting of nucleotides A, C, A, T and G, with the $5'$-end at the first A nucleotide and the $3'$-end at the G nucleotide. A simpler notation for this DNA molecule is $5'$-ACATG-$3'$ or $3'$-GTACA-$5'$.

The same sequence of nucleotides A, C, A, T and G could also be linked in the opposite way. Then the phosphate group of the first A nucleotide would be connected to the hydroxyl group of the C nucleotide (instead of the other way round), and analogously for

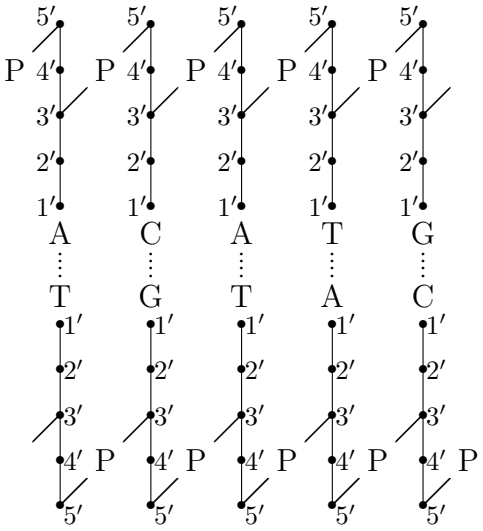


Figure 2.8: Schematical picture of a double-stranded DNA molecule.

the other nucleotides. The resulting strand is depicted in Figure 2.7(b) and it is denoted by 3′-ACATG-5′ or by 5′-GTACA-3′. The orientation of 3′-ACATG-5′ is opposite to that of 5′-ACATG-3′. For example, the G in 5′-ACATG-3′ can be connected by a phosphodiester bond to the C in 5′-CAT-3′, whereas the G in 3′-ACATG-5′ cannot.

Two single-stranded DNA molecules can bind through their bases, forming a double-stranded DNA molecule, as illustrated in Figure 2.8. The base of a nucleotide in one strand is connected to the base of a nucleotide in the other strand. Consecutive nucleotides in one strand are connected to consecutive nucleotides in the other strand. This is the second type of bond between nucleotides, called a *hydrogen bond*. In fact, one pair of nucleotides forms two or three hydrogen bonds, depending on the bases.

Hydrogen bonds between two nucleotides can be formed only if the nucleotides satisfy a complementarity constraint. An A can bind only (by two hydrogen bonds) to a T and vice versa. Similarly, a C can bind only (by three hydrogen bonds) to a G and vice versa. Hence, A and T are each other's *complements* and so are C and G. This complementarity is known as the *Watson-Crick complementarity*, after the scientists Watson and Crick that discovered it. A pair of complementary nucleotides connected by hydrogen bonds is called a *base pair*.

A second requirement for two strands to form a double-stranded DNA molecule is that they have opposite orientations. Since this is not the case for 5′-ACATG-3′ and 5′-TGTAC-3′, these two strands cannot form a double-stranded molecule. On the other hand, the strands 5′-ACATG-3′ and 3′-TGTAC-5′ *can* form a double-stranded DNA molecule, the one depicted in Figure 2.8.

The chromosomes in a cell contain double-stranded DNA. In total, the 46 chromosomes in a human cell contain over six billion (6×10^9) base pairs.

The hydrogen bonds between complementary strands of DNA are much weaker than the phosphodiester bonds between adjacent nucleotides in the same strand. Actually, hydrogen bonds are not even strong enough to bind a single pair of (complementary) nucleotides. It takes the cooperation of a number of pairs of nucleotides to keep two

5′-ACATG-3′	3′-GTACA-5′	3′-TGTAC-5′	5′-CATGT-3′
3′-TGTAC-5′	5′-CATGT-3′	5′-ACATG-3′	3′-GTACA-5′
(a)	(b)	(c)	(d)

Figure 2.9: Four ways to denote the same double-stranded DNA molecule. (a) Simple notation for the DNA molecule from Figure 2.8. (b) The result after a reflection in the Y-axis. (c) The result after a reflection (of the original notation) in the X-axis. (d) The result after a rotation (of the original notation) by an angle of 180°.

single strands together.

It is worth mentioning that the relative weakness of the hydrogen bonds (as compared to the phosphodiester bonds) is in fact essential for life. Because of this weakness, it is possible to separate the two strands of a double-stranded DNA molecule, while leaving the strands themselves intact. This happens, e.g., during cell division, when two exact copies of the DNA molecule must be made, one copy for each of the two new cells that are formed out of one. In this process, the two strands are separated and each of them serves as a template for a new complementary strand, built up of free nucleotides.[3]

The expression of genes also benefits from the relative weakness of the hydrogen bonds. Recall that a gene is a segment of a DNA molecule. The first step in the expression of a gene is the transcription of the gene into a related molecule called *RNA*. For this, the double-stranded DNA molecule is temporarily split at the gene's location, allowing for RNA to be formed. After the RNA molecule has been released, the two strands of DNA reunite. As a next step, the RNA molecule may be translated into a protein.

Double word notation

For single-stranded DNA molecules like the ones depicted in Figure 2.7, we introduced a simpler notation. For example, the molecule from Figure 2.7(a) can also be denoted by 5′-ACATG-3′. We can do something similar for double-stranded DNA molecules. The result for the molecule from Figure 2.8 is given in Figure 2.9(a). As illustrated by Figure 2.9(b)–(d), the same DNA molecule can be denoted in three more ways, resulting from reflections and a rotation of the molecule.

To simplify the notation even further, we can omit the explicit indication of the orientations in a double-stranded DNA molecule. This does not lead to confusion, when we adopt the convention that the upper strand in the notation is read from 5′-end to 3′-end (reading from left to right). Because the lower strand has an opposite orientation, it is read from 3′-end to 5′-end. The result is a *double word*, and the notation is called the double-word notation. Of course, by rotating a double word by an angle of 180°, we obtain a double word denoting the same molecule. The two possible double words for our example molecule are given in Figure 2.10.

If the rotation yields the same double word, then the molecule denoted is called a *palindrome*[4] – actually palindrome molecules are quite common in molecular biology. An

[3]In fact, the addition of complementary nucleotides already starts when part of the double-stranded molecule is separated.

[4]Unfortunately, the term 'palindrome' has different (though related) meanings in linguistics and in molecular science.

$$\begin{array}{cc} \text{ACATG} & \text{CATGT} \\ \text{TGTAC} & \text{GTACA} \\ \text{(a)} & \text{(b)} \end{array}$$

Figure 2.10: The double-word notation for the double-stranded DNA molecule from Figure 2.8 and Figure 2.9. (a) The double word corresponding to Figure 2.9(a). (b) The result after a rotation of 180°. This double word corresponds to Figure 2.9(d).

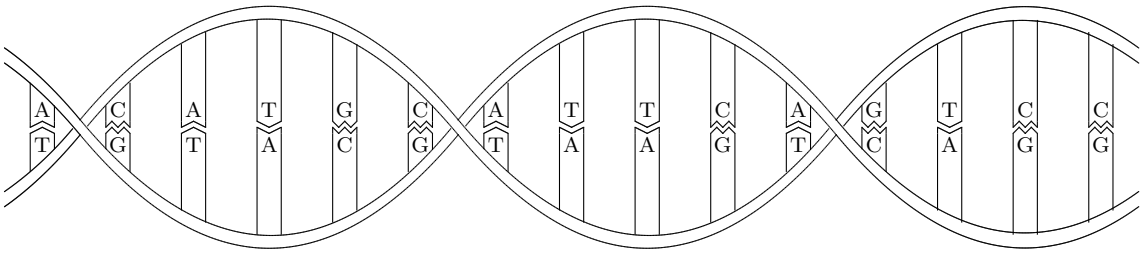


Figure 2.11: The typical double-helix structure of a double-stranded DNA molecule.

example of a palindrome is $\begin{matrix} \text{CATG} \\ \text{GTAC} \end{matrix}$.

We want to emphasize that the double word notation only tells us which nucleotides are connected to which other nucleotides by which type of bond. It does not provide information about the spatial structure of the DNA molecule. The notation may suggest that the DNA molecules are linear, i.e., that the nucleotides in each strand are spatially organized in a straight line. This is, however, not the case, and in fact, it would be physically impossible in vivo. The largest human chromosome would then be more than 8cm long, which would not fit in the average cell. In general, the spatial structure of a double-stranded DNA molecule is very complex. A typical aspect of this structure is that usually the two strands are twisted around each other, like a winding staircase, forming the famous double helix depicted in Figure 2.11.

Nicks, gaps and other deviations

DNA molecules are not always 'perfect'. That is, they are not always comprised of two equally long, perfectly complementary strands of nucleotides, like the molecule in Figure 2.10. There exist many types of deviations in the structure of DNA molecules. We list six of them:

Nick Sometimes the phosphodiester bond between adjacent nucleotides in the same strand is missing. The molecule does not fall apart though, because a phosphodiester bond binds the complementary nucleotides in the other strand. The non-existence of a phosphodiester bond between two nucleotides is referred to as a *nick*. In the double-word notation for DNA molecules, we denote a nick by the symbol $^{\triangledown}$ in the upper word and by the symbol $_{\triangle}$ in the lower word. Hence, the DNA molecule given in Figure 2.12(a) has nicks between A and C in the upper word and between T and A in the lower word.

Gap A *gap* results when one or more consecutive nucleotides in one strand of a DNA molecule miss their complementary nucleotides in the other strand. The nucleotides

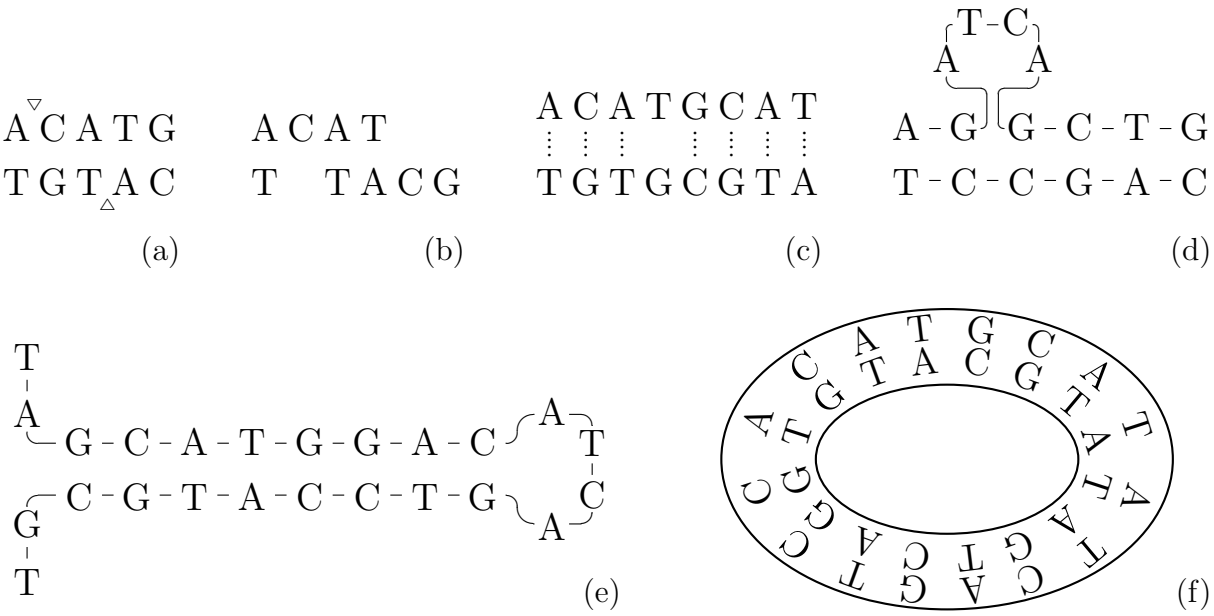


Figure 2.12: Some deviations from the standard double-stranded DNA molecule. (a) A molecule with two nicks. (b) A molecule with two gaps. (c) A molecule with a mismatch between T in the upper strand and G in the lower strand. Hydrogen bonds present are explicitly indicated. (d) A molecule with a bulge in the upper strand. Phosphodiester bonds present are explicitly indicated. (e) A single-stranded molecule with a hairpin loop. Phosphodiester bonds present are explicitly indicated. (f) A circular molecule.

on both sides of the gap (if these nucleotides exist, i.e. if the gap is not located at an end of the molecule) are not linked directly by a phosphodiester bond. This is illustrated in Figure 2.12(b). The molecule depicted here contains two gaps: one in each strand.

When we have a gap at an end of the DNA molecule (like we have in Figure 2.12(b)), the non-complemented nucleotides present at the other strand form a so-called *sticky end*. This name refers to the fact that the non-complemented nucleotides stick easily to a strand of complementary nucleotides.

Mismatch We have a *mismatch*, if two nucleotides at corresponding positions in the strands of a double-stranded DNA molecule are not complementary. As a result, these two nucleotides cannot form proper hydrogen bonds. When there are enough neighbouring nucleotides in both strands that *are* each other's complements, the two strands as a whole can still stick together. This situation is depicted in Figure 2.12(c).

Bulge A *bulge* is a piece of single-stranded DNA inserted between two nucleotides in one strand of a double-stranded DNA molecule. The two nucleotides involved are not directly connected by a phosphodiester bond, whereas their respective complements in the other strand are. This phenomenon is depicted in Figure 2.12(d).[5] Note the similarity between a DNA molecule with a bulge and one with a gap. Both molecules have a subsequence of unpaired nucleotides in one of the strands. In case of a bulge,

[5]In practice, the molecule will be kinked at the site of the bulge. In our example, with the bulge in the upper strand, the molecule will bend down.

the nucleotides flanking the subsequence on the opposite strand are connected by a phosphodiester bond. In case of a gap, this bond is missing.

Hairpin loop When a single strand of DNA or RNA contains a subsequence of nucleotides that is complementary to another subsequence of the same strand in reverse order, we may obtain a *hairpin loop*: the strand folds back and hydrogen bonds between the complementary nucleotides are formed. This is illustrated by Figure 2.12(e). Hairpins occur in vivo, e.g., in RNA molecules that are used in the synthesis of proteins.

Circular molecule DNA molecules may be *circular*. That is, the 5′-end of a strand may get connected by a phosphodiester bond to the 3′-end of the same strand. In case of a double-stranded molecule, this may happen to both strands, as is depicted in Figure 2.12(f). Circular DNA molecules occur, e.g., in viruses and *ciliates* (some kind of unicellular organisms). In the latter case, they are formed as a by-product during the conversion of micronuclear DNA into macronuclear DNA, which are two versions of the DNA each residing in its own nucleus in the cell. One part of this conversion involves the excision (removal) of fragments of DNA from the micronuclear DNA. When a fragment is excised, the two pieces of DNA flanking the fragment are connected directly, and the removed fragment forms a circular DNA molecule, see [Prescott, 1994].

In this thesis, we describe and analyse expressions for DNA molecules that may have nicks and/or gaps. We do not consider other deviations, because we wanted to start simple, with a limited set of operators. This limited set of operators turned out to be rich enough to derive many interesting results from.

2.3 DNA computing

DNA computing is the field of research that studies how DNA molecules can be used to perform computations. By the nature of the subject, DNA computing is a highly interdisciplinary field of research. In this section, we discuss two important contributions to the field. First, we consider *splicing systems*, which form a theoretical model for the way that double-stranded DNA molecules can be modified with the use of *restriction enzymes*. Second, we describe how real (physical) DNA molecules can be used to solve a notoriously hard decision problem.

2.3.1 Splicing systems

Splicing systems were introduced by Tom Head in [1987]. Head's purpose was to relate formal language theory to the world of macromolecules like DNA. He modelled the action of restriction enzymes on DNA in terms of a formal language. Restriction enzymes are enzymes that recognize a specific sequence of nucleotides in a double-stranded DNA molecule, i.e., a sequence specific for the enzyme, and cut the DNA in a special way.

For example, consider a DNA molecule as depicted in Figure 2.13(a). The restriction enzyme *EcoRI* recognizes the segment $\begin{smallmatrix} \text{GAATTC} \\ \text{CTTAAG} \end{smallmatrix}$ and cleaves the molecule in such a way that the two molecules with 5′-*overhangs* (sticky ends) in Figure 2.13(b) result.

When the two molecules get in each other's proximity, they may reassociate in the presence of a ligase to seal the nicks in both strands. Now, when EcoRI finds two molecules

$$\begin{array}{l} \cdots \text{NNNNGAATTCNNNN} \cdots \\ \cdots \text{NNNNCTTAAGNNNN} \cdots \end{array} \quad \text{(a)}$$

$$\begin{array}{ll} \cdots \text{NNNNG-3}' & \text{5}'\text{-AATTCNNNN} \cdots \\ \cdots \text{NNNNCTTAA-5}' & \text{3}'\text{-GNNNN} \cdots \end{array} \quad \text{(b)}$$

Figure 2.13: The effect of restriction enzyme EcoRI. (a) A double-stranded DNA molecule with recognition site $\begin{smallmatrix} \text{GAATTC} \\ \text{CTTAAG} \end{smallmatrix}$. The N's represent arbitrary nucleotides satisfying the complementarity constraint. (b) The two DNA molecules with 5′-overhangs after cleavage by EcoRI.

X_1 and X_2 with the required recognition site $\begin{smallmatrix} \text{GAATTC} \\ \text{CTTAAG} \end{smallmatrix}$, they may both be cleaved. The left-hand molecule derived from X_1 may reassociate with the right-hand molecule derived from X_2, and vice versa, simply because their sticky ends match. In fact, because the sticky ends form the palindrome $\begin{smallmatrix} \text{AATT} \\ \text{TTAA} \end{smallmatrix}$, the left-hand molecule derived from X_1 may reassociate with the left-hand molecule derived from X_2, and similarly for the right-hand molecules. This way, starting from a given set of double-stranded DNA molecules, EcoRI may produce a completely different set of molecules.

EcoRI is not the only restriction enzyme. There exist many restriction enzymes, with different (or sometimes the same) recognition sites and leaving different sticky ends after cleavage. It is also possible to obtain 3′-overhangs instead of 5′-overhangs.

Now, a splicing system is a language-theoretic model for such a system of DNA molecules and enzymes. The main ingredients of this model are

1. a set of initial strings, corresponding to the initial DNA molecules, and

2. rules to obtain new strings from strings with a certain common substring, corresponding to the way new molecules may result from molecules containing the recognition site of a certain restriction enzyme.

Later, different types of splicing systems were introduced, allowing, e.g., more general types of rules. In particular, the effect of restriction enzymes leaving blunt ends (i.e., without overhangs) when they cut a molecule, can also be described. In this case, there are no sticky ends that might facilitate the left-hand side of one molecule to reassociate with the right-hand side of another molecule. Instead, the two submolecules are joined together directly by a ligase enzyme.

Head posed the question what types of languages (sets of strings) may result from a given set of initial strings and a given set of rules. Many researchers followed up on this question, yielding a wealth of results. An overview of the results of the first ten years can be found in [Head et al., 1997].

2.3.2 Adleman's experiment

Although in nature, DNA molecules occur mainly in cells of organisms, they can also be produced (and manipulated) in vitro, in a laboratory. In fact, there are machines available that can synthesize any given single strand of DNA up to about 200 nucleotides. Such relatively short single strands can then be used to generate arbitrarily long double-stranded DNA molecules.

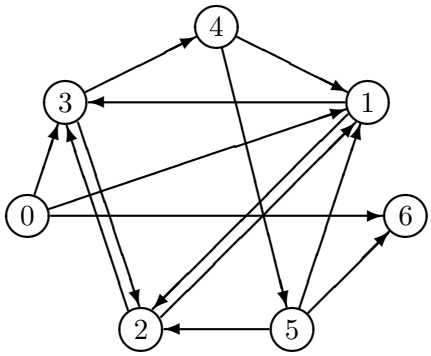


Figure 2.14: Graph for which Adleman solved the directed Hamiltonian path problem using DNA molecules.

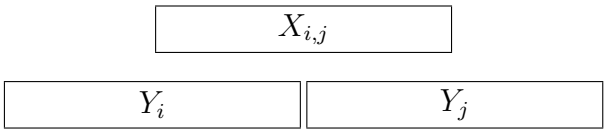


Figure 2.15: Single strands of DNA as used by Adleman to encode nodes and edges. The strands Y_i and Y_j encode the nodes i and j, respectively. The strand $X_{i,j}$ encodes the edge (i, j).

Leonard Adleman [1994] (see also [Adleman, 1998]) realized that DNA molecules in a lab can be used to store information and perform computations. To demonstrate this, he used such molecules to solve a small instance of the *directed Hamiltonian path problem*.

In this problem, we are given a directed graph and two nodes v_{in} and v_{out} in this graph. We must decide whether or not there is a Hamiltonian path in this graph from v_{in} to v_{out}, i.e., a path from v_{in} to v_{out} that visits every node exactly once. This problem is $\mathcal{NP}$-complete, which means that there probably does not exist an efficient algorithm for the problem. The only algorithms known for the problem require more than polynomial time.

Adleman considered the seven-node graph in Figure 2.14, with v_{in} is node 0 and v_{out} is node 6. It can easily be verified that there is a unique Hamiltonian path in this graph, viz $0 \to 1 \to 2 \to 3 \to 4 \to 5 \to 6$.

For every node in the graph, Adleman defined a single-stranded DNA molecule, consisting of twenty nucleotides. Let us use Y_i to denote the DNA molecule for node i.

The edges between the nodes were also encoded as single-stranded DNA molecules of twenty nucleotides. For an edge from node i to node j, Adleman joined the second half of the Watson-Crick complement of Y_i and the first half of the complement of Y_j. Let us call the resulting molecule $X_{i,j}$, as in Figure 2.15.

Note that an edge from j to i would result in a different molecule $X_{j,i}$. Hence, this encoding preserves edge orientation. Indeed, in Adleman's description, the single-stranded molecules Y_i encoding the nodes were lower strands, i.e., their orientation ran from 3'-end to 5'-end.

Adleman took certain quantities of DNA molecules corresponding to the nodes and the edges of the seven-node graph, put them together under the right conditions and in

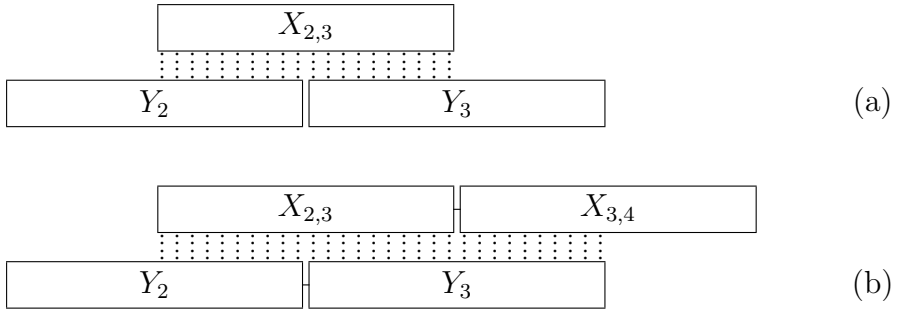


Figure 2.16: Single strands of DNA forming a double-stranded molecule, representing a path in the graph in Adleman's experiment. (a) Hydrogen bonds are established between strands encoding node 2, edge $(2, 3)$ and node 3. (b) Result after a strand encoding edge $(3, 4)$ has attached to the molecule.

the presence of ligase, and waited for the result.

To understand the outcome of this *self-assembly* process, we consider the small example in Figure 2.16. The molecule $X_{2,3}$ (corresponding to the edge from 2 to 3) may form hydrogen bonds with part of Y_2 and part of Y_3, yielding a double-stranded DNA molecule with sticky ends at the left-hand side (consisting of the first half of Y_2) and the right-hand side (the second half of Y_3), as depicted in Figure 2.16(a). The ligase enzyme establishes a connection between the rightmost nucleotide of Y_2 and the leftmost nucleotide of Y_3.

The edge from node 3 to node 4 is encoded as a strand $X_{3,4}$, which may now attach to the sticky end to the right, see Figure 2.16(b). This way, longer and longer DNA molecules arise.

Given a proper encoding of nodes into nucleotide sequences, two nodes (the corresponding lower strands) may only be linked if the graph has an edge between them. In other words, each double-stranded DNA molecule that results corresponds uniquely to a path in the graph. Of course, the paths that are formed are completely random, and hence, most of them are not Hamiltonian paths. There may, however, be some among them.

In order to check for the presence of a Hamiltonian path from node 0 to node 6, Adleman used some biomolecular tools to

1. keep only the molecules beginning with the nucleotide sequence for node 0 and ending with the sequence for node 6

2. keep only the molecules of length 140, i.e., containing seven nodes

3. split the double-stranded DNA molecules into single strands and keep only the molecules containing the nucleotide sequence for every node.

Since there were molecules remaining after these three steps, Adleman could conclude that a Hamiltonian path from 0 to 6 was found.

In his experiment, Adleman exploited three properties of DNA molecules. First, because the molecules are so small, compared to the units from which a computer is built, they are a very efficient means to store information (in this case: concerning the presence of nodes and edges in a graph). Second, the formation of larger molecules (representing

random paths in the graph) from relatively short single strands is an automatic and highly parallel process. Third, this process requires very little energy.

The second property, the parallel computation of paths in the graph, has important implications for the time complexity of the process. While existing algorithms for the Hamiltonian path problem on conventional computers take more than polynomial time, Adleman expected that the time required by his biomolecular algorithm would scale linearly with the number of nodes in the graph.

However, as Adleman [1994] acknowledged, this speed-up comes at a price. First, in general, the number of edges in a graph of N nodes grows quadratically with N. Therefore, the number of *different* strands encoding a node or an edge in the graph should also grow quadratically with N. Even worse, the number of possible paths in a graph of N nodes grows more than exponentially with N. This implies that the number of *copies* of each strand encoding a node or an edge in the graph, should also grow more than exponentially with N, to ensure (with a very high probability) that a Hamiltonian path from v_{in} to v_{out} (if it exists) is among the random paths that are constructed by the self-assembly process. Consequently, the volume of DNA that is needed to solve the Hamiltonian path problem for graphs of practical interest becomes too large. For example, a simple calculation in [Hartmanis, 1995] showed that the weight of DNA required to encode all paths of length N in a graph with $N = 200$ nodes, would be more than the weight of the earth.

Another problem with the biomolecular operations used in the experiment is that they are far from error free. For example, as we have seen in Section 2.2, mismatches may occur. It is possible that two (parts of) strands form hydrogen bonds, although they are not completely complementary. As a consequence, in the case of the Hamiltonian path problem, nodes may be connected that do not have an edge between them in the graph. The algorithms using DNA molecules should be adjusted to that.

Nevertheless, Adleman's experiment made it clear that biomolecules have a computational potential, not just in theory but also in the lab. This inspired many researchers to come up with other applications of this kind, or to examine how the formation of undesirable bonds may be avoided, see, e.g., [Lipton, 1995], [Boneh et al., 1996], [Deaton et al., 1997], and [Kari et al., 2005].

In fact, Adleman's experiment initiated a series of annual conferences, known today as International Conference on DNA Computing and Molecular Programming, see www.dna-computing.org.

Part I

DNA Expressions in General

Chapter 3

Formal DNA Molecules

Before we define the expressions in our DNA language, we want to be more precise about their meaning – the semantics of the DNA expressions. For this purpose, we formalize the double-word notation for DNA molecules that may contain nicks and gaps.

In this chapter, we introduce and analyse the resulting *formal DNA molecules*. We first consider $\mathcal{N}$-words, non-empty strings over the alphabet of the four different nucleotides. We subsequently define the formal DNA molecules and we identify *components* of these molecules. We finally discuss some properties, relations and functions of formal DNA molecules.

3.1 $\mathcal{N}$-words

Let $\mathcal{N} = \{\text{A, C, G, T}\}$ be the alphabet of nucleotides, and let the elements of $\mathcal{N}$ be called $\mathcal{N}$*-letters*. We use the symbol a (possibly with a subscript) to denote single $\mathcal{N}$-letters. A *non-empty* string over $\mathcal{N}$ is called an $\mathcal{N}$*-word*. Obviously, the set $\mathcal{N}^+$ of $\mathcal{N}$-words is closed under concatenation. We reserve the symbol α (possibly with a subscript) to denote $\mathcal{N}$-words.

In general, when α is an $\mathcal{N}$-word (e.g. $\alpha = \text{ACATG}$), it corresponds to two single-stranded DNA molecules: $5'$-α-$3'$ and $3'$-α-$5'$. Only if α happens to be a *palindrome* (in the linguistic sense), e.g., if $\alpha = \text{ACTCA}$, these two DNA molecules are the same.

The symbol c denotes the complement function. It is an endomorphism on $\mathcal{N}^*$, specified by

$$c(\text{A}) = \text{T}, \qquad c(\text{C}) = \text{G}, \qquad c(\text{G}) = \text{C}, \qquad c(\text{T}) = \text{A}.$$

Thus, for an $\mathcal{N}$-word α, $c(\alpha)$ results by replacing each letter of α by its Watson-Crick complement. For example, $c(\text{ACATG}) = \text{TGTAC}$.

Note that $c(\alpha)$ itself is not a DNA molecule, with an orientation. It is just an $\mathcal{N}$-word, corresponding to two single-stranded DNA molecules with opposite orientations. Of course, for a given $\mathcal{N}$-word α, the two strands $5'$-α-$3'$ and $3'$-$c(\alpha)$-$5'$ (for example) are each other's Watson-Crick complement in the molecular sense.

3.2 Definition of formal DNA molecules

In the double-word notation of a perfect, double-stranded DNA molecule, every symbol in the upper word corresponds to a symbol in the lower word. Two such corresponding

symbols denote a base pair – two complementary nucleotides that are connected through a hydrogen bond.

In our formalization of the double-word notation, a base pair is represented by a composite symbol $x = \binom{x^+}{x^-}$. Here x^+ stands for the nucleotide in the upper word and x^- stands for the nucleotide in the lower word.

Since we also want to denote DNA molecules with nicks and gaps, we need to extend this notation. We first consider the gaps. When we have a gap in either of the strands, we still use composite symbols $\binom{x^+}{x^-}$, but the missing nucleotides are denoted by $-$. We thus have $x^+, x^- \in \mathcal{N} \cup \{-\}$. For convenience, we will speak of a base pair also if one of two complementary nucleotides is missing. If both nucleotides are present, we may call the base pair *complete*.

Of course, the value of x^+ restricts the value of x^-, and vice versa. Because of the Watson-Crick complementarity and the fact that a missing nucleotide cannot face another missing nucleotide, only twelve of the twenty-five possible composite symbols $\binom{x^+}{x^-}$ are really allowed: $\binom{\mathrm{A}}{\mathrm{T}}, \binom{\mathrm{C}}{\mathrm{G}}, \binom{\mathrm{G}}{\mathrm{C}}, \binom{\mathrm{T}}{\mathrm{A}}, \binom{\mathrm{A}}{-}, \binom{\mathrm{C}}{-}, \binom{\mathrm{G}}{-}, \binom{\mathrm{T}}{-}, \binom{-}{\mathrm{A}}, \binom{-}{\mathrm{C}}, \binom{-}{\mathrm{G}}, \binom{-}{\mathrm{T}}$. The set containing these twelve composite symbols is denoted by $\mathcal{A}$.

For future use, we partition $\mathcal{A}$ into three subsets: $\mathcal{A}_\pm = \left\{\binom{\mathrm{A}}{\mathrm{T}}, \binom{\mathrm{C}}{\mathrm{G}}, \binom{\mathrm{G}}{\mathrm{C}}, \binom{\mathrm{T}}{\mathrm{A}}\right\}$, $\mathcal{A}_+ = \left\{\binom{\mathrm{A}}{-}, \binom{\mathrm{C}}{-}, \binom{\mathrm{G}}{-}, \binom{\mathrm{T}}{-}\right\}$ and $\mathcal{A}_- = \left\{\binom{-}{\mathrm{A}}, \binom{-}{\mathrm{C}}, \binom{-}{\mathrm{G}}, \binom{-}{\mathrm{T}}\right\}$. The elements of $\mathcal{A}$ are called *$\mathcal{A}$-letters*, the elements of $\mathcal{A}_\pm$ are *double $\mathcal{A}$-letters*, the elements of $\mathcal{A}_+$ are *upper $\mathcal{A}$-letters*, and the elements of $\mathcal{A}_-$ are *lower $\mathcal{A}$-letters*. Letters can be used to form strings. A non-empty string over $\mathcal{A}$ is called an *$\mathcal{A}$-word*. Analogously, we have a *double $\mathcal{A}$-word* (with letters from $\mathcal{A}_\pm$), an *upper $\mathcal{A}$-word* (with letters from $\mathcal{A}_+$) and a *lower $\mathcal{A}$-word* (with letters from $\mathcal{A}_-$).

We also need symbols to denote nicks in a double word. There are three possibilities for the connection structure of two adjacent base pairs in a double word: there can be a nick in the upper word, there can be a nick in the lower word, or there can be no nick at all between the base pairs. Note that there cannot be both a nick in the upper word and a nick in the lower word between two adjacent base pairs. In such a situation, there would be no connection whatsoever between the base pairs, so they would be parts of different DNA molecules.

The case that there is no nick at all is our default; it is not denoted explicitly. A nick in the upper word is denoted by $^\triangledown$ and a nick in the lower word by $_\triangle$. We call $^\triangledown$ and $_\triangle$ the *nick letters* – $^\triangledown$ is the *upper* nick letter, and $_\triangle$ is the *lower* nick letter.

Now, a complete description of a DNA molecule possibly containing nicks and gaps can be given by a non-empty string X over $\mathcal{A}_{\triangledown\triangle} = \mathcal{A} \cup \{^\triangledown, _\triangle\}$.

Example 3.1 The DNA molecules depicted in Figure 2.10(a), Figure 2.12(a) and Figure 2.12(b) are denoted by

$$\begin{aligned} X_1 &= \tbinom{\mathrm{A}}{\mathrm{T}}\tbinom{\mathrm{C}}{\mathrm{G}}\tbinom{\mathrm{A}}{\mathrm{T}}\tbinom{\mathrm{T}}{\mathrm{A}}\tbinom{\mathrm{G}}{\mathrm{C}}, \\ X_2 &= \tbinom{\mathrm{A}}{\mathrm{T}}^{\triangledown}\tbinom{\mathrm{C}}{\mathrm{G}}\tbinom{\mathrm{A}}{\mathrm{T}}_{\triangle}\tbinom{\mathrm{T}}{\mathrm{A}}\tbinom{\mathrm{G}}{\mathrm{C}}, \text{ and} \\ X_3 &= \tbinom{\mathrm{A}}{\mathrm{T}}\tbinom{\mathrm{C}}{-}\tbinom{\mathrm{A}}{\mathrm{T}}\tbinom{\mathrm{T}}{\mathrm{A}}\tbinom{-}{\mathrm{C}}\tbinom{-}{\mathrm{G}}, \end{aligned}$$

respectively. X_1 has length 5, X_2 has length 7, and X_3 has length 6. ■

Not every string over $\mathcal{A}_{\triangledown\triangle}$ represents a DNA molecule. The requirements that strings over $\mathcal{A}_{\triangledown\triangle}$ need to satisfy follow from three observations on DNA molecules:

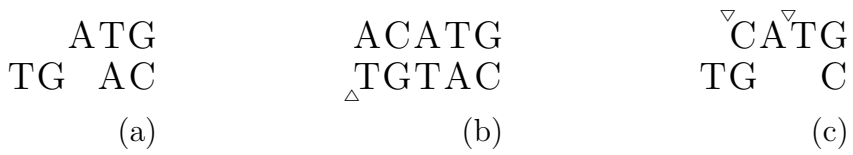


Figure 3.1: Examples of impossible DNA molecules. (a) A gap in one strand that is adjacent to a gap in the other strand. (b) A nick at the left end of the molecule. (c) Nicks between base pairs that are not (both) complete.

1. To enable at least one phosphodiester bond between adjacent base pairs, a gap in one strand cannot be adjacent to a gap in the other strand (see Figure 3.1(a)).

2. A nick may occur only *between* two base pairs. In particular, it cannot occur at the left end or the right end of a DNA molecule (see Figure 3.1(b)).

3. Since a nick is a missing phosphodiester bond between two adjacent nucleotides in the same strand, we really need to have nucleotides on both sides of the nick. Moreover, the complementary nucleotides in the other strand must be present and they must be connected by a phosphodiester bond. Hence, a nick may occur only between two complete base pairs (see Figure 3.1(c)).

Now, we are ready to define our formalization of DNA molecules that may contain nicks and gaps:

Definition 3.2 *A formal DNA molecule is a string* $X = x_1 x_2 \dots x_r$ *with* $r \geq 1$ *and for* $i = 1, \dots, r$, $x_i \in \mathcal{A}_{\triangledown\triangle}$, *satisfying*

1. *if* $x_i \in \mathcal{A}_+$, *then* $x_{i+1} \notin \mathcal{A}_-$ $(i = 1, 2, \dots, r-1)$,

if $x_i \in \mathcal{A}_-$, *then* $x_{i+1} \notin \mathcal{A}_+$ $(i = 1, 2, \dots, r-1)$,

2. $x_1, x_r \in \mathcal{A}$,

3. *if* $x_i \in \{\triangledown, \triangle\}$, *then* $x_{i-1}, x_{i+1} \in \mathcal{A}_\pm$ $(i = 2, 3, \dots, r-1)$.

The language of all formal DNA molecules is denoted by $\mathcal{F}$. Since $X \in \mathcal{F}$ is called a molecule (albeit 'formal'), we will refer to the sequence of (possibly missing) nucleotides x_i^+ and upper nick letters in X as the upper strand of X. The lower strand of X is defined analogously.

Note, however, that it does not make sense to talk about the upper strand and the lower strand of a (physical) DNA molecule, because a rotation of the molecule by an angle of 180° would change the upper strand into a lower strand and vice versa.

If a formal DNA molecule does not contain upper nick letters (or lower nick letters), then we say that its upper strand (lower strand, respectively) is *nick free*. If a formal DNA molecule does not contain nick letters at all, then the molecule itself is called nick free.

When we build up a formal DNA molecule from left to right, the choice of a certain letter completely determines the possibilities for the next letter. For example: a nick letter must be succeeded by a double $\mathcal{A}$-letter; an upper $\mathcal{A}$-letter may be succeeded by either an

other upper $\mathcal{A}$-letter or a double $\mathcal{A}$-letter, or it may terminate the formal DNA molecule (see Definition 3.2). With this in mind, it is easy to construct a right-linear grammar that generates the language $\mathcal{F}$. We thus have:

Lemma 3.3 *The language $\mathcal{F}$ of formal DNA molecules is regular.*

3.3 Components of a formal DNA molecule

Let $X = x_1 \dots x_r$ be a formal DNA molecule, with $x_i \in \mathcal{A}_{\triangledown\triangle}$ for $i = 1, \dots, r$. A *formal DNA submolecule* of X is a substring X^s of X such that X^s is a formal DNA molecule. It is easy to see that

Lemma 3.4 *A substring X^s of a formal DNA molecule X is a formal DNA molecule if and and only if $|X^s| \geq 1$ and $L(X^s), R(X^s) \in \mathcal{A}$.*

Hence, X^s should not be empty and neither its first symbol nor its last symbol should be a nick letter.

If a formal DNA submolecule X^s of X is an upper $\mathcal{A}$-word, a lower $\mathcal{A}$-word or a double $\mathcal{A}$-word, and $|X^s| \geq 2$, then it is possible to simplify the notation for X^s and X. Let $\alpha = a_1 \dots a_l$ be an $\mathcal{N}$-word with $a_i \in \mathcal{N}$ $(i = 1, \dots, l)$, and let X^s be a formal DNA submolecule of X with $X^s = x_{i_0} \dots x_{i_0+l-1}$ for some i_0 with $1 \leq i_0 \leq r-l+1$ (so $|X^s| = l$). If $X^s = \binom{a_1}{-} \cdots \binom{a_l}{-}$, then we may write

$$X^s = \binom{\alpha}{-} \text{ and } X = x_1 \dots x_{i_0-1} \binom{\alpha}{-} x_{i_0+l} \dots x_r.$$

Similarly, if $X^s = \binom{-}{a_1} \binom{-}{a_2} \cdots \binom{-}{a_l}$, then we may write

$$X^s = \binom{-}{\alpha} \text{ and } X = x_1 \dots x_{i_0-1} \binom{-}{\alpha} x_{i_0+l} \dots x_r.$$

Finally, if $X^s = \binom{a_1}{c(a_1)} \binom{a_2}{c(a_2)} \cdots \binom{a_l}{c(a_l)}$, then we may write

$$X^s = \binom{\alpha}{c(\alpha)} \text{ and } X = x_1 \dots x_{i_0-1} \binom{\alpha}{c(\alpha)} x_{i_0+l} \dots x_r.$$

By simplifying the notation, we may seem to extend the alphabet of $\mathcal{F}$ with infinitely many symbols $\binom{\alpha}{-}$, $\binom{-}{\alpha}$ and $\binom{\alpha}{c(\alpha)}$. We want to emphasize, however, that we actually only simplify the *presentation* of the formal DNA molecules. The formal DNA molecules themselves do not change; they are still strings over the finite alphabet $\mathcal{A}_{\triangledown\triangle}$. In particular, the length of a formal DNA molecule $X = x_1 \dots x_r$ with $x_i \in \mathcal{A}_{\triangledown\triangle}$ for $i = 1, \dots, r$ remains r, even if X is written in a simplified notation.

Definition 3.5 *Let X be a formal DNA molecule. Then the decomposition of X is the sequence $x'_1, \dots, x'_k$ of $k \geq 1$ non-empty strings over $\mathcal{A}_{\triangledown\triangle}$ such that*

- $X = x'_1 \dots x'_k$,
- *for $i = 1, \dots, k$, x'_i is either an upper $\mathcal{A}$-word, or a lower $\mathcal{A}$-word, or a double $\mathcal{A}$-word, or a nick letter, and*
- *for $i = 1, \dots, k-1$, if x'_i is an upper $\mathcal{A}$-word, then x'_{i+1} is not an upper $\mathcal{A}$-word, and similarly for lower $\mathcal{A}$-words and double $\mathcal{A}$-words.*

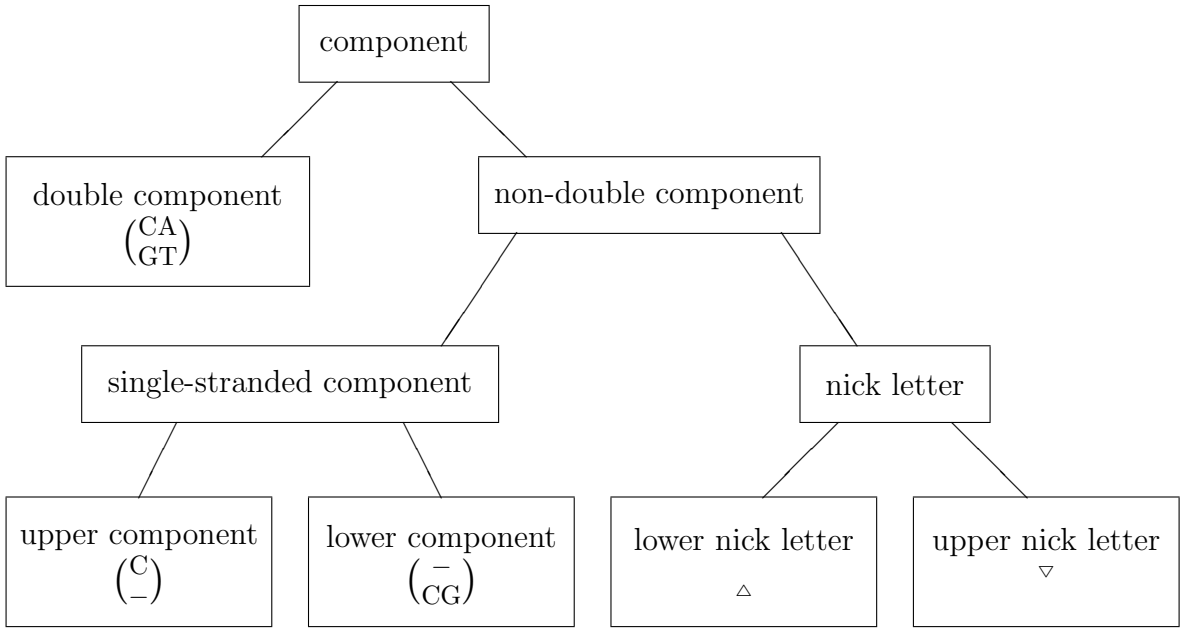


Figure 3.2: Relations between different types of components. Components can be divided into double components and non-double components, non-double components can in turn be divided into single-stranded components and nick letters, etcetera.

Hence, the decomposition of X cannot be simplified any further by concatenating consecutive elements of the same type. For the ease of notation, we will in general omit the commas and write $x'_1 \dots x'_k$ instead of $x'_1, \dots, x'_k$.

Example 3.6 The decompositions of the formal DNA molecules from Example 3.1 (denoting the molecules shown in Figure 2.10, Figure 2.12(a) and Figure 2.12(b)) are

$$\begin{aligned} X_1 &= \tbinom{\text{ACATG}}{\text{TGTAC}}, \\ X_2 &= \tbinom{\text{A}}{\text{T}}^{\triangledown}\tbinom{\text{CA}}{\text{GT}}_{\triangle}\tbinom{\text{TG}}{\text{AC}} \text{ and} \\ X_3 &= \tbinom{\text{A}}{\text{T}}\tbinom{\text{C}}{-}\tbinom{\text{AT}}{\text{TA}}\tbinom{-}{\text{CG}}, \end{aligned}$$

respectively. ■

If $x'_1 \dots x'_k$ for some $k \geq 1$ is the decomposition of a formal DNA molecule X, then the substrings x'_i are called the *components* of X. For $i = 1, \dots, k$, if x'_i is an upper $\mathcal{A}$-word (lower $\mathcal{A}$-word or double $\mathcal{A}$-word), then x'_i is called an *upper component* (*lower component* or *double component*, respectively) of X. If x'_i is not a double component, then we may also call it a *non-double component* of X. Upper components and lower components of X are also called *single-stranded components* of X. The (rooted) tree in Figure 3.2 shows the relations between the different types of components.

Often, we will use pictures as the one in Figure 3.3 to depict a formal DNA molecule. For example, the $\mathcal{N}$-word α_3 in this picture represents the lower component $\binom{-}{\alpha_3}$, the $\mathcal{N}$-word α_5 represents the upper component $\binom{\alpha_5}{-}$, the $\mathcal{N}$-word α_6 represents the double $\mathcal{A}$-word $\binom{\alpha_6}{c(\alpha_6)}$ (which is not a component!), the first occurrence of the symbol $\triangle$ represents

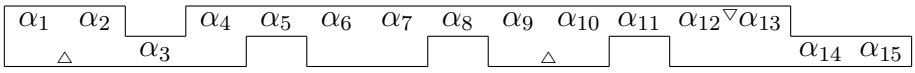


Figure 3.3: Pictorial representation of a formal DNA molecule X. The $\mathcal{N}$-words $\alpha_1, \ldots, \alpha_{15}$ represent upper $\mathcal{A}$-words, lower $\mathcal{A}$-words and double $\mathcal{A}$-words occurring in X. The symbols $\triangle$ and $\triangledown$ represent lower nick letters and upper nick letters, respectively.

a lower nick letter between the double components $\binom{\alpha_1}{c(\alpha_1)}$ and $\binom{\alpha_2}{c(\alpha_2)}$, and the symbol $\triangledown$ represents an upper nick letter between the double components $\binom{\alpha_{12}}{c(\alpha_{12})}$ and $\binom{\alpha_{13}}{c(\alpha_{13})}$.

In a formal DNA molecule, double components and non-double components alternate:

Lemma 3.7 *Let X be a formal DNA molecule and let $x'_1 \ldots x'_k$ for some $k \geq 1$ be the decomposition of X. Then*

- *for $i = 1, \ldots, k-1$, if x'_i is a non-double component, then x'_{i+1} is a double component;*
- *for $i = 1, \ldots, k-1$, if x'_i is a double component, then x'_{i+1} is a non-double component.*

Indeed, the formal DNA molecule depicted in Figure 3.3 consists of a double component $\binom{\alpha_1}{c(\alpha_1)}$, a lower nick letter, a double component $\binom{\alpha_2}{c(\alpha_2)}$, a lower component $\binom{-}{\alpha_3}$, a double component $\binom{\alpha_4}{c(\alpha_4)}$, an upper component $\binom{\alpha_5}{-}$, a double component $\binom{\alpha_6\alpha_7}{c(\alpha_6\alpha_7)}$, an upper component $\binom{\alpha_8}{-}$, a double component $\binom{\alpha_9}{c(\alpha_9)}$, a lower nick letter, a double component $\binom{\alpha_{10}}{c(\alpha_{10})}$, an upper component $\binom{\alpha_{11}}{-}$, a double component $\binom{\alpha_{12}}{c(\alpha_{12})}$, an upper nick letter, a double component $\binom{\alpha_{13}}{c(\alpha_{13})}$ and a lower component $\binom{-}{\alpha_{14}\alpha_{15}}$.

Proof: If for some i with $1 \leq i \leq k-1$, x'_i is a double component, then by the definition of the decomposition, the next component x'_{i+1} is a non-double component. Because nick letters can only occur between two double components and because an upper component cannot occur next to a lower component (see Definition 3.2), the reverse is also true: if for some i with $1 \leq 1 \leq k-1$, x'_i is an upper component, a lower component or a nick letter, then the next component x'_{i+1} is a double component. □

Two special cases of this result will also turn out to be useful:

Corollary 3.8 *Let X be a nick free formal DNA molecule and let $x'_1 \ldots x'_k$ for some $k \geq 1$ be the decomposition of X.*

1. *For $i = 1, \ldots, k$, x'_i is either an upper component, or a lower component, or a double component.*
2. *For $i = 1, \ldots, k-1$,*
   - *if x'_i is a single-stranded component, then x'_{i+1} is a double component, and*
   - *if x'_i is a double component, then x'_{i+1} is a single-stranded component.*

When we observe that by definition the first and the last component of a formal DNA molecule cannot be nick letters, we also find

Corollary 3.9 *Let X be a formal DNA molecule which does not contain any single-stranded component, and let $x'_1 \dots x'_k$ for some $k \geq 1$ be the decomposition of X.*

1. *For $i = 1, \dots, k$, x'_i is either a double component, or an upper nick letter, or a lower nick letter.*
2. *$k = 2m - 1$ for some $m \geq 1$ (hence, k is odd) and*

$$X = \binom{\alpha_1}{c(\alpha_1)} y_1 \binom{\alpha_2}{c(\alpha_2)} y_2 \dots y_{m-1} \binom{\alpha_m}{c(\alpha_m)}$$

for $\mathcal{N}$-words $\alpha_1, \dots, \alpha_m$ and nick letters $y_1, \dots, y_{m-1}$.

3.4 Properties, relations and functions of formal DNA molecules

In this section, we introduce some properties of formal DNA molecules, relations between formal DNA molecules and functions on formal DNA molecules. We need these, to be able to define the syntax and semantics of DNA expressions in Chapter 4.

Properties

Let $X = x_1 \dots x_r$ be a formal DNA molecule, with $x_i \in \mathcal{A}_{\bigtriangledown\bigtriangleup}$ for $i = 1, \dots, r$. Then the upper strand of X is said to *cover* the lower strand *to the right* if $R(X) = x_r \notin \mathcal{A}_-$, hence, if $x_r^+ \neq -$; note that, since x_r is not allowed to be a nick letter (Condition 2 of Definition 3.2), x_r^+ is well defined. Intuitively, in this case, the upper strand extends at least as far to the right as the lower strand.

If $R(X) = x_r \in \mathcal{A}_+$, hence $x_r^- = -$ (the upper strand extends even *beyond* the lower strand to the right), then the upper strand *strictly* covers the lower strand to the right. In an analogous way we can define '(strict) covering *to the left*'.

Of course, the definition of '(strict) covering' can also be formulated for the lower strand. For example, in the formal DNA molecule X_3 from Example 3.1, the lower strand strictly covers the upper strand to the right. Here, the strands extend equally far to the left, and so we say that the upper strand covers the lower strand to the left and vice versa.

Relations

We say that a formal DNA molecule X_1 *prefits* a formal DNA molecule X_2 *by upper strands*, denoted by $X_1 \overline{\sqsubset} X_2$, if the upper strand of X_1 covers the lower strand to the right and the upper strand of X_2 covers the lower strand to the left, hence, if $R(X_1) \notin \mathcal{A}_-$ and $L(X_2) \notin \mathcal{A}_-$. Intuitively, when we write X_1 and X_2 after each other in such a case, the respective upper strands 'make contact'.

Analogously, we define X_1 to *prefit* X_2 *by lower strands* if $R(X_1) \notin \mathcal{A}_+$ and $L(X_2) \notin \mathcal{A}_+$, and write then $X_1 \underline{\sqsubset} X_2$. If either $X_1 \overline{\sqsubset} X_2$ or $X_1 \underline{\sqsubset} X_2$, then we may also say that X_1 *prefits* X_2, and write $X_1 \sqsubset X_2$.

If the order of the formal DNA molecules is clear, then we may also say that X_1 and X_2 *fit together* (by upper/lower strands).

In fact, we used the notion of prefitting already in the definition of a (single) formal DNA molecule X. When we demanded that an element of $\mathcal{A}_+$ cannot be succeeded or

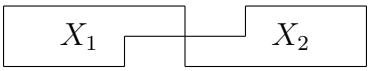


Figure 3.4: Schematic representation of two formal DNA molecules X_1 and X_2 such that the concatenation X_1X_2 is not a formal DNA molecule.

preceded by an element of $\mathcal{A}_-$ (Condition 1 in Definition 3.2), we actually demanded that the formal DNA molecule $\binom{x_i^+}{x_i^-}$ (of length 1) should prefit the formal DNA molecule $\binom{x_{i+1}^+}{x_{i+1}^-}$ (for each i such that neither x_i nor x_{i+1} is a nick letter).

Unlike the set of all $\mathcal{N}$-words $\mathcal{N}^+$, the set of formal DNA molecules $\mathcal{F}$ is not closed under concatenation. Let, for example, X_1 and X_2 be formal DNA molecules, such that the upper strand of X_1 strictly covers the lower strand to the right and the lower strand of X_2 strictly covers the upper strand to the left. Then the concatenation X_1X_2 is not a formal DNA molecule, because Condition 1 of Definition 3.2 is violated for $i = |X_1|$. This is illustrated in Figure 3.4. Thus in particular, even if $X_1 = \binom{\mathrm{A}}{\mathrm{T}}\binom{\mathrm{C}}{\mathrm{G}}\binom{\mathrm{A}}{\mathrm{T}}\binom{\mathrm{T}}{-}\binom{\mathrm{G}}{-}$ and $X_2 = \binom{-}{\mathrm{A}}\binom{-}{\mathrm{C}}\binom{\mathrm{C}}{\mathrm{G}}\binom{\mathrm{A}}{\mathrm{T}}\binom{\mathrm{T}}{\mathrm{A}}$ (so that the respective sticky ends of the DNA molecules form a perfect match), then X_1X_2 is not a formal DNA molecule. As a matter of fact, the following property holds:

Lemma 3.10 *The concatenation of two formal DNA molecules X_1 and X_2 is again a formal DNA molecule, if and only if $X_1 \sqsubset X_2$.*

Functions

We define four endomorphisms on the set $\mathcal{A}_{\triangledown\triangle}^*$: ν^+, ν^-, ν and κ. Let $x \in \mathcal{A}_{\triangledown\triangle}$. Then the functions are defined by

$$\nu^+(x) = \begin{cases} x & \text{if } x \in \mathcal{A} \cup \{\triangle\} \\ \lambda & \text{if } x = \triangledown \end{cases} \tag{3.1}$$

$$\nu^-(x) = \begin{cases} x & \text{if } x \in \mathcal{A} \cup \{\triangledown\} \\ \lambda & \text{if } x = \triangle \end{cases} \tag{3.2}$$

$$\nu(x) = \begin{cases} x & \text{if } x \in \mathcal{A} \\ \lambda & \text{if } x \in \{\triangledown, \triangle\} \end{cases} \tag{3.3}$$

$$\kappa(x) = \begin{cases} x & \text{if } x \in \mathcal{A}_\pm \cup \{\triangledown, \triangle\} \\ \binom{a}{c(a)} & \text{if } x = \binom{a}{-} \text{ for } a \in \mathcal{N} \\ \binom{c(a)}{a} & \text{if } x = \binom{-}{a} \text{ for } a \in \mathcal{N} \end{cases} \tag{3.4}$$

Thus, ν^+ removes all upper nick letters from its argument, ν^- removes all lower nick letters from its argument, ν removes both the upper nick letters and the lower nick letters from its argument, and κ replaces every symbol from $\mathcal{A}_+$ and $\mathcal{A}_-$ in its argument by the corresponding symbol from $\mathcal{A}_\pm$.

From the point of view of the molecules represented, ν^+ replaces all nicks in the upper strand of its argument by phosphodiester bonds, and ν^- does the same for nicks in the

lower strand of its argument. The function ν replaces all nicks in both the upper strand and the lower strand by phosphodiester bonds. Finally, κ provides a complementary nucleotide for every nucleotide in its argument which is not complemented yet. The function does not introduce nicks, i.e., the nucleotides added get connected to their respective neighbours. On the other hand, the nicks present in the argument are not removed by κ.

It is easy to see (by inspecting the effect of the functions on the symbols from $\mathcal{A}_{\bigtriangledown\bigtriangleup}$), that the composition of functions from the set $\{\nu^+, \nu^-, \nu, \kappa\}$ is *commutative*, i.e.,

$$h_2(h_1(X)) = h_1(h_2(X)) \text{ for all } h_1, h_2 \in \{\nu^+, \nu^-, \nu, \kappa\} \text{ and } X \in \mathcal{A}^*_{\bigtriangledown\bigtriangleup}. \tag{3.5}$$

For example, $\kappa(\nu^+(X)) = \nu^+(\kappa(X))$ for each $X \in \mathcal{A}^*_{\bigtriangledown\bigtriangleup}$.

Further, the functions are idempotent. That is, applying the same function more than one time, does not change the result:

$$h(h(X)) = h(X) \text{ for each } h \in \{\nu^+, \nu^-, \nu, \kappa\} \text{ and } X \in \mathcal{A}^*_{\bigtriangledown\bigtriangleup}. \tag{3.6}$$

For example, $\nu(\nu(X)) = \nu(X)$ for each $X \in \mathcal{A}^*_{\bigtriangledown\bigtriangleup}$.

Finally, one can verify that

$$\nu^-(\nu^+(X)) = \nu(X) \text{ for each } X \in \mathcal{A}^*_{\bigtriangledown\bigtriangleup}. \tag{3.7}$$

Hence, ν is equal to the composition of ν^+ and ν^- (and, by commutativity, ν is equal to the composition of ν^- and ν^+).

Because $\mathcal{F}$, the set of formal DNA molecules, is a subset of $\mathcal{A}^*_{\bigtriangledown\bigtriangleup}$, ν^+, ν^-, ν and κ can be applied to $\mathcal{F}$. It is easy to verify that for each $X \in \mathcal{F}$ and $h \in \{\nu^+, \nu^-, \nu, \kappa\}$, also $h(X) \in \mathcal{F}$. For example, because of Condition 3 of Definition 3.2, every nick letter in X is both preceded and succeeded by an element of $\mathcal{A}_\pm$. When such a nick letter is removed from X, by either ν^+, ν^- or ν, these elements of $\mathcal{A}_\pm$ become adjacent and this does not violate any condition of Definition 3.2.

Since we are really interested in $\mathcal{F}$, we will consider the restriction of the functions ν^+, ν^-, ν and κ to this subdomain. In order not to burden our notation too much, we will still use the notation ν^+, ν^-, ν and κ, respectively for these restricted functions, instead of $\nu^+|_{\mathcal{F}}$, etc. – this should, however, not lead to confusion.

For the composition of functions from $\{\nu^+, \nu^-, \nu, \kappa\}$ with the functions L and R we have the following results (they follow directly from the definitions of L, R, ν^+, ν^-, ν and κ and the definition of a formal DNA molecule):

Lemma 3.11 *For each $X \in \mathcal{F}$,*

$$L(\nu^+(X)) = L(\nu^-(X)) = L(\nu(X)) = L(X),$$

$$R(\nu^+(X)) = R(\nu^-(X)) = R(\nu(X)) = R(X),$$

$$L(\kappa(X)), R(\kappa(X)) \in \mathcal{A}_\pm.$$

Chapter 4

DNA Expressions

The formal DNA molecules constitute the basis of our DNA language. They allow us to define the actual elements of the language: the DNA expressions. DNA expressions are strings that denote (formal) DNA molecules, in a similar way that arithmetic expressions denote numbers. They are the central concept of this thesis, and are introduced in this chapter.

After defining the DNA expressions, we examine how one can reconstruct their structure, i.e., how they are built up, from their appearance as flat strings. We subsequently explain how to decide whether or not a given string is a DNA expression. We show that the set of all DNA expressions is a context-free language, by means of a proper context-free grammar. DNA expressions may be represented by their derivation trees in this grammar, but these trees are very large. Therefore, we define another, more concise tree representation: the structure tree of a DNA expression. Finally, we introduce several notions of equivalence, for DNA expressions that denote (almost) the same formal DNA molecule.

4.1 Operators and DNA expressions

The basic building blocks of DNA expressions are $\mathcal{N}$-words. DNA expressions result by applying operators to $\mathcal{N}$-words. The operators we consider in this thesis are $\uparrow$, $\downarrow$ and $\updownarrow$, to be pronounced as *uparrow*, *downarrow* and *updownarrow*, respectively. DNA expressions also contain opening and closing brackets: $\langle$ and $\rangle$, which delimit the scope of the operators – each (occurrence of an) operator acts only on the part of the expression that is contained between its opening and closing brackets. Hence, the set of all DNA expressions, denoted by $\mathcal{D}$, is a language over the alphabet $\Sigma_{\mathcal{D}}$, where $\Sigma_{\mathcal{D}} = \mathcal{N} \cup \{\uparrow, \downarrow, \updownarrow, \langle\ ,\ \rangle\} = \{\mathrm{A}, \mathrm{C}, \mathrm{G}, \mathrm{T}, \uparrow\ , \downarrow, \updownarrow, \langle\ ,\ \rangle\}$.

We will use the symbol E (possibly with annotations like subscripts) to denote a DNA expression. If a string can be either an $\mathcal{N}$-word or a DNA expression, then we use ε (possibly with annotations like subscripts) to denote it.

Informally, a DNA expression is a string of the form $\langle\uparrow \varepsilon_1\varepsilon_2 \ldots \varepsilon_n\rangle$, $\langle\downarrow \varepsilon_1\varepsilon_2 \ldots \varepsilon_n\rangle$ or $\langle\updownarrow \varepsilon_1\rangle$, where $n \geq 1$ and the ε_i's are either $\mathcal{N}$-words or DNA expressions themselves. The ε_i's are called the *arguments* of the operator involved. We say that an operator is *applied* to its arguments. The arguments of the operators $\uparrow$ and $\downarrow$ must satisfy certain conditions, which will be explained shortly.

Clearly, not every string over $\Sigma_{\mathcal{D}}$ is a DNA expression. In particular, every DNA expression contains brackets and at least one operator, which implies that $\mathcal{N}$-words are not DNA expressions.

If E is a DNA expression, then the *semantics* of E, denoted by $\mathcal{S}(E)$, is the formal DNA molecule represented by E. For every DNA expression, there is exactly one such formal DNA molecule, so $\mathcal{S}$ is a mapping from our language of DNA expressions $\mathcal{D}$ into $\mathcal{F}$. When we precisely define the DNA expressions, we will also describe the corresponding semantics. We do not define DNA expressions and their semantics separately, because there are restrictions on the DNA expressions we can construct (the syntax) that are explained best in terms of the molecules denoted (the semantics).

In fact, it is possible to rephrase the semantic restrictions in syntactic terms. That would, however, make the definition far more tedious. In Section 4.3, we discuss how to check whether or not a given string over $\Sigma_{\mathcal{D}}$ is a DNA expression. We will see then, that in order to verify the semantic restrictions, we do not have to compute the complete semantics of (parts of) the DNA expression. In Section 4.5, we give a context-free grammar generating the language of all DNA expressions. This may be considered as a purely syntactic definition of the DNA expressions. The official definition, however, will make use of semantic terms, because that makes the definition easier to understand.

Properties of formal DNA molecules carry over in a natural way to DNA expressions by the following convention:

> property P holds for a DNA expression E_1 (DNA expressions E_1 and E_2)
> $\Longleftrightarrow$
> property P holds for $\mathcal{S}(E_1)$ ($\mathcal{S}(E_1)$ and $\mathcal{S}(E_2)$, respectively).

Thus, e.g., we may say that the upper strand of DNA expression E_1 strictly covers the lower strand to the right, or that DNA expression E_1 prefits DNA expression E_2 by upper strands.

Before we present the formal definition of a DNA expression, we want to provide some intuition for the action of the three operators and for the restrictions that are imposed onto their arguments.

The most elementary expressions in our DNA language are the applications of the operators to a (single) $\mathcal{N}$-word α: $\langle\uparrow \alpha\rangle$, $\langle\downarrow \alpha\rangle$ and $\langle\updownarrow \alpha\rangle$. The expression $\langle\uparrow \alpha\rangle$ denotes the upper $\mathcal{A}$-word $\binom{\alpha}{-}$ (which, in turn, denotes the strand 5′-α-3′), $\langle\downarrow \alpha\rangle$ denotes the lower $\mathcal{A}$-word $\binom{-}{\alpha}$ (the strand 3′-α-5′), and $\langle\updownarrow \alpha\rangle$ denotes the double $\mathcal{A}$-word $\binom{\alpha}{c(\alpha)}$ with upper strand α (the double-stranded DNA molecule $\begin{smallmatrix}5'\text{-}\ \alpha\ \text{-}3'\\ 3'\text{-}c(\alpha)\text{-}5'\end{smallmatrix}$ without nicks).

For example, if $\alpha = \mathrm{ACATG}$, then $\langle\uparrow \alpha\rangle$ denotes $\binom{\mathrm{ACATG}}{-}$, $\langle\downarrow \alpha\rangle$ denotes $\binom{-}{\mathrm{ACATG}}$ and $\langle\updownarrow \alpha\rangle$ denotes $\binom{\mathrm{ACATG}}{\mathrm{TGTAC}}$.

In the basic DNA expressions, the three operators have one argument, an $\mathcal{N}$-word α. In general, however, the operators $\uparrow$ and $\downarrow$ may have more than one argument. Moreover, the arguments of an operator do not have to be $\mathcal{N}$-words; they may also be DNA expressions. Then, starting from the simple, basic DNA expressions, one can build more and more complex DNA expressions. There are, however, some restrictions on the arguments, which we will describe now for each of the operators.

The operator $\uparrow$ can have an arbitrary number $n \geq 1$ of arguments. Each argument ε_i ($i = 1, 2, \ldots, n$) must be either an $\mathcal{N}$-word α, or a DNA expression E. The resulting DNA expression is $\langle\uparrow \varepsilon_1\varepsilon_2 \ldots \varepsilon_n\rangle$.

From the molecular point of view, the effect of the operator $\uparrow$ is threefold: (1) it produces upper strands corresponding to arguments that are $\mathcal{N}$-words α (as in the basic DNA

$$\mathcal{S}\left(\left\langle\uparrow \begin{matrix}\mathrm{C}\\ \mathrm{G}\end{matrix} \quad \mathrm{AT} \quad \begin{matrix}\mathrm{GC}\\ \mathrm{CG}\end{matrix}\right\rangle\right) = \begin{matrix}\mathrm{CATGC}\\ \mathrm{G}\quad\ \mathrm{CG}\end{matrix} \qquad \mathcal{S}\left(\left\langle\uparrow \begin{matrix}\mathrm{A}\\ \mathrm{T}\end{matrix} \quad \begin{matrix}\mathrm{T}\\ \mathrm{A}\end{matrix}\right\rangle\right) = \begin{matrix}\mathrm{AT}\\ \mathrm{TA}\end{matrix} \qquad \text{(a)}$$

$$\mathcal{S}\left(\left\langle\downarrow \mathrm{T} \quad \begin{matrix}\mathrm{CATGC}\\ \mathrm{G}\quad\ \mathrm{CG}\end{matrix} \quad \begin{matrix}\mathrm{AT}\\ \mathrm{TA}\end{matrix}\right\rangle\right) = \begin{matrix}\mathrm{CATGCAT}\\ \mathrm{TG}\quad\ \mathrm{CGTA}\end{matrix} \qquad \text{(b)}$$

$$\mathcal{S}\left(\left\langle\updownarrow \begin{matrix}\mathrm{CATGCAT}\\ \mathrm{TG}\quad\ \mathrm{CGTA}\end{matrix}\right\rangle\right) = \begin{matrix}\mathrm{ACATGCAT}\\ \mathrm{TGTACGTA}\end{matrix} \qquad \text{(c)}$$

Figure 4.1: Examples of the effects of the three operators.[2] (a) The effect of the operator $\uparrow$. (b) The effect of the operator $\downarrow$. (c) The effect of the operator $\updownarrow$.

expression $\langle\uparrow \alpha\rangle$), (2) it repairs all nicks occurring in the upper strands of its arguments by establishing the missing phosphodiester bonds and (3) it fixes such connections between the upper strands of consecutive arguments. In short, $\uparrow$ connects all pairs of adjacent nucleotides in the upper strands of its arguments.

The third type of effect imposes a (semantic) restriction on the arguments of $\uparrow$: consecutive arguments must prefit each other by upper strands. Otherwise, there would be a gap in the upper strand 'between' two arguments, and we would not be able to connect the upper strands. Since we have defined 'prefitting each other by upper strands' only for formal DNA molecules and for DNA expressions, we consider an $\mathcal{N}$-word α here as the DNA expression $\langle\uparrow \alpha\rangle$, which represents the upper $\mathcal{A}$-word $\binom{\alpha}{-}$.

The three types of effect of $\uparrow$ are illustrated by the first example in Figure 4.1(a).

Nicks that are present in the lower strands of the arguments are not repaired by the operator $\uparrow$. As a matter of fact, $\uparrow$ introduces nicks between the lower strands of consecutive arguments if these consecutive arguments also happen to prefit each other by lower strands, i.e., if they have a blunt edge at each other's side. The second example in Figure 4.1(a) shows such a situation.

The operator $\downarrow$ is the dual of $\uparrow$. It can have an arbitrary number $n \geq 1$ of arguments, with each argument ε_i $(i = 1, \ldots, n)$ being either an $\mathcal{N}$-word or a DNA expression. The resulting DNA expression is $\langle\downarrow \varepsilon_1\varepsilon_2 \ldots \varepsilon_n\rangle$.

The effect of this operator is similar to that of $\uparrow$; the only difference is that the roles of the upper strands and the lower strands of the arguments are changed. Consequently, also the requirement on consecutive arguments is changed: for $i = 1, 2, \ldots, n-1$, ε_i must prefit ε_{i+1} by lower strands. Here, when an argument ε_i is an $\mathcal{N}$-word α, it is interpreted as the DNA expression $\langle\downarrow \alpha\rangle$, which denotes the lower $\mathcal{A}$-word $\binom{-}{\alpha}$. The effect of $\downarrow$ is illustrated by Figure 4.1(b).

Unlike the other two operators, $\updownarrow$ can have only one argument ε_1. It is either an $\mathcal{N}$-word or an (arbitrary) DNA expression. The resulting DNA expression is $\langle\updownarrow \varepsilon_1\rangle$.

If ε_1 is a DNA expression E, then, intuitively, in the DNA molecule denoted by E, the operator $\updownarrow$ provides a complementary nucleotide for every nucleotide which is not yet complemented. So it fills up every gap in the DNA molecule. Further, the operator establishes phosphodiester bonds between the nucleotides added and their respective neighbours in the strand. Hence, it does not introduce new nicks. On the other hand, if the DNA molecule denoted by E has nicks already, then these nicks are not repaired by $\updownarrow$. The

[2]The reader should not be diverted by the informal presentation of the examples. Formally, the arguments of our operators are $\mathcal{N}$-words and/or DNA expressions, and not DNA molecules. And formally, the semantics of a DNA expression is not a DNA molecule, but a *formal* DNA molecule.

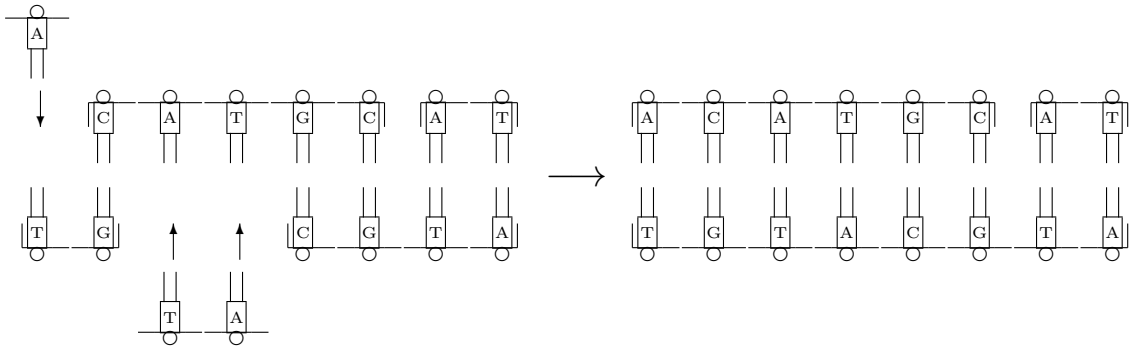


Figure 4.2: Pictorial representation of the effect of the operator $\updownarrow$.

effect of this operator is illustrated in Figure 4.1(c).

The basic DNA expression $\langle \updownarrow \alpha \rangle$ was the result of applying $\updownarrow$ to an $\mathcal{N}$-word α. This result can also be explained in terms of complements, as follows: if the argument of $\updownarrow$ is an $\mathcal{N}$-word α, the operator conceives it as the DNA expression $\langle \uparrow \alpha \rangle$ and then performs the same action as for 'ordinary' DNA expressions.

The notation $\updownarrow$ may be a bit misleading. It may suggest to be a combination of the operators $\uparrow$ and $\downarrow$. It would, e.g., repair nicks in both upper strands and lower strands then, like the function ν does with formal DNA molecules. In fact, an operator with such effect might be more realistic than the separate operators $\uparrow$ and $\downarrow$ that we have, as this effect comes closer to the effect of the enzyme ligase than the separate effects of $\uparrow$ and $\downarrow$.

Indeed, we could have chosen to use other, completely different operators to construct DNA expressions. Our choice for the three operators $\uparrow$, $\downarrow$ and $\updownarrow$ was based on two considerations: (1) the basic two components of a double-stranded DNA molecule are the two strands, and (2) the operators we consider should obey some notion of locality.

In the case of the operators $\uparrow$ and $\downarrow$, 'locality' means that they act on one of the strands – in particular, $\uparrow$ seals (repairs) the nicks only in the upper strand, while $\downarrow$ seals the nicks only in the lower strand. Note that applying both $\uparrow$ and $\downarrow$ (in any order) to one argument will seal any existing nick. In the case of the operator $\updownarrow$, 'locality' means that the string of nucleotides filling in a gap gets also properly connected (bonded) to its neighbours, while the pre-existing nicks are not sealed.

Therefore, in this thesis, we will build a theory with the operators $\uparrow$, $\downarrow$ and $\updownarrow$ as we have introduced them.

There is a nice pictorial interpretation of the operators' effects. We can consider a nucleotide as a puppet, the phosphate group at the $5'$-site and the hydroxyl group at the $3'$-site being its arms. When there is a horizontal connection between two adjacent nucleotides, we can view that as if both puppets raised one arm and joined hands. A phosphate group or a hydroxyl group that is not used for a phosphodiester bond corresponds to an arm hanging down. So in case of a nick, the two nucleotides involved keep the arm on the other one's side down.

Now when the operator $\uparrow$ is applied, the puppets in the upper strand raise their arms and, if there is an adjacent puppet, they connect. The effect of $\downarrow$ can be viewed similarly. Finally, when $\updownarrow$ complements a nucleotide, it inserts a puppet with both arms raised. Either of these arms seizes the arm of a neighbour and makes a connection. This case is depicted in Figure 4.2.

We are ready now to give a formal definition of DNA expressions and their semantics.

Definition 4.1 *A DNA expression is a string in any of the following forms:*

- $\langle\uparrow \varepsilon_1\varepsilon_2\ldots\varepsilon_n\rangle$,
  where $n \geq 1$, for $i = 1, 2, \ldots, n$, ε_i is either an $\mathcal{N}$-word or a DNA expression, and for $i = 1, 2, \ldots, n-1$, $\mathcal{S}^+(\varepsilon_i) \overline{\sqsubset} \mathcal{S}^+(\varepsilon_{i+1})$, where the function $\mathcal{S}^+$ is defined by

$$\mathcal{S}^+(\varepsilon) = \begin{cases} \binom{\alpha}{-} & \text{if } \varepsilon \text{ is an } \mathcal{N}\text{-word } \alpha \\ \mathcal{S}(\varepsilon) & \text{if } \varepsilon \text{ is a DNA expression} \end{cases} . \tag{4.1}$$

  Further,

$$\mathcal{S}(\langle\uparrow \varepsilon_1\varepsilon_2\ldots\varepsilon_n\rangle) = \nu^+(\mathcal{S}^+(\varepsilon_1))y_1\nu^+(\mathcal{S}^+(\varepsilon_2))y_2\ldots y_{n-1}\nu^+(\mathcal{S}^+(\varepsilon_n)) \tag{4.2}$$

  with

$$y_i = \begin{cases} \triangle & \text{if } \mathcal{S}^+(\varepsilon_i) \sqsubseteq \mathcal{S}^+(\varepsilon_{i+1}), \text{ i.e., if both } R(\mathcal{S}^+(\varepsilon_i)) \in \mathcal{A}_\pm \\ & \text{and } L(\mathcal{S}^+(\varepsilon_{i+1})) \in \mathcal{A}_\pm \\ \lambda & \text{otherwise, i.e., if either } R(\mathcal{S}^+(\varepsilon_i)) \in \mathcal{A}_+ \\ & \text{or } L(\mathcal{S}^+(\varepsilon_{i+1})) \in \mathcal{A}_+ \text{ (or both)} \end{cases} \tag{4.3}$$

$$(i = 1, 2, \ldots, n-1).$$

- $\langle\downarrow \varepsilon_1\varepsilon_2\ldots\varepsilon_n\rangle$,
  where $n \geq 1$, for $i = 1, 2, \ldots, n$, ε_i is either an $\mathcal{N}$-word or a DNA expression, and for $i = 1, 2, \ldots, n-1$, $\mathcal{S}^-(\varepsilon_i) \sqsubseteq \mathcal{S}^-(\varepsilon_{i+1})$, where the function $\mathcal{S}^-$ is defined by

$$\mathcal{S}^-(\varepsilon) = \begin{cases} \binom{-}{\alpha} & \text{if } \varepsilon \text{ is an } \mathcal{N}\text{-word } \alpha \\ \mathcal{S}(\varepsilon) & \text{if } \varepsilon \text{ is a DNA expression} \end{cases} . \tag{4.4}$$

  Further,

$$\mathcal{S}(\langle\downarrow \varepsilon_1\varepsilon_2\ldots\varepsilon_n\rangle) = \nu^-(\mathcal{S}^-(\varepsilon_1))y_1\nu^-(\mathcal{S}^-(\varepsilon_2))y_2\ldots y_{n-1}\nu^-(\mathcal{S}^-(\varepsilon_n))$$

  with

$$y_i = \begin{cases} \triangledown & \text{if } \mathcal{S}^-(\varepsilon_i) \overline{\sqsubset} \mathcal{S}^-(\varepsilon_{i+1}), \text{ i.e., if both } R(\mathcal{S}^-(\varepsilon_i)) \in \mathcal{A}_\pm \\ & \text{and } L(\mathcal{S}^-(\varepsilon_{i+1})) \in \mathcal{A}_\pm \\ \lambda & \text{otherwise, i.e., if either } R(\mathcal{S}^-(\varepsilon_i)) \in \mathcal{A}_- \\ & \text{or } L(\mathcal{S}^-(\varepsilon_{i+1})) \in \mathcal{A}_- \text{ (or both)} \end{cases}$$

$$(i = 1, 2, \ldots, n-1).$$

- $\langle\updownarrow \varepsilon_1\rangle$,
  where ε_1 is either an $\mathcal{N}$-word or a DNA expression.

  Further,

$$\mathcal{S}(\langle\updownarrow \varepsilon_1\rangle) = \kappa(\mathcal{S}^+(\varepsilon_1)).$$

  for the function $\mathcal{S}^+$ defined above.

One can verify that indeed, for each DNA expression E satisfying this definition, $\mathcal{S}(E)$ is a formal DNA molecule. Now, the formal language $\mathcal{D}$ is the set of all DNA expressions.

Example 4.2 The DNA expression

$$E = \langle\downarrow \mathrm{T}\,\langle\uparrow\,\langle\updownarrow \mathrm{C}\rangle\,\mathrm{AT}\,\langle\downarrow\,\langle\updownarrow \mathrm{G}\rangle\,\langle\updownarrow \mathrm{C}\rangle\rangle\rangle\,\langle\uparrow\,\langle\updownarrow \mathrm{A}\rangle\,\langle\updownarrow \mathrm{T}\rangle\rangle\rangle\,,$$

uses all three operators. It is easily verified that E denotes the DNA molecule from Figure 4.1(b). ■

We call a DNA expression of the form $\langle\uparrow \varepsilon_1 \ldots \varepsilon_n\rangle$ an $\uparrow$-*expression*, one of the form $\langle\downarrow \varepsilon_1 \ldots \varepsilon_n\rangle$ a $\downarrow$-*expression*, and one of the form $\langle\updownarrow \varepsilon_1\rangle$ an $\updownarrow$-*expression*. Hence, the DNA expression in Example 4.2 is a $\downarrow$-expression.

In this thesis, we will often introduce a general $\uparrow$-expression as '$\langle\uparrow \varepsilon_1 \ldots \varepsilon_n\rangle$ for some $n \geq 1$ and $\mathcal{N}$-words and DNA expressions $\varepsilon_1, \ldots, \varepsilon_n$'. Here, the phrase '$\mathcal{N}$-words and DNA expressions $\varepsilon_1, \ldots, \varepsilon_n$' does *not* necessarily mean that there is at least one argument ε_i that is an $\mathcal{N}$-word and at least one argument ε_i that is a DNA expression. It is just an easy way to express that for $i = 1, \ldots, n$, ε_i is either an $\mathcal{N}$-word or a DNA expression. It is in principle possible that each ε_i is an $\mathcal{N}$-word or that each ε_i is a DNA expression. Of course, we use this type of formulation also for $\downarrow$-expressions.

The formal DNA molecule $\mathcal{S}^+(\varepsilon)$, occurring in the definition of a DNA expression of the form $\langle\uparrow \varepsilon_1\varepsilon_2 \ldots \varepsilon_n\rangle$, can be considered as a kind of 'upper semantics' of the argument ε. Similarly, the formal DNA molecule $\mathcal{S}^-(\varepsilon)$, occurring in the definition of a DNA expression of the form $\langle\downarrow \varepsilon_1\varepsilon_2 \ldots \varepsilon_n\rangle$, can be considered as a kind of 'lower semantics' of the argument ε.

When we define functions Exp^+ and Exp^- by

$$\mathrm{Exp}^+(\varepsilon) = \begin{cases} \langle\uparrow \alpha\rangle & \text{if } \varepsilon \text{ is an } \mathcal{N}\text{-word } \alpha \\ \varepsilon & \text{if } \varepsilon \text{ is a DNA expression} \end{cases} \tag{4.5}$$

and

$$\mathrm{Exp}^-(\varepsilon) = \begin{cases} \langle\downarrow \alpha\rangle & \text{if } \varepsilon \text{ is an } \mathcal{N}\text{-word } \alpha \\ \varepsilon & \text{if } \varepsilon \text{ is a DNA expression,} \end{cases} \tag{4.6}$$

it is easy to see that for every $\mathcal{N}$-word or DNA expression ε, $\mathcal{S}^+(\varepsilon) = \mathcal{S}(\mathrm{Exp}^+(\varepsilon))$ and $\mathcal{S}^-(\varepsilon) = \mathcal{S}(\mathrm{Exp}^-(\varepsilon))$. Consequently, for $\mathcal{N}$-words or DNA expressions ε_1 and ε_2, we have $\mathcal{S}^+(\varepsilon_1) \overline{\sqsubset} \mathcal{S}^+(\varepsilon_2)$, if and only if $\mathrm{Exp}^+(\varepsilon_1) \overline{\sqsubset} \mathrm{Exp}^+(\varepsilon_2)$, where the relation $\overline{\sqsubset}$ is used in the context of formal DNA molecules first, and in the context of DNA expressions next. Analogously, $\mathcal{S}^-(\varepsilon_1) \sqsubseteq \mathcal{S}^-(\varepsilon_2)$, if and only if $\mathrm{Exp}^-(\varepsilon_1) \sqsubseteq \mathrm{Exp}^-(\varepsilon_2)$. The DNA expressions $\mathrm{Exp}^+(\varepsilon)$ and $\mathrm{Exp}^-(\varepsilon)$ can be considered as a kind of 'upper DNA expression' and 'lower DNA expression' corresponding to the argument ε, respectively.

Note that, indeed, the operator $\updownarrow$ does not introduce new nicks in its argument, simply because the function κ does not do so.

We need to mention that the interpretation of the arguments of $\uparrow$-expressions and $\downarrow$-expressions may be ambiguous. For example, consider DNA expression E from Example 4.2. Unless we have additional information, we cannot tell whether the $\mathcal{N}$-word AT is itself an argument of the first occurrence of $\uparrow$, or that it is the concatenation of two arguments A and T. Consequently, we cannot tell, either, how many arguments this occurrence of $\uparrow$ has. This ambiguity occurs whenever an operator $\uparrow$ or $\downarrow$ has consecutive arguments that are $\mathcal{N}$-words, or has an argument that is an $\mathcal{N}$-word α with $|\alpha| \geq 2$.

Fortunately, even though it may be unclear what exactly the arguments of operators $\uparrow$ and $\downarrow$ occurring in a DNA expression are, there can be no doubt about the (formal) DNA molecule denoted by the DNA expression. This is implied by the following result:

Theorem 4.3 *Let $1 \leq i_0 < j_0 \leq n$, and let $E = \langle\uparrow \varepsilon_1 \dots \varepsilon_{i_0-1}\alpha_{i_0} \dots \alpha_{j_0}\varepsilon_{j_0+1} \dots \varepsilon_n\rangle$ be a DNA expression, for some $\mathcal{N}$-words or DNA expressions $\varepsilon_1, \dots, \varepsilon_{i_0-1}, \varepsilon_{j_0+1}, \dots, \varepsilon_n$ and some $\mathcal{N}$-words $\alpha_{i_0}, \dots, \alpha_{j_0}$. Let $\alpha = \alpha_{i_0} \dots \alpha_{j_0}$. Then $\mathcal{S}(E)$ is the same, regardless of the interpretation of α as one argument or as a sequence of separate arguments $\alpha_{i_0}, \dots, \alpha_{j_0}$.*

Hence, any partitioning of an argument α of $\uparrow$ into a sequence of arguments $\alpha_{i_0}, \dots, \alpha_{j_0}$ yields the same semantics. Of course, an analogous result holds for $\downarrow$-expressions.

Proof: When we interpret α as one argument, Equation (4.2) becomes

$$\begin{aligned}\mathcal{S}(E) &= \nu^+(\mathcal{S}^+(\varepsilon_1))y_1 \dots y_{i_0-2}\nu^+(\mathcal{S}^+(\varepsilon_{i_0-1})) \cdot \binom{\alpha}{_} \cdot \\ &\quad \nu^+(\mathcal{S}^+(\varepsilon_{j_0+1}))y_{j_0+1} \dots y_{n-1}\nu^+(\mathcal{S}^+(\varepsilon_n)),\end{aligned} \tag{4.7}$$

where the y_i's are defined by (4.3). Note that $\mathcal{S}^+(\alpha) = \binom{\alpha}{_}$ and also $\nu^+(\mathcal{S}^+(\alpha)) = \binom{\alpha}{_}$. Indeed, because $L(\binom{\alpha}{_}), R(\binom{\alpha}{_}) \in \mathcal{A}_+$, both y_{i_0-1} and y_{j_0} equal λ.[3]

On the other hand, when we interpret α as a sequence of separate arguments $\alpha_{i_0}, \dots, \alpha_{j_0}$, we obtain

$$\begin{aligned}\mathcal{S}(E) &= \nu^+(\mathcal{S}^+(\varepsilon_1))y_1 \dots y_{i_0-2}\nu^+(\mathcal{S}^+(\varepsilon_{i_0-1})) \cdot \binom{\alpha_{i_0}}{_} \dots \binom{\alpha_{j_0}}{_} \cdot \\ &\quad \nu^+(\mathcal{S}^+(\varepsilon_{j_0+1}))y_{j_0+1} \dots y_{n-1}\nu^+(\mathcal{S}^+(\varepsilon_n)),\end{aligned} \tag{4.8}$$

where the y_i's are the same as in (4.7). Because $\binom{\alpha_{i_0}}{_} \dots \binom{\alpha_{j_0}}{_} = \binom{\alpha_{i_0} \dots \alpha_{j_0}}{_} = \binom{\alpha}{_}$, Equation (4.8) reduces to (4.7). □

Note that the interpretation of $\mathcal{N}$-words α of length $|\alpha| \geq 2$ as argument(s) of an operator is unambiguous for the operator $\updownarrow$, because this operator can have only one argument.

Example 4.4 Let $E = \langle\uparrow \mathrm{ACATG}\rangle$. Then there are many possible interpretations of the arguments of the operator $\uparrow$. We might, e.g., interpret E as $\langle\uparrow \alpha_1 \dots \alpha_5\rangle$, with five arguments $\alpha_1 = \mathrm{A}$, $\alpha_2 = \mathrm{C}$, $\alpha_3 = \mathrm{A}$, $\alpha_4 = \mathrm{T}$ and $\alpha_5 = \mathrm{G}$. But we might as well interpret E as $\langle\uparrow \alpha_1\alpha_2\rangle$ with two arguments $\alpha_1 = \mathrm{AC}$ and $\alpha_2 = \mathrm{ATG}$, as $\langle\uparrow \alpha_1\alpha_2\rangle$ with two arguments $\alpha_1 = \mathrm{ACAT}$ and $\alpha_2 = \mathrm{G}$, or as $\langle\uparrow \alpha_1\rangle$ with only one argument $\alpha_1 = \mathrm{ACATG}$. Whatever interpretation we choose, $\mathcal{S}(E) = \binom{\mathrm{ACATG}}{_}$. ■

By the above, we are free to interpret consecutive $\mathcal{N}$-words in a DNA expression as one $\mathcal{N}$-word. This motivates the definition of a *maximal $\mathcal{N}$-word occurrence in a string X* (e.g., in a DNA expression E) as an occurrence (X_1, X_2) of an $\mathcal{N}$-word α in X such that (1) if $X_1 \neq \lambda$ then $R(X_1) \notin \mathcal{N}$ and (2) if $X_2 \neq \lambda$ then $L(X_2) \notin \mathcal{N}$. Hence, the $\mathcal{N}$-word α 'cannot be extended either to the left or to the right'.

Example 4.5 In the DNA expression

$$\langle\downarrow \mathrm{T}\,\langle\uparrow\,\langle\updownarrow \mathrm{C}\rangle\,\mathrm{AT}\,\langle\updownarrow \mathrm{GCAT}\rangle\rangle\rangle$$

[3] If $i_0 = 1$ or $j_0 = n$, then, of course, y_{i_0-1} or y_{j_0}, respectively, does not even exist.

the first occurrence of C and the first occurrence of AT are maximal $\mathcal{N}$-word occurrences. This is, however, not the case with the second occurrences of these $\mathcal{N}$-words, as they can be extended to GCAT. ■

Although we may interpret consecutive $\mathcal{N}$-words in a DNA expression as one $\mathcal{N}$-word, we do not always do so in this thesis. In particular, we still allow occurrences of the operators $\uparrow$ and $\downarrow$ in a DNA expression to have consecutive arguments that are $\mathcal{N}$-words.

Additional terminology

We say that an operator *governs* its argument(s) and everything inside its argument(s). In every DNA expression we can identify an outermost operator. This is the operator which has been performed last. It governs the entire DNA expression.

Because of the 1–1 correspondence between a DNA expression and its outermost operator, we will sometimes use one term while meaning the other. In particular, we may speak of the *arguments of a DNA expression*, while we actually mean the arguments of the outermost operator of a DNA expression. For example, the (three) arguments of the DNA expression from Example 4.2 are the $\mathcal{N}$-word T, the $\uparrow$-expression $\langle\uparrow \langle\updownarrow \mathrm{C}\rangle \mathrm{AT} \langle\downarrow \langle\updownarrow \mathrm{G}\rangle \langle\updownarrow \mathrm{C}\rangle\rangle\rangle$ and the $\uparrow$-expression $\langle\uparrow \langle\updownarrow \mathrm{A}\rangle \langle\updownarrow \mathrm{T}\rangle\rangle$.

We call (an occurrence of) an operator in a DNA expression E which is not the outermost operator, an *inner occurrence* of this operator in E.

An operator may occur more than once in a DNA expression. To denote a specific occurrence of an operator, we may provide the operator with a subscript. For example, we may have $\uparrow_0$ or $\downarrow_1$.

A *DNA subexpression* E^s of a DNA expression E is a substring of E which is itself a DNA expression. If $E^s \neq E$, then we call E^s a *proper DNA subexpression* of E. Clearly, the outermost operator of a proper DNA subexpression of E is an inner occurrence of this operator in E.

We will use the term $\uparrow$-*subexpression* of E to refer to a DNA subexpression of E which is an $\uparrow$-expression. Analogously, we may have a $\downarrow$-*subexpression* and an $\updownarrow$-*subexpression* of E.

For every $\mathcal{N}$-word α occurring in a DNA expression E and for every proper DNA subexpression E^s of E we define its *parent operator* to be the operator which has the $\mathcal{N}$-word or DNA subexpression as an immediate argument. For example, in the DNA expression from Example 4.2, the parent operator of the $\mathcal{N}$-word AT is the first occurrence of the operator $\uparrow$ in the DNA expression; for the second occurrence of the $\mathcal{N}$-word C it is clearly the operator $\updownarrow$ standing in front of it; and the parent operator of the DNA subexpression $\langle\updownarrow \mathrm{G}\rangle$ is the second occurrence of the operator $\downarrow$.

An occurrence of an operator is an *ancestor operator* of an $\mathcal{N}$-word or a DNA subexpression ε occurring in E, if ε is contained in an argument of the operator. For example, the ancestor operators of the second occurrence of the $\mathcal{N}$-word C in the DNA expression from Example 4.2 are: the first occurrence of $\downarrow$ (the outermost operator), the first occurrence of $\uparrow$, the second occurrence of $\downarrow$ and the third occurrence of $\updownarrow$ (the parent operator of C).

If an argument of a certain (occurrence of an) operator is an $\mathcal{N}$-word, then we may call it an $\mathcal{N}$-*word-argument* of the operator. If, on the other hand, the argument is a DNA expression, then we may call it an *expression-argument* of the operator. In particular, if it is an $\uparrow$-expression, then we may call it an $\uparrow$-*argument*. In an analogous way, we define a $\downarrow$-*argument* and an $\updownarrow$-*argument* of an operator. At some point in this thesis, it will be useful

to have a single term for arguments that are not $\updownarrow$-expressions, i.e., for $\mathcal{N}$-word-arguments, $\uparrow$-arguments and $\downarrow$-arguments. We call such arguments *non-$\updownarrow$-arguments*.

Let us assume that the $\mathcal{N}$-word-arguments of a certain $\uparrow$-expression or $\downarrow$-expression E are maximal $\mathcal{N}$-word occurrences. We say that E is *alternating*, if its arguments are maximal $\mathcal{N}$-word occurrences and DNA expressions, alternately. Because by definition, a maximal $\mathcal{N}$-word occurrence cannot be preceded or succeeded by another $\mathcal{N}$-word-argument, this is equivalent to saying that E does not have consecutive expression-arguments. An occurrence of an operator $\uparrow$ or $\downarrow$ is alternating, if the corresponding DNA subexpression is alternating.

Example 4.6 Let

$$\begin{aligned} E_1 &= \langle\uparrow \alpha_1\rangle\,, \\ E_2 &= \langle\uparrow \langle\updownarrow \alpha_1\rangle\rangle\,, \\ E_3 &= \langle\downarrow \langle\uparrow \alpha_1 \langle\updownarrow \alpha_2\rangle\rangle \alpha_3\alpha_4 \langle\updownarrow \alpha_5\rangle\rangle\,, \\ E_4 &= \langle\downarrow \alpha_1 \langle\downarrow \langle\updownarrow \alpha_2\rangle \langle\uparrow \langle\updownarrow \alpha_3\rangle \alpha_4\rangle\rangle\rangle\,. \end{aligned}$$

Both E_1 and E_2 have only one argument, and are by definition alternating. The $\mathcal{N}$-word-arguments α_3 and α_4 of E_3 together form a maximal $\mathcal{N}$-word occurrence. This makes E_3 alternating. Finally, E_4 is alternating, although its second argument $\langle\downarrow \langle\updownarrow \alpha_2\rangle \langle\uparrow \langle\updownarrow \alpha_3\rangle \alpha_4\rangle\rangle$ is not alternating. The $\downarrow$-expression in Example 4.2 is not alternating, because both its second argument $\langle\uparrow \langle\updownarrow \mathrm{C}\rangle \mathrm{AT} \langle\downarrow \langle\updownarrow \mathrm{G}\rangle \langle\updownarrow \mathrm{C}\rangle\rangle\rangle$ and its third argument $\langle\uparrow \langle\updownarrow \mathrm{A}\rangle \langle\updownarrow \mathrm{T}\rangle\rangle$ are DNA expressions. ■

Let E be a DNA expression, and let $\alpha_1, \ldots, \alpha_k$ for some $k \geq 1$ be the maximal $\mathcal{N}$-word occurrences in E, in the order of their occurrence from left to right. Then we will sometimes write E as a function of these maximal $\mathcal{N}$-word occurrences, hence $E = E(\alpha_1, \ldots, \alpha_k)$. Clearly, $\alpha_1, \ldots, \alpha_k$ also show up in the corresponding formal DNA molecule $\mathcal{S}(E)$, and they occur in $\mathcal{S}(E)$ in the same order as in E.

Note, however, that different maximal $\mathcal{N}$-word occurrences α_i in E may end up in the same component of $\mathcal{S}(E)$. Moreover, if the parent operator of a maximal $\mathcal{N}$-word occurrence α_i is $\downarrow$ (which implies that a lower $\mathcal{A}$-word $\binom{-}{\alpha_i}$ is introduced into the semantics), then this lower $\mathcal{A}$-word may be complemented by an occurrence of $\updownarrow$. This would result in a double $\mathcal{A}$-word $\binom{c(\alpha_i)}{\alpha_i}$. Hence, the component of $\mathcal{S}(E)$ in which a maximal $\mathcal{N}$-word occurrence α_i of E appears, is not necessarily an element of $\left\{\binom{\alpha_i}{-}, \binom{-}{\alpha_i}, \binom{\alpha_i}{c(\alpha_i)}\right\}$. For example, if $E = E(\alpha_1, \alpha_2) = \langle\updownarrow \langle\downarrow \alpha_1 \langle\updownarrow \alpha_2\rangle\rangle\rangle$, then $\mathcal{S}(E) = \binom{c(\alpha_1)\alpha_2}{\alpha_1 c(\alpha_2)}$.

4.2 Brackets, arguments and DNA subexpressions

The brackets in a DNA expression determine a structure with different levels. An opening bracket $\langle$ corresponds to an increase of the level by 1, a closing bracket $\rangle$ to a decrease of the level by 1. The resulting levels are called the *nesting levels* of the brackets.

Initially, before the first letter of a DNA expression, the nesting level is 0. Since every opening bracket precedes the corresponding closing bracket, the nesting level is non-negative at any position in a DNA expression. Further, because the number of opening brackets equals the number of closing brackets, the nesting level is back at 0 at the end of a DNA expression. In Figure 4.3, we show the nesting level as a function of the position in

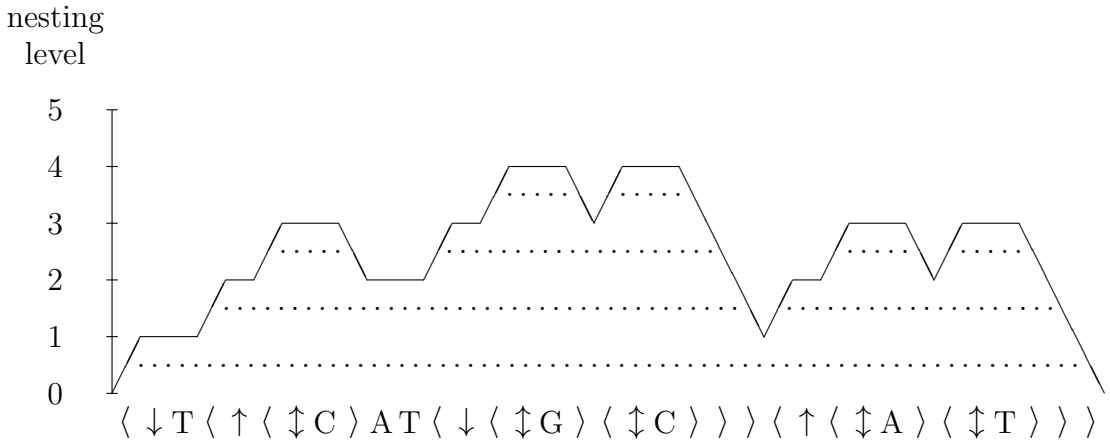


Figure 4.3: Nesting level as a function of the position in the DNA expression from Example 4.2. Horizontal dotted lines connect changes of the nesting level due to pairs of corresponding brackets.

the DNA expression from Example 4.2. The maximal nesting level of a DNA expression is of particular interest. For example, the maximal nesting level of the DNA expression from Figure 4.3 is 4.

A DNA expression consists of an opening bracket, an operator, one or more arguments and a closing bracket. Hence, the nesting level structure of a DNA expression is determined by the nesting level structure of its arguments. In particular, the maximal nesting level of a DNA expression is determined by the maximal nesting levels of those arguments that are DNA expressions themselves:

Lemma 4.7 *Let E be a DNA expression and let $E_1, \ldots, E_r$ for some $r \geq 0$ be the expression-arguments of E.*

1. *If $r = 0$ (i.e., if E only has $\mathcal{N}$-word-arguments), then the maximal nesting level of E is 1.*

2. *If $r \geq 1$, then the maximal nesting level of E is equal to*

$$\max_{j=1}^{r} \textit{(maximal nesting level of } E_j\textit{)} + 1.$$

Of course, in the expression in Claim 2, the expression-arguments E_j are viewed as independent DNA expressions, which start at level 0.

We can use the notion of the nesting level for identifying substrings of a DNA expression. We do this in the following two results.

Lemma 4.8 *Suppose that the opening bracket of a DNA subexpression E^s of a DNA expression E raises the nesting level of E from $l-1$ to l for a certain positive integer l. Then the closing bracket of E^s is the first closing bracket after this opening bracket to lower the nesting level from l to $l-1$. In particular, between the opening bracket and the closing bracket of E^s, the nesting level is at least l.*

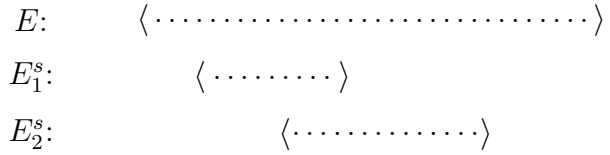


Figure 4.4: Schematic representation of two (hypothetically) overlapping DNA subexpressions E_1^s and E_2^s of a DNA expression E.

Proof: Straightforward by induction on the number of operators occurring in E^s. □

To illustrate this lemma, we have drawn dotted lines between corresponding increases and decreases of the nesting level in Figure 4.3. We can thus use the nesting levels of the brackets in a DNA expression E, to reconstruct the DNA subexpressions that occurred in the recursive definition of E. We proceed with arbitrary arguments of operators occurring in E.

Theorem 4.9 *Let E be a DNA expression, and assume that each $\mathcal{N}$-word-argument of an operator occurring in E is a maximal $\mathcal{N}$-word occurrence. Let $|_0$ be an operator at nesting level l in E. Then (an occurrence of) a substring between $|_0$ and the closing bracket of $|_0$ is an argument of $|_0$, if and only if*

- *either it is a maximal $\mathcal{N}$-word occurrence in E at nesting level l,*
- *or it starts with an opening bracket raising the nesting level from l to $l+1$ and ends with the corresponding closing bracket.*

This result is important, because it enables us to determine the structure of a DNA expression, i.e., how the DNA expression has been built up, even though it is just a sequence of symbols.

Note that by Theorem 4.3, the assumption that each $\mathcal{N}$-word-argument of an operator is in fact a maximal $\mathcal{N}$-word occurrence, is not restrictive.

Clearly, as every DNA (sub-)expression is of the form $\langle |_0 \varepsilon_1 \dots \varepsilon_n \rangle$ for an operator $|_0$ and arguments $\varepsilon_1, \dots, \varepsilon_n$, the arguments are indeed substrings between $|_0$ and the closing bracket of $|_0$. Hence, this theorem covers all arguments of $|_0$.

We do not give a proof for Theorem 4.9. First, because the result is intuitively clear anyway, and second, because the inductive arguments that are used in the proof are a bit tedious, although not extremely complicated. We only mention that both in the proof from left to right and in the proof from right to left, there is a crucial role for Lemma 4.8.

Lemma 4.8 may also be used in a formal proof of the following result. Again, however, because this result is standard in the world of bracketed expressions, we omit the proof.

Theorem 4.10 *Two (occurrences of) DNA subexpressions in a DNA expression E cannot overlap. So either one is contained in the other, or they do not have a common (occurrence of a) substring at all.*

Hence, a situation as depicted in Figure 4.4 is not possible.

Corollary 4.11 *If E^s is a proper DNA subexpression of a DNA expression E, then E^s is contained in an argument of E.*

Proof: Because E^s is a proper DNA subexpression of E, it is a substring of $\varepsilon_1 \dots \varepsilon_n$, the concatenation of the arguments of E. Let ε_i be the first argument that has a non-empty intersection with E^s. Then ε_i contains the opening bracket of E^s, which implies that ε_i is a DNA expression (and not an $\mathcal{N}$-word).

If the opening bracket of E^s is the opening bracket of ε_i, then also the closing brackets must match, so E^s is equal to ε_i. In particular, E^s is contained in ε_i. If the opening bracket of E^s is not the opening bracket of ε_i, then ε_i is clearly not contained in E^s. By Theorem 4.10, E^s must be (properly) contained in ε_i then. □

We conclude this section with a simple, but useful result. It says that arguments of DNA expressions cannot just consist of brackets and operators:

Lemma 4.12 *Let $E \in \mathcal{D}$ be a DNA expression. Every argument of every operator in E contains at least one $\mathcal{N}$-word α.*

Proof: Straightforward by induction on the number of operators occurring in an argument. □

4.3 Recognition of DNA expressions

As mentioned before, not every string over $\Sigma_{\mathcal{D}}$, i.e., consisting of $\mathcal{N}$-words α, operators and brackets, is a DNA expression. Given an arbitrary string E over this alphabet, we may want to verify whether or not it is a DNA expression. A natural way to do this, is simply to check all requirements from the (recursive) definition of a DNA expression, as given in Definition 4.1. One requirement is that the arguments of (occurrences of) operators $\uparrow$ and $\downarrow$ must fit together by upper strands or lower strands, respectively. In this section, we discuss how to check this without explicitly computing the semantics of the arguments.

Before we can examine the arguments of operators, we must look at the structure of the string E we are given. In particular, we must verify (1) that there are as many opening brackets as closing brackets in the string, (2) that each opening brackets comes before the corresponding closing bracket, (3) that the first symbol of the string is an opening bracket and the last symbol is the corresponding closing bracket, (4) that each opening bracket is immediately succeeded by an operator, and (5) that there are no other occurrences of operators in the string.

Next, by using Theorem 4.9, we can determine the arguments ε_i of the outermost operator $|_0$ of the string. If $|_0$ is $\updownarrow$, then there has to be *exactly* one argument; if it is either $\uparrow$ or $\downarrow$, then there has to be *at least* one argument. In particular, we cannot have $E = \langle\uparrow\rangle$, $E = \langle\downarrow\rangle$ or $E = \langle\updownarrow\rangle$. For those arguments that are no (maximal) $\mathcal{N}$-word occurrences, we can check recursively whether they are DNA expressions.

If, up to here, all requirements are met and $|_0$ has only one argument, then the string is a DNA expression. If the number of arguments n is greater than 1 (which implies that $|_0$ is $\uparrow$ or $\downarrow$), then we have to do some more work. We must verify the semantic restriction that the arguments $\varepsilon_1, \dots, \varepsilon_n$ fit together by upper strands or lower strands (depending on the operator), see Definition 4.1. In fact, we may have had to perform such a check already for occurrences of $\uparrow$ and $\downarrow$ *inside* the arguments, when we checked that these arguments are really DNA expressions.

The requirement for ↑-expressions can be expressed formally in terms of $R(\mathcal{S}^+(\varepsilon_i))$ and $L(\mathcal{S}^+(\varepsilon_{i+1}))$ for $i = 1, \ldots, n-1$. However, if we only want to check whether or not two arguments of an operator fit together by upper strands, then we are not interested in the complete semantics of these arguments. In fact, it could be very inefficient to compute the complete semantics for just this check.

Therefore, it would be desirable if we could compute $L(\mathcal{S}^+(\varepsilon_i))$ and $R(\mathcal{S}^+(\varepsilon_i))$ for an $\mathcal{N}$-word or DNA expression ε_i without having to compute $\mathcal{S}^+(\varepsilon_i)$ explicitly. Actually, we only need to know which of the subsets $\mathcal{A}_+$, $\mathcal{A}_-$ and $\mathcal{A}_\pm$ the $\mathcal{A}$-letters $L(\mathcal{S}^+(\varepsilon_i))$ and $R(\mathcal{S}^+(\varepsilon_i))$ belong to. For consecutive arguments ε_i and ε_{i+1}, both $R(\mathcal{S}^+(\varepsilon_i))$ and $L(\mathcal{S}^+(\varepsilon_{i+1}))$ must be in $\mathcal{A}_+ \cup \mathcal{A}_\pm$.

Of course, to check if the arguments $\varepsilon_1, \ldots, \varepsilon_n$ of an operator ↓ fit together by lower strands, we need to answer a similar question for $L(\mathcal{S}^-(\varepsilon_i))$ and $R(\mathcal{S}^-(\varepsilon_i))$. Note that if ε_i is a DNA expression E_i, then $\mathcal{S}^+(\varepsilon_i) = \mathcal{S}^-(\varepsilon_i) = \mathcal{S}(E_i)$. In that case, $L(\mathcal{S}^+(\varepsilon_i)) = L(\mathcal{S}^-(\varepsilon_i))$ and $R(\mathcal{S}^+(\varepsilon_i) = R(\mathcal{S}^-(\varepsilon_i))$.

We can use the following result to recursively determine the subsets of $\mathcal{A}$ that $L(\mathcal{S}^+(\varepsilon_i))$, $R(\mathcal{S}^+(\varepsilon_i))$, $L(\mathcal{S}^-(\varepsilon_i))$ and $R(\mathcal{S}^-(\varepsilon_i))$ belong to:

Lemma 4.13 *Let ε_i be an $\mathcal{N}$-word or a DNA expression.*

1. *If ε_i is an $\mathcal{N}$-word α, then*

$$L(\mathcal{S}^+(\varepsilon_i)), R(\mathcal{S}^+(\varepsilon_i)) \in \mathcal{A}_+,$$
$$L(\mathcal{S}^-(\varepsilon_i)), R(\mathcal{S}^-(\varepsilon_i)) \in \mathcal{A}_-.$$

2. *If ε_i is an ↕-expression, then*

$$L(\mathcal{S}^+(\varepsilon_i)) = L(\mathcal{S}^-(\varepsilon_i)) = L(\mathcal{S}(\varepsilon_i)) \in \mathcal{A}_\pm,$$
$$R(\mathcal{S}^+(\varepsilon_i)) = R(\mathcal{S}^-(\varepsilon_i)) = R(\mathcal{S}(\varepsilon_i)) \in \mathcal{A}_\pm.$$

3. *If ε_i is an ↑-expression $\langle\uparrow \varepsilon_{i,1} \ldots \varepsilon_{i,m}\rangle$ for some $m \geq 1$ and $\mathcal{N}$-words and DNA expressions $\varepsilon_{i,1}, \ldots, \varepsilon_{i,m}$, then*

$$L(\mathcal{S}^+(\varepsilon_i)) = L(\mathcal{S}^-(\varepsilon_i)) = L(\mathcal{S}(\varepsilon_i)) = L(\mathcal{S}^+(\varepsilon_{i,1})),$$
$$R(\mathcal{S}^+(\varepsilon_i)) = R(\mathcal{S}^-(\varepsilon_i)) = R(\mathcal{S}(\varepsilon_i)) = R(\mathcal{S}^+(\varepsilon_{i,m})).$$

4. *If ε_i is a ↓-expression $\langle\downarrow \varepsilon_{i,1} \ldots \varepsilon_{i,m}\rangle$ for some $m \geq 1$ and $\mathcal{N}$-words and DNA expressions $\varepsilon_{i,1}, \ldots, \varepsilon_{i,m}$, then*

$$L(\mathcal{S}^+(\varepsilon_i)) = L(\mathcal{S}^-(\varepsilon_i)) = L(\mathcal{S}(\varepsilon_i)) = L(\mathcal{S}^-(\varepsilon_{i,1})),$$
$$R(\mathcal{S}^+(\varepsilon_i)) = R(\mathcal{S}^-(\varepsilon_i)) = R(\mathcal{S}(\varepsilon_i)) = R(\mathcal{S}^-(\varepsilon_{i,m})).$$

Proof:

1. This claim follows immediately from the observation that for an $\mathcal{N}$-word α, $\mathcal{S}^+(\alpha) = \binom{\alpha}{-}$ and $\mathcal{S}^-(\alpha) = \binom{-}{\alpha}$.

2. Assume that $\varepsilon_i = \langle \updownarrow \varepsilon_{i,1} \rangle$ for an $\mathcal{N}$-word or a DNA expression $\varepsilon_{i,1}$. By the definition of the semantics of an $\updownarrow$-expression, $\mathcal{S}(\varepsilon_i) = \kappa(\mathcal{S}^+(\varepsilon_{i,1}))$. Hence, $L(\mathcal{S}(\varepsilon_i)) = L(\kappa(\mathcal{S}^+(\varepsilon_{i,1})))$ and $R(\mathcal{S}(\varepsilon_i)) = R(\kappa(\mathcal{S}^+(\varepsilon_{i,1})))$. By Lemma 3.11, these are in $\mathcal{A}_\pm$.

3. Assume that $\varepsilon_i = \langle \uparrow \varepsilon_{i,1} \dots \varepsilon_{i,m} \rangle$ for some $m \geq 1$ and $\mathcal{N}$-words and DNA expressions $\varepsilon_{i,1}, \dots, \varepsilon_{i,m}$. According to the definition of an $\uparrow$-expression and its semantics,

   $$\mathcal{S}(\varepsilon_i) = \nu^+(\mathcal{S}^+(\varepsilon_{i,1}))y_1 \dots y_{m-1}\nu^+(\mathcal{S}^+(\varepsilon_{i,m}))$$

   for y_i's as in (4.3). Consequently,

   $$L(\mathcal{S}(\varepsilon_i)) = L(\nu^+(\mathcal{S}^+(\varepsilon_{i,1}))) = L(\mathcal{S}^+(\varepsilon_{i,1})).$$

   The second equality in this derivation follows from Lemma 3.11.

   In a similar way, we find $R(\mathcal{S}(\varepsilon_i)) = R(\mathcal{S}^+(\varepsilon_{i,m}))$.

4. The proof of this claim is analogous to that of the previous claim.

□

Once we know $L(\mathcal{S}^+(\varepsilon_i))$ and $R(\mathcal{S}^+(\varepsilon_i))$ (if $|_0 = \uparrow$) or $L(\mathcal{S}^-(\varepsilon_i))$ and $R(\mathcal{S}^-(\varepsilon_i))$ (if $|_0 = \downarrow$) for $i = 1, \dots, n$, it is easy to check whether or not the arguments fit together by upper strands or lower strands, respectively. If so, then the string E is a DNA expression; otherwise, it is not.

In Figure 4.5, we give a recursive function `CheckExpression`, which uses Lemma 4.13 to decide whether or not a string E over $\Sigma_\mathcal{D}$ is a DNA expression. Whenever the function is called (recursively) for a DNA expression E, it returns the subsets of $\mathcal{A}$ that $L(\mathcal{S}(E))$ and $R(\mathcal{S}(E))$ belong to. These subsets can be used higher up in the recursion to verify that consecutive arguments of operators $\uparrow$ and $\downarrow$ fit together. `CheckExpression` assumes that the brackets and the operators in E are positioned correctly. This implies in particular that it is possible to actually *identify* the arguments of E, using Theorem 4.9.

It is not difficult to verify the assumption about the positioning of the brackets and the corresponding operators in E. One can do this by simply traversing the string from left to right, counting opening brackets (followed by operators) and closing brackets. Then the entire algorithm for the recognition of a DNA expression takes time that is linear in the length of the string.

Concatenation of DNA expressions

By Lemma 3.10, the concatenation of two formal DNA molecules is not necessarily a formal DNA molecule itself. For DNA expressions, the situation is even worse. The mere concatenation of two DNA expressions E_1 and E_2 is *never* a DNA expression, not even if E_1 and E_2 fit together.

This conclusion follows immediately from an examination of the brackets. The first and the last symbol of a DNA expression have to be corresponding opening and closing brackets. However, although the first and the last symbol of the string E_1E_2 are an opening and a closing bracket, respectively, they are not *corresponding* opening and closing brackets.

```
1.   bool CheckExpression (E, L_0, R_0)
       // checks if the string E, whose brackets and operators
       // are positioned correctly, is a DNA expression;
       // if so, then also returns the subsets L_0 and R_0 of A
       // which L(S(E)) and R(S(E)) belong to
2.   {
3.     |_0 = outermost operator of E;
4.     OK = true;
5.     n = 0;   // number of arguments
6.     while (OK and there are arguments of E left)
7.     do   n++;
8.          ε = next argument of E;
9.          if (|_0 ==↕)
10         then if (n == 1)
11.             then if (ε is not an N-word)
                     then   // it should be a DNA expression
12.                       OK = CheckExpression (ε, L_1, R_1);
13.                  fi
14.                  if (OK)   // in particular, if ε is an N-word
15.                  then L_0 = A_±;
16.                       R_0 = A_±;
17.                  fi
18.             else   // n ≥ 2
19.                  OK = false;   // more than one argument for ↕
20.             fi

21.         else   // |_0 ==↑ or |_0 ==↓;
                   // without loss of generality, assume |_0 ==↑
22.             if (ε is an N-word)
23.             then L_1 = A_+;
24.                  R_1 = A_+;
25.             else   // ε should be a DNA expression
26.                  OK = CheckExpression (ε, L_1, R_1);
27.             fi
28.             if (OK)
29.             then if (n == 1)   // first argument
30.                  then L_0 = L_1;
31.                       R_0 = R_1;
32.                  else   // n ≥ 2
33.                       if (R_0 ≠ A_− and L_1 ≠ A_−)
                           // last two arguments fit together
34.                       then R_0 = R_1;
35.                       else OK = false;
36.                       fi
37.                  fi
38.             fi
39.         fi
40.    od

41.    if (n == 0)   // operator without arguments
42.    then OK = false;
43.    fi
44.    return OK;
45.  }
```

Figure 4.5: Pseudo-code of the recursive function `CheckExpression`.

```
1.   ComputeSem (E)
       // computes and returns the semantics of the DNA expression E
2.   {
3.     if (E is an ↕-expression ⟨↕ ε1⟩)
4.     then if (ε1 is an 𝒩-word α1)
5.          then X = (α1 / c(α1));
6.          else    // ε1 is a DNA expression E1
7.               X1 = ComputeSem (E1);
8.               X = κ(X1);
9.          fi
10.         return X;
11.    else   // E is an ↑-expression or a ↓-expression;
              // without loss of generality, assume it is
              // an ↑-expression ⟨↑ ε1 ... εn⟩ for some n ≥ 1
              // and 𝒩-words and DNA expressions ε1, ..., εn
12.         for (i = 1 to n)
13.         do   if (εi is an 𝒩-word αi)
14.              then Xi = (αi / −);
15.              else    // εi is a DNA expression Ei
16.                   Xi = ComputeSem (Ei);
17.              fi
18.              if (i == 1)    // first argument
19.              then X = ν+(Xi);    // semantics up to current argument
20.              else    // i ≥ 2
21.                   if (R(X) ∈ 𝒜± and L(Xi) ∈ 𝒜±)
22.                   then X = X ·△· ν+(Xi);
23.                   else  X = X · ν+(Xi);
24.                   fi
25.              fi
26.         od
27.         return X;
28.    fi
29.  }
```

Figure 4.6: Pseudo-code of the recursive function `ComputeSem`.

Thus, E_1E_2 is just a string consisting of two separate DNA expressions. This is in line with the (natural) interpretation of DNA expressions as DNA molecules. By putting two DNA molecules in each other's vicinity, we do not automatically get a new DNA molecule. It requires a chemical reaction to achieve that. In the world of DNA expressions, the analogue of such a chemical reaction is an operator. In particular, the operators $\uparrow$ and $\downarrow$ that we have defined can be used to combine two or more DNA expressions into one new DNA expression.

4.4 Computing the semantics of a DNA expression

For a given DNA expression E, we can compute the semantics $\mathcal{S}(E)$ directly from the definition, which is part of Definition 4.1. As this definition is recursive (the semantics of a DNA expression is built up of the semantics of the arguments of the DNA expression), it is natural to use a recursive function for this. In Figure 4.6, we give such a function, called `ComputeSem`, which closely follows the definition.

The computational complexity of `ComputeSem`, as it is described in Figure 4.6, is dominated by the calls of the function κ in line 8 and the function ν^+ in lines 19, 22 and 23. Parts of the semantics of E may be subject to these functions more than once, leading to at least a quadratic time complexity in the worst case. We consider two examples of this.

In line 8, the function κ complements its argument X_1. In fact, it only complements the single-stranded components of X_1; the other components are not affected by κ (see the definition in (3.4)). In Figure 4.6, we have not specified how to find the single-stranded components. The most natural way to do this, would be to examine all components of X_1 to see if they are single-stranded.

Example 4.14 Let α be an arbitrary $\mathcal{N}$-word, and let

$$\begin{array}{ll} E_1 = \langle \updownarrow \alpha\alpha \rangle & \\ E_{2p} = \langle \uparrow E_{2p-1} \langle \updownarrow \alpha \rangle \alpha \rangle & (p \geq 1) \\ E_{2p+1} = \langle \updownarrow E_{2p} \rangle & (p \geq 1). \end{array}$$

Hence,

$$\begin{array}{rcl} E_1 & = & \langle \updownarrow \alpha\alpha \rangle \\ E_2 & = & \langle \uparrow \langle \updownarrow \alpha\alpha \rangle \langle \updownarrow \alpha \rangle \alpha \rangle \\ E_3 & = & \langle \updownarrow \langle \uparrow \langle \updownarrow \alpha\alpha \rangle \langle \updownarrow \alpha \rangle \alpha \rangle \rangle \\ E_4 & = & \langle \uparrow \langle \updownarrow \langle \uparrow \langle \updownarrow \alpha\alpha \rangle \langle \updownarrow \alpha \rangle \alpha \rangle \rangle \langle \updownarrow \alpha \rangle \alpha \rangle \\ & \ldots & \end{array}$$

It is easy to prove by induction on p, that for any $p \geq 1$,

- both E_{2p} and E_{2p+1} are DNA expressions,
- $$\begin{array}{rcl} \mathcal{S}(E_{2p}) & = & \underbrace{\binom{\alpha\alpha}{c(\alpha\alpha)}_{\triangle} \cdots \binom{\alpha\alpha}{c(\alpha\alpha)}_{\triangle}}_{p \text{ times}} \binom{\alpha}{c(\alpha)} \binom{\alpha}{-} \\ \mathcal{S}(E_{2p+1}) & = & \underbrace{\binom{\alpha\alpha}{c(\alpha\alpha)}_{\triangle} \cdots \binom{\alpha\alpha}{c(\alpha\alpha)}_{\triangle}}_{p \text{ times}} \binom{\alpha\alpha}{c(\alpha\alpha)} \end{array} \tag{4.9}$$
- $|E_{2p}| = 3 \cdot 3p + (2p+2) \cdot |\alpha|$ and $|E_{2p+1}| = 3 \cdot (3p+1) + (2p+2) \cdot |\alpha|$.

In particular, the lengths of E_{2p} and E_{2p+1} are linear in p.

Now, let $p \geq 1$ and let us apply the function `ComputeSem` to the $\updownarrow$-expression E_{2p+1}, with argument E_{2p}. When we call the function recursively for E_{2p} (in line 7), it returns $X_1 = \mathcal{S}(E_{2p})$, as described in (4.9). This semantics consists of $2p+2$ components. It takes time that is linear in p to examine them all to see if they are single-stranded. Only the last component actually *is* single-stranded, and thus is complemented by the function κ in line 8.

Likewise, at a higher level of the recursion, we have had to examine the $2p, 2p-2, 2p-4, \ldots, 4$ components of $\mathcal{S}(E_{2(p-1)}), \mathcal{S}(E_{2(p-2)}), \mathcal{S}(E_{2(p-3)}), \ldots, \mathcal{S}(E_2)$, respectively. Altogether, this takes time that is quadratic in p, and thus in the length of E_{2p+1}. ■

In lines 19, 22 and 23 of `ComputeSem`, the function ν^+ is applied to the formal DNA molecule X_i. It removes the upper nick letters from this argument. The double components preceding and succeeding such an upper nick letter are merged. The other components of X_i are not affected by ν^+ (see the definition in (3.1)). In Figure 4.6, we have not specified how to find the upper nick letters. The most natural way to do this, would be to examine all components of X_i to see if they are upper nick letters.

Example 4.15 Let α be an arbitrary $\mathcal{N}$-word, and let

$$\begin{array}{ll} E_1 = \langle \updownarrow \alpha\alpha \rangle & \\ E_{2p} = \langle \uparrow E_{2p-1}\ \alpha\, \langle \updownarrow \alpha \rangle\, \langle \updownarrow \alpha \rangle \rangle & (p \geq 1) \\ E_{2p+1} = \langle \downarrow E_{2p} \rangle & (p \geq 1). \end{array}$$

Hence,

$$\begin{array}{rcl} E_1 & = & \langle \updownarrow \alpha\alpha \rangle \\ E_2 & = & \langle \uparrow \langle \updownarrow \alpha\alpha \rangle\, \alpha\, \langle \updownarrow \alpha \rangle\, \langle \updownarrow \alpha \rangle \rangle \\ E_3 & = & \langle \downarrow \langle \uparrow \langle \updownarrow \alpha\alpha \rangle\, \alpha\, \langle \updownarrow \alpha \rangle\, \langle \updownarrow \alpha \rangle \rangle \rangle \\ E_4 & = & \langle \uparrow \langle \downarrow \langle \uparrow \langle \updownarrow \alpha\alpha \rangle\, \alpha\, \langle \updownarrow \alpha \rangle\, \langle \updownarrow \alpha \rangle \rangle \rangle\, \alpha\, \langle \updownarrow \alpha \rangle\, \langle \updownarrow \alpha \rangle \rangle \\ & \ldots & \end{array}$$

It is easy to prove by induction on p, that for any $p \geq 1$,

- both E_{2p} and E_{2p+1} are DNA expressions,
- $$\begin{array}{rcl} \mathcal{S}(E_{2p}) & = & \underbrace{\binom{\alpha\alpha}{c(\alpha\alpha)}\binom{\alpha}{-} \ldots \binom{\alpha\alpha}{c(\alpha\alpha)}\binom{\alpha}{-}}_{p \text{ times}} \binom{\alpha}{c(\alpha)} {}_{\triangle} \binom{\alpha}{c(\alpha)} \\ \mathcal{S}(E_{2p+1}) & = & \underbrace{\binom{\alpha\alpha}{c(\alpha\alpha)}\binom{\alpha}{-} \ldots \binom{\alpha\alpha}{c(\alpha\alpha)}\binom{\alpha}{-}}_{p \text{ times}} \binom{\alpha\alpha}{c(\alpha\alpha)} \end{array} \tag{4.10}$$
- $|E_{2p}| = 3 \cdot 4p + (3p+2) \cdot |\alpha|$ and $|E_{2p+1}| = 3 \cdot (4p+1) + (3p+2) \cdot |\alpha|$.

In particular, the lengths of E_{2p} and E_{2p+1} are linear in p.

Now, let $p \geq 1$ and let us apply the function `ComputeSem` to the $\downarrow$-expression E_{2p+1}, with argument E_{2p}. When we call the function recursively for E_{2p} (in line 16), it returns $X_i = \mathcal{S}(E_{2p})$, as described in (4.10). This semantics consists of $2p+3$ components. It takes time that is linear in p to examine them all to see if they are lower nick letters. Only the last but one component actually *is* a lower nick letter, and thus is removed by the function ν^- in line 19 (of the analogue for $\downarrow$-expressions E of `ComputeSem`).

Likewise, at a higher level of the recursion, we have had to examine the $2p+1, 2p-1, 2p-3, \ldots, 5$ components of $\mathcal{S}(E_{2(p-1)}), \mathcal{S}(E_{2(p-2)}), \mathcal{S}(E_{2(p-3)}), \ldots, \mathcal{S}(E_2)$, respectively. Altogether, this takes time that is quadratic in p, and thus in the length of E_{2p+1}. ■

The quadratic time complexity of `ComputeSem` can be brought back to a linear one, by means of a proper data structure to store the semantics. In particular, we could maintain lists of single-stranded components (to be utilized by κ) and lists of nick letters (to be

utilized by ν^+ and ν^-) occurring in the semantics. This data structure would be very similar to the data structure we propose in Section 9.3 to solve a similar problem.

Here, we choose a different approach to avoid the quadratic time complexity. In the description of this approach, we use E_1^* to denote the DNA expression as a whole, to clearly distinguish this DNA expression from the parameter E of (a recursive call of) `ComputeSem` and the expression-arguments E_i. When we apply `ComputeSem` to E_1^*, we recursively call the function for each DNA subexpression E of E_1^*. We now give the function three additional, boolean parameters $\uparrow$-anc, $\downarrow$-anc and $\updownarrow$-anc, which indicate whether or not the parameter E has ancestor operators $\uparrow$, $\downarrow$ and $\updownarrow$, respectively. We use these three parameters to adjust the semantics of E to the presence of these operators, *already while evaluating* E. Obviously, the three parameters are false, when we call `ComputeSem` for the first time, i.e., for E_1^* itself.

For example, suppose that E is a $\downarrow$-expression, which is a proper DNA subexpression of an $\uparrow$-expression ($\uparrow$-anc is true). Then it does not make sense (in the end) to introduce upper nick letters into $\mathcal{S}(E)$, as they will be removed by the occurrence of $\uparrow$, anyway. Therefore, we simply omit these upper nick letters. This implies that we do not have to apply ν^+ to the arguments of the operator $\uparrow$ any more.

The parameter $\downarrow$-anc is used in an analogous way, which takes away the need to apply ν^- to the arguments of the operator $\downarrow$.

Finally, we complement an $\mathcal{N}$-word-argument of an $\uparrow$-expression or $\downarrow$-expression E, if E is governed by an occurrence of $\updownarrow$ ($\updownarrow$-anc is true). Consequently, we do not have to apply κ to an expression-argument of an operator $\updownarrow$.

Note that this way, the function `ComputeSem` no longer computes the semantics *of* its parameter E, but it computes the semantics *corresponding to* E, in the context of E_1^*.

Example 4.16 Consider the $\uparrow$-expression

$$E_1^* = \langle\uparrow \langle\downarrow \alpha_1 \langle\updownarrow \alpha_2\rangle \langle\updownarrow \alpha_3\rangle\rangle \langle\uparrow \alpha_4 \langle\updownarrow \alpha_5\rangle\rangle\rangle\,,$$

where $\alpha_1, \ldots, \alpha_5$ are arbitrary $\mathcal{N}$-words. When we apply the modified version of `ComputeSem` to E_1^*, the recursive call for its first argument $E_1 = \langle\downarrow \alpha_1 \langle\updownarrow \alpha_2\rangle \langle\updownarrow \alpha_3\rangle\rangle$ will produce $\binom{-}{\alpha_1}\binom{\alpha_2\alpha_3}{c(\alpha_2\alpha_3)}$, while actually $\mathcal{S}(E_1) = \binom{-}{\alpha_1}\binom{\alpha_2}{c(\alpha_2)}^{\triangledown}\binom{\alpha_3}{c(\alpha_3)}$. The upper nick letter is omitted, because it would be removed by the outermost operator $\uparrow$ of E_1^*, anyway. Indeed,

$$\mathcal{S}(E_1^*) = \binom{-}{\alpha_1}\binom{\alpha_2\alpha_3}{c(\alpha_2\alpha_3)}\binom{\alpha_4}{-}\binom{\alpha_5}{c(\alpha_5)}$$

does not contain upper nick letters (and no lower nick letters, either). ■

When we complement $\mathcal{N}$-word-arguments of operators $\uparrow$ and $\downarrow$ because of the presence of an ancestor operator $\updownarrow$, we must be careful not to introduce additional (and incorrect) nick letters.

Example 4.17 Let us apply $\updownarrow$ to the DNA expression E_1^* from Example 4.16, yielding:

$$E_0^* = \langle\updownarrow \langle\uparrow \langle\downarrow \alpha_1 \langle\updownarrow \alpha_2\rangle \langle\updownarrow \alpha_3\rangle\rangle \langle\uparrow \alpha_4 \langle\updownarrow \alpha_5\rangle\rangle\rangle\rangle\,.$$

The $\uparrow$-argument E_1^* of E_0^* has two expression-arguments $E_1 = \langle\downarrow \alpha_1 \langle\updownarrow \alpha_2\rangle \langle\updownarrow \alpha_3\rangle\rangle$ and $E_2 = \langle\uparrow \alpha_4 \langle\updownarrow \alpha_5\rangle\rangle$. When we (recursively) call `ComputeSem` for E_1 and E_2, the boolean parameters $\uparrow$-anc and $\updownarrow$-anc are true, because E_1 and E_2 have $\uparrow$ and $\updownarrow$ as ancestor operators. Consequently, the function yields $X_1 = \binom{c(\alpha_1)\alpha_2\alpha_3}{\alpha_1 c(\alpha_2\alpha_3)}$ and $X_2 = \binom{\alpha_4\alpha_5}{c(\alpha_4\alpha_5)}$ for E_1 and E_2,

respectively. As E_1 and E_2 are consecutive arguments of an operator $\uparrow$, and both $R(X_1)$ and $L(X_2)$ are in $\mathcal{A}_\pm$, we might be tempted to introduce a lower nick letter between X_1 and X_2. This would, however, be incorrect because $\mathcal{S}(E_2) = \binom{\alpha_4}{-}\binom{\alpha_5}{c(\alpha_5)}$ and $L(\mathcal{S}(E_2)) \in \mathcal{A}_+$. Indeed,

$$\mathcal{S}(E_0^*) = \binom{c(\alpha_1)\alpha_2\alpha_3\alpha_4\alpha_5}{\alpha_1 c(\alpha_2\alpha_3\alpha_4\alpha_5)}$$

is nick free. ■

We can avoid the incorrect nick letters by giving `ComputeSem` two more return values L_0 and R_0. For a DNA expression E (the first parameter of `ComputeSem`), L_0 and R_0 indicate the subsets of $\mathcal{A}$ ($\mathcal{A}_+$, $\mathcal{A}_-$ or $\mathcal{A}_\pm$) that $L(\mathcal{S}(E))$ and $R(\mathcal{S}(E))$ belong to, respectively. These values are used (together with $\uparrow$-anc and $\downarrow$-anc) to decide whether or not to introduce a nick letter between two arguments of an operator $\uparrow$ or $\downarrow$.

In Figure 4.7, we give a pseudo-code implementation of `ComputeSem`, which includes the additional features. In this implementation, the values L_0 and R_0 are returned as parameters of the function.

It is important to realize that, when we make a recursive call to `ComputeSem` for an argument E_i of a DNA expression E, we do not have to copy E_i as a sequence of individual characters into the actual parameter of the call. Instead, we can make a 'call by reference'. For example, we may simply pass the starting position of the argument (the position of its opening bracket) in the overall DNA expression E_1^*. When, in the course of `ComputeSem`, we just keep track of the current position in the DNA expression, it does not cost time to determine this starting position. This implies that the time needed to set the actual parameters of the function is constant for each call.

For the efficiency of `ComputeSem`, it is also important that the concatenation of pieces of the semantics (as in lines $30'$ and $31'$ of the function) can be done in constant time. It should not require time that is proportional to the length of the pieces involved. Otherwise, we could easily construct an example DNA expression for which the running time of `ComputeSem` is quadratic, due to these concatenations. Fortunately, it is not difficult to achieve constant time concatenation, e.g., by storing the semantics in linked lists of $\mathcal{A}$-letters and nick letters. With such a data structure, it is also possible to efficiently return X at the end of the function (lines $11'$ and $36'$).

We can now prove that the running time of `ComputeSem` is linear in the length of its argument E.

Theorem 4.18 *Let E be an arbitrary DNA expression. The time required by the function* `ComputeSem` *for E is linear in $|E|$.*

Proof: Let us use $T_{\mathrm{CS}}(E)$ to denote the time required by `ComputeSem` for E.

When `ComputeSem` is applied to E, the function recursively examines all arguments of E. This way, in principle, every letter of E is considered. This implies that $T_{\mathrm{CS}}(E)$ is at least linear in $|E|$.

We now derive an *upper* bound on $T_{\mathrm{CS}}(E)$. For this, we define five constants, which are upper bounds on the time spent in specific parts of `ComputeSem`:

c_1 is the maximum time spent in `ComputeSem` for an $\updownarrow$-expression E, except the time spent in recursive calls of `ComputeSem` and the time spent to obtain the double $\mathcal{A}$-word $\binom{\alpha_1}{c(\alpha_1)}$ for an $\mathcal{N}$-word-argument α_1.

```
1′.  ComputeSem (E, ↑-anc, ↓-anc, ↕-anc, L_0, R_0)
       // computes and returns the semantics
       // corresponding to the DNA expression E
2′.  {
3′.    if (E is an ↕-expression ⟨↕ ε_1⟩)
4′.    then if (ε_1 is an 𝒩-word α_1)
5′.           then X = (α_1 / c(α_1));
6′.           else    // ε_1 is a DNA expression E_1
7′.                X = ComputeSem (E_1, ↑-anc, ↓-anc, true, L_1, R_1);
8′.           fi
9′.           L_0 = 𝒜_±;
10′.          R_0 = 𝒜_±;
11′.          return X;
12′.   else   // E is an ↑-expression or a ↓-expression;
              // without loss of generality, assume it is
              // an ↑-expression ⟨↑ ε_1 ... ε_n⟩ for some n ≥ 1
              // and 𝒩-words and DNA expressions ε_1, ..., ε_n
13′.          for (i = 1 to n)
14′.          do   if (ε_i is an 𝒩-word α_i)
15′.               then if (↕-anc)
16′.                      then X_i = (α_i / c(α_i));
17′.                      else  X_i = (α_i / −);
18′.                      fi
19′.                      L_i = 𝒜_+;
20′.                      R_i = 𝒜_+;
21′.               else    // ε_i is a DNA expression E_i
22′.                    X_i = ComputeSem (E_i, true, ↓-anc, ↕-anc, L_i, R_i);
23′.               fi
24′.               if (i == 1)    // first argument
25′.               then X = X_i;    // semantics up to current argument
26′.                    L_0 = L_i;
27′.                    R_0 = R_i;
28′.               else    // i ≥ 2
29′.                    if (R_0 == 𝒜_± and L_i == 𝒜_± and (not ↓-anc))
30′.                    then X = X ·_△ X_i;
31′.                    else  X = X · X_i;
32′.                    fi
33′.                    R_0 = R_i;
34′.               fi
35′.          od
36′.          return X;
37′.   fi
38′. }
```

Figure 4.7: Pseudo-code of the modified recursive function `ComputeSem`.

Hence, c_1 is the maximum time required for setting the actual parameters in line $1'$ and executing lines $3'$–$11'$ and $37'$, except line $5'$ and the recursive call in line $7'$.

c_2 is the maximum time spent in `ComputeSem` per letter of an $\mathcal{N}$-word-argument α_1 of an $\updownarrow$-expression E.

Hence, c_2 is the maximum time required per letter of α_1 for executing line $5'$. We define c_2 as the time *per letter*, because it is natural to assume that the time required to obtain the double $\mathcal{A}$-word $\binom{\alpha_1}{c(\alpha_1)}$ from the $\mathcal{N}$-word α_1 is (at most) proportional to the length of α_1.

c_3 is the maximum time spent in `ComputeSem` for an $\uparrow$-expression E, except the time spent for each of its arguments $\varepsilon_1, \ldots, \varepsilon_n$.

Hence, c_3 is the maximum time required for setting the actual parameters in line $1'$ and executing lines $3'$, $12'$, $36'$–$37'$ and the initialization of the for-loop in line $13'$.

c_4 is the maximum time spent in `ComputeSem` for an argument ε_i of an $\uparrow$-expression E, except the time spent in recursive calls of `ComputeSem` and the time spent to obtain a double $\mathcal{A}$-word $\binom{\alpha_i}{c(\alpha_i)}$ or an upper $\mathcal{A}$-word $\binom{\alpha_i}{-}$ for an $\mathcal{N}$-word-argument α_i.

Hence, c_4 is the time required for executing lines $14'$–$15'$, $18'$–$21'$, $23'$–$35'$ and the iteration in line $13'$.

c_5 is the maximum time spent in `ComputeSem` per letter of an $\mathcal{N}$-word-argument α_i of an $\uparrow$-expression E.

Hence, c_5 is the time required per letter of α_i for executing lines $16'$ or $17'$. We define c_5 as the time *per letter*, for the same reason as for c_2.

It follows from the observations made before this result (about passing parameters for the recursive calls of `ComputeSem` and about the data structure to store (pieces of) the semantics), that $c_1, \ldots, c_5$ are indeed constants.
Now, let

$$c^* = \max\left\{c_2, \frac{c_1 + c_4}{3}, c_4 + c_5, \frac{c_3 + c_4}{3}\right\}.$$

We prove by induction on the number p of operators occurring in E, that $T_{\mathrm{CS}}(E) \leq c^* \cdot |E| - c_4$. We subtract c_4 here, to be prepared for the additional constant time required for every argument of an $\uparrow$-expression E. We come back to this later.

- If $p = 1$, then E has only $\mathcal{N}$-word-arguments. When we apply `ComputeSem` to E, we do not have recursive calls of the function.

  If E is an $\updownarrow$-expression, then $E = \langle\updownarrow \alpha_1\rangle$ for an $\mathcal{N}$-word α_1. In this case, $|E| = 3+|\alpha_1|$ and

  $$T_{\mathrm{CS}}(E) \;\leq\; c_1 + c_2 \cdot |\alpha_1| \leq 3c^* - c_4 + c^*|\alpha_1| = c^* \cdot |E| - c_4,$$

  where the second inequality follows from $c^* \geq \frac{c_1+c_4}{3}$ (which is equivalent to $c_1 \leq 3c^* - c_4$) and $c^* \geq c_2$.

If E is an $\uparrow$-expression, then $E = \langle\uparrow \alpha_1 \dots \alpha_n\rangle$ for some $n \geq 1$ and $\mathcal{N}$-words $\alpha_1, \dots, \alpha_n$. In this case, $|E| = 3 + |\alpha_1 \dots \alpha_n|$ and

$$\begin{aligned} T_{\mathrm{CS}}(E) &\leq c_3 + (c_4 + c_5 \cdot |\alpha_1|) + \dots + (c_4 + c_5 \cdot |\alpha_n|) \\ &\leq 3c^* - c_4 + (c_4 + c_5) \cdot |\alpha_1| + \dots + (c_4 + c_5) \cdot |\alpha_n|) \\ &\leq 3c^* - c_4 + c^* \cdot |\alpha_1 \dots \alpha_n| = c^* \cdot |E| - c_4, \end{aligned}$$

where the second inequality follows from $c^* \geq \frac{c_3+c_4}{3}$ and $|\alpha_i| \geq 1$ for $i = 1, \dots, n$, and the third inequality follows from $c^* \geq c_4 + c_5$.

If E is a $\downarrow$-expression, then the proof is completely analogous.

- Let $p \geq 1$, and suppose that $T_{\mathrm{CS}}(E) \leq c^* \cdot |E| - c_4$ for all DNA expressions containing at most p operators (induction hypothesis). We now consider a DNA expression E that contains $p + 1$ operators.

  If E is an $\updownarrow$-expression, then $E = \langle\updownarrow E_1\rangle$ for a DNA expression E_1 with p operators. By the induction hypothesis, $T_{\mathrm{CS}}(E_1) \leq c^* \cdot |E_1| - c_4$. In this case, $|E| = 3 + |E_1|$ and

  $$T_{\mathrm{CS}}(E) \leq c_1 + T_{\mathrm{CS}}(E_1) \leq 3c^* - c_4 + c^* \cdot |E_1| - c_4 = c^* \cdot |E| - 2c_4,$$

  where the second inequality follows from $c^* \geq \frac{c_1+c_4}{3}$.

  If E is an $\uparrow$-expression, then $E = \langle\uparrow \varepsilon_1 \dots \varepsilon_n\rangle$ for some $n \geq 1$ and $\mathcal{N}$-words and DNA expressions $\varepsilon_1, \dots, \varepsilon_n$. In this case, $|E| = 3 + |\varepsilon_1 \dots \varepsilon_n|$. We consider the time spent by `ComputeSem` on a single argument ε_i of E.

  If ε_i is an $\mathcal{N}$-word α_i, then this time is bounded by $c_4 + c_5 \cdot |\alpha_i|$. As we have seen in the proof of the base case of the induction, this is at most $c^* \cdot |\alpha_i|$. If, on the other hand, ε_i is a DNA expression E_i, then E_i contains at most p operators. Hence, we can apply the induction hypothesis to it: $T_{\mathrm{CS}}(E_i) \leq c^* \cdot |E_i| - c_4$. Now the time spent by `ComputeSem` on the argument E_i of E is bounded by $c_4 + T_{\mathrm{CS}}(E_i) \leq c^* \cdot |E_i|$. Note that this is where we benefit from the term $-c_4$ in the induction hypothesis.

  We conclude that both if ε_i is an $\mathcal{N}$-word α_i, and if it is a DNA expression E_i, `ComputeSem` spends at most $c^* \cdot |\varepsilon_i|$ on this argument of E. This implies that

  $$\begin{aligned} T_{\mathrm{CS}}(E) &\leq c_3 + c^* \cdot |\varepsilon_1| + \dots + c^* \cdot |\varepsilon_n| \\ &\leq 3c^* - c_4 + c^* \cdot |\varepsilon_1 \dots \varepsilon_n| = c^*|E| - c_4, \end{aligned}$$

  where the second inequality follows from $c^* \geq \frac{c_3+c_4}{3}$.

  If E is a $\downarrow$-expression, then the proof is completely analogous.

We have thus proved that for each DNA expression E, $T_{\mathrm{CS}}(E) \leq c^* \cdot |E| - c_4$. This implies that $T_{\mathrm{CS}}(E)$ is at most linear in $|E|$. □

We briefly discuss the space complexity of the function `ComputeSem`. For this, let us again use E_1^* to denote the DNA expression that we apply the function to.

The assumption that we simply pass the starting position of an expression-argument E_i to a recursive call of `ComputeSem` (rather than copying E_i itself) is also important for the space complexity. It implies that the space required to store the parameters of

a single call is constant. For each DNA subexpression of E_1^*, the function `ComputeSem` is called exactly once. Since there is a 1–1 correspondence between DNA subexpressions and occurrences of operators in a DNA expression, the total space required to store the parameters for all calls is at most linear in $|E_1^*|$.

For every $\mathcal{N}$-word-argument α_i of its parameter E, `ComputeSem` produces an upper $\mathcal{A}$-word, a lower $\mathcal{A}$-word or a double $\mathcal{A}$-word. For an expression-argument, it may produce a nick letter. These $\mathcal{A}$-words and nick letters become parts of the overall semantics $\mathcal{S}(E_1^*)$. Because each part of the semantics is produced only once (and no copies are made) during the execution of `ComputeSem`, the total space required to store (parts of) the semantics is linear in $|\mathcal{S}(E_1^*)|$. It is not too hard to prove that this is at most linear in $|E_1^*|$.

It is instructive to consider examples of DNA expressions for which `ComputeSem` indeed requires linear space, or for which it really requires less than linear space.

Example 4.19 Let α be an arbitrary $\mathcal{N}$-word, and let E_p be defined by

$$E_p = \underbrace{\langle\uparrow \langle\uparrow \ldots \langle\uparrow}_{p \text{ times}} \alpha \underbrace{\rangle \ldots \rangle\rangle}_{p \text{ times}} \qquad (p \geq 1).$$

It is easy to see that for any $p \geq 1$, E_p is a DNA expression, with $|E_p| = 3p + |\alpha|$ and $\mathcal{S}(E_p) = \binom{\alpha}{_}$. When we call the recursive function `ComputeSem` for E_p, the maximum nesting level of the recursion becomes p. This implies that the space we need to store the parameters of subsequent recursive calls is linear in p, and thus linear in $|E_p|$. ■

Example 4.20 Let α be an arbitrary $\mathcal{N}$-word, and let E_p be defined by

$$E_p = \left\langle \uparrow \underbrace{\alpha\alpha \ldots \alpha}_{p \text{ times}} \right\rangle \qquad (p \geq 1).$$

It is easy to see that for any $p \geq 1$, E_p is a DNA expression, with $|E_p| = 3 + p \cdot |\alpha|$ and $\mathcal{S}(E_p) = \binom{\overbrace{\alpha\alpha \ldots \alpha}^{p \text{ times}}}{_}$. Hence, $|\mathcal{S}(E_p)| = p \cdot |\alpha|$, which implies that the space we need to store the result of `ComputeSem` is linear in p, and thus linear in $|E_p|$. ■

Example 4.21 Let α be an arbitrary $\mathcal{N}$-word, and let E_p be defined by

$$E_p = \left\langle \underbrace{\uparrow \underbrace{\langle\uparrow \langle\uparrow \ldots \langle\uparrow}_{p \text{ times}} \alpha \underbrace{\rangle \ldots \rangle\rangle}_{p \text{ times}} \ldots \underbrace{\langle\uparrow \langle\uparrow \ldots \langle\uparrow}_{p \text{ times}} \alpha \underbrace{\rangle \ldots \rangle\rangle}_{p \text{ times}}}_{p \text{ times}} \right\rangle \qquad (p \geq 1).$$

It is easy to see that for any $p \geq 1$, E_p is a DNA expression, with $|E_p| = 3 + p \cdot (3p + |\alpha|) = 3 + 3p^2 + p \cdot |\alpha|$ and $\mathcal{S}(E_p) = \binom{\overbrace{\alpha\alpha \ldots \alpha}^{p \text{ times}}}{_}$. Because $|E_p|$ is quadratic in p, p is linear in $\sqrt{|E_p|}$.

When we call the recursive function `ComputeSem` for E_p, the maximum nesting level of the recursion becomes $p + 1$. This implies that the space we need to store the parameters of subsequent recursive calls is linear in p, and thus linear in $\sqrt{|E_p|}$.

Moreover, $|\mathcal{S}(E_p)| = p \cdot |\alpha|$, which implies that the space we need to store the result of `ComputeSem` is linear in p, and thus linear in $\sqrt{|E_p|}$. ■

We can therefore conclude:

Theorem 4.22 *Let E be an arbitrary DNA expression. In the worst case, the space required by the function* `ComputeSem` *for E is linear in $|E|$.*

4.5 A context-free grammar for $\mathcal{D}$

As we have established in Lemma 3.3, the language $\mathcal{F}$ of formal DNA molecules is regular. This is not the case with the language $\mathcal{D}$ of all DNA expressions. This is intuitively clear from the fact that every DNA expression contains matching brackets $\langle$ and $\rangle$, and that these brackets may be deeply nested. We use this intuition in a formal proof.

Lemma 4.23 *The language $\mathcal{D}$ of DNA expressions is not regular.*

Proof: Let α be an arbitrary $\mathcal{N}$-word. Then $E_1 = \langle \updownarrow \alpha \rangle$ is a DNA expression, and $\mathcal{S}(E_1) = \binom{\alpha}{c(\alpha)}$. By definition, also $E_2 = \langle \updownarrow \langle \updownarrow \alpha \rangle \rangle$ is a DNA expression, with the same semantics. It is easy to see that for arbitrary $p \geq 1$,

$$E_p = \underbrace{\langle \updownarrow \langle \updownarrow \ldots \langle \updownarrow}_{p \text{ times}} \alpha \ \underbrace{\rangle \ldots \rangle \rangle}_{p \text{ times}}$$

is a DNA expression, with $\mathcal{S}(E_p) = \binom{\alpha}{c(\alpha)}$. By the pumping lemma for regular languages (Proposition 2.7), a language requiring brackets to match and containing such DNA expressions is not regular. □

The language $\mathcal{D}$ is, however, context-free, because it can be generated by a context-free grammar. We will give such a grammar, here. It is a 4-tuple $G_1 = (V_1, \Sigma_1, P_1, S_1)$, which is based on three types of non-terminal symbols: E (which denotes a DNA expression), U (a sequence of one or more arguments of an $\uparrow$-expression) and L (a sequence of one or more arguments of a $\downarrow$-expression).

The crucial issue in the construction of a context-free grammar generating $\mathcal{D}$, is that we must somehow incorporate the requirement that consecutive arguments of an operator $\uparrow$ or $\downarrow$ fit together by upper strands or lower strands, respectively. For this, the non-terminal symbols E, U and L have two subscripts. The first subscript denotes whether or not one of the strands of the (sub)molecule represented by the non-terminal has to cover the other strand to the left. If it is $+$, then the upper strand must cover the lower strand to the left; if it is $-$, then the lower strand must cover the upper strand to the left; if it is $\star$, then it does not matter if either strand strictly covers the other strand to the left. The second subscript has an analogous meaning, with respect to covering to the right. For example, the symbol $U_{+,-}$ denotes a sequence of arguments of $\uparrow$, for which the upper strand (of the first argument) must cover the lower strand to the left, and the lower strand (of the last argument) must cover the upper strand to the right.

In addition to the above, G_1 has one more non-terminal symbol: α, which represents an arbitrary $\mathcal{N}$-word. We thus have the following set of non-terminal symbols:

$$\{E_{x,y}, U_{x,y}, L_{x,y} \mid x, y \in \{\star, +, -\}\} \ \cup \ \{\alpha\}.$$

The axiom is $S_1 = E_{\star,\star}$, which denotes a DNA expression without restrictions on the two strands. The alphabet Σ_1 of terminal symbols is equal to $\Sigma_{\mathcal{D}}$: $\Sigma_1 = \{\mathrm{A}, \mathrm{C}, \mathrm{G}, \mathrm{T}, \uparrow, \downarrow, \updownarrow, \langle\ , \rangle\}$.

Before we present the productions in G_1 (i.e., the elements of P_1) we discuss why we have exactly those productions.

We first consider the productions for (rewriting) a non-terminal symbol $E_{x,y}$ with $x, y \in \{\star, +, -\}$, which represents a DNA expression.

By Lemma 4.13(2), for any $\updownarrow$-expression E, we have $L(\mathcal{S}(E)), R(\mathcal{S}(E)) \in \mathcal{A}_{\pm}$. Hence, the upper strand of E covers the lower strand to both the left and the right, and vice

versa. This implies that, regardless of the subscripts x and y, we may rewrite $E_{x,y}$ into any $\updownarrow$-expression. Therefore, we have productions $E_{x,y} \longrightarrow \langle \updownarrow \alpha \rangle$ and $E_{x,y} \longrightarrow \langle \updownarrow E_{\star,\star} \rangle$. Indeed, the non-terminal α occurring in the former production represents an arbitrary $\mathcal{N}$-word, and the non-terminal $E_{\star,\star}$ occurring as argument of $\updownarrow$ in the latter production represents an arbitrary DNA expression, without restrictions on the strands.

By Lemma 4.13(3), for an $\uparrow$-expression E, the values of the functions L and R depend (solely) on the values for the first and the last argument of E, respectively. Therefore, if we want to rewrite $E_{x,y}$ into an $\uparrow$-expression, then the subscripts x and y simply carry over to the non-terminal U representing the arguments of the $\uparrow$-expression. We thus have a production $E_{x,y} \longrightarrow \langle \uparrow U_{x,y} \rangle$. Analogously, we have $E_{x,y} \longrightarrow \langle \downarrow L_{x,y} \rangle$.

Next, consider a non-terminal symbol $U_{x,y}$ for some subscripts $x, y \in \{\star, +, -\}$. This non-terminal must be rewritten into a sequence of $n \geq 1$ arguments for an occurrence of $\uparrow$. We do this in a right-linear, recursive way: we rewrite $U_{x,y}$ into a non-terminal α or E (with some subscripts) representing the first argument, possibly followed by another non-terminal U (with some subscripts), representing the second and later arguments.

If $n \geq 2$, so that we indeed need a new non-terminal symbol U for the second and later arguments, then the subscripts in the right-hand side of the production reflect the requirement that the arguments of $\uparrow$ fit together by upper strands. In particular, if the first argument is a DNA expression, then the second subscript of the non-terminal symbol E representing it must be $+$. Further, the first subscript of the new non-terminal symbol U must be $+$.

Example 4.24 The non-terminal symbol $U_{\star,+}$ represents a sequence of arguments of $\uparrow$ with no restrictions on the left-hand side of the first argument, but for which the upper strand of the last argument must cover the lower strand on the right. We have four productions for this symbol: $U_{\star,+} \longrightarrow \alpha$ (indeed, the upper strand of $\mathcal{S}^+(\alpha) = \binom{\alpha}{-}$ covers the lower strand on the right), $U_{\star,+} \longrightarrow E_{\star,+}$, $U_{\star,+} \longrightarrow \alpha U_{+,+}$ and $U_{\star,+} \longrightarrow E_{\star,+} U_{+,+}$ (see the productions in line 11 below). ■

Example 4.25 The non-terminal symbol $U_{-,\star}$ represents a sequence of arguments of $\uparrow$ for which the lower strand of the first argument must cover the upper strand on the left, and for which there are no restrictions on the right-hand side of the last argument. Because the lower strand of $\mathcal{S}^+(\alpha) = \binom{\alpha}{-}$ does not cover the upper strand on the left, the first argument cannot be an $\mathcal{N}$-word α. Hence, we have only two productions for this symbol: $U_{-,\star} \longrightarrow E_{-,\star}$ and $U_{-,\star} \longrightarrow E_{-,+} U_{+,\star}$ (see the productions in line 16 below). ■

There is, of course, an analogous explanation for the productions for a non-terminal $L_{x,y}$ with $x, y \in \{\star, +, -\}$.

The grammatical structure of an $\mathcal{N}$-word (represented by the non-terminal symbol α) is similar to that of the sequence of arguments of $\uparrow$ or $\downarrow$. An $\mathcal{N}$-word is an arbitrary sequence of $r \geq 1$ $\mathcal{N}$-letters. We obtain this sequence from the non-terminal symbol α by recursively rewriting this symbol into an $\mathcal{N}$-letter, possibly followed by another non-terminal α.

Thus, the set P_1 consists of the following productions:

1. $E_{\star,\star} \longrightarrow \langle \updownarrow \alpha \rangle \mid \langle \updownarrow E_{\star,\star} \rangle \mid \langle \uparrow U_{\star,\star} \rangle \mid \langle \downarrow L_{\star,\star} \rangle$
2. $E_{\star,+} \longrightarrow \langle \updownarrow \alpha \rangle \mid \langle \updownarrow E_{\star,\star} \rangle \mid \langle \uparrow U_{\star,+} \rangle \mid \langle \downarrow L_{\star,+} \rangle$
3. $E_{\star,-} \longrightarrow \langle \updownarrow \alpha \rangle \mid \langle \updownarrow E_{\star,\star} \rangle \mid \langle \uparrow U_{\star,-} \rangle \mid \langle \downarrow L_{\star,-} \rangle$

4. $E_{+,\star} \longrightarrow \langle\updownarrow \alpha\rangle \mid \langle\updownarrow E_{\star,\star}\rangle \mid \langle\uparrow U_{+,\star}\rangle \mid \langle\downarrow L_{+,\star}\rangle$
5. $E_{+,+} \longrightarrow \langle\updownarrow \alpha\rangle \mid \langle\updownarrow E_{\star,\star}\rangle \mid \langle\uparrow U_{+,+}\rangle \mid \langle\downarrow L_{+,+}\rangle$
6. $E_{+,-} \longrightarrow \langle\updownarrow \alpha\rangle \mid \langle\updownarrow E_{\star,\star}\rangle \mid \langle\uparrow U_{+,-}\rangle \mid \langle\downarrow L_{+,-}\rangle$
7. $E_{-,\star} \longrightarrow \langle\updownarrow \alpha\rangle \mid \langle\updownarrow E_{\star,\star}\rangle \mid \langle\uparrow U_{-,\star}\rangle \mid \langle\downarrow L_{-,\star}\rangle$
8. $E_{-,+} \longrightarrow \langle\updownarrow \alpha\rangle \mid \langle\updownarrow E_{\star,\star}\rangle \mid \langle\uparrow U_{-,+}\rangle \mid \langle\downarrow L_{-,+}\rangle$
9. $E_{-,-} \longrightarrow \langle\updownarrow \alpha\rangle \mid \langle\updownarrow E_{\star,\star}\rangle \mid \langle\uparrow U_{-,-}\rangle \mid \langle\downarrow L_{-,-}\rangle$
10. $U_{\star,\star} \longrightarrow \alpha \mid E_{\star,\star} \mid \alpha U_{+,\star} \mid E_{\star,+}U_{+,\star}$
11. $U_{\star,+} \longrightarrow \alpha \mid E_{\star,+} \mid \alpha U_{+,+} \mid E_{\star,+}U_{+,+}$
12. $U_{\star,-} \longrightarrow E_{\star,-} \mid \alpha U_{+,-} \mid E_{\star,+}U_{+,-}$
13. $U_{+,\star} \longrightarrow \alpha \mid E_{+,\star} \mid \alpha U_{+,\star} \mid E_{+,+}U_{+,\star}$
14. $U_{+,+} \longrightarrow \alpha \mid E_{+,+} \mid \alpha U_{+,+} \mid E_{+,+}U_{+,+}$
15. $U_{+,-} \longrightarrow E_{+,-} \mid \alpha U_{+,-} \mid E_{+,+}U_{+,-}$
16. $U_{-,\star} \longrightarrow E_{-,\star} \mid E_{-,+}U_{+,\star}$
17. $U_{-,+} \longrightarrow E_{-,+} \mid E_{-,+}U_{+,+}$
18. $U_{-,-} \longrightarrow E_{-,-} \mid E_{-,+}U_{+,-}$
19. $L_{\star,\star} \longrightarrow \alpha \mid E_{\star,\star} \mid \alpha L_{-,\star} \mid E_{\star,-}L_{-,\star}$
20. $L_{\star,+} \longrightarrow E_{\star,+} \mid \alpha L_{-,+} \mid E_{\star,-}L_{-,+}$
21. $L_{\star,-} \longrightarrow \alpha \mid E_{\star,-} \mid \alpha L_{-,-} \mid E_{\star,-}L_{-,-}$
22. $L_{+,\star} \longrightarrow E_{+,\star} \mid E_{+,-}L_{-,\star}$
23. $L_{+,+} \longrightarrow E_{+,+} \mid E_{+,-}L_{-,+}$
24. $L_{+,-} \longrightarrow E_{+,-} \mid E_{+,-}L_{-,-}$
25. $L_{-,\star} \longrightarrow \alpha \mid E_{-,\star} \mid \alpha L_{-,\star} \mid E_{-,-}L_{-,\star}$
26. $L_{-,+} \longrightarrow E_{-,+} \mid \alpha L_{-,+} \mid E_{-,-}L_{-,+}$
27. $L_{-,-} \longrightarrow \alpha \mid E_{-,-} \mid \alpha L_{-,-} \mid E_{-,-}L_{-,-}$
28. $\alpha \longrightarrow \mathrm{A} \mid \mathrm{C} \mid \mathrm{G} \mid \mathrm{T} \mid \mathrm{A}\alpha \mid \mathrm{C}\alpha \mid \mathrm{G}\alpha \mid \mathrm{T}\alpha$

Note that the first nine lines of the above list can be summarized by

$$E_{x,y} \longrightarrow \langle\updownarrow \alpha\rangle \mid \langle\updownarrow E_{\star,\star}\rangle \mid \langle\uparrow U_{x,y}\rangle \mid \langle\downarrow L_{x,y}\rangle \qquad (x, y \in \{\star, +, -\}).$$

Similarly, sets of two, three or four other lines of productions can be summarized by single lines. However, the description by separate lines for each of the non-terminal symbols makes it easier to refer to a particular production, as we do in the example below.

Note also that there is a subtle relation between grammar G_1 and the recursive functions `CheckExpression` and `ComputeSem` from Figure 4.5 and Figure 4.7. Both functions return in their parameters L_0 and R_0 the subsets of $\mathcal{A}$ ($\mathcal{A}_+$, $\mathcal{A}_-$ or $\mathcal{A}_\pm$) that the left-hand side and the right-hand side of $\mathcal{S}(E)$ belong to. That is, these values are passed *upwards* in the tree of recursive calls of the functions.

In the grammar, the subscripts $+$, $-$, $\star$ of the non-terminal symbols E, U and L indicate the subsets ($\mathcal{A}_+ \cup \mathcal{A}_\pm$, $\mathcal{A}_- \cup \mathcal{A}_\pm$ or the entire set $\mathcal{A}$) that the left-hand side and the right-hand side of the (sub)molecule represented *should* belong to. That is, this information is passed *downwards* in the derivation tree of a DNA expression.

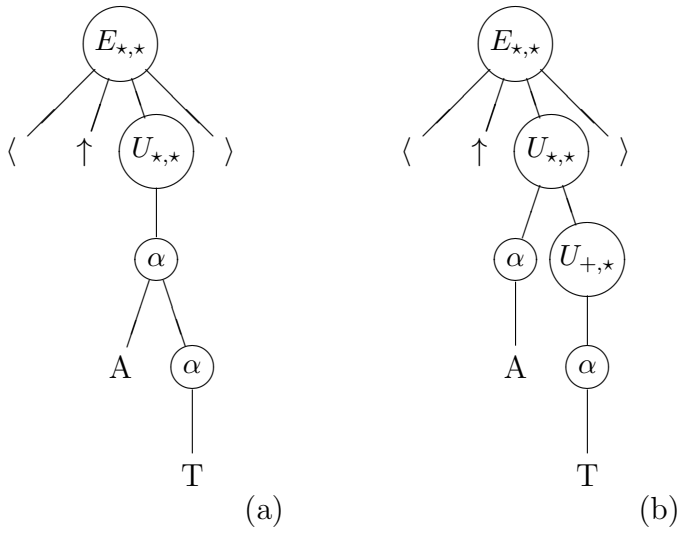


Figure 4.8: Two derivation trees in G_1 for the DNA expression $\langle\uparrow \mathrm{AT}\rangle$ from Example 4.26. (a) The derivation tree corresponding to Derivation (4.11). (b) The derivation tree corresponding to Derivation (4.12).

The context-free grammar G_1 is not entirely unambiguous, as can be seen from the following, simple example:

Example 4.26 Consider the $\uparrow$-expression $E = \langle\uparrow \mathrm{AT}\rangle$. There exist two leftmost derivations of E in G_1:

$$\begin{aligned} E_{\star,\star} &\overset{1,3}{\Longrightarrow} \langle\uparrow U_{\star,\star}\rangle \\ &\overset{10,1}{\Longrightarrow} \langle\uparrow \alpha\rangle \\ &\overset{28,5}{\Longrightarrow} \langle\uparrow \mathrm{A}\alpha\rangle \\ &\overset{28,4}{\Longrightarrow} \langle\uparrow \mathrm{AT}\rangle \end{aligned} \tag{4.11}$$

and

$$\begin{aligned} E_{\star,\star} &\overset{1,3}{\Longrightarrow} \langle\uparrow U_{\star,\star}\rangle \\ &\overset{10,3}{\Longrightarrow} \langle\uparrow \alpha U_{+,\star}\rangle \\ &\overset{28,1}{\Longrightarrow} \langle\uparrow \mathrm{A}U_{+,\star}\rangle \\ &\overset{13,1}{\Longrightarrow} \langle\uparrow \mathrm{A}\alpha\rangle \\ &\overset{28,4}{\Longrightarrow} \langle\uparrow \mathrm{AT}\rangle \end{aligned} \tag{4.12}$$

Here, numbers i, j above an arrow $\Longrightarrow$ indicate that we have used production (i, j) for the corresponding derivation step. In the former derivation, the $\mathcal{N}$-word AT is derived from a single non-terminal symbol α. In the latter derivation, the two $\mathcal{N}$-letters A and T are derived from two independent non-terminal symbols α. The corresponding derivation trees are depicted in Figure 4.8. Indeed, for both trees, the yield is equal to $\langle\uparrow \mathrm{AT}\rangle$. ■

Hence, let α_i be a maximal $\mathcal{N}$-word occurrence of length $|\alpha_i| \geq 2$, with parent operator $\uparrow$ or $\downarrow$. Then α_i can be derived in G_1 from r independent, consecutive symbols α, where

r can take any value satisfying $1 \leq r \leq |\alpha_i|$. Moreover, if $1 < r < |\alpha_i|$, then there are multiple ways to partition α_i over the r symbols α. As follows from a careful inspection of the productions in the grammar, this is the only type of ambiguity occurring in G_1.

Example 4.27 The DNA expression from Example 4.2 is the result of many different derivations in G_1. A leftmost derivation is

$$
\begin{aligned}
E_{\star,\star} &\overset{1,4}{\Longrightarrow} \langle\downarrow L_{\star,\star}\rangle \\
&\overset{19,3}{\Longrightarrow} \langle\downarrow \alpha L_{-,\star}\rangle \\
&\overset{28,4}{\Longrightarrow} \langle\downarrow \mathrm{T} L_{-,\star}\rangle \\
&\overset{25,4}{\Longrightarrow} \langle\downarrow \mathrm{T} E_{-,-} L_{-,\star}\rangle \\
&\overset{9,3}{\Longrightarrow} \langle\downarrow \mathrm{T} \langle\uparrow U_{-,-}\rangle L_{-,\star}\rangle \\
&\overset{18,2}{\Longrightarrow} \langle\downarrow \mathrm{T} \langle\uparrow E_{-,+} U_{+,-}\rangle L_{-,\star}\rangle \\
&\overset{8,1}{\Longrightarrow} \langle\downarrow \mathrm{T} \langle\uparrow \langle\updownarrow \alpha\rangle U_{+,-}\rangle L_{-,\star}\rangle \\
&\overset{28,2}{\Longrightarrow} \langle\downarrow \mathrm{T} \langle\uparrow \langle\updownarrow \mathrm{C}\rangle U_{+,-}\rangle L_{-,\star}\rangle \\
&\overset{15,2}{\Longrightarrow} \langle\downarrow \mathrm{T} \langle\uparrow \langle\updownarrow \mathrm{C}\rangle \alpha U_{+,-}\rangle L_{-,\star}\rangle \\
&\overset{28,5}{\Longrightarrow} \langle\downarrow \mathrm{T} \langle\uparrow \langle\updownarrow \mathrm{C}\rangle \mathrm{A}\alpha U_{+,-}\rangle L_{-,\star}\rangle \\
&\overset{28,4}{\Longrightarrow} \langle\downarrow \mathrm{T} \langle\uparrow \langle\updownarrow \mathrm{C}\rangle \mathrm{AT} U_{+,-}\rangle L_{-,\star}\rangle \\
&\overset{15,1}{\Longrightarrow} \langle\downarrow \mathrm{T} \langle\uparrow \langle\updownarrow \mathrm{C}\rangle \mathrm{AT} E_{+,-}\rangle L_{-,\star}\rangle \\
&\overset{6,4}{\Longrightarrow} \langle\downarrow \mathrm{T} \langle\uparrow \langle\updownarrow \mathrm{C}\rangle \mathrm{AT} \langle\downarrow L_{+,-}\rangle\rangle L_{-,\star}\rangle \\
&\overset{24,2}{\Longrightarrow} \langle\downarrow \mathrm{T} \langle\uparrow \langle\updownarrow \mathrm{C}\rangle \mathrm{AT} \langle\downarrow E_{+,-} L_{-,-}\rangle\rangle L_{-,\star}\rangle \\
&\overset{6,1}{\Longrightarrow} \langle\downarrow \mathrm{T} \langle\uparrow \langle\updownarrow \mathrm{C}\rangle \mathrm{AT} \langle\downarrow \langle\updownarrow \alpha\rangle L_{-,-}\rangle\rangle L_{-,\star}\rangle \\
&\overset{28,3}{\Longrightarrow} \langle\downarrow \mathrm{T} \langle\uparrow \langle\updownarrow \mathrm{C}\rangle \mathrm{AT} \langle\downarrow \langle\updownarrow \mathrm{G}\rangle L_{-,-}\rangle\rangle L_{-,\star}\rangle \\
&\overset{27,2}{\Longrightarrow} \langle\downarrow \mathrm{T} \langle\uparrow \langle\updownarrow \mathrm{C}\rangle \mathrm{AT} \langle\downarrow \langle\updownarrow \mathrm{G}\rangle E_{-,-}\rangle\rangle L_{-,\star}\rangle \\
&\overset{9,1}{\Longrightarrow} \langle\downarrow \mathrm{T} \langle\uparrow \langle\updownarrow \mathrm{C}\rangle \mathrm{AT} \langle\downarrow \langle\updownarrow \mathrm{G}\rangle \langle\updownarrow \alpha\rangle\rangle\rangle L_{-,\star}\rangle \\
&\overset{28,2}{\Longrightarrow} \langle\downarrow \mathrm{T} \langle\uparrow \langle\updownarrow \mathrm{C}\rangle \mathrm{AT} \langle\downarrow \langle\updownarrow \mathrm{G}\rangle \langle\updownarrow \mathrm{C}\rangle\rangle\rangle L_{-,\star}\rangle \\
&\overset{25,2}{\Longrightarrow} \langle\downarrow \mathrm{T} \langle\uparrow \langle\updownarrow \mathrm{C}\rangle \mathrm{AT} \langle\downarrow \langle\updownarrow \mathrm{G}\rangle \langle\updownarrow \mathrm{C}\rangle\rangle\rangle E_{-,\star}\rangle \\
&\overset{7,3}{\Longrightarrow} \langle\downarrow \mathrm{T} \langle\uparrow \langle\updownarrow \mathrm{C}\rangle \mathrm{AT} \langle\downarrow \langle\updownarrow \mathrm{G}\rangle \langle\updownarrow \mathrm{C}\rangle\rangle\rangle \langle\uparrow U_{-,\star}\rangle\rangle \\
&\overset{16,2}{\Longrightarrow} \langle\downarrow \mathrm{T} \langle\uparrow \langle\updownarrow \mathrm{C}\rangle \mathrm{AT} \langle\downarrow \langle\updownarrow \mathrm{G}\rangle \langle\updownarrow \mathrm{C}\rangle\rangle\rangle \langle\uparrow E_{-,+} U_{+,\star}\rangle\rangle \\
&\overset{8,1}{\Longrightarrow} \langle\downarrow \mathrm{T} \langle\uparrow \langle\updownarrow \mathrm{C}\rangle \mathrm{AT} \langle\downarrow \langle\updownarrow \mathrm{G}\rangle \langle\updownarrow \mathrm{C}\rangle\rangle\rangle \langle\uparrow \langle\updownarrow \alpha\rangle U_{+,\star}\rangle\rangle \\
&\overset{28,1}{\Longrightarrow} \langle\downarrow \mathrm{T} \langle\uparrow \langle\updownarrow \mathrm{C}\rangle \mathrm{AT} \langle\downarrow \langle\updownarrow \mathrm{G}\rangle \langle\updownarrow \mathrm{C}\rangle\rangle\rangle \langle\uparrow \langle\updownarrow \mathrm{A}\rangle U_{+,\star}\rangle\rangle \\
&\overset{13,2}{\Longrightarrow} \langle\downarrow \mathrm{T} \langle\uparrow \langle\updownarrow \mathrm{C}\rangle \mathrm{AT} \langle\downarrow \langle\updownarrow \mathrm{G}\rangle \langle\updownarrow \mathrm{C}\rangle\rangle\rangle \langle\uparrow \langle\updownarrow \mathrm{A}\rangle E_{+,\star}\rangle\rangle \\
&\overset{4,1}{\Longrightarrow} \langle\downarrow \mathrm{T} \langle\uparrow \langle\updownarrow \mathrm{C}\rangle \mathrm{AT} \langle\downarrow \langle\updownarrow \mathrm{G}\rangle \langle\updownarrow \mathrm{C}\rangle\rangle\rangle \langle\uparrow \langle\updownarrow \mathrm{A}\rangle \langle\updownarrow \alpha\rangle\rangle\rangle
\end{aligned}
$$

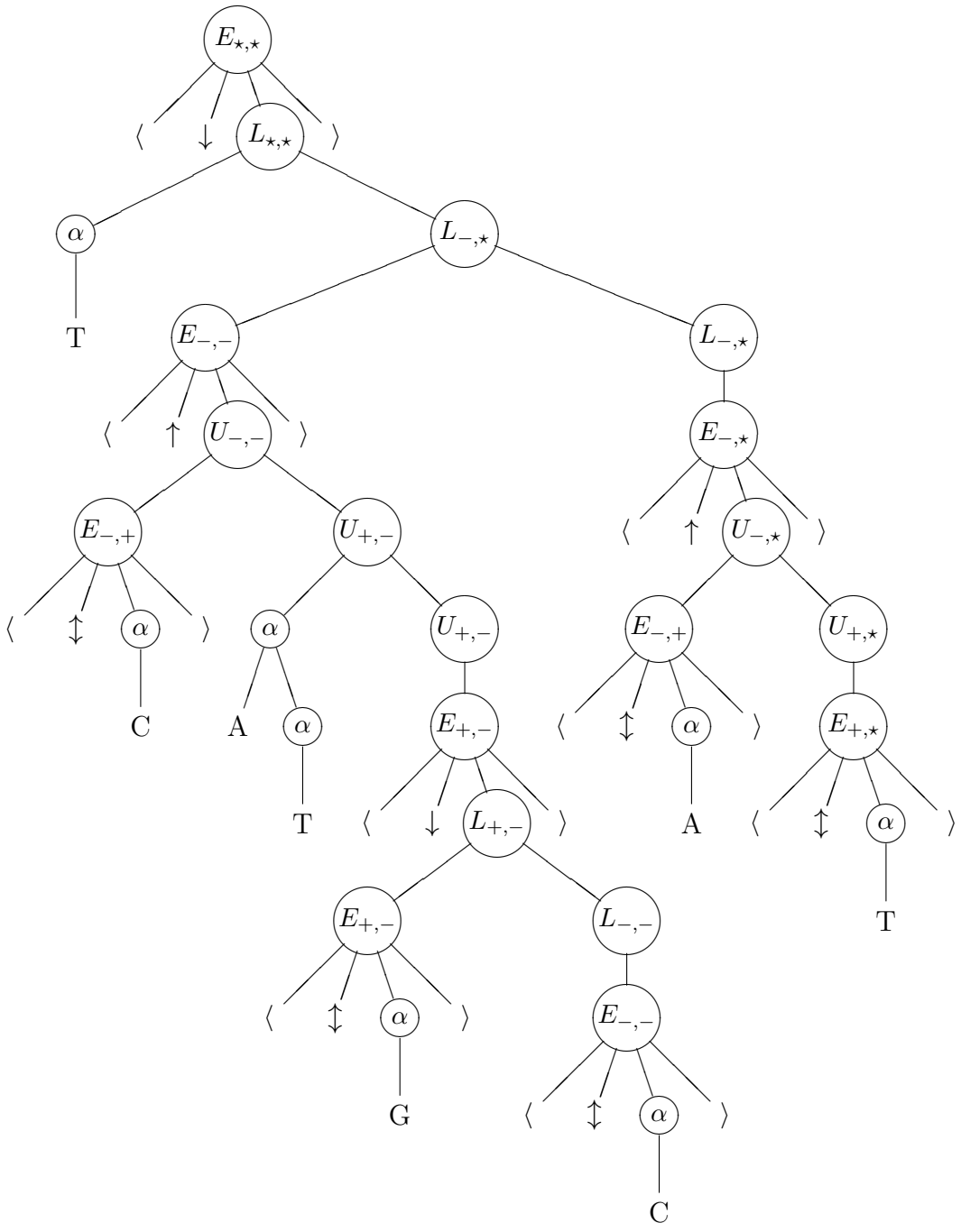


Figure 4.9: The derivation tree corresponding to the derivation in G_1 from Example 4.27.

$$\stackrel{28,4}{\Longrightarrow} \quad \langle\downarrow \mathrm{T}\, \langle\uparrow \langle\updownarrow \mathrm{C}\rangle \mathrm{AT}\, \langle\downarrow \langle\updownarrow \mathrm{G}\rangle \langle\updownarrow \mathrm{C}\rangle\rangle\rangle \langle\uparrow \langle\updownarrow \mathrm{A}\rangle \langle\updownarrow \mathrm{T}\rangle\rangle\rangle\,.$$

Figure 4.9 contains the corresponding derivation tree. Indeed, the yield of the tree is equal to the DNA expression from Example 4.2. ■

Because the definition of G_1 closely follows the definition of DNA expressions, we have

Theorem 4.28 $\mathcal{L}(G_1) = \mathcal{L}_{G_1}(E_{\star,\star})$ *is the language* $\mathcal{D}$ *of all DNA expressions.*

and

Corollary 4.29 *The language $\mathcal{D}$ of DNA expressions is context-free.*

Theorem 4.28 may be viewed as an alternative definition of DNA expressions: a string is a DNA expression, if and only if it is an element of $\mathcal{L}(G_1)$. Unlike Definition 4.1, this alternative definition does not explicitly refer to the semantics. Therefore, it may be considered a 'cleaner' definition. Note, however, that the semantics is implicitly present in G_1, in the subscripts of the non-terminal symbols E, U and L.

4.6 The structure tree of a DNA expression

In Section 4.5, we have seen that we can represent a DNA expression by the corresponding derivation tree in a context-free grammar for $\mathcal{D}$. However, already for the small example DNA expression from Example 4.2, the resulting tree was large (see Figure 4.9).

We now introduce a more concise tree notation for DNA expressions, which, moreover, does not depend on a particular context-free grammar. The resulting trees are again ordered, rooted and node-labelled. After a description of the new tree notation, we will discuss its relation with the derivation trees from Section 4.5.

Let E be an arbitrary DNA expression. For an unambiguous definition of the tree corresponding to E, it is important to know what exactly are the arguments, and in particular the $\mathcal{N}$-word-arguments, of operators $\uparrow$ and $\downarrow$ occurring in E. In line with Theorem 4.9, we assume that each $\mathcal{N}$-word-argument of each operator occurring in E is a maximal $\mathcal{N}$-word occurrence. This is not really essential for the trees we want to describe, but it makes the description 'cleaner'.

We define *the structure tree of E* as follows. For each maximal $\mathcal{N}$-word occurrence α and for each operator occurring in E, we have a node, labelled by this $\mathcal{N}$-word or operator. Recall that there is a 1–1 correspondence between (occurrences of) DNA subexpressions and operators in E. Therefore, every node labelled by an operator corresponds to a DNA subexpression of E.

Of course, a node is something different than the label of that node. Much in the same way as that the occurrence of an operator in a DNA expression is something different that that operator itself. However, to keep the text more readable, we will sometimes say '$\mathcal{N}$-word' or 'operator' ('DNA subexpression') when we actually mean the corresponding node in the tree. This meaning will be clear from the context then.

In the structure tree we draw edges from operators to their arguments. By definition, these arguments are $\mathcal{N}$-words and DNA subexpressions of E, and by assumption, the $\mathcal{N}$-word-arguments are maximal $\mathcal{N}$-word occurrences in E. Indeed, for every such argument, there is a corresponding node. Hence, the edges are well defined.

Because every maximal $\mathcal{N}$-word occurrence and every proper DNA subexpression of E has exactly one parent operator, we indeed obtain a tree. The node labelled by the outermost operator of E is the root of the tree. It corresponds to the entire DNA expression.

Clearly, the node labelled by an operator is the parent of (the nodes corresponding to) its arguments. If an operator has two or more arguments, then its children in the structure tree are arranged from left to right in the same order as the corresponding arguments in the DNA expression.

The leaves of the tree are labelled by the maximal $\mathcal{N}$-word occurrences α of E, and the internal nodes by the operators. As an example, in Figure 4.10 we have drawn the structure tree of the DNA expression from Example 4.2.

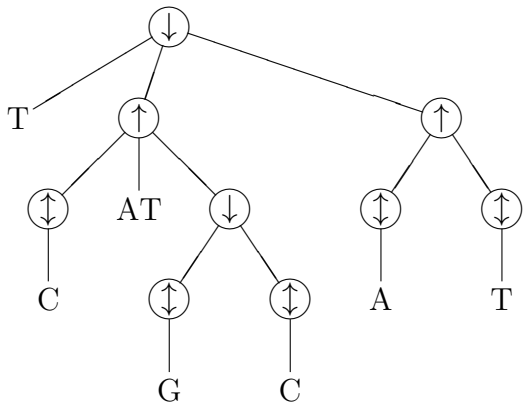


Figure 4.10: The structure tree of the DNA expression from Example 4.2.

We just recalled the correspondence between DNA subexpressions and operators. In fact, in the structure tree of a DNA expression E, a DNA subexpression of E is stored in the subtree rooted in its outermost operator.

There is a very close relation between the maximal nesting level of a DNA expression and the height of the corresponding structure tree:

Lemma 4.30 *Let E be a DNA expression, let l be the maximal nesting level of E, and let t be the structure tree of E. Then the height of t is $l + 1$.*

As we observed in Section 4.2, the maximal nesting level of the DNA expression from Example 4.2 is 4 (see Figure 4.3). Indeed, the height of the corresponding structure tree in Figure 4.10 is $4 + 1 = 5$.

Proof: Straightforward by induction on the number of operators occurring in E. In the induction step, we can apply Lemma 2.1(2) and Lemma 4.7(2). □

The transformation from DNA expressions to structure trees is injective. This means that when we are given a 'syntactically correct' ordered, rooted, node-labelled tree, we can perform the inverse transformation. The syntactic demands we impose on the trees are similar to those for a string over $\Sigma_{\mathcal{D}}$ to be a DNA expression (see Section 4.3):

- internal nodes are labelled by operators and leaves by $\mathcal{N}$-words
- a node labelled by $\updownarrow$ has only one child
- if a node labelled by $\uparrow$ or $\downarrow$ has two or more children, then the DNA subexpressions corresponding to these children fit together by upper strands or lower strands, respectively.

This final requirement is in fact a recursive one, as it presupposes that the subtrees rooted in the children can be interpreted as DNA expressions.

If this assumption is valid, then the prefitting requirement can be checked in a way similar to that for DNA expressions. Suppose that the i^{th} child of a node labelled by $\uparrow$ corresponds to an $\mathcal{N}$-word or a DNA expression ε_i, which has to prefit the $(i+1)^{\text{st}}$ child by upper strands. Then, e.g., $R(\mathcal{S}^+(\varepsilon_i))$ must be an element of $\mathcal{A}_\pm \cup \mathcal{A}_+$.

We can check this condition by walking the rightmost path in the subtree rooted in ε_i. This path ends in a certain $\mathcal{N}$-word α. If the parent of α is a node labelled by $\updownarrow$ or $\uparrow$, then $R(\mathcal{S}^+(\varepsilon_i))$ certainly belongs to $\mathcal{A}_\pm \cup \mathcal{A}_+$. If not (hence, if the parent of α is labelled

by $\downarrow$), then the composite symbol $R(\mathcal{S}^+(\varepsilon_i))$ cannot be an element of $\mathcal{A}_+$, as the lower part of the symbol is not equal to $-$. In order for $R(\mathcal{S}^+(\varepsilon_i))$ to be in $\mathcal{A}_\pm$, there has to be a node labelled by $\updownarrow$ on the path from ε_i down to α (including ε_i), and this is easy to verify.

If (and only if) a tree satisfies the three requirements mentioned, it is a structure tree and thus represents a DNA expression: *the DNA expression of the tree*.

In order to obtain this DNA expression, we have to perform a depth first search walk through the structure tree. In this walk, when entering an internal node X for the first time, we collect the opening bracket $\langle$ and the operator of the node. Next, we collect the argument(s) of the operator by recursively visiting the child(ren) of the node, and finally, when returning to X, we obtain the closing bracket $\rangle$. Apart from this closing bracket, the walk can be considered as a preorder walk.

Although the structure tree of a DNA expression is very different from (in particular, much smaller than) the derivation tree in the context-free grammar from Section 4.5, there is a natural relation between them. In four steps, removing more and more redundant nodes, we can transform the derivation tree of a DNA expression into the structure tree:

1. An internal node of a tree can have an arbitrary number $n \geq 1$ of children. Hence, all arguments of a DNA (sub-)expression can be connected directly to that DNA (sub-)expression. We do not need to recursively break down a sequence of arguments into one argument, possibly followed by another sequence of arguments. This way, we get rid of all nodes labelled by U or L with some combination of subscripts.

2. The label of a leaf in the tree may be an $\mathcal{N}$-word of arbitrary length $r \geq 1$. We do not need to recursively break down an $\mathcal{N}$-word-argument into one $\mathcal{N}$-letter, possibly followed by another $\mathcal{N}$-word. In particular, we get rid of all nodes labelled by α.

3. Occurrences of operators in a DNA expression correspond 1–1 to occurrences of DNA (sub-)expressions. We do not need two nodes for them: a node X for the DNA (sub-)expression and a node for the operator, which always is the second child of X. We can as well remove the second child of X and label X by the operator. This way, we get rid of all labels E with some combination of subscripts.

4. The scope of an operator is formed by the labels of its descendants in the tree. We do not need brackets to delimit the scope explicitly, and can therefore remove the nodes labelled by brackets.

The result of this four-step procedure is exactly the structure tree of a DNA expression as defined earlier in this section. As an illustration of the procedure, Figure 4.11 shows the intermediate trees for the DNA expression from Example 4.2.

4.7 Equivalent DNA expressions

Different DNA expressions may correspond to the same DNA molecule. For example, both $\langle\uparrow \alpha\rangle$ and $\langle\uparrow \langle\uparrow \alpha\rangle\rangle$ denote the formal DNA molecule $\binom{\alpha}{-}$. It is also possible that different DNA expressions denote 'almost the same' DNA molecule for a certain interpretation of 'almost the same'. To express these things formally, we define four binary relations on $\mathcal{D}$.

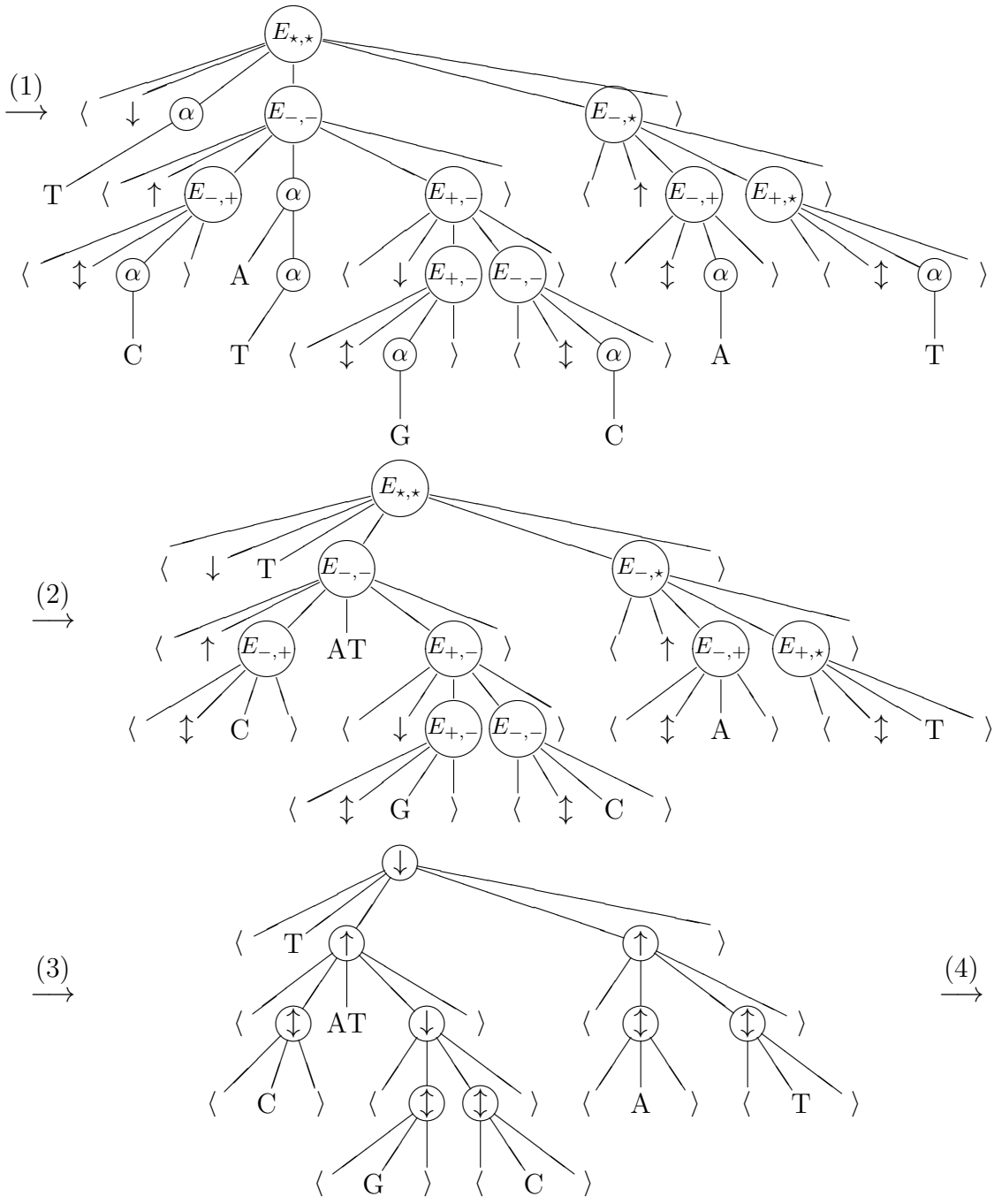


Figure 4.11: The derivation tree of the DNA expression from Example 4.2 is transformed into the structure tree. The original derivation tree is shown in Figure 4.9.
(1) All arguments of a DNA (sub-)expression are connected directly to that DNA (sub-)expression. (2) The $\mathcal{N}$-letters constituting an $\mathcal{N}$-word-argument are substituted by the $\mathcal{N}$-word. (3) The operators move up the tree. They take over the places (the nodes) of the DNA (sub-)expressions that they govern. (4) Brackets are removed. The resulting structure tree is the one from Figure 4.10.

Definition 4.31 *Two DNA expressions E_1 and E_2 are strictly equivalent, or equivalent for short, if $\mathcal{S}(E_1) = \mathcal{S}(E_2)$. We write $E_1 \equiv E_2$ then.*

Hence two DNA expressions are equivalent if they denote exactly the same DNA molecule.

A somewhat weaker version of this relation is

Definition 4.32 *Two DNA expressions E_1 and E_2 are equivalent modulo nicks, if $\nu(\mathcal{S}(E_1)) = \nu(\mathcal{S}(E_2))$. We write $E_1 \underset{\triangledown}{=} E_2$ then.*

Intuitively, E_1 and E_2 are equivalent modulo nicks, if they denote DNA molecules with the same nucleotides at the same positions; the DNA molecules may, however, have nicks at different positions. E_1 may have nicks not occurring in E_2 and/or the other way round. For example, if $\mathcal{S}(E_1) = \binom{\mathrm{A}}{\mathrm{T}}^{\triangledown}\binom{\mathrm{C}}{\mathrm{G}}\binom{\mathrm{A}}{\mathrm{T}}_{\triangle}\binom{\mathrm{T}}{\mathrm{A}}\binom{\mathrm{G}}{\mathrm{C}}$, and $\mathcal{S}(E_2) = \binom{\mathrm{A}}{\mathrm{T}}\binom{\mathrm{C}}{\mathrm{G}}_{\triangle}\binom{\mathrm{A}}{\mathrm{T}}\binom{\mathrm{T}}{\mathrm{A}}\binom{\mathrm{G}}{\mathrm{C}}$ then $E_1 \underset{\triangledown}{=} E_2$. [4]

We further define a variant of this last relation.

Definition 4.33 *A DNA expression E_1 is equivalent to a DNA expression E_2 pre-modulo nicks, if there are strings $X_1, \ldots, X_r$ with $r \geq 1$ over $\mathcal{A}_{\triangledown\triangle}$ and symbols $c_1, \ldots, c_{r-1} \in \{\triangledown, \triangle\}$ such that $\mathcal{S}(E_1) = X_1 c_1 \ldots c_{r-1} X_r$ and $\mathcal{S}(E_2) = X_1 \ldots X_r$. We write $E_1 \ {}_{\triangledown}{\equiv} E_2$ then.*

Hence, if $E_1 \ {}_{\triangledown}{\equiv} E_2$, then $E_1 \underset{\triangledown}{=} E_2$ with the restriction that the DNA molecule denoted by E_2 does not contain nicks not occurring in the DNA molecule denoted by E_1. For example, if $\mathcal{S}(E_1) = \binom{\mathrm{A}}{\mathrm{T}}^{\triangledown}\binom{\mathrm{C}}{\mathrm{G}}\binom{\mathrm{A}}{\mathrm{T}}_{\triangle}\binom{\mathrm{T}}{\mathrm{A}}\binom{\mathrm{G}}{\mathrm{C}}$, and $\mathcal{S}(E_2) = \binom{\mathrm{A}}{\mathrm{T}}\binom{\mathrm{C}}{\mathrm{G}}\binom{\mathrm{A}}{\mathrm{T}}_{\triangle}\binom{\mathrm{T}}{\mathrm{A}}\binom{\mathrm{G}}{\mathrm{C}}$ then $E_1 \ {}_{\triangledown}{\equiv} E_2$. On the other hand, if $\mathcal{S}(E_1)$ is as before and $\mathcal{S}(E_2) = \binom{\mathrm{A}}{\mathrm{T}}\binom{\mathrm{C}}{\mathrm{G}}_{\triangle}\binom{\mathrm{A}}{\mathrm{T}}\binom{\mathrm{T}}{\mathrm{A}}\binom{\mathrm{G}}{\mathrm{C}}$, then $E_1 \ {}_{\triangledown}{\not\equiv} E_2$.

If $E_1 \ {}_{\triangledown}{\equiv} E_2$, we may also write $E_2 \equiv_{\triangledown} E_1$ and say that E_2 is *equivalent post-modulo nicks* to E_1. Thus, the relations ${}_{\triangledown}{\equiv}$ and $\equiv_{\triangledown}$ are each other's inverses: ${}_{\triangledown}{\equiv} \ = \ \equiv_{\triangledown}{}^{-1}$.

It is easy to verify that each of the binary relations $\equiv$, $\underset{\triangledown}{=}$, ${}_{\triangledown}{\equiv}$ and $\equiv_{\triangledown}$ is reflexive and transitive. Further, $\equiv$ and $\underset{\triangledown}{=}$ are symmetric, so these relations are (indeed) equivalence relations.

The relations ${}_{\triangledown}{\equiv}$ and $\equiv_{\triangledown}$ are not symmetric. Hence, in spite of their names, they are no equivalence relations. At first glance, one might think that ${}_{\triangledown}{\equiv}$ and $\equiv_{\triangledown}$ are antisymmetric: 'if a formal DNA molecule $\mathcal{S}(E_1)$ has more nicks than another formal DNA molecule $\mathcal{S}(E_2)$, then certainly $\mathcal{S}(E_2)$ does not have more nicks than $\mathcal{S}(E_1)$'. However, if $E_1 \equiv E_2$, then both $E_1 \ {}_{\triangledown}{\equiv} E_2$ and $E_2 \ {}_{\triangledown}{\equiv} E_1$ (and analogously for $\equiv_{\triangledown}$). Since equivalent DNA expressions E_1 and E_2 are not necessarily the same, ${}_{\triangledown}{\equiv}$ and $\equiv_{\triangledown}$ are not antisymmetric. Consequently, they are no partial orders, either. We might, however, say that they are antisymmetric (and thus partial orders) *up to equivalence*.

It follows immediately from the definition that $E_1 \ {}_{\triangledown}{\equiv} E_2$ implies $E_1 \underset{\triangledown}{=} E_2$ and that $E_1 \equiv_{\triangledown} E_2$ implies $E_1 \underset{\triangledown}{=} E_2$, so ${}_{\triangledown}{\equiv}$ and $\equiv_{\triangledown}$ are refinements of $\underset{\triangledown}{=}$. On the other hand, the equivalence relation $\equiv$ is a refinement both of ${}_{\triangledown}{\equiv}$ and of $\equiv_{\triangledown}$ (and thus certainly of $\underset{\triangledown}{=}$).

We can combine the notions of transitivity and refinement. For example, if $E_1 \equiv E_2$ and $E_2 \equiv_{\triangledown} E_3$, then $E_1 \equiv_{\triangledown} E_3$. The following is also clear: if $E_1 \ {}_{\triangledown}{\equiv} E_2$ and $E_1 \equiv_{\triangledown} E_2$ then $E_1 \equiv E_2$. Thus, the equivalence relation $\equiv$ is the intersection of the relations ${}_{\triangledown}{\equiv}$ and $\equiv_{\triangledown}$. In other words:

[4] Actually, this example is not really appropriate. As we will see in Section 5.1, a DNA expression with the semantics attributed to E_1 does not exist. At the level of formal DNA molecules, however, this example is a good illustration of the notion of equivalence modulo nicks.

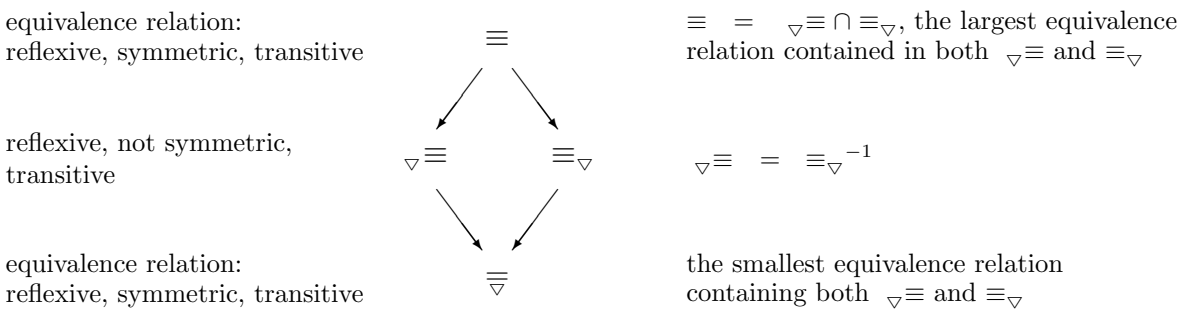


Figure 4.12: Properties of and relations between four binary relations on $\mathcal{D}$.

Lemma 4.34 *The relation $\equiv$ is the largest equivalence relation contained in both ${}_{\triangledown}\equiv$ and $\equiv_{\triangledown}$.*

On the other hand, we have

Lemma 4.35 *The relation $\fallingdotseq$ is the smallest equivalence relation containing both ${}_{\triangledown}\equiv$ and $\equiv_{\triangledown}$.*

Note that there is indeed a unique *smallest* equivalence relation containing both ${}_{\triangledown}\equiv$ and $\equiv_{\triangledown}$, namely the intersection of all equivalence relations containing ${}_{\triangledown}\equiv$ and $\equiv_{\triangledown}$.

Proof: Let R_0 be the smallest equivalence relation containing both ${}_{\triangledown}\equiv$ and $\equiv_{\triangledown}$. We just observed that $\fallingdotseq$ is an equivalence relation containing ${}_{\triangledown}\equiv$ and $\equiv_{\triangledown}$. Hence, R_0 must be a subset of $\fallingdotseq$.

Consider two arbitrary DNA expressions E_1 and E_2 such that $E_1 \fallingdotseq E_2$. By definition, $\nu(\mathcal{S}(E_1)) = \nu(\mathcal{S}(E_2))$, or, in words, E_1 and E_2 denote DNA molecules that have the same nucleotides at the same positions, but may have different nicks.

If we let $E_3 = \langle\downarrow \langle\uparrow E_1\rangle\rangle$, then $E_3 \in \mathcal{D}$ and $\mathcal{S}(E_3) = \nu^-(\nu^+(\mathcal{S}(E_1))) = \nu(\mathcal{S}(E_1))$ by (3.7). Thus, $\mathcal{S}(E_3) = \nu(\mathcal{S}(E_1)) = \nu(\mathcal{S}(E_2))$, or, in words, E_3 denotes a DNA molecule with the same nucleotides at the same positions as E_1 and E_2, but without nicks.

We have $E_1 \,{}_{\triangledown}\equiv\, E_3$ and $E_3 \equiv_{\triangledown} E_2$, implying $E_1 R_0 E_3$ and $E_3 R_0 E_2$. The transitivity of R_0 yields that also $E_1 R_0 E_2$. Because E_1 and E_2 were arbitrary DNA expressions with $E_1 \fallingdotseq E_2$, the equivalence relation $\fallingdotseq$ must be a subset of R_0.

Thus, we can conclude that R_0 is equal to $\fallingdotseq$. □

The results of this section are summarized in Figure 4.12.

Note that, because ${}_{\triangledown}\equiv$ and $\equiv_{\triangledown}$ are each other's inverses and symmetry is an inherent property of equivalence relations, every equivalence relation contained in ${}_{\triangledown}\equiv$ is also contained in $\equiv_{\triangledown}$, and vice versa. Similarly, every equivalence relation containing ${}_{\triangledown}\equiv$ also contains $\equiv_{\triangledown}$, and vice versa. Therefore, we may rephrase the two lemmas above as follows:

Lemma 4.34 The relation $\equiv$ is the largest equivalence relation contained in ${}_{\triangledown}\equiv$.

Lemma 4.35 The relation $\fallingdotseq$ is the smallest equivalence relation containing ${}_{\triangledown}\equiv$.

Of course, in the rephrased statements, we may as well replace the relation ${}_{\triangledown}\equiv$ by $\equiv_{\triangledown}$.

Chapter 5

Basic Results on DNA Expressions

In this chapter, we present some basic results on DNA expressions, which will be used in later chapters of this thesis. We first discuss which formal DNA molecules can be denoted by a DNA expression. After that, we consider two ways to decide if a DNA expression is nick free. Finally, we derive a number of results on equivalence (modulo nicks) between different DNA expressions.

5.1 Expressible formal DNA molecules

Many formal DNA molecules can be denoted by DNA expressions. We call such formal DNA molecules *expressible*. In particular, there exist DNA expressions which denote molecules with gaps and nicks. An example of this was the DNA expression in Example 4.2, which denotes the molecule from Figure 4.1(b), with two gaps and a nick.

Unfortunately, there also exist formal DNA molecules that are not expressible. We will see that the presence of nick letters in a formal DNA molecule determines whether or not it is expressible. We have a number of results concerning nicks in DNA expressions.

Lemma 5.1 *Let $E = \langle \uparrow \varepsilon_1 \dots \varepsilon_n \rangle$ for some $n \geq 1$ and $\mathcal{N}$-words and DNA expressions $\varepsilon_1, \dots, \varepsilon_n$ be an $\uparrow$-expression. Then*

1. *the upper strand of E is nick free;*
2. *the lower strand of E is nick free if and only if*
   (a) *for $i = 1, \dots, n$, the lower strand of $\mathcal{S}^+(\varepsilon_i)$ is nick free, and*
   (b) *for $i = 1, \dots, n-1$, either $R(\mathcal{S}^+(\varepsilon_i)) \in \mathcal{A}_+$ or $L(\mathcal{S}^+(\varepsilon_{i+1})) \in \mathcal{A}_+$ (or both).*

Proof: By definition,

$$\mathcal{S}(E) = \nu^+(\mathcal{S}^+(\varepsilon_1))y_1 \dots y_{n-1}\nu^+(\mathcal{S}^+(\varepsilon_n))$$

with the y_i's from (4.3).

1. Because the function ν^+ removes all upper nick letters from its arguments, and y_i is either $\triangle$ or λ (in particular, y_i is not an upper nick letter), the upper strand of $\mathcal{S}(E)$ is nick free.

2. $\Longrightarrow$ Assume that Condition 2(a) is not valid. Hence, for some i with $1 \leq i \leq n$, $\mathcal{S}^+(\varepsilon_i)$ contains a lower nick letter. Then also $\nu^+(\mathcal{S}^+(\varepsilon_i))$ contains a lower nick letter.

   Assume that Condition 2(b) is not valid. Hence, for some i with $1 \leq i \leq n-1$, both $R(\mathcal{S}^+(\varepsilon_i)) \in \mathcal{A}_\pm$ and $L(\mathcal{S}^+(\varepsilon_{i+1})) \in \mathcal{A}_\pm$. Then by definition, $y_i = \triangle$.

   In both cases, the lower strand of E is not nick free.

   $\Longleftarrow$ Assume that Conditions 2(a) and 2(b) hold for the arguments of E. Because $\mathcal{S}^+(\varepsilon_i)$ does not contain lower nick letters by Condition 2(a), and the function ν^+ certainly does not introduce lower nick letters, the lower strand of $\nu^+(\mathcal{S}^+(\varepsilon_i))$ is nick free for $i = 1, \ldots, n$. Further, Condition 2(b) ensures that for $i = 1, \ldots, n-1$, $y_i = \lambda$.

   As a result, the lower strand of $\mathcal{S}(E)$ is nick free.

□

In an analogous way we prove

Lemma 5.2 *Let $E = \langle \downarrow \varepsilon_1 \ldots \varepsilon_n \rangle$ for some $n \geq 1$ and $\mathcal{N}$-words and DNA expressions $\varepsilon_1, \ldots, \varepsilon_n$ be a $\downarrow$-expression. Then*

1. *the lower strand of E is nick free;*
2. *the upper strand of E is nick free if and only if*
   (a) *for $i = 1, \ldots, n$, the upper strand of $\mathcal{S}^-(\varepsilon_i)$ is nick free, and*
   (b) *for $i = 1, \ldots, n-1$, either $R(\mathcal{S}^-(\varepsilon_i)) \in \mathcal{A}_-$ or $L(\mathcal{S}^-(\varepsilon_{i+1})) \in \mathcal{A}_-$ (or both).*

We finally have

Lemma 5.3 *Let $E = \langle \updownarrow \varepsilon_1 \rangle$ for some $\mathcal{N}$-word or DNA expression ε_1 be an $\updownarrow$-expression. Then*

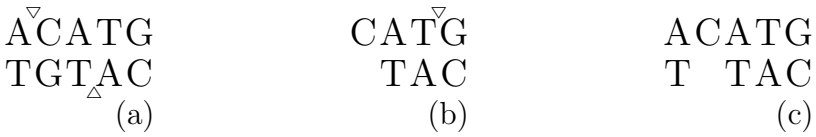
1. *the upper strand of E is nick free if and only if either ε_1 is an $\mathcal{N}$-word α or ε_1 is a DNA expression with a nick free upper strand;*
2. *the lower strand of E is nick free if and only if either ε_1 is an $\mathcal{N}$-word α or ε_1 is a DNA expression with a nick free lower strand.*

Proof: If ε_1 is an $\mathcal{N}$-word α, then $\mathcal{S}(E) = \binom{\alpha}{c(\alpha)}$ and E is nick free altogether.

If, on the other hand ε_1 is a DNA expression E_1, then $\mathcal{S}(E) = \kappa(\mathcal{S}(E_1))$. As the function κ does not introduce and does not repair nicks, the upper (or lower) strand of E is nick free if and only if the upper (lower, respectively) strand of E_1 is nick free. □

Lemmas 5.1 and 5.2 are useful for proving the following result:

Theorem 5.4 *Let E be an arbitrary DNA expression. Then either the upper strand or the lower strand (or both) of E is nick free.*



(a) (b) (c)

Figure 5.1: Three different types of DNA molecules. (a) A molecule which cannot be denoted by a DNA expression, because it has nicks in both strands. (b) A molecule which can be denoted by a DNA expression, because it only has a nick in the upper strand. (c) A molecule which can be denoted by a DNA expression, because it is nick free.

Hence, there do not exist DNA expressions denoting molecules with nicks in both strands.

Proof: If E is an $\uparrow$-expression (or a $\downarrow$-expression), then the claim follows from Lemma 5.1 (Lemma 5.2, respectively).

For $\updownarrow$-expressions $E = \langle \updownarrow \varepsilon_1 \rangle$, with ε_1 an $\mathcal{N}$-word or a DNA expression, we prove the claim by induction on the number p of operators occurring in E.

- If $p = 1$, then $E = \langle \updownarrow \alpha \rangle$ for an $\mathcal{N}$-word α and $\mathcal{S}(E) = \binom{\alpha}{c(\alpha)}$. Clearly, E is nick free altogether.
- Let $p \geq 1$, and suppose that the claim holds for all $\updownarrow$-expressions containing p operators (induction hypothesis). Then consider an arbitrary $\updownarrow$-expression $E = \langle \updownarrow \varepsilon_1 \rangle$ with $p + 1$ operators.

  As $p + 1 \geq 2$, ε_1 must be a DNA expression E_1 containing p operators. If E_1 is an $\uparrow$-expression or a $\downarrow$-expression, then, as we have just seen, at least one of the strands of E_1 is nick free. If, on the other hand, E_1 is an $\updownarrow$-expression, then we know by the induction hypothesis that at least one of the strands of E_1 is nick free.

  Because $\mathcal{S}(E) = \kappa(\mathcal{S}(E_1))$ and the function κ does not introduce (nor repair) nicks in its argument, the claim holds also for E.

□

Consequently, there is, e.g., no DNA expression for the molecule depicted in Figure 5.1(a).

Given Theorem 5.4, we may wonder if there are other limitations on the DNA molecules with gaps and nicks that can be expressed in $\mathcal{D}$. Does there exist a DNA expression for every DNA molecule with nicks in at most one strand? In Chapter 7, we will see that indeed there is. In particular, in Theorem 7.5 and Theorem 7.24, we describe constructions of DNA expressions denoting arbitrary nick free formal DNA molecules. In Theorem 7.46, we do the same for arbitrary formal DNA molecules containing lower nick letters (and no upper nick letters). By a result analogous to Theorem 7.46, we can also construct DNA expressions which denote formal DNA molecules containing upper nick letters (and no lower nick letters). We thus have

Theorem 5.5 *A formal DNA molecule X is expressible, if and only if X does not contain both upper nick letters and lower nick letters.*

Hence, some DNA molecules with nicks are expressible, whereas others are not. In Figure 5.1(b) and (c), we have depicted two DNA molecules that are expressible.

At a later stage, we will study DNA expressions denoting formal DNA molecules without single-stranded components. We can now give the following, general description of such molecules:

Corollary 5.6 *Let X be an expressible formal DNA molecule which does not contain any single-stranded component. Then there exist $\mathcal{N}$-words $\alpha_1, \ldots, \alpha_m$ for some $m \geq 1$, and a nick letter $y \in \{^{\triangledown}, _{\triangle}\}$, such that*

$$X = \binom{\alpha_1}{c(\alpha_1)} y \binom{\alpha_2}{c(\alpha_2)} y \ldots y \binom{\alpha_m}{c(\alpha_m)}.$$

Note that if X is nick free, then $m = 1$, $X = \binom{\alpha_1}{c(\alpha_1)}$ and the nick letter y occurring in the claim is irrelevant.

Proof: By Corollary 3.9(2), there exist $\mathcal{N}$-words $\alpha_1, \ldots, \alpha_m$ and nick letters $y_1, \ldots, y_{m-1}$ for some $m \geq 1$, such that

$$X = \binom{\alpha_1}{c(\alpha_1)} y_1 \binom{\alpha_2}{c(\alpha_2)} y_2 \ldots y_{m-1} \binom{\alpha_m}{c(\alpha_m)}.$$

By Theorem 5.4, the nick letters occurring in X must be all of the same type: either each y_j is an upper nick letter $^{\triangledown}$, or each y_j is a lower nick letter $_{\triangle}$. □

Because by definition, the semantics of an $\updownarrow$-expression is expressible and does not contain any single-stranded component, we have in particular

Corollary 5.7 *Let E be an $\updownarrow$-expression and let $X = \mathcal{S}(E)$. Then there exist $\mathcal{N}$-words $\alpha_1, \ldots, \alpha_m$ for some $m \geq 1$, and a nick letter $y \in \{^{\triangledown}, _{\triangle}\}$, such that*

$$X = \binom{\alpha_1}{c(\alpha_1)} y \binom{\alpha_2}{c(\alpha_2)} y \ldots y \binom{\alpha_m}{c(\alpha_m)}.$$

5.2 Nick free DNA expressions

There is a relatively simple algorithm to decide whether or not a DNA expression E contains nicks or not. This algorithm does not require the explicit computation of the semantics of a DNA expression. It consists only of the recursive application of the appropriate result from Lemma 5.1, Lemma 5.2 and Lemma 5.3, and, if necessary, Lemma 4.13. This takes time that is linear in the length $|E|$ of the DNA expression.

For certain DNA expressions, we do not even need this algorithm:

Lemma 5.8 *Let E be a DNA expression, and let $X = \mathcal{S}(E)$. If each occurrence of $\uparrow$ or $\downarrow$ in E is alternating, then X is nick free.*

Proof: Assume that each occurrence of $\uparrow$ or $\downarrow$ in E is alternating, i.e., that no occurrence of $\uparrow$ or $\downarrow$ in E has consecutive expression-arguments.

Lower nick letters can only be introduced into the semantics of a DNA expression by an occurrence of the operator $\uparrow$. Let $\langle \uparrow_1 \varepsilon_1 \ldots \varepsilon_n \rangle$ be an arbitrary $\uparrow$-subexpression of X, and for $i = 1, \ldots, n$, let $X_i = \mathcal{S}^+(\varepsilon_i)$. Consider any i with $1 \leq i \leq n-1$. By definition, $\uparrow_1$ introduces a lower nick letter between X_i and X_{i+1}, if and only if both $R(X_i) \in \mathcal{A}_{\pm}$ and $L(X_{i+1}) \in \mathcal{A}_{\pm}$. However, by assumption, either ε_i or ε_{i+1} is an $\mathcal{N}$-word. Without loss of generality, assume that ε_i is an $\mathcal{N}$-word α_i. Then $X_i = \mathcal{S}^+(\alpha_i) = \binom{\alpha_i}{-}$ and $R(X_i) \notin \mathcal{A}_{\pm}$. Consequently, $\uparrow_1$ does not introduce any lower nick letter into X.

Analogously, no occurrence of $\downarrow$ in E introduces an upper nick letter into the semantics. We conclude that X is nick free. □

Note that the above result cannot be reversed. If an occurrence of $\uparrow$ or $\downarrow$ in a DNA expression E is not alternating, then $\mathcal{S}(E)$ may be nick free after all.

Example 5.9 The DNA expression

$$E = \langle\uparrow \langle\updownarrow \mathrm{A}\rangle \langle\downarrow \langle\uparrow \mathrm{C} \langle\updownarrow \mathrm{AT}\rangle\rangle \langle\updownarrow \langle\downarrow \mathrm{C}\rangle\rangle\rangle\rangle \tag{5.1}$$

(depicted in Figure 5.1(c)), is nick free, even though both the first occurrence of $\uparrow$ and the first occurrence of $\downarrow$ have two consecutive expression-arguments. In fact, the $\downarrow$-subexpression

$$\langle\downarrow \langle\uparrow \mathrm{C} \langle\updownarrow \mathrm{AT}\rangle\rangle \langle\updownarrow \langle\downarrow \mathrm{C}\rangle\rangle\rangle \tag{5.2}$$

(depicted in Figure 5.1(b)) is not nick free, but the nick occurring in the upper strand is removed by the outermost operator $\uparrow$ of E. The outermost operator does not introduce new nicks. ■

5.3 Some equivalences

There are many general rules concerning equivalence between different DNA expressions. Some of them follow immediately from the definition of the semantics of a DNA expression. For example, for every $\mathcal{N}$-word α,

$$\langle\updownarrow \alpha\rangle \equiv \langle\updownarrow \langle\uparrow \alpha\rangle\rangle \equiv \langle\updownarrow \langle\downarrow c(\alpha)\rangle\rangle\,. \tag{5.3}$$

Another example is: for every DNA expression $\langle\uparrow \varepsilon_1 \dots \varepsilon_n\rangle$, where $n \geq 1$ and for $i = 1, \dots, n$, ε_i is an $\mathcal{N}$-word or a DNA expression,

$$\langle\uparrow \varepsilon_1 \dots \varepsilon_n\rangle \equiv \langle\uparrow E_1 E_2 \dots E_n\rangle\,, \tag{5.4}$$

where for $i = 1, \dots, n$, $E_i = \mathrm{Exp}^+(\varepsilon_i)$.

Other rules are intuitively clear, but a bit less easy to prove. To demonstrate how such rules are proved, we state one rule as a lemma here and give its formal proof.

Lemma 5.10 *Let $1 \leq i_0 \leq j_0 \leq n$, and let ε_i for $i = 1, \dots, n$ be an $\mathcal{N}$-word or a DNA expression. Then*

$$\langle\uparrow \varepsilon_1 \dots \varepsilon_{i_0-1} \langle\uparrow \varepsilon_{i_0} \dots \varepsilon_{j_0}\rangle \varepsilon_{j_0+1} \dots \varepsilon_n\rangle \equiv \langle\uparrow \varepsilon_1 \dots \varepsilon_n\rangle \tag{5.5}$$

if either the left-hand side or the right-hand side of the equivalence is a DNA expression.

Hence, all effects of the inner occurrence of $\uparrow$ in the left-hand side (i.e., creating upper $\mathcal{A}$-words, removing upper nick letters and joining the arguments) can also be achieved by the outermost occurrence of $\uparrow$.

Proof: For $i = 1, \dots, n$, let $E_i = \mathrm{Exp}^+(\varepsilon_i)$, and let $E_{i_0 j_0} = \langle\uparrow \varepsilon_{i_0} \dots \varepsilon_{j_0}\rangle$.

First, we need to prove that if either side of the equivalence in the claim is a DNA expression, then so is the other. If, e.g., the left-hand side is a DNA expression, then in particular $E_{i_0 j_0} = \langle\uparrow \varepsilon_{i_0} \dots \varepsilon_{j_0}\rangle$ is a DNA expression. This implies that $E_i \overline{\sqsubset} E_{i+1}$ ($i = i_0, \dots, j_0 - 1$). We further know that $E_i \overline{\sqsubset} E_{i+1}$ for $i = 1, \dots, i_0 - 2,\ j_0 + 1, \dots, n - 1$. Finally, we have $E_{i_0-1} \overline{\sqsubset} E_{i_0 j_0}$ and $E_{i_0 j_0} \overline{\sqsubset} E_{j_0+1}$.

The last two relations are equivalent to $R(\mathcal{S}(E_{i_0-1})), L(\mathcal{S}(E_{i_0 j_0})) \in \mathcal{A}_\pm \cup \mathcal{A}_+$ and to $R(\mathcal{S}(E_{i_0 j_0})), L(\mathcal{S}(E_{j_0+1})) \in \mathcal{A}_\pm \cup \mathcal{A}_+$, respectively. Now, by Lemma 4.13(3), $L(\mathcal{S}(E_{i_0 j_0})) = L(\mathcal{S}(E_{i_0}))$ and $R(\mathcal{S}(E_{i_0 j_0})) = R(\mathcal{S}(E_{j_0}))$. Hence, $L(\mathcal{S}(E_{i_0})), R(\mathcal{S}(E_{j_0})) \in \mathcal{A}_\pm \cup \mathcal{A}_+$. We already knew that $R(\mathcal{S}(E_{i_0-1})), L(\mathcal{S}(E_{j_0+1})) \in \mathcal{A}_\pm \cup \mathcal{A}_+$. Thus, $E_{i_0-1} \overline{\sqsubset} E_{i_0}$ and $E_{j_0} \overline{\sqsubset} E_{j_0+1}$.

We can conclude that $E_i \sqsubset\!\!\sqsupset E_{i+1}$ for $i = 1, 2, \ldots, n-1$, so that $\langle \uparrow \varepsilon_1 \ldots \varepsilon_n \rangle$ is a DNA expression. The proof in the other direction proceeds along the same lines.

Now, we can concentrate on the claim itself. By definition,

$$\mathcal{S}^+(E_{i_0 j_0}) = \mathcal{S}(E_{i_0 j_0}) = \nu^+(\mathcal{S}^+(\varepsilon_{i_0}))y_{i_0} \ldots y_{j_0-1}\nu^+(\mathcal{S}^+(\varepsilon_{j_0}))$$

and

$$\begin{aligned}&\mathcal{S}(\langle \uparrow \varepsilon_1 \ldots \varepsilon_{i_0-1} \langle \uparrow \varepsilon_{i_0} \ldots \varepsilon_{j_0} \rangle \varepsilon_{j_0+1} \ldots \varepsilon_n \rangle) = \\ &\quad \nu^+(\mathcal{S}^+(\varepsilon_1))y_1 \ldots y_{i_0-2}\nu^+(\mathcal{S}^+(\varepsilon_{i_0-1}))y_{i_0-1} \cdot \nu^+(\mathcal{S}^+(E_{i_0 j_0})) \cdot \\ &\quad y_{j_0}\nu^+(\mathcal{S}^+(\varepsilon_{j_0+1}))y_{j_0+1} \ldots y_{n-1}\nu^+(\mathcal{S}^+(\varepsilon_n)),\end{aligned} \tag{5.6}$$

where the y_i's are defined by

$$y_i = \begin{cases} \triangle & \text{if } E_i \sqsubseteq E_{i+1}, \text{ i.e., if both } R(\mathcal{S}(E_i)) \in \mathcal{A}_\pm \\ & \text{and } L(\mathcal{S}(E_{i+1})) \in \mathcal{A}_\pm \\ \lambda & \text{otherwise, i.e., if } R(\mathcal{S}(E_i)) \in \mathcal{A}_+ \\ & \text{or } L(\mathcal{S}(E_{i+1})) \in \mathcal{A}_+ \text{ (or both)} \end{cases} . \tag{5.7}$$

for $i = 1, \ldots, i_0 - 2, \quad i_0, \ldots, j_0 - 1, \quad j_0 + 1, \ldots, n-1,$

$$y_{i_0-1} = \begin{cases} \triangle & \text{if } E_{i_0-1} \sqsubseteq E_{i_0 j_0}, \text{ i.e., if both } R(\mathcal{S}(E_{i_0-1})) \in \mathcal{A}_\pm \\ & \text{and } L(\mathcal{S}(E_{i_0 j_0})) \in \mathcal{A}_\pm \\ \lambda & \text{otherwise, i.e., if } R(\mathcal{S}(E_{i_0-1})) \in \mathcal{A}_+ \\ & \text{or } L(\mathcal{S}(E_{i_0 j_0})) \in \mathcal{A}_+ \text{ (or both)} \end{cases}$$

$$y_{j_0} = \begin{cases} \triangle & \text{if } E_{i_0 j_0} \sqsubseteq E_{j_0+1}, \text{ i.e., if both } R(\mathcal{S}(E_{i_0 j_0})) \in \mathcal{A}_\pm \\ & \text{and } L(\mathcal{S}(E_{j_0+1})) \in \mathcal{A}_\pm \\ \lambda & \text{otherwise, i.e., if } R(\mathcal{S}(E_{i_0 j_0})) \in \mathcal{A}_+ \\ & \text{or } L(\mathcal{S}(E_{j_0+1})) \in \mathcal{A}_+ \text{ (or both)} \end{cases}$$

We already observed that $L(\mathcal{S}(E_{i_0 j_0})) = L(\mathcal{S}(E_{i_0}))$ and $R(\mathcal{S}(E_{i_0 j_0})) = R(\mathcal{S}(E_{j_0}))$. But then the definitions of y_{i_0-1} and y_{j_0} fit precisely into the general framework of definition (5.7). Hence, definition (5.7) is valid for $i = 1, \ldots, n-1$.

Now we will elaborate on the term $\nu^+(\mathcal{S}^+(E_{i_0 j_0}))$ occurring in (5.6). Because ν^+ is a homomorphism,

$$\begin{aligned}&\nu^+(\mathcal{S}^+(E_{i_0 j_0})) = \nu^+(\nu^+(\mathcal{S}^+(\varepsilon_{i_0}))y_{i_0} \ldots y_{j_0-1}\nu^+(\mathcal{S}^+(\varepsilon_{j_0}))) = \\ &\quad \nu^+(\nu^+(\mathcal{S}^+(\varepsilon_{i_0})))\nu^+(y_{i_0}) \ldots \nu^+(y_{j_0-1})\nu^+(\nu^+(\mathcal{S}^+(\varepsilon_{j_0})))\end{aligned} \tag{5.8}$$

For every i, y_i is either $\triangle$ or λ. Consequently, $\nu^+(y_i) = y_i$ for every i, and in particular for $i = i_0, \ldots j_0 - 1$. Combining this with Property (3.6), we can rewrite the result of (5.8) into

$$\nu^+(\mathcal{S}^+(\varepsilon_{i_0}))y_{i_0} \ldots y_{j_0-1}\nu^+(\mathcal{S}^+(\varepsilon_{j_0}))$$

We can substitute this into (5.6), which yields

$$\begin{aligned}&\mathcal{S}(\langle \uparrow \varepsilon_1 \ldots \varepsilon_{i_0-1} \langle \uparrow \varepsilon_{i_0} \ldots \varepsilon_{j_0} \rangle \varepsilon_{j_0+1} \ldots \varepsilon_n \rangle) = \\ &\nu^+(\mathcal{S}^+(\varepsilon_1))y_1 \ldots y_{i_0-2}\nu^+(\mathcal{S}^+(\varepsilon_{i_0-1}))y_{i_0-1} \quad \nu^+(\mathcal{S}^+(\varepsilon_{i_0}))y_{i_0} \ldots y_{j_0-1}\nu^+(\mathcal{S}^+(\varepsilon_{j_0})) \cdot \\ &\quad y_{j_0}\nu^+(\mathcal{S}^+(\varepsilon_{j_0+1}))y_{j_0+1} \ldots y_{n-1}\nu^+(\mathcal{S}^+(\varepsilon_n))\end{aligned}$$

with y_i's as in (5.7) for $i = 1, \ldots, n-1$. But this exactly equals $\mathcal{S}(\langle\uparrow \varepsilon_1 \ldots \varepsilon_n\rangle)$. □

In fact, (5.4) is a special case of Lemma 5.10. Another special case is

$$\langle\uparrow \langle\uparrow \varepsilon_1 \ldots \varepsilon_n\rangle\rangle \equiv \langle\uparrow \varepsilon_1 \ldots \varepsilon_n\rangle \tag{5.9}$$

if either side of the equivalence is a DNA expression. Under the same condition, we find

$$\langle\uparrow \langle\uparrow \varepsilon_1\rangle \langle\uparrow \varepsilon_2\rangle\rangle \equiv \langle\uparrow \varepsilon_1\varepsilon_2\rangle \tag{5.10}$$

by applying the lemma twice.

For every result on $\uparrow$-expressions there exists an analogous result for $\downarrow$-expressions (and vice versa). For example, the analogous version of Lemma 5.10 is

> *Let $1 \leq i_0 \leq j_0 \leq n$, and let ε_i for $i = 1, \ldots, n$ be an $\mathcal{N}$-word or a DNA expression. Then*
>
> $$\langle\downarrow \varepsilon_1 \ldots \varepsilon_{i_0-1} \langle\downarrow \varepsilon_{i_0} \ldots \varepsilon_{j_0}\rangle \varepsilon_{j_0+1} \ldots \varepsilon_n\rangle \equiv \langle\downarrow \varepsilon_1 \ldots \varepsilon_n\rangle$$
>
> *if either the left-hand side or the right-hand side of the equivalence is a DNA expression.*

Often, we will not formulate the analogous result explicitly. When we use a particular result, we may even refer to the version for $\uparrow$-expressions (if that is the one stated explicitly), while we actually need the version for $\downarrow$-expressions.

The analogue of (5.9) for $\updownarrow$-expressions is clear from the definition of the operator $\updownarrow$ (see Definition 4.1) and from Property (3.6):

$$\langle\updownarrow \langle\updownarrow \varepsilon\rangle\rangle \equiv \langle\updownarrow \varepsilon\rangle \tag{5.11}$$

for every $\mathcal{N}$-word or DNA expression ε.

We proceed with three results concerning the substitution of (occurrences of) $\mathcal{N}$-words or DNA subexpressions in a DNA expression by $\mathcal{N}$-words or DNA subexpressions which are equivalent ((pre/post-)modulo nicks).

Lemma 5.11 *Let E be a DNA expression and let E^s be (an occurrence of) a DNA subexpression in E. Let $E^{s'}$ be a DNA expression such that $E^s \underline{\overline{\triangledown}} E^{s'}$.*

When we substitute (the occurrence of) E^s in E by $E^{s'}$, the resulting string E' is again a DNA expression, and $E \underline{\overline{\triangledown}} E'$.

Proof: By induction on the number p of operators in E which are not in E^s.

- If $p = 0$, then $E = E^s$, and the claim is trivially valid.
- Let $p \geq 0$, and suppose that the claim holds for every DNA expression E and (occurrence of a) DNA subexpression E^s of E such that the number of operators in E which are not in E^s is at most p (induction hypothesis). Now let E be a DNA expression and let E^s be (an occurrence of) a DNA subexpression of E such that there are $p+1$ operators in E which are not in E^s.

  Because $p+1 \geq 1$, E^s is a proper DNA subexpression of E, and E^s is the immediate argument of a DNA subexpression $E^\sigma = \langle|_0 \varepsilon_1 \ldots \varepsilon_{i_0-1} E^s \varepsilon_{i_0+1} \ldots \varepsilon_n\rangle$ of E, for some operator $|_0$, i_0 and n with $1 \leq i_0 \leq n$, and $\mathcal{N}$-words and DNA expressions $\varepsilon_1, \ldots, \varepsilon_n$.

Let us define $E^{\sigma\prime} = \langle|_0 \varepsilon_1 \ldots \varepsilon_{i_0-1} E^{s\prime} \varepsilon_{i_0+1} \ldots \varepsilon_n\rangle$. If $|_0 = \updownarrow$, then we must have $i_0 = n = 1$, and $E^{\sigma\prime}$ a valid $\updownarrow$-expression. Now, assume that $|_0$ is either $\uparrow$ or $\downarrow$. By Condition 2 of Definition 3.2, $L(\mathcal{S}(E^s))$, $R(\mathcal{S}(E^s))$, $L(\mathcal{S}(E^{s\prime}))$ and $R(\mathcal{S}(E^{s\prime}))$ are not nick letters, and thus $L(\mathcal{S}(E^s)) = L(\mathcal{S}(E^{s\prime}))$ and $R(\mathcal{S}(E^s)) = R(\mathcal{S}(E^{s\prime}))$. Consequently, the arguments of $E^{\sigma\prime}$ fit together just like those of E^σ, so that $E^{\sigma\prime}$ is a DNA expression. Now it follows from the definition of the semantics of a DNA expression that $E^\sigma \Eqv_\triangledown E^{\sigma\prime}$.

Substituting E^s in E by $E^{s\prime}$ produces the same overall string E' as substituting E^σ by $E^{\sigma\prime}$. Because the number of operators in E which are not in E^σ is at most p, it follows by induction that E' is a DNA expression satisfying $E \Eqv_\triangledown E'$.

□

It is easy to see that this result remains valid if we replace every occurrence of the relation $\Eqv_\triangledown$ by $\equiv$, ${}_\triangledown\!\equiv$ or $\equiv_\triangledown$.

Lemma 5.12 *Let E be a DNA expression and let ε be (an occurrence of) an $\mathcal{N}$-word or a proper DNA subexpression in E, such that the parent operator of ε is $\uparrow$. Let ε' be an $\mathcal{N}$-word or a DNA expression satisfying $Exp^+(\varepsilon) \Eqv_\triangledown Exp^+(\varepsilon')$.*

When we substitute (the occurrence of) ε in E by ε', the resulting string E' is again a DNA expression, and $E \Eqv_\triangledown E'$.

Proof: If both ε and ε' are DNA expressions, then we simply have a special case of Lemma 5.11.

If both ε and ε' are $\mathcal{N}$-words, then they must be equal, because in that case, $\mathrm{Exp}^+(\varepsilon) = \langle\uparrow \varepsilon\rangle$ and $\mathrm{Exp}^+(\varepsilon') = \langle\uparrow \varepsilon'\rangle$, which are assumed to be equivalent modulo nicks. Then also $E = E'$ and the claim follows immediately.

If ε is an $\mathcal{N}$-word and ε' is a DNA expression, then $\mathrm{Exp}^+(\varepsilon) = \langle\uparrow \varepsilon\rangle$ and $\mathrm{Exp}^+(\varepsilon') = \varepsilon'$. By assumption, $\langle\uparrow \varepsilon\rangle \Eqv_\triangledown \varepsilon'$. Let E^s be the DNA subexpression of E which ε is an immediate argument of: $E^s = \langle\uparrow \varepsilon_1 \ldots \varepsilon_{i_0-1} \varepsilon \varepsilon_{i_0+1} \ldots \varepsilon_n\rangle$ for some i_0 and n with $1 \le i_0 \le n$ and $\mathcal{N}$-words and DNA expressions $\varepsilon_1, \ldots, \varepsilon_{i_0-1}, \varepsilon_{i_0+1}, \ldots, \varepsilon_n$. Now, by Lemma 5.10, $E^s \equiv \langle\uparrow \varepsilon_1 \ldots \varepsilon_{i_0-1} \langle\uparrow \varepsilon\rangle \varepsilon_{i_0+1} \ldots \varepsilon_n\rangle$. Let us use $E^{s\prime}$ to denote the right-hand side of this equivalence.

By Lemma 5.11, we can replace E^s in E by $E^{s\prime}$ and the overall result E'' is a DNA expression equivalent to E. In E'' we can replace $\langle\uparrow \varepsilon\rangle$ by the DNA expression ε', and again by Lemma 5.11, the resulting overall string E' is a DNA expression satisfying $E'' \Eqv_\triangledown E'$. By the transitivity of the relation $\Eqv_\triangledown$, we also have $E \Eqv_\triangledown E'$.

For the case that ε is a DNA expression and ε' is an $\mathcal{N}$-word, the proof is analogous.

□

When we apply a special case of Lemma 5.12 n times, we obtain

Corollary 5.13 *Let $n \ge 1$, and let for $i = 1, \ldots, n$, ε_i and ε'_i be an $\mathcal{N}$-word or a DNA expression, $E_i = Exp^+(\varepsilon_i)$ and $E'_i = Exp^+(\varepsilon'_i)$.*

Then

if $E_i \Eqv_\triangledown E'_i$ for $i = 1, \ldots, n$, then $\langle\uparrow \varepsilon_1 \ldots \varepsilon_n\rangle \Eqv_\triangledown \langle\uparrow \varepsilon'_1 \ldots \varepsilon'_n\rangle$

if either of $\langle\uparrow \varepsilon_1 \ldots \varepsilon_n\rangle$ and $\langle\uparrow \varepsilon'_1 \ldots \varepsilon'_n\rangle$ is a DNA expression, i.e. if, e.g. $\varepsilon_i \sqsubset \varepsilon_{i+1}$ for $i = 1, \ldots, n-1$.

Both in Lemma 5.12 and in Corollary 5.13, we might also replace every occurrence of the relation $\overset{\triangledown}{\equiv}$ by $\equiv$, ${}_{\triangledown}\equiv$ or $\equiv_{\triangledown}$, and the operator $\uparrow$ by $\downarrow$ (in which case we must use the function Exp^- instead of Exp^+) or $\updownarrow$ (in which case n must be equal to 1 in Corollary 5.13).

Next, we give a number of results that deal with the exchange of outermost operators between a DNA expression and its argument(s). Such manipulations will be used to obtain a DNA expression with a specific structure. Again, we state (and prove) only one of two possible versions of each of the results. There exist analogous results in which every occurrence of the operator $\uparrow$ is replaced by $\downarrow$ and (if applicable) vice versa.

Lemma 5.14 *Let $E = \langle\updownarrow \langle\uparrow \varepsilon_1 \dots \varepsilon_n\rangle\rangle$ with $n \geq 1$ be an $\updownarrow$-expression, such that for $i = 1, \dots, n$, ε_i is a DNA expression (i.e., not an $\mathcal{N}$-word). Then $E \equiv_{\triangledown} \langle\uparrow \langle\updownarrow \varepsilon_1\rangle \dots \langle\updownarrow \varepsilon_n\rangle\rangle$.*

Note that the right-hand side of the equivalence in the claim is indeed a DNA expression. By Lemma 4.13(2), $L(\mathcal{S}(\langle\updownarrow \varepsilon_i\rangle)), R(\mathcal{S}(\langle\updownarrow \varepsilon_i\rangle)) \in \mathcal{A}_{\pm}$ for $i = 1, \dots, n$, and thus the arguments of the operator $\uparrow$ in the right-hand side fit together by upper strands.

If, for example, $n = 2$, and $\mathcal{S}(\varepsilon_1) = \binom{\mathrm{A}}{\mathrm{T}}^{\triangledown}\binom{\mathrm{C}}{\mathrm{G}}$ and $\mathcal{S}(\varepsilon_2) = \binom{\mathrm{A}}{-}\binom{\mathrm{T}}{\mathrm{A}}_{\triangle}\binom{\mathrm{G}}{\mathrm{C}}$, then $\mathcal{S}(\langle\updownarrow \langle\uparrow \varepsilon_1\varepsilon_2\rangle\rangle) = \binom{\mathrm{A}}{\mathrm{T}}\binom{\mathrm{C}}{\mathrm{G}}\binom{\mathrm{A}}{\mathrm{T}}\binom{\mathrm{T}}{\mathrm{A}}_{\triangle}\binom{\mathrm{G}}{\mathrm{C}}$, while $\mathcal{S}(\langle\uparrow \langle\updownarrow \varepsilon_1\rangle \langle\updownarrow \varepsilon_2\rangle\rangle) = \binom{\mathrm{A}}{\mathrm{T}}\binom{\mathrm{C}}{\mathrm{G}}_{\triangle}\binom{\mathrm{A}}{\mathrm{T}} \cdot \binom{\mathrm{T}}{\mathrm{A}}_{\triangle}\binom{\mathrm{G}}{\mathrm{C}}$.

Proof: By definition,

$$\mathcal{S}(E) = \mathcal{S}(\langle\updownarrow \langle\uparrow \varepsilon_1 \dots \varepsilon_n\rangle\rangle) = \kappa(\mathcal{S}(\langle\uparrow \varepsilon_1 \dots \varepsilon_n\rangle)) =$$
$$\kappa(\nu^+(\mathcal{S}^+(\varepsilon_1))y_1 \dots y_{n-1}\nu^+(\mathcal{S}^+(\varepsilon_n))) = \kappa(\nu^+(\mathcal{S}^+(\varepsilon_1)))y_1 \dots y_{n-1}\kappa(\nu^+(\mathcal{S}^+(\varepsilon_n))),$$

where for $i = 1, \dots, n-1$, $y_i \in \{\triangle, \lambda\}$, and the actual value of y_i depends on ε_i and ε_{i+1} (see (4.3)). Because ε_i is a DNA expression, $\mathcal{S}^+(\varepsilon_i) = \mathcal{S}(\varepsilon_i)$ for $i = 1, \dots, n$, and because of the commutativity of κ and ν^+ (see (3.5)), these functions may be interchanged. Hence, we get:

$$\mathcal{S}(E) = \nu^+(\kappa(\mathcal{S}(\varepsilon_1)))y_1 \dots y_{n-1}\nu^+(\kappa(\mathcal{S}(\varepsilon_n))).$$

On the other hand,

$$\mathcal{S}(\langle\uparrow \langle\updownarrow \varepsilon_1\rangle \dots \langle\updownarrow \varepsilon_n\rangle\rangle) = \nu^+(\mathcal{S}^+(\langle\updownarrow \varepsilon_1\rangle))y'_1 \dots y'_{n-1}\nu^+(\mathcal{S}^+(\langle\updownarrow \varepsilon_n\rangle)) =$$
$$\nu^+(\kappa(\mathcal{S}(\varepsilon_1)))y'_1 \dots y'_{n-1}\nu^+(\kappa(\mathcal{S}(\varepsilon_n))),$$

where, for $i = 1, \dots, n-1$, $y'_i \in \{\triangle, \lambda\}$, and the value of y'_i is determined by the arguments $\langle\updownarrow \varepsilon_i\rangle$ and $\langle\updownarrow \varepsilon_{i+1}\rangle$. However, by Lemma 4.13(2), $L(\mathcal{S}(\langle\updownarrow \varepsilon_i\rangle)), R(\mathcal{S}(\langle\updownarrow \varepsilon_i\rangle)) \in \mathcal{A}_{\pm}$ for $i = 1, \dots, n$, so that every y'_i is equal to $\triangle$.

Consequently, $E \equiv_{\triangledown} \langle\uparrow \langle\updownarrow \varepsilon_1\rangle \dots \langle\updownarrow \varepsilon_n\rangle\rangle$. □

Lemma 5.14 cannot always be reversed. For example, if we have a DNA expression $\langle\uparrow \langle\updownarrow \varepsilon_1\rangle \dots \langle\updownarrow \varepsilon_n\rangle\rangle$, we do not a priori know that $\langle\updownarrow \langle\uparrow \varepsilon_1 \dots \varepsilon_n\rangle\rangle$ is a DNA expression, because the arguments $\varepsilon_1, \dots, \varepsilon_n$ of $\uparrow$ may not fit together by upper strands. Only if they do, we can say that $\langle\uparrow \langle\updownarrow \varepsilon_1\rangle \dots \langle\updownarrow \varepsilon_n\rangle\rangle \ {}_{\triangledown}\equiv \langle\updownarrow \langle\uparrow \varepsilon_1 \dots \varepsilon_n\rangle\rangle$.

For a variant of Lemma 5.14, we do not have to worry about syntactic constraints:

Corollary 5.15 *For all $\mathcal{N}$-words $\alpha_1, \dots, \alpha_n$ with $n \geq 1$, we have*

$$\langle\updownarrow \alpha_1 \dots \alpha_n\rangle \equiv_{\triangledown} \langle\uparrow \langle\updownarrow \alpha_1\rangle \dots \langle\updownarrow \alpha_n\rangle\rangle .$$

Note that the concatenation of $n \geq 1$ $\mathcal{N}$-words α_i is itself a (one) $\mathcal{N}$-word, so that the left-hand side of the claim is indeed a DNA expression.

Proof: We can rewrite $\langle \updownarrow \alpha_1 \dots \alpha_n \rangle$ as follows:

$$\langle \updownarrow \alpha_1 \dots \alpha_n \rangle \equiv \langle \updownarrow \langle \uparrow \alpha_1 \dots \alpha_n \rangle \rangle \equiv \langle \updownarrow \langle \uparrow \langle \uparrow \alpha_1 \rangle \dots \langle \uparrow \alpha_n \rangle \rangle \rangle \equiv_{\triangledown}$$

$$\langle \uparrow \langle \updownarrow \langle \uparrow \alpha_1 \rangle \rangle \dots \langle \updownarrow \langle \uparrow \alpha_n \rangle \rangle \rangle \equiv \langle \uparrow \langle \updownarrow \alpha_1 \rangle \dots \langle \updownarrow \alpha_n \rangle \rangle$$

The first and the last equivalence follow from (5.3), the second one from Lemma 5.10 and the third one from Lemma 5.14. □

Theorem 5.16 *Let $\varepsilon_1, \dots, \varepsilon_{n-1}, \varepsilon_{n,2}, \dots, \varepsilon_{n,m}$ with $n, m \geq 1$ be $\mathcal{N}$-words and DNA expressions, and let $E_{n,1}$ be a DNA expression, such that*

- $\mathcal{S}^+(\varepsilon_i) \overline{\sqsubset} \mathcal{S}^+(\varepsilon_{i+1})$ *for* $i = 1, \dots, n-2$,
- $\mathcal{S}^+(\varepsilon_{n-1}) \overline{\sqsubset} \mathcal{S}(E_{n,1})$,
- $\mathcal{S}(E_{n,1}) \sqsubseteq \mathcal{S}^-(\varepsilon_{n,2})$ *and*
- $\mathcal{S}^-(\varepsilon_{n,j}) \sqsubseteq \mathcal{S}^-(\varepsilon_{n,j+1})$ *for* $j = 2, \dots, m-1$.

Let $E = \langle \uparrow \varepsilon_1 \dots \varepsilon_{n-1} \langle \downarrow E_{n,1} \varepsilon_{n,2} \dots \varepsilon_{n,m} \rangle \rangle$ *and* $E' = \langle \downarrow \langle \uparrow \varepsilon_1 \dots \varepsilon_{n-1} E_{n,1} \rangle \varepsilon_{n,2} \dots \varepsilon_{n,m} \rangle$.

1. *The strings E and E' are DNA expressions satisfying $E \overline{\equiv}_{\triangledown} E'$.*
2. *Each occurrence of $\uparrow$ or $\downarrow$ in E is alternating, if and only if each occurrence of $\uparrow$ or $\downarrow$ in E' is alternating. In particular, in this case, both E and E' are nick free, and $E \equiv E'$.*

Note that the requirement that $E_{n,1}$ be a DNA expression (i.e., not an $\mathcal{N}$-word) is quite natural. If $n \geq 2$ (or $m \geq 2$), it simply *has* to be a DNA expression, in order for E (or E', respectively) to be a DNA expression. If $E_{n,1}$ were an $\mathcal{N}$-word $\alpha_{n,1}$ here, then the lower strand of $\langle \downarrow \alpha_{n,1} \varepsilon_{n,2} \dots \varepsilon_{n,m} \rangle$ would strictly cover the upper strand to the left, and thus ε_{n-1} and $\langle \downarrow \alpha_{n,1} \varepsilon_{n,2} \dots \varepsilon_{n,m} \rangle$ would not fit together by upper strands in E (and similarly for E' if $m \geq 2$).

What we actually do in Theorem 5.16, is moving the outermost operator $\downarrow$ of the last argument $\langle \downarrow E_{n,1} \varepsilon_{n,2} \dots \varepsilon_{n,m} \rangle$ of the DNA expression E to the left of the DNA expression. To ensure that the arguments of the two operators $\uparrow$ and $\downarrow$ still fit together by upper or lower strands, respectively, i.e., that the resulting string is still a DNA expression, we also have to shift one of the closing brackets.

For the structure tree of the DNA expression E, this action corresponds to a *rotation* to the left on the root of the tree. If we want to transform the structure tree of E' back into the structure tree of E, then we have to perform a rotation to the right on the root of the tree. This is depicted in Figure 5.2.

As an aside, we wish to mention that tree rotations are a well-known operation in computer science. Usually, they are performed in *binary* trees, i.e., trees in which each node has at most two children, see, e.g., [Cormen et al., 1990, Section 14.2]. In our case, the two main nodes involved in the rotation (the ones labelled by $\uparrow$ and $\downarrow$ in Figure 5.2) may have an arbitrary (positive) number of children. It is, however, important that the lower node of the two is either the first child or the last child of the other.

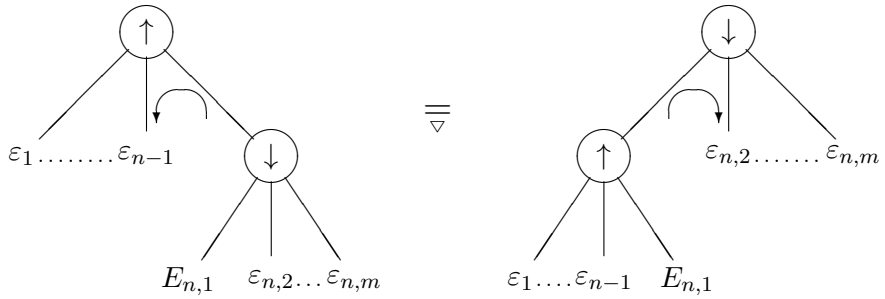


Figure 5.2: Analogue of Theorem 5.16(1) for structure trees of DNA expressions.

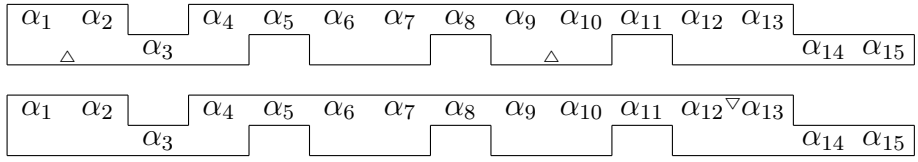


Figure 5.3: The two formal DNA molecules that occur in Example 5.17. (a) The molecule denoted by the DNA expression E from (5.12). (b) The molecule denoted by the DNA expression E' from (5.13).

Example 5.17 Let

$$
\begin{aligned}
E \;=\; \big\langle\uparrow \underbrace{\langle\updownarrow \alpha_1\rangle}_{\varepsilon_1} \underbrace{\langle\downarrow \langle\updownarrow \alpha_2\rangle\, \alpha_3\, \langle\uparrow \langle\updownarrow \alpha_4\rangle\rangle\rangle}_{\varepsilon_2} \underbrace{\alpha_5}_{\varepsilon_3} \underbrace{\langle\downarrow \langle\updownarrow \alpha_6\rangle\, \langle\updownarrow \alpha_7\rangle\rangle}_{\varepsilon_4} \\
\underbrace{\alpha_8}_{\varepsilon_5} \underbrace{\langle\uparrow \langle\updownarrow \alpha_9\rangle\, \langle\updownarrow \alpha_{10}\rangle\, \alpha_{11}\rangle}_{\varepsilon_6} \Big\langle \downarrow \underbrace{\langle\updownarrow \alpha_{12}\rangle}_{E_{7,1}} \underbrace{\langle\downarrow \langle\updownarrow \alpha_{13}\rangle\, \alpha_{14}\rangle}_{\varepsilon_{7,2}} \underbrace{\alpha_{15}}_{\varepsilon_{7,3}} \Big\rangle \,\big\rangle\,,
\end{aligned}
\tag{5.12}
$$

where $\alpha_1, \ldots, \alpha_{15}$ are arbitrary $\mathcal{N}$-words. In this case, $n = 7$ and $m = 3$. Indeed, the last argument of the $\uparrow$-expression E is a $\downarrow$-argument. We have depicted the formal DNA molecule denoted by E in Figure 5.3(a).

When we apply Theorem 5.16, we obtain

$$
\begin{aligned}
E' \;=\; \big\langle\downarrow\; \big\langle\uparrow\; \langle\updownarrow \alpha_1\rangle\, \langle\downarrow \langle\updownarrow \alpha_2\rangle\, \alpha_3\, \langle\uparrow \langle\updownarrow \alpha_4\rangle\rangle\rangle\, \alpha_5\, \langle\downarrow \langle\updownarrow \alpha_6\rangle\, \langle\updownarrow \alpha_7\rangle\rangle \\
\alpha_8\, \langle\uparrow \langle\updownarrow \alpha_9\rangle\, \langle\updownarrow \alpha_{10}\rangle\, \alpha_{11}\rangle\, \langle\updownarrow \alpha_{12}\rangle\; \big\rangle\, \langle\downarrow \langle\updownarrow \alpha_{13}\rangle\, \alpha_{14}\rangle\, \alpha_{15}\; \big\rangle\,.
\end{aligned}
\tag{5.13}
$$

We have depicted the formal DNA molecule denoted by E' in Figure 5.3(b). It is clear from the pictures that E and E' are equivalent modulo nicks. Both E and E' contain occurrences of $\uparrow$ and $\downarrow$ with consecutive expression-arguments. ■

Proof of Theorem 5.16:

1. By Definition 4.1 and Lemma 4.13, E and E' are indeed DNA expressions. Now by definition,

$$
\begin{aligned}
\mathcal{S}(E) \;=\; & \nu^+(\mathcal{S}^+(\varepsilon_1))y_1 \ldots y_{n-2}\nu^+(\mathcal{S}^+(\varepsilon_{n-1}))y_{n-1} \cdot \\
& \nu^+\Big(\nu^-(\mathcal{S}(E_{n,1}))y_{n,1}\nu^-(\mathcal{S}^-(\varepsilon_{n,2}))y_{n,2} \ldots y_{n,m-1}\nu^-(\mathcal{S}^-(\varepsilon_{n,m}))\Big)
\end{aligned}
$$

and

$$\begin{aligned}\mathcal{S}(E') &= \nu^-\Big(\nu^+(\mathcal{S}^+(\varepsilon_1))y_1 \ldots y_{n-2}\nu^+(\mathcal{S}^+(\varepsilon_{n-1}))y_{n-1}\nu^+(\mathcal{S}(E_{n,1}))\Big) \cdot \\ &\qquad y_{n,1}\nu^-(\mathcal{S}^-(\varepsilon_{n,2}))y_{n,2} \ldots y_{n,m-1}\nu^-(\mathcal{S}^-(\varepsilon_{n,m})),\end{aligned}$$

where the y_i's are either $_{\triangle}$ or λ and the $y_{n,j}$'s are either $^{\triangledown}$ or λ (depending on the formal DNA molecules preceding and succeeding them). It is not hard to see that each y_i in $\mathcal{S}(E)$ is equal to the corresponding y_i in $\mathcal{S}(E')$, and that the same property holds for each $y_{n,j}$.

When we observe that $\nu^+(^{\triangledown}) = \nu^-(_{\triangle}) = \lambda$ and that, by (3.7), for each $X \in \mathcal{A}^*_{\triangledown\triangle}$, $\nu^-(\nu^+(X)) = \nu^+(\nu^-(X)) = \nu(X)$, we can rewrite the expressions for $\mathcal{S}(E)$ and $\mathcal{S}(E')$ into:

$$\begin{aligned}\mathcal{S}(E) &= \nu^+(\mathcal{S}^+(\varepsilon_1))y_1 \ldots y_{n-2}\nu^+(\mathcal{S}^+(\varepsilon_{n-1}))y_{n-1} \cdot \\ &\qquad \nu(\mathcal{S}(E_{n,1}))\nu(\mathcal{S}^-(\varepsilon_{n,2})) \ldots \nu(\mathcal{S}^-(\varepsilon_{n,m}))\end{aligned}$$

and

$$\begin{aligned}\mathcal{S}(E') &= \nu(\mathcal{S}^+(\varepsilon_1)) \ldots \nu(\mathcal{S}^+(\varepsilon_{n-1}))\nu(\mathcal{S}(E_{n,1})) \cdot \\ &\qquad y_{n,1}\nu^-(\mathcal{S}^-(\varepsilon_{n,2}))y_{n,2} \ldots y_{n,m-1}\nu^-(\mathcal{S}^-(\varepsilon_{n,m})).\end{aligned}$$

Indeed, $S(E)$ and $S(E')$ can differ only in the occurrences of nicks. Hence, $E \underset{\triangledown}{=} E'$.

2. Assume that each occurrence of $\uparrow$ or $\downarrow$ in E is alternating, i.e., that for each occurrence of $\uparrow$ or $\downarrow$ in E, the arguments are $\mathcal{N}$-words and DNA expressions, alternately.

   Then in particular, the first $n-1$ arguments $\varepsilon_1, \ldots, \varepsilon_{n-1}$ of the outermost operator $\uparrow$ of E are $\mathcal{N}$-words and DNA expressions, alternately. Because the n^{th} argument is a $\downarrow$-expression, ε_{n-1} must be an $\mathcal{N}$-word (provided that $n \geq 2$).

   Now, let us consider the outermost operator $\downarrow$ of the last argument of E. Its last $m-1$ arguments $\varepsilon_{n,2}, \ldots, \varepsilon_{n,m}$ are $\mathcal{N}$-words and DNA expressions, alternately. Because the first argument of $\downarrow$ is the DNA expression $E_{n,1}$, $\varepsilon_{n,2}$ must be an $\mathcal{N}$-word (provided that $m \geq 2$).

   The above observations imply that in E', both the first occurrence of $\uparrow$ and the outermost operator $\downarrow$ are alternating.

   All other occurrences of $\uparrow$ and $\downarrow$ in E' occur inside an argument ε_i (with $i \leq n-1$), inside the argument $E_{n,1}$ or inside an argument $\varepsilon_{n,j}$ (with $j \geq 2$). These arguments already occurred in E. By assumption, the occurrences of $\uparrow$ or $\downarrow$ in them are alternating.

   By Claim 1, $E \underset{\triangledown}{=} E'$. By Lemma 5.8, however, both E and E' are nick free. This implies that E and E' are (strictly) equivalent: $E \equiv E'$.

   On the other hand, assume that each occurrence of $\uparrow$ or $\downarrow$ in E' is alternating. Then we can prove in an analogous way that this is also true for each occurrence of $\uparrow$ or $\downarrow$ in E. This implies that both E and E' are nick free, and thus that $E \equiv E'$.

□

For a special case we can combine Theorem 5.16(1) with Corollary 5.15:

Corollary 5.18 *Let $\varepsilon_1, \ldots, \varepsilon_{n-1}$ with $n \geq 1$ be $\mathcal{N}$-words and DNA expressions, and let $\alpha_{n,1}$ and $\alpha_{n,2}$ be $\mathcal{N}$-words, such that*

- $\mathcal{S}^+(\varepsilon_i)\overline{\sqsubset}\mathcal{S}^+(\varepsilon_{i+1})$ *for* $i = 1, \ldots, n-2$ *and*
- $\mathcal{S}^+(\varepsilon_{n-1})\overline{\sqsubset}\mathcal{S}(\langle\updownarrow \alpha_{n,1}\rangle)$.

The strings $E' = \langle\downarrow \langle\uparrow \varepsilon_1 \ldots \varepsilon_{n-1} \langle\updownarrow \alpha_{n,1}\rangle\rangle \langle\updownarrow \alpha_{n,2}\rangle\rangle$ and $E'' = \langle\uparrow \varepsilon_1 \ldots \varepsilon_{n-1} \langle\updownarrow \alpha_{n,1}\alpha_{n,2}\rangle\rangle$ are DNA expressions satisfying $E'\overline{\overline{\triangledown}}E''$.

Proof: By Theorem 5.16(1), E' and $E = \langle\uparrow \varepsilon_1 \ldots \varepsilon_{n-1} \langle\downarrow \langle\updownarrow \alpha_{n,1}\rangle \langle\updownarrow \alpha_{n,2}\rangle\rangle\rangle$ are DNA expressions for which $E'\overline{\overline{\triangledown}}E$. By Corollary 5.15, the DNA subexpression $E^s = \langle\downarrow \langle\updownarrow \alpha_{n,1}\rangle \langle\updownarrow \alpha_{n,2}\rangle\rangle$ of E satisfies $E^s {}_{\triangledown}\equiv \langle\updownarrow \alpha_{n,1}\alpha_{n,2}\rangle$. Consequently, by Lemma 5.11, also E'' is a DNA expression and $E {}_{\triangledown}\equiv E''$. By transitivity, $E'\overline{\overline{\triangledown}}E''$. □

By Theorem 5.16, we can manipulate an $\uparrow$-expression (or a $\downarrow$-expression) that has a $\downarrow$-expression (an $\uparrow$-expression, respectively) as its first or last argument. We now consider $\uparrow$-expressions with $\downarrow$-arguments that are not the first or last argument.

Theorem 5.19 *Let $E = \langle\uparrow \varepsilon_1 \ldots \varepsilon_n\rangle$ for some $n \geq 1$ and $\mathcal{N}$-words and DNA expressions $\varepsilon_1, \ldots, \varepsilon_n$ be a DNA expression. Let $\varepsilon_{i_1}, \ldots, \varepsilon_{i_r}$ for some $r \geq 1$ and $2 \leq i_1 < \ldots < i_r \leq n-1$ be $\downarrow$-arguments of E that have at least two arguments themselves. Hence, for $j = 1, \ldots, r$, $\varepsilon_{i_j} = \langle\downarrow \varepsilon_{i_j,1} \ldots \varepsilon_{i_j,m_j}\rangle$ for some $m_j \geq 2$ and $\mathcal{N}$-words and DNA expressions $\varepsilon_{i_j,1}, \ldots, \varepsilon_{i_j,m_j}$, and*

$$\begin{aligned} E = \langle\uparrow & \varepsilon_1 \ldots \varepsilon_{i_1-1} \langle\downarrow \varepsilon_{i_1,1}\varepsilon_{i_1,2} \ldots \varepsilon_{i_1,m_1-1}\varepsilon_{i_1,m_1}\rangle \varepsilon_{i_1+1} \ldots \varepsilon_{i_r-1} \\ & \langle\downarrow \varepsilon_{i_r,1}\varepsilon_{i_r,2} \ldots \varepsilon_{i_r,m_r-1}\varepsilon_{i_r,m_r}\rangle \varepsilon_{i_r+1} \ldots \varepsilon_n \rangle . \end{aligned}$$

1. *The string*

$$\begin{aligned} E' = \langle\downarrow & \langle\uparrow \varepsilon_1 \ldots \varepsilon_{i_1-1}\varepsilon_{i_1,1}\rangle \varepsilon_{i_1,2} \ldots \varepsilon_{i_1,m_1-1} \\ & \langle\uparrow \varepsilon_{i_1,m_1}\varepsilon_{i_1+1} \ldots\rangle \ldots \langle\uparrow \ldots \varepsilon_{i_r-1}\varepsilon_{i_r,1}\rangle \\ & \varepsilon_{i_r,2} \ldots \varepsilon_{i_r,m_r-1} \langle\uparrow \varepsilon_{i_r,m_r}\varepsilon_{i_r+1} \ldots \varepsilon_n\rangle \rangle \end{aligned}$$

   is a DNA expression satisfying $E\overline{\overline{\triangledown}}E'$.

2. *If each occurrence of $\uparrow$ or $\downarrow$ in E is alternating, then so is each occurrence of $\uparrow$ or $\downarrow$ in E'. In particular, in this case, both E and E' are nick free, and $E \equiv E'$.*

Note that in fact, we have $n \geq 3$, because we assume that $r \geq 1$ and $2 \leq i_1 \leq n-1$.

Note also that $\varepsilon_{i_1}, \ldots, \varepsilon_{i_r}$ are not necessarily *all* $\downarrow$-arguments ε_i of E with $2 \leq i \leq n-1$ and having at least two arguments themselves. There may be others, which we simply leave unchanged.

Note further that each of the 'new' $\uparrow$-arguments of E', i.e., each of $\langle\uparrow \varepsilon_1 \ldots \varepsilon_{i_1-1}\varepsilon_{i_1,1}\rangle$, $\langle\uparrow \varepsilon_{i_j,m_j}\varepsilon_{i_j+1} \ldots \varepsilon_{i_{j+1}-1}\varepsilon_{i_{j+1},1}\rangle$ for $j = 1, \ldots, r-1$, and $\langle\uparrow \varepsilon_{i_r,m_r}\varepsilon_{i_r+1} \ldots \varepsilon_n\rangle$, has at least two arguments itself.

In Figure 5.4, we have drawn the structure trees of the DNA expressions E and E'. They illustrate the essence of Theorem 5.19: the outermost operator $\uparrow$ of E (the label of the root of the structure tree of E) moves inwards: its function is taken over by $r+1$ inner occurrences of $\uparrow$ in E'. On the other hand, the operators $\downarrow$ from the $\downarrow$-arguments

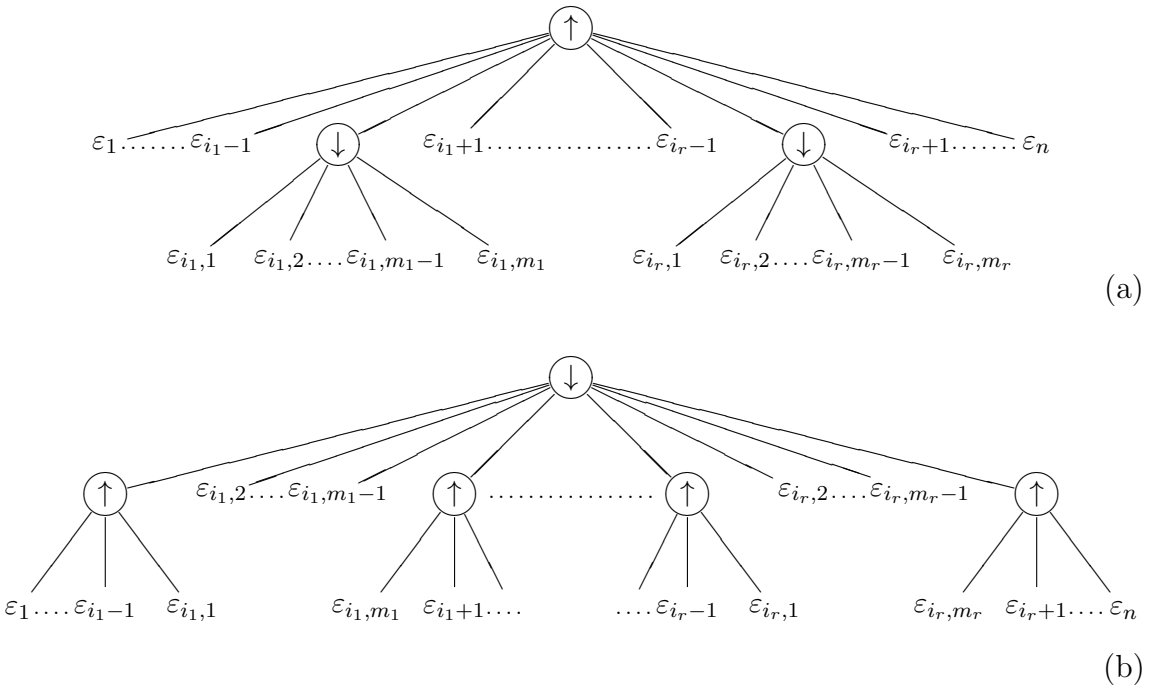


Figure 5.4: Analogue of Theorem 5.19 for structure trees of DNA expressions. (a) The structure tree of E. (b) The structure tree of E'.

$\varepsilon_{i_1}, \ldots, \varepsilon_{i_r}$ of E (the labels of certain children of the root) move outwards: their function is taken over by the outermost operator $\downarrow$ of E'.

Note that as a result, E' contains one operator more than E. The outermost operator $\uparrow$ of E and r occurrences of $\downarrow$ in E have been replaced by the outermost operator $\downarrow$ of E' and $r+1$ occurrences of $\uparrow$.

There is an easy way to deal with $\downarrow$-arguments with only one argument. Let ε_i with $2 \leq i \leq n-1$ be such a $\downarrow$-argument. Because the arguments of the $\uparrow$-expression E must fit together by upper strands, the argument of ε_i cannot be an $\mathcal{N}$-word. Hence, $\varepsilon_i = \langle\downarrow E_i\rangle$ for a DNA expression E_i. The only effect of $\downarrow$ on $\mathcal{S}(E_i)$ is that it removes the lower nick letters occurring in it (if any). Consequently, $\mathcal{S}(\varepsilon_i) = \mathcal{S}(\langle\downarrow E_i\rangle) \equiv_\triangledown \mathcal{S}(E_i)$. Now, by Lemma 5.11, when we replace $\varepsilon_i = \langle\downarrow E_i\rangle$ in E by E_i, the resulting string is a DNA expression E' satisfying $E \equiv_\triangledown E'$.

It is interesting to consider two special cases of Theorem 5.19. If a $\downarrow$-argument ε_{i_j} of E has exactly two arguments, hence $\varepsilon_{i_j} = \big\langle\downarrow \varepsilon_{i_j,1}\varepsilon_{i_j,2}\big\rangle$, then the resulting DNA expression E' has two consecutive $\uparrow$-arguments: $\big\langle\uparrow \ldots \varepsilon_{i_j,1}\big\rangle$ and $\big\langle\uparrow \varepsilon_{i_j,2} \ldots\big\rangle$. Conversely, if two of the $\downarrow$-arguments ε_{i_j} and ε_{i_k} of E are consecutive, say $i_k = i_j + 1$, then E' has an $\uparrow$-argument with exactly two arguments: $\big\langle\uparrow \varepsilon_{i_j,m_j}\varepsilon_{i_k,1}\big\rangle$.

Example 5.20 We again consider the $\uparrow$-expression E from (5.12). This time, however, we focus on the first two $\downarrow$-arguments:

$$\begin{aligned} E \;=\; \big\langle\uparrow\; & \underbrace{\langle\updownarrow \alpha_1\rangle}_{\varepsilon_1}\; \underbrace{\langle\downarrow \langle\updownarrow \alpha_2\rangle\, \alpha_3\, \langle\uparrow \langle\updownarrow \alpha_4\rangle\rangle\rangle}_{\varepsilon_{i_1}=\varepsilon_2}\; \underbrace{\alpha_5}_{\varepsilon_3}\; \underbrace{\langle\downarrow \langle\updownarrow \alpha_6\rangle\, \langle\updownarrow \alpha_7\rangle\rangle}_{\varepsilon_{i_2}=\varepsilon_4} \\ & \underbrace{\alpha_8}_{\varepsilon_5}\; \underbrace{\langle\uparrow \langle\updownarrow \alpha_9\rangle\, \langle\updownarrow \alpha_{10}\rangle\, \alpha_{11}\rangle}_{\varepsilon_6}\; \underbrace{\langle\downarrow \langle\updownarrow \alpha_{12}\rangle\, \langle\downarrow \langle\updownarrow \alpha_{13}\rangle\, \alpha_{14}\rangle\, \alpha_{15}\rangle}_{\varepsilon_7}\; \big\rangle\,, \end{aligned} \tag{5.14}$$

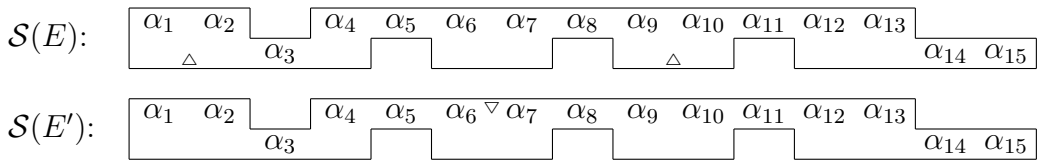


Figure 5.5: The two formal DNA molecules that occur in Example 5.20. (a) The molecule denoted by the DNA expression E from (5.14). (b) The molecule denoted by the DNA expression E' from (5.15).

where $\alpha_1, \ldots, \alpha_{15}$ are arbitrary $\mathcal{N}$-words. In this case, $n = 7$ and $r = 2$. Indeed, ε_{i_1} and ε_{i_2} are $\downarrow$-arguments with $2 \leq i_1 = 2 < i_2 = 4 \leq n-1 = 6$ that have at least two arguments themselves. Actually, ε_{i_1} and ε_{i_2} are *all* $\downarrow$-arguments of E that satisfy these requirements. E has another $\downarrow$-argument, ε_7, but that cannot be considered as an ε_{i_j}, because it is the last argument of E. In fact, we already dealt with ε_7, when we considered E for the first time, in Example 5.17. In Figure 5.5(a), we have again depicted the formal DNA molecule denoted by E.

When we apply Theorem 5.19, we obtain

$$\begin{array}{rcl} E' & = & \big\langle\downarrow \langle\uparrow \langle\updownarrow \alpha_1\rangle \langle\updownarrow \alpha_2\rangle\rangle \ \alpha_3 \ \langle\uparrow \langle\uparrow \langle\updownarrow \alpha_4\rangle\rangle \alpha_5 \langle\updownarrow \alpha_6\rangle\rangle \\ & & \quad \langle\uparrow \langle\updownarrow \alpha_7\rangle \alpha_8 \langle\uparrow \langle\updownarrow \alpha_9\rangle \langle\updownarrow \alpha_{10}\rangle \alpha_{11}\rangle \langle\downarrow \langle\updownarrow \alpha_{12}\rangle \langle\downarrow \langle\updownarrow \alpha_{13}\rangle \alpha_{14}\rangle \alpha_{15}\rangle\rangle \ \big\rangle \, . \end{array} \tag{5.15}$$

Figure 5.5(b) shows the formal DNA molecule denoted by E'.

It is clear from the pictures that E and E' are equivalent modulo nicks (see Claim 1). Also, the $\downarrow$-argument ε_{i_2} of E has exactly two arguments. Consequently, the second and the third $\uparrow$-argument of E' are consecutive arguments.

Claim 2 is not applicable, because four occurrences of the operators $\uparrow$ and $\downarrow$ in E have consecutive expression-arguments. ■

Proof of Theorem 5.19: Let us consider a $\downarrow$-argument ε_{i_j} with $1 \leq j \leq r$. By assumption, ε_{i_j} is neither the first argument, nor the last argument of the $\uparrow$-expression E. Hence, it must fit together by upper strands with the preceding argument ε_{i_j-1} and the succeeding argument ε_{i_j+1}. This implies that neither the first argument, nor the last argument of (the $\downarrow$-expression) ε_{i_j} can be an $\mathcal{N}$-word. Both $\varepsilon_{i_j,1}$ and ε_{i_j,m_j} are DNA expressions.

1. By induction on r, the number of $\downarrow$-arguments that we consider.

   - If $r = 1$, then we consider only one $\downarrow$-argument ε_{i_1}.
     As we have just observed, both the first argument $\varepsilon_{i_1,1}$ and the last argument ε_{i_1,m_1} of ε_{i_1} are DNA expressions. We now successively apply Lemma 5.10, Theorem 5.16(1) (together with Lemma 5.11) and once more Theorem 5.16(1):

$$\begin{array}{rcl} E & = & \langle\uparrow \varepsilon_1 \ldots \varepsilon_{i_1-1} \langle\downarrow \varepsilon_{i_1,1}\varepsilon_{i_1,2} \ldots \varepsilon_{i_1,m_1-1}\varepsilon_{i_1,m_1}\rangle \varepsilon_{i_1+1} \ldots \varepsilon_n\rangle \\ & \equiv & \langle\uparrow \varepsilon_1 \ldots \varepsilon_{i_1-1} \langle\uparrow \langle\downarrow \varepsilon_{i_1,1}\varepsilon_{i_1,2} \ldots \varepsilon_{i_1,m_1-1}\varepsilon_{i_1,m_1}\rangle \varepsilon_{i_1+1} \ldots \varepsilon_n\rangle \ \rangle \\ & \overset{\equiv}{\underset{\triangledown}{}} & \langle\uparrow \varepsilon_1 \ldots \varepsilon_{i_1-1} \langle\downarrow \varepsilon_{i_1,1}\varepsilon_{i_1,2} \ldots \varepsilon_{i_1,m_1-1} \langle\uparrow \varepsilon_{i_1,m_1}\varepsilon_{i_1+1} \ldots \varepsilon_n\rangle\rangle \ \rangle \\ & \overset{\equiv}{\underset{\triangledown}{}} & \langle\downarrow \langle\uparrow \varepsilon_1 \ldots \varepsilon_{i_1-1}\varepsilon_{i_1,1}\rangle \varepsilon_{i_1,2} \ldots \varepsilon_{i_1,m_1-1} \langle\uparrow \varepsilon_{i_1,m_1}\varepsilon_{i_1+1} \ldots \varepsilon_n\rangle \ \rangle \\ & = & E'. \end{array}$$

     Indeed, E' is a DNA expression satisfying $E \overset{\equiv}{\underset{\triangledown}{}} E'$.

- Let $\rho \geq 1$, and suppose that the claim holds for all $\uparrow$-expressions $E = \langle\uparrow \varepsilon_1 \ldots \varepsilon_n\rangle$ and $\downarrow$-arguments $\varepsilon_{i_1}, \ldots, \varepsilon_{i_r}$ of E, for which $1 \leq r \leq \rho$, $2 \leq i_1 < \ldots < i_r \leq n-1$ and each ε_{i_j} has at least two arguments (induction hypothesis).

  Now, assume that $r = \rho + 1$. Hence,

$$\begin{aligned} E &= \big\langle\uparrow \varepsilon_1 \ldots \varepsilon_{i_1-1} \big\langle\downarrow \varepsilon_{i_1,1}\varepsilon_{i_1,2} \ldots \varepsilon_{i_1,m_1-1}\varepsilon_{i_1,m_1}\big\rangle \varepsilon_{i_1+1} \ldots \varepsilon_{i_\rho-1} \\ &\quad \big\langle\downarrow \varepsilon_{i_\rho,1}\varepsilon_{i_\rho,2} \ldots \varepsilon_{i_\rho,m_\rho-1}\varepsilon_{i_\rho,m_\rho}\big\rangle \varepsilon_{i_\rho+1} \ldots \varepsilon_{i_{\rho+1}-1} \\ &\quad \big\langle\downarrow \varepsilon_{i_{\rho+1},1}\varepsilon_{i_{\rho+1},2} \ldots \varepsilon_{i_{\rho+1},m_{\rho+1}-1}\varepsilon_{i_{\rho+1},m_{\rho+1}}\big\rangle \varepsilon_{i_{\rho+1}+1} \ldots \varepsilon_n \big\rangle . \end{aligned}$$

  Recall that the ε_{i_j}'s occurring in the claim are not necessarily all $\downarrow$-arguments of E. We now simply ignore the first ρ ε_{i_j}'s. We thus view E as

$$\begin{aligned} E &= \big\langle\uparrow \varepsilon_1 \ldots \varepsilon_{i_1-1}\varepsilon_{i_1}\varepsilon_{i_1+1} \ldots \varepsilon_{i_\rho-1}\varepsilon_{i_\rho}\varepsilon_{i_\rho+1} \ldots \varepsilon_{i_{\rho+1}-1} \\ &\quad \big\langle\downarrow \varepsilon_{i_{\rho+1},1}\varepsilon_{i_{\rho+1},2} \ldots \varepsilon_{i_{\rho+1},m_{\rho+1}-1}\varepsilon_{i_{\rho+1},m_{\rho+1}}\big\rangle \varepsilon_{i_{\rho+1}+1} \ldots \varepsilon_n \big\rangle . \end{aligned}$$

  We apply the induction hypothesis to E and the $\downarrow$-argument $\varepsilon_{i_{\rho+1}}$:

$$\begin{aligned} E &\underset{\triangledown}{\equiv} \big\langle\downarrow \big\langle\uparrow \varepsilon_1 \ldots \varepsilon_{i_1-1}\varepsilon_{i_1}\varepsilon_{i_1+1} \ldots \varepsilon_{i_\rho-1}\varepsilon_{i_\rho}\varepsilon_{i_\rho+1} \ldots \varepsilon_{i_{\rho+1}-1}\varepsilon_{i_{\rho+1},1}\big\rangle \\ &\quad \varepsilon_{i_{\rho+1},2} \ldots \varepsilon_{i_{\rho+1},m_{\rho+1}-1} \big\langle\uparrow \varepsilon_{i_{\rho+1},m_{\rho+1}}\varepsilon_{i_{\rho+1}+1} \ldots \varepsilon_n\big\rangle \big\rangle \\ &= \big\langle\downarrow \big\langle\uparrow \varepsilon_1 \ldots \varepsilon_{i_1-1} \big\langle\downarrow \varepsilon_{i_1,1}\varepsilon_{i_1,2} \ldots \varepsilon_{i_1,m_1-1}\varepsilon_{i_1,m_1}\big\rangle \varepsilon_{i_1+1} \ldots \varepsilon_{i_\rho-1} \\ &\quad \big\langle\downarrow \varepsilon_{i_\rho,1}\varepsilon_{i_\rho,2} \ldots \varepsilon_{i_\rho,m_\rho-1}\varepsilon_{i_\rho,m_\rho}\big\rangle \varepsilon_{i_\rho+1} \ldots \varepsilon_{i_{\rho+1}-1}\varepsilon_{i_{\rho+1},1} \big\rangle \\ &\quad \varepsilon_{i_{\rho+1},2} \ldots \varepsilon_{i_{\rho+1},m_{\rho+1}-1} \big\langle\uparrow \varepsilon_{i_{\rho+1},m_{\rho+1}}\varepsilon_{i_{\rho+1}+1} \ldots \varepsilon_n\big\rangle \big\rangle . \end{aligned}$$

  Let us use E_1 to denote the first argument of the resulting $\downarrow$-expression. E_1 is an $\uparrow$-expression with (among others) $\downarrow$-arguments $\varepsilon_{i_1}, \ldots, \varepsilon_{i_\rho}$ with $2 \leq i_1 < \ldots < i_\rho$. Moreover, ε_{i_ρ} is not the last argument of E_1, because the last argument of E_1 is $\varepsilon_{i_{\rho+1},1}$. By assumption, each of the $\downarrow$-arguments $\varepsilon_{i_1}, \ldots, \varepsilon_{i_\rho}$ has at least two arguments.

  Hence, we can apply the induction hypothesis to E_1 and these $\downarrow$-arguments. When we combine this with Lemma 5.11 and subsequently use Lemma 5.10, we find

$$\begin{aligned} E &\underset{\triangledown}{\equiv} \big\langle\downarrow \big\langle\downarrow \big\langle\uparrow \varepsilon_1 \ldots \varepsilon_{i_1-1}\varepsilon_{i_1,1}\big\rangle \varepsilon_{i_1,2} \ldots \varepsilon_{i_1,m_1-1} \\ &\quad \big\langle\uparrow \varepsilon_{i_1,m_1}\varepsilon_{i_1+1} \ldots\big\rangle \ldots \big\langle\uparrow \ldots \varepsilon_{i_\rho-1}\varepsilon_{i_\rho,1}\big\rangle \\ &\quad \varepsilon_{i_\rho,2} \ldots \varepsilon_{i_\rho,m_\rho-1} \big\langle\uparrow \varepsilon_{i_\rho,m_\rho}\varepsilon_{i_\rho+1} \ldots \varepsilon_{i_{\rho+1}-1}\varepsilon_{i_{\rho+1},1}\big\rangle \big\rangle \\ &\quad \varepsilon_{i_{\rho+1},2} \ldots \varepsilon_{i_{\rho+1},m_{\rho+1}-1} \big\langle\uparrow \varepsilon_{i_{\rho+1},m_{\rho+1}}\varepsilon_{i_{\rho+1}+1} \ldots \varepsilon_n\big\rangle \big\rangle \\ &\equiv \big\langle\downarrow \big\langle\uparrow \varepsilon_1 \ldots \varepsilon_{i_1-1}\varepsilon_{i_1,1}\big\rangle \varepsilon_{i_1,2} \ldots \varepsilon_{i_1,m_1-1} \\ &\quad \big\langle\uparrow \varepsilon_{i_1,m_1}\varepsilon_{i_1+1} \ldots\big\rangle \ldots \big\langle\uparrow \ldots \varepsilon_{i_\rho-1}\varepsilon_{i_\rho,1}\big\rangle \\ &\quad \varepsilon_{i_\rho,2} \ldots \varepsilon_{i_\rho,m_\rho-1} \big\langle\uparrow \varepsilon_{i_\rho,m_\rho}\varepsilon_{i_\rho+1} \ldots \varepsilon_{i_{\rho+1}-1}\varepsilon_{i_{\rho+1},1}\big\rangle \\ &\quad \varepsilon_{i_{\rho+1},2} \ldots \varepsilon_{i_{\rho+1},m_{\rho+1}-1} \big\langle\uparrow \varepsilon_{i_{\rho+1},m_{\rho+1}}\varepsilon_{i_{\rho+1}+1} \ldots \varepsilon_n\big\rangle \big\rangle \\ &= E'. \end{aligned}$$

  We conclude again that E' is a DNA expression satisfying $E \underset{\triangledown}{\equiv} E'$.

2. In the inductive proof of the previous claim, we did not only use Theorem 5.16(1), but also Lemma 5.10 to rewrite E into E'. Consequently, in order to prove that each occurrence of $\uparrow$ or $\downarrow$ in E' is alternating, given that this is the case for E, it would not suffice to refer to Theorem 5.16(2). We would also need to consider the effects of Lemma 5.10. Instead of doing that, we give a direct proof, which resembles the proof of Theorem 5.16(2).

   Assume that each occurrence of $\uparrow$ or $\downarrow$ in E is alternating, i.e., that for each occurrence of $\uparrow$ or $\downarrow$ in E, the arguments are $\mathcal{N}$-words and DNA expressions, alternately.

We first examine the implications of this for the arguments of the outermost operator $\uparrow$ of E. For $j = 1, \ldots, r$, both ε_{i_j-1} and ε_{i_j+1} (the arguments preceding and succeeding the $\downarrow$-argument ε_{i_j}) must be $\mathcal{N}$-words. In particular, for $j = 1, \ldots, r-1$, there must be at least an $\mathcal{N}$-word ε_{i_j+1} which separates the $\downarrow$-arguments ε_{i_j} and $\varepsilon_{i_{j+1}}$.

Next, we consider a $\downarrow$-argument ε_{i_j} with $1 \leq j \leq r$. As we observed at the beginning of the proof, both the first argument $\varepsilon_{i_j,1}$ and the last argument ε_{i_j,m_j} of ε_{i_j} are DNA expressions. By assumption, ε_{i_j} has at least two arguments, and the arguments are $\mathcal{N}$-words and DNA expressions, alternately. Hence, ε_{i_j} has an odd number of arguments (at least three), and both $\varepsilon_{i_j,2}$ and ε_{i_j,m_j-1} are $\mathcal{N}$-words.

We now switch to E'. The arguments of the outermost operator $\downarrow$ of E' are an $\uparrow$-expression $\langle\uparrow \varepsilon_1 \ldots \varepsilon_{i_1-1}\varepsilon_{i_1,1}\rangle$, a sequence of arguments $\varepsilon_{i_1,2}, \ldots, \varepsilon_{i_1,m_1-1}$ coming from ε_{i_1}, another $\uparrow$-expression, again a sequence of arguments coming from an ε_{i_j}, and so on. By the above, the sequences of arguments coming from an ε_{i_j} are $\mathcal{N}$-words and DNA expressions alternately. Moreover, they start with the $\mathcal{N}$-word $\varepsilon_{i_j,2}$ and end with the $\mathcal{N}$-word ε_{i_j,m_j-1}. Consequently, the arguments of the outermost operator $\downarrow$ of E' are $\mathcal{N}$-words and DNA expressions, alternately.

Let E'_1 be the first $\uparrow$-argument $\langle\uparrow \varepsilon_1 \ldots \varepsilon_{i_1-1}\varepsilon_{i_1,1}\rangle$ of E'. The first $i_1 - 1$ arguments $\varepsilon_1, \ldots, \varepsilon_{i_1-1}$ of E'_1 were consecutive arguments of E. Hence, by assumption, they are $\mathcal{N}$-words and DNA expressions, alternately. Moreover, the last of these arguments, ε_{i_1-1}, is an $\mathcal{N}$-word, and $\varepsilon_{i_1,1}$ is a DNA expression. Consequently, the arguments of E'_1 are $\mathcal{N}$-words and DNA expressions, alternately.

Analogously, the arguments of the last $\uparrow$-argument $\langle\uparrow \varepsilon_{i_r,m_r}\varepsilon_{i_r+1} \ldots \varepsilon_n\rangle$ of E' are $\mathcal{N}$-words and DNA expressions, alternately. Finally, for $j = 1, \ldots, r-1$, the arguments of the $\uparrow$-argument $\langle\uparrow \varepsilon_{i_j,m_j}\varepsilon_{i_j+1} \ldots \varepsilon_{i_{j+1}-1}\varepsilon_{i_{j+1},1}\rangle$ of E' are the DNA expression ε_{i_j,m_j}, an alternating sequence of $\mathcal{N}$-words and DNA expressions $\varepsilon_{i_j+1}, \ldots, \varepsilon_{i_{j+1}-1}$ (which starts with the $\mathcal{N}$-word ε_{i_j+1} and ends with the $\mathcal{N}$-word $\varepsilon_{i_{j+1}-1}$), and the DNA expression $\varepsilon_{i_{j+1},1}$. Hence, also these arguments are $\mathcal{N}$-words and DNA expressions, alternately.

All other occurrences of $\uparrow$ and $\downarrow$ in E' occur inside an argument ε_i (with $i \neq i_j$ for all j's) or inside an argument $\varepsilon_{i_j,k}$. These ε_i's and $\varepsilon_{i_j,k}$'s already occurred in E. By assumption, each occurrence of $\uparrow$ or $\downarrow$ in them is alternating.

By Claim 1, $E \underset{\triangledown}{=} E'$. By Lemma 5.8, however, both E and E' are nick free. This implies that E and E' are (strictly) equivalent: $E \equiv E'$.

□

Theorem 5.19 can be reversed. That is, we can also start from E' and conclude that E is a DNA expression satisfying $E \underset{\triangledown}{=} E'$ (or even $E \equiv E'$):

Theorem 5.21 *Let $E' = \langle\downarrow \varepsilon_1 \ldots \varepsilon_n\rangle$ for some $n \geq 1$ and $\mathcal{N}$-words and DNA expressions $\varepsilon_1, \ldots, \varepsilon_n$ be a DNA expression. Let $\varepsilon_{i_1}, \ldots, \varepsilon_{i_r}, \varepsilon_{i_{r+1}}$ for some $r \geq 1$ and $1 = i_1 < \ldots < i_r < i_{r+1} = n$ be $\uparrow$-arguments of E' that have at least two arguments themselves. Hence, for $j = 1, \ldots, r, r+1$, $\varepsilon_{i_j} = \langle\uparrow \varepsilon_{i_j,1} \ldots \varepsilon_{i_j,m_j}\rangle$ for some $m_j \geq 2$ and $\mathcal{N}$-words and DNA*

expressions $\varepsilon_{i_j,1}, \ldots, \varepsilon_{i_j,m_j}$, *and*

$$E' = \big\langle\downarrow \langle\uparrow \varepsilon_{1,1} \ldots \varepsilon_{1,m_1-1}\varepsilon_{1,m_1}\rangle \varepsilon_2 \ldots \varepsilon_{i_2-1} \langle\uparrow \varepsilon_{i_2,1}\varepsilon_{i_2,2} \ldots \varepsilon_{i_2,m_2-1}\varepsilon_{i_2,m_2}\rangle \\ \varepsilon_{i_2+1} \ldots \varepsilon_{i_r-1} \langle\uparrow \varepsilon_{i_r,1}\varepsilon_{i_r,2} \ldots \varepsilon_{i_r,m_r-1}\varepsilon_{i_r,m_r}\rangle \\ \varepsilon_{i_r+1} \ldots \varepsilon_{n-1} \big\langle\uparrow \varepsilon_{n,1}\varepsilon_{n,2} \ldots \varepsilon_{n,m_{r+1}}\big\rangle \big\rangle .$$

1. *The string*

$$E = \big\langle\uparrow \varepsilon_{1,1} \ldots \varepsilon_{1,m_1-1} \langle\downarrow \varepsilon_{1,m_1}\varepsilon_2 \ldots \varepsilon_{i_2-1}\varepsilon_{i_2,1}\rangle \varepsilon_{i_2,2} \ldots \varepsilon_{i_2,m_2-1} \\ \langle\downarrow \varepsilon_{i_2,m_2}\varepsilon_{i_2+1} \ldots\rangle \ldots \langle\downarrow \ldots \varepsilon_{i_r-1}\varepsilon_{i_r,1}\rangle \varepsilon_{i_r,2} \ldots \varepsilon_{i_r,m_r-1} \\ \langle\downarrow \varepsilon_{i_r,m_r}\varepsilon_{i_r+1} \ldots \varepsilon_{n-1}\varepsilon_{n,1}\rangle \varepsilon_{n,2} \ldots \varepsilon_{n,m_{r+1}} \big\rangle$$

   is a DNA expression satisfying $E \underset{\triangledown}{\equiv} E'$.

2. *If each occurrence of* $\uparrow$ *or* $\downarrow$ *in* E' *is alternating, then so is each occurrence of* $\uparrow$ *or* $\downarrow$ *in* E. *In particular, in this case, both* E *and* E' *are nick free, and* $E \equiv E'$.

Note that in fact, we have $n \geq 2$, because we assume that $r \geq 1$ and $1 = i_1 < i_{r+1} = n$.

Proof:

1. We could prove this claim by induction, similar to the proof of Theorem 5.19(1). Instead, we give a proof that makes use of Theorem 5.19(1) itself.

   We first observe that both the last argument ε_{1,m_1} of ε_1 and the first argument $\varepsilon_{n,1}$ of ε_n must be DNA expressions. Otherwise, the arguments of E' would not fit together by lower strands. When we apply Theorem 5.16(1) two times (the second time in combination with Lemma 5.11) and subsequently apply Lemma 5.10, we find

$$\begin{aligned} E' &\underset{\triangledown}{\equiv} \big\langle\uparrow\ \varepsilon_{1,1} \ldots \varepsilon_{1,m_1-1} \big\langle\downarrow\ \varepsilon_{1,m_1}\varepsilon_2 \ldots \varepsilon_{i_2-1} \langle\uparrow \varepsilon_{i_2,1}\varepsilon_{i_2,2} \ldots \varepsilon_{i_2,m_2-1}\varepsilon_{i_2,m_2}\rangle \\ &\qquad \varepsilon_{i_2+1} \ldots \varepsilon_{i_r-1} \langle\uparrow \varepsilon_{i_r,1}\varepsilon_{i_r,2} \ldots \varepsilon_{i_r,m_r-1}\varepsilon_{i_r,m_r}\rangle \\ &\qquad \varepsilon_{i_r+1} \ldots \varepsilon_{n-1} \big\langle\uparrow \varepsilon_{n,1}\varepsilon_{n,2} \ldots \varepsilon_{n,m_{r+1}}\big\rangle \big\rangle \big\rangle \\ &\underset{\triangledown}{\equiv} \big\langle\uparrow\ \varepsilon_{1,1} \ldots \varepsilon_{1,m_1-1} \big\langle\uparrow\ \langle\downarrow \varepsilon_{1,m_1}\varepsilon_2 \ldots \varepsilon_{i_2-1} \langle\uparrow \varepsilon_{i_2,1}\varepsilon_{i_2,2} \ldots \varepsilon_{i_2,m_2-1}\varepsilon_{i_2,m_2}\rangle \\ &\qquad \varepsilon_{i_2+1} \ldots \varepsilon_{i_r-1} \langle\uparrow \varepsilon_{i_r,1}\varepsilon_{i_r,2} \ldots \varepsilon_{i_r,m_r-1}\varepsilon_{i_r,m_r}\rangle \\ &\qquad \varepsilon_{i_r+1} \ldots \varepsilon_{n-1}\varepsilon_{n,1} \big\rangle \varepsilon_{n,2} \ldots \varepsilon_{n,m_{r+1}} \big\rangle \big\rangle \\ &\equiv \big\langle\uparrow\ \varepsilon_{1,1} \ldots \varepsilon_{1,m_1-1} \langle\downarrow\ \varepsilon_{1,m_1}\varepsilon_2 \ldots \varepsilon_{i_2-1} \langle\uparrow \varepsilon_{i_2,1}\varepsilon_{i_2,2} \ldots \varepsilon_{i_2,m_2-1}\varepsilon_{i_2,m_2}\rangle \\ &\qquad \varepsilon_{i_2+1} \ldots \varepsilon_{i_r-1} \langle\uparrow \varepsilon_{i_r,1}\varepsilon_{i_r,2} \ldots \varepsilon_{i_r,m_r-1}\varepsilon_{i_r,m_r}\rangle \\ &\qquad \varepsilon_{i_r+1} \ldots \varepsilon_{n-1}\varepsilon_{n,1} \big\rangle \varepsilon_{n,2} \ldots \varepsilon_{n,m_{r+1}} \big\rangle . \end{aligned}$$

   Let us use E'' to denote the resulting DNA expression, and let us use E_1 to denote the $\downarrow$-argument

$$\langle\downarrow \varepsilon_{1,m_1}\varepsilon_2 \ldots \varepsilon_{i_2-1} \langle\uparrow \varepsilon_{i_2,1}\varepsilon_{i_2,2} \ldots \varepsilon_{i_2,m_2-1}\varepsilon_{i_2,m_2}\rangle \\ \varepsilon_{i_2+1} \ldots \varepsilon_{i_r-1} \langle\uparrow \varepsilon_{i_r,1}\varepsilon_{i_r,2} \ldots \varepsilon_{i_r,m_r-1}\varepsilon_{i_r,m_r}\rangle \\ \varepsilon_{i_r+1} \ldots \varepsilon_{n-1}\varepsilon_{n,1} \big\rangle$$

   of E''.

   If $r = 1$, then $i_2 = i_{r+1} = n$, E_1 reduces to $\langle\downarrow \varepsilon_{1,m_1}\varepsilon_2 \ldots \varepsilon_{n-1}\varepsilon_{n,1}\rangle$, and

$$E'' = \big\langle\uparrow \varepsilon_{1,1} \ldots \varepsilon_{1,m_1-1} \langle\downarrow \varepsilon_{1,m_1}\varepsilon_2 \ldots \varepsilon_{n-1}\varepsilon_{n,1}\rangle \varepsilon_{n,2} \ldots \varepsilon_{n,m_{r+1}}\big\rangle ,$$

which equals the string E from the claim. In this case, indeed, E is a DNA expression satisfying $E \underset{\triangledown}{\equiv} E'$.

If, on the other hand, $r \geq 2$, then E_1 has $r-1 \geq 1$ $\uparrow$-arguments $\langle\uparrow \varepsilon_{i_j,1}\varepsilon_{i_j,2}\ldots\varepsilon_{i_j,m_j-1}\ \varepsilon_{i_j,m_j}\rangle$ (with $2 \leq j \leq r$), each of which is neither the first argument, nor the last argument of E_1 and has at least two arguments itself. Hence, we can apply Theorem 5.19(1) (in combination with Lemma 5.11) to E_1 and subsequently apply Lemma 5.10:

$$\begin{aligned} E' \underset{\triangledown}{\equiv} \langle\uparrow \varepsilon_{1,1}\ldots\varepsilon_{1,m_1-1}\,\langle\uparrow\langle\downarrow \varepsilon_{1,m_1}\varepsilon_2\ldots\varepsilon_{i_2-1}\varepsilon_{i_2,1}\rangle\,\varepsilon_{i_2,2}\ldots\varepsilon_{i_2,m_2-1} \\ \langle\downarrow \varepsilon_{i_2,m_2}\varepsilon_{i_2+1}\ldots\rangle\ldots\langle\downarrow\ldots\varepsilon_{i_r-1}\varepsilon_{i_r,1}\rangle\,\varepsilon_{i_r,2}\ldots\varepsilon_{i_r,m_r-1} \\ \langle\downarrow \varepsilon_{i_r,m_r}\varepsilon_{i_r+1}\ldots\varepsilon_{n-1}\varepsilon_{n,1}\rangle\,\rangle\,\varepsilon_{n,2}\ldots\varepsilon_{n,m_{r+1}}\,\rangle \\ \equiv \langle\uparrow \varepsilon_{1,1}\ldots\varepsilon_{1,m_1-1}\,\langle\downarrow \varepsilon_{1,m_1}\varepsilon_2\ldots\varepsilon_{i_2-1}\varepsilon_{i_2,1}\rangle\,\varepsilon_{i_2,2}\ldots\varepsilon_{i_2,m_2-1} \\ \langle\downarrow \varepsilon_{i_2,m_2}\varepsilon_{i_2+1}\ldots\rangle\ldots\langle\downarrow\ldots\varepsilon_{i_r-1}\varepsilon_{i_r,1}\rangle\,\varepsilon_{i_r,2}\ldots\varepsilon_{i_r,m_r-1} \\ \langle\downarrow \varepsilon_{i_r,m_r}\varepsilon_{i_r+1}\ldots\varepsilon_{n-1}\varepsilon_{n,1}\rangle\,\varepsilon_{n,2}\ldots\varepsilon_{n,m_{r+1}}\,\rangle \\ = E. \end{aligned}$$

We conclude that also in this case, E is an $\uparrow$-expression satisfying $E \underset{\triangledown}{\equiv} E'$.

2. The proof of this claim is similar to that of Theorem 5.19(2). For each occurrence of $\uparrow$ or $\downarrow$ in E (whether it is the outermost operator $\uparrow$, or an operator $\downarrow$ governing a 'new' $\downarrow$-argument of E, or any other occurrence), we establish that its arguments are $\mathcal{N}$-words and DNA expressions, alternately, given that this is the case for each occurrence of $\uparrow$ or $\downarrow$ in E'. We leave the details to the reader.

□

Let $E = E(\alpha_1, \ldots, \alpha_k)$ for some $k \geq 1$ and $\mathcal{N}$-words $\alpha_1, \ldots, \alpha_k$ be an arbitrary DNA expression. We define the $\mathcal{N}$-word α_E as the concatenation of the $\mathcal{N}$-words $\alpha'_1, \ldots, \alpha'_k$, where

$$\alpha'_i = \begin{cases} \alpha_i & \text{if the parent operator of } \alpha_i \text{ in } E \text{ is } \updownarrow \text{ or } \uparrow \\ c(\alpha_i) & \text{if the parent operator of } \alpha_i \text{ in } E \text{ is } \downarrow \end{cases} \qquad (i = 1, \ldots, k).$$

For example, if $E = \langle\updownarrow \langle\uparrow \alpha_1 \langle\updownarrow \alpha_2\rangle \langle\downarrow \langle\updownarrow \alpha_3\rangle \alpha_4\rangle\rangle\rangle$, then $\alpha_E = \alpha_1\alpha_2\alpha_3 c(\alpha_4)$. The notation α_E is in particular useful, when E is an $\updownarrow$-expression or E is the argument of an $\updownarrow$-expression. This is the case in the final result of this section, which deals with $\updownarrow$-expressions.

Lemma 5.22 *Let $E = E(\alpha_1, \ldots, \alpha_k)$ for some $k \geq 1$ and $\mathcal{N}$-words $\alpha_1, \ldots, \alpha_k$ be an $\updownarrow$-expression. Then $E \underset{\triangledown}{\equiv} \langle\updownarrow \alpha_E\rangle$,*

Example 5.23 Consider $E = \langle\updownarrow \langle\uparrow \alpha_1 \langle\updownarrow \alpha_2\rangle \langle\downarrow \langle\updownarrow \alpha_3\rangle \alpha_4\rangle\rangle\rangle$, for which $\alpha_E = \alpha_1\alpha_2\alpha_3 c(\alpha_4)$. We have $\mathcal{S}(E) = \binom{\alpha_1\alpha_2}{c(\alpha_1\alpha_2)} \triangle \binom{\alpha_3 c(\alpha_4)}{c(\alpha_3)\alpha_4}$, whereas $\mathcal{S}(\langle\updownarrow \alpha_E\rangle) = \binom{\alpha_1\alpha_2\alpha_3 c(\alpha_4)}{c(\alpha_1\alpha_2\alpha_3)\alpha_4}$. Indeed, $E \underset{\triangledown}{\equiv} \langle\updownarrow \alpha_E\rangle$. ■

Proof of Lemma 5.22: By induction on the number p of operators occurring in E.

- If $p = 1$, then apparently $\updownarrow$ is the only operator in E, and its (only) argument must be an $\mathcal{N}$-word α_1: $E = \langle\updownarrow \alpha_1\rangle$. Then with $\alpha_E = \alpha_1$, we have $E = \langle\updownarrow \alpha_E\rangle$, so that certainly $E \underset{\triangledown}{\equiv} \langle\updownarrow \alpha_E\rangle$.

- If $p = 2$, then the argument of the outermost operator $\updownarrow$ of E is a DNA expression E_1: $E = \langle\updownarrow E_1\rangle$. E_1 contains only one operator and this operator can only have a maximal $\mathcal{N}$-word occurrence α_1 as its argument. There are three possibilities:
  - $E_1 = \langle\updownarrow \alpha_1\rangle$, but then, by (5.11), $E = \langle\updownarrow \langle\updownarrow \alpha_1\rangle\rangle \equiv \langle\updownarrow \alpha_1\rangle = \langle\updownarrow \alpha_E\rangle$ with $\alpha_E = \alpha_1$;
  - $E_1 = \langle\uparrow \alpha_1\rangle$, but then, by (5.3), $E = \langle\updownarrow \langle\uparrow \alpha_1\rangle\rangle \equiv \langle\updownarrow \alpha_1\rangle = \langle\updownarrow \alpha_E\rangle$ with $\alpha_E = \alpha_1$;
  - $E_1 = \langle\downarrow \alpha_1\rangle$, but then, by (5.3), $E = \langle\updownarrow \langle\downarrow \alpha_1\rangle\rangle \equiv \langle\updownarrow c(\alpha_1)\rangle = \langle\updownarrow \alpha_E\rangle$ with $\alpha_E = c(\alpha_1)$.
- Let $p \geq 2$, and suppose that the claim is valid for all $\updownarrow$-expressions containing at most p operators (induction hypothesis). Now let E be an arbitrary $\updownarrow$-expression with $p+1$ operators. $E = \langle\updownarrow E_1\rangle$ for a DNA expression E_1.

  Again we distinguish three cases:
  - E_1 is an $\updownarrow$-expression $\langle\updownarrow E_{1,1}\rangle$ for a DNA expression $E_{1,1}$. But then $E = \langle\updownarrow \langle\updownarrow E_{1,1}\rangle\rangle \equiv \langle\updownarrow E_{1,1}\rangle$ by equivalence (5.11). Obviously, the resulting DNA expression contains the same maximal $\mathcal{N}$-word occurrences α_i (and in the same order, with the same parent operators) as E. It contains, however, only p operators, and thus the claim follows from the induction hypothesis.
  - E_1 is an $\uparrow$-expression, so $E = \langle\updownarrow \langle\uparrow \varepsilon_1 \ldots \varepsilon_m\rangle\rangle$ for some $m \geq 1$ and $\mathcal{N}$-words and DNA expressions $\varepsilon_1, \ldots, \varepsilon_m$. For $j = 1, \ldots, m$, let
    $$\varepsilon'_j = \mathrm{Exp}^+(\varepsilon_j) = \begin{cases} \langle\uparrow \alpha\rangle & \text{if } \varepsilon_j \text{ is an } \mathcal{N}\text{-word } \alpha \\ \varepsilon_j & \text{if } \varepsilon_j \text{ is a DNA expression} \end{cases}.$$
    Then by equivalence (5.4) and Lemma 5.11,
    $$E = \langle\updownarrow \langle\uparrow \varepsilon_1 \ldots \varepsilon_m\rangle\rangle \equiv \langle\updownarrow \langle\uparrow \varepsilon'_1 \ldots \varepsilon'_m\rangle\rangle .$$
    Because every ε'_j is a DNA expression, we can apply Lemma 5.14:
    $$\langle\updownarrow \langle\uparrow \varepsilon'_1 \ldots \varepsilon'_m\rangle\rangle \equiv_{\triangledown} \langle\uparrow \langle\updownarrow \varepsilon'_1\rangle \ldots \langle\updownarrow \varepsilon'_m\rangle\rangle .$$
    Now consider an argument $\langle\updownarrow \varepsilon'_j\rangle$ with $1 \leq j \leq m$. If ε_j is an $\mathcal{N}$-word α, then $\varepsilon'_j = \langle\uparrow \alpha\rangle$ and $\langle\updownarrow \varepsilon'_j\rangle = \langle\updownarrow \langle\uparrow \alpha\rangle\rangle$, which contains $2 \leq p$ operators. If, on the other hand, ε_j is a DNA expression, then $\varepsilon'_j = \varepsilon_j$ and $\langle\updownarrow \varepsilon'_j\rangle = \langle\updownarrow \varepsilon_j\rangle$. This $\updownarrow$-expression contains at most p operators.

    In both cases, by the induction hypothesis, $\langle\updownarrow \varepsilon'_j\rangle \ {}_{\triangledown} \equiv \left\langle\updownarrow \alpha_{\varepsilon'_j}\right\rangle$. Now, by Lemma 5.11 and Corollary 5.15,
    $$\langle\uparrow \langle\updownarrow \varepsilon'_1\rangle \ldots \langle\updownarrow \varepsilon'_m\rangle\rangle \ {}_{\triangledown}\equiv \langle\uparrow \left\langle\updownarrow \alpha_{\varepsilon'_1}\right\rangle \ldots \left\langle\updownarrow \alpha_{\varepsilon'_m}\right\rangle\rangle \ {}_{\triangledown}\equiv \left\langle\updownarrow \alpha_{\varepsilon'_1} \ldots \alpha_{\varepsilon'_m}\right\rangle .$$
    Indeed, $\alpha_{\varepsilon'_1} \ldots \alpha_{\varepsilon'_m}$ is the concatenation of all maximal $\mathcal{N}$-word occurrences α_i (or the complement of α_i, if its parent operator is $\downarrow$) in E: $\alpha_{\varepsilon'_1} \ldots \alpha_{\varepsilon'_m} = \alpha_E$.
    When we combine all equivalences (pre-/post-modulo nicks), we conclude that $E \overline{\overline{\triangledown}} \langle\updownarrow \alpha_E\rangle$. Because the DNA expression $\langle\updownarrow \alpha_E\rangle$ is nick free, we even have $E \ {}_{\triangledown}\equiv \langle\updownarrow \alpha_E\rangle$.
  - E_1 is a $\downarrow$-expression. The proof for this case is completely analogous to that for the previous case.

□

Part II

Minimal DNA Expressions

Chapter 6

The Length of a DNA Expression

The complexity of an algorithm is often expressed as a function of the length of its input (see Section 2.1). Hence, when we want to analyse the complexity of algorithms that operate on DNA expressions, it is important to know the length of the DNA expressions at hand. Apart from this application, it is also *intrinsically* interesting to know how long a DNA expression denoting a certain formal DNA molecule may be. Therefore, in this and later chapters, we examine the length of a DNA expression.

We concentrate on *lower bounds* for the length of a DNA expression denoting a given (expressible) formal DNA molecule. Obviously, there does not exist an upper bound on the length of such a DNA expression. Indeed, consider an arbitrary DNA expression $E = \langle |_0 \varepsilon_1 \dots \varepsilon_n \rangle$, where $|_0$ is an operator, $n \geq 1$ and $\varepsilon_1, \dots, \varepsilon_n$ are the arguments of E. Then $E' = \langle |_0 E \rangle = \langle |_0 \langle |_0 \varepsilon_1 \dots \varepsilon_n \rangle \rangle$ is an equivalent DNA expression, for which $|E'| = |E| + 3$. This way, we can find arbitrarily long, equivalent DNA expressions.

In Section 6.1, we relate the length of a DNA expression to the number of operators occurring in it. After that, in Section 6.2, we focus on the semantics of a DNA expression, the formal DNA molecule. In particular, we identify specific (blocks of) components in a molecule, count these (blocks of) components and analyse the counting functions. We use the results of this to derive lower bounds for the length of a DNA expression with the desired semantics, in Section 6.3.

6.1 The operators in a DNA expression

Let X be a string over $\mathcal{A}_{\triangledown\triangle}$. We use $|X|_{\mathcal{A}}$ to denote the number of $\mathcal{A}$-letters occurring in X. One can easily verify that $|\cdot|_{\mathcal{A}}$ is a homomorphism from $\mathcal{A}^*_{\triangledown\triangle}$ to the non-negative integers. Obviously, if X is a nick free formal DNA molecule, then $|X|_{\mathcal{A}}$ equals the length of X. It is also easy to see that for an arbitrary formal DNA molecule X, $|X|_{\mathcal{A}}$ equals the length of $\nu(X)$. One may wonder why we introduce the new notation $|X|_{\mathcal{A}}$, while we could as well use the notation $|\nu(X)|$. The reason is that we often know that a certain formal DNA molecule X is nick free. In that case, it is useless to apply the function ν to X, whereas the notation $|\nu(X)|$ would suggest that we do that.

We make a basic observation.

Lemma 6.1 *Let E be a DNA expression denoting a formal DNA molecule X, and let p be the number of operators occurring in E. Then*

$$|E| = 3 \cdot p + |X|_{\mathcal{A}}.$$

Note that a DNA expression consists of operators and corresponding brackets on the one hand, and $\mathcal{N}$-letters on the other hand. Hence, Lemma 6.1 implies that $|X|_{\mathcal{A}}$ does not only count the number of $\mathcal{A}$-letters occurring in the formal DNA molecule X, but also the number of $\mathcal{N}$-letters occurring in *any* DNA expression E denoting X.

Proof: By induction on p.

- If $p = 1$, then E is $\langle\uparrow \alpha_1\rangle$, $\langle\downarrow \alpha_1\rangle$ or $\langle\updownarrow \alpha_1\rangle$ for an $\mathcal{N}$-word α_1. The corresponding formal DNA molecule X is $\binom{\alpha_1}{-}$, $\binom{-}{\alpha_1}$ or $\binom{\alpha_1}{c(\alpha_1)}$, respectively. In each of the cases,

  $$|E| = 3 + |\alpha_1| = 3 \cdot p + |X|_{\mathcal{A}}.$$

- Let $p \geq 1$, and suppose that the claim holds for all DNA expressions containing at most p operators (induction hypothesis). Now assume that E contains $p + 1$ operators. E is either an $\uparrow$-expression, or a $\downarrow$-expression or an $\updownarrow$-expression.

  – If E is an $\uparrow$-expression, hence $E = \langle\uparrow \varepsilon_1 \ldots \varepsilon_n\rangle$ for some $n \geq 1$ and $\mathcal{N}$-words and DNA expressions $\varepsilon_1, \ldots, \varepsilon_n$, then by definition,

  $$X = \mathcal{S}(E) = \nu^+(\mathcal{S}^+(\varepsilon_1))\ y_1\ \nu^+(\mathcal{S}^+(\varepsilon_2))\ y_2\ \ldots\ y_{n-1}\ \nu^+(\mathcal{S}^+(\varepsilon_n)),$$

  where the y_i's are $\triangle$ or λ (see (4.3)). The function ν^+ removes all upper nick letters occurring in its argument, but it does not affect the occurring $\mathcal{A}$-letters. Obviously, for $i = 1, \ldots, n-1$, $|y_i|_{\mathcal{A}} = 0$. This implies that

  $$\begin{aligned} |X|_{\mathcal{A}} &= |\nu^+(\mathcal{S}^+(\varepsilon_1))|_{\mathcal{A}} + |y_1|_{\mathcal{A}} + |\nu^+(\mathcal{S}^+(\varepsilon_2))|_{\mathcal{A}} + |y_2|_{\mathcal{A}} + \cdots + |y_{n-1}|_{\mathcal{A}} \\ &\quad + |\nu^+(\mathcal{S}^+(\varepsilon_n))|_{\mathcal{A}} \\ &= |\mathcal{S}^+(\varepsilon_1)|_{\mathcal{A}} + |\mathcal{S}^+(\varepsilon_2)|_{\mathcal{A}} + \cdots + |\mathcal{S}^+(\varepsilon_n)|_{\mathcal{A}}. \end{aligned}$$

  Apart from the outermost operator, all operators in E occur in the arguments $\varepsilon_1, \ldots, \varepsilon_n$. For $i = 1, \ldots, n$, let p_i be the number of operators occurring in ε_i. Then

  $$p_1 + p_2 + \cdots + p_n = p.$$

  If an argument ε_i is an $\mathcal{N}$-word α_i, then $\mathcal{S}^+(\varepsilon_i) = \binom{\alpha_i}{-}$, $p_i = 0$ and

  $$|\varepsilon_i| = |\alpha_i| = 3 \cdot p_i + |\mathcal{S}^+(\varepsilon_i)|_{\mathcal{A}}.$$

  If, on the other hand, an argument ε_i is a DNA expression, then $\mathcal{S}^+(\varepsilon_i) = \mathcal{S}(\varepsilon_i)$, $1 \leq p_i \leq p$ and by the induction hypothesis,

  $$|\varepsilon_i| = 3 \cdot p_i + |\mathcal{S}(\varepsilon_i)|_{\mathcal{A}} = 3 \cdot p_i + |\mathcal{S}^+(\varepsilon_i)|_{\mathcal{A}}.$$

  When we combine all equations, we obtain

  $$\begin{aligned} |E| &= 3 + |\varepsilon_1| + \cdots + |\varepsilon_n| \\ &= 3 + (3 \cdot p_1 + |\mathcal{S}^+(\varepsilon_1)|_{\mathcal{A}}) + \cdots + (3 \cdot p_n + |\mathcal{S}^+(\varepsilon_n)|_{\mathcal{A}}) \\ &= 3 \cdot (p + 1) + |X|_{\mathcal{A}}. \end{aligned}$$

– If E is a $\downarrow$-expression containing $p+1$ operators, then the proof is analogous.

– Finally, if E is an $\updownarrow$-expression, then $E = \langle \updownarrow E_1 \rangle$ for a DNA expression E_1 containing $p \geq 1$ operators. Hence,

$$X = \mathcal{S}(E) = \kappa(\mathcal{S}(E_1)).$$

Because the function κ does not change the *number* of $\mathcal{A}$-letters occurring in its argument, we have

$$|X|_{\mathcal{A}} = |\kappa(\mathcal{S}(E_1))|_{\mathcal{A}} = |\mathcal{S}(E_1)|_{\mathcal{A}}.$$

We can apply the induction hypothesis to E_1:

$$|E_1| = 3 \cdot p + |\mathcal{S}(E_1)|_{\mathcal{A}}.$$

We then find

$$|E| = 3 + |E_1| = 3 + 3 \cdot p + |\mathcal{S}(E_1)|_{\mathcal{A}} = 3 \cdot (p+1) + |X|_{\mathcal{A}}.$$

Hence, the claim is also valid for every DNA expression E that contains $p+1$ operators.

□

6.2 Blocks of components of a formal DNA molecule

Obviously, the minimal length of a DNA expression denoting a certain (expressible) formal DNA molecule X depends on X. We will see that it particularly depends on three simple counting functions of X. In this section, we study these counting functions. Two of the functions count certain subsequences (or blocks) of components of X. We first introduce these subsequences.

By Lemma 3.7, the components of a formal DNA molecule are double components and non-double components alternately. The non-double components are upper components, lower components, upper nick letters and lower nick letters. In Section 3.3, we categorized these components as single-stranded components and nick letters (see Figure 3.2). We now make a different categorization:

Definition 6.2 *Let X be a formal DNA molecule and let $x'_1 \dots x'_k$ for some $k \geq 1$ be the decomposition of X.*

- *An $\uparrow$-component x'_i of X is an upper component or a lower nick letter occurring in X.*
- *A $\downarrow$-component x'_i of X is a lower component or an upper nick letter occurring in X.*

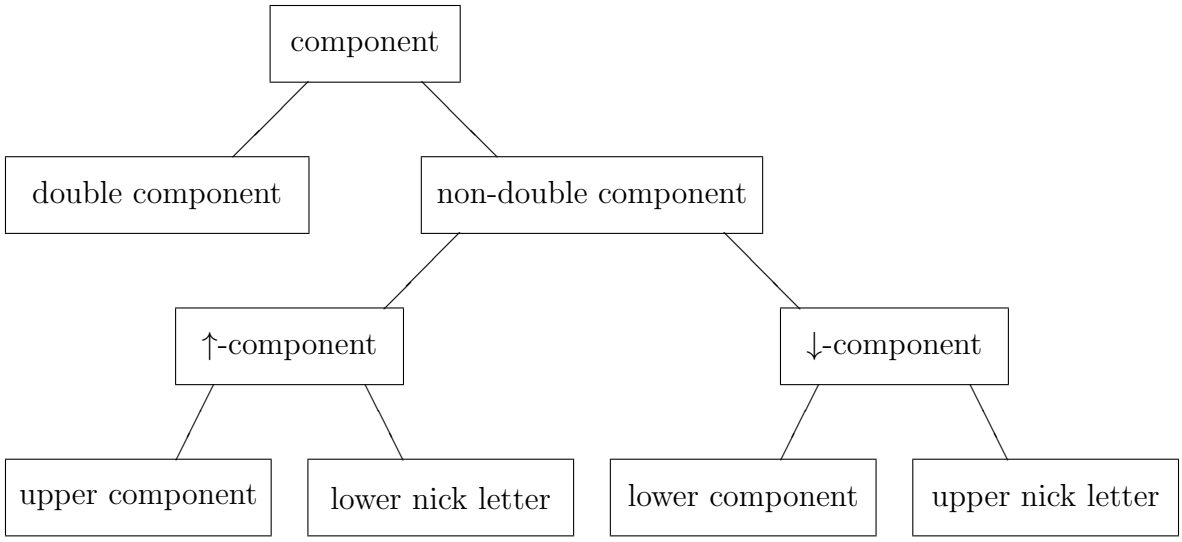


Figure 6.1: Relations between different types of components. Components can be divided into double components and non-double components, non-double components can in turn be divided into ↑-components and ↓-components, et cetera.

Recall that if $X = \mathcal{S}(E)$ for a DNA expression E, then upper components and lower nick netters occurring in X are the products of an operator ↑. Similarly, lower components and upper nick letters are produced by an operator ↓. This explains the terms ↑-*component* and ↓-*component*. Intuitively, one may regard an ↑-component as a component that 'breaks' the lower strand of a molecule. There is, of course, an analogous interpretation of a ↓-component. The new categorization is depicted in Figure 6.1.

We consider sequences of components of X. We make a simple observation, which follows immediately from the definition of the decomposition of a formal DNA molecule, Definition 3.5.

Lemma 6.3 *Let X be a formal DNA molecule, let $x'_1 \dots x'_k$ for some $k \geq 1$ be the decomposition of X and let X^s be a formal DNA submolecule of X which is a subsequence of the components of X: $X^s = x'_{a_0} \cdot \ldots \cdot x'_{a_1}$ with $1 \leq a_0 \leq a_1 \leq k$.*

Then the decomposition of X^s is $x'_{a_0} \dots x'_{a_1}$.

Hence, the components of X^s are simply the components of X that X^s is built up of.

When we ignore the double components, a formal DNA molecule consists of ↑-components and ↓-components. A (maximal) series of ↑-components is succeeded by a (maximal) series of ↓-components, which in turn is succeeded by a (maximal) series of ↑-components, and so on. We are interested in these maximal series, which we call *primitive ↑-blocks* and *primitive ↓-blocks*, respectively. A formal definition of these blocks also includes the double components occurring in the formal DNA molecule:

Definition 6.4 *Let X be a formal DNA molecule and let $x'_1 \dots x'_k$ for some $k \geq 1$ be the decomposition of X.*

A primitive ↑-block of X is an occurrence (Y_1, Y_2) of a non-empty substring X_1 of X such that $Y_1 = x'_1 \dots x'_{a_0-1}$ and $Y_2 = x'_{a_1+1} \dots x'_k$ for some a_0 and a_1 with $1 \leq a_0 \leq a_1 \leq k$ (hence $X_1 = x'_{a_0} \dots x'_{a_1}$), and

- *X_1 contains at least one non-double component,*

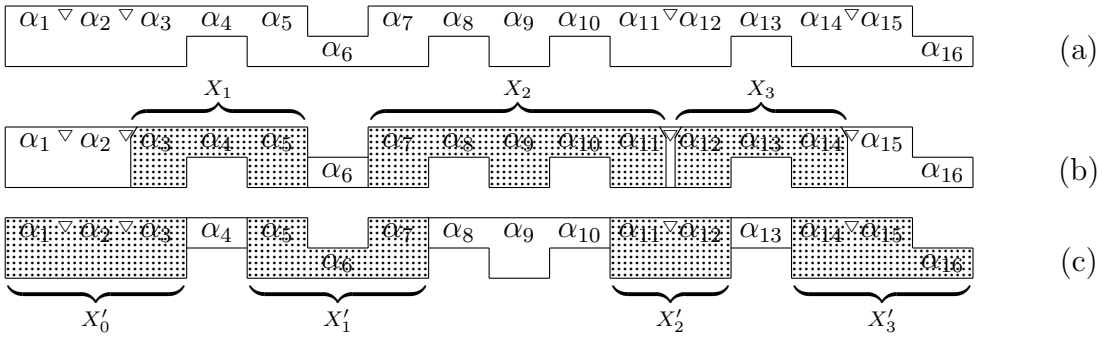


Figure 6.2: Primitive ↑-blocks and primitive ↓-blocks. (a) An example formal DNA molecule X that contains (upper) nick letters. (b) The primitive ↑-blocks of X. Note that the upper nick letters are not part of these blocks. (c) The primitive ↓-blocks of X.

- *each non-double component of X_1 is an ↑-component,*
- – *either $a_0 = 1$ (hence Y_1 is empty),*
  – *or $a_0 \geq 2$ and x'_{a_0-1} is a ↓-component,*

  and
- – *either $a_1 = k$ (hence Y_2 is empty),*
  – *or $a_1 \leq k-1$ and x'_{a_1+1} is a ↓-component.*

Note that a primitive ↑-block starts with the double component preceding the series of ↑-components (if such a double component exists) and it ends with the double component succeeding the series of ↑-components (again, if such a double component exists).

A primitive ↑-block of a formal DNA molecule X is formally defined as an *occurrence* (Y_1, Y_2) of a substring X_1 of X satisfying certain conditions. However, when the occurrence is clear from the context, we will often refer to a primitive ↑-block by the substring X_1 itself.

The definition of a primitive ↓-block is completely analogous to that of a primitive ↑-block. We may use the term *primitive block* to refer to either a primitive ↑-block, or a primitive ↓-block.

In Figure 6.2, we have indicated the primitive ↑-blocks and the primitive ↓-blocks of a certain formal DNA molecule containing upper nick letters.

Our first result on primitive blocks, dealing with certain simple types of formal DNA molecules, follows immediately from the definition.

Lemma 6.5 *Let X be a formal DNA molecule.*

1. *If $X = \binom{\alpha_1}{c(\alpha_1)}$ for an $\mathcal{N}$-word α_1, then X does not have any primitive block.*
2. *If X has at least one ↑-component, but does not have any ↓-component, then X is a primitive ↑-block of itself and X does not have any primitive ↓-block.*
3. *If X has at least one ↓-component, but does not have any ↑-component, then X is a primitive ↓-block of itself and X does not have any primitive ↑-block.*

Formal DNA molecules of the form $\binom{\alpha_1}{c(\alpha_1)}$ for an $\mathcal{N}$-word α_1 will come back frequently in the remainder of this chapter and in later chapters. Often, we are not interested in the actual $\mathcal{N}$-letters occurring in such a molecule (hence in α_1), but only in the shape of the molecule, for example, when we want to except molecules of this type from a certain statement. In order not to burden the text with unnecessary details, we may speak of a *double-complete* formal DNA molecule, when we mean a formal DNA molecule of the form $\binom{\alpha_1}{c(\alpha_1)}$ for an $\mathcal{N}$-word α_1.

We already mentioned that a primitive $\uparrow$-block of a formal DNA molecule X starts with the double component preceding a series of $\uparrow$-components, and ends with the double component succeeding this series (if these double components exist). We formalize and extend this observation for a primitive $\uparrow$-block which is not equal to X.

Lemma 6.6 *Let X be a formal DNA molecule, let $x'_1 \dots x'_k$ for some $k \geq 1$ be the decomposition of X, and let $X_1 = x'_{a_0} \dots x'_{a_1}$ for some a_0 and a_1 with $1 \leq a_0 \leq a_1 \leq k$ be a primitive $\uparrow$-block of X.*

1. *If $a_0 \geq 2$, then $a_0 < a_1 \leq k$, x'_{a_0-1} is a $\downarrow$-component, x'_{a_0} is a double component and x'_{a_0+1} is an $\uparrow$-component of X.*

2. *If $a_1 \leq k-1$, then $1 \leq a_0 < a_1$, x'_{a_1+1} is a $\downarrow$-component, x'_{a_1} is a double component and x'_{a_1-1} is an $\uparrow$-component of X.*

Proof:

1. Assume that $a_0 \geq 2$. Then, by definition, x'_{a_0-1} is a $\downarrow$-component, and by Lemma 3.7, x'_{a_0} is a double component. Because X_1 contains at least one $\uparrow$-component, we must have $a_0 < a_1$. Hence, x'_{a_0+1} is part of X_1, and by the definition of a primitive $\uparrow$-block, it is a double component or an $\uparrow$-component. By Lemma 3.7, it has to be an $\uparrow$-component.

2. The proof of this claim is analogous to that of the previous claim.

□

For the formal DNA molecule from Figure 6.2, the following result is clear from the picture. We will prove it in general.

Lemma 6.7 *Let X be a formal DNA molecule and let $x'_1 \dots x'_k$ for some $k \geq 1$ be the decomposition of X.*

1. *The primitive $\uparrow$-blocks of X are pairwise disjoint.*

2. *Each $\uparrow$-component of X occurs in a (exactly one) primitive $\uparrow$-block.*

3. *Let $X_1 = x'_{a_0} \dots x'_{a_1}$ and $X_2 = x'_{a_2} \dots x'_{a_3}$ with $1 \leq a_0 \leq a_1 < a_2 \leq a_3 \leq k$ be two consecutive primitive $\uparrow$-blocks of X. Then x'_{a_1} and x'_{a_2} are double components of X, $a_2 - a_1 \geq 2$ and $x'_{a_1} \dots x'_{a_2}$ is a primitive $\downarrow$-block of X.*

4. *Let $X_1 = x'_{a_0} \dots x'_{a_1}$ with $1 \leq a_0 \leq a_1 \leq k$ be the first primitive $\uparrow$-block of X. If $a_0 \geq 2$, then x'_{a_0} is a double component and $x'_1 \dots x'_{a_0}$ is a primitive $\downarrow$-block of X.*

5. *Let* $X_1 = x'_{a_0} \dots x'_{a_1}$ *with* $1 \leq a_0 \leq a_1 \leq k$ *be the last primitive* $\uparrow$*-block of* X*. If* $a_1 \leq k-1$*, then* x'_{a_1} *is a double component and* $x'_{a_1} \dots x'_k$ *is a primitive* $\downarrow$*-block of* X*.*

6. *Let* $X_1 = x'_{a_0} \dots x'_{a_1}$ *with* $1 \leq a_0 \leq a_1 \leq k$ *be a primitive* $\uparrow$*-block of* X *and let* $X_2 = x'_{a_2} \dots x'_{a_3}$ *with* $1 \leq a_2 \leq a_3 \leq k$ *be a primitive* $\downarrow$*-block of* X*. If* X_1 *and* X_2 *have a non-empty intersection, then*

   - *either* $1 \leq a_0 < a_1 = a_2 < a_3 \leq k$ *and* $x'_{a_1} (= x'_{a_2})$ *is a double component,*
   - *or* $1 \leq a_2 < a_3 = a_0 < a_1 \leq k$ *and* $x'_{a_3} (= x'_{a_0})$ *is a double component.*

Because of the analogous definitions of primitive $\uparrow$-blocks and primitive $\downarrow$-blocks, we have analogous results for primitive $\downarrow$-blocks.

By Claim 1, we can unambiguously order the different primitive $\uparrow$-blocks of a formal DNA molecule X, according to their occurrence in X. We can speak of the first, the second, ..., the last primitive $\uparrow$-block of a formal DNA molecule. Hence, Claims 3–5 make sense.

Claim 2 allows us to speak of the primitive $\uparrow$-block *of* a $\uparrow$-component, being the (unique) primitive $\uparrow$-block that the $\uparrow$-component is part of. Analogously, we have the primitive $\downarrow$-block of a $\downarrow$-component.

Finally, by Claim 6, if a primitive $\uparrow$-block and a primitive $\downarrow$-block of X have a non-empty intersection, then they overlap and the intersection consists only of the double component that one of the blocks ends with and the other one starts with. Therefore, it should not be confusing to say, e.g., that a primitive $\uparrow$-block is succeeded by a primitive $\downarrow$-block. For example, the primitive $\uparrow$-block X_2 of the formal DNA molecule from Figure 6.2 is succeeded by the primitive $\downarrow$-block X'_2, which in turn is succeeded by the primitive $\uparrow$-block X_3.

Proof of Lemma 6.7:

1. Let $X_1 = x'_{a_0} \dots x'_{a_1}$ with $1 \leq a_0 \leq a_1 \leq k$ and $X_2 = x'_{a_2} \dots x'_{a_3}$ with $1 \leq a_2 \leq a_3 \leq k$ be two different primitive $\uparrow$-blocks of X. Without loss of generality, assume that $a_0 \neq a_2$ (otherwise consider a_1 and a_3 and mirror the arguments). In particular, assume that $a_0 < a_2$.

   Because $1 \leq a_0 < a_2$, we have $a_2 \geq 2$ and thus, by definition, x'_{a_2-1} is a $\downarrow$-component. Further, because $a_0 \leq a_2 - 1$, we must have $a_1 < a_2 - 1$, because otherwise (the primitive $\uparrow$-block) X_1 would contain the $\downarrow$-component x'_{a_2-1}.

   Consequently, $a_0 \leq a_1 < a_2 - 1 < a_2 \leq a_3$, and X_1 and X_2 are disjoint.

2. Let x'_{i_0} be an arbitrary $\uparrow$-component of X. By Lemma 3.7, each component x'_i with $i \equiv i_0 \pmod 2$ is an $\uparrow$-component or a $\downarrow$-component and each component x'_i with $i \equiv i_0 + 1 \pmod 2$ is a double component.

   Now, let b_0 be the smallest index with $1 \leq b_0 \leq i_0$ and $b_0 \equiv i_0 \pmod 2$ such that each of $x'_{b_0}, x'_{b_0+2}, \dots, x'_{i_0-2}, x'_{i_0}$ is an $\uparrow$-component. Further, let b_1 be the largest index with $i_0 \leq b_1 \leq k$ and $b_1 \equiv i_0 \pmod 2$ such that each of $x'_{i_0}, x'_{i_0+2}, \dots, x'_{b_1-2}, x'_{b_1}$ is an $\uparrow$-component. Because x'_{i_0} itself is an $\uparrow$-component, b_0 and b_1 are well defined.

   We subsequently define indices a_0 and a_1 by

$$a_0 = \begin{cases} 1 & \text{if } b_0 \leq 2 \\ b_0 - 1 & \text{if } b_0 \geq 3 \end{cases} \qquad \text{and} \qquad a_1 = \begin{cases} k & \text{if } b_1 \geq k-1 \\ b_1 + 1 & \text{if } b_1 \leq k-2, \end{cases}$$

and let $X_1 = x'_{a_0} \dots x'_{a_1}$. It is easy to see that $1 \leq a_0 \leq b_0 \leq i_0 \leq b_1 \leq a_1 \leq k$. Indeed, X_1 contains x'_{i_0}.

By the definition of b_0 and b_1 and by the fact that $b_0 - 1 \leq a_0 \leq b_0$ and $b_1 \leq a_1 \leq b_1 + 1$, each of the components $x'_{a_0}, \dots, x'_{a_1}$ is either an $\uparrow$-component, or a double component. Further, either $a_0 = 1$, or $a_0 \geq 2$. In the latter case, $a_0 = b_0 - 1$, and hence $x'_{a_0-1} = x'_{b_0-2}$ is a $\downarrow$-component. Similarly, either $a_1 = k$, or $a_1 \leq k - 1$. In the latter case, $a_1 = b_1 + 1$ and $x'_{a_1+1} = x'_{b_1+2}$ is a $\downarrow$-component. Consequently, X_1 is a primitive $\uparrow$-block.

Because, by Claim 1, primitive $\uparrow$-blocks are pairwise disjoint, the $\uparrow$-component x'_{i_0} does not occur in any other primitive $\uparrow$-block of X.

3. In the proof of Claim 1, we have established that $a_1 < a_2 - 1 < a_2$, hence that $a_2 - a_1 \geq 2$. By Lemma 6.6, $1 \leq a_0 < a_1$, $a_2 < a_3 \leq k$, both x'_{a_1+1} and x'_{a_2-1} are $\downarrow$-components, both x'_{a_1} and x'_{a_2} are double components and both x'_{a_1-1} and x'_{a_2+1} are $\uparrow$-components.

   By Claim 2, each $\uparrow$-component occurs in a primitive $\uparrow$-block of X. Now, the existence of an $\uparrow$-component *between* X_1 and X_2 would contradict the assumption that they are consecutive. Hence, the sequence $x'_{a_1} \dots x'_{a_2}$ only consists of double components and $\downarrow$-components, and at least one of them is a $\downarrow$-component.

   This implies that $x'_{a_1} \dots x'_{a_2}$ satisfies all conditions of a primitive $\downarrow$-block.

4. The proof of this claim is similar to that of the previous claim.

   We first establish that if $a_0 \geq 2$, then x'_{a_0-1} is a $\downarrow$-component, x'_{a_0} is a double component and x'_{a_0+1} is an $\uparrow$-component. We subsequently prove that $x'_1 \dots x'_{a_0}$ does not contain any $\uparrow$-component, because otherwise we could find a primitive $\uparrow$-block before the *first* primitive $\uparrow$-block X_1. These properties together imply that $x'_1 \dots x'_{a_0}$ is a primitive $\downarrow$-block.

5. The proof of this claim is analogous to that of the previous claim.

6. Assume that X_1 and X_2 have a non-empty intersection. Because both X_1 and X_2 are built up of (complete) components of X, so is their intersection. Let x'_{i_0} be a component of X that occurs both in X_1 and X_2. Then $a_0 \leq i_0 \leq a_1$ and $a_2 \leq i_0 \leq a_3$.

   By the definition of (the primitive $\uparrow$-block) X_1, x'_{i_0} is not a $\downarrow$-component, and by the definition of (the primitive $\downarrow$-block) X_2, x'_{i_0} is not an $\uparrow$-component. Hence, it is a double component. Because X_1 contains at least one $\uparrow$-component, either $a_0 < i_0$, or $i_0 < a_1$ (or both).

   Assume that $a_0 < i_0$. Then by Lemma 3.7, x'_{i_0-1} is an $\uparrow$-component. This component cannot be part of (the primitive $\downarrow$-block) X_2. Because x'_{i_0} *is* part of X_2, we must have $a_2 = i_0$. Now, because X_2 contains at least one $\downarrow$-component, $i_0 < a_3$ and x'_{i_0+1} is a $\downarrow$-component. This component, in turn cannot be part of X_1, which implies that $a_1 = i_0$. In this case, we have the first subclaim.

   If, on the other hand, we assume that $i_0 < a_1$, then we obtain the second subclaim.

□

When we combine Lemma 6.5(1) and Lemma 6.7(2), we find

Corollary 6.8 *A formal DNA molecule does not have any primitive block, if and only if it is double-complete.*

We now consider formal DNA molecules which are not double-complete.

Lemma 6.9 *Let X be a formal DNA molecule which is not double-complete.*

1. *X can be considered as an alternating sequence of (all its) primitive $\uparrow$-blocks and (all its) primitive $\downarrow$-blocks. Any two consecutive primitive blocks in this sequence share (only) a double component of X.*
2. *(a) The first non-double component of X is an $\uparrow$-component, if and only if the alternating sequence from Claim 1 starts with a primitive $\uparrow$-block.*
   (b) The last non-double component of X is an $\uparrow$-component, if and only if the alternating sequence from Claim 1 ends with a primitive $\uparrow$-block.

Note that in Claim 1 we say "X can be considered as an alternating sequence ...," rather than 'X is an alternating sequence ...' The reason for this is that consecutive primitive blocks in this sequence (e.g., a primitive $\uparrow$-block and the primitive $\downarrow$-block succeeding it) are not disjoint. As we say in the second part of the claim, they share a double component. In the alternating sequence, we must, of course, include this double component only once.

It is easily verified that all claims are valid for the formal DNA molecule depicted in Figure 6.2. For this molecule, the alternating sequence is $X'_0, X_1, X'_1, X_2, X'_2, X_3, X'_3$.

Proof:

1. Without loss of generality, assume that X contains at least one $\uparrow$-component, and let $X_1, \ldots, X_r$ for some $r \geq 0$ be all primitive $\uparrow$-blocks of X, in the order of their occurrence in X. By Lemma 6.7(2), $r \geq 1$. By Lemma 6.7(1), the primitive $\uparrow$-blocks of X are pairwise disjoint. Now, consider a primitive $\uparrow$-block X_j of X with $1 \leq j \leq r-1$ (if this exists). By Lemma 6.7(3), there exists a primitive $\downarrow$-block X'_j of X that starts with the last (double) component of X_j and ends with the first (double) component of X_{j+1}.

   Assume that neither X_1 is a prefix of X, nor X_r is a suffix of X. Then by Lemma 6.7(4) and (5), both the prefix X'_0 of X that ends with the first (double) component of X_1 and the suffix X'_r of X that starts with the last (double) component of X_r are primitive $\downarrow$-blocks of X. We then have the alternating sequence $X'_0 X_1 X'_1 \ldots X_r X'_r$ of all primitive $\uparrow$-blocks $X_1, \ldots, X_r$ and primitive $\downarrow$-blocks $X'_0, X'_1, \ldots, X'_r$ which completely cover X. In fact, when we include each double component which is shared by a primitive $\uparrow$-block and a primitive $\downarrow$-block only once, this sequence is equal to X.

   We still have to prove that the alternating sequence includes *all* primitive $\downarrow$-blocks of X. Suppose that there is a primitive $\downarrow$-block X'_{00} of X different from the ones occurring in the sequence. By (the analogue for primitive $\downarrow$-blocks of) Lemma 6.7(1), it would have an empty intersection with each of $X'_0, X'_1, \ldots, X'_r$. Then, because X'_{00} is a substring of X, it should be contained in a primitive $\uparrow$-block X_j for some j with $1 \leq j \leq r$. By definition, the primitive $\downarrow$-block X'_{00} contains at least one $\downarrow$-component. This $\downarrow$-component would also be part of the primitive $\uparrow$-block X_j, which is impossible.

   We conclude that in this case, the claim holds. The proofs for the cases that either X_1 is a prefix of X, or X_r is a suffix of X (or both) are analogous.

2. (a) Assume that the alternating sequence from Claim 1 starts with a primitive $\uparrow$-block X_0 of X. By definition, a primitive $\uparrow$-block contains at least one $\uparrow$-component and does not contain any $\downarrow$-component. Hence, the first non-double component of X is the first non-double component of its prefix X_0, which is an $\uparrow$-component.

   Conversely, if the first non-double component of X is an $\uparrow$-component, then the alternating sequence cannot start with a primitive $\downarrow$-block. Hence, it must start with a primitive $\uparrow$-block.

   (b) The proof of this subclaim is analogous to that of the previous subclaim.

□

We now define functions that count the primitive $\uparrow$-blocks, the primitive $\downarrow$-blocks and the double components occurring in a formal DNA molecule X.

Definition 6.10 *Let X be a formal DNA molecule.*

- $B_\uparrow(X)$ *is the number of primitive $\uparrow$-blocks of X.*
- $B_\downarrow(X)$ *is the number of primitive $\downarrow$-blocks of X.*
- $n_\updownarrow(X)$ *is the number of double components of X.*

For the formal DNA molecule X from Figure 6.2, we have $B_\uparrow(X) = 3$, $B_\downarrow(X) = 4$ and $n_\updownarrow(X) = 10$.

We are interested in the values of the functions $B_\uparrow$, $B_\downarrow$ and $n_\updownarrow$ for formal DNA molecules X. Sometimes, however, it will be convenient to have the possibility to provide an argument $X = \lambda$. In line with the intuition, we define

$$B_\uparrow(\lambda) = B_\downarrow(\lambda) = n_\updownarrow(\lambda) = 0.$$

In addition to the three new counting functions, we will use $\#_\triangledown(X)$, $\#_\triangle(X)$, $\#_{\triangledown,\triangle}(X)$, $\#_\uparrow(E)$, $\#_\downarrow(E)$, $\#_{\uparrow,\downarrow}(E)$ and $\#_\updownarrow(E)$. Here, X and E may be arbitrary strings, but often X will be a formal DNA molecule and E will be a DNA expression.

The following result is immediate from Lemma 3.7:

Lemma 6.11 *Let X be a formal DNA molecule and let $x'_1 \dots x'_k$ for some $k \geq 1$ be the decomposition of X.*

If x'_1 is a double component, then $n_\updownarrow(X) = \lceil \frac{k}{2} \rceil$. If x'_1 is not a double component, then $n_\updownarrow(X) = \lfloor \frac{k}{2} \rfloor$.

The different counting functions on formal DNA molecules can be related to each other:

Lemma 6.12 *Let X be a formal DNA molecule.*

1. $B_\uparrow(X) \leq B_\downarrow(X) + 1$.
2. $B_\downarrow(X) \leq B_\uparrow(X) + 1$.
3. *If X does not contain any single-stranded component, then $n_\updownarrow(X) = \#_{\triangledown,\triangle}(X) + 1$.*
4. *If X contains at least one nick letter, then $n_\updownarrow(X) \geq \#_{\triangledown,\triangle}(X) + 1$.*

5. $n_{\updownarrow}(X) \geq \#_{\triangledown,\triangle}(X)$.

Proof:

1, 2. If X is double-complete, then by Lemma 6.5(1), X does not have any primitive block. Hence, Claims 1 and 2 are trivially valid.

If X is not double-complete, then by Lemma 6.9(1), the primitive $\uparrow$-blocks and primitive $\downarrow$-blocks occur in X alternately. Hence, the difference between their numbers of occurrences can be at most 1. Again, Claims 1 and 2 follow immediately.

3. This claim follows immediately from Corollary 3.9(2).

4. Let $x'_1 \dots x'_k$ for some $k \geq 1$ be the decomposition of X. By the definition of a formal DNA molecule, every nick letter occurring in X is preceded and succeeded by a double component. Hence, for each component x'_i of X that is a nick letter, $i \geq 2$ and x'_{i-1} is a double component. Further, if x'_{i_0} is the last nick letter occurring in X, then $i_0 \leq k-1$ and also x'_{i_0+1} is a double component. Obviously, all these double components are different, and thus $n_{\updownarrow}(X) \geq \#_{\triangledown,\triangle}(X) + 1$.

5. If X is nick free (hence $\#_{\triangledown,\triangle}(X) = 0$), then the claim holds because $n_{\updownarrow}(X) \geq 0$. If X is not nick free, then the claim follows from Claim 4.

□

In the next result, we consider nick free formal DNA molecules. In such formal DNA molecules, each non-double component is a single-stranded component (see Figure 3.2), each $\uparrow$-component is an upper component and each $\downarrow$-component is a lower component (see Figure 6.1). In fact, all claims below can directly be generalized to formal DNA molecules that may contain nick letters. However, because we will use the claims only in the nick free case, we give the nick free formulation.

Lemma 6.13 *Let X be a nick free formal DNA molecule.*

1. *(a) $B_{\uparrow}(X) = 0$ if and only if X does not contain any upper component.*
   (b) $B_{\downarrow}(X) = 0$ if and only if X does not contain any lower component.
2. *X is not double-complete, if and only if X contains at least one single-stranded component.*
3. *Assume that X is not double-complete.*
   (a) If both the first single-stranded component and the last single-stranded component of X are upper components, then $B_{\uparrow}(X) = B_{\downarrow}(X) + 1$.
   (b) If the first single-stranded component of X is a lower component and the last single-stranded component of X is an upper component, then $B_{\uparrow}(X) = B_{\downarrow}(X)$.
   (c) If the first single-stranded component of X is an upper component and the last single-stranded component of X is a lower component, then $B_{\uparrow}(X) = B_{\downarrow}(X)$.
   (d) If both the first single-stranded component and the last single-stranded component of X are lower components, then $B_{\uparrow}(X) = B_{\downarrow}(X) - 1$.

Proof:

1. (a) By the definition of a primitive $\uparrow$-block, if X does not have any upper component, then $B_\uparrow(X) = 0$. Conversely, by Lemma 6.7(2), if $B_\uparrow(X) = 0$, then X cannot have an upper component.

   (b) The proof of this subclaim is analogous to that of the previous subclaim.

2. This claim is immediate from Corollary 3.8(1).

3. All four subclaims follow immediately from Lemma 6.9.

□

When a formal DNA molecule is denoted by a DNA expression, the values of the functions $B_\uparrow$, $B_\downarrow$ and $n_\updownarrow$ for the full molecule can be related to the values for its constituents. In order to establish these relations, we will systematically analyse the effects of the operators $\uparrow$, $\downarrow$ and $\updownarrow$ on the values of these counting functions.

Some of the relations we obtain, are not directly useful for the ultimate goal of this chapter, the determination of lower bounds on the length of a DNA expression. For the sake of completeness, we give these relations anyway.

In the definition of the semantics of an $\uparrow$-expression, a $\downarrow$-expression and an $\updownarrow$-expression, there is an important role for the functions ν^+, ν^- and κ, respectively (see Definition 4.1). We first consider the effects of ν^+ on the values of the counting functions. Of course, the effects of ν^- are analogous. We will subsequently examine the effects of κ.

The function ν^+ removes all upper nick letters occurring in its argument. The removal of even a single upper nick letter may affect the counting numbers.

Lemma 6.14 *Let X be a formal DNA molecule containing at least one upper nick letter, and let X' be the string that results from X by removing one upper nick letter. Then X' is a formal DNA molecule and*

1. $B_\uparrow(X') \leq B_\uparrow(X)$,
2. $B_\uparrow(X') \geq B_\uparrow(X) - 1$,
3. $B_\downarrow(X') \leq B_\downarrow(X)$,
4. $B_\downarrow(X') \geq B_\downarrow(X) - 1$ *and*
5. $n_\updownarrow(X') = n_\updownarrow(X) - 1$.

Proof: Because X is not double-complete, by Lemma 6.9(1), it can be considered as an alternating sequence of all its primitive $\uparrow$-blocks and all its primitive $\downarrow$-blocks.

Let $x'_1 \dots x'_k$ for some $k \geq 1$ be the decomposition of X and let x'_{i_0} with $1 \leq i_0 \leq k$ be the upper nick letter that is removed. Hence, $X' = x'_1 \dots x'_{i_0-1} x'_{i_0+1} \dots x'_k$.

By the definition of a formal DNA molecule (Definition 3.2), nick letters may occur only between two double components. Consequently, $2 \leq i_0 \leq k-1$ and both x'_{i_0-1} and x'_{i_0+1} are double components. When x'_{i_0} is removed, the two double components become adjacent and thus form one larger double component $x'_{i_0-1}x'_{i_0+1}$. Indeed, the resulting string X' satisfies all conditions of a formal DNA molecule.

Because two double components merge into one and the other double components are not affected by the removal of x'_{i_0}, $n_\updownarrow(X') = n_\updownarrow(X) - 1$, which is Claim 5.

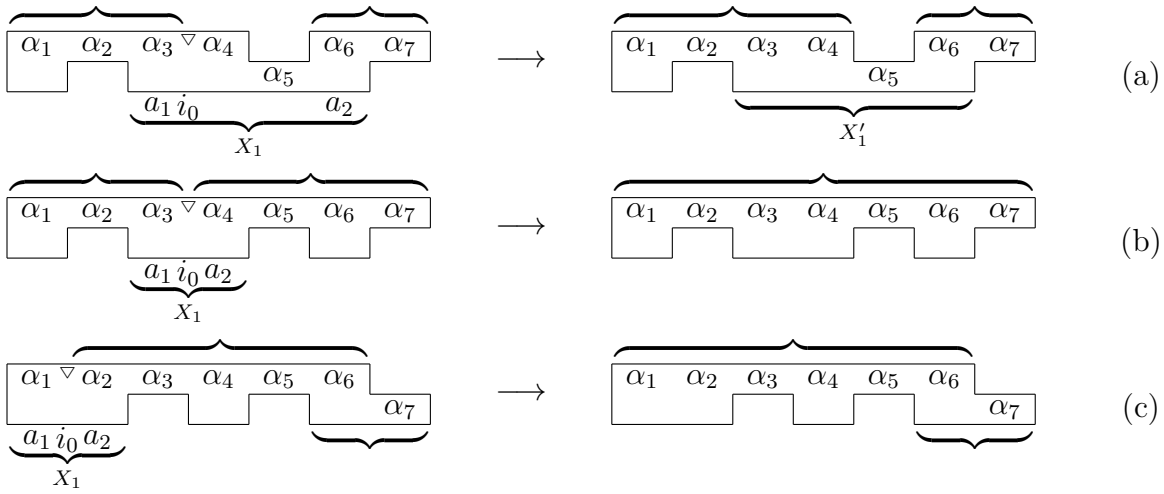


Figure 6.3: The effect on primitive blocks (indicated by the braces) of removing an upper nick letter in three different cases. (a) The upper nick letter removed is the first, but not the only ↓-component of its primitive ↓-block X_1. The last (double) component of the preceding primitive ↑-block is replaced by a larger double component. (b) The upper nick letter removed is the only ↓-component of its primitive ↓-block X_1. The primitive ↑-block preceding X_1 and the one succeeding X_1 merge into one. (c) The upper nick letter removed is the only ↓-component of its primitive ↓-block X_1. X_1 is not preceded by a primitive ↑-block. The first (double) component of the primitive ↑-block succeeding X_1 is replaced by a larger double component.

The upper nick letter x'_{i_0} is a ↓-component. By Lemma 6.7(2), there is (exactly) one primitive ↓-block X_1 of X containing x'_{i_0}, say $X_1 = x'_{a_1} \dots x'_{i_0-1} x'_{i_0} x'_{i_0+1} \dots x'_{a_2}$ for some a_1 and a_2 with $1 \leq a_1 \leq i_0 \leq a_2 \leq k$. Because $2 \leq i_0 \leq k-1$ and both x'_{i_0-1} and x'_{i_0+1} are double components, it follows from the definition of a primitive ↓-block that $a_1 \leq i_0 - 1$ and $i_0 + 1 \leq a_2$.

Now, there are two possibilities: either X_1 contains ↓-components other than x'_{i_0}, or it does not.

- In the former case, either $a_1 < i_0 - 1$ or $i_0 + 1 < a_2$ (or both). When x'_{i_0} is removed from X_1, the result $X'_1 = x'_{a_1} \dots x'_{i_0-1} x'_{i_0+1} \dots x'_{a_2}$ still contains at least one ↓-component. It is easily verified that it also satisfies the other conditions of a primitive ↓-block of X'. Hence, the primitive ↓-block X_1 has been replaced by the primitive ↓-block X'_1.

  If x'_{i_0} was the first ↓-component of X_1 (hence, $a_1 = i_0 - 1$) and X_1 was preceded in X by a primitive ↑-block, then the last component of this primitive ↑-block was the double component x'_{i_0-1} and has now become the double component $x'_{i_0-1} x'_{i_0+1}$. This effect is depicted in Figure 6.3(a). Analogously, the first component of the primitive ↑-block succeeding X_1 may have changed. None of the other primitive ↑-blocks and primitive ↓-blocks of X is affected by the removal of x'_{i_0}. Hence, in this case, $B_\uparrow(X') = B_\uparrow(X)$ and $B_\downarrow(X') = B_\downarrow(X)$.

- Now, consider the case that the upper nick letter x'_{i_0} is the only ↓-component of X_1. Then X_1 only consists of the double component x'_{i_0-1}, the upper nick letter x_{i_0} and the double component x'_{i_0+1}: $a_1 = i_0 - 1$ and $a_2 = i_0 + 1$. When we remove x'_{i_0}, we

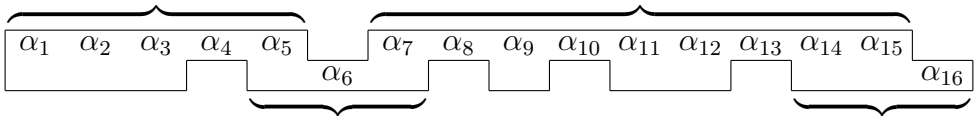


Figure 6.4: The result of applying ν^+ to the formal DNA molecule X from Figure 6.2. The primitive $\uparrow$-blocks and primitive $\downarrow$-blocks of $\nu^+(X)$ are also indicated.

lose the primitive $\downarrow$-block X_1. By Lemma 6.7(3), the other primitive $\downarrow$-blocks of X (if any) are separated from X_1 by at least one $\uparrow$-component. They are not affected by the removal of x'_{i_0}, hence $B_\downarrow(X') = B_\downarrow(X) - 1$.

For the effect on $B_\uparrow(X)$, we distinguish a number of subcases. If X_1 is both preceded and succeeded in X by a primitive $\uparrow$-block (hence $a_1 \geq 2$ and $a_2 \leq k-1$), then these two blocks merge into one when the upper nick letter x'_{i_0} separating them is removed (see Figure 6.3(b)). The other primitive $\uparrow$-blocks of X (if any) are not affected. In this case, $B_\uparrow(X') = B_\uparrow(X) - 1$.

If, on the other hand, X_1 is the first primitive block of X and is succeeded in X by a primitive $\uparrow$-block ($a_1 = 1$ and $a_2 \leq k-1$), then the effect of the removal of x'_{i_0} is smaller: the first component of the succeeding primitive $\uparrow$-block, which used to be the double component x'_{a_2}, is replaced by the double component $x'_{a_1}x'_{a_2}$ (see Figure 6.3(c)). The *number* of primitive $\uparrow$-blocks remains the same: $B_\uparrow(X') = B_\uparrow(X)$. This is also the case if X_1 is preceded but not succeeded in X by a primitive $\uparrow$-block ($a_1 \geq 2$ and $a_2 = k$) and if X_1 is the only primitive block of X ($a_1 = 1$ and $a_2 = k$). Note that in the last case, $k = 3$, $B_\uparrow(X') = B_\uparrow(X) = 0$, $B_\downarrow(X) = 1$, $B_\downarrow(X') = 0$ and the resulting formal DNA molecule X' consists only of the double component $x'_1x'_3$.

In all cases we considered, $B_\uparrow(X')$ and $B_\downarrow(X')$ satisfy the inequalities in Claims 1–4. □

We now examine the effects of removing *all* upper nick letters, by means of the function ν^+. We first consider an example.

Example 6.15 In Figure 6.4, we have depicted $\nu^+(X)$ for the formal DNA molecule X from Figure 6.2. As we have established before,

$$B_\uparrow(X) = 3,\ B_\downarrow(X) = 4 \ \text{ and } \ n_\updownarrow(X) = 10.$$

After the removal of the $\#_\triangledown(X) = 4$ upper nick letters, the numbers are

$$B_\uparrow(\nu^+(X)) = 2,\ B_\downarrow(\nu^+(X)) = 2 \ \text{ and } \ n_\updownarrow(\nu^+(X)) = 6.$$

■

We use Lemma 6.14 to determine (upper bounds and lower bounds on) the effects of the function ν^+ on the counting numbers for *arbitrary* formal DNA molecules.

Lemma 6.16 *Let X be a formal DNA molecule. Then $\nu^+(X)$ is a formal DNA molecule and*

1. $B_\uparrow(\nu^+(X)) \leq B_\uparrow(X)$,

2. $B_{\uparrow}(\nu^+(X)) \geq B_{\uparrow}(X) - \#_{\triangledown}(X)$,
3. $B_{\downarrow}(\nu^+(X)) \leq B_{\downarrow}(X)$,
4. $B_{\downarrow}(\nu^+(X)) \geq B_{\downarrow}(X) - \#_{\triangledown}(X)$ *and*
5. $n_{\updownarrow}(\nu^+(X)) = n_{\updownarrow}(X) - \#_{\triangledown}(X)$.

Proof: Already at the definition of the function ν^+, we observed that for each formal DNA molecule X, also $\nu^+(X)$ is a formal DNA molecule.

When we apply ν^+ to X, we remove all upper nick letters occurring in X at once. We may, however, also remove the upper nick letters one by one. In whatever order we remove the upper nick letters, the final result is the same: $\nu^+(X)$. At each of the $\#_{\triangledown}(X)$ removals, we can apply Lemma 6.14. Now, each of the claims follows from the repeated application of the corresponding claim from Lemma 6.14. □

By the above result (and the analogue for $\nu^-(X)$), we can derive invariants for the functions ν^+, ν^- and ν: the function $n_{\updownarrow}(\cdot) - \#_{\triangledown}(\cdot)$ is an invariant for the function ν^+, the function $n_{\updownarrow}(\cdot) - \#_{\triangle}(\cdot)$ is an invariant for the function ν^-, and the function $n_{\updownarrow}(\cdot) - \#_{\triangledown,\triangle}(\cdot)$ is an invariant for the function ν:

Corollary 6.17 *Let X be a formal DNA molecule.*

1. $n_{\updownarrow}(\nu^+(X)) - \#_{\triangledown}(\nu^+(X)) = n_{\updownarrow}(X) - \#_{\triangledown}(X)$
2. $n_{\updownarrow}(\nu^-(X)) - \#_{\triangle}(\nu^-(X)) = n_{\updownarrow}(X) - \#_{\triangle}(X)$
3. $n_{\updownarrow}(\nu(X)) - \#_{\triangledown,\triangle}(\nu(X)) = n_{\updownarrow}(X) - \#_{\triangledown,\triangle}(X)$

Proof: By the definition of the function ν^+, $\#_{\triangledown}(\nu^+(X)) = 0$ and $\#_{\triangle}(\nu^+(X)) = \#_{\triangle}(X)$. Now, Claim 1 follows immediately from Lemma 6.16(5) Then by analogy, we also have Claim 2. Finally, Claim 3 follows from the other two, because the function ν is the composition of ν^+ and ν^-:

$$\begin{aligned}
& n_{\updownarrow}(\nu(X)) - \#_{\triangledown,\triangle}(\nu(X)) \\
&= n_{\updownarrow}(\nu^+(\nu^-(X))) - \#_{\triangledown}(\nu^+(\nu^-(X))) - \#_{\triangle}(\nu^+(\nu^-(X))) \\
&= n_{\updownarrow}(\nu^-(X)) - \#_{\triangledown}(\nu^-(X)) - \#_{\triangle}(\nu^-(X)) \\
&= n_{\updownarrow}(X) - \#_{\triangledown}(X) - \#_{\triangle}(X) \\
&= n_{\updownarrow}(X) - \#_{\triangledown,\triangle}(X).
\end{aligned}$$

□

Instead of ν^+ or ν^-, which remove nick letters from their arguments, we may apply κ to a formal DNA molecule. This function complements all single-stranded components of its argument, i.e., it substitutes them by the corresponding double $\mathcal{A}$-words[1]. In some sense, we can also regard *this* as 'removing' non-double components.

We will formulate inequalities for the values of the three counting functions after the application of κ. As we did with the removal of nick letters, we first examine the effects of the complementation of one single-stranded component, in particular of a lower component. For the complementation of an upper component, we have, of course, analogous (in fact: equal) inequalities.

[1]We do not say *double components*, because in general the corresponding double $\mathcal{A}$-words are not components of the resulting formal DNA molecule.

Lemma 6.18 *Let X be a formal DNA molecule containing at least one lower component, and let X' be the string that results from X by complementing one lower component of X. Then X' is a formal DNA molecule and*

1. $B_{\uparrow}(X') \leq B_{\uparrow}(X)$,
2. $B_{\uparrow}(X') \geq B_{\uparrow}(X) - 1$,
3. $B_{\downarrow}(X') \leq B_{\downarrow}(X)$,
4. $B_{\downarrow}(X') \geq B_{\downarrow}(X) - 1$.
5. $n_{\updownarrow}(X') \leq n_{\updownarrow}(X) + 1$ *and*
6. $n_{\updownarrow}(X') \geq n_{\updownarrow}(X) - 1$.

Proof: Let $\binom{-}{c(\alpha_1)}$ for an $\mathcal{N}$-word α_1 be the lower component that is complemented.

The main difference between Claims 1–4 of this result and Lemma 6.14(1)–(4) is that the $\downarrow$-component $\binom{-}{c(\alpha_1)}$ which is 'removed' here is not necessarily preceded and succeeded in X by a double component. It may also be the first and/or the last component of X. This does, however, not change the structure of the proof. We distinguish the same cases and in each of the cases we have the same effect on $B_{\uparrow}(X)$ and $B_{\downarrow}(X)$ as in the proof of Lemma 6.14. For example, let X_1 be the primitive $\downarrow$-block of $\binom{-}{c(\alpha_1)}$. If the lower component $\binom{-}{c(\alpha_1)}$ is preceded in X by a double component $\binom{\alpha_0}{c(\alpha_0)}$ and succeeded in X by a double component $\binom{\alpha_2}{c(\alpha_2)}$, then we obtain one double component $\binom{\alpha_0\alpha_1\alpha_2}{c(\alpha_0\alpha_1\alpha_2)}$ when we replace $\binom{-}{c(\alpha_1)}$ by the corresponding double $\mathcal{A}$-word. This may or may not affect $B_{\uparrow}(X)$ and $B_{\downarrow}(X)$, depending on whether or not X_1 contains other $\downarrow$-components, and on whether or not X_1 is preceded and/or succeeded in X by a primitive $\uparrow$-block.

We turn to Claims 5 and 6, where we consider $n_{\updownarrow}(X')$. If $\binom{-}{c(\alpha_1)}$ is the only component of X, hence if $X = \binom{-}{c(\alpha_1)}$, then $X' = \binom{\alpha_1}{c(\alpha_1)}$, $n_{\updownarrow}(X) = 0$ and $n_{\updownarrow}(X') = 1$.

Now, assume that $\binom{-}{c(\alpha_1)}$ is not the only component of X. If it is the first component, then by Lemma 3.7, it is succeeded in X by a double component. This double component is extended to the left by $\binom{\alpha_1}{c(\alpha_1)}$ when we complement our lower component. Because the other double components of X are not affected, the *number* of double components remains the same: $n_{\updownarrow}(X') = n_{\updownarrow}(X)$. Analogously, if $\binom{-}{c(\alpha_1)}$ is the last component of X, then $n_{\updownarrow}(X') = n_{\updownarrow}(X)$. Finally, if $\binom{-}{c(\alpha_1)}$ is neither the first component, nor the last component of X, then it is both preceded and succeeded in X by a double component. These two double components merge into one when we complement $\binom{-}{c(\alpha_1)}$. Hence, we lose one double component: $n_{\updownarrow}(X') = n_{\updownarrow}(X) - 1$.

We can thus conclude that

$$n_{\updownarrow}(X') = n_{\updownarrow}(X) + b_{\mathrm{f}} + b_{\mathrm{l}} - 1, \tag{6.1}$$

where

$$b_{\mathrm{f}} = \begin{cases} 1 & \text{if } \binom{-}{c(\alpha_1)} \text{ is the first component of } X \\ 0 & \text{otherwise} \end{cases} \qquad \text{and}$$

$$b_{\mathrm{l}} \;=\; \begin{cases} 1 & \text{if } \binom{-}{c(\alpha_1)} \text{ is the last component of } X \\ 0 & \text{otherwise.} \end{cases}$$

Indeed, $n_{\updownarrow}(X')$ satisfies the inequalities in Claims 5 and 6. □

When we apply the function κ to a formal DNA molecule X, all single-stranded components of X are complemented. The new values for the counting functions can be related to the old ones, as follows:

Lemma 6.19 *Let X be a formal DNA molecule, and let $r \geq 0$ be the number of single-stranded components of X. Then $\kappa(X)$ is a formal DNA molecule and*

1. $B_{\uparrow}(\kappa(X)) \leq B_{\uparrow}(X)$,
2. $B_{\uparrow}(\kappa(X)) \geq B_{\uparrow}(X) - r$,
3. $B_{\downarrow}(\kappa(X)) \leq B_{\downarrow}(X)$,
4. $B_{\downarrow}(\kappa(X)) \geq B_{\downarrow}(X) - r$,
5. $n_{\updownarrow}(\kappa(X)) \leq n_{\updownarrow}(X) + 2 - r$,
6. $n_{\updownarrow}(\kappa(X)) \leq n_{\updownarrow}(X) + 1$ *and*
7. $n_{\updownarrow}(\kappa(X)) \geq n_{\updownarrow}(X) - r$.

One might think that Claim 6 does not add much to Claim 5. Only for the case that $r = 0$, it is (slightly) stronger than Claim 5. Sometimes, however, it is useful to have an upper bound on $n_{\updownarrow}(\kappa(X))$ that does not depend on the number of single-stranded components of X.

Proof: Already at the definition of the function κ, we observed that for each formal DNA molecule X, also $\kappa(X)$ is a formal DNA molecule.

When we complement the r single-stranded components of X one by one, we obtain $\kappa(X)$ (cf. the proof of Lemma 6.16). At each of these steps, we can apply Lemma 6.18 (or the analogue for upper components). Now, each of Claims 1–4 is the result of the repeated application of the corresponding claim from Lemma 6.18. Likewise, Claim 7 follows from Lemma 6.18(6).

5. When we complement one single-stranded component of X, this does not affect the position of the remaining $r - 1$ single-stranded components. That is, let x'_{i_0} be one of these remaining single-stranded components. Then x'_{i_0} is the first component of X, if and only if it is the first component of the resulting formal DNA molecule X'. Analogously, x'_{i_0} is the last component of X, if and only if it is the last component of X'.

   Hence, when we successively complement all single-stranded components of X, there will be at most one for which the variable b_{f} from (6.1) is 1, and there will be at most one for which the variable b_{l} is 1. Then the total contribution of all variables b_{f} and b_{l} is at most 2, which proves the claim.

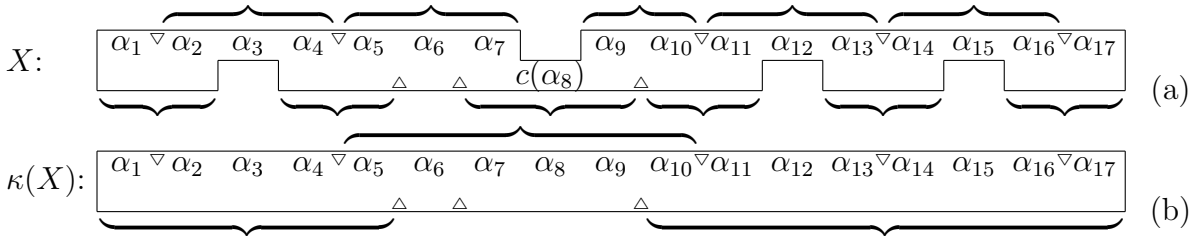


Figure 6.5: The formal DNA molecule from Example 6.20, which achieves the lower bounds from Lemma 6.19(2) and (4). (a) The original formal DNA molecule X, with four single-stranded components, $B_{\uparrow}(X) = 5$ and $B_{\downarrow}(X) = 6$. (b) $\kappa(X)$, with $B_{\uparrow}(\kappa(X)) = 1$ and $B_{\downarrow}(\kappa(X)) = 2$.

6. We may regard this claim as a corollary to the previous claim, when we separately observe that it holds trivially for the case that X does not have any single-stranded component (i.e., that $\kappa(X) = X$). We may also prove it as follows:

   The function κ neither introduces, nor removes nick letters. In particular, it does not change the *number* of nick letters in the formal DNA molecule X: $\#_{\triangledown,\triangle}(\kappa(X)) = \#_{\triangledown,\triangle}(X)$. When we combine this with Lemma 6.12(3) and (5), we obtain

$$n_{\updownarrow}(\kappa(X)) = \#_{\triangledown,\triangle}(\kappa(X)) + 1 = \#_{\triangledown,\triangle}(X) + 1 \leq n_{\updownarrow}(X) + 1.$$

□

As an aside, we investigate which formal DNA molecules achieve the lower bounds for $B_{\uparrow}(\kappa(X))$ and $B_{\downarrow}(\kappa(X))$ from Lemma 6.19(2) and (4). Intuitively, whenever we complement an upper component in such a molecule, the corresponding primitive $\uparrow$-block disappears and there are a preceding and a succeeding primitive $\downarrow$-block that merge into one. The complementation of a lower component has analogous effects.

We first consider an example.

Example 6.20 The formal DNA molecule depicted in Figure 6.5 has $r = 4$ single-stranded components. After application of the function κ, both the number of primitive $\uparrow$-blocks and the number of primitive $\downarrow$-blocks have decreased by 4. Note that in the original molecule each upper component is 'enclosed' by two upper nick letters, and the (only) lower component is 'enclosed' by two lower nick letters. ■

In two steps, we will derive a formal characterization of (arbitrary) formal DNA molecules that achieve the two lower bounds. Some of the arguments we use in the proofs, in fact also underlay Lemma 6.18(1)–(4). Because we did not give a detailed proof of those claims, it does not suffice here to simply refer to their proof. Therefore, we work out the argumentation completely.

Lemma 6.21 *Let X be a formal DNA molecule. If X contains*

- *either a lower component that is not the only $\downarrow$-component of its primitive $\downarrow$-block,*
- *or an upper component that is not the only $\uparrow$-component of its primitive $\uparrow$-block*

(or both), then X achieves neither of the lower bounds from Lemma 6.19(2) and Lemma 6.19(4).

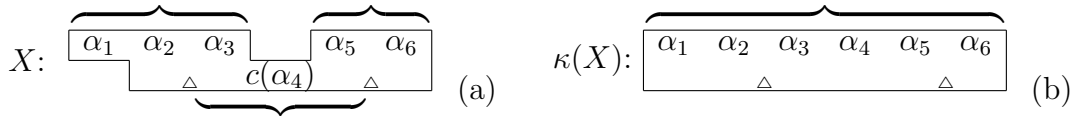


Figure 6.6: The formal DNA molecule from Example 6.22, which does not achieve the lower bounds from Lemma 6.19(2) and (4), see Lemma 6.21. (a) The original formal DNA molecule X, with two single-stranded components, $B_\uparrow(X) = 2$ and $B_\downarrow(X) = 1$. The upper component is not the only $\uparrow$-component of its primitive $\uparrow$-block. (b) $\kappa(X)$, with $B_\uparrow(\kappa(X)) = 1$ and $B_\downarrow(\kappa(X)) = 0$.

Example 6.22 Consider the formal DNA molecule X depicted in Figure 6.6(a), which has $r = 2$ single-stranded components: an upper component and a lower component. The upper component is not the only $\uparrow$-component in its primitive $\uparrow$-block. Indeed, $B_\uparrow(\kappa(X)) = 1 > 2 - 2 = B_\uparrow(X) - r$ and $B_\downarrow(\kappa(X)) = 0 > 1 - 2 = B_\downarrow(X) - r$. ■

Proof of Lemma 6.21: Let $x'_1 \dots x'_k$ for some $k \geq 1$ be the decomposition of X. Assume that x'_{i_0} with $1 \leq i_0 \leq k$ is a lower component of X which is not the only $\downarrow$-component of its primitive $\downarrow$-block X_1.

Let X' be the formal DNA molecule that results when we complement x'_{i_0}, and let X'_1 be the substring of X' that corresponds to X_1. By Lemma 3.7, we must have $k \geq 3$ and x'_{i_0} is preceded and/or succeeded in X by a double component. By the definition of a primitive $\downarrow$-block, these components are parts of X_1. Consequently, when we complement x'_{i_0}, the resulting (extended) double component is part of X'_1. Because the other components of X are not affected by the complementation, X'_1 is a sequence of components of X'.

Because x'_{i_0} is not the only $\downarrow$-component of X_1, there is at least one $\downarrow$-component left in X'_1. It is easy to verify that X'_1 also satisfies the other conditions of a primitive $\downarrow$-block. Hence, one primitive $\downarrow$-block (X_1) has been replaced by another (X'_1). Moreover, as in the proof of Lemma 6.14, the other primitive blocks of X (if any) are not affected, possibly apart from an extension of a double component. In particular, the *number* of primitive $\uparrow$-blocks and the *number* of primitive $\downarrow$-blocks are unchanged: $B_\uparrow(X') = B_\uparrow(X)$ and $B_\downarrow(X') = B_\downarrow(X)$.

Let us use r to denote the number of single-stranded components of X. Clearly, $r \geq 1$, the number of single-stranded components of X' is $r - 1$ and $\kappa(X) = \kappa(X')$. When we apply Lemma 6.19(2) and (4) to X', we obtain:

$$\begin{aligned} B_\uparrow(\kappa(X)) &= B_\uparrow(\kappa(X')) \geq B_\uparrow(X') - (r-1) = B_\uparrow(X) - (r-1) \qquad \text{and} \\ B_\downarrow(\kappa(X)) &= B_\downarrow(\kappa(X')) \geq B_\downarrow(X') - (r-1) = B_\downarrow(X) - (r-1). \end{aligned}$$

Hence, X achieves neither of the lower bounds from Lemma 6.19(2) and (4).

The proof for the case that X contains an upper component that is not the only $\uparrow$-component of its primitive $\uparrow$-block, is analogous. □

Lemma 6.23 *Let X be a formal DNA molecule and let $x'_1 \dots x'_k$ for some $k \geq 1$ be the decomposition of X.*

1. *X achieves the lower bound from Lemma 6.19(2), if and only if*
   (a) *for each lower component x'_{i_0} of X, $3 \leq i_0 \leq k-2$ and both x'_{i_0-2} and x'_{i_0+2} are lower nick letters, and*

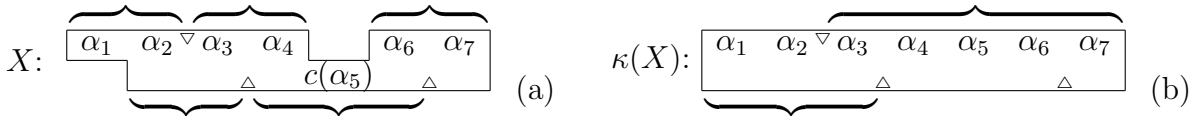


Figure 6.7: The formal DNA molecule from Example 6.24, which achieves the lower bound from Lemma 6.19(2), but does not achieve the lower bound from Lemma 6.19(4). (a) The original formal DNA molecule X, with two single-stranded components, $B_{\uparrow}(X) = 3$ and $B_{\downarrow}(X) = 2$. The upper component is not enclosed by two upper nick letters. (b) $\kappa(X)$, with $B_{\uparrow}(\kappa(X)) = 1$ and $B_{\downarrow}(\kappa(X)) = 1$.

   (b) for each upper component x'_{i_0} of X,
      - *if $i_0 \geq 3$, then x'_{i_0-2} is an upper nick letter, and*
      - *if $i_0 \leq k-2$, then x'_{i_0+2} is an upper nick letter.*

2. *X achieves the lower bound from Lemma 6.19(4), if and only if*

   (a) for each upper component x'_{i_0} of X, $3 \leq i_0 \leq k-2$ and both x'_{i_0-2} and x'_{i_0+2} are upper nick letters, and

   (b) for each lower component x'_{i_0} of X,
      - *if $i_0 \geq 3$, then x'_{i_0-2} is a lower nick letter, and*
      - *if $i_0 \leq k-2$, then x'_{i_0+2} is a lower nick letter.*

3. *X achieves the lower bounds from both Lemma 6.19(2) and Lemma 6.19(4), if and only if*

   (a) for each lower component x'_{i_0} of X, $3 \leq i_0 \leq k-2$ and both x'_{i_0-2} and x'_{i_0+2} are lower nick letters, and

   (b) for each upper component x'_{i_0} of X, $3 \leq i_0 \leq k-2$ and both x'_{i_0-2} and x'_{i_0+2} are upper nick letters.

Note that, if x'_{i_0} with $i_0 \geq 3$ is a single-stranded component and x'_{i_0-2} is a nick letter, then in fact $i_0 \geq 4$, because (the formal DNA molecule) X cannot start with a nick letter. Analogously, if x'_{i_0} with $i_0 \leq k-2$ is a single-stranded component and x'_{i_0+2} is a nick letter, then in fact $i_0 \leq k-3$.

Example 6.24 The formal DNA molecule X depicted in Figure 6.7(a) has $r = 2$ single-stranded components: an upper component and a lower component. The lower component is enclosed by two lower nick letters, but the upper component is not preceded by an upper nick letter. Figure 6.7(b) shows $\kappa(X)$. We have $B_{\uparrow}(\kappa(X)) = 1 = 3 - 2 = B_{\uparrow}(X) - r$ and $B_{\downarrow}(\kappa(X)) = 1 > 2 - 2 = B_{\downarrow}(X) - r$. ■

Proof: Recall that if X is not double-complete, then by Lemma 6.9(1), X can be considered as an alternating sequence of all its primitive $\uparrow$-blocks and all its primitive $\downarrow$-blocks.

1. $\Longrightarrow$ Assume that X does not satisfy Condition 1a or Condition 1b (or both). There is at least one single-stranded component that is responsible for the violation of the condition concerned. Let $r \geq 1$ be the number of single-stranded components of X.

   First, we assume that X does not satisfy Condition 1a. Then X has a lower component x'_{i_0}, such that

- either $i_0 \leq 2$,
- or $i_0 \geq k-1$,
- or $3 \leq i_0 \leq k-2$ and either x'_{i_0-2} or x'_{i_0+2} (or both) is not a lower nick letter.

Let X_1 be the primitive $\downarrow$-block of x'_{i_0}. We consider several cases:

- If x'_{i_0} is not the only $\downarrow$-component of X_1, then by Lemma 6.21, X does not achieve the lower bound from Lemma 6.19(2).
- If $B_\uparrow(X) = 0$, i.e., if X does not have any primitive $\uparrow$-block, then $B_\uparrow(X) - r \leq -1$ and X certainly does not achieve the lower bound from Lemma 6.19(2).
- Now assume that x'_{i_0} is the only $\downarrow$-component of X_1 and that $B_\uparrow(X) > 0$. Let X' be the formal DNA molecule that we obtain from X when we complement x'_{i_0}. Clearly, the number of single-stranded components of X' is $r-1$, and $\kappa(X) = \kappa(X')$.

  If $i_0 \leq 2$, then x'_{i_0} is the first non-double component of X. Because, by definition, each primitive $\uparrow$-block contains at least one $\uparrow$-component, the primitive $\downarrow$-block X_1 is not preceded in X by a primitive $\uparrow$-block. By assumption, $B_\uparrow(X) > 0$. Hence, X_1 must be succeeded by a primitive $\uparrow$-block X_2, which starts with the double component x'_{i_0+1}. When we complement the lower component x'_{i_0}, we lose the primitive $\downarrow$-block X_1, and the first (double) component of the primitive $\uparrow$-block X_2 is simply extended to the left. The number of primitive $\uparrow$-blocks remains the same: $B_\uparrow(X') = B_\uparrow(X)$. Now, when we apply Lemma 6.19(2) to X', we find:

  $$B_\uparrow(\kappa(X)) = B_\uparrow(\kappa(X')) \geq B_\uparrow(X') - (r-1) = B_\uparrow(X) - (r-1).$$

  In this case, X does not achieve the lower bound from Lemma 6.19(2).

  If $i_0 \geq k-1$, then in a completely analogous way, we come to the same conclusion.

  Finally, we assume that $3 \leq i_0 \leq k-2$. Then either x'_{i_0-2} or x'_{i_0+2} (or both) is not a lower nick letter. Without loss of generality, assume that the component x'_{i_0-2} is not a lower nick letter. By Lemma 3.7, both x'_{i_0-1} and x'_{i_0+1} are double components of X. Because x'_{i_0} is the only $\downarrow$-component of X_1, $X_1 = x'_{i_0-1}x'_{i_0}x'_{i_0+1}$, and both x'_{i_0-2} and x'_{i_0+2} are $\uparrow$-components. Now, x'_{i_0-2} must be an upper component, which is part of a primitive $\uparrow$-block X_0, ending with the double component x'_{i_0-1}. X_1 is succeeded in X by a primitive $\uparrow$-block X_2, which starts with the double component x'_{i_0+1}. The existence of the upper component x'_{i_0-2} implies that $r \geq 2$.

  When we complement the lower component x'_{i_0}, we obtain an extended double component $x'_{i_0-1}\kappa(x'_{i_0})x'_{i_0+1}$. Thus, we lose the primitive $\downarrow$-block X_1, and the two primitive $\uparrow$-blocks X_0 and X_2 form one 'large' primitive $\uparrow$-block X'_{02} of the resulting formal DNA molecule X'. Hence, $B_\uparrow(X') = B_\uparrow(X) - 1$. We now consider the upper component x'_{i_0-2} of X'. Its primitive $\uparrow$-block X'_{02} in X' contains at least one more $\uparrow$-component, viz x'_{i_0+2}. By Lemma 6.21, X' does not achieve the lower bound from Lemma 6.19(2):

  $$B_\uparrow(\kappa(X')) \geq B_\uparrow(X') - (r-1) + 1.$$

Consequently,

$$B_\uparrow(\kappa(X)) = B_\uparrow(\kappa(X')) \geq B_\uparrow(X') - (r-1) + 1 = B_\uparrow(X) - (r-1).$$

We conclude that also in this case, X does not achieve the lower bound from Lemma 6.19(2).

Now, we assume that X does satisfy Condition 1a and hence does not satisfy Condition 1b. Then X has an upper component x'_{i_0}, such that either $i_0 \geq 3$ and x'_{i_0-2} is not an upper nick letter, or $i_0 \leq k-2$ and x'_{i_0+2} is not an upper nick letter. Without loss of generality, we assume that $i_0 \geq 3$ and x'_{i_0-2} is not an upper nick letter. Let X_1 be the primitive $\uparrow$-block that x'_{i_0} is part of.

Because X satisfies Condition 1a and x'_{i_0} is not a lower nick letter, x'_{i_0-2} cannot be a lower component. Hence, it must be either an upper component or a lower nick letter. In both cases, it is an $\uparrow$-component of X, which implies that x'_{i_0} is not the only $\uparrow$-component of X_1. Again by Lemma 6.21, X does not achieve the lower bound from Lemma 6.19(2).

$\Longleftarrow$ By induction on the number r of single-stranded components of X.

- If $r = 0$, then X trivially satisfies Conditions 1a and 1b from the claim. On the other hand, in this case, the function κ has no effect on X: $\kappa(X) = X$. Hence, X also trivially achieves the lower bound from Lemma 6.19(2):

  $$B_\uparrow(\kappa(X)) = B_\uparrow(X) = B_\uparrow(X) - r.$$

- Let $r \geq 0$, and suppose that the lower bound is achieved by each formal DNA molecule satisfying Conditions 1a and 1b and containing r single-stranded components (induction hypothesis). Now let X be a formal DNA molecule that satisfies the two conditions and contains $r+1$ single-stranded components.

  Let x'_{i_0} with $1 \leq i_0 \leq k$ be an arbitrary single-stranded component of X, and let X' be the formal DNA molecule that results after complementing x'_{i_0}. We prove that $B_\uparrow(X') = B_\uparrow(X) - 1$.

  First, we assume that x'_{i_0} is a lower component. By Condition 1a, $3 \leq i_0 \leq k-2$ and x'_{i_0-2} and x'_{i_0+2} are lower nick letters, i.e., $\uparrow$-components. As before, both x'_{i_0-1} and x'_{i_0+1} are double components of X, and the primitive $\downarrow$-block containing x'_{i_0} is $X_1 = x'_{i_0-1}x'_{i_0}x'_{i_0+1}$.

  In X', the primitive $\downarrow$-block X_1 has been replaced by the double component $x'_{i_0-1}\kappa(x'_{i_0})x'_{i_0+1}$. Further, the two primitive $\uparrow$-blocks containing the lower nick letters x'_{i_0-2} and x'_{i_0+2}, respectively, have merged into one. Consequently, in this case, $B_\uparrow(X') = B_\uparrow(X) - 1$.

  Now, we assume that x'_{i_0} is an upper component. Let X_1 be the primitive $\uparrow$-block of x'_{i_0}. By Condition 1b, either $i_0 \leq 2$, or $i_0 \geq 3$ and x'_{i_0-2} is an upper nick letter. In both cases, x'_{i_0} is the first $\uparrow$-component of X_1. In an analogous way, we can find that x'_{i_0} is the last, and thus the only $\uparrow$-component of X_1. Besides x'_{i_0}, X_1 may only contain one or two double components.

  When we complement x'_{i_0}, the primitive $\uparrow$-block X_1 turns into a double component of X'. Other primitive $\uparrow$-blocks of X are not affected by the complementation. Also in this case, $B_\uparrow(X') = B_\uparrow(X) - 1$.

Before we can apply the induction hypothesis to X', we must verify that it satisfies Conditions 1a and 1b. For that purpose, we examine the single-stranded components of X'. Obviously, these are the single-stranded components of X different from x'_{i_0}. We no longer assume that x'_{i_0} is a lower component of X or that it is an upper component of X. It may be either of the two.

Let x'_{i_1} with $3 \leq i_1 \leq k-2$ and $i_1 \neq i_0$ be an arbitrary lower component of X different from x'_{i_0}. By assumption, x'_{i_1-2} and x'_{i_1+2} are lower nick letters. Neither these lower nick letters, nor the double components x'_{i_1-1} and x'_{i_1+1} or the lower component x'_{i_1} itself are affected by the complementation of x'_{i_0}. They occur in X' like they do in X. Hence, X' satisfies Condition 1a.

Now, let x'_{i_1} with $1 \leq i_1 \leq k$ and $i_1 \neq i_0$ be an arbitrary upper component of X different from x'_{i_0}. This upper component also occurs in X'. If $i_1 \geq 3$, then x'_{i_1-1} is a double component and by assumption, x'_{i_1-2} is an upper nick letter. These components are not affected when we complement x'_{i_0}. Analogously, if $i_1 \leq k-2$, then the double component x'_{i_1+1} and the upper nick letter x'_{i_1+2} are not affected. Obviously, the complementation does not introduce components before x'_1 (relevant if $i_1 \leq 2$) or after x'_k (relevant if $i_1 \geq k-1$). Hence, X' satisfies Condition 1b, just like X.

Because X' has r single-stranded components, we can apply the induction hypothesis to it. When we observe that $\kappa(X) = \kappa(X')$, we find

$$B_\uparrow(\kappa(X)) = B_\uparrow(\kappa(X')) = B_\uparrow(X') - r = B_\uparrow(X) - (r+1).$$

We conclude that X achieves the lower bound from Lemma 6.19(2).

2. The proof of this claim is analogous to that of the previous claim.

3. The condition on the lower components from Claim 1 implies the condition on the lower components from Claim 2, and conversely for the conditions on the upper components. Hence, Conditions 1a, 1b, 2a and 2b (together) are equivalent to Conditions 1a and 2a (together), which are equal to Conditions 3a and 3b. Now, the claim follows immediately from the other two claims.

□

We can now conclude that many formal DNA molecules that achieve either of the lower bounds from Lemma 6.19(2) and (4) are not expressible:

Corollary 6.25 *Let X be a formal DNA molecule containing at least one upper component and at least one lower component. If X achieves either the lower bound from Lemma 6.19(2), or the one from Lemma 6.19(4) (or both), then X is not expressible.*

Proof: Let $x'_1 \dots x'_k$ for some $k \geq 1$ be the decomposition of X. Assume that X achieves the lower bound from Lemma 6.19(2). Then X satisfies the two conditions in Lemma 6.23(1).

Because X contains at least one lower component, by Condition 1a, it also contains at least two lower nick letters. Let x'_{i_0} with $1 \leq i_0 \leq k$ be an upper component of X. By Lemma 3.7, double components and non-double components alternate in X. Because X also contains a lower component (and two lower nick letters), we must have either $i_0 \geq 3$ or $i_0 \leq k-2$ (or both). Then by Condition 1b, X contains at least one upper nick letter.

By Theorem 5.4, formal DNA molecules containing nick letters in both strands are not expressible.

The proof for the case that X achieves the lower bound from Lemma 6.19(4) is analogous. □

This concludes the aside that we started after the proof of Lemma 6.19, i.e., the analysis of formal DNA molecules that achieve the lower bounds from Lemma 6.19(2) and (4).

6.3 Lower bounds for the length of a DNA expression

In the previous section, we have analysed the values of the counting functions $B_\uparrow$, $B_\downarrow$ and $n_\updownarrow$ for arbitrary formal DNA molecules. We now study these values for molecules denoted by DNA expressions. The results are useful to determine lower bounds for the length of a DNA expression.

We first examine $\uparrow$-expressions with (exactly) two arguments. Using induction, we will later extend the results to $\uparrow$-expressions with an arbitrary number of arguments. Of course, we can find analogous results for $\downarrow$-expressions.

Recall that the effect of the operator $\uparrow$ is threefold: (1) it produces upper $\mathcal{A}$-words corresponding to arguments that are $\mathcal{N}$-words, (2) it removes nick letters from the upper strands of its arguments, and (3) it connects the upper strands of consecutive arguments. When we examine the effect of $\uparrow$ on the values of the counting functions, we must take into account the contributions of each of these three aspects.

The values of the counting functions for upper $\mathcal{A}$-words are independent of the $\mathcal{A}$-word at hand and follow immediately from the definitions. In Lemma 6.16, we already considered the effects on the counting numbers of removing the upper nick letters from a (single) formal DNA molecule. Now, we will in particular study the effects of connecting the upper strands of the formal DNA molecules corresponding to the (two) arguments of an $\uparrow$-expression.

Lemma 6.26 *Let $E = \langle\uparrow \varepsilon_1\varepsilon_2\rangle$ be an $\uparrow$-expression, where ε_1 and ε_2 are $\mathcal{N}$-words or DNA expressions. Further, let $X_1 = \mathcal{S}^+(\varepsilon_1)$, $X_2 = \mathcal{S}^+(\varepsilon_2)$ and*

$$X = \mathcal{S}(E) = \nu^+(X_1)y_1\nu^+(X_2),$$

where $y_1 = \triangle$ if both $R(X_1) \in \mathcal{A}_\pm$ and $L(X_2) \in \mathcal{A}_\pm$, and $y_1 = \lambda$ otherwise (as in Definition 4.1).

1. $B_\uparrow(X) \leq B_\uparrow(\nu^+(X_1)) + B_\uparrow(\nu^+(X_2)) + |y_1|$.
2. $B_\uparrow(X) \geq B_\uparrow(\nu^+(X_1)) + B_\uparrow(\nu^+(X_2)) - 1$.
3. $B_\downarrow(X) = B_\downarrow(\nu^+(X_1)) + B_\downarrow(\nu^+(X_2))$.
4. $n_\updownarrow(X) = n_\updownarrow(\nu^+(X_1)) + n_\updownarrow(\nu^+(X_2))$.

Note that $|y_1| = 1$ if $y_1 = \triangle$, and $|y_1| = 0$ if $y_1 = \lambda$.

Proof: We first make a remark on the $\downarrow$-components of the formal DNA molecules we consider. Because the function ν^+ removes the upper nick letters from its argument, each $\downarrow$-component of $\nu^+(X_1)$ and $\nu^+(X_2)$ (and thus of X) is in fact a lower component. For the

consistency of the terminology, we will, however, use the term ↓-component throughout the proof.

Recall again that if a formal DNA molecule is not double-complete, then by Lemma 6.9(1), it can be considered as an alternating sequence of all its primitive ↑-blocks and all its primitive ↓-blocks.

X is simply the concatenation of $\nu^+(X_1)$ and $\nu^+(X_2)$, possibly separated by a lower nick letter y_1. Clearly, none of the primitive ↑-blocks, primitive ↓-blocks and double components present in $\nu^+(X_1)$ or $\nu^+(X_2)$ is split up by this concatenation. We may have some effects, however, on the blocks and components at the border between $\nu^+(X_1)$ and $\nu^+(X_2)$.

By definition, X_1 and X_2 fit together by upper strands. Hence, both $R(X_1) \in \mathcal{A}_\pm \cup \mathcal{A}_+$ and $L(X_2) \in \mathcal{A}_\pm \cup \mathcal{A}_+$. Because, by Lemma 3.11, $R(\nu^+(X_1)) = R(X_1)$, the last component of $\nu^+(X_1)$ is either a double component or an upper component. Analogously, the first component of $\nu^+(X_2)$ is either a double component or an upper component.

Now assume that $\nu^+(X_1)$ is not double-complete and ends with a primitive ↓-block. Then by Lemma 6.9(2b), the last non-double component of $\nu^+(X_1)$ is a ↓-component. This implies that the last component of $\nu^+(X_1)$ cannot be an upper component; it must be a double component.

Analogously, if $\nu^+(X_2)$ is not double-complete and starts with a primitive ↓-block, then the first component of $\nu^+(X_2)$ is a double component.

1, 2. We distinguish three cases:

• If $\nu^+(X_1)$ does not end with a primitive ↑-block, then either $\nu^+(X_1)$ is double-complete, or it ends with a primitive ↓-block. In both cases, the last component of $\nu^+(X_1)$ is a double component x'_1. If, in addition, $\nu^+(X_2)$ does not start with a primitive ↑-block, then, analogously, its first component is a double component x'_2. Consequently, $R(X_1) \in \mathcal{A}_\pm$ and $L(X_2) \in \mathcal{A}_\pm$ and $y_1 = \triangle$. This lower nick letter is part of a primitive ↑-block X_{12} of X, which also contains the double components x'_1 (from $\nu^+(X_1)$) and x'_2 (from $\nu^+(X_2)$).

Each ↑-component of $\nu^+(X_1)$ (if any) is separated from y_1 in X by at least one ↓-component. Otherwise, the last non-double component of $\nu^+(X_1)$ would be an ↑-component and, by Lemma 6.9(2b), $\nu^+(X_1)$ would end with a primitive ↑-block. Analogously, each ↑-component of $\nu^+(X_2)$ (if any) is separated from y_1 in X by at least one ↓-component. Consequently, all primitive ↑-blocks of $\nu^+(X_1)$ and $\nu^+(X_2)$ are also primitive ↑-blocks of X, and $X_{12} = x'_1 y_1 x'_2$ is an additional primitive ↑-block:

$$B_\uparrow(X) = B_\uparrow(\nu^+(X_1)) + B_\uparrow(\nu^+(X_2)) + 1.$$

• If $\nu^+(X_1)$ does not end with a primitive ↑-block, then, as before, its last component is a double component x'_1. Either this component is the only component of $\nu^+(X_1)$ (if $\nu^+(X_1)$ is double-complete), or it is preceded in $\nu^+(X_1)$ by a ↓-component (if $\nu^+(X_1)$ ends with a primitive ↓-block). Now, if $\nu^+(X_2)$ is not double-complete and starts with a primitive ↑-block, then in X, this primitive ↑-block is extended to the left by x'_1 and possibly a lower nick letter y_1. Whether $y_1 = \triangle$ or $y_1 = \lambda$ depends on the first component of $\nu^+(X_2)$. It is, however, not important for the *number* of primitive ↑-blocks. The primitive ↑-blocks of $\nu^+(X_1)$ and the other primitive ↑-blocks of $\nu^+(X_2)$ simply reappear as primitive ↑-blocks of X. Hence,

$$B_\uparrow(X) = B_\uparrow(\nu^+(X_1)) + B_\uparrow(\nu^+(X_2)).$$

end $\nu^+(X_1)$	start $\nu^+(X_2)$	y_1	$B_\uparrow(X)$
double-complete	double-complete	$\triangle$	$B_\uparrow(\nu^+(X_1)) + B_\uparrow(\nu^+(X_2)) + 1 = 1$
double-complete	prim. $\downarrow$-block	$\triangle$	$B_\uparrow(\nu^+(X_1)) + B_\uparrow(\nu^+(X_2)) + 1 = B_\uparrow(\nu^+(X_2)) + 1$
double-complete	prim. $\uparrow$-block	$\triangle$ / λ	$B_\uparrow(\nu^+(X_1)) + B_\uparrow(\nu^+(X_2)) = B_\uparrow(\nu^+(X_2))$
prim. $\downarrow$-block	double-complete	$\triangle$	$B_\uparrow(\nu^+(X_1)) + B_\uparrow(\nu^+(X_2)) + 1 = B_\uparrow(\nu^+(X_1)) + 1$
prim. $\downarrow$-block	prim. $\downarrow$-block	$\triangle$	$B_\uparrow(\nu^+(X_1)) + B_\uparrow(\nu^+(X_2)) + 1$
prim. $\downarrow$-block	prim. $\uparrow$-block	$\triangle$ / λ	$B_\uparrow(\nu^+(X_1)) + B_\uparrow(\nu^+(X_2))$
prim. $\uparrow$-block	double-complete	$\triangle$ / λ	$B_\uparrow(\nu^+(X_1)) + B_\uparrow(\nu^+(X_2)) = B_\uparrow(\nu^+(X_1))$
prim. $\uparrow$-block	prim. $\downarrow$-block	$\triangle$ / λ	$B_\uparrow(\nu^+(X_1)) + B_\uparrow(\nu^+(X_2))$
prim. $\uparrow$-block	prim. $\uparrow$-block	$\triangle$ / λ	$B_\uparrow(\nu^+(X_1)) + B_\uparrow(\nu^+(X_2)) - 1$

Table 6.1: Number of primitive $\uparrow$-blocks of a formal DNA molecule $X = \mathcal{S}(\langle\uparrow \varepsilon_1\varepsilon_2\rangle) = \nu^+(X_1)y_1\nu^+(X_2)$ for all possible combinations of $\nu^+(X_1)$ and $\nu^+(X_2)$. The formal DNA molecule $\nu^+(X_1)$ either is double-complete, or ends with a primitive $\downarrow$-block or a primitive $\uparrow$-block. Similarly, $\nu^+(X_2)$ either is double-complete, or starts with a primitive $\downarrow$-block or a primitive $\uparrow$-block (see the proof of Lemma 6.26(1) and (2)).

We obtain, of course, the same equality, if $\nu^+(X_1)$ ends with a primitive $\uparrow$-block and $\nu^+(X_2)$ does not start with a primitive $\uparrow$-block.

• If both $\nu^+(X_1)$ ends with a primitive $\uparrow$-block and $\nu^+(X_2)$ starts with a primitive $\uparrow$-block, then these two primitive $\uparrow$-blocks form one primitive $\uparrow$-block in X. Again, it does not matter if $y_1 = \triangle$ or $y_1 = \lambda$. A lower nick letter, which is an $\uparrow$-component, would fit perfectly into the combined primitive $\uparrow$-block. The other primitive $\uparrow$-blocks of $\nu^+(X_1)$ and $\nu^+(X_2)$ are not affected. In this case, we lose one primitive $\uparrow$-block:

$$B_\uparrow(X) = B_\uparrow(\nu^+(X_1)) + B_\uparrow(\nu^+(X_2)) - 1.$$

We have summarized the possibilities in Table 6.1. In all cases, $B_\uparrow(X)$ satisfies the inequalities in Claims 1 and 2.

3. In principle, the concatenation of two formal DNA molecules may cause a decrease of the total number of primitive $\downarrow$-blocks. When we concatenate a formal DNA molecule ending with a primitive $\downarrow$-block and a formal DNA molecule starting with a primitive $\downarrow$-block, these two primitive $\downarrow$-blocks merge into one.

   However, if both $\nu^+(X_1)$ ends with a primitive $\downarrow$-block X_{11} and $\nu^+(X_2)$ starts with a primitive $\downarrow$-block X_{21}, then both the last double component of $\nu^+(X_1)$ and the first double component of $\nu^+(X_2)$ are double components. Hence, $R(X_1) \in \mathcal{A}_\pm$ and $L(X_2) \in \mathcal{A}_\pm$. Then the primitive $\downarrow$-blocks X_{11} and X_{21} are separated in X by the lower nick letter y_1, which is an $\uparrow$-component. They do not merge into one and there is a 1–1 correspondence between the primitive $\downarrow$-blocks of $\nu^+(X_1)$ and $\nu^+(X_2)$ and the (same) primitive $\downarrow$-blocks of X:

$$B_\downarrow(X) = B_\downarrow(\nu^+(X_1)) + B_\downarrow(\nu^+(X_2)).$$

It is easily verified that this 1–1 correspondence certainly exists, if either $\nu^+(X_1)$ does not end with a primitive $\downarrow$-block, or $\nu^+(X_2)$ does not start with a primitive $\downarrow$-block (or both).

4. If both the last component of $\nu^+(X_1)$ and the first component of $\nu^+(X_2)$ are double components, then $R(X_1) \in \mathcal{A}_\pm$ and $L(X_2) \in \mathcal{A}_\pm$. By definition, the two double components are separated in X by the lower nick letter y_1. Hence, there is a 1–1 correspondence between the double components of $\nu^+(X_1)$ and $\nu^+(X_2)$ and the (same) double components of X:

$$n_{\updownarrow}(X) = n_{\updownarrow}(\nu^+(X_1)) + n_{\updownarrow}(\nu^+(X_2)).$$

It is easily verified that this 1–1 correspondence certainly exists, if either the last component of $\nu^+(X_1)$, or the first component of $\nu^+(X_2)$ (or both) is not a double component.

□

We now consider arbitrary DNA expressions.

Lemma 6.27 *Let E be a DNA expression, and let $X = \mathcal{S}(E)$.*

1. *If $E = \langle\uparrow \varepsilon_1 \dots \varepsilon_n\rangle$, where $n \geq 1$ and the arguments $\varepsilon_1, \dots, \varepsilon_n$ are $\mathcal{N}$-words and DNA expressions, then let, for $i = 1, \dots, n$, $X_i = \mathcal{S}^+(\varepsilon_i)$.*

$$\begin{aligned}
B_\uparrow(X) &\leq B_\uparrow(X_1) + \dots + B_\uparrow(X_n) + \#_\triangle(X) - \#_\triangle(X_1) - \dots - \#_\triangle(X_n), && (6.2)\\
B_\uparrow(X) &\geq B_\uparrow(X_1) + \dots + B_\uparrow(X_n) - (n-1) - \#_\triangledown(X_1) - \dots - \#_\triangledown(X_n), && (6.3)\\
B_\downarrow(X) &\leq B_\downarrow(X_1) + \dots + B_\downarrow(X_n), && (6.4)\\
B_\downarrow(X) &\geq B_\downarrow(X_1) + \dots + B_\downarrow(X_n) - \#_\triangledown(X_1) - \dots - \#_\triangledown(X_n) \quad \textit{and} && (6.5)\\
n_{\updownarrow}(X) &= n_{\updownarrow}(X_1) + \dots + n_{\updownarrow}(X_n) - \#_\triangledown(X_1) - \dots - \#_\triangledown(X_n). && (6.6)
\end{aligned}$$

2. *If $E = \langle\downarrow \varepsilon_1 \dots \varepsilon_n\rangle$, where $n \geq 1$ and the arguments $\varepsilon_1, \dots, \varepsilon_n$ are $\mathcal{N}$-words and DNA expressions, then let, for $i = 1, \dots, n$, $X_i = \mathcal{S}^-(\varepsilon_i)$.*

$$\begin{aligned}
B_\downarrow(X) &\leq B_\downarrow(X_1) + \dots + B_\downarrow(X_n) + \#_\triangledown(X) - \#_\triangledown(X_1) - \dots - \#_\triangledown(X_n),\\
B_\downarrow(X) &\geq B_\downarrow(X_1) + \dots + B_\downarrow(X_n) - (n-1) - \#_\triangle(X_1) - \dots - \#_\triangle(X_n),\\
B_\uparrow(X) &\leq B_\uparrow(X_1) + \dots + B_\uparrow(X_n),\\
B_\uparrow(X) &\geq B_\uparrow(X_1) + \dots + B_\uparrow(X_n) - \#_\triangle(X_1) - \dots - \#_\triangle(X_n) \quad \textit{and}\\
n_{\updownarrow}(X) &= n_{\updownarrow}(X_1) + \dots + n_{\updownarrow}(X_n) - \#_\triangle(X_1) - \dots - \#_\triangle(X_n).
\end{aligned}$$

3. *If $E = \langle\updownarrow E_1\rangle$ for a DNA expression E_1, then let $X_1 = \mathcal{S}(E_1)$ and let $r \geq 0$ be the number of single-stranded components of X_1.*

$$\begin{aligned}
B_\uparrow(X) &\leq B_\uparrow(X_1), && (6.7)\\
B_\uparrow(X) &\geq B_\uparrow(X_1) - r, && (6.8)\\
B_\downarrow(X) &\leq B_\downarrow(X_1), && (6.9)\\
B_\downarrow(X) &\geq B_\downarrow(X_1) - r, && (6.10)\\
n_{\updownarrow}(X) &\leq n_{\updownarrow}(X_1) + 2 - r, && (6.11)\\
n_{\updownarrow}(X) &\leq n_{\updownarrow}(X_1) + 1 \quad \textit{and} && (6.12)\\
n_{\updownarrow}(X) &\geq n_{\updownarrow}(X_1) - r. && (6.13)
\end{aligned}$$

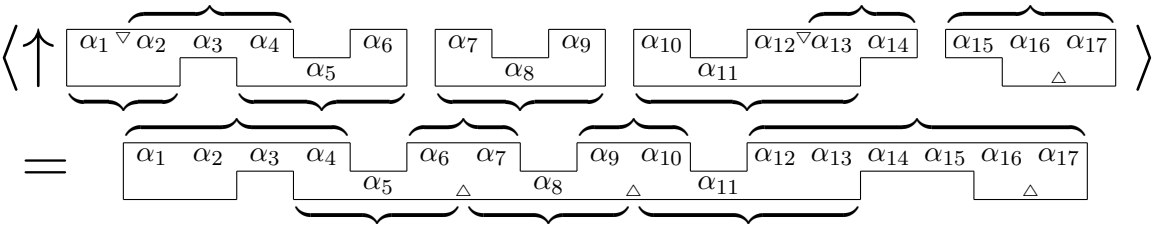


Figure 6.8: Pictorial representation of the ↑-expression E from Example 6.28, for which the values of the three counting functions are calculated. The primitive ↑-blocks and the primitive ↓-blocks of the formal DNA molecules involved are also indicated.

Example 6.28 As an illustration of Claim 1, consider

$$E = \langle\uparrow\ \langle\downarrow\ \langle\updownarrow\,\alpha_1\rangle\ \langle\uparrow\ \langle\updownarrow\,\alpha_2\rangle\,\alpha_3\,\langle\updownarrow\,\alpha_4\rangle\rangle\,\alpha_5\,\langle\updownarrow\,\alpha_6\rangle\rangle\ \ \langle\downarrow\ \langle\updownarrow\,\alpha_7\rangle\,\alpha_8\,\langle\updownarrow\,\alpha_9\rangle\rangle$$
$$\langle\downarrow\ \langle\updownarrow\,\alpha_{10}\rangle\,\alpha_{11}\,\langle\updownarrow\,\alpha_{12}\rangle\ \langle\uparrow\ \langle\updownarrow\,\alpha_{13}\rangle\,\alpha_{14}\rangle\rangle\ \ \langle\uparrow\,\alpha_{15}\,\langle\updownarrow\,\alpha_{16}\rangle\,\langle\updownarrow\,\alpha_{17}\rangle\rangle\ \rangle\,.$$

Here $n = 4$, the arguments $\varepsilon_1, \ldots, \varepsilon_4$ have been indicated using additional white space, and

$$\begin{aligned}
X_1 &= \tbinom{\alpha_1}{c(\alpha_1)} {}^{\triangledown}\tbinom{\alpha_2}{c(\alpha_2)} \tbinom{\alpha_3}{-} \tbinom{\alpha_4}{c(\alpha_4)} \tbinom{-}{\alpha_5} \tbinom{\alpha_6}{c(\alpha_6)},\\
X_2 &= \tbinom{\alpha_7}{c(\alpha_7)} \tbinom{-}{\alpha_8} \tbinom{\alpha_9}{c(\alpha_9)},\\
X_3 &= \tbinom{\alpha_{10}}{c(\alpha_{10})} \tbinom{-}{\alpha_{11}} \tbinom{\alpha_{12}}{c(\alpha_{12})} {}^{\triangledown}\tbinom{\alpha_{13}}{c(\alpha_{13})} \tbinom{\alpha_{14}}{-}, \text{ and}\\
X_4 &= \tbinom{\alpha_{15}}{-} \tbinom{\alpha_{16}}{c(\alpha_{16})} {}_{\triangle}\tbinom{\alpha_{17}}{c(\alpha_{17})}.
\end{aligned}$$

Then

$$X = \mathcal{S}(E) = \tbinom{\alpha_1\alpha_2}{c(\alpha_1\alpha_2)} \tbinom{\alpha_3}{-} \tbinom{\alpha_4}{c(\alpha_4)} \tbinom{-}{\alpha_5} \tbinom{\alpha_6}{c(\alpha_6)} {}_{\triangle}\tbinom{\alpha_7}{c(\alpha_7)} \tbinom{-}{\alpha_8} \cdot \tbinom{\alpha_9}{c(\alpha_9)} {}_{\triangle}\tbinom{\alpha_{10}}{c(\alpha_{10})} \tbinom{-}{\alpha_{11}} \tbinom{\alpha_{12}\alpha_{13}}{c(\alpha_{12}\alpha_{13})} \tbinom{\alpha_{14}\alpha_{15}}{-} \tbinom{\alpha_{16}}{c(\alpha_{16})} {}_{\triangle}\tbinom{\alpha_{17}}{c(\alpha_{17})}$$

and

$$\begin{aligned}
B_\uparrow(X) = 4 &< (1+0+1+1)+3-(0+0+0+1),\\
B_\uparrow(X) = 4 &> (1+0+1+1)-3-(1+0+1+0),\\
B_\downarrow(X) = 3 &< 2+1+1+0,\\
B_\downarrow(X) = 3 &> (2+1+1+0)-(1+0+1+0),\\
n_\updownarrow(X) = 9 &= (4+2+3+2)-(1+0+1+0).
\end{aligned}$$

This example is depicted in Figure 6.8. ■

Note that for ↑-expressions as described in Claim 1, the inequality

$$B_\uparrow(X) \leq B_\uparrow(X_1) + \cdots + B_\uparrow(X_n)$$

does not hold in general. Lower nick letters added between the arguments of ↑ may introduce new primitive ↑-blocks. Indeed, for the ↑-expression we considered above, $B_\uparrow(X) = 4$, whereas $B_\uparrow(X_1) + B_\uparrow(X_2) + B_\uparrow(X_3) + B_\uparrow(X_4) = 3$. The difference $\#_{\triangle}(X) - \#_{\triangle}(X_1) - \cdots - \#_{\triangle}(X_n)$ in inequality (6.2) accounts for the lower nick letters added.

Note also that in Claim 3, we do not consider ↕-expressions of the form $\langle\updownarrow\,\alpha_1\rangle$ for an $\mathcal{N}$-word α_1. For such ↕-expressions, however, the values of $B_\uparrow$, $B_\downarrow$ and $n_\updownarrow$ are trivial: if

$X = \mathcal{S}(\langle\updownarrow \alpha_1\rangle) = \binom{\alpha_1}{c(\alpha_1)}$, then $B_\uparrow(X) = B_\downarrow(X) = 0$ and $n_\updownarrow(X) = 1$.

Proof of Lemma 6.27:

1. Let E be an $\uparrow$-expression as described in the claim. We prove the five equations by induction on n.

   - If $n = 1$, then $E = \langle\uparrow \varepsilon_1\rangle$ for an $\mathcal{N}$-word or a DNA expression ε_1 and the five equations we have to prove reduce to:

     $$\begin{aligned} B_\uparrow(X) &\leq B_\uparrow(X_1) + \#_\triangle(X) - \#_\triangle(X_1), \\ B_\uparrow(X) &\geq B_\uparrow(X_1) - 0 - \#_\triangledown(X_1), \\ B_\downarrow(X) &\leq B_\downarrow(X_1), \\ B_\downarrow(X) &\geq B_\downarrow(X_1) - \#_\triangledown(X_1) \qquad \text{and} \\ n_\updownarrow(X) &= n_\updownarrow(X_1) - \#_\triangledown(X_1). \end{aligned}$$

     If ε_1 is an $\mathcal{N}$-word α_1, then $E = \langle\uparrow \alpha_1\rangle$ and by definition, $X = X_1 = \binom{\alpha_1}{-}$. Hence, $\#_\triangledown(X_1) = 0$ and the five equations are trivially valid, with $B_\uparrow(X) = 1$, $\#_\triangle(X) = 0$, $B_\downarrow(X) = 0$ and $n_\updownarrow(X) = 0$.
     If, on the other hand, ε_1 is a DNA expression E_1, then $E = \langle\uparrow E_1\rangle$ and by definition, $X = \nu^+(X_1)$. Because the function ν^+ neither introduces nor removes lower nick letters (hence, $\#_\triangle(X) = \#_\triangle(X_1)$), the five equations are just special cases of the ones in Lemma 6.16. Whereas, in Lemma 6.16, we considered an arbitrary formal DNA molecule X, we have an expressible formal DNA molecule X_1 here.

   - Let $n \geq 1$ and suppose that equations (6.2)–(6.6) hold for all $\uparrow$-expressions with n arguments (induction hypothesis). Now let E be an arbitrary $\uparrow$-expression with $n+1$ arguments: $E = \langle\uparrow \varepsilon_1 \ldots \varepsilon_n \varepsilon_{n+1}\rangle$ for $\mathcal{N}$-words and DNA expressions $\varepsilon_1, \ldots, \varepsilon_n, \varepsilon_{n+1}$.
     By Lemma 5.10,

     $$E \equiv \langle\uparrow \langle\uparrow \varepsilon_1 \ldots \varepsilon_n\rangle \varepsilon_{n+1}\rangle\,,$$

     i.e., $X = \mathcal{S}(E) = \mathcal{S}(\langle\uparrow \langle\uparrow \varepsilon_1 \ldots \varepsilon_n\rangle \varepsilon_{n+1}\rangle)$. Let $X' = \mathcal{S}(\langle\uparrow \varepsilon_1 \ldots \varepsilon_n\rangle)$. By definition,

     $$X = \nu^+(X') y_1 \nu^+(X_{n+1}),$$

     where $y_1 = \triangle$ if $R(X'), L(X_{n+1}) \in \mathcal{A}_\pm$ and $y_1 = \lambda$ otherwise. By Lemma 5.1(1), the semantics of an $\uparrow$-expression does not contain upper nick letters. Hence $\nu^+(X') = X'$. Because the function ν^+ neither introduces, nor removes lower nick letters,

     $$\#_\triangle(X) = \#_\triangle(X') + |y_1| + \#_\triangle(X_{n+1}),$$

     which can be rewritten as

     $$|y_1| = \#_\triangle(X) - \#_\triangle(X') - \#_\triangle(X_{n+1}).$$

     We can now make the following derivation:

     $$\begin{aligned} B_\uparrow(X) &\leq B_\uparrow(\nu^+(X')) + B_\uparrow(\nu^+(X_{n+1})) + |y_1| \\ &= B_\uparrow(X') + B_\uparrow(\nu^+(X_{n+1})) + \#_\triangle(X) - \#_\triangle(X') - \#_\triangle(X_{n+1}) \end{aligned}$$

$$\begin{aligned}
&\leq B_\uparrow(X') + B_\uparrow(X_{n+1}) + \#_\triangle(X) - \#_\triangle(X') - \#_\triangle(X_{n+1}) \\
&\leq B_\uparrow(X_1) + \cdots + B_\uparrow(X_n) + \#_\triangle(X') - \#_\triangle(X_1) - \cdots - \#_\triangle(X_n) \\
&\quad + B_\uparrow(X_{n+1}) + \#_\triangle(X) - \#_\triangle(X') - \#_\triangle(X_{n+1}) \\
&= B_\uparrow(X_1) + \cdots + B_\uparrow(X_n) + B_\uparrow(X_{n+1}) \\
&\quad + \#_\triangle(X) - \#_\triangle(X_1) - \cdots - \#_\triangle(X_n) - \#_\triangle(X_{n+1}).
\end{aligned}$$

Here, the three inequalities follow from Lemma 6.26(1), Lemma 6.16(1) and the induction hypothesis (in particular, inequality (6.2) for $X' = \mathcal{S}(\langle\uparrow \varepsilon_1 \ldots \varepsilon_n\rangle)$), respectively. We have thus obtained inequality (6.2) for $X = \mathcal{S}(E)$.

In a similar way, using the other claims from Lemma 6.26 and Lemma 6.16 and the other equations from the induction hypothesis, we find inequality (6.3) for $X = \mathcal{S}(E)$:

$$\begin{aligned}
B_\uparrow(X) &\geq B_\uparrow(\nu^+(X')) + B_\uparrow(\nu^+(X_{n+1})) - 1 \\
&= B_\uparrow(X') + B_\uparrow(\nu^+(X_{n+1})) - 1 \\
&\geq B_\uparrow(X') + B_\uparrow(X_{n+1}) - \#_\triangledown(X_{n+1}) - 1 \\
&\geq B_\uparrow(X_1) + \cdots + B_\uparrow(X_n) - (n-1) - \#_\triangledown(X_1) - \cdots - \#_\triangledown(X_n) \\
&\quad + B_\uparrow(X_{n+1}) - \#_\triangledown(X_{n+1}) - 1 \\
&= B_\uparrow(X_1) + \cdots + B_\uparrow(X_n) + B_\uparrow(X_{n+1}) - n \\
&\quad - \#_\triangledown(X_1) - \cdots - \#_\triangledown(X_n) - \#_\triangledown(X_{n+1}),
\end{aligned}$$

inequality (6.4) for $X = \mathcal{S}(E)$:

$$\begin{aligned}
B_\downarrow(X) &= B_\downarrow(\nu^+(X')) + B_\downarrow(\nu^+(X_{n+1})) \\
&= B_\downarrow(X') + B_\downarrow(\nu^+(X_{n+1})) \\
&\leq B_\downarrow(X') + B_\downarrow(X_{n+1}) \\
&\leq B_\downarrow(X_1) + \cdots + B_\downarrow(X_n) + B_\downarrow(X_{n+1}),
\end{aligned}$$

inequality (6.5) for $X = \mathcal{S}(E)$:

$$\begin{aligned}
B_\downarrow(X) &= B_\downarrow(\nu^+(X')) + B_\downarrow(\nu^+(X_{n+1})) \\
&= B_\downarrow(X') + B_\downarrow(\nu^+(X_{n+1})) \\
&\geq B_\downarrow(X') + B_\downarrow(X_{n+1}) - \#_\triangledown(X_{n+1}) \\
&\geq B_\downarrow(X_1) + \cdots + B_\downarrow(X_n) - \#_\triangledown(X_1) - \cdots - \#_\triangledown(X_n) \\
&\quad + B_\downarrow(X_{n+1}) - \#_\triangledown(X_{n+1}) \\
&= B_\downarrow(X_1) + \cdots + B_\downarrow(X_n) + B_\downarrow(X_{n+1}) \\
&\quad - \#_\triangledown(X_1) - \cdots - \#_\triangledown(X_n) - \#_\triangledown(X_{n+1}),
\end{aligned}$$

and equality (6.6) for $X = \mathcal{S}(E)$:

$$\begin{aligned}
n_\updownarrow(X) &= n_\updownarrow(\nu^+(X')) + n_\updownarrow(\nu^+(X_{n+1})) \\
&= n_\updownarrow(X') + n_\updownarrow(\nu^+(X_{n+1})) \\
&= n_\updownarrow(X') + n_\updownarrow(X_{n+1}) - \#_\triangledown(X_{n+1}) \\
&= n_\updownarrow(X_1) + \cdots + n_\updownarrow(X_n) - \#_\triangledown(X_1) - \cdots - \#_\triangledown(X_n) \\
&\quad + n_\updownarrow(X_{n+1}) - \#_\triangledown(X_{n+1}) \\
&= n_\updownarrow(X_1) + \cdots + n_\updownarrow(X_n) + n_\updownarrow(X_{n+1}) \\
&\quad - \#_\triangledown(X_1) - \cdots - \#_\triangledown(X_n) - \#_\triangledown(X_{n+1}).
\end{aligned}$$

We conclude that the five equations are also valid for the $\uparrow$-expression E with $n+1$ arguments.

2. The proof of this claim is analogous to that of the previous claim.

3. Let E be an $\updownarrow$-expression $\langle\updownarrow E_1\rangle$ for a DNA expression E_1, let $X_1 = \mathcal{S}(E_1)$ and let $r \geq 0$ be the number of single-stranded components of X_1. By definition, $X = \kappa(X_1)$. Now, all equations are just special cases of the ones in Lemma 6.19.

□

In the proof of Lemma 6.27(1), we observed that if $n = 1$ and ε_1 is a DNA expression E_1, equations (6.2)–(6.6) are in fact special cases of the claims from Lemma 6.16. Likewise, the proof of Lemma 6.27(3) consisted mainly of the observation that equations (6.7)–(6.13) are special cases of the claims from Lemma 6.19.

For *expressible* formal DNA molecules X, we can also walk the other direction. We can consider the claims from Lemma 6.16 as special cases of equations (6.2)–(6.6) from Lemma 6.27(1). If we take $n = 1$ and let ε_1 be a DNA expression denoting X, then Lemma 6.16 follows from the observation that $\nu^+(X) = \nu^+(\mathcal{S}(\varepsilon_1)) = \mathcal{S}(\langle\uparrow \varepsilon_1\rangle)$.

Similarly, if X is an expressible formal DNA molecule and E_1 is a DNA expression denoting X, then Lemma 6.19 follows from Lemma 6.27(3) and the observation that $\kappa(X) = \kappa(\mathcal{S}(E_1)) = \mathcal{S}(\langle\updownarrow E_1\rangle)$.

By inequalities (6.7) and (6.9) from Lemma 6.27(3), the values of the functions $B_\uparrow$ and $B_\downarrow$ do not increase when we apply the operator $\updownarrow$ to a DNA expression E_1. There exists, however, a much stronger result concerning $B_\uparrow$ and $B_\downarrow$ for $\updownarrow$-expressions:

Lemma 6.29 *Let E be an $\updownarrow$-expression, and let $X = \mathcal{S}(E)$. Then $B_\uparrow(X) + B_\downarrow(X) \leq 1$.*

Proof: Let $E = \langle\updownarrow \varepsilon_1\rangle$, where ε_1 is an $\mathcal{N}$-word or a DNA expression. By the definition of the semantics of an $\updownarrow$-expression, $X = \kappa(\mathcal{S}^+(\varepsilon_1))$.

Hence, X neither contains upper components, nor lower components. Each $\uparrow$-component of X has to be a lower nick letter and each $\downarrow$-component of X has to be an upper nick letter. By Theorem 5.4, X does not both contain lower nick letters and upper nick letters. This implies that X either does not contain any $\uparrow$-component or does not contain any $\downarrow$-component (or both). Then by definition, $B_\uparrow(X) = 0$, or $B_\downarrow(X) = 0$ (or both). Now, the claim follows from Lemma 6.12(1) and (2). □

After all this introductory work, we are ready to calculate lower bounds for the number of occurrences of the operators $\uparrow$ and $\downarrow$ and for the number of occurrences of the operator $\updownarrow$ in a DNA expression.

Theorem 6.30 *Let E be a DNA expression, and let $X = \mathcal{S}(E)$.*

1. *If E is an $\uparrow$-expression, then*

$$\begin{aligned} \#_{\uparrow,\downarrow}(E) &\geq 1 + B_\downarrow(X) \quad \textit{and} \\ \#_\updownarrow(E) &\geq n_\updownarrow(X). \end{aligned}$$

2. *If E is a $\downarrow$-expression, then*

$$\begin{aligned} \#_{\uparrow,\downarrow}(E) &\geq 1 + B_\uparrow(X) \quad \textit{and} \\ \#_\updownarrow(E) &\geq n_\updownarrow(X). \end{aligned}$$

3. If E is an $\updownarrow$-expression, then

$$\begin{aligned} \#_{\uparrow,\downarrow}(E) &\geq B_{\uparrow}(X), && (6.14)\\ \#_{\uparrow,\downarrow}(E) &\geq B_{\downarrow}(X) \quad \textit{and} && (6.15)\\ \#_{\updownarrow}(E) &\geq n_{\updownarrow}(X). \end{aligned}$$

Proof: By induction on the number p of operators occurring in E.

- If $p = 1$, then E is $\langle\uparrow \alpha_1\rangle$, $\langle\downarrow \alpha_1\rangle$ or $\langle\updownarrow \alpha_1\rangle$ for an $\mathcal{N}$-word α_1.

  If $E = \langle\uparrow \alpha_1\rangle$, then $\#_{\uparrow,\downarrow}(E) = 1$, $\#_{\updownarrow}(E) = 0$, $X = \binom{\alpha_1}{-}$ and $B_{\downarrow}(X) = n_{\updownarrow}(X) = 0$. Hence, the inequalities in Claim 1 are valid.

  If $E = \langle\downarrow \alpha_1\rangle$, then Claim 2 is applicable and the inequalities in this claim are verified analogously.

  If $E = \langle\updownarrow \alpha_1\rangle$, then $\#_{\uparrow,\downarrow}(E) = 0$, $\#_{\updownarrow}(E) = 1$, $X = \binom{\alpha_1}{c(\alpha_1)}$, $B_{\uparrow}(X) = B_{\downarrow}(X) = 0$ and $n_{\updownarrow}(X) = 1$. Indeed, these values satisfy the inequalities in Claim 3.

- Let $p \geq 1$, and suppose that the lower bounds hold for all DNA expressions containing at most p operators (induction hypothesis). Now let E be an arbitrary DNA expression that contains $p + 1$ operators. E is either an $\uparrow$-expression, or a $\downarrow$-expression or an $\updownarrow$-expression. We consider each of these cases separately.

  – If E is an $\uparrow$-expression $\langle\uparrow \varepsilon_1 \ldots \varepsilon_n\rangle$, where $n \geq 1$ and $\varepsilon_1, \ldots, \varepsilon_n$ are the arguments of E, then let for $i = 1, \ldots, n$, $X_i = \mathcal{S}^+(\varepsilon_i)$. The arguments are $\mathcal{N}$-words, $\uparrow$-expressions, $\downarrow$-expressions and $\updownarrow$-expressions.

  By definition, if an argument ε_i is an $\mathcal{N}$-word α, then $\#_{\uparrow,\downarrow}(\varepsilon_i) = \#_{\updownarrow}(\varepsilon_i) = 0$, $X_i = \mathcal{S}^+(\varepsilon_i) = \binom{\alpha}{-}$ and $B_{\downarrow}(X_i) = n_{\updownarrow}(X_i) = 0$. If, on the other hand, an argument ε_i is a DNA expression, then $X_i = \mathcal{S}^+(\varepsilon_i) = \mathcal{S}(\varepsilon_i)$. Because such an argument contains at most p operators, the induction hypothesis provides us with lower bounds for $\#_{\uparrow,\downarrow}(\varepsilon_i)$ and $\#_{\updownarrow}(\varepsilon_i)$. For $\updownarrow$-expressions ε_i, we use lower bound (6.15) for $\#_{\uparrow,\downarrow}(\varepsilon_i)$. We first consider $\#_{\uparrow,\downarrow}(E)$:

$$\begin{aligned} \#_{\uparrow,\downarrow}(E) &= 1 + \sum_{i=1}^{n} \#_{\uparrow,\downarrow}(\varepsilon_i)\\ &= 1 + \sum_{\mathcal{N}\text{-words } \varepsilon_i} \#_{\uparrow,\downarrow}(\varepsilon_i) + \sum_{\uparrow\text{-expr. } \varepsilon_i} \#_{\uparrow,\downarrow}(\varepsilon_i)\\ &\quad + \sum_{\downarrow\text{-expr. } \varepsilon_i} \#_{\uparrow,\downarrow}(\varepsilon_i) + \sum_{\updownarrow\text{-expr. } \varepsilon_i} \#_{\uparrow,\downarrow}(\varepsilon_i)\\ &\geq 1 + \sum_{\mathcal{N}\text{-words } \varepsilon_i} B_{\downarrow}(X_i) + \sum_{\uparrow\text{-expr. } \varepsilon_i} (1 + B_{\downarrow}(X_i))\\ &\quad + \sum_{\downarrow\text{-expr. } \varepsilon_i} (1 + B_{\uparrow}(X_i)) + \sum_{\updownarrow\text{-expr. } \varepsilon_i} B_{\downarrow}(X_i). \end{aligned}$$

  Obviously, the term $1 + B_{\downarrow}(X_i)$ for an $\uparrow$-expression ε_i is greater than $B_{\downarrow}(X_i)$. For the $\downarrow$-expressions ε_i, we use Lemma 6.12(2) to replace the terms $1 + B_{\uparrow}(X_i)$ by $B_{\downarrow}(X_i)$.

When subsequently, we apply inequality (6.4) from Lemma 6.27(1), we obtain the desired lower bound for $\#_{\uparrow,\downarrow}(E)$:

$$\#_{\uparrow,\downarrow}(E) \geq 1 + \sum_{i=1}^{n} B_{\downarrow}(X_i) \geq 1 + B_{\downarrow}(X).$$

The lower bound for $\#_{\updownarrow}(E)$ is easy to calculate. First, we observe that for each argument ε_i, either by definition (if ε_i is an $\mathcal{N}$-word), or by the induction hypothesis (if ε_i is a DNA expression), $\#_{\updownarrow}(\varepsilon_i) \geq n_{\updownarrow}(X_i)$. Next, we apply equality (6.6) from Lemma 6.27(1):

$$\#_{\updownarrow}(E) = \sum_{i=1}^{n} \#_{\updownarrow}(\varepsilon_i) \geq \sum_{i=1}^{n} n_{\updownarrow}(X_i) = n_{\updownarrow}(X) + \sum_{i=1}^{n} \#_{\triangledown}(X_i) \geq n_{\updownarrow}(X).$$

– If E is a $\downarrow$-expression, then the proof is analogous.

– If E is an $\updownarrow$-expression, then its only argument must be a DNA expression E_1: $E = \langle \updownarrow E_1 \rangle$. Let $X_1 = \mathcal{S}(E_1)$. Because E_1 contains p operators, we can apply the induction hypothesis to it.

If E_1 is an $\uparrow$-expression, then we additionally apply inequality (6.9) from Lemma 6.27(3) and Lemma 6.12(1):

$$\#_{\uparrow,\downarrow}(E) = \#_{\uparrow,\downarrow}(E_1) \geq 1 + B_{\downarrow}(X_1) \geq 1 + B_{\downarrow}(X) \geq B_{\uparrow}(X).$$

Note that by this series of inequalities, we have proved both inequality (6.14) and inequality (6.15) for the case that E_1 is a $\uparrow$-expression. Analogously, if E_1 is a $\downarrow$-expression, then

$$\#_{\uparrow,\downarrow}(E) = \#_{\uparrow,\downarrow}(E_1) \geq 1 + B_{\uparrow}(X_1) \geq 1 + B_{\uparrow}(X) \geq B_{\downarrow}(X).$$

Finally, if E_1 is an $\updownarrow$-expression, then $X = \kappa(X_1) = X_1$. Hence, by the induction hypothesis,

$$\begin{aligned} \#_{\uparrow,\downarrow}(E) &= \#_{\uparrow,\downarrow}(E_1) \geq B_{\uparrow}(X_1) = B_{\uparrow}(X) \quad \text{and} \\ \#_{\uparrow,\downarrow}(E) &= \#_{\uparrow,\downarrow}(E_1) \geq B_{\downarrow}(X_1) = B_{\downarrow}(X). \end{aligned}$$

To calculate a lower bound for $\#_{\updownarrow}(E)$, we do not have to distinguish different cases. By the induction hypothesis, $\#_{\updownarrow}(E_1) \geq n_{\updownarrow}(X_1)$, regardless of the outermost operator of E_1. Then by inequality (6.12) from Lemma 6.27(3),

$$\#_{\updownarrow}(E) = 1 + \#_{\updownarrow}(E_1) \geq 1 + n_{\updownarrow}(X_1) \geq n_{\updownarrow}(X).$$

□

It is only a small step from the number of operators occurring in a DNA expression to the length of that DNA expression:

Theorem 6.31 *Let E be a DNA expression, and let $X = \mathcal{S}(E)$.*

1. *If E is an $\uparrow$-expression, then $|E| \geq 3 + 3 \cdot B_{\downarrow}(X) + 3 \cdot n_{\updownarrow}(X) + |X|_{\mathcal{A}}$.*
2. *If E is a $\downarrow$-expression, then $|E| \geq 3 + 3 \cdot B_{\uparrow}(X) + 3 \cdot n_{\updownarrow}(X) + |X|_{\mathcal{A}}$.*
3. *If E is an $\updownarrow$-expression, then*

$$\begin{aligned} |E| &\geq 3 \cdot B_{\uparrow}(X) + 3 \cdot n_{\updownarrow}(X) + |X|_{\mathcal{A}} \quad \textit{and} \\ |E| &\geq 3 \cdot B_{\downarrow}(X) + 3 \cdot n_{\updownarrow}(X) + |X|_{\mathcal{A}}. \end{aligned} \tag{6.16}$$

4. *If $E = \langle \updownarrow \alpha_1 \rangle$ for an $\mathcal{N}$-word α_1, then $|E| = 3 \cdot n_{\updownarrow}(X) + |X|_{\mathcal{A}}$.*
5. *If $E = \langle \updownarrow E_1 \rangle$ for a DNA expression E_1, then $|E| \geq 3 + 3 \cdot n_{\updownarrow}(X) + |X|_{\mathcal{A}}$.*
6. *Unless $E = \langle \updownarrow \alpha_1 \rangle$ for an $\mathcal{N}$-word α_1, $|E| \geq 3 + 3 \cdot n_{\updownarrow}(X) + |X|_{\mathcal{A}}$.*

Note that the starting point for this result is a DNA expression E. Some formal DNA molecules X can only be denoted by certain types of DNA expressions. If X contains single-stranded components, then E cannot be an $\updownarrow$-expression. If X contains upper (or lower) nick letters, then E cannot be an $\uparrow$-expression ($\downarrow$-expression, respectively). Hence, given a formal DNA molecule X, some of the claims from this result may not be applicable.

In Theorem 7.42 and Theorem 7.46, we will see that the lower bounds from Claims 1 and 2 are tight. They are achieved by the shortest $\uparrow$-expressions and $\downarrow$-expressions for a given formal DNA molecule. As we will observe after the statement of Theorem 7.5, the lower bounds from Claim 3 are tight for nick free formal DNA molecules, where $B_{\uparrow}(X) = B_{\downarrow}(X) = 0$ and $n_{\updownarrow}(X) = 1$. However, in Theorem 8.19, we will find that these two lower bounds are not tight for expressible formal DNA molecules containing nick letters: there do not exist $\updownarrow$-expressions denoting such molecules that achieve these lower bounds.

Proof: Claims 1–3 follow immediately from Lemma 6.1 and Theorem 6.30.

4. If $E = \langle \updownarrow \alpha_1 \rangle$ for an $\mathcal{N}$-word α_1, then $X = \binom{\alpha_1}{c(\alpha_1)}$ and $n_{\updownarrow}(X) = 1$. Hence, both sides of the equality in the claim evaluate to $3 + |\alpha_1|$.

5. Assume that $E = \langle \updownarrow E_1 \rangle$ for a DNA expression E_1, and let $X_1 = \mathcal{S}(E_1)$. By definition, $X = \kappa(X_1)$ and hence $|X|_{\mathcal{A}} = |X_1|_{\mathcal{A}}$. We distinguish three cases.

   If E_1 is an $\uparrow$-expression, then by Claim 1 and inequality (6.12) from Lemma 6.27(3),

$$\begin{aligned} |E| &= 3 + |E_1| \\ &\geq 3 + 3 + 3 \cdot B_{\downarrow}(X_1) + 3 \cdot n_{\updownarrow}(X_1) + |X_1|_{\mathcal{A}} \\ &\geq 3 + 3 + 0 + 3 \cdot n_{\updownarrow}(X_1) + |X_1|_{\mathcal{A}} \\ &\geq 3 + 3 \cdot n_{\updownarrow}(X) + |X|_{\mathcal{A}}. \end{aligned}$$

   If E_1 is a $\downarrow$-expression. then the inequality $|E| \geq 3 + 3 \cdot n_{\updownarrow}(X) + |X|_{\mathcal{A}}$ is obtained in an analogous way.

   Finally, if E_1 is an $\updownarrow$-expression, then $X = X_1$. Hence, by Claim 3,

$$|E| = 3 + |E_1| \geq 3 + 0 + 3 \cdot n_{\updownarrow}(X_1) + |X_1|_{\mathcal{A}} = 3 + 3 \cdot n_{\updownarrow}(X) + |X|_{\mathcal{A}}.$$

   For each type of DNA expression E_1, we obtain the inequality from the claim. Hence, the claim is valid.

6. This claim follows immediately from Claims 1, 2 and 5.

□

Chapter 7

The Construction of Minimal DNA Expressions

If a formal DNA molecule is expressible, then there exist (infinitely) many DNA expressions denoting it. Theorem 6.31 provides us with lower bounds for the length of such DNA expressions. We are interested in the shortest DNA expressions for a given molecule, possibly achieving the applicable lower bound(s).

Definition 7.1 *A DNA expression E is minimal if for every DNA expression E' with $E' \equiv E$, $|E'| \geq |E|$.*

Hence, when a DNA expression is minimal, we cannot find a shorter DNA expression with the same semantics.

In principle, there may be different minimal DNA expressions for the same formal DNA molecule.

Example 7.2 Let $X = \binom{\alpha_1}{-}\binom{\alpha_2}{c(\alpha_2)}\binom{-}{\alpha_3}$. Then both $E = \langle\uparrow \alpha_1 \langle\downarrow \langle\updownarrow \alpha_2\rangle \alpha_3\rangle\rangle$ and $E' = \langle\downarrow \langle\uparrow \alpha_1 \langle\updownarrow \alpha_2\rangle\rangle \alpha_3\rangle$ denote X, and $|E| = |E'|$. It is easy to verify that E and E' achieve the lower bounds given in Theorem 6.31(1) and (2) for $\uparrow$-expressions and $\downarrow$-expressions, respectively. Hence, there do not exist shorter $\uparrow$-expressions or $\downarrow$-expressions for X. Because X contains single-stranded components, it cannot be denoted by an $\updownarrow$-expression. Consequently, E and E' are indeed minimal. ■

The following result is immediately deduced from Lemma 6.1:

Corollary 7.3 *A DNA expression E containing p operators is minimal if and only if every DNA expression E' with $E' \equiv E$ contains at least p operators.*

The next result is so natural that we will not refer to it when we use it. Nevertheless, it is good to state it explicitly:

Lemma 7.4 *A DNA expression E is minimal if and only if each DNA subexpression of E is minimal.*

Proof: Let E be an arbitrary DNA expression.

From right to left, the claim is obvious, because by definition E is a DNA subexpression of itself.

Now suppose that a DNA subexpression E^s of E is not minimal. Then there must exist a DNA expression $E^{s\prime}$ such that $E^{s\prime} \equiv E^s$ and $|E^{s\prime}| < |E^s|$. Let us substitute E^s in

E by $E^{s\prime}$. By Lemma 5.11, the resulting DNA expression E' is equivalent to E. Because $|E'| < |E|$, E cannot be minimal. □

In this chapter, we describe how to construct minimal DNA expressions for a given formal DNA molecule. In Section 7.1, we consider *nick free* formal DNA molecules. Subsequently, in Section 7.2, we extend the results to formal DNA molecules that contain nicks.

7.1 Minimal DNA expressions for a nick free formal DNA molecule

Minimal $\updownarrow$-expressions are limited to one simple type of formal DNA molecules:

Theorem 7.5 *An $\updownarrow$-expression E is minimal if and only if $E = \langle \updownarrow \alpha_1 \rangle$ for an $\mathcal{N}$-word α_1. In that case, E is the unique minimal DNA expression denoting $\mathcal{S}(E) = \binom{\alpha_1}{c(\alpha_1)}$.*

Note that, if indeed $E = \langle \updownarrow \alpha_1 \rangle$ for an $\mathcal{N}$-word α_1 and we let $X = \mathcal{S}(E) = \binom{\alpha_1}{c(\alpha_1)}$, then $B_\uparrow(X) = B_\downarrow(X) = 0$, $n_\updownarrow(X) = 1$ and

$$|E| = 3 + |\alpha_1| = 3 \cdot n_\updownarrow(X) + |X|_{\mathcal{A}}.$$

Hence, E achieves the lower bounds from Theorem 6.31(3). By Corollary 5.7, the only nick free formal DNA molecules that can be denoted by an $\updownarrow$-expression are precisely of the form $\binom{\alpha_1}{c(\alpha_1)}$ for an $\mathcal{N}$-word α_1. We can therefore say that the lower bounds from Theorem 6.31(3) are tight for nick free formal DNA molecules.

Intuitively, this result can be understood in the following way: An expression-argument of $\updownarrow$ denotes a formal DNA molecule, which may or may not contain single-stranded components. All single-stranded components are complemented by the operator $\updownarrow$. It is not efficient to first generate single-stranded components by means of the operators $\uparrow$ and $\downarrow$, and then to complement them by means of $\updownarrow$. It certainly is not efficient to apply $\updownarrow$ to an argument without single-stranded components, because then the operator has no effect at all. Consequently, an $\updownarrow$-expression $\langle \updownarrow E_1 \rangle$ for a DNA expression E_1 is not likely to be minimal.

Proof: Let $E = \langle \updownarrow \varepsilon_1 \rangle$ be an $\updownarrow$-expression, where ε_1 is an $\mathcal{N}$-word or a DNA expression, and let $X = \mathcal{S}(E)$. We consider the two possibilities for ε_1 separately.

- If ε_1 is an $\mathcal{N}$-word α_1, hence $E = \langle \updownarrow \alpha_1 \rangle$, then $X = \binom{\alpha_1}{c(\alpha_1)}$.

  Now let E' be another, equivalent DNA expression. E' is not equal to $\langle \updownarrow \alpha \rangle$ for an $\mathcal{N}$-word α, because the only $\mathcal{N}$-word α such that $\mathcal{S}(\langle \updownarrow \alpha \rangle) = \binom{\alpha_1}{c(\alpha_1)}$ is $\alpha = \alpha_1$.

  Because $n_\updownarrow(X) = 1$, Theorem 6.31(6) implies that E' is longer than E:

  $$|E'| \geq 3 + 3 \cdot 1 + |X|_{\mathcal{A}} = 6 + |\alpha_1| > 3 + |\alpha_1| = |E|.$$

  Thus, E is the unique minimal DNA expression denoting X.

- If ε_1 is a DNA expression E_1, hence $E = \langle\updownarrow E_1\rangle$, then by definition $X = \kappa(X_1)$ with $X_1 = \mathcal{S}(E_1)$. By Corollary 5.7, there exist $\mathcal{N}$-words $\alpha_1, \ldots, \alpha_m$ for some $m \geq 1$ and a nick letter $y \in \{^{\triangledown}, _{\triangle}\}$ such that

$$X = \binom{\alpha_1}{c(\alpha_1)} y \binom{\alpha_2}{c(\alpha_2)} y \ldots y \binom{\alpha_m}{c(\alpha_m)}.$$

If $m = 1$, hence $X = \binom{\alpha_1}{c(\alpha_1)}$ is nick free, then E is not minimal, because we have just observed that the unique minimal DNA expression denoting $\binom{\alpha_1}{c(\alpha_1)}$ is $\langle\updownarrow \alpha_1\rangle$.

If $m \geq 2$, then E_1 cannot be an $\updownarrow$-expression $\langle\updownarrow \alpha\rangle$ for an $\mathcal{N}$-word α, because otherwise $X = \mathcal{S}(E) = \mathcal{S}(\langle\updownarrow E_1\rangle) = \mathcal{S}(\langle\updownarrow \langle\updownarrow \alpha\rangle\rangle) = \binom{\alpha}{c(\alpha)}$ would be nick free. Hence, by Theorem 6.31(6),

$$|E| = 3 + |E_1| \geq 3 + 3 + 3 \cdot n_{\updownarrow}(X_1) + |X_1|_{\mathcal{A}}. \tag{7.1}$$

The fact that $X = \kappa(X_1)$ implies that $|X_1|_{\mathcal{A}} = |X|_{\mathcal{A}}$ and that $\#_{\triangledown,\triangle}(X_1) = \#_{\triangledown,\triangle}(X) = m - 1 \geq 1$. Combining the latter observation with Lemma 6.12(4), we find

$$n_{\updownarrow}(X_1) \geq \#_{\triangledown,\triangle}(X_1) + 1 = m - 1 + 1 = n_{\updownarrow}(X).$$

When we substitute everything into (7.1), we obtain:

$$|E| \geq 3 + 3 + 3 \cdot n_{\updownarrow}(X) + |X|_{\mathcal{A}} = 6 + 3 \cdot m + |X|_{\mathcal{A}}.$$

Now without loss of generality, assume that $y = _{\triangle}$ and consider the DNA expression $E' = \langle\uparrow \langle\updownarrow \alpha_1\rangle \langle\updownarrow \alpha_2\rangle \ldots \langle\updownarrow \alpha_m\rangle\rangle$. It is easily verified that

$$\mathcal{S}(E') = \binom{\alpha_1}{c(\alpha_1)}_{\triangle}\binom{\alpha_2}{c(\alpha_2)}_{\triangle} \cdots {}_{\triangle}\binom{\alpha_m}{c(\alpha_m)} = X$$

and that

$$|E'| = 3 + 3 \cdot m + |X|_{\mathcal{A}} < |E|.$$

Thus, also in this case, $E = \langle\updownarrow E_1\rangle$ is not minimal.

□

The fact that the only minimal $\updownarrow$-expressions are $\langle\updownarrow \alpha_1\rangle$ for $\mathcal{N}$-words α_1, implies the following:

Corollary 7.6 *Let X be an expressible formal DNA molecule which is not double-complete. Then each minimal DNA expression denoting X is either an $\uparrow$-expression or a $\downarrow$-expression.*

Theorem 7.5 describes the (unique) minimal DNA expression denoting a double-complete formal DNA molecule. This result is quite straightforward. For other nick free formal DNA molecules, the construction of minimal DNA expressions and the corresponding proof of correctness is more involved.

In Section 6.2, we defined the primitive $\uparrow$-blocks and primitive $\downarrow$-blocks of a formal DNA molecule. Here, these notions turn out to be useful, again. For a nick free formal DNA molecule, however, each $\uparrow$-component is an upper component and each $\downarrow$-component is a lower component. To reflect this in our terminology, we will use the term primitive *upper blocks* rather than primitive $\uparrow$-blocks, and the term primitive *lower blocks* rather than primitive $\downarrow$-blocks. We will use the new, but equivalent terminology only in the context of nick free formal DNA molecules.

We will find that, in general, there may be different minimal DNA expressions denoting the same nick free formal DNA molecule. Sometimes, there exist minimal DNA expressions for a formal DNA molecule with different outermost operators, $\uparrow$ or $\downarrow$, as is the case for the molecule we have seen in Example 7.2. Moreover, given an outermost operator, there may be different ways to partition a formal DNA molecule into parts that are (intuitively speaking) generated by the outermost operator and parts that are generated by other operators in the DNA expression, at a higher nesting level. Each of these ways may lead to one or (many) more minimal DNA expressions.

We first consider a very special partitioning of a nick free formal DNA molecule, which is induced by the primitive lower blocks occurring in it. It is based on an observation that follows immediately from Lemma 6.7(1):

Corollary 7.7 *Let X be a nick free formal DNA molecule and let $X_1, \ldots, X_{r_0}$ for some $r_0 \geq 0$ be the primitive lower blocks of X in the order of their occurrence in X. Then there exists a unique sequence of substrings $Y_0, Y_1, \ldots, Y_{r_0}$ of X, such that $X = Y_0X_1Y_1 \ldots X_{r_0}Y_{r_0}$.*

We will analyse the partitioning $Y_0, X_1, Y_1, \ldots, X_{r_0}, Y_{r_0}$ mentioned in this observation, and the substrings $Y_0, Y_1, \ldots, Y_{r_0}$ occurring in it. To allow for unambiguous and direct references to this partitioning and these substrings, we first formally define them. Note that in Section 6.2, we have already defined and analysed the primitive lower blocks $X_1, \ldots, X_{r_0}$ occurring in the partitioning. We just called them primitive $\downarrow$-blocks there.

Definition 7.8 *Let X be a nick free formal DNA molecule, let $X_1, \ldots, X_{r_0}$ for some $r_0 \geq 0$ be the primitive lower blocks of X in the order of their occurrence in X, and let $Y_0, \ldots, Y_{r_0}$ be the substrings of X such that $X = Y_0X_1Y_1 \ldots X_{r_0}Y_{r_0}$.*

- *The primitive lower block partitioning of X is the sequence $Y_0, X_1, Y_1, \ldots, X_{r_0}, Y_{r_0}$.*
- *A maximal upper sequence of X is the occurrence $(Y_0X_1Y_1 \ldots X_j, X_{j+1}Y_{j+1} \ldots X_{r_0}Y_{r_0})$ of a substring Y_j with $0 \leq j \leq r_0$ and $Y_j \neq \lambda$.*
- *The maximal upper prefix of X is the occurrence $(\lambda, X_1Y_1 \ldots X_{r_0}Y_{r_0})$ of Y_0.*
- *The maximal upper suffix of X is the occurrence $(Y_0X_1Y_1 \ldots X_{r_0}, \lambda)$ of Y_{r_0}.*
- *An internal maximal upper sequence of X is the occurrence $(Y_0X_1Y_1 \ldots X_j, X_{j+1}Y_{j+1} \ldots X_{r_0}Y_{r_0})$ of a substring Y_j with $1 \leq j \leq r_0 - 1$.*

Hence, if $r_0 \geq 1$, then the maximal upper prefix of X is the substring of X preceding the first primitive lower block and the maximal upper suffix of X is the substring of X succeeding the last primitive lower block. An internal maximal upper sequence of X is the substring of X separating two consecutive primitive lower blocks.

For notational convenience, we will in general omit the commas and write $Y_0X_1Y_1 \dots X_{r_0}Y_{r_0}$ instead of $Y_0, X_1, Y_1, \dots, X_{r_0}, Y_{r_0}$ to describe the primitive lower block partitioning.

As usual, although formally maximal upper sequences, the maximal upper prefix, the maximal upper suffix and internal maximal upper sequences of a nick free formal DNA molecule X are defined as *occurrences* of substrings of X, we will often refer to them by the substrings themselves (and in fact, we already did this right after the definition). Implicitly, however, we keep associating to them a position in X. For example, if both the maximal upper prefix and the maximal upper suffix of X are equal to λ, then they are *not* equal, because the occurrence of the maximal upper prefix is (λ, X) and the occurrence of the maximal upper suffix is (X, λ).

Also, if for example the maximal upper prefix Y_0 of X is empty, then we keep including it in the notation for the primitive lower block partitioning. We will not write $X_1Y_1 \dots X_{r_0}Y_{r_0}$, because formally, the primitive lower block partitioning is defined as $Y_0, X_1, Y_1, \dots, X_{r_0}, Y_{r_0}$, with Y_0 and a comma preceding the first primitive lower block X_1. Moreover, by the inclusion of Y_0, it is always clear which substrings from the primitive lower block partitioning $Y_0X_1Y_1 \dots X_{r_0}Y_{r_0}$ denote the primitive lower blocks (the second one, the fourth one, and so on), the maximal upper prefix (the first one), the internal maximal upper sequences (the third one, the fifth one, and so on) and the maximal upper suffix (the last one). Of course, we have the same convention for the maximal upper suffix.

Intuitively, a maximal upper sequence Y of X is 'maximal' in the sense that it cannot be extended either to the left or to the right by a 'block' or by a 'block' . We will make this intuition more formal in Lemma 7.12(1).

The *primitive upper block partitioning* of a nick free formal DNA molecule is defined analogously to the primitive lower block partitioning. Also, a *maximal lower sequence*, the *maximal lower prefix*, the *maximal lower suffix* and an *internal maximal lower sequence* are defined analogously to the upper counterparts.

Most results in the remainder of this section will be stated only for primitive lower blocks, (internal) maximal upper sequences and the maximal upper prefix and suffix. Of course, there exist analogous results for primitive upper blocks, (internal) maximal lower sequences and the maximal lower prefix and suffix. Whenever we need one of these analogous results, we will simply refer to the other version, i.e., the one that we have stated.

As an example, in Figure 7.1, we have indicated the primitive lower block partitioning and the primitive upper block partitioning of a certain nick free formal DNA molecule. This figure also illustrates the difference between a maximal upper (lower) sequence and a primitive upper (lower, respectively) block. A maximal upper sequence seems to be a 'short version' of a primitive upper block, We will come back to the relation between the primitive upper blocks and the maximal upper sequences of a nick free formal DNA molecule in Lemma 7.13.

By Lemma 6.5(1), if X is double-complete, then it does not have any primitive block. Hence, by definition, the primitive lower block partitioning is equal to $Y_0 = X$. As, obviously, $X \neq \lambda$, X itself is its only maximal upper sequence. Note that in this case, the maximal upper sequence does not contain any upper component. This cannot occur with primitive upper blocks: by definition, each primitive upper block contains at least

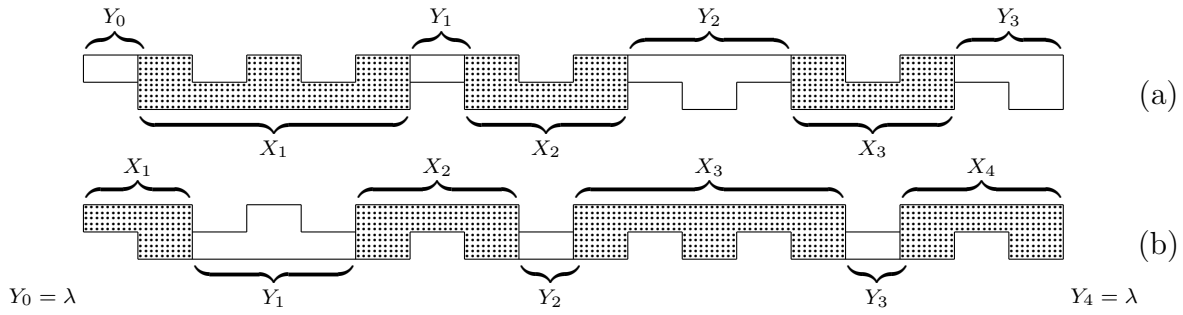


Figure 7.1: Two partitionings of a nick free formal DNA molecule X. (a) The primitive lower block partitioning of X. X_1, X_2, X_3 are the primitive lower blocks of X, Y_0, Y_1, Y_2, Y_3 are the maximal upper sequences, Y_0 is the maximal upper prefix and Y_3 is the maximal upper suffix of X. (b) The primitive upper block partitioning of X. Here, X_1, X_2, X_3, X_4 are the primitive upper blocks and Y_1, Y_2, Y_3 are the maximal lower sequences of X. Both the maximal lower prefix Y_0 and the maximal lower suffix Y_4 are empty.

one upper component. In Lemma 7.12(4), we will see that this is the only case in which a maximal upper sequence does not contain any upper component. In Lemma 7.11, we will examine the primitive lower block partitioning for some more types of formal DNA molecules.

For a nick free formal DNA molecule X, we use $n_{\text{mus}}(X)$ to denote the number of maximal upper sequences of X, and we use $n_{\text{imus}}(X)$ to denote the number of internal maximal upper sequences of X. We will see that there exist close relations between $n_{\text{mus}}(X)$ and $B_{\uparrow}(X)$ on the one hand, and between $n_{\text{imus}}(X)$ and $B_{\downarrow}(X)$ on the other hand. However, some constructions we will encounter are based on (internal) maximal upper sequences, rather than on primitive blocks. To perform calculations on the results of these constructions (in Lemma 7.22 and the proof of Lemma 7.23(1) it is useful to also have the new notations $n_{\text{mus}}(X)$ and $n_{\text{imus}}(X)$.

We first give the relation between $n_{\text{imus}}(X)$ and $B_{\downarrow}(X)$. In Lemma 7.13(3), we will consider the relation between $n_{\text{mus}}(X)$ and $B_{\uparrow}(X)$.

Lemma 7.9 *Let X be a nick free formal DNA molecule.*

1. *If $B_{\downarrow}(X) = 0$, then $n_{imus}(X) = 0$.*
2. *If $B_{\downarrow}(X) \geq 1$, then $n_{imus}(X) = B_{\downarrow}(X) - 1$.*

Proof: By definition, an internal maximal upper sequence of X is a substring of X separating two consecutive primitive lower blocks. If X does not have any primitive lower block, then it certainly does not have any internal maximal upper sequence: $n_{\text{imus}}(X) = 0$. If X has $B_{\downarrow}(X) \geq 1$ primitive lower blocks, then the number of internal maximal upper sequences is 1 less. □

Some properties of (internal) maximal upper sequences follow easily from the definition. Nevertheless, it is useful to state them explicitly:

Lemma 7.10 *Let X be a nick free formal DNA molecule.*

1. *Each internal maximal upper sequence is not empty, and thus is a maximal upper sequence.*

2. *The maximal upper sequences of X are pairwise disjoint.*

3. *X is an alternating sequence of (all) its primitive lower blocks and (all) its maximal upper sequences.*

Claim 1 implies that the term *internal maximal upper sequence* is proper.

In Lemma 6.9(1), we used the formulation "can be considered as" to express the relation between a formal DNA molecule X that is not double-complete and the alternating sequence of primitive $\uparrow$-blocks and primitive $\downarrow$-blocks of X. Any two consecutive primitive blocks in this sequence share a double component.

Here, we can safely say that a nick free formal DNA molecule *is* an alternating sequence of its primitive lower blocks and its maximal upper sequences.

Proof: Let $Y_0X_1Y_1 \dots X_{r_0}Y_{r_0}$ for some $r_0 \geq 0$ be the primitive lower block partitioning of X.

1. Let Y_j with $1 \leq j \leq r_0 - 1$ be an internal maximal upper sequence. By definition, Y_j is preceded in X by the primitive lower block X_j and succeeded in X by the primitive lower block X_{j+1}. By Lemma 6.7(3), Y_j contains at least one component of X. Then by definition, Y_j is a maximal upper sequence.

2. Each maximal upper sequence is one of the substrings Y_j from the primitive lower block partitioning of X. Clearly, the substrings Y_j are pairwise disjoint; two different substrings Y_{j_1} and Y_{j_2} are separated by at least a primitive lower block X_j. Then certainly the maximal upper sequences of X are pairwise disjoint.

3. By definition, $X = Y_0X_1Y_1 \dots X_{r_0}Y_{r_0}$ is an alternating sequence of (all) its primitive lower blocks $X_1, \dots, X_{r_0}$ and the substrings $Y_0, Y_1, \dots, Y_{r_0}$. Because the maximal upper sequences of X are defined as the non-empty substrings Y_j, and because, by Claim 1, at least $Y_1, \dots, Y_{r_0-1}$ are non-empty, X is also an alternating sequence of (all) its primitive lower blocks and (all) its maximal upper sequences.

□

Different statements about the maximal upper sequences, maximal upper prefix or maximal upper suffix of a certain nick free formal DNA molecule may be equivalent. If one statement is valid, then so is the other.

Lemma 7.11 *Let X be a nick free formal DNA molecule and let $Y_0X_1Y_1 \dots X_{r_0}Y_{r_0}$ for some $r_0 \geq 0$ be the primitive lower block partitioning of X.*

1. *The following two statements are equivalent:*

   (a) *X does not contain any maximal upper sequence.*

   (b) *X does not contain any upper component and contains at least one lower component.*

2. *The following seven statements are equivalent:*

   (a) *$r_0 = 0$.*

   (b) *X does not contain any lower component.*

   (c) *$Y_0 = X$.*

(d) *$Y_{r_0} = X$.*

(e) *The maximal upper prefix of X is equal to the maximal upper suffix of X.*

(f) *X is a maximal upper sequence of itself.*

(g) *X is the only maximal upper sequence of itself.*

3. *If X is not double-complete, then the following four statements are equivalent:*

   (a) *The maximal upper prefix of X is empty.*

   (b) *The maximal lower prefix of X is not empty.*

   (c) *The alternating sequence from Lemma 6.9(1) starts with a primitive lower block.*

   (d) *The first single-stranded component of X is a lower component.*

4. *If X is not double-complete, then the following four statements are equivalent:*

   (a) *The maximal upper suffix of X is empty.*

   (b) *The maximal lower suffix of X is not empty.*

   (c) *The alternating sequence from Lemma 6.9(1) ends with a primitive lower block.*

   (d) *The last single-stranded component of X is a lower component.*

Note that by definition, e.g., saying that the maximal upper prefix of X is empty is equivalent to saying that it is not a maximal upper sequence. For our example formal DNA molecule from Figure 7.1, none of the statements in this result is true. For example, the maximal upper prefix is not empty, whereas the maximal lower prefix *is* empty.

Proof:

1. By Lemma 7.10(3), X is an alternating sequence of (all) its primitive lower blocks and (all) its maximal upper sequences. Hence, X does not contain any maximal upper sequence, if and only if $X = X_1$ is a primitive lower block of itself. By the definition of a primitive lower block, this is the case, if and only if X does not contain any upper component and contains at least one lower component.

2. We first observe that Statements 2f and 2g are equivalent, because maximal upper sequences are not empty and by Lemma 7.10(2), different maximal upper sequences of X are disjoint.

   Next, we prove that each of Statements 2b–2f is equivalent to the 'central' Statement 2a.

   The equivalence of Statements 2b and 2a follows from Lemma 6.13(1b) and the fact that r_0 is the number of primitive lower blocks of X, i.e., that $r_0 = B_{\downarrow}(X)$.

   We observe that by definition, $X = Y_0 X_1 Y_1 \dots X_{r_0} Y_{r_0}$.

   Assume that Statement 2a is true, i.e., that $r_0 = 0$. Then by definition, $X = Y_0 = Y_{r_0}$ and both the maximal upper prefix and the maximal upper suffix of X are equal to the occurrence (λ, λ) of $Y_0 = Y_{r_0}$. Hence, Statements 2c, 2d and 2e are true. Because the formal DNA molecule $X = Y_0$ is not empty, it is, by definition, a maximal upper sequence. Hence, also Statement 2f is true.

2c $\Longrightarrow$ 2a and 2d $\Longrightarrow$ 2a: If $Y_0 = X$ or $Y_{r_0} = X$, while by definition, $X = Y_0X_1Y_1 \ldots X_{r_0}Y_{r_0}$, then r_0 must be 0, because the primitive lower blocks $X_1, \ldots, X_{r_0}$ are not empty.

2e $\Longrightarrow$ 2a: If the maximal upper prefix and the maximal upper suffix of X are equal, then the occurrence $(\lambda, X_1Y_1 \ldots X_{r_0}Y_{r_0})$ of Y_0 is equal to the occurrence $(Y_0X_1Y_1 \ldots X_{r_0}, \lambda)$ of Y_{r_0}. Again, because primitive lower blocks are not empty, this is only possible, if $r_0 = 0$.

2f $\Longrightarrow$ 2a: If X is a maximal upper sequence of itself, then by Lemma 7.10(3), it cannot have any primitive lower block. Hence, $r_0 = 0$.

3. Assume that X is not double-complete.

   If the maximal upper prefix Y_0 of X is empty, then certainly $r_0 \geq 1$, because otherwise, X would be empty. In this case, the primitive lower block X_1 is a prefix of X. Hence, the alternating sequence from Lemma 6.9(1) starts with a primitive lower block.

   If, on the other hand, the alternating sequence from Lemma 6.9(1) starts with a primitive lower block X_1, then X_1 is a prefix of X. Hence, by definition, the maximal upper prefix Y_0 of X is empty.

   We have thus proved that Statements 3a and 3c are equivalent. In an analogous way, we can prove that the maximal lower prefix of X is empty, if and only if the alternating sequence from Lemma 6.9(1) starts with a primitive upper block. Negating both sides of this equivalence, we find that the maximal lower prefix of X is not empty, if and only if the alternating sequence from Lemma 6.9(1) starts with a primitive lower block. Hence, also Statement 3b is equivalent to Statement 3c.

   Finally, the equivalence of Statements 3c and 3d follows immediately from Lemma 6.9(2a).

4. The proof of this claim is analogous to that of the previous claim.

□

The next result deals with the components occurring in and surrounding a maximal upper sequence.

Lemma 7.12 *Let X be a nick free formal DNA molecule and let $x'_1 \ldots x'_k$ be the decomposition of X.*

1. *Let $Y = x'_{b_0} \ldots x'_{b_1}$ with $1 \leq b_0 \leq b_1 \leq k$ be a maximal upper sequence of X.*
   (a) *If $b_0 \geq 2$, then $b_0 \geq 3$, x'_{b_0-2} is a lower component of X, x'_{b_0-1} is a double component of X and x'_{b_0} is an upper component of X.*
   (b) *If $b_1 \leq k-1$, then $b_1 \leq k-2$, x'_{b_1+2} is a lower component of X, x'_{b_1+1} is a double component of X and x'_{b_1} is an upper component of X.*
2. *Each maximal upper sequence of X is an alternating sequence of upper components and double components of X.*
3. *Each upper component of X occurs in a (exactly one) maximal upper sequence of X.*

4. (a) If X is double-complete, then the only maximal upper sequence of X is X itself.

(b) If X is not double-complete, then each maximal upper sequence of X contains at least one upper component.

Claims 1 and 3 can be considered as the maximal upper sequence-versions of Lemma 6.6 and Lemma 6.7(2), respectively. For Claims 2 and 4, there does not exist a corresponding result for primitive ↑-blocks. Already by definition, a primitive ↑-block does not contain any ↓-components and contains at least one ↑-component.

By definition, an internal maximal upper sequence of X is both preceded and succeeded in X by a primitive lower block. Hence, by Claim 1, each internal maximal upper sequence both starts with an upper component and ends with an upper component.

Moreover, by the same claim, there do not exist maximal upper sequences that start with the second component or end with the last but one component of X.

Proof: Let $Y_0X_1Y_1 \dots X_{r_0}Y_{r_0}$ be the primitive lower block partitioning of X. $X_1, \dots, X_{r_0}$ are the primitive lower blocks of X. A maximal upper sequence is a non-empty substring Y_j.

1. By definition, there exists j with $0 \leq j \leq r_0$, such that Y is the substring Y_j.

   (a) If $b_0 \geq 2$, then $j > 0$ and Y is preceded in X by the primitive lower block X_j. X_j ends with the component x'_{b_0-1} and $b_0 - 1 \leq k - 1$. Now, the claim follows from (the analogue for primitive ↓-blocks of) Lemma 6.6(2).

   (b) The proof of this subclaim is analogous to that of the previous subclaim.

2. By definition, a primitive lower block of X is a sequence of components of X. Then also the substrings Y_j preceding, separating and succeeding the primitive lower blocks are sequences of components of X. This is, of course, in particular true for every maximal upper sequence of X.

   By Lemma 6.7(2), each lower component of X occurs in a primitive lower block of X. Hence, a maximal upper sequence Y_j does not contain any lower component. Now, the claim follows from Corollary 3.8.

3. By definition, a primitive lower block of X does not contain any upper component. Hence, each upper component of X must be part of a (non-empty) substring Y_j, i.e., of a maximal upper sequence. Obviously, there is only one maximal upper sequence Y_j that applies.

4. (a) Shortly after the definition of the primitive lower block partitioning, we already considered this partitioning for double-complete formal DNA molecules. We also mentioned that such a molecule is itself its only maximal upper sequence.

   (b) Assume that X is not double-complete, and let $Y = x'_{b_0} \dots x'_{b_1}$ with $1 \leq b_0 \leq b_1 \leq k$ be an arbitrary maximal upper sequence of X.

   If $b_0 \geq 2$, then by Claim 1a, the first component of Y is an upper component. Analogously, if $b_1 \leq k-1$, then the last component of Y is an upper component.

   Finally, if $b_0 = 1$ and $b_1 = k$, then $Y = X$, and by Claim 2, X is an alternating sequence of upper components and double components. Because X is not double-complete, $Y = X$ contains at least one upper component.

□

The example from Figure 7.1 suggested that a maximal upper sequence of a nick free formal DNA molecule is a 'short version' of a primitive upper block. We now formalize this suggestion and prove that it holds in general for molecules that are not double-complete.

Lemma 7.13 *Let X be a nick free formal DNA molecule and let $x'_1 \dots x'_k$ be the decomposition of X.*

1. *Each primitive upper block of X contains exactly one maximal upper sequence of X.*

   In particular, let $X'_1 = x'_{a_0} \dots x'_{a_1}$ with $1 \leq a_0 \leq a_1 \leq k$ be an arbitrary primitive upper block of X, and let

   $$b_0 = \begin{cases} 1 & \text{if } a_0 = 1 \\ a_0 + 1 & \text{if } a_0 \geq 2 \end{cases} \qquad \text{and} \qquad b_1 = \begin{cases} k & \text{if } a_1 = k \\ a_1 - 1 & \text{if } a_1 \leq k-1. \end{cases}$$

   Then $x'_{b_0} \dots x'_{b_1}$ is a maximal upper sequence of X.

2. *Assume that X is not double-complete.*

   Each maximal upper sequence of X is contained in exactly one primitive upper block of X.

   In particular, let $Y = x'_{b_0} \dots x'_{b_1}$ with $1 \leq b_0 \leq b_1 \leq k$ be an arbitrary maximal upper sequence of X, and let

   $$a_0 = \begin{cases} 1 & \text{if } b_0 = 1 \\ b_0 - 1 & \text{if } b_0 \geq 3 \end{cases} \qquad \text{and} \qquad a_1 = \begin{cases} k & \text{if } b_1 = k \\ b_1 + 1 & \text{if } b_1 \leq k-2. \end{cases}$$

   Then $x'_{a_0} \dots x'_{a_1}$ is a primitive upper block of X.

3. *If X is not double-complete, then there is a bijection (induced by inclusion) between primitive upper blocks of X on the one hand and maximal upper sequences of X on the other hand. In particular, $B_\uparrow(X) = n_{mus}(X)$.*

Note that by Lemma 7.12(1), a maximal upper sequence of X cannot start with the second component or end with the last but one component of X. Hence, Claim 2 covers all possible maximal upper sequences in a formal DNA molecule that is not double-complete.

Proof: Recall that if X is not double-complete, then by Lemma 6.9(1), X can be considered as an alternating sequence of all its primitive upper blocks all its primitive lower blocks.

1. The existence of the primitive upper block X'_1 implies that X is not double-complete. We distinguish four cases:

   - If $a_0 = 1$ and $a_1 = k$, then $X'_1 = X$. By the definition of a primitive upper block, X does not contain any lower component. Hence, by Lemma 7.11(2), $Y_0 = X = x'_1 \dots x'_k$ is a maximal upper sequence of itself.

- If $a_0 = 1$ and $a_1 \leq k-1$, then by Lemma 6.6(2), $1 = a_0 < a_1$ and x'_{a_1} is a double component. Further, the alternating sequence of primitive upper blocks and primitive lower blocks starts with X'_1 and proceeds with a primitive lower block X_1. X_1 is the first primitive lower block of X and starts with the double component x'_{a_1}. By definition, the non-empty prefix $Y_0 = x'_1 \dots x'_{a_1-1}$ is a maximal upper sequence of X.
- The case that $a_0 \geq 2$ and $a_1 = k$ is symmetric to the previous case. In an analogous way, we find that the non-empty suffix $Y_1 = x'_{a_0+1} \dots x'_k$ is a maximal upper sequence of X.
- If $a_0 \geq 2$ and $a_1 \leq k-1$, then by Lemma 6.6, $a_0 < a_1$, both x'_{a_0} and x'_{a_1} are double components, and both x'_{a_0+1} and x'_{a_1-1} are upper components. In the alternating sequence, X'_1 is preceded by a primitive lower block X_0 and succeeded by a primitive lower block X_1. X_0 ends with the double component x'_{a_0} and X_1 starts with the double component x'_{a_1}. By definition, the sequence of components $Y_1 = x'_{a_0+1} \dots x'_{a_1-1}$ separating X_0 and X_1 is an internal maximal upper sequence. In particular, by Lemma 7.10(1), it is a maximal upper sequence.

In each of the cases, the maximal upper sequence we obtain is equal to $x'_{b_0} \dots x'_{b_1}$ for indices b_0 and b_1 as defined in the claim.

Let us, for simplicity, use Y to denote this maximal upper sequence, in all four cases. Y is a subsequence of the components in X'_1. By Lemma 7.10(2), the maximal upper sequences of X are pairwise disjoint, Hence, if X'_1 contains another maximal upper sequence, then that one must be contained in the components of X'_1 that are not in Y.

The only components of X'_1 that may not be in Y are the first component x'_{a_0} and the last component x'_{a_1}. These components then are double components that are part of primitive lower blocks. By definition, components of a primitive lower block are not in a maximal upper sequence.

We conclude that X'_1 does not contain any maximal upper sequence other than Y.

2. We again distinguish four cases:

- If $b_0 = 1$ and $b_1 = k$, then $Y = X$. By Lemma 7.12(2) and (4b), $Y = X$ does not contain any lower component, but does contain at least one upper component. Hence, by Lemma 6.5(2), $X = x'_1 \dots x'_k$ is a primitive upper block of itself.
- If $b_0 = 1$ and $b_1 \leq k-2$, then by definition, Y is succeeded in X by a primitive lower block X_1. X_1 is the first primitive lower block of X. By Lemma 6.7(4), its first component x'_{b_1+1} is a double component and the prefix $x'_1 \dots x'_{b_1+1}$ of X is a primitive upper block.
- The case that $b_0 \geq 3$ and $b_1 = k$ is symmetric to the previous case. In an analogous way, we find that the suffix $x'_{b_0-1} \dots x'_k$ is a primitive upper block of X.
- If $b_0 \geq 3$ and $b_1 \leq k-2$, then by definition, Y is preceded in X by a primitive lower block X_1 and succeeded in X by a primitive lower block X_2. X_1 and X_2 are consecutive primitive lower blocks of X. Now by Lemma 6.7(3), both

the last component x'_{b_0-1} of X_1 and the first component x'_{b_1+1} of X_2 are double components, and $x'_{b_0-1} \ldots, x'_{b_1+1}$ is a primitive upper block.

Note that in the last three cases, where the primitive upper block we obtain is larger than the maximal upper sequence X_1, the additional components are double components.

In each of the four cases, the primitive upper block we obtain is equal to $x'_{a_0} \ldots x'_{a_1}$ for indices a_0 and a_1 as defined in the claim. The maximal upper sequence Y cannot also be contained in another primitive upper block, because, by Lemma 6.7(1), primitive upper blocks are pairwise disjoint.

3. Assume that X is not double-complete. Consider the relation between primitive upper blocks and maximal upper sequences of X induced by inclusion. By Claim 1, this relation is a function from the set of primitive upper blocks into the set of maximal upper sequences. By Claim 2, this function is both injective and surjective.

□

By Lemma 7.10(3), a nick free formal DNA molecule is an alternating sequence of all its primitive lower blocks and all its maximal upper sequences. When we take a subsequence X^s of these primitive lower blocks and maximal upper sequences, we can, in turn, consider the primitive lower blocks and maximal upper sequences of X^s:

Lemma 7.14 *Let X be a nick free formal DNA molecule, and let X^s be a non-empty subsequence of consecutive primitive lower blocks and maximal upper sequences of X.*

1. *The primitive lower blocks of X^s are precisely the primitive lower blocks of X occurring in X^s.*

2. *The maximal upper sequences of X^s are precisely the maximal upper sequences of X occurring in X^s.*

We cannot extend this result to arbitrary formal DNA submolecules of X. In general, it is not true that the primitive lower blocks and maximal upper sequences of a submolecule X^s of X are precisely the primitive lower blocks and maximal upper sequences of X restricted to X^s.

Example 7.15 Let $X = \binom{\alpha_1}{-}\binom{\alpha_2}{c(\alpha_2)}\binom{-}{\alpha_3}\binom{\alpha_4}{c(\alpha_4)}$ for $\mathcal{N}$-words $\alpha_1, \alpha_2, \alpha_3, \alpha_4$, and let $X^s = \binom{\alpha_1}{-}\binom{\alpha_2}{c(\alpha_2)}$. The only primitive lower block of X is $X_1 = \binom{\alpha_2}{c(\alpha_2)}\binom{-}{\alpha_3}\binom{\alpha_4}{c(\alpha_4)}$ and the only maximal upper sequence of X is $Y_0 = \binom{\alpha_1}{-}$. When we restrict X_1 to X^s, we obtain $\binom{\alpha_2}{c(\alpha_2)}$, but this is not a primitive lower block of X^s. In fact, X^s does not have any primitive lower block. Hence, X^s is a maximal upper sequence of itself. Obviously, this maximal upper sequence is not equal to Y_0 restricted to X^s. ■

Proof of Lemma 7.14: Because X is nick free and X^s is a non-empty substring of X, by Lemma 3.4, X^s is a nick free formal DNA molecule. This implies that indeed primitive lower blocks and maximal upper sequences are defined for X^s.

First, we consider a special case. If X is double-complete, then by Lemma 6.5(1), it does not have any primitive lower block. Indeed, by Lemma 7.12(4a), X is the only maximal upper sequence of itself. Hence, X^s, which is a non-empty subsequence of the

primitive lower blocks and maximal upper sequences of X, must be equal to X. Then the claims are trivially valid.

Conversely, if X^s is double-complete, then it cannot contain any primitive lower block of X, because by definition, a primitive lower block contains at least one lower component. Hence, X^s must be equal to one maximal upper sequence of X. Apparently, X contains a maximal upper sequence that is double-complete. By Lemma 7.12(4), X must be double-complete itself, and X must be equal to this double-complete maximal upper sequence. Again, we conclude that $X^s = X$ and that the claims are trivially valid.

Now, let us assume that X is not double-complete and (thus) that X^s is not double-complete. Let $x'_1 \dots x'_k$ for some $k \geq 1$ be the decomposition of X. Both primitive lower blocks (by definition) and maximal upper sequences (by Lemma 7.12(2)) of X are built up of components of X. Hence, so is X^s: there exist i_0 and j_0 with $1 \leq i_0 \leq j_0 \leq k$ such that $X^s = x'_{i_0} \dots x'_{j_0}$. By Lemma 6.3, each component of X^s is also a component of X.

1. We first prove that each primitive lower block of X^s is one of the primitive lower blocks of X occurring in X^s. We subsequently establish the reverse statement.

   • Let $X^s_1 = x'_{a_0} \dots x'_{a_1}$ with $i_0 \leq a_0 \leq a_1 \leq j_0$ be a primitive lower block of X^s. By definition, X^s_1 contains at least one lower component and does not contain any upper component.

   If $a_0 = i_0$, then the first single-stranded component of X^s is a lower component. By Lemma 7.12(2) and (4b), a maximal upper sequence of X does not contain any lower component. but does contain at least one upper component. Hence, X^s (seen as an alternating subsequence of primitive lower blocks and maximal upper sequences of X) starts with a primitive lower block. In particular, $x'_{a_0} = x'_{i_0}$ is the first component of a primitive lower block X_1 of X.

   If $a_0 \geq i_0 + 1$, then by Lemma 6.6(1) $a_0 < a_1 \leq j_0$, x'_{a_0-1} is an upper component, x'_{a_0} is a double component and x'_{a_0+1} is a lower component of X^s (and thus of X). By Lemma 6.7(2), x'_{a_0+1} occurs in a primitive lower block X_1 of X. By definition, (the upper component) x'_{a_0-1} is not part of X_1. Hence, the first component of X_1 must be either (the double component) x'_{a_0} or (the lower component) x'_{a_0+1}. By Lemma 6.6(1), it must be x'_{a_0}. Also in this case, x'_{a_0} is the first component of a primitive lower block X_1 of X.

   Analogously, we can prove that x'_{a_1} is the last component of a primitive lower block X_2 of X.

   Now we observe that by definition, different primitive lower blocks of X are separated by at least one (internal) maximal upper sequence, that each maximal upper sequence of X contains at least one upper component, and that $X^s_1 = x'_{a_0} \dots x'_{a_1}$ does not contain any upper component. Consequently, X_1 and X_2 must be the same primitive lower block of X: $X^s_1 = X_1 = X_2$ is a primitive lower block of X.

   • In a similar way, we can prove that each primitive lower block X_1 of X which occurs in X^s is a primitive lower block of X^s.

   First, we establish that the first component of X_1 is also the first component of a primitive lower block X^s_1 of X^s. For this we examine both the case that X^s (seen as an alternating subsequence of primitive lower blocks and maximal upper sequences of X) starts with X_1 and the case that X^s does not start with X_1. Analogously, we find that the last component of X_1 is also the last component of a primitive lower

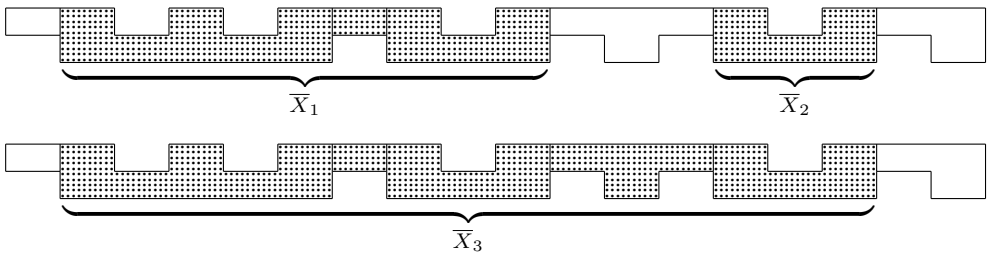


Figure 7.2: Three different lower blocks $\overline{X}_1, \overline{X}_2, \overline{X}_3$ of the formal DNA molecule from Figure 7.1. The vertical lines inside $\overline{X}_1$ and $\overline{X}_3$ indicate the boundaries between the primitive lower blocks and the maximal upper sequences constituting these lower blocks.

block X_2^s of X^s. Finally, we prove that X_1^s and X_2^s are the same primitive lower block, because otherwise, they would be separated in X^s by at least one (internal) maximal upper sequence, and X_1 would contain an upper component.

2. The proof of this claim is straightforward, because we can apply the previous claim.

   By definition, a maximal upper sequence of X^s is a non-empty substring of X^s that either precedes the first primitive lower block of X^s, or separates two consecutive primitive lower blocks of X^s, or succeeds the last primitive lower block of X^s. Hence, it is a non-empty substring of X that either precedes the first primitive lower block of X occurring in X^s, or separates two consecutive primitive lower blocks of X occurring in X^s, or succeeds the last primitive lower block of X occurring in X^s. This implies that it is a maximal upper sequence of X occurring in X^s.

   When we simply reverse the above argumentation, we find that a maximal upper sequence of X occurring in X^s is also a maximal upper sequence of X^s.

□

We now define a particular type of subsequence of primitive lower blocks and maximal upper sequences. This subsequence both starts and ends with a primitive lower block.

Definition 7.16 *Let X be a nick free formal DNA molecule and let $Y_0X_1Y_1 \ldots X_{r_0}Y_{r_0}$ for some $r_0 \geq 0$ be the primitive lower block partitioning of X.*

A lower block is an occurrence $(Y_0X_1Y_1 \ldots Y_{j_1-1}, Y_{j_2}X_{j_2+1} \ldots X_{r_0}Y_{r_0})$ of a substring $X_{j_1}Y_{j_1} \ldots X_{j_2}$ of X for some j_1 and j_2 with $1 \leq j_1 \leq j_2 \leq r_0$.

Often, we will refer to a lower block simply by the substring involved. The actual occurrence will be clear from the context, e.g., from the indices j_1 and j_2. To distinguish a lower block from a primitive lower block, we will use $\overline{X}_j$ (for a certain index j) to denote a lower block, instead of X_j.

Indeed, as the name suggests, a lower block is a generalization of a primitive lower block. If in the definition $j_1 = j_2$, then we have the primitive lower block X_{j_1}. In general, however, a lower block may contain more than one primitive lower blocks.

The definition of an *upper block* of a nick free formal DNA molecule is analogous to that of a lower block.

In Figure 7.2, we have indicated three different lower blocks $\overline{X}_1$, $\overline{X}_2$, $\overline{X}_3$ of the formal DNA molecule from Figure 7.1. $\overline{X}_1$ contains two primitive lower blocks, $\overline{X}_2$ consists of only one primitive lower block and $\overline{X}_3$ contains all three primitive lower blocks. If a nick

free formal DNA molecule does not have any lower component, then by definition, it does not have a primitive lower block, let alone a lower block.

We can easily determine the values of $B_{\downarrow}(\overline{X}_1)$ and $B_{\uparrow}(\overline{X}_1)$ for a lower block $\overline{X}_1$:

Lemma 7.17 *Let X be a nick free formal DNA molecule and let $\overline{X}_1$ be a lower block of X.*

1. *$\overline{X}_1$ contains at least one single-stranded component, and both the first single-stranded component and the last single-stranded component of $\overline{X}_1$ are lower components.*
2. *Consider $\overline{X}_1$ as an alternating sequence of primitive lower blocks and maximal upper sequences of X (see Definition 7.16).*
   (a) *$B_{\downarrow}(\overline{X}_1)$ is equal to the number of primitive lower blocks of X occurring in $\overline{X}_1$.*
   (b) *$B_{\uparrow}(\overline{X}_1) = B_{\downarrow}(\overline{X}_1) - 1$, which is equal to the number of maximal upper sequences of X occurring in $\overline{X}_1$.*

Proof:

1. By definition, each primitive lower block of X contains at least one lower component and does not contain any upper component of X. Hence, the first single-stranded component and the last single-stranded component of a primitive lower block are well defined, and both of them are lower components. Now, the claim follows immediately from the definition of a lower block.

2. (a) By Lemma 7.14(1), the primitive lower blocks of $\overline{X}_1$ are precisely the primitive lower blocks of X occurring in the alternating sequence mentioned. In particular, $B_{\downarrow}(\overline{X}_1)$ is equal to the number of primitive lower blocks in this alternating sequence.

   (b) By Claim 1 and Lemma 6.13(3d), $B_{\uparrow}(\overline{X}_1) = B_{\downarrow}(\overline{X}_1) - 1$. By Claim 2a, this is 1 less than the number of primitive lower blocks of X occurring in the alternating sequence. Because, by definition, the alternating sequence both starts and ends with a primitive lower block, the number of maximal upper sequences occurring in it is also 1 less than the number of primitive lower blocks.

□

We have used the primitive lower blocks of a nick free formal DNA molecule X to define the primitive lower block partitioning of X. We now use the lower blocks of X to define a *lower block partitioning*:

Definition 7.18 *Let X be a nick free formal DNA molecule.*

A lower block partitioning of X is a sequence $Y_0, \overline{X}_1, Y_1, \ldots, \overline{X}_r, Y_r$ for some $r \geq 0$ such that

- *$X = Y_0\overline{X}_1Y_1 \ldots \overline{X}_rY_r$, and*
- *for $j = 1, \ldots, r$, $\overline{X}_j$ is a lower block of X, and*
- *for each primitive lower block X_1 of X, there is a j with $1 \leq j \leq r$, such that X_1 is contained in $\overline{X}_j$.*

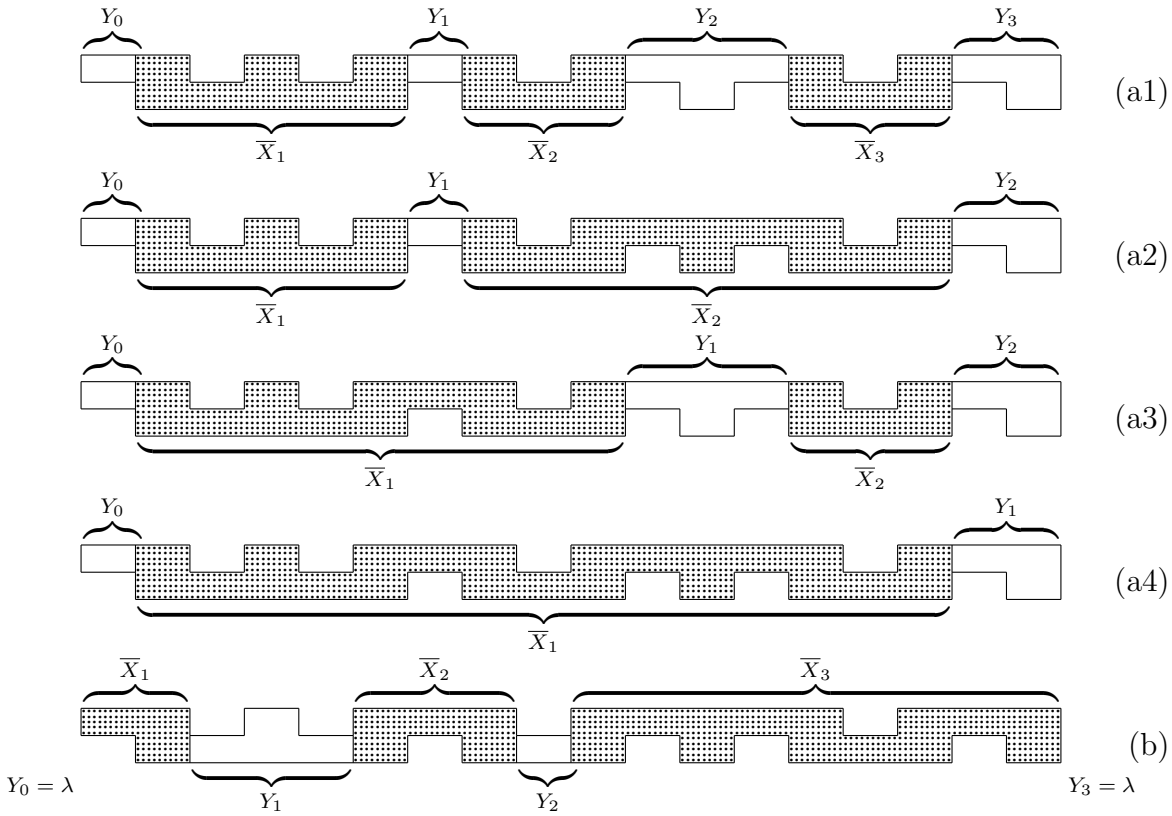


Figure 7.3: Different partitionings of the formal DNA molecule X from Figure 7.1, for which $B_{\uparrow}(X) = 4$ and $B_{\downarrow}(X) = 3$. (a1) (Once more) the primitive lower block partitioning of X. (a2),(a3) Two other lower block partitionings of X. (a4) Yet another lower block partitioning of X: the one defined by one lower block $\overline{X}_1$ containing all primitive lower blocks. (b) An upper block partitioning of X, different from the primitive upper block partitioning.

Hence, a lower block partitioning of X is a partitioning of X based on (disjoint) lower blocks, which together contain all primitive lower blocks. In other words, the set of primitive lower blocks has been partitioned into lower blocks.

Usually, we will write $Y_0\overline{X}_1Y_1 \ldots \overline{X}_rY_r$ instead of $Y_0, \overline{X}_1, Y_1, \ldots, \overline{X}_r, Y_r$ to describe a lower block partitioning. We may also use the symbol $\mathcal{P}$ to refer to a particular lower block partitioning.

Of course, an *upper block partitioning* of a nick free formal DNA molecule is defined analogously. We will see later that lower block partitionings and upper block partitionings are crucial for the construction of minimal $\uparrow$-expressions and $\downarrow$-expressions.

It is easily verified that the primitive lower block partitioning of a nick free formal DNA molecule is in particular a lower block partitioning. In general, however, a nick free formal DNA molecule may have more lower block partitionings than just that one. In Lemma 7.22, we will see how many lower block partitionings there are for a given molecule. In Figure 7.3, we give four lower block partitionings and an upper block partitioning for the formal DNA molecule from Figure 7.1.

For certain formal DNA molecules, there exists only one lower block partitioning, with a very basic structure:

Lemma 7.19 *Let X be a nick free formal DNA molecule. The following four statements are equivalent:*

1. $B_{\downarrow}(X) = 0$.
2. *X does not contain any lower component.*
3. *$Y_0 = X$ is a lower block partitioning of X.*
4. *$Y_0 = X$ is the only lower block partitioning of X.*

Note that we have already established the equivalence of Claims 1 and 2 in Lemma 6.13(1b). We included both statements in the present result to allow for direct transitions between either of these statements and either of the other statements.

Particular instances of molecules for which the four statements are true, are the double-complete formal DNA molecules.

Proof: (1) $\Longrightarrow$ (4) Assume that X does not have any primitive lower block. Because a lower block of a formal DNA molecule contains at least one primitive lower block, X certainly does not contain any lower block. Hence, if $\mathcal{P} = Y_0\overline{X}_1Y_1 \ldots \overline{X}_rY_r$ is a lower block partitioning of X, then r must be 0: $\mathcal{P} = Y_0 = X$. This 'sequence' is indeed a lower block partitioning, because it trivially satisfies the third condition of Definition 7.18.

(4) $\Longrightarrow$ (3) This implication is obviously true.

(3) $\Longrightarrow$ (1) Assume that $Y_0 = X$ is a lower block partitioning of X. Then by the third condition of Definition 7.18, X cannot contain any primitive lower block. □

Note that if X does not have any upper component and has at least one lower component, then by Lemma 6.5(3) and Lemma 6.7(1), the only primitive lower block of X is $X_1 = X$. Hence, by definition, the only lower block partitioning of X is $\mathcal{P} = Y_0X_1Y_1 = Y_0XY_1$ with $Y_0 = Y_1 = \lambda$. Even though Y_0 and Y_1 are empty, we cannot write $\mathcal{P} = X$ in this case. because formally, $\mathcal{P}$ is the sequence Y_0, X_1, Y_1 (with two commas, see also our remark after Definition 7.8).

The definition of a lower block partitioning $Y_0\overline{X}_1Y_1 \ldots \overline{X}_rY_r$ is largely based on conditions on the substrings $\overline{X}_j$ occurring in it. Often, however, we are particularly interested in the substrings Y_j occurring in it. When we focus on them, we get the following, equivalent 'alternative definition':

Lemma 7.20 *Let X be a nick free formal DNA molecule. A sequence $\mathcal{P} = Y_0\overline{X}_1Y_1 \ldots \overline{X}_rY_r$ with $r \geq 0$ is a lower block partitioning of X, if and only if*

- *$X = Y_0\overline{X}_1Y_1 \ldots \overline{X}_rY_r$, and*
- *Y_0 is the maximal upper prefix of X and Y_r is the maximal upper suffix of X, and*
- *for $j = 1, \ldots, r-1$, Y_j is an internal maximal upper sequence of X.*

Proof: We first consider the case that X does not have any lower component. Then by Lemma 7.19, the only lower block partitioning of X is $Y_0 = X$. On the other hand, by Lemma 7.11(2), the maximal upper prefix and the maximal lower suffix of X are equal to X. Because $X \neq \lambda$, the only sequence $\mathcal{P}$ satisfying the first two conditions in the claim is $Y_0 = X$. This 'sequence' also trivially satisfies the third condition.

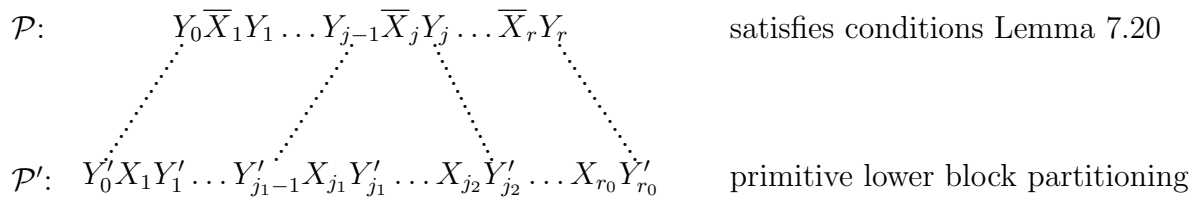


Figure 7.4: Corresponding substrings in a sequence $\mathcal{P}$ satisfying the conditions of Lemma 7.20 and the primitive lower block partitioning $\mathcal{P}'$ (see the proof of Lemma 7.20).

Hence, for this case, the claim holds. We now assume that X has at least one lower component. By Lemma 6.13(1b), X also has at least one primitive lower block.

$\Longrightarrow$ Assume that $\mathcal{P}$ is a lower block partitioning of X.

By definition, each primitive lower block of X is contained in a lower block $\overline{X}_j$ with $1 \leq j \leq r$. This is in particular true for the first primitive lower block X_1. By definition, a lower block is an alternating sequence of primitive lower blocks and maximal upper sequences, starting with a primitive lower block. This implies that the first lower block $\overline{X}_1$ in $\mathcal{P}$ starts with the first primitive lower block X_1. Consequently, Y_0 is the maximal upper prefix of X. Analogously, Y_r is the maximal upper suffix of X.

Consider any substring Y_j with $1 \leq j \leq r-1$ occurring in $\mathcal{P}$. Y_j succeeds the lower block $\overline{X}_j$ in $\mathcal{P}$ and precedes the lower block $\overline{X}_{j+1}$ in $\mathcal{P}$. $\overline{X}_j$ ends with a primitive lower block X_1 and $\overline{X}_{j+1}$ starts with another primitive lower block X_2. Because each primitive lower block of X is contained in one of the lower blocks occurring in $\mathcal{P}$, X_1 and X_2 are *consecutive* primitive lower blocks of X. By definition, the string Y_j separating them is an internal maximal upper sequence of X.

$\Longleftarrow$ Assume that $\mathcal{P}$ satisfies the conditions in the claim.

Let $\mathcal{P}' = Y'_0X_1Y'_1\ldots X_{r_0}Y'_{r_0}$ for some $r_0 \geq 0$ [1] be the primitive lower block partitioning of X. The strings $Y_0, Y_1, \ldots, Y_r$ occurring in $\mathcal{P}$ are the maximal upper prefix, (some of the) internal maximal upper sequences and the maximal upper suffix of X. Hence, for each Y_j, there exists j_1 with $0 \leq j_1 \leq r_0$ such that Y_j equals the substring Y'_{j_1} from the primitive lower block partitioning $\mathcal{P}'$.

Consider a substring $\overline{X}_j$ with $1 \leq j \leq r$ occurring in $\mathcal{P}$ (see Figure 7.4). $\overline{X}_j$ is preceded in $\mathcal{P}$ by Y_{j-1} and succeeded in $\mathcal{P}$ by Y_j. Let Y'_{j_1-1} be the substring in $\mathcal{P}'$ that is equal to Y_{j-1} and let Y'_{j_2} be the substring in $\mathcal{P}'$ that is equal to Y_j. Clearly, $0 \leq j_1 - 1 < j_2 \leq r_0$, and hence $j_1 \leq j_2$. It is easily verified that $\overline{X}_j = X_{j_1}Y'_{j_1}\ldots X_{j_2}$ satisfies the definition of a lower block of X.

By definition, a primitive lower block of X neither intersects with the maximal upper prefix, nor with any internal maximal upper sequence or the maximal upper suffix of X. Then it certainly does not intersect with any of the strings $Y_0, Y_1, \ldots, Y_r$ occurring in $\mathcal{P}$. Hence, for each primitive lower block X_{j_0} of X, there must be a j with $1 \leq j \leq r$, such that $\overline{X}_j$ contains X_{j_0}.

We conclude that $\mathcal{P}$ is a lower block partitioning of X. □

When we combine the above result with Definition 7.8 and Lemma 7.10(1), we find:

Lemma 7.21 *Let X be a nick free formal DNA molecule and let $\mathcal{P} = Y_0\overline{X}_1Y_1\ldots\overline{X}_rY_r$ for some $r \geq 0$ be a lower block partitioning of X. For $j = 0, 1, \ldots, r$, Y_j is either a*

[1] In fact, $r_0 \geq 1$, because we assume that X contains at least one primitive lower block.

maximal upper sequence of X, or the empty string λ. The latter case may occur only for $j = 0$ and $j = r$.

Definition 7.18 and Lemma 7.21 provide us with the following intuitive understanding of a lower block partitioning. A lower block partitioning of a nick free formal DNA molecule X consists of subsequences $Y_0, \overline{X}_1, Y_1, \ldots, \overline{X}_r, Y_r$ of components of X, where each subsequence Y_j contains at least one upper component, but does not contain lower components, and each $\overline{X}_j$ starts with a 'block' and ends with a 'block' . Exceptions to this (may) occur if X is double-complete, or if the maximal upper prefix Y_0 or the maximal upper suffix Y_r of X is empty.

By Lemma 7.20, a lower block partitioning $Y_0\overline{X}_1Y_1 \ldots \overline{X}_rY_r$ of a nick free formal DNA molecule X is completely determined by the internal maximal upper sequences $Y_1, \ldots, Y_{r-1}$ occurring in it. Each internal maximal upper sequence of X may or may not occur. Because the occurrence of one internal maximal upper sequence is independent of the occurrence of the others, we have the following result:

Lemma 7.22 *Let X be a nick free formal DNA molecule. Then the number of different lower block partitionings of X is $2^{n_{imus}(X)}$.*

In Figure 7.3, we gave four different lower block partitionings of our example formal DNA molecule X. For this molecule, we have $B_{\downarrow}(X) = 3$ and, by Lemma 7.9(2), $n_{\text{imus}}(X) = 2$. By the above, there do not exist other lower block partitionings of X.

We can relate the values of $B_{\uparrow}$, $B_{\downarrow}$ and $n_{\updownarrow}$ for a nick free formal DNA molecule X to the values for the substrings of X in a lower block partitioning:

Lemma 7.23 *Let X be a nick free formal DNA molecule and let $Y_0\overline{X}_1Y_1 \ldots \overline{X}_rY_r$ for some $r \geq 0$ be a lower block partitioning of X.*

1. $B_{\uparrow}(X) = B_{\uparrow}(Y_0) + B_{\uparrow}(\overline{X}_1) + B_{\uparrow}(Y_1) + \cdots + B_{\uparrow}(\overline{X}_r) + B_{\uparrow}(Y_r)$,
   where for $j = 0, 1, \ldots, r$,

$$B_{\uparrow}(Y_j) = \begin{cases} 0 & \text{if } Y_j \text{ is empty or double-complete} \\ 1 & \text{otherwise.} \end{cases}$$

2. $B_{\downarrow}(X) = B_{\downarrow}(\overline{X}_1) + \cdots + B_{\downarrow}(\overline{X}_r)$,
   and for $j = 0, 1, \ldots, r$, $B_{\downarrow}(Y_j) = 0$.

3. $n_{\updownarrow}(X) = n_{\updownarrow}(Y_0) + n_{\updownarrow}(\overline{X}_1) + n_{\updownarrow}(Y_1) + \cdots + n_{\updownarrow}(\overline{X}_r) + n_{\updownarrow}(Y_r)$.

Note that if $B_{\downarrow}(X) = 0$, then by Lemma 7.19, the only lower block partitioning of X is $Y_0 = X$. In that case, $r = 0$ and the claims are nearly trivial.

Proof: Let us use $\mathcal{P}$ to denote the lower block partitioning under consideration.

By definition, each lower block $\overline{X}_j$ occurring in $\mathcal{P}$ is an alternating sequence of primitive lower blocks and maximal upper sequences of X, containing at least one primitive lower block. By Lemma 7.21, each substring Y_j occurring in $\mathcal{P}$ is either a maximal upper sequence of X or the empty string λ. In both cases, Y_j can be considered as an alternating sequence of primitive lower blocks and maximal upper sequences of X – one which does not contain any primitive lower block.

Certain notions are not defined, and certain results are not applicable to an empty substring Y_j. Therefore, in the remainder of the proof, we will often restrict ourselves to the non-empty substrings Y_j. Note that only the maximal upper prefix Y_0 of X and the maximal upper suffix Y_r of X may be empty, and that by definition, $B_\uparrow(\lambda) = B_\downarrow(\lambda) = n_\updownarrow(\lambda) = 0$.

1. If X is double-complete, then by Lemma 7.19, the only lower block partitioning of X is $Y_0 = X$. In this case, the claim is trivially valid.

   We now assume that X is not double-complete. By Lemma 7.12(4b), each maximal upper sequence of X contains at least one upper component, and by Lemma 7.13(3), $B_\uparrow(X) = n_{\text{mus}}(X)$.

   Each non-empty substring Y_j or $\overline{X}_j$ occurring in $\mathcal{P}$ can be considered as an alternating sequence of primitive lower blocks and maximal upper sequences of X. In particular, the maximal upper sequences of X are distributed over these substrings. By Lemma 7.14(2), the maximal upper sequences of one such substring are precisely the maximal upper sequences of X occurring in it. In particular, for each non-empty substring Y_j, $n_{\text{mus}}(Y_j) = 1$.

   By Lemma 7.17(1), none of the lower blocks $\overline{X}_j$ is double-complete. Because each maximal upper sequence of X contains at least one upper component, none of the non-empty substrings Y_j is double-complete, either. Hence, the number of maximal upper sequences of any of the non-empty substrings $\overline{X}_j$ and Y_j is equal to the number of primitive upper blocks of it.

   Consequently,

$$\begin{aligned} B_\uparrow(X) &= n_{\text{mus}}(X) \\ &= \sum_{\text{non-empty } Y_j's} n_{\text{mus}}(Y_j) + \sum_{j=1}^{r} n_{\text{mus}}(\overline{X}_j) \\ &= \sum_{\text{non-empty } Y_j's} B_\uparrow(Y_j) + \sum_{j=1}^{r} B_\uparrow(\overline{X}_j) \\ &= \sum_{j=0}^{r} B_\uparrow(Y_j) + \sum_{j=1}^{r} B_\uparrow(\overline{X}_j). \end{aligned}$$

2. By the definition of a lower block partitioning, the primitive lower blocks of X are distributed over the lower blocks $\overline{X}_1, \ldots, \overline{X}_r$. By Lemma 7.17(2a), for $j = 1, \ldots, r$, the number of primitive lower blocks of X occurring in $\overline{X}_j$ is equal to $B_\downarrow(\overline{X}_j)$. Consequently,

$$B_\downarrow(X) = B_\downarrow(\overline{X}_1) + \cdots + B_\downarrow(\overline{X}_r)$$

   Each non-empty substring Y_j is a maximal upper sequence, which, by Lemma 7.12(2), does not contain any lower component. Hence by definition, whether a substring Y_j is empty or not, $B_\downarrow(Y_j) = 0$.

3. By definition, a primitive lower block of X is a sequence of components of X. By Lemma 7.12(2), so is a maximal upper sequence of X. Then also each lower block $\overline{X}_j$ and each substring Y_j occurring in $\mathcal{P}$ is a sequence of components of X.

   Hence, the components of X are distributed over the non-empty substrings Y_j and $\overline{X}_j$. This holds in particular for the double components of X. By Lemma 6.3, the double components of such a non-empty substring are precisely the double components of X occurring in it.

   Consequently,

$$\begin{aligned} n_{\updownarrow}(X) &= \sum_{\text{non-empty } Y_j's} n_{\updownarrow}(Y_j) + \sum_{j=1}^{r} n_{\updownarrow}(\overline{X}_j) \\ &= \sum_{j=0}^{r} n_{\updownarrow}(Y_j) + \sum_{j=1}^{r} n_{\updownarrow}(\overline{X}_j). \end{aligned}$$

□

Recall that Theorem 6.31 gives lower bounds on the length $|E|$ of a DNA expression E denoting a certain formal DNA molecule X, in terms of $B_{\uparrow}(X)$, $B_{\downarrow}(X)$, $n_{\updownarrow}(X)$ and $|X|_{\mathcal{A}}$. We are now ready to construct DNA expressions that achieve these lower bounds for nick free formal DNA molecules with at least one single-stranded component.

Theorem 7.24 *Let X be a nick free formal DNA molecule which contains at least one single-stranded component, and let $x'_1 \dots x'_k$ for some $k \geq 1$ be the decomposition of X.*

1. *If $B_{\uparrow}(X) \geq B_{\downarrow}(X)$, then*

   - *let $\mathcal{P} = Y_0\overline{X}_1Y_1 \dots \overline{X}_rY_r$ for some $r \geq 0$ be an arbitrary lower block partitioning of X;*
   - *for $j = 1, \dots, r$, let E_j be an arbitrary minimal DNA expression denoting $\overline{X}_j$;*
   - *for $j = 0, 1, \dots, r$, let $Y_j = x'_{a_j} \dots x'_{b_j}$ for some $a_j \geq 1$ and $b_j \leq k$;*
   - *for $j = 0, 1, \dots, r$ and for $i = a_j, \dots, b_j$, let*

$$\varepsilon_i = \begin{cases} \alpha_i & \text{if } x'_i = \binom{\alpha_i}{-} \text{ for an } \mathcal{N}\text{-word } \alpha_i \\ \langle \updownarrow \alpha_i \rangle & \text{if } x'_i = \binom{\alpha_i}{c(\alpha_i)} \text{ for an } \mathcal{N}\text{-word } \alpha_i; \end{cases} \quad \text{and} \tag{7.2}$$

   - *let*

$$E = \langle \uparrow \varepsilon_{a_0} \dots \varepsilon_{b_0} E_1 \varepsilon_{a_1} \dots \varepsilon_{b_1} \ \dots \ E_r \varepsilon_{a_r} \dots \varepsilon_{b_r} \rangle . \tag{7.3}$$

   Then

   (a) all ingredients needed to construct E (i.e., the lower block partitioning $\mathcal{P}$, the minimal DNA expressions E_j, the indices a_j and b_j, and the arguments ε_i) are well defined, and

   (b) E is a minimal DNA expression denoting X, and

$$|E| = 3 + 3 \cdot B_{\downarrow}(X) + 3 \cdot n_{\updownarrow}(X) + |X|_{\mathcal{A}}. \tag{7.4}$$

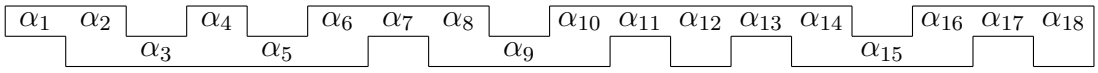


Figure 7.5: The formal DNA molecule from Figure 7.1 with occurring $\mathcal{N}$-words indicated.

2. *If* $B_{\downarrow}(X) \geq B_{\uparrow}(X)$, *then*

   - *let* $\mathcal{P} = Y_0\overline{X}_1Y_1 \ldots \overline{X}_rY_r$ *for some* $r \geq 0$ *be an arbitrary upper block partitioning of* X;
   - *for* $j = 1, \ldots, r$, *let* E_j *be an arbitrary minimal DNA expression denoting* $\overline{X}_j$;
   - *for* $j = 0, 1, \ldots, r$, *let* $Y_j = x'_{a_j} \ldots x'_{b_j}$ *for some* $a_j \geq 1$ *and* $b_j \leq k$;
   - *for* $j = 0, 1, \ldots, r$ *and for* $i = a_j, \ldots, b_j$, *let*

$$\varepsilon_i = \begin{cases} \alpha_i & \text{if } x'_i = \binom{-}{\alpha_i} \text{ for an } \mathcal{N}\text{-word } \alpha_i \\ \langle \updownarrow \alpha_i \rangle & \text{if } x'_i = \binom{\alpha_i}{c(\alpha_i)} \text{ for an } \mathcal{N}\text{-word } \alpha_i; \end{cases} \qquad \text{and}$$

   - *let*

$$E = \langle \downarrow \varepsilon_{a_0} \ldots \varepsilon_{b_0} E_1 \varepsilon_{a_1} \ldots \varepsilon_{b_1} \ \ldots \ E_r \varepsilon_{a_r} \ldots \varepsilon_{b_r} \rangle \,. \tag{7.5}$$

   Then

   (a) *all ingredients needed to construct* E *(i.e., the upper block partitioning* $\mathcal{P}$, *the minimal DNA expressions* E_j, *the indices* a_j *and* b_j, *and the arguments* ε_i*) are well defined, and*

   (b) E *is a minimal DNA expression denoting* X, *and*

$$|E| = 3 + 3 \cdot B_{\uparrow}(X) + 3 \cdot n_{\updownarrow}(X) + |X|_{\mathcal{A}}. \tag{7.6}$$

Claim 2 is completely analogous to Claim 1. We only consider an *upper* block partitioning $\mathcal{P}$, *lower* components $\binom{-}{\alpha_i}$ in the definition of the ε_i's, and a $\downarrow$-*expression* $E = \langle \downarrow \ldots \rangle$, whose length $|E|$ is expressed in terms of (a.o.) $B_{\uparrow}(X)$.

Note that, if, for example, $B_{\uparrow}(X) \geq B_{\downarrow}(X)$ and the maximal upper prefix Y_0 of X is empty, then we may choose $a_0 = 1$ and $b_0 = 0$ in Claim 1. If, on the other hand, a substring Y_j is non-empty (which is certainly the case if $1 \leq j \leq r-1$), then $1 \leq a_j \leq b_j \leq k$.

Note further that the construction for minimal DNA expressions described in this result is recursive. The minimal DNA expression specified, which denotes the entire formal DNA molecule X, may have arguments E_j that are themselves minimal DNA expressions denoting formal DNA submolecules $\overline{X}_j$ of X. In the proof, we will see that this recursion is well defined.

Note finally that if $B_{\uparrow}(X) = B_{\downarrow}(X)$, then both Claim 1 and Claim 2 are applicable, and we have both a minimal $\uparrow$-expression and a minimal $\downarrow$-expression denoting X. This is the case in Example 7.26 below.

Example 7.25 In Figure 7.5, we have specified names for the components of the formal DNA molecule from (a.o.) Figure 7.1 and Figure 7.3. For this formal DNA molecule X, we have $B_{\uparrow}(X) = 4$ and $B_{\downarrow}(X) = 3$. Hence, by Theorem 7.24(1), we can construct a

minimal DNA expression denoting X from a lower block partitioning of X. Because X has two internal maximal upper sequences ($\binom{\alpha_7}{-}$ and $\binom{\alpha_{11}}{-}\binom{\alpha_{12}}{c(\alpha_{12})}\binom{\alpha_{13}}{-}$), there are, by Lemma 7.22, four different lower block partitionings of X. We will consider two of them, the ones depicted in Figure 7.3(a3) and (a4).

For the former lower block partitioning, $r = 2$ and

$$\begin{aligned}
Y_0 &= \tbinom{\alpha_1}{-}, \\
\overline{X}_1 &= \tbinom{\alpha_2}{c(\alpha_2)}\tbinom{-}{\alpha_3}\tbinom{\alpha_4}{c(\alpha_4)}\tbinom{-}{\alpha_5}\tbinom{\alpha_6}{c(\alpha_6)}\tbinom{\alpha_7}{-}\tbinom{\alpha_8}{c(\alpha_8)}\tbinom{-}{\alpha_9}\tbinom{\alpha_{10}}{c(\alpha_{10})}, \\
Y_1 &= \tbinom{\alpha_{11}}{-}\tbinom{\alpha_{12}}{c(\alpha_{12})}\tbinom{\alpha_{13}}{-}, \\
\overline{X}_2 &= \tbinom{\alpha_{14}}{c(\alpha_{14})}\tbinom{-}{\alpha_{15}}\tbinom{\alpha_{16}}{c(\alpha_{16})}, \\
Y_2 &= \tbinom{\alpha_{17}}{-}\tbinom{\alpha_{18}}{c(\alpha_{18})}.
\end{aligned}$$

We have $B_{\downarrow}(\overline{X}_1) = 2 > B_{\uparrow}(\overline{X}_1) = 1$. When we (recursively) apply Theorem 7.24(2) to $\overline{X}_1$ and Theorem 7.24(1) to the primitive upper block $\binom{\alpha_6}{c(\alpha_6)}\binom{\alpha_7}{-}\binom{\alpha_8}{c(\alpha_8)}$ of $\overline{X}_1$, we find that a minimal DNA expression denoting $\overline{X}_1$ is

$$E_1 = \langle\downarrow \langle\updownarrow \alpha_2\rangle \alpha_3 \langle\updownarrow \alpha_4\rangle \alpha_5 \langle\uparrow \langle\updownarrow \alpha_6\rangle \alpha_7 \langle\updownarrow \alpha_8\rangle\rangle \alpha_9 \langle\updownarrow \alpha_{10}\rangle\rangle\,.$$

Further, $B_{\downarrow}(\overline{X}_2) = 1 > B_{\uparrow}(\overline{X}_2) = 0$, and again by Theorem 7.24(2), a minimal DNA expression denoting $\overline{X}_2$ is

$$E_2 = \langle\downarrow \langle\updownarrow \alpha_{14}\rangle \alpha_{15} \langle\updownarrow \alpha_{16}\rangle\rangle\,.$$

Now, by Theorem 7.24(1), a minimal DNA expression denoting X is

$$\begin{aligned}
E = \langle\uparrow \quad & \alpha_1 \quad \langle\downarrow \langle\updownarrow \alpha_2\rangle \alpha_3 \langle\updownarrow \alpha_4\rangle \alpha_5 \langle\uparrow \langle\updownarrow \alpha_6\rangle \alpha_7 \langle\updownarrow \alpha_8\rangle\rangle \alpha_9 \langle\updownarrow \alpha_{10}\rangle\rangle \\
& \alpha_{11} \langle\updownarrow \alpha_{12}\rangle \alpha_{13} \quad \langle\downarrow \langle\updownarrow \alpha_{14}\rangle \alpha_{15} \langle\updownarrow \alpha_{16}\rangle\rangle \quad \alpha_{17} \langle\updownarrow \alpha_{18}\rangle \quad \rangle\,.
\end{aligned} \tag{7.7}$$

Here, we used additional white space to clearly indicate the arguments corresponding to different substrings $\overline{X}_j$ and Y_j of the lower block partitioning.

According to the lower block partitioning depicted in Figure 7.3(a4), $r = 1$ and

$$\begin{aligned}
Y_0 &= \tbinom{\alpha_1}{-}, \\
\overline{X}_1 &= \tbinom{\alpha_2}{c(\alpha_2)}\tbinom{-}{\alpha_3}\tbinom{\alpha_4}{c(\alpha_4)}\tbinom{-}{\alpha_5}\tbinom{\alpha_6}{c(\alpha_6)}\tbinom{\alpha_7}{-}\tbinom{\alpha_8}{c(\alpha_8)}\tbinom{-}{\alpha_9} \\
&\quad \cdot\tbinom{\alpha_{10}}{c(\alpha_{10})}\tbinom{\alpha_{11}}{-}\tbinom{\alpha_{12}}{c(\alpha_{12})}\tbinom{\alpha_{13}}{-}\tbinom{\alpha_{14}}{c(\alpha_{14})}\tbinom{-}{\alpha_{15}}\tbinom{\alpha_{16}}{c(\alpha_{16})}, \\
Y_1 &= \tbinom{\alpha_{17}}{-}\tbinom{\alpha_{18}}{c(\alpha_{18})}.
\end{aligned}$$

We now have $B_{\downarrow}(\overline{X}_1) = 3$ and $B_{\uparrow}(\overline{X}_1) = 2$. By Theorem 7.24(2), a minimal DNA expression E_1 denoting $\overline{X}_1$ can be constructed from an upper block partitioning of $\overline{X}_1$. Contrary to the previous case, $\overline{X}_1$ contains an internal maximal lower sequence, $\binom{-}{\alpha_9}$. Hence, there exist two different upper block partitionings of $\overline{X}_1$, which yield different minimal DNA expressions E_1. We arbitrarily choose the primitive upper block partitioning, which includes all maximal lower sequences of $\overline{X}_1$, in particular $\binom{-}{\alpha_9}$. For the primitive upper

blocks $\binom{\alpha_6}{c(\alpha_6)}\binom{\alpha_7}{-}\binom{\alpha_8}{c(\alpha_8)}$ and $\binom{\alpha_{10}}{c(\alpha_{10})}\binom{\alpha_{11}}{-}\binom{\alpha_{12}}{c(\alpha_{12})}\binom{\alpha_{13}}{-}\binom{\alpha_{14}}{c(\alpha_{14})}$ of $\overline{X}_1$, we find a minimal DNA-expression with Theorem 7.24(1). The resulting minimal DNA-expression for $\overline{X}_1$ is

$$E_1 = \langle\downarrow\ \langle\updownarrow\,\alpha_2\rangle\,\alpha_3\,\langle\updownarrow\,\alpha_4\rangle\,\alpha_5\ \ \langle\uparrow\,\langle\updownarrow\,\alpha_6\rangle\,\alpha_7\,\langle\updownarrow\,\alpha_8\rangle\rangle\ \ \alpha_9 \\ \langle\uparrow\,\langle\updownarrow\,\alpha_{10}\rangle\,\alpha_{11}\,\langle\updownarrow\,\alpha_{12}\rangle\,\alpha_{13}\,\langle\updownarrow\,\alpha_{14}\rangle\rangle\ \ \alpha_{15}\,\langle\updownarrow\,\alpha_{16}\rangle\,\rangle$$

and the corresponding minimal DNA-expression denoting X is

$$E = \langle\uparrow\quad\alpha_1\quad\langle\downarrow\ \langle\updownarrow\,\alpha_2\rangle\,\alpha_3\,\langle\updownarrow\,\alpha_4\rangle\,\alpha_5\ \ \langle\uparrow\,\langle\updownarrow\,\alpha_6\rangle\,\alpha_7\,\langle\updownarrow\,\alpha_8\rangle\rangle\ \ \alpha_9 \\ \langle\uparrow\,\langle\updownarrow\,\alpha_{10}\rangle\,\alpha_{11}\,\langle\updownarrow\,\alpha_{12}\rangle\,\alpha_{13}\,\langle\updownarrow\,\alpha_{14}\rangle\rangle\ \ \alpha_{15}\,\langle\updownarrow\,\alpha_{16}\rangle\,\rangle\quad\alpha_{17}\,\langle\updownarrow\,\alpha_{18}\rangle\quad\rangle\,.$$

Indeed, both minimal DNA expressions for X have length

$$|E| = 39 + |X|_{\mathcal{A}} = 3 + 3\cdot 3 + 3\cdot 9 + |X|_{\mathcal{A}} = 3 + 3\cdot B_{\downarrow}(X) + 3\cdot n_{\updownarrow}(X) + |X|_{\mathcal{A}}.$$

■

Example 7.26 Consider the nick free formal DNA molecule

$$X = \tbinom{\alpha_1}{-}\tbinom{\alpha_2}{c(\alpha_2)}\tbinom{-}{\alpha_3}\tbinom{\alpha_4}{c(\alpha_4)}\tbinom{\alpha_5}{-}\tbinom{\alpha_6}{c(\alpha_6)}\tbinom{\alpha_7}{-}\tbinom{\alpha_8}{c(\alpha_8)}\tbinom{-}{\alpha_9}\tbinom{\alpha_{10}}{c(\alpha_{10})}, \tag{7.8}$$

for which $B_{\uparrow}(X) = B_{\downarrow}(X) = 2$. By Theorem 7.24, we can construct minimal $\uparrow$-expressions (based on lower block partitionings) and minimal $\downarrow$-expressions (based on upper block partitionings) for X. By Lemma 7.22 and Lemma 7.9(2), X has two lower block partitionings and two upper block partitionings. We have depicted them in Figure 7.6. We carry out the construction for the upper block partitioning $Y_0\overline{X}_1Y_1$ from Figure 7.6(d). Here $Y_0 = \lambda$,

$$\overline{X}_1 = \tbinom{\alpha_1}{-}\tbinom{\alpha_2}{c(\alpha_2)}\tbinom{-}{\alpha_3}\tbinom{\alpha_4}{c(\alpha_4)}\tbinom{\alpha_5}{-}\tbinom{\alpha_6}{c(\alpha_6)}\tbinom{\alpha_7}{-}\tbinom{\alpha_8}{c(\alpha_8)}$$

and $Y_1 = \binom{-}{\alpha_9}\binom{\alpha_{10}}{c(\alpha_{10})}$. By Theorem 7.24(2), the resulting minimal $\downarrow$-expression is $E_d = \langle\downarrow E_1\alpha_9\,\langle\updownarrow\,\alpha_{10}\rangle\rangle$, where E_1 is a minimal DNA expression denoting $\overline{X}_1$.

As $B_{\uparrow}(\overline{X}_1) = 2 > B_{\downarrow}(\overline{X}_1) = 1$, we can (recursively) apply Theorem 7.24(1) to construct E_1. The result is

$$E_1 = \langle\uparrow\,\alpha_1\,\langle\downarrow\,\langle\updownarrow\,\alpha_2\rangle\,\alpha_3\,\langle\updownarrow\,\alpha_4\rangle\rangle\,\alpha_5\,\langle\updownarrow\,\alpha_6\rangle\,\alpha_7\,\langle\updownarrow\,\alpha_8\rangle\rangle\,.$$

This way, we can construct a minimal DNA expression denoting X for each of the four partitionings from Figure 7.6:

$$E_a = \langle\uparrow\,\alpha_1\quad\langle\downarrow\,\langle\updownarrow\,\alpha_2\rangle\,\alpha_3\,\langle\updownarrow\,\alpha_4\rangle\rangle\quad\alpha_5\,\langle\updownarrow\,\alpha_6\rangle\,\alpha_7\quad\langle\downarrow\,\langle\updownarrow\,\alpha_8\rangle\,\alpha_9\,\langle\updownarrow\,\alpha_{10}\rangle\rangle\rangle\,, \tag{7.9}$$

$$E_b = \langle\uparrow\,\alpha_1\quad\langle\downarrow\,\langle\updownarrow\,\alpha_2\rangle\,\alpha_3\,\langle\uparrow\,\langle\updownarrow\,\alpha_4\rangle\,\alpha_5\,\langle\updownarrow\,\alpha_6\rangle\,\alpha_7\,\langle\updownarrow\,\alpha_8\rangle\rangle\,\alpha_9\,\langle\updownarrow\,\alpha_{10}\rangle\rangle\rangle\,, \tag{7.10}$$

$$E_c = \langle\downarrow\,\langle\uparrow\,\alpha_1\,\langle\updownarrow\,\alpha_2\rangle\rangle\quad\alpha_3\quad\langle\uparrow\,\langle\updownarrow\,\alpha_4\rangle\,\alpha_5\,\langle\updownarrow\,\alpha_6\rangle\,\alpha_7\,\langle\updownarrow\,\alpha_8\rangle\rangle\quad\alpha_9\,\langle\updownarrow\,\alpha_{10}\rangle\rangle\,, \tag{7.11}$$

$$E_d = \langle\downarrow\,\langle\uparrow\,\alpha_1\,\langle\downarrow\,\langle\updownarrow\,\alpha_2\rangle\,\alpha_3\,\langle\updownarrow\,\alpha_4\rangle\rangle\,\alpha_5\,\langle\updownarrow\,\alpha_6\rangle\,\alpha_7\,\langle\updownarrow\,\alpha_8\rangle\rangle\quad\alpha_9\,\langle\updownarrow\,\alpha_{10}\rangle\rangle\,. \tag{7.12}$$

All these minimal DNA expressions have length

$$\begin{aligned}|E| &= 24 + |X|_{\mathcal{A}} = 3 + 3\cdot 2 + 3\cdot 5 + |X|_{\mathcal{A}} \\ &= 3 + 3\cdot B_{\downarrow}(X) + 3\cdot n_{\updownarrow}(X) + |X|_{\mathcal{A}} = 3 + 3\cdot B_{\uparrow}(X) + 3\cdot n_{\updownarrow}(X) + |X|_{\mathcal{A}}.\end{aligned}$$

■

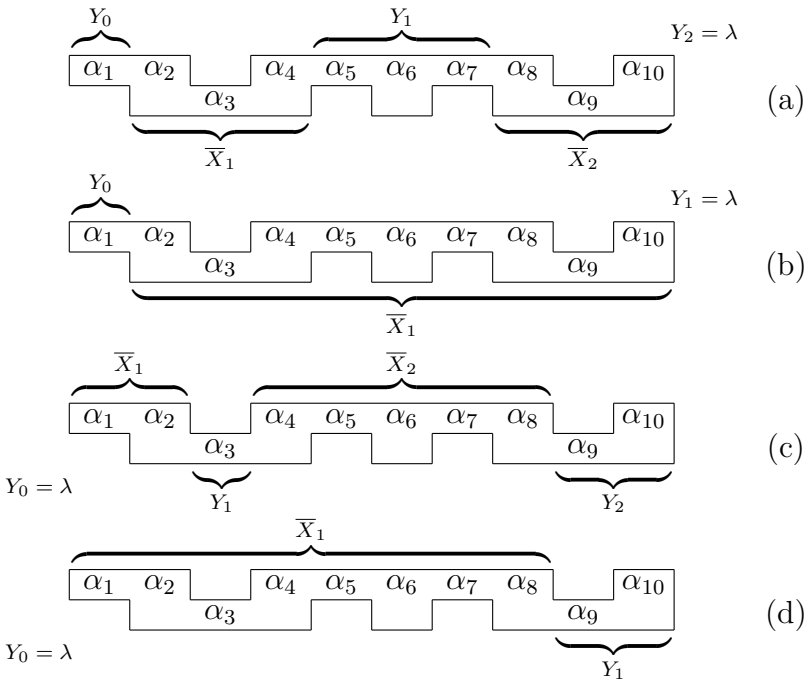


Figure 7.6: Partitionings of the formal DNA molecule X from Example 7.26. (a) The primitive lower block partitioning of X. (b) The second lower block partitioning of X. (c) The primitive upper block partitioning of X. (d) The second upper block partitioning of X.

In Example 7.26, each lower block partitioning and each upper block partitioning of X yielded a single minimal DNA expression. This is not always the case. As we observed in Example 7.25, there may be more than one minimal DNA expression E_i denoting a lower block $\overline{X}_i$ occurring in a lower block partitioning. Then we also find more than one minimal $\uparrow$-expression corresponding to the same lower block partitioning. This is the case if $B_{\downarrow}(\overline{X}_i) \geq 3$. Of course, an analogous property holds for minimal $\downarrow$-expressions corresponding to an upper block partitioning. In Section 8.5, we address the number of minimal DNA expressions denoting a given formal DNA molecule.

Proof of Theorem 7.24: First we prove for Claim 1 that most ingredients needed to construct E are well defined. Assuming that also the last ingredients are well defined, we prove that E is indeed a DNA expression denoting X. The proof of the corresponding part of Claim 2 is completely analogous. Later, we will prove the remaining parts of both claims.

- Lower block partitionings are defined for every nick free formal DNA molecule. By Lemma 7.22, there exist $2^{n_{\text{imus}}(X)} \geq 1$ lower block partitionings for X. We may choose any of these. For example, we may let $\mathcal{P}$ be the primitive lower block partitioning.

  By Lemma 7.21, Y_0 is either a maximal upper sequence of X or the empty string λ, $Y_1, \ldots, Y_{r-1}$ are maximal upper sequences, and Y_r is again either a maximal upper sequence of the empty string λ. If $Y_0 = \lambda$, then we may take $a_0 = 1$ and $b_0 = 0$. Analogously, if $Y_r = \lambda$, then we may take $a_r = k+1$ and $b_r = k$. By Lemma 7.12(2), each maximal upper sequence Y_j is an alternating sequence of upper components and double components of X. Hence, there exist a_j and b_j with $1 \leq a_j \leq b_j \leq k$

such that $Y_j = x'_{a_j} \dots x'_{b_j}$.

Moreover, for each x'_i with $a_j \leq i \leq b_j$ for some j with $0 \leq j \leq r$, there exists an $\mathcal{N}$-word α_i, such that x'_i is either (the upper component) $\binom{\alpha_i}{-}$, or (the double component) $\binom{\alpha_i}{c(\alpha_i)}$. Consequently, the arguments ε_i are well defined.

- Let us, for a moment, assume that the minimal DNA expressions $E_1, \dots, E_r$ denoting the lower blocks $\overline{X}_1, \dots, \overline{X}_r$, respectively, are also well defined, i.e., that $\overline{X}_1, \dots, \overline{X}_r$ are expressible.

  Clearly, for each applicable i, $\mathcal{S}^+(\varepsilon_i) = x'_i$. Because x'_i is either an upper component or a double component of X, consecutive arguments ε_i and ε_{i+1} fit together by upper strands.

  For $j = 1, \dots, r$, if $Y_{j-1} = x'_{a_{j-1}} \dots x'_{b_{j-1}} \neq \lambda$ (which is certainly the case if $j \geq 2$), then it is a maximal upper sequence. It is succeeded in X by the lower block $\overline{X}_j$. By Lemma 7.12(1b), $R(\mathcal{S}^+(\varepsilon_{b_{j-1}})) = R(x'_{b_{j-1}}) = R(Y_{j-1}) = R(\binom{\alpha_{b_{j-1}}}{-}) \in \mathcal{A}_+$ and $L(\mathcal{S}^+(E_j)) = L(\overline{X}_j) = L(x'_{b_{j-1}+1}) \in \mathcal{A}_\pm$. Hence, the argument $\varepsilon_{b_{j-1}}$ prefits E_j by upper strands.

  Analogously, for $j = 1, \dots, r$, if $Y_j = x'_{a_j} \dots x'_{b_j} \neq \lambda$ (which is certainly the case if $j \leq r-1$), then the argument E_j prefits ε_{a_j} by upper strands.

  Consequently, E is indeed a DNA expression.

- For notational convenience, we assume that both Y_0 and Y_r are non-empty.[2] Then by Lemma 5.10,

$$E \equiv E' = \langle\uparrow \langle\uparrow \varepsilon_{a_0} \dots \varepsilon_{b_0}\rangle E_1 \langle\uparrow \varepsilon_{a_1} \dots \varepsilon_{b_1}\rangle \ \dots \ E_r \langle\uparrow \varepsilon_{a_r} \dots \varepsilon_{b_r}\rangle\rangle\,. \tag{7.13}$$

  For an arbitrary j with $0 \leq j \leq r$, we consider the argument $\langle\uparrow \varepsilon_{a_j} \dots \varepsilon_{b_j}\rangle$ of E'. We established before that for $i = a_j, \dots, b_j$, $\mathcal{S}^+(\varepsilon_i) = x'_i$ is either an upper component or a double component of X. By definition, such components are nick free. Moreover, the upper components and double components occur in $x'_{a_j} \dots x'_{b_j}$, alternately. In particular, we do not have two consecutive double components. Hence, the operator $\uparrow$ does not introduce (lower) nick letters between its arguments, and

$$\mathcal{S}(\langle\uparrow \varepsilon_{a_j} \dots \varepsilon_{b_j}\rangle) = \nu^+(x'_{a_j}) \dots \nu^+(x'_{b_j}) = x'_{a_j} \dots x'_{b_j} = Y_j.$$

  We use this to determine $\mathcal{S}(E')$:

$$\begin{aligned} &\mathcal{S}(E') \\ &\quad = \nu^+(Y_0) y_1 \nu^+(\overline{X}_1) y_2 \nu^+(Y_1) y_3 \dots y_{2r-1} \nu^+(\overline{X}_r) y_{2r} \nu^+(Y_r) \\ &\quad = Y_0 y_1 \overline{X}_1 y_2 Y_1 y_3 \dots y_{2r-1} \overline{X}_r y_{2r} Y_r, \end{aligned} \tag{7.14}$$

  where the y_i's are defined as in (4.3). The second equality in (7.14) holds because each maximal upper sequence Y_j and each lower block $\overline{X}_j$ is nick free.

  By Lemma 7.12(1), for $j = 1, \dots, r$, $R(Y_{j-1}) \in \mathcal{A}_+$, $L(\overline{X}_j), R(\overline{X}_j) \in \mathcal{A}_\pm$ and $L(Y_j) \in \mathcal{A}_+$. Consequently, all y_i's are empty, and thus

$$\mathcal{S}(E) = \mathcal{S}(E') = Y_0 \overline{X}_1 Y_1 \dots \overline{X}_r Y_r = X.$$

[2] If, for example, $Y_0 = \lambda$, then the 'argument' $\langle\uparrow \varepsilon_{a_0} \dots \varepsilon_{b_0}\rangle = \langle\uparrow\rangle$ in (7.13) would not make sense.

In order to complete the proof, we must prove that the minimal DNA expressions E_j occurring in Claims 1 and 2 are well defined, and that E has the specified length (see (7.4) and (7.6)) and is minimal. We do this simultaneously for both claims, by induction on the lower of $B_\uparrow(X)$ and $B_\downarrow(X)$. For the ease of formulation, our induction hypothesis will be that Claims 1 and 2 are valid for certain formal DNA molecules X, but we will only prove the parts of the claims we have not yet proved.

- Assume that either $B_\uparrow(X) = 0$, or $B_\downarrow(X) = 0$.

  1. We assume that $B_\uparrow(X) \geq B_\downarrow(X) = 0$. By Lemma 7.19, X does not contain any lower component and the only lower block partitioning of X is $\mathcal{P} = Y_0 = X$. Because $\mathcal{P}$ does not contain any lower block $\overline{X}_j$, the construction of E in Claim 1 does not require any minimal DNA expression E_j:

     $$E = \langle \uparrow \varepsilon_1 \dots \varepsilon_k \rangle ,$$

     where for $i = 1, \dots, k$, ε_i is defined by (7.2). For $i = 1, \dots, k$,

     $$|\varepsilon_i| = \begin{cases} |\alpha_i| = |\binom{\alpha_i}{-}|_{\mathcal{A}} = |x'_i|_{\mathcal{A}} \\ \qquad \text{if } x'_i = \binom{\alpha_i}{-} \text{ for an } \mathcal{N}\text{-word } \alpha_i \\ |\langle \updownarrow \alpha_i \rangle| = 3 + |\alpha_i| = 3 + |\binom{\alpha_i}{c(\alpha_i)}|_{\mathcal{A}} = 3 + |x'_i|_{\mathcal{A}} \\ \qquad \text{if } x'_i = \binom{\alpha_i}{c(\alpha_i)} \text{ for an } \mathcal{N}\text{-word } \alpha_i. \end{cases} \tag{7.15}$$

     Consequently,

     $$\begin{aligned} |E| &= 3 + \sum_{i=1}^{k} |\varepsilon_i| \\ &= 3 + 3 \cdot n_\updownarrow(X) + \sum_{i=1}^{k} |x'_i|_{\mathcal{A}} \\ &= 3 + 3 \cdot B_\downarrow(X) + 3 \cdot n_\updownarrow(X) + |X|_{\mathcal{A}}. \end{aligned}$$

     By Theorem 6.31(1), this is the minimal length possible for an $\uparrow$-expression denoting X. In order to conclude that E is a minimal DNA expression for X, we have to verify that there does not exist a shorter $\updownarrow$-expression or $\downarrow$-expression for X.

     By assumption, X contains at least one single-stranded component, which must be an upper component. By definition, the semantics of an $\updownarrow$-expression does not contain any single-stranded component. Consequently, there does not exist any $\updownarrow$-expression denoting X, let alone an $\updownarrow$-expression shorter than E.

     By Lemma 6.13(1a), $B_\uparrow(X)$ must be positive, and hence larger than $B_\downarrow(X)$. By Lemma 6.12(1), $B_\uparrow(X) = 1$. Now, let E' be an arbitrary $\downarrow$-expression denoting X. Then by Theorem 6.31(2), E' is longer than E:

     $$|E'| \geq 3 + 3 \cdot B_\uparrow(X) + 3 \cdot n_\updownarrow(X) + |X|_{\mathcal{A}} = 6 + 3 \cdot n_\updownarrow(X) + |X|_{\mathcal{A}} > |E|.$$

     Hence, only $\uparrow$-expressions denoting X may be minimal, and in particular, E is minimal.

2. The proof of this claim is analogous to that of the previous claim.

We conclude that both claims are valid, if either $B_{\uparrow}(X) = 0$ or $B_{\downarrow}(X) = 0$.

• We now make the induction step. Let $p \geq 0$, and suppose that for each nick free formal DNA molecule X with at least one single-stranded component and either $B_{\uparrow}(X) \leq p$, or $B_{\downarrow}(X) \leq p$, Claims 1 and 2 are valid (induction hypothesis). Now, let X be a nick free formal DNA molecule for which the lower of $B_{\uparrow}(X)$ and $B_{\downarrow}(X)$ is $p+1$.

1. Let us assume that $B_{\uparrow}(X) \geq B_{\downarrow}(X) = p+1$, and let $\mathcal{P} = Y_0\overline{X}_1Y_1 \ldots \overline{X}_rY_r$ be an arbitrary lower block partitioning of X. As $B_{\downarrow}(X) = p+1 \geq 1$, Lemma 7.23(2) implies in particular that $r \geq 1$.

   For an arbitrary j with $1 \leq j \leq r$, we consider the lower block $\overline{X}_j$. Obviously $B_{\downarrow}(\overline{X}_j) \leq B_{\downarrow}(X) = p+1$. Then by Lemma 7.17(2b), $B_{\uparrow}(\overline{X}_j) = B_{\downarrow}(\overline{X}_j) - 1 \leq p$. Now, by the induction hypothesis, we can construct a minimal $\downarrow$-expression E'_j denoting $\overline{X}_j$, for which

   $$|E'_j| = 3+3\cdot B_{\uparrow}(\overline{X}_j)+3\cdot n_{\updownarrow}(\overline{X}_j)+|\overline{X}_j|_{\mathcal{A}} = 3\cdot B_{\downarrow}(\overline{X}_j)+3\cdot n_{\updownarrow}(\overline{X}_j)+|\overline{X}_j|_{\mathcal{A}}. \quad (7.16)$$

   In particular, $\overline{X}_j$ is expressible and minimal DNA expressions denoting $\overline{X}_j$ are defined. Let E_j be an arbitrary minimal DNA expression denoting $\overline{X}_j$. Because E_j and E'_j are equivalent and both of them are minimal, $|E_j| = |E'_j|$.

   We have thus proved that the minimal DNA expressions $E_1, \ldots, E_r$ are well defined, and we also know their lengths. $E_1, \ldots, E_r$ were the last ingredients we needed to construct the string

   $$E = \langle \uparrow \varepsilon_{a_0} \ldots \varepsilon_{b_0} E_1 \varepsilon_{a_1} \ldots \varepsilon_{b_1} \ \ldots \ E_r \varepsilon_{a_r} \ldots \varepsilon_{b_r} \rangle ,$$

   as described in the claim. By the first part of the proof, we know that E is a DNA expression denoting X.

   We consider a substring Y_j with $0 \leq j \leq r$ and an argument ε_i with $a_j \leq i \leq b_j$.

   Also now, the length $|\varepsilon_i|$ of ε_i satisfies (7.15). Because, by Lemma 6.3, the double components of X occurring in Y_j are precisely the double components of Y_j,

   $$\sum_{i=a_j}^{b_j} |\varepsilon_i| = 3 \cdot n_{\updownarrow}(Y_j) + \sum_{i=a_0}^{b_0} |x'_i|_{\mathcal{A}} = 3 \cdot n_{\updownarrow}(Y_j) + |Y_j|_{\mathcal{A}}. \quad (7.17)$$

   When we combine (7.16) and (7.17) with Lemma 7.23(2) and (3), we can establish the length of E:

   $$\begin{aligned} |E| &= 3 + \sum_{j=0}^{r} \sum_{i=a_j}^{b_j} |\varepsilon_i| + \sum_{j=1}^{r} |E_j| \\ &= 3 + \sum_{j=0}^{r} \left(3 \cdot n_{\updownarrow}(Y_j) + |Y_j|_{\mathcal{A}}\right) + \sum_{j=1}^{r} \left(3 \cdot B_{\downarrow}(\overline{X}_j) + 3 \cdot n_{\updownarrow}(\overline{X}_j) + |\overline{X}_j|_{\mathcal{A}}\right) \\ &= 3 + 3 \cdot \sum_{j=1}^{r} B_{\downarrow}(\overline{X}_j) + 3 \cdot \left(\sum_{j=0}^{r} n_{\updownarrow}(Y_j) + \sum_{j=1}^{r} n_{\updownarrow}(\overline{X}_j) \right) \end{aligned}$$

$$
\begin{aligned}
&+ \sum_{j=0}^{r} |Y_j|_{\mathcal{A}} + \sum_{j=1}^{r} |\overline{X}_j|_{\mathcal{A}} \\
= \;& 3 + 3 \cdot B_{\downarrow}(X) + 3 \cdot n_{\updownarrow}(X) + |X|_{\mathcal{A}}. \qquad (7.18)
\end{aligned}
$$

By Theorem 6.31(1), this is the minimal length for an $\uparrow$-expression denoting X.

As in the base case, we see that there does not exist any $\updownarrow$-expression denoting X. If E' is a $\downarrow$-expression denoting X, then by Theorem 6.31(2), E' cannot be shorter than E:

$$
\begin{aligned}
|E'| &\geq 3 + 3 \cdot B_{\uparrow}(X) + 3 \cdot n_{\updownarrow}(X) + |X|_{\mathcal{A}} \\
&\geq 3 + 3 \cdot B_{\downarrow}(X) + 3 \cdot n_{\updownarrow}(X) + |X|_{\mathcal{A}} = |E|.
\end{aligned}
$$

Hence, E is a minimal DNA expression for X with the prescribed length.

We conclude that this claim is also valid for X.

2. The induction step for this claim is analogous to that for the previous claim.

The inductive proof also makes clear that the recursive construction induced by this result is well defined. If either $B_{\uparrow}(X) = 0$, or $B_{\downarrow}(X) = 0$, then we can directly construct a minimal DNA expression denoting X. If the lower of $B_{\uparrow}(X)$ and $B_{\downarrow}(X)$ is $p+1 \geq 1$, then the construction includes minimal DNA expressions E_j denoting formal DNA submolecules $\overline{X}_j$ for which the lower of $B_{\uparrow}(\overline{X}_j)$ and $B_{\downarrow}(\overline{X}_j)$ is at most p. □

Theorem 7.5 and Theorem 7.24 describe minimal DNA expressions for all nick free formal DNA molecules. We have thus, in particular, proved that all nick free formal DNA molecules are expressible (see Theorem 5.5).

For future reference, we prove a property of the arguments of the minimal DNA expressions we construct according to Theorem 7.24.

Lemma 7.27 *Let X be a nick free formal DNA molecule which contains at least one single-stranded component, and let E be a minimal DNA expression denoting X as described in Theorem 7.24 (equation (7.3) or equation (7.5)).*

Then the arguments of E are $\mathcal{N}$-words and DNA expressions, alternately. In particular, each $\mathcal{N}$-word-argument of E is a maximal $\mathcal{N}$-word occurrence in E.

Note that by Theorem 4.3, we can always make sure that the $\mathcal{N}$-word-arguments of a DNA expression are maximal $\mathcal{N}$-word occurrences, simply by joining consecutive $\mathcal{N}$-word-arguments into one $\mathcal{N}$-word. The $\mathcal{N}$-word-arguments we mean here, however, are the $\mathcal{N}$-words ε_i, as they literally occur in Theorem 7.24.

Proof: Without loss of generality, assume that E has been constructed according to Theorem 7.24(1). Hence, $B_{\uparrow}(X) \geq B_{\downarrow}(X)$ and E is an $\uparrow$-expression.

Let $x'_1 \dots x'_k$ for some $k \geq 1$ be the decomposition of X. Further, let $Y_0\overline{X}_1Y_1 \dots \overline{X}_rY_r$ for some $r \geq 0$ be the lower block partitioning of X that E is based on, where for $j = 0, 1, \dots, r$, $Y_j = x'_{a_j} \dots x'_{b_j}$ for some a_j and b_j. Finally, let

$$
E = \langle \uparrow \varepsilon_{a_0} \dots \varepsilon_{b_0} E_1 \varepsilon_{a_1} \dots \varepsilon_{b_1} \;\dots\; E_r \varepsilon_{a_r} \dots \varepsilon_{b_r} \rangle ,
$$

as specified in Theorem 7.24(1).

We first prove that no $\mathcal{N}$-word-argument of E is succeeded by another $\mathcal{N}$-word-argument. After that, we prove an analogous property for expression-arguments.

- Consider an arbitrary $\mathcal{N}$-word-argument of E.

  For $j = 1, \ldots, r$, E_j is a minimal DNA expression denoting $\overline{X}_j$, and in particular E_j is not an $\mathcal{N}$-word. Hence, the $\mathcal{N}$-word we consider must be equal to $\varepsilon_i = \alpha_i$ with $a_j \leq i \leq b_j$ for some j with $0 \leq j \leq r$. By construction, $x'_i = \mathcal{S}^+(\varepsilon_i) = \binom{\alpha_i}{-}$ is an upper component of X, which is part of the maximal upper sequence $Y_j = x'_{a_j} \ldots x'_{b_j}$.

  If $i \leq b_j - 1$, then by Corollary 3.8(2) x'_{i+1} is a double component $\binom{\alpha_{i+1}}{c(\alpha_{i+1})}$ of X for an $\mathcal{N}$-word α_{i+1}, which is also part of Y_j. Hence, the argument succeeding $\varepsilon_i = \alpha_i$ in E is the $\updownarrow$-expression $\varepsilon_{i+1} = \langle \updownarrow \alpha_{i+1} \rangle$.

  If $i = b_j$ and $j = r$, then $i = b_r$ and ε_i is the last argument of E. If $i = b_j$ and $j \leq r - 1$, then ε_i is succeeded by the (minimal) DNA expression E_{j+1}.

  In none of the cases, ε_i is succeeded in E by an $\mathcal{N}$-word-argument.

- Consider an arbitrary expression-argument ε of E.

  By construction, either ε is E_j for some j with $1 \leq j \leq r$, or it is an $\updownarrow$-expression $\varepsilon_i = \langle \updownarrow \alpha_i \rangle$ for an $\mathcal{N}$-word α_i with $a_j \leq i \leq b_j$ for some j with $0 \leq j \leq r$.

  – In the former case, $\varepsilon = E_j$ denotes the lower block $\overline{X}_j$ of X.

  If Y_j, the substring succeeding $\overline{X}_j$ in the lower block partitioning, is empty, then by Lemma 7.21, we must have $j = r$. This implies that $\varepsilon = E_r$ is the last argument of E.

  If, on the other hand, $Y_j = x'_{a_j} \ldots x'_{b_j}$ is not empty, then it is a maximal upper sequence. By Lemma 7.12(1a), the first component x'_{a_j} of Y_j is an upper component $\binom{\alpha_{a_j}}{-}$ for an $\mathcal{N}$-word α_{a_j}. Hence, the argument succeeding $\varepsilon = E_j$ in E is (the $\mathcal{N}$-word) $\varepsilon_{a_j} = \alpha_{a_j}$.

  – In the latter case, $\varepsilon = \varepsilon_i = \langle \updownarrow \alpha_i \rangle$ denotes the double component $x'_i = \binom{\alpha_i}{c(\alpha_i)}$ of X, which is part of the maximal upper sequence $Y_j = x'_{a_j} \ldots x'_{b_j}$.

  If $i \leq b_j - 1$, then x'_i is succeeded in Y_j by the upper component $\binom{\alpha_{i+1}}{-}$ for an $\mathcal{N}$-word α_{i+1}. This upper component corresponds to an $\mathcal{N}$-word-argument $\varepsilon_{i+1} = \alpha_{i+1}$, which succeeds $\varepsilon_i = \langle \updownarrow \alpha_i \rangle$ in E.

  If, on the other hand, $i = b_j$, then the maximal upper sequence Y_j ends by a double component. By Lemma 7.12(1b), this is only possible if $i = b_j = k$. This implies that $j = r$ and that the argument $\varepsilon = \langle \updownarrow \alpha_i \rangle$ is the last argument of E.

  In none of the cases, the DNA expression ε is succeeded in E by another DNA expression.

□

By Theorem 7.5 and Theorem 7.24, we cannot only *construct* a minimal DNA expression E for a given nick free formal DNA molecule. We can also calculate the length $|E|$ of this minimal DNA expression without having to explicitly perform the construction. The length is simply a function of some elementary structural properties of the formal DNA molecule.

By definition, all minimal DNA expressions denoting a certain formal DNA molecule have the same length. Hence, when we know the length of a minimal DNA expression *we can construct* for a nick free formal DNA molecule, we also know the length of an

arbitrary minimal DNA expression denoting the same molecule. Subsequently, we can use that knowledge to derive other properties of this (arbitrary) minimal DNA expression. In this section, we restrict ourselves to a few properties. In Section 8.1 and Section 8.3, we will present more of them.

We can combine the two values for $|E|$ from Theorem 7.24(1) and (2).

Corollary 7.28 *Let X be a nick free formal DNA molecule which contains at least one single-stranded component, and let E be a minimal DNA expression denoting X. Then*

$$|E| = 3 + 3 \cdot p + 3 \cdot n_{\updownarrow}(X) + |X|_{\mathcal{A}},$$

where p is the minimum of $B_{\downarrow}(X)$ and $B_{\uparrow}(X)$.

Once again, E is not necessarily the result of the construction from Theorem 7.24, but it has the same length as a minimal DNA expression denoting X that *is* the result of that construction.

We will see later that sometimes it is sufficient to have an upper bound on the length of a minimal DNA expression for a nick free formal DNA molecule. We give two such upper bounds here. They follow from the previous result, except for double-complete molecules.

Corollary 7.29 *Let X be a nick free formal DNA molecule and let E be a minimal DNA expression denoting X.*

1. $$|E| \leq 3 + 3 \cdot B_{\downarrow}(X) + 3 \cdot n_{\updownarrow}(X) + |X|_{\mathcal{A}}. \tag{7.19}$$

2. $$|E| \leq 3 + 3 \cdot B_{\uparrow}(X) + 3 \cdot n_{\updownarrow}(X) + |X|_{\mathcal{A}}.$$

Proof:

1. If X does not contain any single-stranded component, hence $X = \binom{\alpha_1}{c(\alpha_1)}$ for an $\mathcal{N}$-word α_1, then by definition $B_{\downarrow}(X) = 0$, $n_{\updownarrow}(X) = 1$ and $|X|_{\mathcal{A}} = |\alpha_1|$. By Theorem 7.5, $E = \langle \updownarrow \alpha_1 \rangle$ is the unique minimal DNA expression denoting X. Then the left hand side of (7.19) evaluates to $3 + |\alpha_1|$ and the right hand side evaluates to $6 + |\alpha_1|$. Indeed, the inequality holds.

   If X contains at least one single-stranded component, then the claim follows from Corollary 7.28, because obviously the minimum p of $B_{\downarrow}(X)$ and $B_{\uparrow}(X)$ satisfies $p \leq B_{\downarrow}(X)$.

2. The proof of this claim is analogous to that of the previous claim.

□

As we mentioned after the statement of Theorem 7.24, if X is a nick free formal DNA molecule containing at least one single-stranded component and $B_{\uparrow}(X) = B_{\downarrow}(X)$, then there both exist a minimal $\uparrow$-expression and a minimal $\downarrow$-expression denoting X.

We now show that if X is nick free and $B_{\uparrow}(X) \neq B_{\downarrow}(X)$, then all minimal DNA expressions are of the same type: they are either $\uparrow$-expressions or $\downarrow$-expressions, depending on which of $B_{\uparrow}(X)$ and $B_{\downarrow}(X)$ is higher.

Lemma 7.30 *Let X be a nick free formal DNA molecule.*

1. *If $B_{\uparrow}(X) > B_{\downarrow}(X)$, then each minimal DNA expression denoting X is an $\uparrow$-expression.*
2. *If $B_{\downarrow}(X) > B_{\uparrow}(X)$, then each minimal DNA expression denoting X is a $\downarrow$-expression.*

Proof:

1. Assume that $B_{\uparrow}(X) > B_{\downarrow}(X)$ and let E be a minimal DNA expression denoting X. Then by the definition of a primitive upper block, X contains at least one upper component. Because the semantics of an $\updownarrow$-expression does not contain single-stranded components, E cannot be an $\updownarrow$-expression.

   By Corollary 7.28,

   $$|E| = 3 + 3 \cdot B_{\downarrow}(X) + 3 \cdot n_{\updownarrow}(X) + |X|_{\mathcal{A}}.$$

   By Theorem 6.31(2), each $\downarrow$-expression E' denoting X satisfies

   $$|E'| \geq 3 + 3 \cdot B_{\uparrow}(X) + 3 \cdot n_{\updownarrow}(X) + |X|_{\mathcal{A}} > |E|,$$

   because $B_{\uparrow}(X) > B_{\downarrow}(X)$. Consequently, a $\downarrow$-expression E' denoting X cannot be minimal and E has to be an $\uparrow$-expression.

2. The proof of this claim is analogous to that of the previous claim.

□

When we combine this result with Lemma 7.17(2b), we immediately obtain

Corollary 7.31 *Let X be a nick free formal DNA molecule.*

1. *Let $Y_0\overline{X}_1Y_1 \ldots \overline{X}_rY_r$ for some $r \geq 0$ be an arbitrary lower block partitioning of X. Then for $j = 1, \ldots, r$, each minimal DNA expression E_j denoting $\overline{X}_j$ is a $\downarrow$-expression.*

   In particular, if X contains at least one single-stranded component and $B_{\uparrow}(X) \geq B_{\downarrow}(X)$, then for $j = 1, \ldots, r$, the minimal DNA expression E_j occurring in Theorem 7.24(1) is a $\downarrow$-expression.

2. *Let $Y_0\overline{X}_1Y_1 \ldots \overline{X}_rY_r$ for some $r \geq 0$ be an arbitrary upper block partitioning of X. Then for $j = 1, \ldots, r$, each minimal DNA expression E_j denoting $\overline{X}_j$ is an $\uparrow$-expression.*

   In particular, if X contains at least one single-stranded component and $B_{\downarrow}(X) \geq B_{\uparrow}(X)$, then for $j = 1, \ldots, r$, the minimal DNA expression E_j occurring in Theorem 7.24(2) is an $\uparrow$-expression.

Note that this result is trivially valid if X is double-complete. In that case, by Lemma 7.19, the only lower (or upper) block partitioning of X is $Y_0 = X$, for which $r = 0$.

We can tell exactly when the argument list of the minimal $\uparrow$-expression we construct starts or ends with a $\downarrow$-expression.

Lemma 7.32 *Let X be a nick free formal DNA molecule which contains at least one single-stranded component, let $B_{\uparrow}(X) \geq B_{\downarrow}(X)$, and let E be a minimal $\uparrow$-expression denoting X as described in Theorem 7.24(1).*

1. *The first single-stranded component of X is a lower component, if and only if the first argument of E is a $\downarrow$-argument.*

2. *The last single-stranded component of X is a lower component, if and only if the last argument of E is a $\downarrow$-argument.*

Of course, there is an analogous result for the minimal $\downarrow$-expressions from Theorem 7.24(2).

Proof:

1. $\Longrightarrow$ Assume that the first single-stranded component of X is a lower component. Then by Lemma 7.11(3a) and (3d), the maximal upper prefix Y_0 of X is empty. By the construction from Theorem 7.24(1) and Corollary 7.31(1), the first argument of E is a $\downarrow$-argument.

   $\Longleftarrow$ Assume that the first argument of E is a $\downarrow$-argument. In the construction from Theorem 7.24(1), the arguments corresponding to the maximal upper prefix Y_0 of X are $\mathcal{N}$-word-arguments and $\updownarrow$-arguments. Because the first argument of E is not such an argument, Y_0 must be empty. By Lemma 7.11(3a) and (3d), the first single-stranded component of X is a lower component.

2. The proof of this claim is analogous to that of the previous claim.

□

We can combine this result with Lemma 6.13(3):

Corollary 7.33 *Let X be a nick free formal DNA molecule which contains at least one single-stranded component, let $B_{\uparrow}(X) \geq B_{\downarrow}(X)$, and let E be a minimal $\uparrow$-expression denoting X as described in Theorem 7.24(1).*

1. *$B_{\uparrow}(X) > B_{\downarrow}(X)$, if and only if neither the first argument, nor the last argument of E is a $\downarrow$-argument.*

2. *$B_{\uparrow}(X) = B_{\downarrow}(X)$, if and only if either the first argument, or the last argument of E is a $\downarrow$-argument (and not both of them).*

3. *It is impossible that both the first argument and the last argument of E are $\downarrow$-arguments.*

Again, there is an analogous result for the minimal $\downarrow$-expressions from Theorem 7.24(2).

By Theorem 7.5, we know that for a double-complete formal DNA molecule, there is exactly one minimal DNA expression. Theorem 7.24, however, only provides us with *a particular construction* of minimal DNA expressions for nick free formal DNA molecules containing single-stranded components. We are still far from a complete description of the language of *all* minimal DNA expressions for *arbitrary*, expressible formal DNA molecules. Nevertheless, we can already draw one conclusion about this language:

Lemma 7.34 *The language of all minimal DNA expressions is not regular.*

Note that by Lemma 4.23, the language $\mathcal{D}$ of all DNA expressions (minimal or not) is not regular, either.

Proof: Let α be an arbitrary $\mathcal{N}$-word and let $l \geq 1$. Then consider the nick free formal DNA molecule

$$X_l = \left(\binom{\alpha}{c(\alpha)} \binom{\alpha}{-} \binom{\alpha}{c(\alpha)} \binom{-}{\alpha} \right)^l \cdot \binom{\alpha}{c(\alpha)} \binom{\alpha}{-} \binom{\alpha}{c(\alpha)} \cdot \left(\binom{-}{\alpha} \binom{\alpha}{c(\alpha)} \binom{\alpha}{-} \binom{\alpha}{c(\alpha)} \right)^l$$

It is not hard to prove by induction on l that $B_{\uparrow}(X_l) = 2l + 1$ and $B_{\downarrow}(X_l) = 2l$, that $\mathcal{P}_l = Y_0 \overline{X}_1 Y_1$ with $Y_0 = \binom{\alpha}{c(\alpha)} \binom{\alpha}{-}$

$$\begin{aligned} \overline{X}_1 = \quad & \binom{\alpha}{c(\alpha)} \binom{-}{\alpha} \cdot \left(\binom{\alpha}{c(\alpha)} \binom{\alpha}{-} \binom{\alpha}{c(\alpha)} \binom{-}{\alpha} \right)^{l-1} \cdot \binom{\alpha}{c(\alpha)} \binom{\alpha}{-} \binom{\alpha}{c(\alpha)} \cdot \\ & \left(\binom{-}{\alpha} \binom{\alpha}{c(\alpha)} \binom{\alpha}{-} \binom{\alpha}{c(\alpha)} \right)^{l-1} \cdot \binom{-}{\alpha} \binom{\alpha}{c(\alpha)} \end{aligned}$$

and $Y_1 = \binom{\alpha}{-} \binom{\alpha}{c(\alpha)}$ is a lower block partitioning of X_l, and that

$$E_l = \left(\langle \uparrow \langle \updownarrow \alpha \rangle \alpha \langle \downarrow \langle \updownarrow \alpha \rangle \alpha \rangle^l \langle \uparrow \langle \updownarrow \alpha \rangle \alpha \langle \updownarrow \alpha \rangle \rangle \left(\alpha \langle \updownarrow \alpha \rangle \rangle \alpha \langle \updownarrow \alpha \rangle \rangle \right)^l \right.$$

is a minimal DNA expression denoting X_l based on $\mathcal{P}_l$ as described in Theorem 7.24. It follows from the pumping lemma for regular languages (Proposition 2.7), that a language requiring brackets to match and containing such DNA expressions is not regular. □

7.2 Minimal DNA expressions for a formal DNA molecule with nick letters

For expressible formal DNA molecules that contain nick letters, it is easy to say what type of DNA expressions ($\uparrow$-expressions, $\downarrow$-expressions or $\updownarrow$-expressions) can be minimal: the outermost operator is determined by the type of nicks.

Lemma 7.35 *Let X be an expressible formal DNA molecule.*

1. *If X contains at least one lower nick letter $^{\triangle}$, then each minimal DNA expression denoting X is an $\uparrow$-expression.*

2. *If X contains at least one upper nick letter $^{\triangledown}$, then each minimal DNA expression denoting X is a $\downarrow$-expression.*

Proof:

1. Assume that X contains at least one lower nick letter. By Lemma 5.2(1), there does not exist any $\downarrow$-expression denoting X, let alone a minimal $\downarrow$-expression. Now it follows from Corollary 7.6 that each minimal DNA expression denoting X is an $\uparrow$-expression.

2. The proof of this claim is analogous to that of the previous claim.

□

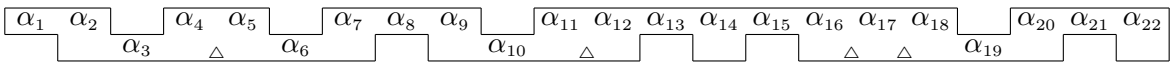


Figure 7.7: A formal DNA molecule containing lower nick letters.

Given an expressible formal DNA molecule with nick letters, it is, of course, not sufficient to know the *type* of the minimal DNA expressions denoting it. We want to construct the minimal DNA expressions themselves. For this, we decompose a formal DNA molecule into nick free pieces and nick letters, as follows:

Definition 7.36 *Let X be a formal DNA molecule. The nick free decomposition of X is the sequence $Z_1, y_1, Z_2, y_2, \ldots, y_{m-1}, Z_m$ for some $m \geq 1$ such that*

- *$X = Z_1 y_1 Z_2 y_2 \ldots y_{m-1} Z_m$, and*
- *for $h = 1, \ldots, m$, Z_h is nick free, and*
- *for $h = 1, \ldots, m-1$, $y_h \in \{^{\triangledown}, _{\triangle}\}$.*

To simplify the notation, we will in general skip the commas and write $Z_1 y_1 Z_2 y_2 \ldots y_{m-1} Z_m$ instead of $Z_1, y_1, Z_2, y_2, \ldots, y_{m-1}, Z_m$ to denote the nick free decomposition of a formal DNA molecule X.

Obviously, because the substrings Z_h in the definition are nick free and the y_h's are precisely the nick letters occurring in X, the nick free decomposition of a formal DNA molecule is unambiguously defined.

The definition extends to all formal DNA molecules X, whether they are expressible or not. If, however, X is expressible, then by Theorem 5.4, all nick letters y_h are equal: either each of them is an upper nick letter, or each of them is a lower nick letter.

Example 7.37 Consider the formal DNA molecule X depicted in Figure 7.7. This molecule contains four lower nick letters and no upper nick letters. The nick free decomposition of X is $Z_{1\triangle} Z_{2\triangle} Z_{3\triangle} Z_{4\triangle} Z_5$, where

$$\begin{aligned}
Z_1 &= \tbinom{\alpha_1}{-}\tbinom{\alpha_2}{c(\alpha_2)}\tbinom{-}{\alpha_3}\tbinom{\alpha_4}{c(\alpha_4)},\\
Z_2 &= \tbinom{\alpha_5}{c(\alpha_5)}\tbinom{-}{\alpha_6}\tbinom{\alpha_7}{c(\alpha_7)}\tbinom{\alpha_8}{-}\tbinom{\alpha_9}{c(\alpha_9)}\tbinom{-}{\alpha_{10}}\tbinom{\alpha_{11}}{c(\alpha_{11})},\\
Z_3 &= \tbinom{\alpha_{12}}{c(\alpha_{12})}\tbinom{\alpha_{13}}{-}\tbinom{\alpha_{14}}{c(\alpha_{14})}\tbinom{\alpha_{15}}{-}\tbinom{\alpha_{16}}{c(\alpha_{16})}, \qquad (7.20)\\
Z_4 &= \tbinom{\alpha_{17}}{c(\alpha_{17})},\\
Z_5 &= \tbinom{\alpha_{18}}{c(\alpha_{18})}\tbinom{-}{\alpha_{19}}\tbinom{\alpha_{20}}{c(\alpha_{20})}\tbinom{\alpha_{21}}{-}\tbinom{\alpha_{22}}{c(\alpha_{22})}.
\end{aligned}$$

■

We give two properties of the substrings Z_h in the nick free decomposition of a formal DNA molecule X. Although the second one is formulated in a formal way, its intuitive meaning is simple: each Z_h is a subsequence of the components of X. Indeed, in Example 7.37, we wrote the Z_h's as sequences of upper components, lower components and double components.

Lemma 7.38 *Let X be a formal DNA molecule, let $x'_1 \dots x'_k$ for some $k \geq 1$ be the decomposition of X, and let $Z_1 y_1 Z_2 y_2 \dots y_{m-1} Z_m$ for some $m \geq 1$ be the nick free decomposition of X.*

Then for $h = 1, \dots, m$,

1. *Z_h is a formal DNA submolecule of X and in particular $Z_h \neq \lambda$.*
2. *there exist a_h and b_h with $1 \leq a_h \leq b_h \leq k$ such that $Z_1 y_1 \dots Z_{h-1} y_{h-1} = x'_1 \dots x'_{a_h - 1}$ and $y_h Z_{h+1} \dots y_{m-1} Z_m = x'_{b_h+1} \dots x'_k$ (hence, $Z_h = x'_{a_h} \dots x'_{b_h}$).*

Proof: Consider the substring Z_h with $1 \leq h \leq m$.

1. By the definition of a formal DNA molecule, $X \neq \lambda$, $L(X), R(X) \in \mathcal{A}$ and nick letters do not occur in consecutive positions in X. This implies that $Z_h \neq \lambda$. Because Z_h is nick free, in particular $L(Z_h), R(Z_h) \in \mathcal{A}$.

   Now the claim follows from Lemma 3.4.

2. If $h = 1$, then $Z_1 y_1 \dots Z_{h-1} y_{h-1} = \lambda$ and the value $a_h = 1$ suffices for the first part of the claim.

   If $h \geq 2$, then $R(Z_1 y_1 \dots Z_{h-1} y_{h-1}) = y_{h-1} \in \{{}^{\triangledown}, {}^{\triangle}\}$. Because each nick letter occurring in X is by definition a component of X, there exists $a_h \geq 2$ such that $y_{h-1} = x'_{a_h - 1}$ [3] and $Z_1 y_1 \dots Z_{h-1} y_{h-1} = x'_1 \dots x'_{a_h - 1}$.

   In an analogous way, we find a value $b_h \leq k$ such that $y_h Z_{h+1} \dots y_{m-1} Z_m = x'_{b_h+1} \dots x'_k$.

   By Claim 1, $Z_h = x'_{a_h} \dots x'_{b_h}$ is non-empty. Hence, $a_h \leq b_h$.

□

From now on, we will only consider nick free decompositions of *expressible* formal DNA molecules. We will demonstrate that minimal DNA expressions denoting an expressible molecule X containing nick letters (e.g., minimal $\uparrow$-expressions denoting a molecule with lower nick letters) can be constructed from some special DNA expressions denoting the nick free pieces of X. These special DNA expressions are operator-minimal DNA expressions:

Definition 7.39 *A DNA expression E is operator-minimal if for every DNA expression E' with the same outermost operator as E and with $E' \equiv E$, $|E'| \geq |E|$.*

For example, an $\uparrow$-expression E denoting a formal DNA molecule X is operator-minimal, if there does not exist a shorter $\uparrow$-expression denoting X. Obviously, each minimal DNA expression is also operator-minimal.

Example 7.40 We continue with the formal DNA molecule X from Example 7.37, which is depicted in Figure 7.7. The second formal DNA submolecule occurring in the nick free decomposition of X is

$$Z_2 = \binom{\alpha_5}{c(\alpha_5)} \binom{-}{\alpha_6} \binom{\alpha_7}{c(\alpha_7)} \binom{\alpha_8}{-} \binom{\alpha_9}{c(\alpha_9)} \binom{-}{\alpha_{10}} \binom{\alpha_{11}}{c(\alpha_{11})} \tag{7.21}$$

(see (7.20)). We have $B_{\uparrow}(Z_2) = 1$ and $B_{\downarrow}(Z_2) = 2$.

[3] In fact, $a_h \geq 3$, because X cannot start with the nick letter $y_{h-1} = x'_{a_h - 1}$.

By Lemma 7.30(2), each minimal DNA expression E_2 denoting Z_2 is a $\downarrow$-expression. When we apply Theorem 7.24 to Z_2, we obtain

$$E_2 = \langle\downarrow \langle\updownarrow \alpha_5\rangle \alpha_6 \langle\uparrow \langle\updownarrow \alpha_7\rangle \alpha_8 \langle\updownarrow \alpha_9\rangle\rangle \alpha_{10} \langle\updownarrow \alpha_{11}\rangle\rangle\,,$$

for which (indeed)

$$|E_2| = 18 + |Z_2|_{\mathcal{A}} = 3 + 3 \cdot 1 + 3 \cdot 4 + |Z_2|_{\mathcal{A}} = 3 + 3 \cdot B_{\uparrow}(Z_2) + 3 \cdot n_{\updownarrow}(Z_2) + |Z_2|_{\mathcal{A}}.$$

Now let E_2' be an $\uparrow$-expression denoting Z_2. By Theorem 6.31(1),

$$|E_2'| \geq 3 + 3 \cdot B_{\downarrow}(Z_2) + 3 \cdot n_{\updownarrow}(Z_2) + |Z_2|_{\mathcal{A}} = 3 + 3 \cdot 2 + 3 \cdot 4 + |Z_2|_{\mathcal{A}} = 21 + |Z_2|_{\mathcal{A}}.$$

In other words, by Lemma 6.1, the $\uparrow$-expression E_2' contains at least 7 operators, whereas the $\downarrow$-expression E_2 contains 6 operators. Indeed, an $\uparrow$-expression denoting Z_2 will never be minimal. If, however, $|E_2'| = 21 + |Z_2|_{\mathcal{A}}$, then E_2' *is* operator-minimal. It is not difficult to construct an operator-minimal $\uparrow$-expression denoting Z_2. We can simply take

$$E_2' = \langle\uparrow E_2\rangle = \langle\uparrow \langle\downarrow \langle\updownarrow \alpha_5\rangle \alpha_6 \langle\uparrow \langle\updownarrow \alpha_7\rangle \alpha_8 \langle\updownarrow \alpha_9\rangle\rangle \alpha_{10} \langle\updownarrow \alpha_{11}\rangle\rangle\rangle\,, \tag{7.22}$$

because $\mathcal{S}(E_2') = \nu^+(\mathcal{S}(E_2)) = \mathcal{S}(E_2) = Z_2$. Another operator-minimal $\uparrow$-expression denoting Z_2, which is less directly related to E_2, is

$$E_2'' = \langle\uparrow \langle\downarrow \langle\updownarrow \alpha_5\rangle \alpha_6 \langle\updownarrow \alpha_7\rangle\rangle \alpha_8 \langle\downarrow \langle\updownarrow \alpha_9\rangle \alpha_{10} \langle\updownarrow \alpha_{11}\rangle\rangle\rangle\,. \tag{7.23}$$

■

Lemma 7.4 directly relates the minimality of a DNA expression E to the minimality of the DNA subexpressions of E. For operator-minimal DNA expressions, we have a weaker result. Its proof is similar to the second part of the proof of Lemma 7.4:

Lemma 7.41 *If a DNA expression E is operator-minimal, then each proper DNA subexpression of E is minimal.*

This result cannot be reversed. It is not sufficient for a DNA expression to be operator-minimal that all its proper DNA subexpressions are minimal. For example, the DNA expression $E = \langle\uparrow \langle\uparrow \alpha_1\rangle\rangle$ has only one proper DNA subexpression: $E^s = \langle\uparrow \alpha_1\rangle$. It follows from Theorem 7.24(1) that E^s is minimal, whereas E is clearly not operator-minimal.

The construction from Theorem 7.24(1) can be reused to construct an operator-minimal $\uparrow$-expression for a nick free formal DNA molecule. Of course, an operator-minimal $\downarrow$-expression can be constructed in a completely analogous way (cf. Theorem 7.24(2)).

Theorem 7.42 *Let X be a nick free formal DNA molecule and let $x_1' \ldots x_k'$ for some $k \geq 1$ be the decomposition of X.*

- *Let $\mathcal{P} = Y_0\overline{X}_1Y_1 \ldots \overline{X}_rY_r$ for some $r \geq 0$ be an arbitrary lower block partitioning of X;*
- *for $j = 1, \ldots, r$, let E_j be an arbitrary minimal DNA expression denoting $\overline{X}_j$;*
- *for $j = 0, 1, \ldots, r$, let $Y_j = x_{a_j}' \ldots x_{b_j}'$ for some $a_j \geq 1$ and $b_j \leq k$;*

- *for* $j = 0, 1, \ldots, r$ *and for* $i = a_j, \ldots, b_j$, *let*

$$\varepsilon_i = \begin{cases} \alpha_i & \text{if } x'_i = \binom{\alpha_i}{-} \text{ for an } \mathcal{N}\text{-word } \alpha_i \\ \langle \updownarrow \alpha_i \rangle & \text{if } x'_i = \binom{\alpha_i}{c(\alpha_i)} \text{ for an } \mathcal{N}\text{-word } \alpha_i; \end{cases} \quad \text{and}$$

- *let*

$$E = \langle \uparrow \varepsilon_{a_0} \ldots \varepsilon_{b_0} E_1 \varepsilon_{a_1} \ldots \varepsilon_{b_1} \ \ldots \ E_r \varepsilon_{a_r} \ldots \varepsilon_{b_r} \rangle \,. \tag{7.24}$$

Then

(a) all ingredients needed to construct E *(i.e., the lower block partitioning* $\mathcal{P}$*, the minimal DNA expressions* E_j*, the indices* a_j *and* b_j*, and the arguments* ε_i*) are well defined, and*

(b) E *is an operator-minimal* $\uparrow$*-expression denoting* X*, and*

$$|E| = 3 + 3 \cdot B_{\downarrow}(X) + 3 \cdot n_{\updownarrow}(X) + |X|_{\mathcal{A}}. \tag{7.25}$$

By Theorem 7.24, we know how to construct minimal DNA expressions E_j denoting the formal DNA submolecules $\overline{X}_j$. Hence, the specification above is complete.

Note that there are two subtle differences between Theorem 7.42 and Theorem 7.24(1).

First, we do not demand the formal DNA molecule X to contain at least one single-stranded component. There also exists an operator-minimal $\uparrow$-expression denoting $X = \binom{\alpha_1}{c(\alpha_1)}$ for an $\mathcal{N}$-word α_1. For this formal DNA molecule, there exists exactly one lower block partitioning, namely $\mathcal{P} = X$. The corresponding operator-minimal $\uparrow$-expression is $E = \langle \uparrow \langle \updownarrow \alpha_1 \rangle \rangle$.

Second, we do not restrict ourselves to formal DNA molecules X with $B_{\uparrow}(X) \geq B_{\downarrow}(X)$. There exist operator-minimal $\uparrow$-expressions for every nick free formal DNA molecule. Indeed, we have given operator-minimal $\uparrow$-expressions for the formal DNA molecule Z_2 from (7.21), for which $B_{\uparrow}(Z_2) < B_{\downarrow}(Z_2)$.

Note finally that the operator-minimal $\uparrow$-expression described in Theorem 7.42 achieves the lower bound from Theorem 6.31(1). Analogously, operator-minimal $\downarrow$-expressions denoting a nick free formal DNA molecule achieve the lower bound from Theorem 6.31(2).

Example 7.43 Indeed, the two operator-minimal $\uparrow$-expressions E'_2 and E''_2 we have given in Example 7.40, which denote the formal DNA molecule Z_2 from (7.21), can be constructed according to the description in Theorem 7.42. Both the maximal upper prefix and the maximal upper suffix of Z_2 are empty, but Z_2 does have one internal maximal upper sequence, viz $\binom{\alpha_8}{-}$. Hence, by Lemma 7.22, there are two lower block partitionings of Z_2. The first one is $\mathcal{P}' = Y'_0 \overline{X}'_1 Y'_1 = Y'_0 Z_2 Y'_1$, where the (empty) maximal upper prefix and maximal upper suffix of Z_2 are denoted by Y'_0 and Y'_1, respectively. The second one is the primitive lower block partitioning $\mathcal{P}'' = Y''_0 \overline{X}''_1 Y''_1 \overline{X}''_2 Y''_2$, where the maximal upper prefix and maximal upper suffix are denoted by Y''_0 and Y''_2, respectively, and

$$\begin{aligned} \overline{X}''_1 &= \tbinom{\alpha_5}{c(\alpha_5)} \tbinom{-}{\alpha_6} \tbinom{\alpha_7}{c(\alpha_7)}, \\ Y''_1 &= \tbinom{\alpha_8}{-}, \\ \overline{X}''_2 &= \tbinom{\alpha_9}{c(\alpha_9)} \tbinom{-}{\alpha_{10}} \tbinom{\alpha_{11}}{c(\alpha_{11})}. \end{aligned}$$

The DNA expression E_2' from (7.22) corresponds to $\mathcal{P}'$ and the DNA expression E_2'' from (7.23) corresponds to $\mathcal{P}''$. ■

Example 7.44 Consider the nick free formal DNA molecule X from Figure 7.5, for which $B_\uparrow(X) = 4$, $B_\downarrow(X) = 3$ and $n_\updownarrow(X) = 9$. By Lemma 7.30(1), each minimal DNA expression for X is an $\uparrow$-expression. We use the 'lower analogue' of Theorem 7.42 to construct an operator-minimal $\downarrow$-expression denoting X. Let $\mathcal{P}$ be the upper block partitioning of X from Figure 7.3(b): $\mathcal{P} = Y_0\overline{X}_1Y_1\overline{X}_2Y_2\overline{X}_3Y_3$, where both the maximal lower prefix Y_0 and the maximal lower suffix Y_3 are empty,

$$\begin{aligned}
\overline{X}_1 &= \tbinom{\alpha_1}{-}\tbinom{\alpha_2}{c(\alpha_2)},\\
Y_1 &= \tbinom{-}{\alpha_3}\tbinom{\alpha_4}{c(\alpha_4)}\tbinom{-}{\alpha_5},\\
\overline{X}_2 &= \tbinom{\alpha_6}{c(\alpha_6)}\tbinom{\alpha_7}{-}\tbinom{\alpha_8}{c(\alpha_8)},\\
Y_2 &= \tbinom{-}{\alpha_9},\\
\overline{X}_3 &= \tbinom{\alpha_{10}}{c(\alpha_{10})}\tbinom{\alpha_{11}}{-}\tbinom{\alpha_{12}}{c(\alpha_{12})}\tbinom{\alpha_{13}}{-}\tbinom{\alpha_{14}}{c(\alpha_{14})}\tbinom{-}{\alpha_{15}}\tbinom{\alpha_{16}}{c(\alpha_{16})}\tbinom{\alpha_{17}}{-}\tbinom{\alpha_{18}}{c(\alpha_{18})}.
\end{aligned}$$

For the upper block $\overline{X}_1$, we have $B_\uparrow(\overline{X}_1) = 1$ and $B_\downarrow(\overline{X}_1) = 0$. Hence, by Lemma 7.19, the only lower block partitioning of $\overline{X}_1$ is $\mathcal{P}_1 = \overline{X}_1$. Now, by Theorem 7.24(1),

$$E_1 = \langle\uparrow \alpha_1 \langle\updownarrow \alpha_2\rangle\rangle$$

is a minimal DNA expression denoting $\overline{X}_1$. Analogously, we find a minimal DNA expression denoting $\overline{X}_2$:

$$E_2 = \langle\uparrow \langle\updownarrow \alpha_6\rangle \alpha_7 \langle\updownarrow \alpha_8\rangle\rangle\,.$$

Finally, $B_\uparrow(\overline{X}_3) = 2$ and $B_\downarrow(\overline{X}_3) = 1$. When we recursively apply Theorem 7.24(1) to the upper block $\overline{X}_3$ and Theorem 7.24(2) to its lower block $\binom{\alpha_{14}}{c(\alpha_{14})}\binom{-}{\alpha_{15}}\binom{\alpha_{16}}{c(\alpha_{16})}$, we also obtain a minimal DNA expression denoting $\overline{X}_3$:

$$E_3 = \langle\uparrow \langle\updownarrow \alpha_{10}\rangle \alpha_{11} \langle\updownarrow \alpha_{12}\rangle \alpha_{13} \langle\downarrow \langle\updownarrow \alpha_{14}\rangle \alpha_{15} \langle\updownarrow \alpha_{16}\rangle\rangle \alpha_{17} \langle\updownarrow \alpha_{18}\rangle\rangle\,.$$

The resulting operator-minimal $\downarrow$-expression denoting the entire formal DNA molecule X is

$$\begin{aligned}
E' &= \langle\downarrow E_1\alpha_3 \langle\updownarrow \alpha_4\rangle \alpha_5 E_2\alpha_9 E_3\rangle\\
&= \langle\downarrow\langle\uparrow \alpha_1 \langle\updownarrow \alpha_2\rangle\rangle \alpha_3 \langle\updownarrow \alpha_4\rangle \alpha_5 \langle\uparrow \langle\updownarrow \alpha_6\rangle \alpha_7 \langle\updownarrow \alpha_8\rangle\rangle \alpha_9 \qquad (7.26)\\
&\quad \langle\uparrow \langle\updownarrow \alpha_{10}\rangle \alpha_{11} \langle\updownarrow \alpha_{12}\rangle \alpha_{13} \langle\downarrow \langle\updownarrow \alpha_{14}\rangle \alpha_{15} \langle\updownarrow \alpha_{16}\rangle\rangle \alpha_{17} \langle\updownarrow \alpha_{18}\rangle\rangle\ \rangle\,.
\end{aligned}$$

We have

$$|E'| = 42 + |X|_{\mathcal{A}} = 3 + 3\cdot 4 + 3\cdot 9 + |X|_{\mathcal{A}} = 3 + 3\cdot B_\uparrow(X) + 3\cdot n_\updownarrow(X) + |X|_{\mathcal{A}},$$

which fits in with the analogue for $\downarrow$-expressions of (7.25). ■

Proof of Theorem 7.42: First, we can prove that the lower block partitioning $\mathcal{P}$, the indices a_j and b_j and the arguments ε_i of E are well defined. For this, we refer to the corresponding part of the proof of Theorem 7.24, because that carries over entirely.

Each lower block $\overline{X}_j$ occurring in $\mathcal{P}$ is a nick free formal DNA molecule which, by Lemma 7.17(1), contains at least one single-stranded component. By Theorem 7.24, we

can construct a (minimal) DNA expression denoting $\overline{X}_j$. Hence, the minimal DNA expressions $E_1, \ldots, E_r$ needed to construct E are well defined. Note that in the proof of Theorem 7.24, we needed induction to prove that the E_j's occurring there were well defined. Here, we can simply benefit from the result of that.

For the proof that E is a DNA expression denoting X, we again refer to the corresponding part of the proof of Theorem 7.24.

We finally prove that E has the specified length and (thus) is operator-minimal. Also in this case, we do not need the inductive set-up we used in the proof of Theorem 7.24. However, the main ingredients from the induction step in that proof can be reused.

For an arbitrary j with $1 \leq j \leq r$, we consider the lower block $\overline{X}_j$. By Lemma 7.17(2b), $B_\uparrow(\overline{X}_j) = B_\downarrow(\overline{X}_j) - 1$. When we apply Theorem 7.24(2), we find that a minimal DNA expression E_j denoting $\overline{X}_j$ has length

$$|E_j| = 3 + 3 \cdot B_\uparrow(\overline{X}_j) + 3 \cdot n_\updownarrow(\overline{X}_j) + |\overline{X}_j|_{\mathcal{A}} = 3 \cdot B_\downarrow(\overline{X}_j) + 3 \cdot n_\updownarrow(\overline{X}_j) + |\overline{X}_j|_{\mathcal{A}}.$$

We can then calculate the length of E, in the same way as we did in the proof of Theorem 7.24 (in particular, see the derivation of (7.18)):

$$|E| = \ldots = 3 + 3 \cdot B_\downarrow(X) + 3 \cdot n_\updownarrow(X) + |X|_{\mathcal{A}}.$$

Because this equals the lower bound on the length of an $\uparrow$-expression denoting X from Theorem 6.31(1), E is operator-minimal. □

By a proof that is nearly[4] identical to that of Lemma 7.27, we find

Lemma 7.45 *Let X be a nick free formal DNA molecule and let E be an operator-minimal $\uparrow$-expression denoting X as described in Theorem 7.42 (equation (7.24)).*

Then the arguments of E are $\mathcal{N}$-words and DNA expressions, alternately. In particular, each $\mathcal{N}$-word-argument of E is a maximal $\mathcal{N}$-word occurrence in E.

We use the operator-minimal DNA expressions from Theorem 7.42 to obtain minimal DNA expressions for expressible formal DNA molecules containing nick letters.

Theorem 7.46 *Let X be a formal DNA molecule which contains at least one lower nick letter $_\triangle$, and does not contain any upper nick letter $^\triangledown$.*

- *Let $Z_1{}_\triangle Z_2{}_\triangle \ldots {}_\triangle Z_m$ for some $m \geq 2$ be the nick free decomposition of X;*
- *for $h = 1, \ldots, m$, let E_h be an operator-minimal $\uparrow$-expression denoting Z_h, and let the string $\widehat{E}_h$ be the sequence of the arguments of E_h (hence, if $E_h = \langle\uparrow \varepsilon_{h,1} \ldots \varepsilon_{h,n_h}\rangle$ for some $n_h \geq 1$ and $\mathcal{N}$-words and DNA expressions $\varepsilon_{h,1}, \ldots, \varepsilon_{h,n_h}$, then $\widehat{E}_h = \varepsilon_{h,1} \ldots \varepsilon_{h,n_h}$); and*
- *let $E = \left\langle\uparrow \widehat{E}_1 \ldots \widehat{E}_m\right\rangle$.*

Then

(a) all ingredients needed to construct E (i.e., the nick free decomposition and the operator-minimal $\uparrow$-expressions E_j) are well defined, and

[4]Only the observation that $B_\uparrow(X) \geq B_\downarrow(X)$ in the proof of Lemma 7.27 is not valid here. This observation is, however, not important for the correctness of either of the proofs.

(b) E is a minimal DNA expression denoting X, and

$$|E| = 3 + 3 \cdot B_{\downarrow}(X) + 3 \cdot n_{\updownarrow}(X) + |X|_{\mathcal{A}}. \tag{7.27}$$

Also in this case, the (minimal) $\uparrow$-expression described achieves the lower bound for $\uparrow$-expressions from Theorem 6.31(1).

Of course, by an analogous result, we can construct a minimal DNA expression for an arbitrary formal DNA molecule that contains at least one upper nick letter and does not contain any lower nick letter. The two analogous results imply in particular that all formal DNA molecules with nicks in (exactly) one strand are indeed expressible. This completes the proof of Theorem 5.5.

Example 7.47 In Example 7.37, we have established that the nick free decomposition for the formal DNA molecule from Figure 7.7 is $Z_1 {}_{\triangle}Z_2 {}_{\triangle}Z_3 {}_{\triangle}Z_4 {}_{\triangle}Z_5$, where $Z_1, \ldots, Z_5$ are given in (7.20). Because none of Z_1, Z_3, Z_4, Z_5 has an internal maximal upper sequence, there exists exactly one lower block partitioning for each of them. Hence, for each of them, Theorem 7.42 specifies one operator-minimal $\uparrow$-expression:

$$\begin{aligned}
E_1 &= \langle\uparrow \alpha_1 \langle\downarrow \langle\updownarrow \alpha_2\rangle \alpha_3 \langle\updownarrow \alpha_4\rangle\rangle\rangle\,, \\
E_3 &= \langle\uparrow \langle\updownarrow \alpha_{12}\rangle \alpha_{13} \langle\updownarrow \alpha_{14}\rangle \alpha_{15} \langle\updownarrow \alpha_{16}\rangle\rangle\,, \\
E_4 &= \langle\uparrow \langle\updownarrow \alpha_{17}\rangle\rangle\,, \\
E_5 &= \langle\uparrow \langle\downarrow \langle\updownarrow \alpha_{18}\rangle \alpha_{19} \langle\updownarrow \alpha_{20}\rangle\rangle \alpha_{21} \langle\updownarrow \alpha_{22}\rangle\rangle\,.
\end{aligned}$$

The formal DNA submolecule Z_2 has one internal maximal upper sequence, giving rise to two different lower block partitionings. As we observed in Example 7.43, the DNA expressions E_2' and E_2'' from (7.22) and (7.23) are the operator-minimal $\uparrow$-expressions corresponding to these lower block partitionings.

To construct a minimal DNA expression denoting the entire formal DNA molecule X, we may arbitrarily choose either of E_2' and E_2''. When we choose E_2'', we obtain

$$\begin{aligned}
E = \langle\uparrow \quad & \alpha_1 \langle\downarrow \langle\updownarrow \alpha_2\rangle \alpha_3 \langle\updownarrow \alpha_4\rangle\rangle \quad \langle\downarrow \langle\updownarrow \alpha_5\rangle \alpha_6 \langle\updownarrow \alpha_7\rangle\rangle \alpha_8 \langle\downarrow \langle\updownarrow \alpha_9\rangle \alpha_{10} \langle\updownarrow \alpha_{11}\rangle\rangle \\
& \langle\updownarrow \alpha_{12}\rangle \alpha_{13} \langle\updownarrow \alpha_{14}\rangle \alpha_{15} \langle\updownarrow \alpha_{16}\rangle \quad \langle\updownarrow \alpha_{17}\rangle \\
& \langle\downarrow \langle\updownarrow \alpha_{18}\rangle \alpha_{19} \langle\updownarrow \alpha_{20}\rangle\rangle \alpha_{21} \langle\updownarrow \alpha_{22}\rangle \quad \rangle\,.
\end{aligned} \tag{7.28}$$

Indeed,

$$|E| = 54 + |X|_{\mathcal{A}} = 3 + 3 \cdot 4 + 3 \cdot 13 + |X|_{\mathcal{A}} = 3 + 3 \cdot B_{\downarrow}(X) + 3 \cdot n_{\updownarrow}(X) + |X|_{\mathcal{A}}.$$

■

Note that in the above example, both $B_{\uparrow}(Z_1) = B_{\downarrow}(Z_1) = 1$ and $B_{\uparrow}(Z_5) = B_{\downarrow}(Z_5) = 1$. Hence, by Theorem 7.24, E_1 and E_5 are not only operator-minimal, but also ('absolutely') minimal, and there also exist minimal $\downarrow$-expressions denoting Z_1 and Z_5. In the current context, however, we must choose the (operator-)minimal $\uparrow$-expressions.

Proof of Theorem 7.46: By assumption, X contains at least one lower nick letter ${}_{\triangle}$ and does not contain any upper nick letter ${}^{\triangledown}$. Hence, each nick letter y_h in the definition of the nick free decomposition is a lower nick letter. Indeed, the nick free decomposition of X is $Z_1 {}_{\triangle}Z_2 {}_{\triangle} \ldots {}_{\triangle}Z_m$ with $m \geq 2$.

By definition, each Z_h is nick free. Hence, by Theorem 7.42, there indeed exists an operator-minimal $\uparrow$-expression E_h denoting Z_h.

By the definition of a formal DNA molecule, a nick letter may occur only between two elements of $\mathcal{A}_\pm$. Hence, for $h = 1, \ldots, m-1$, $R(Z_h), L(Z_{h+1}) \in \mathcal{A}_\pm$. Consequently, the DNA expressions $E_1, \ldots, E_m$ fit together by upper strands (so that $\langle\uparrow E_1 \ldots E_m\rangle$ is a DNA expression), and

$$\mathcal{S}(\langle\uparrow E_1 E_2 \ldots E_m\rangle) = \nu^+(Z_1)_{\triangle}\nu^+(Z_2)_{\triangle}\ldots_{\triangle}\nu^+(Z_m).$$

Because $Z_1, \ldots, Z_m$ are nick free, this is equal to X. By Lemma 5.10, $E = \left\langle\uparrow \widehat{E}_1 \ldots \widehat{E}_m\right\rangle$ is also a DNA expression denoting X.

By Theorem 7.42, for $h = 1, \ldots, m$,

$$|E_h| = 3 + 3 \cdot B_\downarrow(Z_h) + 3 \cdot n_\updownarrow(Z_h) + |Z_h|_{\mathcal{A}},$$

and thus

$$|\widehat{E}_h| = 3 \cdot B_\downarrow(Z_h) + 3 \cdot n_\updownarrow(Z_h) + |Z_h|_{\mathcal{A}}. \tag{7.29}$$

Since each Z_h is nick free, we certainly have $\#_\triangledown(Z_h) = 0$. Now, when we apply equations (6.4), (6.5) and (6.6) from Lemma 6.27(1) to $\langle\uparrow E_1 E_2 \ldots E_m\rangle$, we find

$$B_\downarrow(Z_1) + \cdots + B_\downarrow(Z_m) = B_\downarrow(X) \text{ and} \tag{7.30}$$
$$n_\updownarrow(Z_1) + \cdots + n_\updownarrow(Z_m) = n_\updownarrow(X). \tag{7.31}$$

We use (7.29), (7.30) and (7.31) to calculate $|E|$:

$$\begin{aligned}
|E| &= 3 + \sum_{h=1}^{m} |\widehat{E}_h| \\
&= 3 + \sum_{h=1}^{m} (3 \cdot B_\downarrow(Z_h) + 3 \cdot n_\updownarrow(Z_h) + |Z_h|_{\mathcal{A}}) \\
&= 3 + 3 \cdot \sum_{h=1}^{m} B_\downarrow(Z_h) + 3 \cdot \sum_{h=1}^{m} n_\updownarrow(Z_h) + \sum_{h=1}^{m} |Z_h|_{\mathcal{A}} \\
&= 3 + 3 \cdot B_\downarrow(X) + 3 \cdot n_\updownarrow(X) + |X|_{\mathcal{A}}.
\end{aligned}$$

By Theorem 6.31(1), this is the minimal length of an $\uparrow$-expression denoting X. Then by Lemma 7.35(1), E is a minimal DNA expression for X. □

One may wonder if we really need the concept of operator-minimality in the construction of minimal DNA expressions for formal DNA molecules containing nick letters.

Let X be a formal DNA molecule, with nick free decomposition $Z_{1\triangle}Z_{2\triangle}\ldots_{\triangle}Z_m$ for some $m \geq 2$. Then a minimal DNA expression denoting X might also be constructed by (1) determining minimal DNA expressions $E_1, \ldots, E_m$ denoting the nick free formal DNA submolecules $Z_1, \ldots, Z_m$, respectively, (2) defining the $\uparrow$-expression $\langle\uparrow E_1 \ldots E_m\rangle$ with these minimal DNA expressions as arguments, and (3) removing redundant operators $\uparrow$ according to Lemma 5.10, i.e., replacing those DNA expressions E_h that are themselves $\uparrow$-expressions by their respective arguments.

In order to make step (3) as effective as possible, we should in step (1) choose for $\uparrow$-expressions E_h whenever we can. In particular, if, for some h with $1 \leq h \leq m$, Z_h contains at least one single-stranded component and $B_\uparrow(Z_h) = B_\downarrow(Z_h)$, then E_h should be an $\uparrow$-expression as specified by Theorem 7.24(1) (and not a $\downarrow$-expression as specified by Theorem 7.24(2)).

Example 7.48 Let us apply the alternative method just described to the formal DNA molecule X from Figure 7.7. The nick free decomposition of X is $Z_1 {}_{\triangle}Z_2 {}_{\triangle}Z_3 {}_{\triangle}Z_4 {}_{\triangle}Z_5$, where $Z_1, \ldots, Z_5$ are given in (7.20).

As we observed in Example 7.40, each minimal DNA expression denoting

$$Z_2 = \binom{\alpha_5}{c(\alpha_5)}\binom{-}{\alpha_6}\binom{\alpha_7}{c(\alpha_7)}\binom{\alpha_8}{-}\binom{\alpha_9}{c(\alpha_9)}\binom{-}{\alpha_{10}}\binom{\alpha_{11}}{c(\alpha_{11})}$$

is a $\downarrow$-expression. According to the alternative method, such a $\downarrow$-expression appears unchanged in the resulting minimal DNA expression for X. However, in the minimal DNA expression E denoting X from (7.28), the arguments corresponding to Z_2 are $\langle\downarrow \langle\updownarrow \alpha_5\rangle \alpha_6 \langle\updownarrow \alpha_7\rangle\rangle$, α_8 and $\langle\downarrow \langle\updownarrow \alpha_9\rangle \alpha_{10} \langle\updownarrow \alpha_{11}\rangle\rangle$, which is not just one $\downarrow$-expression.

Hence, E cannot be obtained by the alternative method, whereas it can be obtained by the construction from Theorem 7.46, which is based on operator-minimal $\uparrow$-expressions. ■

We can conclude from the above example, that there exist minimal DNA expressions for molecules with lower nick letters that cannot be obtained by the alternative method. In Theorem 8.15, we will see that such DNA expressions do not exist for the construction from Theorem 7.46: each minimal DNA expression denoting a formal DNA molecule with lower nick letters fits into the description from Theorem 7.46.

For nick free formal DNA molecules X, Corollary 7.29 gave two general upper bounds on the length of minimal DNA expressions denoting X. The upper bounds can be used, even when it is unknown if the DNA expression involved is an $\uparrow$-expression, a $\downarrow$-expression, or an $\updownarrow$-expression.

These upper bounds are not generally valid in the case with nicks. For example, it is not true in general that a minimal DNA expression E denoting a formal DNA molecule X with at least one nick letter satisfies

$$|E| \leq 3 + 3 \cdot B_{\uparrow}(X) + 3 \cdot n_{\updownarrow}(X) + |X|_{\mathcal{A}}. \tag{7.32}$$

Example 7.49 Consider the formal DNA molecule

$$X = \binom{\alpha_1}{c(\alpha_1)}\binom{-}{\alpha_2}\binom{\alpha_3}{c(\alpha_3)}{}_{\triangle}\binom{\alpha_4}{c(\alpha_4)}\binom{-}{\alpha_5}\binom{\alpha_6}{c(\alpha_6)},$$

where $\alpha_1, \ldots, \alpha_6$ are arbitrary $\mathcal{N}$-words. For this molecule, $B_{\uparrow}(X) = 1$, $B_{\downarrow}(X) = 2$ and $n_{\updownarrow}(X) = 4$. By Theorem 7.46,

$$E = \langle\uparrow \langle\downarrow \langle\updownarrow \alpha_1\rangle \alpha_2 \langle\updownarrow \alpha_3\rangle\rangle \langle\downarrow \langle\updownarrow \alpha_4\rangle \alpha_5 \langle\updownarrow \alpha_6\rangle\rangle\rangle$$

is a minimal DNA expression denoting X, and

$$\begin{aligned} |E| &= 3 + 3 \cdot B_{\downarrow}(X) + 3 \cdot n_{\updownarrow}(X) + |X|_{\mathcal{A}} = 21 + |X|_{\mathcal{A}} \\ &> 3 + 3 \cdot B_{\uparrow}(X) + 3 \cdot n_{\updownarrow}(X) + |X|_{\mathcal{A}} = 18 + |X|_{\mathcal{A}}. \end{aligned}$$

The violation of (7.32) is due to the lower nick letter occurring in X. The fact that $B_{\downarrow}(X) > B_{\uparrow}(X)$ may suggest that a minimal DNA expression for X should be a $\downarrow$-expression. However, by Lemma 7.35(1), it must be an $\uparrow$-expression. ■

For expressible formal DNA molecules with nick letters, we always have to consider the *type* of nick letters that occur. In the case of lower nick letters, each minimal DNA expression is an $\uparrow$-expression, whose length is given by equality (7.27) in Theorem 7.46.

In the case of upper nick letters, each minimal DNA expression is a $\downarrow$-expression, whose length is given by an analogous equality.

We finally consider the *arguments* of the minimal DNA expressions we construct by Theorem 7.46. These are precisely the arguments of the operator-minimal $\uparrow$-expressions $E_1, \ldots, E_m$ denoting the nick free pieces $Z_1, \ldots, Z_m$. By Theorem 4.3, we may assume that the $\mathcal{N}$-word-arguments of each E_h are maximal $\mathcal{N}$-word occurrences in E_h. Moreover, if E_h happens to be constructed according to Theorem 7.42, then this assumption is also justified by Lemma 7.45.

Lemma 7.50 *Let X be an expressible formal DNA molecule which contains at least one lower nick letter $_\triangle$, and let $Z_1{}_\triangle Z_2{}_\triangle \ldots {}_\triangle Z_m$ for some $m \geq 2$ be the nick free decomposition of X. For $h = 1, \ldots, m$, let E_h be an operator-minimal $\uparrow$-expression denoting Z_h, and assume that each $\mathcal{N}$-word-argument of E_h is a maximal $\mathcal{N}$-word occurrence in E_h. Finally, let E be the minimal DNA expression denoting X which is based on $E_1, \ldots, E_m$, as described in Theorem 7.46.*

1. *Each $\mathcal{N}$-word-argument of E is a maximal $\mathcal{N}$-word occurrence in E.*
2. *E has (at least) two consecutive expression-arguments.*

Hence, in one respect, the arguments of E resemble the arguments of (operator-)minimal DNA expressions denoting nick free formal DNA molecules, which we considered in Lemma 7.27 and Lemma 7.45, in another respect they do not.

Claim 1 rules out the possibility that two or more, originally distinct $\mathcal{N}$-word-arguments are concatenated by the construction from Theorem 7.46. Each $\mathcal{N}$-word-argument of any E_h remains a maximal $\mathcal{N}$-word occurrence, when it becomes an argument of E. In particular, there does not exist an index h with $1 \leq h \leq m-1$, such that both the last argument of E_h and the first argument of E_{h+1} are $\mathcal{N}$-words.

Proof: Consider any h with $1 \leq h \leq m-1$. By definition, the lower nick letter between Z_h and Z_{h+1} is both preceded and succeeded in X by a double $\mathcal{A}$-letter: $R(Z_h), L(Z_{h+1}) \in \mathcal{A}_\pm$. This implies that both the last component of Z_h and the first component of Z_{h+1} are double components.

E_h is an $\uparrow$-expression which denotes Z_h. If the last argument of E_h were an $\mathcal{N}$-word, then the last component of Z_h would be an upper component. Hence, the last argument of E_h is a DNA expression. Analogously, the first argument of E_{h+1} is a DNA expression.

1. Consider an arbitrary $\mathcal{N}$-word-argument α of E. By the construction from Theorem 7.46, there exists h with $1 \leq h \leq m$, such that α is an $\mathcal{N}$-word-argument of E_h. By assumption, α is a maximal $\mathcal{N}$-word occurrence in E_h.

   If α is the last argument of E_h, then h must be equal to m and α is also the last argument of E, which is succeeded by the closing bracket $\rangle$ of E. If α is not the last argument of E_h, then the argument succeeding it in E_h is a DNA expression. Obviously, this argument also succeeds α in E.

   In neither case, α is succeeded in E by another $\mathcal{N}$-word.

2. Consider any h with $1 \leq h \leq m-1$. By construction, the last argument of E_h and the first argument of E_{h+1} are consecutive arguments of E. As we have observed at the beginning of the proof, both arguments are DNA expressions.

□

Chapter 8

All Minimal DNA Expressions

In Chapter 7, we have described how to construct a minimal DNA expression for a given (expressible) formal DNA molecule. For many formal DNA molecules, the applicable construction is not completely deterministic. It may yield several different, equivalent DNA expressions, each of which is minimal (see, e.g., Example 7.26). To prove the minimality of the resulting DNA expressions, we used the lower bounds from Chapter 6.

Both for the analysis in Chapter 6 and for the constructions in Chapter 7, the starting point was a formal DNA molecule. In this chapter, our starting point is a minimal DNA expression. We study the set of *all* minimal DNA expressions.

In Section 8.1, we demonstrate that each minimal DNA expression satisfies the descriptions from Chapter 7, i.e., that there do not exist other minimal DNA expressions. Subsequently, Section 8.2 elaborates a bit on the shortest $\updownarrow$-expressions possible for molecules with nicks. In Section 8.3. we present a characterization of minimal DNA expressions. By this, we can recognize a minimal DNA expression without determining its length. After that, we consider the structure trees of minimal DNA expressions, in Section 8.4. Finally, in Section 8.5, we calculate the number of different minimal DNA expressions for a given expressible formal DNA molecule.

8.1 Reverse construction of a minimal DNA expression

For double-complete formal DNA molecules, our specification of minimal DNA expressions is complete. By Theorem 7.5, the only minimal DNA expression denoting the molecule $\binom{\alpha_1}{c(\alpha_1)}$ for an $\mathcal{N}$-word α_1 is $\langle\updownarrow \alpha_1\rangle$.

By Theorem 7.24 and Theorem 7.46, we also know how to construct minimal DNA expressions for expressible formal DNA molecules containing single-stranded components and/or nick letters. It is, however, not a priori clear that there do not exist other minimal DNA expressions denoting these molecules. There might be minimal DNA expressions which do not fit the constructions given in Theorem 7.24 and Theorem 7.46.

Actually, all we know about the structure of these DNA expressions is what the outermost operators may be. Let X be an expressible formal DNA molecule which contains single-stranded components and/or nick letters. By Corollary 7.6, each minimal DNA expression denoting X is an $\uparrow$-expression or a $\downarrow$-expression. In many cases, the outermost operator of the minimal DNA expression(s) is completely determined. If X is nick free and $B_\uparrow(X) \neq B_\downarrow(X)$, then the outermost operator is determined by Lemma 7.30. If X

contains nick letters, then the outermost operator is determined by Lemma 7.35.

In this section, we will investigate the structure of arbitrary minimal $\uparrow$-expressions and arbitrary minimal $\downarrow$-expressions. In fact, we will find that there do not exist minimal $\uparrow$-expressions and $\downarrow$-expressions other than those described by Theorem 7.24 and Theorem 7.46. Because results on $\uparrow$-expressions and results on $\downarrow$-expressions are completely analogous and can be proved in a completely analogous way, we will only give the results for the $\uparrow$-expressions.

Initially, we consider *operator-minimal* $\uparrow$-expressions rather than minimal $\uparrow$-expressions. However, because minimal DNA expressions are in particular operator-minimal, the results we achieve for operator-minimal $\uparrow$-expressions are certainly valid for minimal $\uparrow$-expressions.

We start with a simple (but essential) result:

Lemma 8.1 *Let E be an operator-minimal $\uparrow$-expression denoting a certain formal DNA molecule X (which may contain nick letters).*

Then no argument of E is an $\uparrow$-expression.

Proof: Assume that $E = \langle\uparrow \varepsilon_1 \dots \varepsilon_n\rangle$ for some $n \geq 1$ and $\mathcal{N}$-words and DNA expressions $\varepsilon_1, \dots, \varepsilon_n$. Suppose that for some i, the argument ε_i is an $\uparrow$-expression $E_i = \langle\uparrow \varepsilon_{i,1} \dots \varepsilon_{i,n_i}\rangle$ for some $n_i \geq 1$ and arguments $\varepsilon_{i,1}, \dots, \varepsilon_{i,n_i}$. Then by Lemma 5.10, the DNA expression

$$E' = \langle\uparrow \varepsilon_1 \dots \varepsilon_{i-1}\varepsilon_{i,1} \dots \varepsilon_{i,n_i}\varepsilon_{i+1} \dots \varepsilon_n\rangle\,,$$

which is three letters shorter than E and has the same outermost operator, is equivalent to E. This contradicts the operator-minimality of E. □

If an argument of an operator-minimal DNA expression is itself a DNA expression, then it must be minimal. Hence, when we combine Theorem 7.5 and Lemma 8.1, we find

Corollary 8.2 *Let E be an operator-minimal $\uparrow$-expression denoting a certain formal DNA molecule X (which may contain nick letters).*

Then each argument of E is either an $\mathcal{N}$-word α, or an $\updownarrow$-expression $\langle\updownarrow \alpha\rangle$ for an $\mathcal{N}$-word α, or a $\downarrow$-expression.

Let X be a nick free formal DNA molecule that contains at least one single-stranded component and for which $B_\uparrow(X) \geq B_\downarrow(X)$. When we construct a minimal $\uparrow$-expression E denoting X according to Theorem 7.24(1), the arguments of E are $\mathcal{N}$-words α_i, $\updownarrow$-expressions $\langle\updownarrow \alpha_i\rangle$ for $\mathcal{N}$-words α_i, and $\downarrow$-expressions E_j denoting lower blocks X_j of X. In particular, for each argument ε of E, $\mathcal{S}^+(\varepsilon)$ is nick free.

The same properties hold for the arguments of an operator-minimal $\uparrow$-expression denoting a nick free formal DNA molecule, which is constructed according to Theorem 7.42.

Finally, let X be an expressible formal DNA molecule containing $m - 1 \geq 1$ lower nick letters, and let $E_1, \dots, E_m$ be operator-minimal $\uparrow$-expressions denoting the nick free pieces $Z_1, \dots, Z_m$ of X. If $E_1, \dots, E_m$ have been constructed according to Theorem 7.42, then by the above, for each argument ε of any of the E_h's, $\mathcal{S}^+(\varepsilon)$ is nick free. Now, when we use $E_1, \dots, E_m$ to construct a minimal $\uparrow$-expression E denoting the entire formal DNA molecule X according to Theorem 7.46, the arguments of E are precisely the arguments of $E_1, \dots, E_m$. Hence, again, for each argument ε of E, $\mathcal{S}^+(\varepsilon)$ is nick free.

In general, whether or not the formal DNA molecule X denoted by an $\uparrow$-expression E contains nick letters, there may be nicks in the arguments of E.

Example 8.3 Consider the DNA expression

$$E = \langle\uparrow \alpha_1 \langle\downarrow \langle\updownarrow \alpha_2\rangle \langle\updownarrow \alpha_3\rangle \alpha_4\rangle\rangle,$$

for $\mathcal{N}$-words $\alpha_1, \alpha_2, \alpha_3, \alpha_4$. We have $\mathcal{S}(E) = \binom{\alpha_1}{-}\binom{\alpha_2\alpha_3}{c(\alpha_2\alpha_3)}\binom{-}{\alpha_4}$, which is a nick free formal DNA molecule. The second argument of E, however, is not nick free, because $\mathcal{S}(\langle\downarrow \langle\updownarrow \alpha_2\rangle \langle\updownarrow \alpha_3\rangle \alpha_4\rangle) = \binom{\alpha_2}{c(\alpha_2)}^{\triangledown}\binom{\alpha_3}{c(\alpha_3)}\binom{-}{\alpha_4}$. ■

We will now see that an $\uparrow$-expression cannot be operator-minimal, let alone minimal, if it has at least one argument denoting a molecule with nicks. Hence, the property that for each argument ε, $\mathcal{S}^+(\varepsilon)$ is nick free, does not only hold for the (operator-)-minimal $\uparrow$-expressions that we construct according to Theorem 7.24(1), Theorem 7.42 or Theorem 7.46; it holds for any operator-minimal $\uparrow$-expression, no matter how it has been constructed.

Indeed, the $\uparrow$-expression E from Example 8.3, whose second argument has a nick in the upper strand, is not operator-minimal. The equivalent $\uparrow$-expression

$$E' = \langle\uparrow \alpha_1 \langle\downarrow \langle\updownarrow \alpha_2\alpha_3\rangle \alpha_4\rangle\rangle$$

is three letters shorter.

Lemma 8.4 *Let $E = \langle\uparrow \varepsilon_1 \dots \varepsilon_n\rangle$, where $n \geq 1$ and $\varepsilon_1, \dots, \varepsilon_n$ are $\mathcal{N}$-words and DNA expressions, be an operator-minimal DNA expression denoting a certain formal DNA molecule X (which may contain nick letters).*

Then for $i = 1, \dots, n$, $X_i = \mathcal{S}^+(\varepsilon_i)$ is nick free.

Hence, each nick letter occurring in X (if any) has been introduced by the outermost operator $\uparrow$ of E.

Proof: For $i = 1, \dots, n$, let $X_i = \mathcal{S}^+(\varepsilon_i)$. Then

$$X = \nu^+(X_1)y_1\nu^+(X_2)y_2 \dots y_{n-1}\nu^+(X_n), \tag{8.1}$$

where for $i = 1, \dots, n-1$, y_i is either equal to λ or to $\triangle$, depending on $R(X_i)$ and $L(X_{i+1})$ (see Definition 4.1).

Consider an argument ε_i for some i with $1 \leq i \leq n$. By Corollary 8.2, ε_i is either an $\mathcal{N}$-word α, or an $\updownarrow$-expression $\langle\updownarrow \alpha\rangle$ for an $\mathcal{N}$-word α, or a $\downarrow$-expression.

If ε_i is an $\mathcal{N}$-word α, then $X_i = \mathcal{S}^+(\varepsilon_i) = \binom{\alpha}{-}$, which is indeed nick free.

If ε_i is an $\updownarrow$-expression $\langle\updownarrow \alpha\rangle$ for an $\mathcal{N}$-word α, then $X_i = \mathcal{S}(\varepsilon_i) = \binom{\alpha}{c(\alpha)}$, which is also nick free.

Now, assume that ε_i is a $\downarrow$-expression E_i. By Lemma 5.2(1), $X_i = \mathcal{S}(E_i)$ does not contain lower nick letters. Consequently, $\nu^+(X_i)$ does not contain any nick letters.

By Theorem 6.31(2),

$$|E_i| \geq 3 + 3 \cdot B_{\uparrow}(X_i) + 3 \cdot n_{\updownarrow}(X_i) + |X_i|_{\mathcal{A}}, \tag{8.2}$$

and by Lemma 6.16(1),

$$B_{\uparrow}(X_i) \geq B_{\uparrow}(\nu^+(X_i)). \tag{8.3}$$

Suppose that X_i is not nick free. Then it must contain at least one upper nick letter: $\#_{\triangledown}(X_i) \geq 1$. Hence, by Lemma 6.16(5),

$$n_{\updownarrow}(X_i) > n_{\updownarrow}(\nu^+(X_i)). \tag{8.4}$$

Substituting (8.3) and (8.4) into (8.2), we obtain

$$|E_i| > 3 + 3 \cdot B_{\uparrow}(\nu^+(X_i)) + 3 \cdot n_{\updownarrow}(\nu^+(X_i)) + |X_i|_{\mathcal{A}}.$$

On the other hand, because $\nu^+(X_i)$ is nick free, we know by Corollary 7.29(2) that a minimal DNA-expression E_i' denoting $\nu^+(X_i)$ satisfies

$$|E_i'| \leq 3 + 3 \cdot B_{\uparrow}(\nu^+(X_i)) + 3 \cdot n_{\updownarrow}(\nu^+(X_i)) + |X_i|_{\mathcal{A}}$$

(obviously, $|X_i|_{\mathcal{A}} = |\nu^+(X_i)|_{\mathcal{A}}$).

Now, if we replace the argument E_i of E by the shorter DNA expression E_i', which satisfies $E_{i\ \triangledown} \equiv E_i'$, then by Lemma 5.11, the resulting string E' is a DNA expression satisfying $E_{\ \triangledown} \equiv E'$. Because $\nu^+(\mathcal{S}^+(E_i')) = \nu^+(\nu^+(X_i)) = \nu^+(X_i)$ and by Lemma 3.11, $L(\nu^+(X_i)) = L(X_i)$ and $R(\nu^+(X_i)) = R(X_i)$, we even have $\mathcal{S}(E') = X$ (see (8.1)). In other words, $E \equiv E'$.

Because E' is shorter than E and has the same outermost operator as E, E would not be operator-minimal, which contradicts our assumption. Hence, also in the case that ε_i is a $\downarrow$-expression E_i, must X_i be nick free. □

By Lemma 8.4, the expression-arguments of an operator-minimal $\uparrow$-expression E are nick free. It is not difficult to generalize this to arbitrary proper DNA subexpressions of E.

Corollary 8.5 *Let E be an operator-minimal $\uparrow$-expression denoting a certain formal DNA molecule X (which may contain nick letters). Then each proper DNA subexpression of E is nick free.*

Proof: Let E_1 be a proper DNA subexpression of E. By Lemma 7.41, E_1 is minimal. Hence, if E_1 is an $\updownarrow$-subexpression of E, then by Theorem 7.5, $E_1 = \langle \updownarrow \alpha_1 \rangle$ for an $\mathcal{N}$-word α_1, which is indeed nick free.

Now, assume that E_1 is a $\downarrow$-subexpression of E. Let E_0 be the DNA subexpression of E that E_1 is an argument of. If E_0 is equal to E, then by assumption E_0 is an operator-minimal $\uparrow$-expression. If, on the other hand, E_0 is a proper DNA subexpression of E, then E_0 is minimal. Hence, by Theorem 7.5, and Lemma 8.1, E_0 is an $\uparrow$-expression. In particular, also in this case, E_0 is an operator-minimal $\uparrow$-expression. Now for both cases, the claim follows from Lemma 8.4, applied to E_0.

The proof for the case that E_1 is an $\uparrow$-subexpression of E is analogous. However, in that case, we do not have to consider the possibility that E_0 is equal to E, because the operator-minimal $\uparrow$-expression E cannot have an $\uparrow$-argument E_1. □

In Lemma 7.17 and Lemma 7.23(2), we considered the value of the function $B_{\downarrow}$ for a single lower block and for the substrings occurring in a lower block partitioning, respectively. We now consider this value for the formal DNA submolecules corresponding to the arguments of an operator-minimal $\uparrow$-expression.

Lemma 8.6 *Let $E = \langle \uparrow \varepsilon_1 \dots \varepsilon_n \rangle$, where $n \geq 1$ and $\varepsilon_1, \dots, \varepsilon_n$ are $\mathcal{N}$-words and DNA expressions, be an operator-minimal DNA expression denoting a certain formal DNA molecule X (which may contain nick letters). For $i = 1, \dots, n$, let $X_i = \mathcal{S}^+(\varepsilon_i)$.*

1. *For $i = 1, \dots, n$, if ε_i is a $\downarrow$-expression E_i, then $X_i = \mathcal{S}(E_i)$ and $B_{\uparrow}(X_i) = B_{\downarrow}(X_i) - 1$. Hence, X_i contains at least one single-stranded component and both the first single-stranded component and the last single-stranded component of X_i are lower components.*

2. $$\sum_{\downarrow\text{-}expr.\ \varepsilon_i} B_\downarrow(X_i) = B_\downarrow(X_1) + \cdots + B_\downarrow(X_n) = B_\downarrow(X).$$

Proof:

1. Consider an argument ε_i that is a $\downarrow$-expression E_i. By definition, $X_i = \mathcal{S}(E_i)$, and by Lemma 8.4, X_i is nick free. Because E is operator-minimal, E_i is minimal.

   Suppose that $B_\uparrow(X_i) \geq B_\downarrow(X_i)$.

   • If $B_\uparrow(X_i) = 0$, then also $B_\downarrow(X_i) = 0$, and by Lemma 6.13(1), X_i does not contain any single-stranded component. Hence, by Lemma 6.13(2), $X_i = \binom{\alpha}{c(\alpha)}$ for an $\mathcal{N}$-word α. This, however, leads to a contradiction, because by Theorem 7.5, the unique minimal DNA expression denoting $X_i = \binom{\alpha}{c(\alpha)}$ is $E_i' = \langle\updownarrow \alpha\rangle$.

   • If $B_\uparrow(X_i) \geq 1$, then X_i contains at least one upper component. By Theorem 7.24(1), there exists a minimal $\uparrow$-expression E_i' denoting X_i.

   Because $E_i \equiv E_i'$ and both of them are minimal, they are equally long. Now, let us substitute the argument E_i of E by E_i'. By Lemma 5.11, the resulting overall string E' is a DNA expression satisfying $E \equiv E'$. Obviously, E' has the same length as E and has the same outermost operator, which implies that E' is operator-minimal, just like E.

   This, however, contradicts Lemma 8.1, because E' is an $\uparrow$-expression and one of its arguments, E_i', is also an $\uparrow$-expression.

   Both if $B_\uparrow(X_i) = 0$ and if $B_\uparrow(X_i) \geq 1$, we end up in a contradiction. This implies that $B_\uparrow(X_i) < B_\downarrow(X_i)$. By Lemma 6.12(2), $B_\uparrow(X_i) = B_\downarrow(X_i) - 1$. Hence, X_i contains at least one lower component and by Lemma 6.13(3), both the first single-stranded component and the last single-stranded component of X_i are lower components.

2. Consider an argument ε_i for some i with $1 \leq i \leq n$.

   By Corollary 8.2, ε_i is either an $\mathcal{N}$-word α, or an $\updownarrow$-expression $\langle\updownarrow \alpha\rangle$ for an $\mathcal{N}$-word α, or a $\downarrow$-expression. If ε_i is an $\mathcal{N}$-word α (in which case $X_i = \binom{\alpha}{_}$) or an $\updownarrow$-expression $\langle\updownarrow \alpha\rangle$ ($X_i = \binom{\alpha}{c(\alpha)}$), then obviously $B_\downarrow(X_i) = 0$. This gives us the first equality in the claim.

   By Lemma 8.4, $X_i = \mathcal{S}^+(\varepsilon_i)$ is nick free. In particular, $\#_\triangledown(X_i) = 0$. But then the second equality in the claim follows immediately from inequalities (6.4) and (6.5) in Lemma 6.27(1).

□

Definition 4.1, and in particular equation (4.2), describes the semantics of a general $\uparrow$-expression. Now that we have derived some properties of operator-minimal $\uparrow$-expressions, we can simplify the definition of the semantics for such $\uparrow$-expressions.

Lemma 8.7 *Let $E = \langle\uparrow \varepsilon_1 \ldots \varepsilon_n\rangle$, where $n \geq 1$ and $\varepsilon_1, \ldots, \varepsilon_n$ are $\mathcal{N}$-words and DNA expressions, be an operator-minimal DNA expression denoting a certain formal DNA molecule X. For $i = 1, \ldots, n$, let $X_i = \mathcal{S}^+(\varepsilon_i)$. Then*

$$X = X_1 y_1 X_2 y_2 \ldots y_{n-1} X_n,$$

where for $i = 1, \ldots, n-1$, $y_i = \triangle$ if $R(X_i), L(X_{i+1}) \in \mathcal{A}_\pm$, and $y_i = \lambda$ otherwise.

Here, for $i = 1, \ldots, n-1$, $R(X_i), L(X_{i+1}) \in \mathcal{A}_\pm$, if and only if both ε_i and ε_{i+1} are expression-arguments.

Hence, for operator-minimal $\uparrow$-expressions, there is a nick between two consecutive arguments ε_i and ε_{i+1}, if and only if both ε_i and ε_{i+1} are expression-arguments.

Proof: By the definition of the semantics of an $\uparrow$-expression (equation (4.2)),

$$X = \nu^+(X_1)y_1\nu^+(X_2)y_2 \ldots y_{n-1}\nu^+(X_n),$$

where for $i = 1, \ldots, n-1$, $y_i = \triangle$ if $R(X_i), L(X_{i+1}) \in \mathcal{A}_\pm$, and $y_i = \lambda$ otherwise. By Lemma 8.4, for $i = 1, \ldots, n$, X_i is nick free, and in particular, $\nu^+(X_i) = X_i$. We can thus reduce the semantics to

$$X = X_1y_1X_2y_2 \ldots y_{n-1}X_n,$$

with y_i's as before. This is the first part of the claim.

Next, consider any i with $1 \leq i \leq n-1$. By Corollary 8.2, ε_i is either an $\mathcal{N}$-word α, or an $\updownarrow$-expression $\langle \updownarrow \alpha \rangle$ for an $\mathcal{N}$-word α, or a $\downarrow$-expression.

If ε_i is an $\mathcal{N}$-word α, then $X_i = \mathcal{S}^+(\varepsilon_i) = \binom{\alpha}{-}$ and $R(X_i) \notin \mathcal{A}_\pm$.

If ε_i an $\updownarrow$-expression $\langle \updownarrow \alpha \rangle$ for an $\mathcal{N}$-word α, then $X_i = \mathcal{S}(\varepsilon_i) = \binom{\alpha}{c(\alpha)}$ and $R(X_i) \in \mathcal{A}_\pm$.

Finally, if ε_i is a $\downarrow$-expression, then by Lemma 8.6(1), X_i contains at least one single-stranded component and the last single-stranded component of X_i is a lower component. Because ε_i has to prefit ε_{i+1} by upper strands, this lower component cannot be the last component of X_i. By Corollary 3.8(1), the last component of X_i must be a double component. This implies that $R(X_i) \in \mathcal{A}_\pm$.

We conclude that $R(X_i) \in \mathcal{A}_\pm$, if and only if ε_i is an expression-argument. Analogously, we find that $L(X_{i+1}) \in \mathcal{A}_\pm$, if and only if ε_{i+1} is an expression-argument. Consequently, $R(X_i), L(X_{i+1} \in \mathcal{A}_\pm$, if and only if both ε_i and ε_{i+1} are expression-arguments. □

As we observed before, if an $\uparrow$-expression or a $\downarrow$-expression has two or more consecutive $\mathcal{N}$-word-arguments, then these arguments may be substituted by one argument being the concatenation of the $\mathcal{N}$-words. By Theorem 4.3, this substitution does not change the semantics of the DNA expression. In fact, the original DNA expression and the new DNA expression cannot even be distinguished from each other, unless one explicitly indicates what the arguments of each of them are. Hence, we do not lose generality when we assume that the arguments of an $\uparrow$-expression or a $\downarrow$-expression are maximal $\mathcal{N}$-word occurrences and DNA expressions. In the remainder of this section, we will explicitly make this assumption.

We now examine the relation between components of the formal DNA molecule X denoted by an operator-minimal $\uparrow$-expression E and the components of the arguments of E.

Lemma 8.8 *Let $E = \langle \uparrow \varepsilon_1 \ldots \varepsilon_n \rangle$, where $n \geq 1$ and $\varepsilon_1, \ldots, \varepsilon_n$ are maximal $\mathcal{N}$-word occurrences and DNA expressions, be an operator-minimal DNA expression denoting a certain formal DNA molecule X (which may contain nick letters). For $i = 1, \ldots, n$, let $X_i = \mathcal{S}^+(\varepsilon_i)$, and let $x'_1 \ldots x'_k$ for some $k \geq 1$ be the decomposition of X. Then for $i = 1, \ldots, n$,*

1. *each component of X_i is also a component of X (at a corresponding position);*
2. *there exist a_i and b_i with $1 \leq a_i \leq b_i \leq k$ such that $X_i = x'_{a_i} \dots x'_{b_i}$.*

In Claim 1, we added the phrase '(at a corresponding position)' to avoid misinterpretations, as illustrated by the following example:

Example 8.9 Consider the DNA expression

$$E = \langle\uparrow \langle\uparrow \alpha_1 \langle\updownarrow \alpha_2\rangle \alpha_1\rangle \alpha_1\rangle$$

for $\mathcal{N}$-words α_1 and α_2. We have $X_1 = \mathcal{S}^+(\varepsilon_1) = \binom{\alpha_1}{-}\binom{\alpha_2}{c(\alpha_2)}\binom{\alpha_1}{-}$, $X_2 = \mathcal{S}^+(\varepsilon_2) = \binom{\alpha_1}{-}$ and $X = \mathcal{S}(E) = \binom{\alpha_1}{-}\binom{\alpha_2}{c(\alpha_2)}\binom{\alpha_1\alpha_1}{-}$. Indeed, for each component of X_1 and each component of X_2, we can find an 'equal' component of X. However, the last component of X_1 and the only component of X_2 have merged into one component of X. The position in $X = X_1X_2$ of the component which is 'equal' to, e.g., the only component $\binom{\alpha_1}{-}$ of X_2 does not correspond to the position of this component in X_2. Not surprisingly, the $\uparrow$-expression E is not operator-minimal. ■

Proof of Lemma 8.8: By Lemma 8.7,

$$X = X_1y_1X_2y_2 \dots y_{n-1}X_n, \tag{8.5}$$

where for $i = 1, \dots, n-1$, $y_i = \triangle$ if both $R(X_i)$ and $L(X_{i+1})$ are double $\mathcal{A}$-letters, and $y_i = \lambda$ otherwise.

1. Obviously, no component of any X_i is split up over different components of X. Because X satisfies (8.5), it is also clear that no component of any X_i simply disappears. In particular, the X_i's do not have nick letters that are removed. To complete the proof, we have to demonstrate that no component of any X_i merges into a larger component of X.

   It is immediate from (8.5), that different components of the same X_i do not merge into the same component of X. By definition, if for some i with $1 \leq i \leq n-1$, $y_i = \triangle$, then it is a component of X by itself. Hence, only if $y_i = \lambda$ and the last component of X_i and the first component of X_{i+1} are of the same type (upper component, lower component or double component) then these two components merge into one component of X. No other component of any X_i merges into a larger component of X.

   Consider any i with $1 \leq i \leq n-1$, such that $y_i = \lambda$. Because the arguments ε_i and ε_{i+1} have to fit together by upper strands, neither the last component of X_i, nor the first component of X_{i+1} can be a lower component. Because $y_i = \lambda$, the two components cannot both be double components, either. At least one of them has to be an upper component.

   Without loss of generality, assume that the last component of X_i is an upper component. By Corollary 8.2 and Lemma 8.6(1), ε_i is a maximal $\mathcal{N}$-word occurrence. By definition, ε_{i+1} is not an $\mathcal{N}$-word, and thus is either an $\updownarrow$-expression $\langle\updownarrow \alpha\rangle$ for an $\mathcal{N}$-word α, or a $\downarrow$-expression. In neither case, the first component of X_{i+1} is an upper component. Consequently, the last component of X_i and the first component of X_{i+1} do not merge into one component.

2. This claim follows immediately from (8.5) and the previous claim.

□

We temporarily focus on *nick free* formal DNA molecules. We first rephrase Lemma 8.7 for the nick free case. By Lemma 8.7, a nick free, operator-minimal ↑-expression E cannot have consecutive expression-arguments. By definition, it cannot have consecutive arguments that are maximal $\mathcal{N}$-word occurrences, either. We thus have

Corollary 8.10 *Let $E = \langle\uparrow \varepsilon_1 \dots \varepsilon_n\rangle$, where $n \geq 1$ and $\varepsilon_1, \dots, \varepsilon_n$ are maximal $\mathcal{N}$-word occurrences and DNA expressions, be an operator-minimal DNA expression denoting a certain nick free formal DNA molecule X. For $i = 1, \dots, n$, let $X_i = \mathcal{S}^+(\varepsilon_i)$.*

Then

$$X = X_1 X_2 \dots X_n, \tag{8.6}$$

and the arguments $\varepsilon_1, \dots, \varepsilon_n$ are maximal $\mathcal{N}$-word occurrences and DNA expressions, alternately.

Recall that in Lemma 7.27 and Lemma 7.45, we considered the arguments of the (operator-)minimal ↑-expressions and ↓-expressions we obtain for nick free formal DNA molecules according to the constructions from Theorem 7.24 and Theorem 7.42. We established that the arguments form an alternating sequence of $\mathcal{N}$-words and DNA expressions. By the above, we have the same result for *arbitrary* operator-minimal ↑-expressions denoting nick free formal DNA molecules.

Recall also that by Lemma 5.8, if each occurrence of ↑ or ↓ in a DNA expression E is alternating, then $X = \mathcal{S}(E)$ is nick free. We can use Corollary 8.10 to prove that for operator-minimal ↑-expressions, the converse is also true.

Corollary 8.11 *Let E be an operator-minimal ↑-expression denoting a certain formal DNA molecule X. The following four statements are equivalent:*

1. *X is nick free.*
2. *X does not contain lower nick letters.*
3. *(The outermost operator ↑ of) E is alternating.*
4. *Each occurrence of ↑ or ↓ in E is alternating.*

Proof: The equivalence of statements 1 and 2 follows from Lemma 5.1(1). We now prove that statements 3 and 4 are also equivalent to statement 1.

1 $\Longrightarrow$ 3 This implication follows directly from Corollary 8.10.

3 $\Longleftrightarrow$ 4 Consider an arbitrary *inner* occurrence of ↑ or ↓ in E, and let E^s be the (proper) DNA subexpression of E governed by it. By Lemma 7.41, E^s is minimal, and by Corollary 8.5, E^s is nick free. Hence, we can apply Corollary 8.10 to E^s, and conclude that the occurrence of ↑ or ↓ that we consider is alternating.

This implies that the outermost operator ↑ of E is alternating, if and only if *each* occurrence of ↑ or ↓ in E is alternating.

4 $\Longrightarrow$ 1 This implication follows directly from Lemma 5.8.

□

We now consider the $\downarrow$-arguments of an operator-minimal $\uparrow$-expression denoting a nick free molecule.

Lemma 8.12 *Let $E = \langle\uparrow \varepsilon_1 \dots \varepsilon_n\rangle$, where $n \geq 1$ and $\varepsilon_1, \dots, \varepsilon_n$ are maximal $\mathcal{N}$-word occurrences and DNA expressions, be an operator-minimal DNA expression denoting a certain nick free formal DNA molecule X. For $i = 1, \dots, n$, let $X_i = \mathcal{S}^+(\varepsilon_i)$ and let $x'_{i,1} \dots x'_{i,k_i}$ for some $k_i \geq 1$ be the decomposition of X_i.*

1. *For $i = 1, \dots, n-1$, if ε_i is a $\downarrow$-expression, then $k_i \geq 2$, x'_{i,k_i-1} is a lower component of X, x'_{i,k_i} is a double component of X, ε_{i+1} is a maximal $\mathcal{N}$-word occurrence α and $X_{i+1} = \binom{\alpha}{-}$ is an upper component of X.*

2. *For $i = 2, \dots, n$, if ε_i is a $\downarrow$-expression, then $k_i \geq 2$, $x'_{i,2}$ is a lower component of X, $x'_{i,1}$ is a double component of X, ε_{i-1} is a maximal $\mathcal{N}$-word occurrence α and $X_{i-1} = \binom{\alpha}{-}$ is an upper component of X.*

3. *For $i = 1, \dots, n$, if ε_i is a $\downarrow$-expression, then X_i is a lower block of X.*

Proof: Claims 1 and 2 are completely analogous and so are their proofs. We give the proof of Claim 2. First, however, we make some general observations, which are also useful in the proof of Claim 3.

Consider a $\downarrow$-argument ε_i of E, with $1 \leq i \leq n$. By Lemma 8.4, $X_i = \mathcal{S}(\varepsilon_i)$ is nick free. By Lemma 8.6(1), X_i contains at least one single-stranded component and the first single-stranded component of X_i is a lower component of X_i. Hence, by Corollary 3.8(1), $x'_{i,1}$ is either a lower component, or a double component of X_i. In the latter case, by Corollary 3.8(2), $k_i \geq 2$ and $x'_{i,2}$ is a lower component of X_i. By Lemma 8.8(1), these are also components of X.

We now examine properties of X_i (for the $\downarrow$-argument ε_i) which are specific for Claims 2 and 3, respectively.

2. Assume that $2 \leq i \leq n$.

   Because ε_{i-1} has to prefit ε_i by upper strands, $x'_{i,1}$ cannot be a lower component. This implies that it is a double component of X_i (and of X), $k_i \geq 2$ and $x'_{i,2}$ is a lower component of X_i (and of X).

   By Corollary 8.10, ε_{i-1} is a maximal $\mathcal{N}$-word occurrence α. Hence, $X_{i-1} = \binom{\alpha}{-}$ is an upper $\mathcal{A}$-word. By Lemma 8.8(1), it is an upper component of X.

3. The first single-stranded component of X_i, which is a lower component, is also a lower component of X. Either this lower component is $x'_{i,1}$, or it is $x'_{i,2}$. Let us use $x'_{i,\cdot}$ to denote it. By Lemma 6.7(2), $x'_{i,\cdot}$ is part of a primitive lower block X'_1 of X. We examine the relation between the first component $x'_{i,1}$ of X_i and the primitive lower block X'_1.

   If $x'_{i,\cdot} = x'_{i,1}$, then obviously $x'_{i,1}$ is part of X'_1. If, on the other hand, $x'_{i,\cdot} = x'_{i,2}$, then $x'_{i,1}$ is a double component. By Lemma 6.6(1), the primitive lower block X'_1 cannot start with the lower component $x'_{i,2}$ that it contains. Hence, also in this case, $x'_{i,1}$ is part of X'_1.

   If $i = 1$, then $x'_{i,1}$ is the first component of X, and clearly, it is also the first component of X'_1. If, on the other hand, $i \geq 2$, then by Claim 2, ε_{i-1} is a maximal

$\mathcal{N}$-word occurrence α, and $X_{i-1} = \binom{\alpha}{-}$ is an upper component of X. Because, by Corollary 8.10, $X = X_1 X_2 \ldots X_n$, this upper component immediately precedes $x'_{i,1}$ in X. By definition, it is not part of any primitive lower block. Hence, also in this case, $x'_{i,1}$ is the first component of X'_1.

Completely analogously, we can prove that the last component x'_{i,k_i} of X_i is the last component of a primitive lower block (which may be different from X'_1). We conclude that X_i is a substring of X which starts with a primitive lower block and ends with a primitive lower block. In other words, X_i is a lower block of X.

□

We have established many properties of arbitrary operator-minimal ↑-expressions. We use them to prove that the constructions from Theorem 7.24(1), Theorem 7.42 and Theorem 7.46 are the only ways to obtain a minimal or operator-minimal ↑-expression. We start with minimal ↑-expressions denoting nick free formal DNA molecules. They are described by Theorem 7.24(1):

Theorem 8.13 *Let $E = \langle\uparrow \varepsilon_1 \ldots \varepsilon_n\rangle$, where $n \geq 1$ and $\varepsilon_1, \ldots, \varepsilon_n$ are maximal $\mathcal{N}$-word occurrences and DNA expressions, be a minimal DNA expression denoting a certain nick free formal DNA molecule X. For $i = 1, \ldots, n$, let $X_i = \mathcal{S}^+(\varepsilon_i)$.*

Let $\varepsilon_{i_1}, \varepsilon_{i_2}, \ldots, \varepsilon_{i_r}$, with $0 \leq r \leq n$ and $i_1 < i_2 < \cdots < i_r$, be all ↓-arguments of E. Finally, let $Y_0, Y_1, \ldots, Y_r$ be defined by

$$
\begin{aligned}
Y_0 &= \begin{cases} X_1 \ldots X_n & \text{if } r = 0 \\ X_1 \ldots X_{i_1 - 1} & \text{if } r \geq 1 \end{cases} \\
Y_j &= X_{i_j + 1} \ldots X_{i_{j+1} - 1} \qquad (j = 1, \ldots, r-1) \\
Y_r &= \begin{cases} X_1 \ldots X_n & \text{if } r = 0 \\ X_{i_r + 1} \ldots X_n & \text{if } r \geq 1 \end{cases}
\end{aligned}
$$

1. *$\mathcal{P} = Y_0 X_{i_1} Y_1 X_{i_2} Y_2 \ldots X_{i_r} Y_r$ is a lower block partitioning of X.*
2. *E satisfies the description of a minimal DNA expression denoting X and based on $\mathcal{P}$, given in Theorem 7.24(1).*

Intuitively, Y_0 is the concatenation of X_i's preceding the first ↓-argument ε_{i_1}, Y_r is the concatenation of X_i's succeeding the last ↓-argument ε_{i_r}, and for $j = 1, \ldots, r-1$, Y_j is the concatenation of X_i's separating the ↓-expressions ε_{i_j} and $\varepsilon_{i_{j+1}}$.

Proof: Because a minimal DNA expression is in particular operator-minimal, all earlier results in this section are also applicable to E.

By Corollary 8.2, each argument ε_i of E which is not a ↓-expression, is either a maximal $\mathcal{N}$-word occurrence α, or an ↕-expression $\langle\updownarrow \alpha\rangle$ for an $\mathcal{N}$-word α. The corresponding formal DNA molecule $X_i = \mathcal{S}^+(\varepsilon_i)$ is $\binom{\alpha}{-}$ or $\binom{\alpha}{c(\alpha)}$, respectively. By Lemma 8.8(1), this is a component of X; in particular, it is an upper component or a double component of X.

1. We verify that $\mathcal{P} = Y_0 X_{i_1} Y_1 X_{i_2} Y_2 \ldots X_{i_r} Y_r$ satisfies all conditions of a lower block partitioning of X.

   If $r \geq 1$, then, as we observed before the proof, Y_0 is the concatenation of X_i's preceding ε_{i_1}, Y_r is the concatenation of X_i's succeeding ε_{i_r} and for $j = 1, \ldots, r-1$,

Y_j is the concatenation of X_i's between ε_{i_j} and $\varepsilon_{i_{j+1}}$. Now, both if $r = 0$ and if $r \geq 1$, it follows immediately that

$$Y_0 X_{i_1} Y_1 X_{i_2} Y_2 \dots X_{i_r} Y_r = X_1 \dots X_n.$$

By Corollary 8.10, this equals X.

By Lemma 8.12(3), for $j = 1, \dots, r$, X_{i_j} is a lower block of X.

For $j = 0, \dots, r$, Y_j is the concatenation of a number of upper components $\binom{\alpha}{-}$ and double components $\binom{\alpha}{c(\alpha)}$ for $\mathcal{N}$-words α. Because, by definition, a primitive lower block of X contains at least one lower component, no Y_j contains a (complete) primitive lower block.

Suppose that X has a primitive lower block that is partly in a sequence Y_j and partly in a lower block X_{i_j}. Then this primitive lower block would intersect with another primitive lower block, because each lower block X_{i_j} starts with a (complete) primitive lower block and ends with a (complete) primitive lower block. This would contradict Lemma 6.7(1). Hence, each primitive lower block is contained in one of the lower blocks X_{i_j}.

We conclude that $Y_0 X_{i_1} Y_1 X_{i_2} Y_2 \dots X_{i_r} Y_r$ is a lower block partitioning of X.

Note that, if $r = 0$, hence if no argument ε_i of E is a $\downarrow$-expression, then for $i = 1, \dots, n$, X_i is an upper component or a double component of X. By Lemma 7.19, the only lower block partitioning of X is $\mathcal{P} = X$. Indeed, $Y_0 = X_1 \dots X_n = X$ in this case.

2. We first establish that Theorem 7.24(1) is applicable to X.

   By assumption, X is nick free. By Theorem 7.5, X is not equal to $\binom{\alpha}{c(\alpha)}$ for an $\mathcal{N}$-word α. because the only minimal DNA expression denoting such a formal DNA molecule is $\langle \updownarrow \alpha \rangle$, whereas E is an $\uparrow$-expression. Hence, by Lemma 6.13(2), X contains at least one single-stranded component.

   Finally, by Lemma 7.30(2), if $B_\downarrow(X) > B_\uparrow(X)$, then each minimal DNA expression denoting X would be a $\downarrow$-expression. Because E is an $\uparrow$-expression, we must have $B_\uparrow(X) \geq B_\downarrow(X)$. Indeed, Theorem 7.24(1) applies to X.

   By Claim 1, $\mathcal{P} = Y_0 X_{i_1} Y_1 X_{i_2} Y_2 \dots X_{i_r} Y_r$ is a lower block partitioning of X.

   Consider any argument ε_{i_j} with $1 \leq j \leq r$. By definition, ε_{i_j} is a $\downarrow$-expression denoting X_{i_j}. In particular, because E is minimal, ε_{i_j} is a *minimal* $\downarrow$-expression denoting X_{i_j}.

   When we define indices a_j and b_j for $j = 0, \dots, r$ by

   $$\begin{array}{ll} a_0 = 1, & \\ a_j = i_j + 1 & (j = 1, \dots, r), \\ b_j = i_{j+1} - 1 & (j = 0, \dots, r-1), \text{ and} \\ b_r = n, & \end{array}$$

   it is easy to verify that for $j = 0, \dots, r$, $Y_j = X_{a_j} \dots X_{b_j}$. Indeed, each of the X_i's in such a sequence is a component of X.

Consider any argument ε_i of E such that $a_j \leq i \leq b_j$ for some j with $0 \leq j \leq r$, in other words: any ε_i that is not a $\downarrow$-expression. As we observed before, ε_i is either a maximal $\mathcal{N}$-word occurrence α (in which case $X_i = \mathcal{S}^+(\varepsilon_i) = \binom{\alpha}{_}$), or an $\updownarrow$-expression $\langle \updownarrow \alpha \rangle$ for an $\mathcal{N}$-word α (in which case $X_i = \binom{\alpha}{c(\alpha)}$). Hence, ε_i satisfies equation (7.2) from Theorem 7.24(1).

Finally, by the definition of the a_j's and the b_j's,

$$E = \langle \uparrow \varepsilon_{a_0} \dots \varepsilon_{b_0} \quad \varepsilon_{i_1} \quad \varepsilon_{a_1} \dots \varepsilon_{b_1} \dots \quad \varepsilon_{i_r} \quad \varepsilon_{a_r} \dots \varepsilon_{b_r} \rangle ,$$

where we added white space before and after the arguments ε_{i_j} to emphasize the correspondence with the arguments E_j in equation (7.3) from Theorem 7.24(1).

□

As all auxiliary results in this section deal with operator-minimal $\uparrow$-expressions, it is a small step from Theorem 8.13 to a characterization of the operator-minimal $\uparrow$-expressions denoting a nick free formal DNA molecule. In this characterization, we only refer to the construction from Theorem 7.42 instead of the one from Theorem 7.24(1).

Theorem 8.14 *Let $E = \langle \uparrow \varepsilon_1 \dots \varepsilon_n \rangle$, where $n \geq 1$ and $\varepsilon_1, \dots, \varepsilon_n$ are maximal $\mathcal{N}$-word occurrences and DNA expressions, be an operator-minimal DNA expression denoting a certain nick free formal DNA molecule X. For $i = 1, \dots, n$, let $X_i = \mathcal{S}^+(\varepsilon_i)$.*

Let $\varepsilon_{i_1}, \varepsilon_{i_2}, \dots, \varepsilon_{i_r}$, with $0 \leq r \leq n$ and $i_1 < i_2 < \dots < i_r$, be all $\downarrow$-arguments of E. Finally, let $Y_0, Y_1, \dots, Y_r$ be defined by

$$\begin{aligned} Y_0 &= \begin{cases} X_1 \dots X_n & \text{if } r = 0 \\ X_1 \dots X_{i_1 - 1} & \text{if } r \geq 1 \end{cases} \\ Y_j &= X_{i_j + 1} \dots X_{i_{j+1} - 1} \qquad (j = 1, \dots, r-1) \\ Y_r &= \begin{cases} X_1 \dots X_n & \text{if } r = 0 \\ X_{i_r + 1} \dots X_n & \text{if } r \geq 1 \end{cases} \end{aligned}$$

1. *$\mathcal{P} = Y_0 X_{i_1} Y_1 X_{i_2} Y_2 \dots X_{i_r} Y_r$ is a lower block partitioning of X.*

2. *E satisfies the description of an operator-minimal DNA expression denoting X and based on $\mathcal{P}$, given in Theorem 7.42.*

Proof: The proof of Claim 1 is identical to that of Theorem 8.13(1).

The proof of Claim 2 is a bit shorter than that of Theorem 8.13(2). In order to demonstrate that Theorem 7.42 is applicable to X (rather than Theorem 7.24(1)), we do not have to elaborate on single-stranded components occurring in X, nor on $B_\uparrow(X)$ and $B_\downarrow(X)$. We only have to observe that X is nick free. When we read "because E is operator-minimal" instead of "because E is minimal", the rest of the proof is identical. □

We finally characterize the minimal $\uparrow$-expressions for a formal DNA molecule X containing (lower) nick letters. The characterization in fact reduces to a characterization of operator-minimal $\uparrow$-expressions denoting nick free formal DNA submolecules of X, which is provided by Theorem 8.14 above.

Theorem 8.15 *Let $E = \langle\uparrow \varepsilon_1 \dots \varepsilon_n\rangle$, where $n \geq 1$ and $\varepsilon_1, \dots, \varepsilon_n$ are maximal $\mathcal{N}$-word occurrences and DNA expressions, be a minimal DNA expression denoting a certain formal DNA molecule X which contains at least one nick letter. Let $Z_1 {}_\triangle Z_2 {}_\triangle \dots {}_\triangle Z_m$ for some $m \geq 2$ be the nick free decomposition of X.*

Then E satisfies the description of a minimal DNA expression denoting X given in Theorem 7.46. Hence, there exist indices $i_0, i_1, \dots, i_m$, such that

- *$i_0 = 0 < i_1 < i_2 < \dots < i_{m-1} < i_m = n$, and*
- *for $h = 1, \dots, m$, $\left\langle\uparrow \varepsilon_{i_{h-1}+1} \dots \varepsilon_{i_h}\right\rangle$ is an operator-minimal $\uparrow$-expression denoting Z_h.*

Note that the indices $i_0, i_1, \dots, i_m$ are unique. Suppose that $j_0, j_1, \dots, j_m$ is another sequence of indices that satisfies the conditions, and that h_0 is the smallest value for which $i_{h_0} \neq j_{h_0}$. As $i_0 = j_0 = 0$, $h_0 \geq 1$. Then, however, both $\left\langle\uparrow \varepsilon_{i_{h_0-1}+1} \dots \varepsilon_{i_{h_0}}\right\rangle$ and $\left\langle\uparrow \varepsilon_{j_{h_0-1}+1} \dots \varepsilon_{j_{h_0}}\right\rangle = \left\langle\uparrow \varepsilon_{i_{h_0-1}+1} \dots \varepsilon_{j_{h_0}}\right\rangle$ would denote Z_{h_0}. This is impossible, because by Lemma 4.12, each argument ε_i contains at least one $\mathcal{N}$-word α, and thus contributes at least an $\mathcal{A}$-word to the semantics.

Proof: By assumption, X contains at least one nick letter, and by Lemma 5.1(1), X does not contain upper nick letters. Hence, indeed $m \geq 2$ and each nick letter occurring in the nick free decomposition of X is a lower nick letter.

For $i = 1, \dots, n$, let $X_i = \mathcal{S}^+(\varepsilon_i)$. By Lemma 8.7,

$$X = X_1 y_1 X_2 y_2 \dots y_{n-1} X_n, \tag{8.7}$$

where for $i = 1, \dots, n-1$, y_i is either equal to $\triangle$ or to λ, depending on $R(X_i)$ and $L(X_{i+1})$.

On the other hand, we have

$$X = Z_1 {}_\triangle Z_2 {}_\triangle \dots {}_\triangle Z_m.$$

Because by Lemma 8.4, the X_i's themselves are nick free, the lower nick letters occurring in the nick free decomposition of X do not occur in them. Each of the lower nick letters must correspond to a y_i in (8.7).

More formally, there exist indices $i_1, i_2, \dots, i_{m-1}$ such that

- $1 \leq i_1 < i_2 < \dots < i_{m-1} \leq n-1$,
- for $h = 1, \dots, m-1$, $y_{i_h} = \triangle$ and the occurrence

  $$(X_1 y_1 \dots y_{i_h - 1} X_{i_h}, X_{i_h+1} y_{i_h+1} \dots y_{n-1} X_n) \tag{8.8}$$

  of y_{i_h} in X is equal to the occurrence

  $$(Z_1 {}_\triangle \dots {}_\triangle Z_h, Z_{h+1} {}_\triangle \dots {}_\triangle Z_m) \tag{8.9}$$

  of $\triangle$ in X, and
- for $i \in \{1, \dots, n-1\} \setminus \{i_1, \dots, i_{m-1}\}$, $y_i = \lambda$.

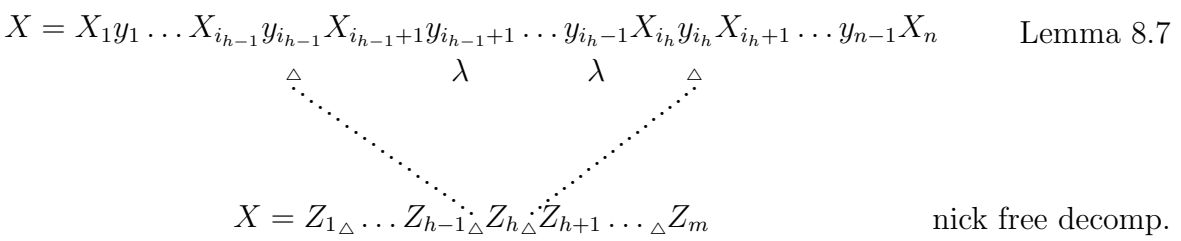


Figure 8.1: Corresponding lower nick letters in two descriptions of a formal DNA molecule denoted by a minimal ↑-expression (see the proof of Theorem 8.15).

This formal description is illustrated by Figure 8.1.

Let us define $i_0 = 0$ and $i_m = n$, and consider any h with $1 \leq h \leq m$. It is clear from (8.8) and (8.9) that

$$Z_h = X_{i_{h-1}+1} y_{i_{h-1}+1} \dots y_{i_h-1} X_{i_h}, \tag{8.10}$$

which is equal to $X_{i_{h-1}+1} X_{i_{h-1}+2} \dots X_{i_h}$, because every y_i with $i_{h-1} + 1 \leq i \leq i_h - 1$ is equal to λ. Note that by Lemma 7.38(1), $Z_h \neq \lambda$. Indeed, because $i_{h-1} < i_h$, $X_{i_{h-1}+1} X_{i_{h-1}+2} \dots X_{i_h} \neq \lambda$.

Now, consider the ↑-expression $E_h = \langle \uparrow \varepsilon_{i_{h-1}+1} \dots \varepsilon_{i_h} \rangle$. By definition,

$$\mathcal{S}(E_h) = \nu^+(X_{i_{h-1}+1}) y_{i_{h-1}+1} \dots y_{i_h-1} \nu^+(X_{i_h}), \tag{8.11}$$

where $y_{i_{h-1}+1}, \dots, y_{i_h-1}$ are the same as in, e.g., (8.10). That is, each of them is empty. Because all X_i's are nick free, (8.11) reduces to

$$\mathcal{S}(E_h) = X_{i_{h-1}+1} X_{i_{h-1}+2} \dots X_{i_h} = Z_h.$$

Hence, E_h is an ↑-expression denoting Z_h. Suppose that E_h is not operator-minimal, i.e., that there exists an ↑-expression $E'_h = \langle \uparrow \varepsilon_{h,1} \dots \varepsilon_{h,n_h} \rangle$ for some $n_h \geq 1$ and $\mathcal{N}$-words and DNA expressions $\varepsilon_{h,1}, \dots, \varepsilon_{h,n_h}$, such that $\mathcal{S}(E'_h) = \mathcal{S}(E_h) = Z_h$ and $|E'_h| < |E_h|$. Then apparently,

$$|\varepsilon_{h,1} \dots \varepsilon_{h,n_h}| < |\varepsilon_{i_{h-1}+1} \dots \varepsilon_{i_h}| \tag{8.12}$$

and

$$\begin{aligned} E &= \langle \uparrow \varepsilon_1 \dots \varepsilon_n \rangle \\ &\equiv \left\langle \uparrow \varepsilon_1 \dots \varepsilon_{i_{h-1}} \left\langle \uparrow \varepsilon_{i_{h-1}+1} \dots \varepsilon_{i_h} \right\rangle \varepsilon_{i_h+1} \dots \varepsilon_n \right\rangle \\ &\equiv \left\langle \uparrow \varepsilon_1 \dots \varepsilon_{i_{h-1}} \langle \uparrow \varepsilon_{h,1} \dots \varepsilon_{h,n_h} \rangle \varepsilon_{i_h+1} \dots \varepsilon_n \right\rangle \\ &\equiv \left\langle \uparrow \varepsilon_1 \dots \varepsilon_{i_{h-1}} \varepsilon_{h,1} \dots \varepsilon_{h,n_h} \varepsilon_{i_h+1} \dots \varepsilon_n \right\rangle. \end{aligned}$$

The second equivalence in this derivation is valid by Lemma 5.11, the other two by Lemma 5.10. Because of (8.12), the resulting ↑-expression $\langle \uparrow \varepsilon_1 \dots \varepsilon_{i_{h-1}} \varepsilon_{h,1} \dots \varepsilon_{h,n_h} \varepsilon_{i_h+1} \dots \varepsilon_n \rangle$ is shorter than E. This, however, contradicts the fact that E is minimal. Consequently, E_h must be operator-minimal. □

For each type of expressible formal DNA molecule X, we have described how to construct a minimal DNA expression denoting X and what the length of this DNA expression is. We have also demonstrated that there do not exist minimal DNA expressions for X other than the ones satisfying the description. We give a short overview of the results:

Summary 8.16 *Let X be an expressible formal DNA molecule.*

1. *If $X = \binom{\alpha_1}{c(\alpha_1)}$ for an $\mathcal{N}$-word α_1, then the only minimal DNA expression denoting X is $E = \langle \updownarrow \alpha_1 \rangle$ (see Theorem 7.5).*

   The length of this minimal DNA expression is

   $$|E| = 3 \cdot n_{\updownarrow}(X) + |X|_{\mathcal{A}}.$$

2. *If X is nick free, contains at least one single-stranded component and $B_{\uparrow}(X) = B_{\downarrow}(X)$, then the only minimal DNA expressions denoting X are $\uparrow$-expressions based on a lower block partitioning of X as described in Theorem 7.24(1), and $\downarrow$-expressions based on an upper block partitioning of X as described in Theorem 7.24(2) (see also Theorem 8.13).*

   The length of a minimal DNA expression E is

   $$\begin{aligned} |E| &= 3 + 3 \cdot B_{\downarrow}(X) + 3 \cdot n_{\updownarrow}(X) + |X|_{\mathcal{A}} \\ &= 3 + 3 \cdot B_{\uparrow}(X) + 3 \cdot n_{\updownarrow}(X) + |X|_{\mathcal{A}}. \end{aligned}$$

3. *If X is nick free and $B_{\uparrow}(X) > B_{\downarrow}(X)$, then the only minimal DNA expressions denoting X are $\uparrow$-expressions based on a lower block partitioning of X, as described in Theorem 7.24(1) (see also Theorem 8.13).*

   The length of a minimal DNA expression E is

   $$|E| = 3 + 3 \cdot B_{\downarrow}(X) + 3 \cdot n_{\updownarrow}(X) + |X|_{\mathcal{A}}.$$

4. *If X is nick free and $B_{\downarrow}(X) > B_{\uparrow}(X)$, then the only minimal DNA expressions denoting X are $\downarrow$-expressions based on an upper block partitioning of X, as described in Theorem 7.24(2) (see also Theorem 8.13).*

   The length of a minimal DNA expression E is

   $$|E| = 3 + 3 \cdot B_{\uparrow}(X) + 3 \cdot n_{\updownarrow}(X) + |X|_{\mathcal{A}}.$$

5. *If X contains at least one lower nick letter, then the only minimal DNA expressions denoting X are $\uparrow$-expressions based on operator-minimal $\uparrow$-expressions for the formal DNA submolecules $Z_1, Z_2, \ldots, Z_m$ occurring in the nick free decomposition $Z_1 {}_{\triangle} Z_2 {}_{\triangle} \ldots {}_{\triangle} Z_m$ of X, as described in Theorem 7.46 (see also Theorem 8.15).*

   The operator-minimal $\uparrow$-expressions denoting a (nick free) formal DNA submolecule Z_h are in turn based on a lower block partitioning of Z_h, as described in Theorem 7.42 (see also Theorem 8.14).

   The length of a minimal DNA expression E denoting X is

   $$|E| = 3 + 3 \cdot B_{\downarrow}(X) + 3 \cdot n_{\updownarrow}(X) + |X|_{\mathcal{A}}.$$

6. *If X contains at least one upper nick letter, then the only minimal DNA expressions denoting X are $\downarrow$-expressions based on operator-minimal $\downarrow$-expressions for the formal DNA submolecules $Z_1, Z_2, \ldots, Z_m$ occurring in the nick free decomposition $Z_1{}^{\triangledown}Z_2{}^{\triangledown}\ldots{}^{\triangledown}Z_m$ of X, analogous to the description in Theorem 7.46 (see also Theorem 8.15).*

   The operator-minimal $\downarrow$-expressions denoting a (nick free) formal DNA submolecule Z_h are in turn based on an upper block partitioning of Z_h, analogous to the description in Theorem 7.42 (see also Theorem 8.14).

   The length of a minimal DNA expression E denoting X is

$$|E| = 3 + 3 \cdot B_{\uparrow}(X) + 3 \cdot n_{\updownarrow}(X) + |X|_{\mathcal{A}}.$$

In each of the cases, a minimal DNA expression achieves the applicable lower bound on its length from Theorem 6.31.

Now, we can also say more about the numbers of operators occurring in a minimal or operator-minimal DNA expression.

A minimal $\updownarrow$-expression contains only one occurrence of an operator, the operator $\updownarrow$. We observe once more that minimal $\uparrow$-expressions and $\downarrow$-expressions achieve the lower bounds on their lengths from Theorem 6.31(1) and (2), respectively.

Next, consider an operator-minimal $\uparrow$-expression E denoting a formal DNA molecule X. If X is nick free, then by Theorem 8.14, E can be obtained by the construction from Theorem 7.42. This implies in particular that E achieves the lower bound from Theorem 6.31(1). If X contains (lower) nick letters, then E is minimal and thus also achieves the lower bound. Analogously, each operator-minimal $\downarrow$-expression achieves the lower bound from Theorem 6.31(2).

Because the lower bounds in Theorem 6.31(1) and (2) followed immediately from Theorem 6.30, we have

Corollary 8.17 *Let E be a DNA expression, and let $X = \mathcal{S}(E)$.*

1. *If E is an operator-minimal $\uparrow$-expression, then*

$$\begin{aligned} \#_{\uparrow,\downarrow}(E) &= 1 + B_{\downarrow}(X) \quad \textit{and} \\ \#_{\updownarrow}(E) &= n_{\updownarrow}(X). \end{aligned}$$

2. *If E is an operator-minimal $\downarrow$-expression, then*

$$\begin{aligned} \#_{\uparrow,\downarrow}(E) &= 1 + B_{\uparrow}(X) \quad \textit{and} \\ \#_{\updownarrow}(E) &= n_{\updownarrow}(X). \end{aligned}$$

3. *If E is a minimal $\updownarrow$-expression, then*

$$\begin{aligned} \#_{\uparrow,\downarrow}(E) &= 0 \quad \textit{and} \\ \#_{\updownarrow}(E) &= 1. \end{aligned}$$

Because a minimal DNA expression is certainly operator-minimal, Claims 1 and 2 apply in particular to minimal $\uparrow$-expressions and minimal $\downarrow$-expressions, respectively.

We conclude this section with two examples of types of formal DNA molecules for which the minimal DNA expressions are unique.

Lemma 8.18 *Let X be an expressible formal DNA molecule.*

1. *If X is nick free, contains at least one upper component and does not contain any lower component, then the only minimal DNA expression denoting X is an $\uparrow$-expression whose arguments are maximal $\mathcal{N}$-word occurrences α_i and $\updownarrow$-expressions $\langle \updownarrow \alpha_i \rangle$ for $\mathcal{N}$-words α_i, alternately.*

2. *If X does not contain any single-stranded component and contains at least one lower nick letter, then let $\binom{\alpha_1}{c(\alpha_1)} \triangle \binom{\alpha_2}{c(\alpha_2)} \triangle \ldots \triangle \binom{\alpha_m}{c(\alpha_m)}$ for some $m \geq 2$ and $\mathcal{N}$-words $\alpha_1, \alpha_2, \ldots, \alpha_m$ be the nick free decomposition of X. The only minimal DNA expression denoting X is*

$$E = \langle \uparrow \langle \updownarrow \alpha_1 \rangle \langle \updownarrow \alpha_2 \rangle \ldots \langle \updownarrow \alpha_m \rangle \rangle .$$

Of course, there is an analogous result about nick free molecules with at least one lower component and without upper components, and about molecules without single-stranded components and with upper nick letters. For these types of molecules, the unique minimal DNA expressions are $\downarrow$-expressions.

Note that by Lemma 6.5(2), the formal DNA molecule from Claim 1 is a primitive upper block of itself.

Note also that for the formal DNA molecule X from Claim 2, $B_{\downarrow}(X) = 0$ and $n_{\updownarrow}(X) = m$. Indeed, the minimal $\uparrow$-expression E for that case has length

$$|E| = 3 + 3 \cdot m + |\alpha_1 \alpha_2 \ldots \alpha_m| = 3 + 3 \cdot B_{\downarrow}(X) + 3 \cdot n_{\updownarrow}(X) + |X|_{\mathcal{A}}.$$

Proof:

1. Assume that X is nick free, contains at least one upper component and does not contain any lower component.

   By Lemma 6.13(1), $B_{\uparrow}(X) > 0$ and $B_{\downarrow}(X) = 0$. By Summary 8.16(3), each minimal DNA expression denoting X is an $\uparrow$-expression based on a lower block partitioning $\mathcal{P}$ of X, as described in Theorem 7.24(1). By Lemma 7.19, there is only one lower block partitioning of X: $\mathcal{P} = Y_0 = X$. Now, Theorem 7.24(1) specifies one minimal $\uparrow$-expression, whose arguments are $\mathcal{N}$-words α_i and $\updownarrow$-expressions $\langle \updownarrow \alpha_i \rangle$ for $\mathcal{N}$-words α_i. By Lemma 7.27, these types of arguments alternate.

2. Assume that X does not contain any single-stranded component and contains at least one lower nick letter.

   By Corollary 5.6, the nick free decomposition of X satisfies the description in the claim. By Summary 8.16(5), each minimal DNA expression denoting X is based on operator-minimal $\uparrow$-expressions for $Z_1 = \binom{\alpha_1}{c(\alpha_1)}, \ldots, Z_m = \binom{\alpha_m}{c(\alpha_m)}$, as described in Theorem 7.46. For $h = 1, \ldots, m$, an operator minimal $\uparrow$-expression denoting $Z_h = \binom{\alpha_h}{c(\alpha_h)}$ is in turn based on a lower block partitioning $\mathcal{P}_h$ of Z_h, as described in Theorem 7.42. Because Z_h does not contain any lower component, the only lower

block partitioning of Z_h is $\mathcal{P}_h = Z_h = \binom{\alpha_h}{c(\alpha_h)}$. Hence, the only operator-minimal $\uparrow$-expression denoting Z_h we can obtain by the construction from Theorem 7.42 is $\langle\uparrow \langle\updownarrow \alpha_h\rangle\rangle$. Now, Theorem 7.46 specifies one minimal DNA expression denoting X:

$$E = \langle\uparrow \langle\updownarrow \alpha_1\rangle \langle\updownarrow \alpha_2\rangle \dots \langle\updownarrow \alpha_m\rangle\rangle .$$

□

8.2 Operator-minimal $\updownarrow$-expressions

In earlier sections, we have not explicitly paid attention to operator-minimal $\updownarrow$-expressions. The reason for that is that we do not need them to construct (other) minimal DNA expressions.

In fact, we did consider one type of operator-minimal $\updownarrow$-expressions, namely the minimal $\updownarrow$-expressions $\langle\updownarrow \alpha_1\rangle$ for $\mathcal{N}$-words α_1, see Theorem 7.5. For the sake of completeness, we will now study other operator-minimal $\updownarrow$-expressions.

By Corollary 5.7, if E is an $\updownarrow$-expression and $X = \mathcal{S}(E)$, then there exist $m \geq 1$ $\mathcal{N}$-words $\alpha_1, \dots, \alpha_m$, and a nick letter $y \in \{^{\triangledown}, _{\triangle}\}$, such that $X = \binom{\alpha_1}{c(\alpha_1)} y \binom{\alpha_2}{c(\alpha_2)} y \dots y \binom{\alpha_m}{c(\alpha_m)}$. X does not contain any single-stranded component.

If $m = 1$, then $X = \binom{\alpha_1}{c(\alpha_1)}$. This is the type of molecules we dealt with in Theorem 7.5. Hence, for such molecules, we already know the (unique) (operator-)minimal $\updownarrow$-expressions.

From now on, we assume that $m \geq 2$. Without loss of generality, we assume that the nick letters y occurring in X are lower nick letters. We first describe how to construct *an* operator-minimal $\updownarrow$-expression denoting X.

Theorem 8.19 *Let X be an expressible formal DNA molecule which does not contain any single-stranded component and contains at least one lower nick letter.*

- *Let $\binom{\alpha_1}{c(\alpha_1)}_{\triangle}\binom{\alpha_2}{c(\alpha_2)}_{\triangle} \dots {}_{\triangle}\binom{\alpha_m}{c(\alpha_m)}$ for some $m \geq 2$ and $\mathcal{N}$-words $\alpha_1, \alpha_2, \dots, \alpha_m$ be the nick free decomposition of X;*
- *let α'_1 be a (possibly empty) string over $\mathcal{N}$, and let α''_1 be an $\mathcal{N}$-word, such that $\alpha_1 = \alpha'_1\alpha''_1$;*
- *let α'_m be an $\mathcal{N}$-word, and let α''_m be a (possibly empty) string over $\mathcal{N}$, such that $\alpha_m = \alpha'_m\alpha''_m$; and*
- *let*

$$E = \langle\updownarrow \langle\uparrow \alpha'_1 \langle\updownarrow \alpha''_1\rangle \langle\updownarrow \alpha_2\rangle \langle\updownarrow \alpha_3\rangle \dots \langle\updownarrow \alpha_{m-1}\rangle \langle\updownarrow \alpha'_m\rangle \alpha''_m\rangle\rangle .$$

Then

(a) all ingredients needed to construct E (i.e., the nick free decomposition, the strings α'_1 and α''_m, and the $\mathcal{N}$-words α''_1 and α'_m) are well defined, and

(b) E is an operator-minimal $\updownarrow$-expression denoting X and

$$|E| = 6 + 3 \cdot m + |X|_{\mathcal{A}} = 6 + 3 \cdot n_{\updownarrow}(X) + |X|_{\mathcal{A}}. \tag{8.13}$$

By Lemma 6.5(2) and Lemma 6.7(1), X is the only primitive $\uparrow$-block of itself and X does not contain any primitive $\downarrow$-block. Hence, $B_{\uparrow}(X) = 1$ and $B_{\downarrow}(X) = 0$, and we can rewrite (8.13) as follows:

$$\begin{aligned} |E| &= 3 + 3 \cdot B_{\uparrow}(X) + 3 \cdot n_{\updownarrow}(X) + |X|_{\mathcal{A}} \\ &> 3 + 3 \cdot B_{\downarrow}(X) + 3 \cdot n_{\updownarrow}(X) + |X|_{\mathcal{A}} \\ &= 3 + 3 \cdot n_{\updownarrow}(X) + |X|_{\mathcal{A}}. \end{aligned}$$

This implies that, unlike the (operator-)minimal DNA expressions we have constructed before, the operator-minimal ↕-expression E we construct here does not achieve the applicable lower bounds from Theorem 6.31. Apparently, these lower bounds are not tight for ↕-expressions denoting formal DNA molecules with (lower) nick letters.

Proof: Indeed, by Corollary 5.6, the nick free decomposition of X satisfies the description in the claim.

Obviously, if we let $\alpha'_1 = \lambda$ and $\alpha''_1 = \alpha_1$, then we have a string α'_1 and an $\mathcal{N}$-word α''_1 for which $\alpha_1 = \alpha'_1\alpha''_1$. Hence, there exists at least one pair of strings α'_1 and α''_1 that satisfies this equality. If $|\alpha_1| \geq 2$, then we can also take an arbitrary partitioning of α_1 into $\mathcal{N}$-words α'_1 and α''_1.

Analogously, there exists at least one combination of an $\mathcal{N}$-word α'_m and a string α''_m, such that $\alpha_m = \alpha'_m\alpha''_m$.

Let

$$E_1 = \langle \uparrow \alpha'_1 \langle \updownarrow \alpha''_1 \rangle \langle \updownarrow \alpha_2 \rangle \langle \updownarrow \alpha_3 \rangle \ldots \langle \updownarrow \alpha_{m-1} \rangle \langle \updownarrow \alpha'_m \rangle \alpha''_m \rangle$$

be the argument of E. For notational convenience, let us assume that both $\alpha'_1 \neq \lambda$ and $\alpha''_m \neq \lambda$.[1] By definition, $\mathcal{S}^+(\alpha'_1) = \binom{\alpha'_1}{-}$, $\mathcal{S}^+(\langle \updownarrow \alpha''_1 \rangle) = \binom{\alpha''_1}{c(\alpha''_1)}$, $\mathcal{S}^+(\langle \updownarrow \alpha'_m \rangle) = \binom{\alpha'_m}{c(\alpha'_m)}$, $\mathcal{S}^+(\alpha''_m) = \binom{\alpha''_m}{-}$ and for $h = 2, \ldots, m-1$, $\mathcal{S}^+(\langle \updownarrow \alpha_h \rangle) = \binom{\alpha_h}{c(\alpha_h)}$. Hence, the arguments of the operator $\uparrow$ occurring in E_1 fit together by upper strands. Indeed, E_1 is a DNA expression, and so is $E = \langle \updownarrow E_1 \rangle$. Because

$$\mathcal{S}(E_1) = \binom{\alpha'_1}{-}\binom{\alpha''_1}{c(\alpha''_1)}_{\triangle}\binom{\alpha_2}{c(\alpha_2)}_{\triangle} \cdots {}_{\triangle}\binom{\alpha_{m-1}}{c(\alpha_{m-1})}_{\triangle}\binom{\alpha'_m}{c(\alpha'_m)}\binom{\alpha''_m}{-}$$

and hence

$$\begin{aligned} \mathcal{S}(E) &= \binom{\alpha'_1}{c(\alpha'_1)}\binom{\alpha''_1}{c(\alpha''_1)}_{\triangle}\binom{\alpha_2}{c(\alpha_2)}_{\triangle} \cdots {}_{\triangle}\binom{\alpha_{m-1}}{c(\alpha_{m-1})}_{\triangle}\binom{\alpha'_m}{c(\alpha'_m)}\binom{\alpha''_m}{c(\alpha''_m)} \\ &= \binom{\alpha_1}{c(\alpha_1)}_{\triangle}\binom{\alpha_2}{c(\alpha_2)}_{\triangle} \cdots {}_{\triangle}\binom{\alpha_{m-1}}{c(\alpha_{m-1})}_{\triangle}\binom{\alpha_m}{c(\alpha_m)}, \end{aligned}$$

the ↕-expression E denotes X.

E contains $m+2$ operators. Hence, by Lemma 6.1, the length $|E|$ of E satisfies (8.13). Now, the only thing left to be proved is that E is operator-minimal, i.e., that there does not exist a shorter ↕-expression denoting X.

By Lemma 8.18(2), the unique minimal DNA expression denoting X is the $\uparrow$-expression

$$E' = \langle \uparrow \langle \updownarrow \alpha_1 \rangle \langle \updownarrow \alpha_2 \rangle \ldots \langle \updownarrow \alpha_m \rangle \rangle,$$

which contains $m+1$ operators and thus has length

$$|E'| = 3 + 3 \cdot m + |X|_{\mathcal{A}} = 3 + 3 \cdot n_{\updownarrow}(X) + |X|_{\mathcal{A}}.$$

[1] If either of these strings is empty, then E_1 has fewer arguments and is actually easier to analyse.

Consequently, any $\updownarrow$-expression denoting X contains at least one operator more and thus is at least three letters longer than E'. This implies that it is at least as long as E. Indeed, E is operator-minimal. □

We now show that the specification of operator-minimal $\updownarrow$-expressions given above is complete. That is, for the formal DNA molecules considered in Theorem 8.19, there do not exist operator-minimal $\updownarrow$-expressions other than the ones specified there.

Theorem 8.20 *Let E be an operator-minimal $\updownarrow$-expression, denoting a certain formal DNA molecule X which contains at least one lower nick letter.*

Then E satisfies the description of an operator-minimal $\updownarrow$-expression denoting X given in Theorem 8.19.

Proof: By the definition of the semantics of an $\updownarrow$-expression, X does not contain any single-stranded component. Indeed, Theorem 8.19 is applicable to X.

By Corollary 5.7, the nick free decomposition of X is

$$\binom{\alpha_1}{c(\alpha_1)} \triangle \binom{\alpha_2}{c(\alpha_2)} \triangle \cdots \triangle \binom{\alpha_m}{c(\alpha_m)}$$

for some $m \geq 2$ and $\mathcal{N}$-words $\alpha_1, \ldots, \alpha_m$.

If $E = \langle \updownarrow \alpha \rangle$ for an $\mathcal{N}$-word α, then $X = \mathcal{S}(E) = \binom{\alpha}{c(\alpha)}$ would not contain any lower nick letter. Hence, $E = \langle \updownarrow E_1 \rangle$ for a DNA expression E_1.

Let $X_1 = \mathcal{S}(E_1)$. By definition, $X = \kappa(X_1)$. Because the function κ does not introduce, nor remove nick letters, X_1 contains $m-1$ lower nick letters, just like X. As E is operator-minimal, its argument E_1 must be minimal. Hence, by Lemma 7.35(1), E_1 is an $\uparrow$-expression, and by Theorem 7.46,

$$|E_1| = 3 + 3 \cdot B_{\downarrow}(X_1) + 3 \cdot n_{\updownarrow}(X_1) + |X_1|_{\mathcal{A}}. \tag{8.14}$$

Let $Z_{1,1} \triangle Z_{1,2} \triangle \cdots \triangle Z_{1,m}$ be the nick free decomposition of X_1. By Theorem 8.15, there exist operator-minimal $\uparrow$-expressions $E_{1,1}, E_{1,2}, \ldots, E_{1,m}$ denoting $Z_{1,1}, Z_{1,2}, \ldots, Z_{1,m}$, respectively, such that $E_1 = \left\langle \uparrow \widehat{E}_{1,1} \widehat{E}_{1,2} \ldots \widehat{E}_{1,m} \right\rangle$, where for $h = 1, \ldots, m$, $\widehat{E}_{1,h}$ is the sequence of the arguments of $E_{1,h}$. In order to determine what exactly each $E_{1,h}$ may be, we first analyse the $Z_{1,h}$'s.

By definition, each lower nick letter occurring in (the formal DNA molecule) X_1 is both preceded and succeeded by a double $\mathcal{A}$-letter. Hence, for $h = 1, \ldots, m$, $Z_{1,h}$ contains at least one double component: $n_{\updownarrow}(Z_{1,h}) \geq 1$. The double components of X_1 are distributed over the $Z_{1,h}$'s. Hence, by Lemma 6.3,

$$n_{\updownarrow}(X_1) = n_{\updownarrow}(Z_{1,1}) + n_{\updownarrow}(Z_{1,2}) + \cdots + n_{\updownarrow}(Z_{1,m}) \geq m. \tag{8.15}$$

When we substitute this into (8.14), we obtain

$$|E_1| \geq 3 + 3 \cdot B_{\downarrow}(X_1) + 3 \cdot m + |X_1|_{\mathcal{A}} \geq 3 + 3 \cdot m + |X_1|_{\mathcal{A}}, \tag{8.16}$$

and hence,

$$|E| = 3 + |E_1| \geq 6 + 3 \cdot m + |X_1|_{\mathcal{A}} = 6 + 3 \cdot m + |X|_{\mathcal{A}}. \tag{8.17}$$

By Theorem 8.19, there exists an operator-minimal $\updownarrow$-expression denoting X with length (exactly) $6 + 3 \cdot m + |X|_{\mathcal{A}}$. Because, obviously, all operator-minimal $\updownarrow$-expressions denoting X have the same length, we must have equality in (8.16) and (8.17):

$$|E_1| = 3 + 3 \cdot m + |X_1|_{\mathcal{A}}$$

and

$$|E| = 6 + 3 \cdot m + |X|_{\mathcal{A}}.$$

Consequently, $B_{\downarrow}(X_1) = 0$, $n_{\updownarrow}(X_1) = m$ and in particular, for $h = 1, \ldots, m$, $n_{\updownarrow}(Z_{1,h}) = 1$. By Lemma 6.7(2), X_1 does not have any $\downarrow$-component. Then certainly, none of the $Z_{1,h}$'s has a $\downarrow$-component. Hence, by definition, for $h = 1, \ldots, m$, $B_{\downarrow}(Z_{1,h}) = 0$.

The relation $X = \kappa(X_1)$ implies that for $h = 1, \ldots, m$, $\binom{\alpha_h}{c(\alpha_h)} = \kappa(Z_{1,h})$. Now, consider any h with $1 \leq h \leq m$.

• If $2 \leq h \leq m-1$, then $Z_{1,h}$ is both preceded and succeeded in X_1 by a lower nick letter. Hence, it both starts and ends with a double component. Because $n_{\updownarrow}(Z_{1,h}) = 1$, these have to be the same double component: $Z_{1,h} = \binom{\alpha_h}{c(\alpha_h)}$, as $\kappa(Z_{1,h}) = \binom{\alpha_h}{c(\alpha_h)}$.

By Lemma 7.19, the only lower block partitioning of $Z_{1,h}$ is $Y_0 = Z_{1,h} = \binom{\alpha_h}{c(\alpha_h)}$. Hence, by Theorem 7.42 and Theorem 8.14, the only operator-minimal $\uparrow$-expression denoting $Z_{1,h}$ is $E_{1,h} = \langle\uparrow \langle\updownarrow \alpha_h\rangle\rangle$.

• If $h = 1$, then $Z_{1,h} = Z_{1,1}$ ends with a double component $\binom{\alpha_1''}{c(\alpha_1'')}$ for an $\mathcal{N}$-word α_1''. Because $n_{\updownarrow}(Z_{1,1}) = 1$, $Z_{1,1}$ does not contain any other double component. Hence, by Corollary 3.8, $\binom{\alpha_1''}{c(\alpha_1'')}$ is preceded in $Z_{1,1}$ by at most one other component, which then is a single-stranded component.

If $\binom{\alpha_1''}{c(\alpha_1'')}$ is not preceded in $Z_{1,1}$ by any other component, then $Z_{1,1} = \binom{\alpha_1''}{c(\alpha_1'')} = \binom{\alpha_1}{c(\alpha_1)}$.

If $\binom{\alpha_1''}{c(\alpha_1'')}$ is preceded in $Z_{1,1}$ by a single-stranded component, then this cannot be a lower component, because $Z_{1,1}$ does not contain $\downarrow$-components. Consequently, $\binom{\alpha_1''}{c(\alpha_1'')}$ is preceded in $Z_{1,1}$ by an upper component $\binom{\alpha_1'}{-}$ for an $\mathcal{N}$-word α_1': $Z_{1,1} = \binom{\alpha_1'}{-}\binom{\alpha_1''}{c(\alpha_1'')}$. Because $\kappa(Z_{1,1}) = \binom{\alpha_1'}{c(\alpha_1')}\binom{\alpha_1''}{c(\alpha_1'')} = \binom{\alpha_1'\alpha_1''}{c(\alpha_1'\alpha_1'')} = \binom{\alpha_1}{c(\alpha_1)}$, $\alpha_1'\alpha_1'' = \alpha_1$. In this case, both α_1' and α_1'' must contain at least one $\mathcal{N}$-letter, which implies that $|\alpha_1| \geq 2$.

We proceed in the same way as for the case $2 \leq h \leq m-1$. By Lemma 7.19, both if $Z_{1,1} = \binom{\alpha_1}{c(\alpha_1)}$ and if $Z_{1,1} = \binom{\alpha_1'}{-}\binom{\alpha_1''}{c(\alpha_1'')}$ for a partitioning (α_1', α_1'') of α_1, the only lower block partitioning of $Z_{1,1}$ is $Y_0 = Z_{1,1}$. Hence, by Theorem 7.42 and Theorem 8.14, there is exactly one operator-minimal $\uparrow$-expression $E_{1,1}$ denoting $Z_{1,1}$: either $E_{1,1} = \langle\uparrow \langle\updownarrow \alpha_1\rangle\rangle$ (if $Z_{1,1} = \binom{\alpha_1}{c(\alpha_1)}$), or $E_{1,1} = \langle\uparrow \alpha_1' \langle\updownarrow \alpha_1''\rangle\rangle$ (if $Z_{1,1} = \binom{\alpha_1'}{-}\binom{\alpha_1''}{c(\alpha_1'')}$ for a partitioning (α_1', α_1'') of α_1). Both types of operator-minimal $\uparrow$-expressions fit in with the format $E_{1,1} = \langle\uparrow \alpha_1' \langle\updownarrow \alpha_1''\rangle\rangle$ for a (possibly empty) string α_1' over $\mathcal{N}$ and an $\mathcal{N}$-word α_1'' with $\alpha_1 = \alpha_1'\alpha_1''$.

• The case $h = m$ is dealt with analogously to the case $h = 1$. After having established that $Z_{1,m}$ begins with a double component, which may be succeeded by an upper component, we find that $E_{1,m} = \langle\uparrow \langle\updownarrow \alpha_m'\rangle \alpha_m''\rangle$ for an $\mathcal{N}$-word α_m' and a (possibly empty) string α_m'' over $\mathcal{N}$ with $\alpha_m = \alpha_m'\alpha_m''$.

We conclude that

$$\begin{aligned} E &= \langle\updownarrow E_1\rangle \\ &= \left\langle\updownarrow \left\langle\uparrow \widehat{E}_{1,1}\widehat{E}_{1,2}\widehat{E}_{1,3}\ldots\widehat{E}_{1,m-1}\widehat{E}_{1,m}\right\rangle\right\rangle \end{aligned}$$

$$= \langle \updownarrow \langle \uparrow \alpha'_1 \langle \updownarrow \alpha''_1 \rangle \langle \updownarrow \alpha_2 \rangle \langle \updownarrow \alpha_3 \rangle \ldots \langle \updownarrow \alpha_{m-1} \rangle \langle \updownarrow \alpha'_m \rangle \alpha''_m \rangle \rangle .$$

Indeed, E satisfies the description from Theorem 8.19. □

By Theorem 8.19 and Theorem 8.20, we do not have much freedom in the construction of an operator-minimal $\updownarrow$-expression for a formal DNA molecule containing lower nick letters. In particular, the operators occurring in it are fixed. We thus have, as an addition to Corollary 8.17(3),

Corollary 8.21 *Let E be a DNA expression, and let $X = \mathcal{S}(E)$.*

1. *If X contains at least one lower nick letter and E is an operator-minimal $\updownarrow$-expression, then*

$$\begin{aligned} \#_{\uparrow}(E) &= 1, \\ \#_{\downarrow}(E) &= 0 \quad \textit{and} \\ \#_{\updownarrow}(E) &= n_{\updownarrow}(X) + 1. \end{aligned}$$

2. *If X contains at least one upper nick letter and E is an operator-minimal $\updownarrow$-expression, then*

$$\begin{aligned} \#_{\uparrow}(E) &= 0, \\ \#_{\downarrow}(E) &= 1 \quad \textit{and} \\ \#_{\updownarrow}(E) &= n_{\updownarrow}(X) + 1. \end{aligned}$$

8.3 Characterization of minimal DNA expressions

In Chapter 7, we have learned how to construct a minimal DNA expression denoting an arbitrary (expressible) formal DNA molecule. We also observed that we can calculate the length of a minimal DNA expression directly from the formal DNA molecule. We do not have to explicitly construct a minimal DNA expression for this.

Now suppose that we are given a DNA expression E and that we want to decide whether or not it is minimal. Then we can use the following three-stage approach: we first determine the semantics $\mathcal{S}(E)$ of E, we subsequently calculate the length of a minimal DNA expression denoting $\mathcal{S}(E)$, and we finally count the length of E and check if it is equal to this minimal length.

There is also another, direct method. This method is based on a characterization of minimal DNA expressions by six properties of (the arguments of) the operators occurring in them. When we want to know if a given DNA expression E is minimal, we simply check if E has all the properties. If so, then it is minimal; if not, then it is not. We do not need the semantics for this.

In this section, we describe this characterization. We first prove that the six properties are necessary for a DNA expression to be minimal. That is, each minimal DNA expression automatically has these properties. In Lemma 8.25, we will prove that the properties are also sufficient.

Lemma 8.22 *Let E be a minimal DNA expression.*

($\mathcal{D}_{\text{Min}}$.1) *Each occurrence of the operator $\updownarrow$ in E has as its argument an $\mathcal{N}$-word α (i.e., not a DNA expression).*

($\mathcal{D}_{\text{Min}}$.2) *No occurrence of the operator $\uparrow$ in E has an $\uparrow$-argument, and no occurrence of the operator $\downarrow$ in E has a $\downarrow$-argument.*

($\mathcal{D}_{\text{Min}}$.3) *Unless $E = \langle\uparrow \alpha\rangle$ or $E = \langle\downarrow \alpha\rangle$ for an $\mathcal{N}$-word α, each occurrence of an operator $\uparrow$ or $\downarrow$ in E has at least two arguments.*

($\mathcal{D}_{\text{Min}}$.4) *Each inner occurrence of an operator $\uparrow$ or $\downarrow$ in E is alternating.*

($\mathcal{D}_{\text{Min}}$.5) *For each inner occurrence of an operator $\uparrow$ or $\downarrow$ in E,*

- *the first argument is either an $\mathcal{N}$-word α or an $\updownarrow$-expression $\langle\updownarrow \alpha\rangle$ for an $\mathcal{N}$-word α, and*
- *the last argument is either an $\mathcal{N}$-word α or an $\updownarrow$-expression $\langle\updownarrow \alpha\rangle$ for an $\mathcal{N}$-word α.*

($\mathcal{D}_{\text{Min}}$.6) *If the outermost operator of E is $\uparrow$ or $\downarrow$, then*

- *either its first argument is an $\mathcal{N}$-word α or an $\updownarrow$-expression $\langle\updownarrow \alpha\rangle$ for an $\mathcal{N}$-word α,*
- *or its last argument is an $\mathcal{N}$-word α or an $\updownarrow$-expression $\langle\updownarrow \alpha\rangle$ for an $\mathcal{N}$-word α,*
- *or it has two consecutive expression-arguments.*

As we will see in the proof, Properties ($\mathcal{D}_{\text{Min}}$.1) and ($\mathcal{D}_{\text{Min}}$.2) follow immediately from earlier results on (operator-)minimal DNA expressions. In addition to Property ($\mathcal{D}_{\text{Min}}$.2), no occurrence of $\updownarrow$ has an $\updownarrow$-argument in a minimal DNA expression. We have not included this in Property ($\mathcal{D}_{\text{Min}}$.2), as it already follows from Property ($\mathcal{D}_{\text{Min}}$.1).

Intuitively, Property ($\mathcal{D}_{\text{Min}}$.3) means that the effect of an operator $\uparrow$ or $\downarrow$ with a single argument is often too small to justify the presence of the operator. In particular such an operator cannot express its ability to connect consecutive arguments. Property ($\mathcal{D}_{\text{Min}}$.4) ensures that the arguments of the DNA expression are nick free (see Lemma 5.8). It is not efficient to first introduce nick letters and to later remove them. All nick letters in the formal DNA molecule denoted can be produced by the outermost operator.

Let X^s be the submolecule denoted by a proper $\uparrow$-expression. By Property ($\mathcal{D}_{\text{Min}}$.5) (and Properties ($\mathcal{D}_{\text{Min}}$.3) and ($\mathcal{D}_{\text{Min}}$.4)), both the first and the last single-stranded component of X^s are upper components. Hence, $B_\uparrow(X^s) = B_\downarrow(X^s) + 1$. In fact, the submolecule is an upper block. This justifies the use of the operator $\uparrow$ here. If either the first or the last single-stranded component were a lower component, then it would be more efficient to have this lower component produced by the parent operator of the $\uparrow$-subexpression, which is an occurrence of $\downarrow$.

The *outermost* operator has a weaker property. Assume that the outermost operator is $\uparrow$. Then, by Property ($\mathcal{D}_{\text{Min}}$.6), it is possible that this operator has only one of the two subproperties from Property ($\mathcal{D}_{\text{Min}}$.5) (which is, e.g., the case if the formal DNA molecule X denoted is nick free and $B_\uparrow(X) = B_\downarrow(X)$), or that the operator has two consecutive expression-arguments (which, by Corollary 8.11(2) and (3), is the case if and only if X contains lower nick letters).

The basic DNA expressions $\langle\uparrow \alpha\rangle$, $\langle\downarrow \alpha\rangle$ and $\langle\updownarrow \alpha\rangle$ for $\mathcal{N}$-words α (which are minimal) trivially have all six properties.

Example 8.23 We repeat the minimal DNA expression we constructed in Example 7.47, for a formal DNA molecule containing lower nick letters, see Equation (7.28):

$$\begin{array}{ll} E = \langle\uparrow & \alpha_1 \langle\downarrow \langle\updownarrow \alpha_2\rangle \alpha_3 \langle\updownarrow \alpha_4\rangle\rangle \quad \langle\downarrow \langle\updownarrow \alpha_5\rangle \alpha_6 \langle\updownarrow \alpha_7\rangle\rangle \alpha_8 \langle\downarrow \langle\updownarrow \alpha_9\rangle \alpha_{10} \langle\updownarrow \alpha_{11}\rangle\rangle \\ & \langle\updownarrow \alpha_{12}\rangle \alpha_{13} \langle\updownarrow \alpha_{14}\rangle \alpha_{15} \langle\updownarrow \alpha_{16}\rangle \quad \langle\updownarrow \alpha_{17}\rangle \\ & \langle\downarrow \langle\updownarrow \alpha_{18}\rangle \alpha_{19} \langle\updownarrow \alpha_{20}\rangle\rangle \alpha_{21} \langle\updownarrow \alpha_{22}\rangle \quad \rangle\,. \end{array} \tag{8.18}$$

It is easily verified that indeed, E has the six properties from Lemma 8.22. Although Property ($\mathcal{D}_{\text{Min}}$.5) is only about inner occurrences of $\uparrow$ and $\downarrow$, its two subproperties also happen to be valid for the outermost operator $\uparrow$ of E. In fact, this operator has all three subproperties from Property ($\mathcal{D}_{\text{Min}}$.6). ■

Proof of Lemma 8.22: In the proofs of Properties ($\mathcal{D}_{\text{Min}}$.1), ($\mathcal{D}_{\text{Min}}$.2) and ($\mathcal{D}_{\text{Min}}$.3), we will consider an arbitrary occurrence of a certain operator in E. It is worth noting that in principle, this may be the outermost operator of E. Hence, the DNA subexpression E^s governed by it may be equal to E.

($\mathcal{D}_{\text{Min}}$.1) Consider an arbitrary occurrence of the operator $\updownarrow$ in E and let E^s be the DNA subexpression of E governed by it. Because E is minimal, so is E^s. Now, Theorem 7.5 implies that E^s is of the form $\langle\updownarrow \alpha\rangle$ for an $\mathcal{N}$-word α.

($\mathcal{D}_{\text{Min}}$.2) Consider an arbitrary occurrence of the operator $\uparrow$ in E and let E^s be the DNA subexpression of E governed by it. Because E is minimal, so is E^s. By Lemma 8.1, E^s does not have any $\uparrow$-argument.

Analogously, no occurrence of the operator $\downarrow$ in E has a $\downarrow$-argument.

($\mathcal{D}_{\text{Min}}$.3) Assume that E is not equal to $\langle\uparrow \alpha\rangle$ or $\langle\downarrow \alpha\rangle$ for an $\mathcal{N}$-word α.

Consider an arbitrary occurrence of the operator $\uparrow$ in E and let E^s be the DNA subexpression of E governed by it. Because E is minimal, so is E^s.

Suppose that E^s has only one argument ε_1: $E^s = \langle\uparrow \varepsilon_1\rangle$. By Lemma 8.4, $\mathcal{S}^+(\varepsilon_1)$ is nick free. Now if ε_1 were a DNA expression E_1, then by definition, $\mathcal{S}(E^s) = \mathcal{S}(\langle\uparrow E_1\rangle) = \nu^+(\mathcal{S}(E_1)) = \mathcal{S}(E_1)$. Because E_1 is three letters shorter than E^s, E^s would not be minimal. Consequently, ε_1 must be an $\mathcal{N}$-word α_1: $E^s = \langle\uparrow \alpha_1\rangle$.

Because $E \neq \langle\uparrow \alpha\rangle$ for any $\mathcal{N}$-word α, $E^s \neq E$, i.e., E^s is a proper DNA subexpression of E. By Properties ($\mathcal{D}_{\text{Min}}$.1) and ($\mathcal{D}_{\text{Min}}$.2), the parent operator of (the $\uparrow$-expression) E^s is $\downarrow$. Because $\mathcal{S}(E^s) = \binom{\alpha_1}{_}$ does not fit together by lower strands with any other formal DNA molecule, E^s must be the only argument of its parent operator $\downarrow$. Hence, E has a DNA subexpression $\langle\downarrow E^s\rangle = \langle\downarrow \langle\uparrow \alpha_1\rangle\rangle$. This DNA subexpression is, however, equivalent to the shorter DNA subexpression $\langle\uparrow \alpha_1\rangle$, which contradicts the minimality of E.

Hence, our hypothesis that E^s has only one argument must be wrong.

The proof for an occurrence of the operator $\downarrow$ in E is analogous.

($\mathcal{D}_{\text{Min}}$.4) Consider an arbitrary inner occurrence of the operator $\uparrow$ in E and let E^s be the (proper) DNA subexpression of E governed by it. Clearly, by Property ($\mathcal{D}_{\text{Min}}$.1), E

is not an $\updownarrow$-expression. E^s is minimal and by Corollary 8.5, E^s is nick free. Hence, by Corollary 8.10, E^s is alternating.

The proof for an inner occurrence of the operator $\downarrow$ in E is analogous.

Note that we in fact proved this property already in the proof of Corollary 8.11.

($\mathcal{D}_{\text{Min}}$.5) Consider an arbitrary inner occurrence of the operator $\uparrow$ in E, and let E_1^s be the DNA subexpression of E governed by it. E_1^s is the argument of a DNA subexpression E_0^s of E. Both E_1^s and E_0^s are minimal. Because E_1^s is an $\uparrow$-expression, Properties ($\mathcal{D}_{\text{Min}}$.1) and ($\mathcal{D}_{\text{Min}}$.2) imply that E_0^s is a $\downarrow$-expression.

Let $X_1^s = \mathcal{S}(E_1^s)$. By (the analogue for $\downarrow$-expressions of) Lemma 8.4, X_1^s is nick free. By Theorem 8.13, E_1^s satisfies the construction from Theorem 7.24(1). In particular, X_1^s contains at least one single-stranded component and $B_\uparrow(X) \geq B_\downarrow(X)$. By Lemma 8.6(1), we even have $B_\uparrow(X_1^s) = B_\downarrow(X_1^s) + 1$. Now by Corollary 7.33(1), neither the first argument, nor the last argument of E_1^s is a $\downarrow$-argument. Hence, by Corollary 8.2, each of them is either an $\mathcal{N}$-word α, or an $\updownarrow$-expression $\langle\updownarrow \alpha\rangle$ for an $\mathcal{N}$-word α.

The proof for an inner occurrence of the operator $\downarrow$ in E is analogous.

($\mathcal{D}_{\text{Min}}$.6) Let $X = \mathcal{S}(E)$. Assume that the outermost operator of E is $\uparrow$ and that E does not have two consecutive expression-arguments. By Corollary 8.11(1) and (3), X is nick free.

By Theorem 8.13, E satisfies the construction from Theorem 7.24(1). In particular, X contains at least one single-stranded component and $B_\uparrow(X) \geq B_\downarrow(X)$. Then by Corollary 7.33(3), it is impossible that both the first argument, and the last argument of E are $\downarrow$-arguments. Hence, by Corollary 8.2, either the first argument, or the last argument of E (or both) is an $\mathcal{N}$-word α or an $\updownarrow$-expression $\langle\updownarrow \alpha\rangle$ for an $\mathcal{N}$-word α.

The proof for the case that the outermost operator of E is $\downarrow$ is analogous.

□

In the proof of Properties ($\mathcal{D}_{\text{Min}}$.5) and ($\mathcal{D}_{\text{Min}}$.6) above, there was a crucial role for the construction of minimal $\uparrow$-expressions, as described in Theorem 7.24(1). It is instructive to also consider another proof of these two properties, which directly relates them to the concept of minimality: if a DNA expression does not have these properties, then we can easily find a shorter, equivalent DNA expression.

Alternative proof of Properties ($\mathcal{D}_{\text{Min}}$.5) and ($\mathcal{D}_{\text{Min}}$.6): Let E be a DNA expression with Properties ($\mathcal{D}_{\text{Min}}$.1)–($\mathcal{D}_{\text{Min}}$.4).

($\mathcal{D}_{\text{Min}}$.5) Assume that E does not have Property ($\mathcal{D}_{\text{Min}}$.5). Without loss of generality, assume that there is an inner occurrence $\uparrow_1$ *of the operator* $\uparrow$ in E, such that *the last argument* of $\uparrow_1$ is neither an $\mathcal{N}$-word α, nor an $\updownarrow$-expression $\langle\updownarrow \alpha\rangle$ for an $\mathcal{N}$-word α. We prove that E is not minimal.

Let $E_1^s = \langle\uparrow_1 \varepsilon_1 \ldots \varepsilon_n\rangle$ for some $n \geq 1$ and $\mathcal{N}$-words and DNA expressions $\varepsilon_1, \ldots, \varepsilon_n$ be the DNA subexpression governed by $\uparrow_1$.

E is not equal to $\langle\uparrow \alpha\rangle$ or $\langle\downarrow \alpha\rangle$ for an $\mathcal{N}$-word α, because such DNA expressions do not have inner occurrences of operators. Hence, by Property ($\mathcal{D}_{\text{Min}}$.3), $n \geq 2$.

By Properties ($\mathcal{D}_{\text{Min}}$.1) and ($\mathcal{D}_{\text{Min}}$.2), E_1^s is the argument of a $\downarrow$-expression E_0^s. By the same properties, each argument ε_i of $\uparrow_1$ is either an $\mathcal{N}$-word α, or an $\updownarrow$-expression $\langle\updownarrow \alpha\rangle$ for an $\mathcal{N}$-word α, or a $\downarrow$-expression. Now by assumption, the last argument ε_n must be a $\downarrow$-expression. Hence, E_0^s has the following shape:

$$E_0^s = \langle\downarrow_0 \ldots \langle\uparrow_1 \varepsilon_1 \ldots \varepsilon_{n-1} \langle\downarrow_2 \varepsilon_{n,1} \ldots \varepsilon_{n,m_n}\rangle\rangle \ldots\rangle$$

for some $m_n \geq 1$ and $\mathcal{N}$-words and DNA expressions $\varepsilon_{n,1}, \ldots, \varepsilon_{n,m_n}$. Because the arguments of $\uparrow_1$ must fit together by upper strands, the first argument $\varepsilon_{n,1}$ of $\downarrow_2$ is a DNA expression. Moreover, by Property ($\mathcal{D}_{\text{Min}}$.4), each occurrence of $\uparrow$ or $\downarrow$ in $E_1^s = \langle\uparrow_1 \varepsilon_1 \ldots \varepsilon_{n-1} \langle\downarrow_2 \varepsilon_{n,1} \ldots \varepsilon_{n,m_n}\rangle\rangle$ is alternating.

We show that the effects of $\downarrow_2$ can as well be achieved by the operator $\downarrow_0$, by transferring the last $m_n - 1$ arguments of $\downarrow_2$ to $\downarrow_0$. By Theorem 5.16(1) and (2),

$$\begin{aligned} E_1^s &= \langle\uparrow_1 \varepsilon_1 \ldots \varepsilon_{n-1} \langle\downarrow_2 \varepsilon_{n,1} \ldots \varepsilon_{n,m_n}\rangle\rangle \\ &\equiv \langle\downarrow_2 \langle\uparrow_1 \varepsilon_1 \ldots \varepsilon_{n-1}\varepsilon_{n,1}\rangle \varepsilon_{n,2} \ldots \varepsilon_{n,m_n}\rangle\,. \end{aligned}$$

Then by Lemma 5.11 and Lemma 5.10,

$$\begin{aligned} E_0^s &= \langle\downarrow_0 \ldots \langle\uparrow_1 \varepsilon_1 \ldots \varepsilon_{n-1} \langle\downarrow_2 \varepsilon_{n,1} \ldots \varepsilon_{n,m_n}\rangle\rangle \ldots\rangle \\ &\equiv \langle\downarrow_0 \ldots \langle\downarrow_2 \langle\uparrow_1 \varepsilon_1 \ldots \varepsilon_{n-1}\varepsilon_{n,1}\rangle \varepsilon_{n,2} \ldots \varepsilon_{n,m_n}\rangle \ldots\rangle \\ &\equiv \langle\downarrow_0 \ldots \langle\uparrow_1 \varepsilon_1 \ldots \varepsilon_{n-1}\varepsilon_{n,1}\rangle \varepsilon_{n,2} \ldots \varepsilon_{n,m_n} \ldots\rangle\,. \end{aligned}$$

Let us use $E_0^{s\prime}$ to denote the resulting $\downarrow$-expression. Clearly, $E_0^{s\prime}$ is three letters shorter than E_0^s. When we substitute E_0^s in E by $E_0^{s\prime}$, by Lemma 5.11, the result is a (shorter) DNA expression that is equivalent to E. We conclude that E is not minimal.

($\mathcal{D}_{\text{Min}}$.6) Assume that E does not have Property ($\mathcal{D}_{\text{Min}}$.6). Without loss of generality, assume that the outermost operator of E is $\uparrow$. Hence, $E = \langle\uparrow \varepsilon_1 \ldots \varepsilon_n\rangle$ for some $n \geq 1$ and $\mathcal{N}$-words and DNA expressions $\varepsilon_1, \ldots, \varepsilon_n$. We prove that E is not minimal.

By Properties ($\mathcal{D}_{\text{Min}}$.1) and ($\mathcal{D}_{\text{Min}}$.2), each argument ε_i of E is either an $\mathcal{N}$-word α, or an $\updownarrow$-expression $\langle\updownarrow \alpha\rangle$ for an $\mathcal{N}$-word α, or a $\downarrow$-expression. By assumption, both the first argument and the last argument are $\downarrow$-expressions:

$$\begin{aligned} \varepsilon_1 &= \langle\downarrow \varepsilon_{1,1} \ldots \varepsilon_{1,m_1-1}\varepsilon_{1,m_1}\rangle\,, \text{ and} \\ \varepsilon_n &= \langle\downarrow \varepsilon_{n,1}\varepsilon_{n,2} \ldots \varepsilon_{n,m_n}\rangle \end{aligned}$$

for some $m_1, m_n \geq 1$ and $\mathcal{N}$-words and DNA expressions $\varepsilon_{1,1}, \ldots, \varepsilon_{1,m_1-1}, \varepsilon_{1,m_1}, \varepsilon_{n,1}$, $\varepsilon_{n,2}, \ldots, \varepsilon_{n,m_n}$. By Property ($\mathcal{D}_{\text{Min}}$.3), $n, m_1, m_n \geq 2$.

By assumption, the outermost operator $\uparrow$ does not have two consecutive expression-arguments. Hence, by Property ($\mathcal{D}_{\text{Min}}$.4), each occurrence of $\uparrow$ or $\downarrow$ in E is alternating.

We can apply Theorem 5.21(1) and (2) to E (with $r = 1$):

$$\begin{aligned} E &= \langle\uparrow \langle\downarrow \varepsilon_{1,1} \ldots \varepsilon_{1,m_1-1}\varepsilon_{1,m_1}\rangle \varepsilon_2 \ldots \varepsilon_{n-1} \langle\downarrow \varepsilon_{n,1}\varepsilon_{n,2} \ldots \varepsilon_{n,m_n}\rangle\rangle \\ &\equiv \langle\downarrow \varepsilon_{1,1} \ldots \varepsilon_{1,m_1-1} \langle\uparrow \varepsilon_{1,m_1}\varepsilon_2 \ldots \varepsilon_{n-1}\varepsilon_{n,1}\rangle \varepsilon_{n,2} \ldots \varepsilon_{n,m_n}\rangle\,. \end{aligned}$$

Let us use E' to denote the resulting DNA expression. As two occurrences of the operator $\downarrow$ in E have been replaced by one occurrence in E', E' is shorter than E. Also in this case, E is not minimal.

□

We use $\mathcal{D}_{\text{Min}}$ to denote the set of DNA expressions that have Properties ($\mathcal{D}_{\text{Min}}$.1)–($\mathcal{D}_{\text{Min}}$.6). Membership of $\mathcal{D}_{\text{Min}}$ carries over to DNA subexpressions:

Lemma 8.24 *A DNA expression E is in $\mathcal{D}_{Min}$ if and only if each DNA subexpression of E is in $\mathcal{D}_{Min}$.*

This result is not immediate, as Properties ($\mathcal{D}_{\text{Min}}$.3) and ($\mathcal{D}_{\text{Min}}$.6) are 'non-local' properties of E: Property ($\mathcal{D}_{\text{Min}}$.3) takes into account the *total* DNA expression E and Property ($\mathcal{D}_{\text{Min}}$.6) is a property of the *outermost* operator of E.

Proof: Let E be an arbitrary DNA expression.

Clearly, if each DNA subexpression of E is in $\mathcal{D}_{\text{Min}}$, then so is E, because E is a DNA subexpression of itself.

Now assume that $E \in \mathcal{D}_{\text{Min}}$ and let E^s be a proper DNA subexpression of E. We will prove that E^s has the six properties from Lemma 8.22, and thus is in $\mathcal{D}_{\text{Min}}$.

Each occurrence of an operator in E^s is also an occurrence of that operator in E, with the same arguments. In particular, each inner occurrence of an operator $\uparrow$ or $\downarrow$ in E^s is also an inner occurrence of that operator in E, with the same arguments. Hence, Properties ($\mathcal{D}_{\text{Min}}$.1), ($\mathcal{D}_{\text{Min}}$.2), ($\mathcal{D}_{\text{Min}}$.4) and ($\mathcal{D}_{\text{Min}}$.5) are valid for E^s, simply because they are valid for E.

Because E^s is a *proper* DNA subexpression of E, E cannot be equal to $\langle\uparrow \alpha\rangle$ or $\langle\downarrow \alpha\rangle$ for an $\mathcal{N}$-word α; such DNA expressions do not have proper DNA subexpressions. Hence, by Property ($\mathcal{D}_{\text{Min}}$.3) for E, each occurrence of an operator $\uparrow$ or $\downarrow$ in E has at least two arguments. Of course, the same holds for such an occurrence in E^s, which means that Property ($\mathcal{D}_{\text{Min}}$.3) is also valid for E^s. Note that indeed, E^s cannot be equal to $\langle\uparrow \alpha\rangle$ or $\langle\downarrow \alpha\rangle$ for an $\mathcal{N}$-word α.

If the outermost operator of E^s is $\uparrow$ or $\downarrow$, then it is an inner occurrence of that operator in E. Hence, by Property ($\mathcal{D}_{\text{Min}}$.5), both its first argument is an $\mathcal{N}$-word α or an $\updownarrow$-expression $\langle\updownarrow \alpha\rangle$ for an $\mathcal{N}$-word α, and its last argument is an $\mathcal{N}$-word α or an $\updownarrow$-expression $\langle\updownarrow \alpha\rangle$ for an $\mathcal{N}$-word α. This implies that also Property ($\mathcal{D}_{\text{Min}}$.6) is valid for E^s. Note that by Property ($\mathcal{D}_{\text{Min}}$.4) for E, the outermost operator of E^s cannot have two consecutive expression-arguments. □

We can now prove that all elements of $\mathcal{D}_{\text{Min}}$ are minimal, which means that the six properties are indeed sufficient.

Lemma 8.25 *Each DNA expression $E \in \mathcal{D}_{Min}$ is minimal.*

Proof: Let E be an arbitrary DNA expression in $\mathcal{D}_{\text{Min}}$. We prove that E is minimal, by induction on the number p of operators occurring in E.

- If $p = 1$, then apparently the outermost operator of E is the only operator occurring in E. E does not have any expression-argument. Hence, either $E = \langle\uparrow \alpha\rangle$, or $E = \langle\downarrow \alpha\rangle$ or $E = \langle\updownarrow \alpha\rangle$ for an $\mathcal{N}$-word α. Indeed, these are minimal DNA expressions, which denote $\binom{\alpha}{-}$, $\binom{-}{\alpha}$ and $\binom{\alpha}{c(\alpha)}$, respectively.

- Let $p \geq 1$, and suppose that each DNA expression in $\mathcal{D}_{\text{Min}}$ that contains at most p operators is minimal (induction hypothesis).

  Now consider a DNA expression E in $\mathcal{D}_{\text{Min}}$ that contains $p+1$ operators. Because $p+1 \geq 2$, Property ($\mathcal{D}_{\text{Min}}$.1) implies that E is not an $\updownarrow$-expression.

  Assume that E is an $\uparrow$-expression: $E = \langle \uparrow \varepsilon_1 \dots \varepsilon_n \rangle$ for some $n \geq 1$ and $\mathcal{N}$-words and DNA expressions $\varepsilon_1, \dots, \varepsilon_n$. In fact, since E contains $p+1 \geq 2$ operators, it follows from Property ($\mathcal{D}_{\text{Min}}$.3) that $n \geq 2$. Let $X = \mathcal{S}(E)$ and for $i = 1, \dots, n$, let $X_i = \mathcal{S}^+(\varepsilon_i)$. By definition,

  $$X = \nu^+(X_1) y_1 \nu^+(X_2) y_2 \dots y_{n-1} \nu^+(X_n),$$

  where the y_i's are defined by (4.3).

  By Property ($\mathcal{D}_{\text{Min}}$.2), E does not have $\uparrow$-arguments. Hence, its arguments are $\mathcal{N}$-words, $\updownarrow$-expressions and $\downarrow$-expressions. Consider an arbitrary argument ε_i of E.

  If ε_i is an $\mathcal{N}$-word α, then $X_i = \binom{\alpha}{-}$. Because in this case $B_\downarrow(X_i) = n_\updownarrow(X_i) = 0$,

  $$|\varepsilon_i| = |\alpha| = |X_i|_{\mathcal{A}} = 3 \cdot B_\downarrow(X_i) + 3 \cdot n_\updownarrow(X_i) + |X_i|_{\mathcal{A}}.$$

  If ε_i is an $\updownarrow$-expression, then by Property ($\mathcal{D}_{\text{Min}}$.1), it is of the form $\langle \updownarrow \alpha \rangle$ for an $\mathcal{N}$-word α. In this case, $X_i = \binom{\alpha}{c(\alpha)}$, $B_\downarrow(X_i) = 0$ and $n_\updownarrow(X_i) = 1$. Hence,

  $$|\varepsilon_i| = |\,\langle \updownarrow \alpha \rangle\,| = 3 + |X_i|_{\mathcal{A}} = 3 \cdot B_\downarrow(X_i) + 3 \cdot n_\updownarrow(X_i) + |X_i|_{\mathcal{A}}.$$

  Finally, assume that ε_i is a $\downarrow$-expression E_i. Because E_i does not contain the outermost operator $\uparrow$ of E, it contains at most p operators. By Lemma 8.24, $E_i \in \mathcal{D}_{\text{Min}}$, and by the induction hypothesis, E_i is a minimal DNA expression.

  All occurrences of $\uparrow$ and $\downarrow$ in E_i are inner occurrences in E. Hence, by Property ($\mathcal{D}_{\text{Min}}$.4) and Lemma 5.8, $X_i = \mathcal{S}(E_i)$ is nick free.

  By Property ($\mathcal{D}_{\text{Min}}$.3), E_i has at least two arguments. By Property ($\mathcal{D}_{\text{Min}}$.5), the first one is either an $\mathcal{N}$-word α_1 or an $\updownarrow$-expression $\langle \updownarrow \alpha_1 \rangle$ for an $\mathcal{N}$-word α_1. In the latter case, by Property ($\mathcal{D}_{\text{Min}}$.4), the second argument is an $\mathcal{N}$-word α_2. In both cases, X_i contains at least one single-stranded component and the first single-stranded component is a lower component. Analogously, the last single-stranded component of X_i is a lower component.

  Then by Lemma 6.13(3d), $B_\downarrow(X_i) = B_\uparrow(X_i) + 1$. Consequently, by Theorem 7.24(2),

  $$\begin{aligned} |\varepsilon_i| = |E_i| &= 3 + 3 \cdot B_\uparrow(X_i) + 3 \cdot n_\updownarrow(X_i) + |X_i|_{\mathcal{A}} \\ &= 3 \cdot B_\downarrow(X_i) + 3 \cdot n_\updownarrow(X_i) + |X_i|_{\mathcal{A}}. \end{aligned}$$

  We are now ready to calculate the length of E:

  $$\begin{aligned} |E| &= 3 + \sum_{i=1}^{n} |\varepsilon_i| \\ &= 3 + 3 \cdot \sum_{i=1}^{n} B_\downarrow(X_i) + 3 \cdot \sum_{i=1}^{n} n_\updownarrow(X_i) + \sum_{i=1}^{n} |X_i|_{\mathcal{A}}. \end{aligned}$$

For $i = 1, \ldots, n$, X_i is nick free and in particular, $\#_{\triangledown}(X_i) = 0$. But then Equations (6.4), (6.5) and (6.6) from Lemma 6.27(1) imply that

$$|E| = 3 + 3 \cdot B_{\downarrow}(X) + 3 \cdot n_{\updownarrow}(X) + |X|_{\mathcal{A}}. \tag{8.19}$$

Because (the $\uparrow$-expression) E achieves the lower bound from Theorem 6.31(1), it is operator-minimal. The only thing left to be proved is that it is also minimal.

If X contains nick letters, then by definition, these must be lower nick letters. By Lemma 7.35(1), each minimal DNA expression denoting X is an $\uparrow$-expression. In particular, E, which is an operator-minimal $\uparrow$-expression, is minimal. Indeed, $|E|$ equals the length of a minimal DNA expression denoting X, as specified in Theorem 7.46.

Now assume that X is nick free and recall that $n \geq 2$. By Corollary 8.11(1) and (3), E is alternating. Then by Property ($\mathcal{D}_{\text{Min}}$.6), either the first argument, or the last argument of E (or both arguments) is an $\mathcal{N}$-word α_1 or an $\updownarrow$-expression $\langle \updownarrow \alpha_1 \rangle$ for an $\mathcal{N}$-word α_1. Without loss of generality, assume that the first argument of E is an $\mathcal{N}$-word α_1 or an $\updownarrow$-expression $\langle \updownarrow \alpha_1 \rangle$ for an $\mathcal{N}$-word α_1. In the latter case, the second argument must be an $\mathcal{N}$-word α_2. In both cases, $X = \mathcal{S}(E)$ contains at least one single-stranded component and the first single-stranded component is an upper component. Then it follows from Lemma 6.13(3a) and (3c) that $B_{\uparrow}(X) \geq B_{\downarrow}(X)$, and from Theorem 7.24(1) that there exists at least one minimal $\uparrow$-expression denoting X. Consequently, the operator-minimal $\uparrow$-expression E is also minimal.

The proof for the case that E is a $\downarrow$-expression is analogous.

□

When we combine Lemma 8.22 and Lemma 8.25, we obtain the characterization that we set out for:

Theorem 8.26 *A DNA expression E is minimal if and only if $E \in \mathcal{D}_{Min}$.*

Hence, the minimal DNA expressions can be characterized by six properties, Properties ($\mathcal{D}_{\text{Min}}$.1) – ($\mathcal{D}_{\text{Min}}$.6). One may replace one or more of these properties by other, new properties, and yet retain a valid characterization. However, none of the six properties can simply be omitted, i.e., none of the properties follows from the remaining set of properties. For each of the properties, there exist DNA expressions without that particular property (which thus are not minimal), that do have the other five properties. Examples of this are given in Table 8.1.

We use the characterization to derive other properties of minimal DNA expressions.

Lemma 8.27 *Let E be a minimal DNA expression.*

1. *(a) For each proper $\uparrow$-subexpression of E, the parent operator is $\downarrow$.*
   (b) For each proper $\downarrow$-subexpression of E, the parent operator is $\uparrow$.
2. *Each proper $\uparrow$-subexpression or $\downarrow$-subexpression of E has at least two arguments.*
3. *If E is nick free, then it has at least one $\mathcal{N}$-word-argument α.*
4. *Each proper DNA subexpression of E has at least one $\mathcal{N}$-word-argument α.*

Prop.	E	$X = \mathcal{S}(E)$	E^*
$(\mathcal{D}_{\text{Min}}.1)$	$\langle\updownarrow \langle\updownarrow \alpha_1\rangle\rangle$	$\binom{\alpha_1}{c(\alpha_1)}$	$\langle\updownarrow \alpha_1\rangle$
$(\mathcal{D}_{\text{Min}}.1)$	$\langle\uparrow \alpha_1 \langle\updownarrow \langle\uparrow \alpha_2 \langle\updownarrow \alpha_3\rangle\rangle\rangle\rangle$	$\binom{\alpha_1}{-}\binom{\alpha_2\alpha_3}{c(\alpha_2\alpha_3)}$	$\langle\uparrow \alpha_1 \langle\updownarrow \alpha_2\alpha_3\rangle\rangle$
$(\mathcal{D}_{\text{Min}}.2)$	$\langle\uparrow \alpha_1 \langle\uparrow \langle\updownarrow \alpha_2\rangle \alpha_3\rangle\rangle$	$\binom{\alpha_1}{-}\binom{\alpha_2}{c(\alpha_2)}\binom{\alpha_3}{-}$	$\langle\uparrow \alpha_1 \langle\updownarrow \alpha_2\rangle \alpha_3\rangle$
$(\mathcal{D}_{\text{Min}}.3)$	$\langle\uparrow \langle\updownarrow \alpha_1\rangle\rangle$	$\binom{\alpha_1}{c(\alpha_1)}$	$\langle\updownarrow \alpha_1\rangle$
$(\mathcal{D}_{\text{Min}}.3)$	$\langle\downarrow \alpha_1 \langle\uparrow \langle\updownarrow \alpha_2\rangle\rangle\rangle$	$\binom{-}{\alpha_1}\binom{\alpha_2}{c(\alpha_2)}$	$\langle\downarrow \alpha_1 \langle\updownarrow \alpha_2\rangle\rangle$
$(\mathcal{D}_{\text{Min}}.4)$	$\langle\uparrow \alpha_1 \langle\downarrow \langle\updownarrow \alpha_2\rangle \langle\updownarrow \alpha_3\rangle\rangle\rangle$	$\binom{\alpha_1}{-}\binom{\alpha_2\alpha_3}{c(\alpha_2\alpha_3)}$	$\langle\uparrow \alpha_1 \langle\updownarrow \alpha_2\alpha_3\rangle\rangle$
$(\mathcal{D}_{\text{Min}}.5)$	$\langle\uparrow \alpha_1 \langle\downarrow \langle\uparrow \alpha_2 \langle\updownarrow \alpha_3\rangle\rangle \alpha_4\rangle\rangle$	$\binom{\alpha_1\alpha_2}{-}\binom{\alpha_3}{c(\alpha_3)}\binom{-}{\alpha_4}$	$\langle\uparrow \alpha_1\alpha_2 \langle\downarrow \langle\updownarrow \alpha_3\rangle \alpha_4\rangle\rangle$
$(\mathcal{D}_{\text{Min}}.5)$	$\langle\uparrow \langle\downarrow \alpha_1 \langle\uparrow \langle\updownarrow \alpha_2\rangle \alpha_3\rangle\rangle \alpha_4\rangle$	$\binom{-}{\alpha_1}\binom{\alpha_2}{c(\alpha_2)}\binom{\alpha_3\alpha_4}{-}$	$\langle\uparrow \langle\downarrow \alpha_1 \langle\updownarrow \alpha_2\rangle\rangle \alpha_3\alpha_4\rangle$
$(\mathcal{D}_{\text{Min}}.6)$	$\langle\uparrow \langle\downarrow \alpha_1 \langle\updownarrow \alpha_2\rangle\rangle \alpha_3 \langle\downarrow \langle\updownarrow \alpha_4\rangle \alpha_5\rangle\rangle$	$\binom{-}{\alpha_1}\binom{\alpha_2}{c(\alpha_2)}\binom{\alpha_3}{-}\binom{\alpha_4}{c(\alpha_4)}\binom{-}{\alpha_5}$	$\langle\downarrow \alpha_1 \langle\uparrow \langle\updownarrow \alpha_2\rangle \alpha_3 \langle\updownarrow \alpha_4\rangle\rangle \alpha_5\rangle$

Table 8.1: Examples of DNA expressions with all six properties from Lemma 8.22 except one. The first column mentions the property that is not valid, the second column contains a corresponding DNA expression E, the third column gives the formal DNA molecule X denoted by E, and the fourth column contains a minimal DNA expression E^* denoting X. As usual, the α_i's occurring represent (arbitrary) $\mathcal{N}$-words.

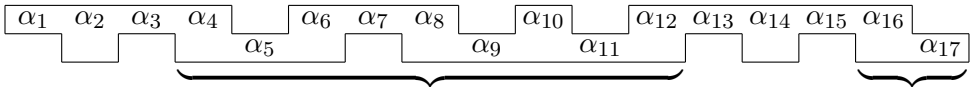


Figure 8.2: Nick free formal DNA molecule X that is denoted by the minimal DNA expression E in (8.20). The lower block partitioning used to construct E is also indicated.

5. *(a) Each proper $\uparrow$-subexpression or $\downarrow$-subexpression of E which is not the first argument of its parent operator has as its (own) first argument an $\updownarrow$-expression $\langle\updownarrow \alpha\rangle$ for an $\mathcal{N}$-word α.*

   (b) Each proper $\uparrow$-subexpression or $\downarrow$-subexpression of E which is not the last argument of its parent operator has as its (own) last argument an $\updownarrow$-expression $\langle\updownarrow \alpha\rangle$ for an $\mathcal{N}$-word α.

6. *For each proper $\uparrow$-subexpression or $\downarrow$-subexpression of E, either the first argument, or the last argument is an $\updownarrow$-expression $\langle\updownarrow \alpha\rangle$ for an $\mathcal{N}$-word α.*

7. *Each proper $\uparrow$-subexpression or $\downarrow$-subexpression of E which is neither the first argument, nor the last argument of E, has an odd number of arguments (at least three), the first one and the last one of which are $\updownarrow$-expressions $\langle\updownarrow \alpha\rangle$ for $\mathcal{N}$-words α.*

In fact, we already proved and used Claims 1 and 2 in the proof of Lemma 8.22 and the proof of Lemma 8.24, respectively. For the sake of clearness, we also include these claims here.

Example 8.28 Consider the nick free formal DNA molecule X depicted in Figure 8.2. Using the construction from Theorem 7.24, we can obtain different minimal DNA expressions denoting X. In fact, because $B_{\uparrow}(X) = B_{\downarrow}(X) = 3$, we can obtain both minimal $\uparrow$-

expressions and minimal ↓-expressions. If we start with the lower block partitioning indicated in the picture (to construct a minimal ↑-expression), then the minimal ↓-expressions denoting the two lower blocks are fixed. The result is

$$E = \langle\uparrow \quad \alpha_1 \langle\updownarrow \alpha_2\rangle \alpha_3 \quad \langle\downarrow \langle\updownarrow \alpha_4\rangle \alpha_5 \langle\uparrow \langle\updownarrow \alpha_6\rangle \alpha_7 \langle\updownarrow \alpha_8\rangle\rangle \alpha_9 \langle\updownarrow \alpha_{10}\rangle \alpha_{11} \langle\updownarrow \alpha_{12}\rangle\rangle \\ \alpha_{13} \langle\updownarrow \alpha_{14}\rangle \alpha_{15} \quad \langle\downarrow \langle\updownarrow \alpha_{16}\rangle \alpha_{17}\rangle \quad \rangle\,. \tag{8.20}$$

We examine which of the last three claims apply to which DNA subexpressions of E.

Claim 5a is applicable to all proper ↑-subexpressions and ↓-subexpressions of E. The first argument of E itself is *not* an ↕-expression.

Claim 5b is applicable to the only proper ↑-subexpression and to the first proper ↓-subexpression of E. Both for E itself and for the second proper ↓-expression $\langle\downarrow \langle\updownarrow \alpha_{16}\rangle \alpha_{17}\rangle$ of E, the last argument is *not* an ↕-expression.

Claim 6 is applicable to all proper ↑-subexpressions and ↓-subexpressions of E. Neither the first argument, nor the last argument of E itself is an ↕-expression.

Claim 7 is applicable to the only proper ↑-subexpression and to the first proper ↓-subexpression of E. Neither E itself, nor the second proper ↓-expression $\langle\downarrow \langle\updownarrow \alpha_{16}\rangle \alpha_{17}\rangle$ of E, has an odd number of arguments, let alone an odd number of arguments with additional properties. ■

Note that in general, even when a claim is not applicable to a DNA (sub-)expression, it may be true for that DNA (sub-)expression. For example, the DNA expression E above has eight arguments and four of these arguments are $\mathcal{N}$-words (cf. Claims 2 and 4). As another example, the minimal DNA expression from Equations (7.28) and (8.18) has as its last argument the ↕-expression $\langle\updownarrow \alpha_{22}\rangle$ (cf. Claims 5b and 6).

Proof of Lemma 8.27:

1. (a) Consider an arbitrary proper ↑-subexpression E^s of E. By Property ($\mathcal{D}_{\text{Min}}$.1), its parent operator cannot be ↕, and by Property ($\mathcal{D}_{\text{Min}}$.2), its parent operator cannot be ↑. Hence, the parent operator of E^s is ↓ (cf., for example, the first paragraph of the proof of Lemma 8.22($\mathcal{D}_{\text{Min}}$.5)).

   (b) The proof of this subclaim is analogous to that of the previous subclaim.

2. Consider an arbitrary proper ↑-subexpression E^s of E. Because the DNA expressions $\langle\uparrow \alpha\rangle$ or $\langle\downarrow \alpha\rangle$ for $\mathcal{N}$-words α do not have proper DNA subexpressions, we may assume that E is not equal to such a DNA expression. Now, by Property ($\mathcal{D}_{\text{Min}}$.3), E^s has at least two arguments (cf. the proof of Lemma 8.24).

   The proof for a proper ↓-subexpression E^s of E is analogous.

3. Assume that E is nick free. If E is an ↕-expression, then the claim follows from Property ($\mathcal{D}_{\text{Min}}$.1).

   Now assume that E is an ↑-expression. By Corollary 8.11(1) and (3), E is alternating. Hence, if E has at least two arguments, then at least one of these arguments is a maximal $\mathcal{N}$-word occurrence. If, on the other hand, E has only one argument, then by Property ($\mathcal{D}_{\text{Min}}$.3), this must be an $\mathcal{N}$-word α.

   The proof for a ↓-expression E is analogous.

4. Consider an arbitrary proper DNA subexpression E^s of E. E^s is minimal and by Corollary 8.5, E^s is nick free. Now this claim follows immediately from the previous claim.

5. (a) Consider an arbitrary proper $\uparrow$-subexpression E_2^s of E, which is not the first argument of its parent operator. Let ε_1 be the preceding argument (either an $\mathcal{N}$-word, or a DNA expression) of the parent operator.

   By Claim 1a, the parent operator is $\downarrow$. By Property ($\mathcal{D}_{\text{Min}}$.5), the first argument of E_2^s is either an $\mathcal{N}$-word α or an $\updownarrow$-expression $\langle\updownarrow\alpha\rangle$ for an $\mathcal{N}$-word α. If it were an $\mathcal{N}$-word α, then by Lemma 4.13(3) and (1), $L(\mathcal{S}(E_2^s))$ would be in $\mathcal{A}_+$ and ε_1 and E_2^s would not fit together by lower strands, which is required by $\downarrow$. Hence, the first argument of E_2^s is an $\updownarrow$-expression $\langle\updownarrow\alpha\rangle$ for an $\mathcal{N}$-word α.

   The proof for a proper $\downarrow$-subexpression of E which is not the first argument of its parent operator is analogous.

   (b) The proof of this subclaim is analogous to that of the previous subclaim.

6. Consider an arbitrary proper $\uparrow$-subexpression E^s of E. As in the proof of Claim 2, we may assume that E is not equal to $\langle\uparrow\alpha\rangle$ or $\langle\downarrow\alpha\rangle$ for an $\mathcal{N}$-word α.

   By Claim 1a, the parent operator of E^s is $\downarrow$, and by Property ($\mathcal{D}_{\text{Min}}$.3), this operator $\downarrow$ has at least two arguments. Hence, either E^s is not the first argument, or E^s is not the last argument of its parent operator (or both). By Claim 5, in the former case, the first argument of E^s is $\langle\updownarrow\alpha\rangle$ for an $\mathcal{N}$-word α, and in the latter case, the last argument of E^s is $\langle\updownarrow\alpha\rangle$ for an $\mathcal{N}$-word α.

   The proof for a proper $\downarrow$-subexpression E^s of E is analogous.

7. Consider an arbitrary proper $\uparrow$-subexpression E_1^s of E, which is neither the first argument, nor the last argument of E. By Claim 1a, E_1^s is the argument of a $\downarrow$-subexpression E_0^s of E.

   If $E_0^s = E$, then E_1^s is neither the first argument, nor the last argument of its parent operator $\downarrow$. Hence by Claim 5, both the first argument and the last argument of E_1^s are $\updownarrow$-expressions $\langle\updownarrow\alpha\rangle$ for $\mathcal{N}$-words α.

   If $E_0^s \neq E$, then the outermost operator of E_0^s is an inner occurrence of $\downarrow$ in E. By Property ($\mathcal{D}_{\text{Min}}$.5), (the $\uparrow$-expression) E_1^s cannot be the first argument or the last argument of E_0^s. Again by Claim 5, both the first argument and the last argument of E_1^s are $\updownarrow$-expressions $\langle\updownarrow\alpha\rangle$ for $\mathcal{N}$-words α.

   By Claim 2, E_1^s has at least two arguments, which by Property ($\mathcal{D}_{\text{Min}}$.4) are maximal $\mathcal{N}$-word occurrences and DNA expressions, alternately. Now the claim follows immediately.

   The proof for a proper $\downarrow$-subexpression E_1^s of E is analogous.

□

It is worth noting that *operator*-minimal DNA expressions may be characterized by (variants of) Properties ($\mathcal{D}_{\text{Min}}$.1)–($\mathcal{D}_{\text{Min}}$.5). Property ($\mathcal{D}_{\text{Min}}$.6), which justifies the use of a certain outermost operator, is not applicable now. In operator-minimal DNA expressions, the outermost operator does not have to be justified, because by definition, these DNA expressions are only compared to equivalent DNA expressions with the same outermost operator.

In this thesis, we do not work out the characterization of operator-minimal DNA expressions, because we mainly study them as means to construct *minimal* DNA expressions (for expressible formal DNA molecules with nick letters).

8.4 The structure tree of a minimal DNA expression

As we observed in Section 4.6, each DNA expression has a unique representation as an ordered, rooted, node-labelled tree: the structure tree. In particular, such a unique tree-representation exists for every minimal DNA expression. The resulting structure trees are also called minimal.

In Section 8.3, we have proved that minimal DNA expressions can be characterized by six properties of the operators occurring in them. These properties can be directly translated into properties characterizing minimal structure trees. Also other results on minimal DNA expressions can be translated into tree-terminology. For several results, we will now give the structure tree versions.

Let t be the structure tree of a DNA expression E.

Corollary 7.3 t is minimal if and only if for every DNA expression E' with $E' \equiv E$, the structure tree of E' contains at least as many internal nodes as t.

Lemma 7.4 and Lemma 8.24 t is minimal if and only if each subtree of t rooted in an internal node of t is minimal.

Theorem 8.26 (and Lemma 8.22) t is minimal if and only if

- **($\mathcal{D}_{\mathbf{Min}}$.1)** each node labelled by $\updownarrow$ in t has a (single) child labelled by an $\mathcal{N}$-word α, and
- **($\mathcal{D}_{\mathbf{Min}}$.2)** no node labelled by $\uparrow$ in t has a child labelled by $\uparrow$, and no node labelled by $\downarrow$ in t has a child labelled by $\downarrow$, and
- **($\mathcal{D}_{\mathbf{Min}}$.3)** unless $E = \langle\uparrow \alpha\rangle$ or $E = \langle\downarrow \alpha\rangle$ for an $\mathcal{N}$-word α, each node labelled by either $\uparrow$ or $\downarrow$ in t has at least two children, and
- **($\mathcal{D}_{\mathbf{Min}}$.4)** for each non-root labelled by either $\uparrow$ or $\downarrow$ in t, the children are labelled by an $\mathcal{N}$-word α or by an operator, alternately, and
- **($\mathcal{D}_{\mathbf{Min}}$.5)** for each non-root labelled by either $\uparrow$ or $\downarrow$ in t, the first child is labelled by either an $\mathcal{N}$-word α or the operator $\updownarrow$, and also the last child is labelled by either an $\mathcal{N}$-word α or the operator $\updownarrow$, and
- **($\mathcal{D}_{\mathbf{Min}}$.6)** if the root of t is labelled by either $\uparrow$ or $\downarrow$, then either its first child is labelled by an $\mathcal{N}$-word α or the operator $\updownarrow$, or its last child is labelled by an $\mathcal{N}$-word α or the operator $\updownarrow$, or it has two consecutive children labelled by an operator.

Lemma 8.27 If t is minimal, then

1. (a) each non-root labelled by $\uparrow$ in t has as parent a node labelled by $\downarrow$;
   (b) each non-root labelled by $\downarrow$ in t has as parent a node labelled by $\uparrow$;
2. each non-root labelled by either $\uparrow$ or $\downarrow$ in t has at least two children;
3. if E is nick free, then the root has at least one child labelled by an $\mathcal{N}$-word α;
4. each non-root labelled by an operator in t has at least one child labelled by an $\mathcal{N}$-word α;
5. (a) each non-root labelled by either $\uparrow$ or $\downarrow$ in t which is not the first child of its parent has as its (own) first child a node labelled by $\updownarrow$;

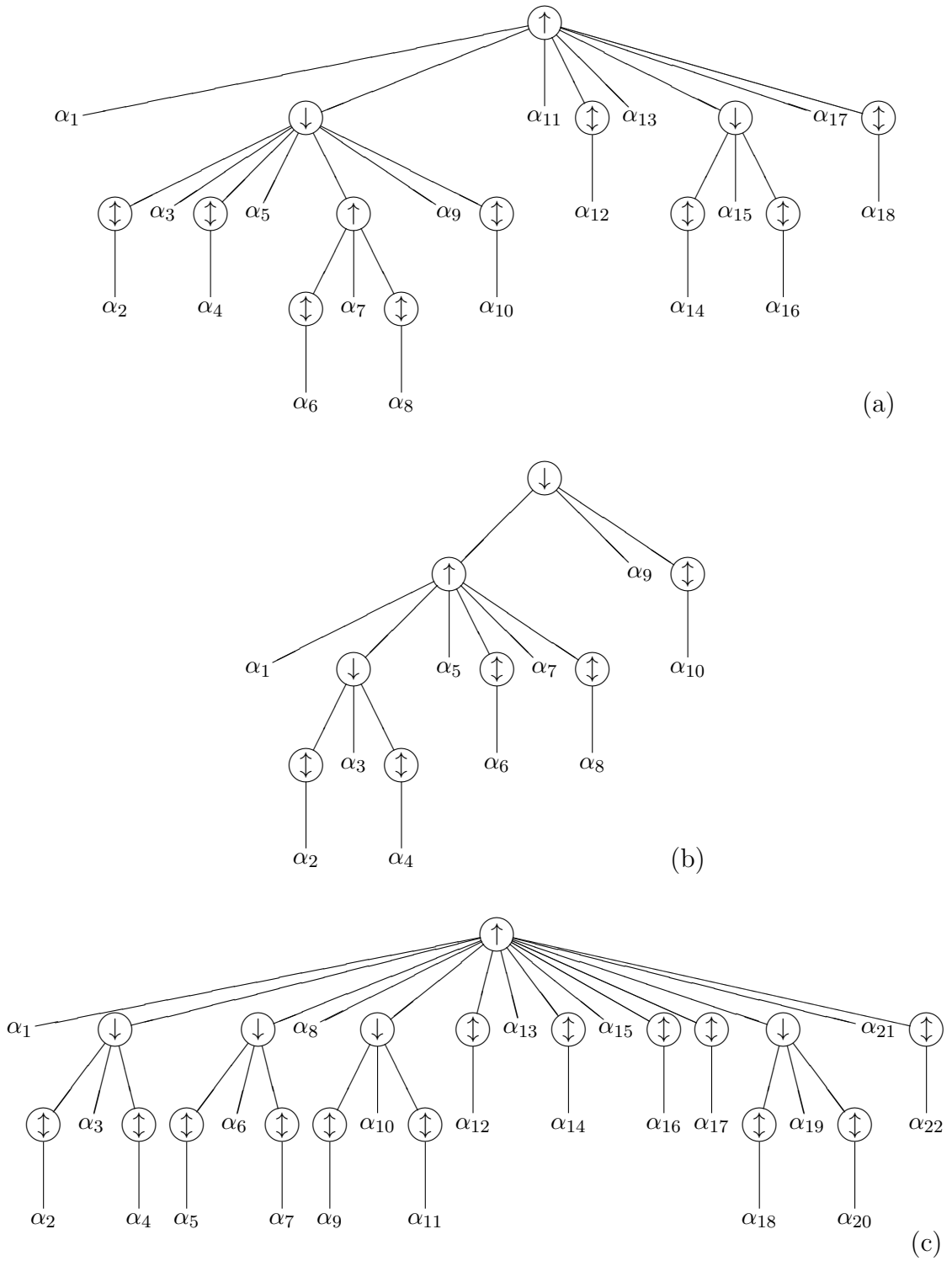


Figure 8.3: Three minimal structure trees. (a) The structure tree of the minimal DNA expression from Equation (7.7), denoting the nick free formal DNA molecule from (a.o.) Figure 7.5. (b) The structure tree of the minimal DNA expression E_d from Equation (7.12), denoting the nick free formal DNA molecule from Figure 7.6. (c) The structure tree of the minimal DNA expression from Equation (7.28), denoting the formal DNA molecule from Figure 7.7, which contains four lower nick letters.

(b) each non-root labelled by either $\uparrow$ or $\downarrow$ in t which is not the last child of its parent has as its (own) last child a node labelled by $\updownarrow$;

6. for each non-root labelled by either $\uparrow$ or $\downarrow$ in t, either the first child, or the last child is labelled by $\updownarrow$;
7. each non-root labelled by either $\uparrow$ or $\downarrow$ in t which is neither the first child nor the last child of the root, has an odd number of children (at least three), the first one and the last one of which are labelled by $\updownarrow$.

In Figure 8.3, we have drawn the structure trees of three minimal DNA expressions we have constructed in the course of Chapter 7. One can verify that these structure trees exhibit all properties we have just listed. Especially the last property, corresponding to Lemma 8.27(7), comes out clearly.

8.5 The number of (operator-)minimal DNA expressions

In Chapter 7, Section 8.1 and Section 8.3, we have seen examples of formal DNA molecules for which the minimal DNA expression is unique (e.g., in Lemma 8.18) and examples of formal DNA molecules for which there exist more than one minimal DNA expression (e.g., in Example 7.25 and Example 7.26). In this section, we will calculate the *number* of minimal DNA expressions denoting an *arbitrary* expressible formal DNA molecule X.

We first introduce some notation.

Definition 8.29 *Let X be an expressible formal DNA molecule. Then*

- $n_{min}(X)$ *is the number of minimal DNA expressions denoting X,*
- $n_{min\uparrow}(X)$ *is the number of minimal $\uparrow$-expressions denoting X,*
- $n_{min\downarrow}(X)$ *is the number of minimal $\downarrow$-expressions denoting X,*
- $n_{min\updownarrow}(X)$ *is the number of minimal $\updownarrow$-expressions denoting X,*
- $n_{opermin\uparrow}(X)$ *is the number of operator-minimal $\uparrow$-expressions denoting X,*
- $n_{opermin\downarrow}(X)$ *is the number of operator-minimal $\downarrow$-expressions denoting X,*
- $n_{opermin\updownarrow}(X)$ *is the number of operator-minimal $\updownarrow$-expressions denoting X.*

Obviously, $n_{\min}(X) = n_{\min\uparrow}(X) + n_{\min\downarrow}(X) + n_{\min\updownarrow}(X)$. For each type of expressible formal DNA molecule, we can easily obtain values for (some of) the seven functions and relations between the functions:

Lemma 8.30 *Let X be an expressible formal DNA molecule.*

1. *If X is double-complete, then*

$$n_{min\uparrow}(X) = 0, \tag{8.21}$$
$$n_{opermin\uparrow}(X) = 1, \tag{8.22}$$
$$n_{min\downarrow}(X) = 0, \tag{8.23}$$

$$n_{opermin_{\downarrow}}(X) = 1, \tag{8.24}$$
$$n_{min_{\updownarrow}}(X) = n_{opermin_{\updownarrow}}(X) = 1 \text{ and} \tag{8.25}$$
$$n_{min}(X) = 1. \tag{8.26}$$

2. *If X is nick free, contains at least one single-stranded component and $B_{\uparrow}(X) = B_{\downarrow}(X)$, then*

$$n_{min_{\uparrow}}(X) = n_{opermin_{\uparrow}}(X) \geq 1, \tag{8.27}$$
$$n_{min_{\downarrow}}(X) = n_{opermin_{\downarrow}}(X) \geq 1, \tag{8.28}$$
$$n_{min_{\updownarrow}}(X) = n_{opermin_{\updownarrow}}(X) = 0 \text{ and} \tag{8.29}$$
$$n_{min}(X) = n_{min_{\uparrow}}(X) + n_{min_{\downarrow}}(X). \tag{8.30}$$

3. *If X is nick free and $B_{\uparrow}(X) > B_{\downarrow}(X)$, then*

$$n_{min_{\uparrow}}(X) = n_{opermin_{\uparrow}}(X) \geq 1, \tag{8.31}$$
$$n_{min_{\downarrow}}(X) = 0, \tag{8.32}$$
$$n_{opermin_{\downarrow}}(X) \geq 1, \tag{8.33}$$
$$n_{min_{\updownarrow}}(X) = n_{opermin_{\updownarrow}}(X) = 0 \text{ and} \tag{8.34}$$
$$n_{min}(X) = n_{min_{\uparrow}}(X). \tag{8.35}$$

4. *If X is nick free and $B_{\downarrow}(X) > B_{\uparrow}(X)$, then*

$$n_{min_{\uparrow}}(X) = 0,$$
$$n_{opermin_{\uparrow}}(X) \geq 1,$$
$$n_{min_{\downarrow}}(X) = n_{opermin_{\downarrow}}(X) \geq 1,$$
$$n_{min_{\updownarrow}}(X) = n_{opermin_{\updownarrow}}(X) = 0 \text{ and}$$
$$n_{min}(X) = n_{min_{\downarrow}}(X).$$

5. *If X contains at least one lower nick letter, then let $Z_1{}_{\triangle}Z_2{}_{\triangle}\ldots{}_{\triangle}Z_m$ for some $m \geq 2$ be the nick free decomposition of X.*

$$n_{min_{\uparrow}}(X) = n_{opermin_{\uparrow}}(X) = n_{opermin_{\uparrow}}(Z_1) \times \cdots \times n_{opermin_{\uparrow}}(Z_m) \geq 1, \tag{8.36}$$
$$n_{min_{\downarrow}}(X) = n_{opermin_{\downarrow}}(X) = 0, \tag{8.37}$$
$$n_{min_{\updownarrow}}(X) = 0 \text{ and} \tag{8.38}$$
$$n_{min}(X) = n_{min_{\uparrow}}(X). \tag{8.39}$$

   (a) *If X does not contain any single-stranded component, then*

$$n_{opermin_{\updownarrow}}(X) = |Z_1| \times |Z_m|. \tag{8.40}$$

   (b) *If X contains at least one single-stranded component, then*

$$n_{opermin_{\updownarrow}}(X) = 0. \tag{8.41}$$

6. *If X contains at least one upper nick letter, then let $Z_1{}^{\triangledown}Z_2{}^{\triangledown}\ldots{}^{\triangledown}Z_m$ for some $m \geq 2$ be the nick free decomposition of X.*

$$\begin{aligned} n_{min\uparrow}(X) = n_{opermin\uparrow}(X) &= 0, \\ n_{min\downarrow}(X) = n_{opermin\downarrow}(X) &= n_{opermin\downarrow}(Z_1) \times \cdots \times n_{opermin\downarrow}(Z_m) \geq 1, \\ n_{min\updownarrow}(X) &= 0 \text{ and} \\ n_{min}(X) &= n_{min\downarrow}(X). \end{aligned}$$

   (a) *If X does not contain any single-stranded component, then*

$$n_{opermin\updownarrow}(X) = |Z_1| \times |Z_m|.$$

   (b) *If X contains at least one single-stranded component, then*

$$n_{opermin\updownarrow}(X) = 0.$$

Proof: As Claim 3 is analogous to Claim 4, and Claim 5 is analogous to Claim 6, so are the corresponding proofs. Hence, it is sufficient to prove Claims 1, 2, 3 and 5.

It follows immediately from the semantics of a DNA expression that for certain formal DNA molecules X, there does not exist any $\uparrow$-expression, $\downarrow$-expression or $\updownarrow$-expression denoting X. Hence, the corresponding number of operator-minimal DNA expressions is also 0. This is expressed by equations (8.29), (8.34), (8.37) and (8.41).

The correctness of equations (8.21), (8.23), (8.25)–(8.28), (8.30)–(8.32), (8.35), (8.38) and (8.39) follows directly from Summary 8.16.

We now prove the correctness of the remaining five equations.

1. Assume that $X = \binom{\alpha_1}{c(\alpha_1)}$ for an $\mathcal{N}$-word α_1.

   By Theorem 7.42 and Theorem 8.14, we can construct an operator-minimal $\uparrow$-expression denoting X from an arbitrary lower block partitioning of X and there is no other way to construct an operator-minimal $\uparrow$-expression. X obviously does not contain any lower component, By Lemma 7.19, there exists exactly one lower block partitioning of X: $\mathcal{P} = Y_0 = X$. Hence, there is also exactly one operator-minimal $\uparrow$-expression denoting X: $E = \langle\uparrow \langle\updownarrow \alpha_1\rangle\rangle$, and $n_{\text{opermin}\uparrow}(X) = 1$, which is equation (8.22).

   The proof of equation (8.24) is completely analogous.

3. Assume that X is nick free.

   By (the 'lower analogue' of) Theorem 7.42, an operator-minimal $\downarrow$-expression denoting X can be constructed from an arbitrary upper block partitioning of X. By Lemma 7.22, the number of upper block partitionings of any nick free formal DNA molecule is positive. Hence, $n_{\text{opermin}\downarrow}(X) \geq 1$, regardless of the values of $B_\uparrow(X)$ and $B_\downarrow(X)$. This gives equation (8.33).

5. Assume that X contains at least one lower nick letter, and let $Z_{1_\triangle}Z_{2_\triangle}\ldots{}_\triangle Z_m$ for some $m \geq 2$ be the nick free decomposition of X.

   By Theorem 7.46, Lemma 7.35(1) and Theorem 8.15, we can construct a minimal DNA expression denoting X from arbitrary operator-minimal $\uparrow$-expressions $E_1, \ldots, E_m$ denoting $Z_1, \ldots, Z_m$, respectively, and there is no other way to obtain a minimal DNA expression.

For $h = 1, \ldots, m$, there are $n_{\text{opermin}\uparrow}(Z_h) \geq 1$ operator-minimal $\uparrow$-expressions E_h denoting Z_h. Different choices for an E_h yield different minimal DNA expressions, because we simply copy the arguments of E_h into the minimal DNA expression. Finally, let $1 \leq h_1 \leq m$. Then the choice of an operator-minimal $\uparrow$-expression for Z_{h_1} is independent of the choices for the other Z_h's. Hence,

$$n_{\min}(X) = n_{\text{opermin}\uparrow}(Z_1) \times n_{\text{opermin}\uparrow}(Z_2) \times \cdots \times n_{\text{opermin}\uparrow}(Z_m) \geq 1.$$

Because the resulting minimal DNA expressions are $\uparrow$-expressions, we also have equation (8.36).

(a) Assume that X does not contain any single-stranded component. Then by Corollary 5.6, $X = \binom{\alpha_1}{c(\alpha_1)} \triangle \binom{\alpha_2}{c(\alpha_2)} \triangle \cdots \triangle \binom{\alpha_m}{c(\alpha_m)}$ for $\mathcal{N}$-words $\alpha_1, \alpha_2, \ldots, \alpha_m$. Apparently, for $h = 1, \ldots, m$, $Z_h = \binom{\alpha_h}{c(\alpha_h)}$.

By Theorem 8.19 and Theorem 8.20, the (only) operator-minimal $\updownarrow$-expressions denoting X are

$$\langle \updownarrow \langle \uparrow \alpha'_1 \langle \updownarrow \alpha''_1 \rangle \langle \updownarrow \alpha_2 \rangle \langle \updownarrow \alpha_3 \rangle \ldots \langle \updownarrow \alpha_{m-1} \rangle \langle \updownarrow \alpha'_m \rangle \alpha''_m \rangle \rangle \,,$$

where α'_1 and α''_m are (possibly empty) strings over $\mathcal{N}$, and α''_1 and α'_m are $\mathcal{N}$-words, such that $\alpha_1 = \alpha'_1\alpha''_1$ and $\alpha_m = \alpha'_m\alpha''_m$.

Because (the $\mathcal{N}$-word) α''_1 must be non-empty, the string α'_1 may consist of the first 0, 1, 2, ... or $|\alpha_1| - 1$ letters of α_1. Hence, there are $|\alpha_1| = |Z_1|$ possible pairs (α'_1, α''_1). Likewise, there are $|\alpha_m| = |Z_m|$ possible pairs (α'_m, α''_m).

Obviously, the choice for a pair (α'_1, α''_1) is independent of the choice for a pair (α'_m, α''_m). Consequently, $n_{\text{opermin}\updownarrow}(X) = |Z_1| \times |Z_m|$, which is equation (8.40).

□

The only thing we do not know yet, are the (exact) numbers of operator-minimal $\uparrow$-expressions and $\downarrow$-expressions denoting a nick free formal DNA molecule which contains at least one single-stranded component. To achieve these numbers, we will establish a bijection $f_{\langle\rangle}$ between these DNA expressions and certain sequences of brackets. As the number of such sequences is well known in the field of combinatorics, we also have the desired numbers of operator-minimal DNA expressions.

By Theorem 7.42 and Theorem 8.14, operator-minimal $\uparrow$-expressions and $\downarrow$-expressions for a nick free formal DNA molecule X are based on lower block partitionings and upper block partitionings of (submolecules of) X. In order to prove that the function $f_{\langle\rangle}$ is really a bijection, we first establish a bijection between lower (or upper) block partitionings and certain sequences of numbers, called ordered partitions.

Definition 8.31 *Let $N \geq 0$ be an integer. An ordered partition of N is a sequence of positive integers $(t_1, \ldots, t_r)$ for some $r \geq 0$ such that $t_1 + \cdots + t_r = N$.*

As an illustration of this definition, Table 8.2 contains all ordered partitions for $N = 0, 1, 2, 3, 4$. Note that there does exist an ordered partition of $N = 0$. For this partition, however, $r = 0$, because the numbers $t_1, \ldots, t_r$ occurring in the definition must be positive. Obviously, for each ordered partition of a number $N \geq 1$, we must have $r \geq 1$.

Ordered partitions are well known in combinatorics. As an aside, we consider the number of ordered partitions of a non-negative integer N. If $N \geq 1$, then each ordered

N	Ordered partitions of N
0	$()$
1	(1)
2	$(1,1),(2)$
3	$(1,1,1),(1,2),(2,1),(3)$
4	$(1,1,1,1),(1,1,2),(1,2,1),(2,1,1),(1,3),(2,2),(3,1),(4)$

Table 8.2: All ordered partitions for $N = 0, 1, 2, 3, 4$.

partition corresponds to a placing of separators | in a sequence of N 0's. For example, the ordered partition $(1, 4, 2, 5)$ of $N = 12$ (with $r = 4$) corresponds to the sequence 0|0000|00|00000. There are $N - 1$ spaces between the N 0's, and in each space, we may or may not put a separator. Hence, the total number of ordered partitions of $N \geq 1$ is 2^{N-1}. This is a standard result in the theory of partitions, see, e.g., [Hall, 1967, page 29]. When we add the case $N = 0$, we obtain

Proposition 8.32 *Let $N \geq 0$ be an integer. Then the number of different ordered partitions of N is*

$$\begin{cases} 1 & \text{if } N = 0, \\ 2^{N-1} & \text{if } N \geq 1. \end{cases}$$

We now define functions on lower block partitionings and upper block partitionings, as follows:

Definition 8.33 *Let X be a nick free formal DNA molecule.*
For a lower block partitioning $\mathcal{P} = Y_0\overline{X}_1Y_1\overline{X}_2Y_2 \ldots \overline{X}_rY_r$ of X (with $r \geq 0$),

$$f_{B_\downarrow}(\mathcal{P}) = (B_\downarrow(\overline{X}_1), B_\downarrow(\overline{X}_2), \ldots, B_\downarrow(\overline{X}_r)).$$

For an upper block partitioning $\mathcal{P} = Y_0\overline{X}_1Y_1\overline{X}_2Y_2 \ldots \overline{X}_rY_r$ of X (with $r \geq 0$),

$$f_{B_\uparrow}(\mathcal{P}) = (B_\uparrow(\overline{X}_1), B_\uparrow(\overline{X}_2), \ldots, B_\uparrow(\overline{X}_r)).$$

Hence, $f_{B_\downarrow}$ maps a lower block partitioning onto a sequence of numbers, and $f_{B_\uparrow}$ maps an upper block partitioning onto a sequence of numbers. In particular, if $\overline{X}_j$ is a lower block of X, then by Lemma 7.17(2a), $B_\downarrow(\overline{X}_j)$ is equal to the number of primitive lower blocks of X occurring in $\overline{X}_j$. Hence, for a lower block partitioning $\mathcal{P} = Y_0\overline{X}_1Y_1 \ldots \overline{X}_rY_r$ of X, $f_{B_\downarrow}(\mathcal{P})$ lists the numbers of primitive lower blocks occurring in $\overline{X}_1, \ldots, \overline{X}_r$, respectively. Of course, we have an analogous interpretation of $f_{B_\uparrow}(\mathcal{P})$ for an upper block partitioning of X.

Note that if $B_\downarrow(X) = 0$ for a nick free formal DNA molecule X, then by Lemma 7.19, the only lower block partitioning of X is $\mathcal{P} = X$ (with $r = 0$), for which $f_{B_\downarrow}(\mathcal{P}) = ()$. Analogously, if $B_\uparrow(X) = 0$, then the only upper block partitioning of X is $\mathcal{P} = X$, for which $f_{B_\uparrow}(\mathcal{P}) = ()$.

Example 8.34 When we apply $f_{B_\downarrow}$ and $f_{B_\uparrow}$ to the four lower block partitionings and the upper block partitioning from Figure 7.3, we obtain

$$\begin{aligned} f_{B_\downarrow}(\mathcal{P}_{\mathrm{a}1}) &= (1,1,1), \\ f_{B_\downarrow}(\mathcal{P}_{\mathrm{a}2}) &= (1,2), \\ f_{B_\downarrow}(\mathcal{P}_{\mathrm{a}3}) &= (2,1), \\ f_{B_\downarrow}(\mathcal{P}_{\mathrm{a}4}) &= (3) \text{ and} \\ f_{B_\uparrow}(\mathcal{P}_{\mathrm{b}}) &= (1,1,2). \end{aligned}$$

Here, the subscript of $\mathcal{P}$ refers to the subfigure of Figure 7.3. ■

In the above example, the four function values of $f_{B_\downarrow}$ are (different) ordered partitions of $B_\downarrow(X) = 3$. This is not a coincidence:

Lemma 8.35 *Let X be a nick free formal DNA molecule.*

1. *The function $f_{B_\downarrow}$ is a bijection from the lower block partitionings of X onto the ordered partitions of $B_\downarrow(X)$.*

2. *The function $f_{B_\uparrow}$ is a bijection from the upper block partitionings of X onto the ordered partitions of $B_\uparrow(X)$.*

As can be read from Table 8.2, if $B_\downarrow(X) = 0$, then the only ordered partition of $B_\downarrow(X)$ is the empty sequence () (with $r = 0$). Indeed, as we have just observed, this is the image under $f_{B_\downarrow}$ of the only lower block partitioning in this case.

We want to emphasize that the function $f_{B_\downarrow}$ (or $f_{B_\uparrow}$) is a bijection from lower (upper, respectively) block partitionings onto ordered partitions, *only* when the formal DNA molecule X is fixed. For example, there are infinitely many lower block partitionings that are mapped by $f_{B_\downarrow}$ to the empty sequence (), namely the lower block partitionings of all nick free formal DNA molecules X with $B_\downarrow(X) = 0$.

Finally, let $\mathcal{P}$ be an arbitrary lower block partitioning of X. Then Claim 1 implies in particular that $\mathcal{P}$ is completely characterized by the sequence $f_{B_\downarrow}(\mathcal{P})$.

Proof:

1. Let $\mathcal{P} = Y_0\overline{X}_1Y_1 \ldots \overline{X}_rY_r$ for some $r \geq 0$ be a lower block partitioning of X. By Lemma 7.17(2b), for $j = 1, \ldots, r$, $B_\downarrow(\overline{X}_j) = B_\uparrow(\overline{X}_j)+1 \geq 1$, and by Lemma 7.23(2), $B_\downarrow(\overline{X}_1) + \cdots + B_\downarrow(\overline{X}_r) = B_\downarrow(X)$. Indeed, $f_{B_\downarrow}(\mathcal{P}) = (B_\downarrow(\overline{X}_1), \ldots, B_\downarrow(\overline{X}_r))$ is an ordered partition of $B_\downarrow(X)$. Hence, $f_{B_\downarrow}$ is a mapping from the lower block partitionings of X into the set of these ordered partitions.

   We will now prove that $f_{B_\downarrow}$ is surjective and injective, and thus is a bijection. Let $\mathcal{P}' = Y'_0X'_1Y'_1 \ldots X'_{r_0}Y'_{r_0}$ for some $r_0 \geq 0$ be the primitive lower block partitioning of X. By definition, $r_0 = B_\downarrow(X)$ and $X'_1, \ldots, X'_{r_0}$ are precisely all primitive lower blocks of X.

   • Let $(t_1, t_2, \ldots, t_r)$ for some $r \geq 0$ be an arbitrary ordered partition of $B_\downarrow(X)$. Then consider the following partitioning of X:

$$\mathcal{P} = Y'_0\overline{X}_1Y'_{t_1}\overline{X}_2Y'_{t_1+t_2} \ldots \overline{X}_rY'_{t_1+\cdots+t_r}. \tag{8.42}$$

Since all t_j's are positive and $t_1+\cdots+t_r = B_\downarrow(X) = r_0$, the sequences $Y'_{t_1}, Y'_{t_1+t_2}, \ldots, Y'_{t_1+\cdots+t_{r-1}}$ occurring in (8.42) are different internal maximal upper sequences and $Y'_{t_1+\cdots+t_r}$ is the maximal upper suffix Y'_{r_0} of X. Hence, by Lemma 7.20, $\mathcal{P}$ is a lower block partitioning of X.

Note that for $j = 1, \ldots, r$, the substring $\overline{X}_j$ is equal to the alternating subsequence

$$X'_{t_1+\cdots+t_{j-1}+1} Y'_{t_1+\cdots+t_{j-1}+1} \ldots Y'_{t_1+\cdots+t_j-1} X'_{t_1+\cdots+t_j}$$

of primitive lower blocks and maximal upper sequences of X. Indeed, this is a lower block.

By Lemma 7.17(2a), for $j = 1, \ldots, r$,

$$B_\downarrow(\overline{X}_j) = (t_1 + \cdots + t_j) - (t_1 + \cdots + t_{j-1} + 1) + 1 = t_j.$$

Consequently,

$$f_{B_\downarrow}(\mathcal{P}) = (B_\downarrow(\overline{X}_1), B_\downarrow(\overline{X}_2), \ldots, B_\downarrow(\overline{X}_r)) = (t_1, t_2, \ldots, t_r).$$

We have thus constructed a lower block partitioning of X which is mapped by $f_{B_\downarrow}$ onto the ordered partition $(t_1, t_2, \ldots, t_r)$. Because this was an arbitrary ordered partition of $B_\downarrow(X)$, $f_{B_\downarrow}$ is surjective.

• Let

$$\begin{aligned} \mathcal{P}_1 &= Y_{1,0}\overline{X}_{1,1}Y_{1,1}\ldots\overline{X}_{1,r_1}Y_{1,r_1}, \text{ and} \\ \mathcal{P}_2 &= Y_{2,0}\overline{X}_{2,1}Y_{2,1}\ldots\overline{X}_{2,r_2}Y_{2,r_2} \end{aligned}$$

for some $r_1, r_2 \geq 0$ be two lower block partitionings of X, for which $f_{B_\downarrow}(\mathcal{P}_1) = f_{B_\downarrow}(\mathcal{P}_2) = (t_1, \ldots, t_r)$ for some $r \geq 0$. By the definition of $f_{B_\downarrow}$, $r_1 = r_2 = r$.

We now prove that the substrings occurring in the definition of $\mathcal{P}_1$ are equal to the ones occurring in the definition of $\mathcal{P}_2$: for $j = 0, \ldots, r$, $Y_{1,j} = Y_{2,j}$ and for $j = 1, \ldots, r$, $\overline{X}_{1,j} = \overline{X}_{2,j}$. We do this by induction on j.

– If $j = 0$, then by Lemma 7.20, $Y_{1,0} = Y_{2,0}$ is the maximal upper prefix Y'_0 of X.

– Let $0 \leq j \leq r-1$, and suppose that $Y_{1,j} = Y_{2,j}$ (induction hypothesis). We now consider $\overline{X}_{1,j+1}$, $\overline{X}_{2,j+1}$, $Y_{1,j+1}$ and $Y_{2,j+1}$.

The lower block $\overline{X}_{1,j+1}$ succeeds $Y_{1,j}$ in $\mathcal{P}_1$ and the lower block $\overline{X}_{2,j+1}$ succeeds $Y_{2,j}$ in $\mathcal{P}_2$. By the definition of a lower block, both $\overline{X}_{1,j+1}$ and $\overline{X}_{2,j+1}$ are alternating sequences of primitive lower blocks and maximal upper sequences of X, starting and ending with a primitive lower block. Because both of them succeed $Y_{1,j} = Y_{2,j}$ in X, they start with the same primitive lower block. By assumption, $B_\downarrow(\overline{X}_{1,j+1}) = B_\downarrow(\overline{X}_{2,j+1}) = t_{j+1}$, and hence, by Lemma 7.17(2a), $\overline{X}_{1,j+1}$ and $\overline{X}_{2,j+1}$ contain the same number of primitive lower blocks of X. This implies that they also end with the same primitive lower block, and thus that they are equal.

Now, if $j+1 < r$, then by Lemma 7.20, both $Y_{1,j+1}$ and $Y_{2,j+1}$ are the internal maximal upper sequence succeeding $\overline{X}_{1,j+1} = \overline{X}_{2,j+1}$ in X. If $j+1 = r$, then $Y_{1,j+1} = Y_{2,j+1}$ is the maximal upper suffix Y'_{r_0} of X. In both cases, $Y_{1,j+1} = Y_{2,j+1}$.

We conclude that $\mathcal{P}_1$ and $\mathcal{P}_2$ are equal, and thus that $f_{B_\downarrow}$ is injective.

2. The proof of this claim is analogous to that of the previous claim.

□

By the above result, in particular, the *number* of different lower block partitionings of a certain nick free formal DNA molecule X must be equal to the *number* of different ordered partitions of $B_{\downarrow}(X)$. Indeed, by Lemma 7.22, the former number is $2^{n_{\mathrm{imus}}(X)}$, which, by Lemma 7.9, is equal to

$$\begin{cases} 2^0 = 1 & \text{if } B_{\downarrow}(X) = 0, \\ 2^{B_{\downarrow}(X)-1} & \text{if } B_{\downarrow}(X) \geq 1. \end{cases}$$

By Proposition 8.32, this equals the number of ordered partitions of $B_{\downarrow}(X)$.

Lemma 8.35 relates lower block partitionings and upper block partitionings of a nick free formal DNA molecule to ordered partitions. We will now establish a relation between, on the one hand, operator minimal $\uparrow$-expressions and $\downarrow$-expressions denoting such molecules and, on the other hand, certain sequences of brackets.

Definition 8.36 *Let $p \geq 0$. A sequence of p well-nested pairs of brackets is a string $Z = x_1 \ldots x_{2p}$ such that*

- *for $i = 1, \ldots, 2p$, $x_i \in \{\langle, \rangle\}$, and*
- *$\#_{\langle}(Z) = \#_{\rangle}(Z)$, and*
- *for $i = 0, \ldots, 2p$, $\#_{\langle}(x_1 \ldots x_i) \geq \#_{\rangle}(x_1 \ldots x_i)$.*

The number p, i.e., the number of pairs of brackets in Z, is denoted by $n_{pb}(Z)$.

The third condition of the definition intuitively says that, when we read Z from left to right, the number of closing brackets $\rangle$ we have read so far never exceeds the number of opening brackets $\langle$ we have read so far.

Note that the inequality in this third condition is automatically valid for $i = 0$ and $i = 2p$. In the former case, $x_1 \ldots x_i = \lambda$, and $\#_{\langle}(\lambda) = \#_{\rangle}(\lambda) = 0$. In the latter case, $x_1 \ldots x_i = Z$, and $\#_{\langle}(Z) = \#_{\rangle}(Z) = p$ by the second condition.

Example 8.37 Some sequences of well-nested pairs of brackets are

λ (the empty string, with $p = 0$),
$\langle\langle\rangle\rangle\,\langle\rangle$ (with $p = 3$), and
$\langle\rangle\,\langle\rangle\,\langle\langle\rangle\rangle$ (with $p = 4$).

■

Sequences of well-nested pairs of brackets are well known in the theory of formal languages. The language containing (all) these sequences can be generated by a simple context-free grammar, which has axiom S, terminal symbols $\langle$ and $\rangle$, and *productions* $S \longrightarrow \lambda$, $S \longrightarrow SS$, $S \longrightarrow \langle S \rangle$.

The following result gives two ways to extend sequences of well-nested pairs of brackets. The result is in accordance with the context-free grammar we just considered, but its correctness also follows immediately from Definition 8.36:

Lemma 8.38 *Let Z_1 and Z_2 be two sequences of well-nested pairs of brackets.*

1. *The concatenation Z_1Z_2 is a sequence of well-nested pairs of brackets.*
2. *$\langle Z_1\rangle$ is a sequence of well-nested pairs of brackets (and of course, so is $\langle Z_2\rangle$).*

The converse of Claim 2 is not true. For example, we can write $\langle\langle\rangle\rangle\,\langle\rangle$ as $\langle Z\rangle$, with $Z = \langle\rangle\rangle\,\langle$. Z is not a sequence of well-nested pairs of brackets, because it violates the third condition of the definition.

Let $Z = x_1 \dots x_{2p}$ for some $p \geq 0$ be a sequence of well-nested pairs of brackets. We just observed that for $i = 0$ and for $i = 2p$, the number of opening brackets and the number of closing brackets in the prefix $x_1 \dots x_i$ of Z are equal. There may be other indices i for which this is the case. We use all such indices to partition Z.

Therefore, let $0 = i_0 < i_1 < \cdots < i_r = 2p$ for some $r \geq 0$ be all indices i such that $\#_\langle(x_1 \dots x_i) = \#_\rangle(x_1 \dots x_i)$. Obviously, if $p \geq 1$, then also $r \geq 1$. Also, all i_j's are even, which implies in particular that for $j = 1, \dots, r$, $i_{j-1} + 1 < i_j$. Consider any j with $1 \leq j \leq r$. By definition,

$$\begin{array}{ll} \#_\langle(x_1 \dots x_{i_{j-1}}) = \#_\rangle(x_1 \dots x_{i_{j-1}}) & \text{(also if } j = 1\text{, i.e., if } i_{j-1} = i_0 = 0\text{)}, \\ 1 \leq i_{j-1} + 1 \leq 2p \qquad \text{and} & \\ \#_\langle(x_1 \dots x_{i_{j-1}+1}) \geq \#_\rangle(x_1 \dots x_{i_{j-1}+1}). & \end{array}$$

Consequently, $x_{i_{j-1}+1}$ must be an opening bracket $\langle$. Analogously, x_{i_j} is a closing bracket $\rangle$.

Now define Z_j as the substring of Z between (and not including) the opening bracket $x_{i_{j-1}+1}$ and the closing bracket x_{i_j}:

$$Z_j = x_{i_{j-1}+2} \dots x_{i_j-1}.$$

We can then write $Z = \langle Z_1\rangle \dots \langle Z_r\rangle$. It is easily verified that each Z_j is itself a sequence of well-nested pairs of brackets and that this partitioning of Z is unique. When we combine this observation with Lemma 8.38, we obtain

Lemma 8.39 *A string Z is a sequence of well-nested pairs of brackets, if and only if there exist strings $Z_1, \dots, Z_r$ for some $r \geq 0$, such that*

- *$Z = \langle Z_1\rangle \dots \langle Z_r\rangle$, and*
- *for $j = 1, \dots, r$, Z_j is itself a sequence of well-nested pairs of brackets.*

In this case, the partitioning of Z as $\langle Z_1\rangle \dots \langle Z_r\rangle$ is unique.

This result explains the term 'sequence of well-nested pairs of brackets'. Note that in Lemma 4.8 and Theorem 4.9, we used a similar construction to determine the nested structure of a DNA expression and identify the arguments of any operator occurring in it.

Example 8.40 The three sequences of well-nested pairs of brackets from Example 8.37 can be partitioned as follows:

$$\begin{array}{ll} \lambda = \lambda & \text{(with } r = 0\text{)}, \\ \langle\langle\rangle\rangle\,\langle\rangle = \langle Z_1\rangle\,\langle Z_2\rangle & \text{(with } r = 2,\ Z_1 = \langle\rangle \text{ and } Z_2 = \lambda\text{), and} \\ \langle\rangle\,\langle\rangle\,\langle\langle\rangle\rangle = \langle Z_1\rangle\,\langle Z_2\rangle\,\langle Z_3\rangle & \text{(with } r = 3,\ Z_1 = Z_2 = \lambda \text{ and } Z_3 = \langle\rangle\text{)}. \end{array}$$

■

p	C_p	p	C_p	p	C_p	p	C_p
0	1	5	42	10	16,796	15	9,694,845
1	1	6	132	11	58,786	16	35,357,670
2	2	7	429	12	208,012	17	129,644,790
3	5	8	1,430	13	742,900	18	477,638,700
4	14	9	4,862	14	2,674,440	19	1,767,263,190

Table 8.3: The Catalan numbers C_p for $p = 0, 1, \ldots, 19$.

Sequences of well-nested pairs of brackets are also well known in combinatorics. The number of such sequences is one of the many combinatorial interpretations of the Catalan numbers $C_p = \frac{1}{p+1}\binom{2p}{p}$ for $p \geq 0$:

Proposition 8.41 *The number of different sequences of $p \geq 0$ well-nested pairs of brackets is $\frac{1}{p+1}\binom{2p}{p}$.*

[Stanley, 2015] lists a large number of combinatorial interpretations of the Catalan numbers. In particular, [Stanley, 2015, Exercise 77] deals with *ballot sequences*: sequences of 1's and -1's, which are equivalent to our sequences of opening brackets and closing brackets. A simple and elegant proof of Proposition 8.41 can be found in [Cohen, 1978, pages 131-132]. In this proof, H's are substituted for opening brackets and D's for closing brackets.

Table 8.3 lists the Catalan numbers C_p for $p = 0, 1, \ldots, 19$. As the table suggests, the sequence of the Catalan numbers exhibits an exponential growth. In fact, we have for $p \geq 1$,

$$C_p = \frac{1}{p+1}\binom{2p}{p} = \frac{2p \cdot (2p-1)}{(p+1)p} \cdot \frac{1}{p}\binom{2(p-1)}{p-1} = (4 - \frac{6}{p+1})C_{p-1}.$$

Hence, the sequence grows almost as fast as the sequence 4^p for $p \geq 0$.[2]

We return to the world of DNA expressions. We define a mapping from operator-minimal $\uparrow$-expressions and $\downarrow$-expressions onto sequences of brackets.

Definition 8.42 *Let E be an operator-minimal $\uparrow$-expression or $\downarrow$-expression, denoting a certain formal DNA molecule X (which may contain nick letters), and let $E_1, \ldots, E_r$ for some $r \geq 0$ be the $\uparrow$-arguments and $\downarrow$-arguments of E, in the order of their occurrence in E. Then*

$$f_{\langle\rangle}(E) = \langle f_{\langle\rangle}(E_1)\rangle \ldots \langle f_{\langle\rangle}(E_r)\rangle .$$

The definition of the function $f_{\langle\rangle}$ is recursive: $f_{\langle\rangle}(E)$ is defined in terms of $f_{\langle\rangle}(E_j)$ for the $\uparrow$-arguments and $\downarrow$-arguments E_j of E. The $\mathcal{N}$-word-arguments and $\updownarrow$-arguments are ignored. Because E is operator-minimal, its arguments are minimal. In particular, its $\uparrow$-arguments and $\downarrow$-arguments E_j are operator-minimal. Indeed, $f_{\langle\rangle}$ is defined for such

[2] We could have reached this conclusion also using Stirling's approximation formula for $n!$ ($n! \approx \sqrt{2\pi n}\left(\frac{n}{e}\right)^n$, see, e.g., [Feller, 1968, pages 52-54]). We prefer the derivation above, as it is more elementary.

arguments. This implies that the recursion is well defined. If E does not have any $\uparrow$-arguments or $\downarrow$-arguments, then $f_{\langle\rangle}(E) = \lambda$.

Note that by Lemma 8.1, if E is an $\uparrow$-expression, then the arguments E_j occurring in the above definition are $\downarrow$-expressions. Analogously, if E is a $\downarrow$-expression, then all E_j's are $\uparrow$-expressions.

The function $f_{\langle\rangle}$ is defined for arbitrary operator-minimal $\uparrow$-expressions and $\downarrow$-expressions. Here, however, we are particularly interested in operator-minimal DNA expressions denoting *nick free* formal DNA molecules. Therefore, we use two such DNA expressions to illustrate the definition of the function.

Example 8.43 Equation (7.7) in Example 7.25 contains a minimal (and thus operator-minimal) $\uparrow$-expression E for the formal DNA molecule depicted in Figure 7.5, corresponding to the lower block partitioning in Figure 7.3(a3). For this DNA expression E, we have $r = 2$ and

$$f_{\langle\rangle}(E) = \langle\langle\rangle\rangle\,\langle\rangle\,.$$

Equation (7.26) in Example 7.44 contains an operator-minimal $\downarrow$-expression for the same formal DNA molecule. It corresponds to the upper block partitioning in Figure 7.3(b). For this DNA expression E', we have $r = 3$ and

$$f_{\langle\rangle}(E') = \langle\rangle\,\langle\rangle\,\langle\langle\rangle\rangle\,.$$

■

We now prove some properties of the function $f_{\langle\rangle}$.

Lemma 8.44 *Let E be an operator-minimal $\uparrow$-expression or $\downarrow$-expression, denoting a certain formal DNA molecule X (which may contain nick letters).*

1. *$f_{\langle\rangle}(E)$ is a sequence of well-nested pairs of brackets.*
2. *$f_{\langle\rangle}(E)$ results from E by removing from E all letters except the brackets $\langle$ and $\rangle$ corresponding to inner occurrences of the operators $\uparrow$ and $\downarrow$.*
3. *(a) If E is an $\uparrow$-expression, then*

$$n_{pb}(f_{\langle\rangle}(E)) = B_{\downarrow}(X).$$

(b) If E is a $\downarrow$-expression, then

$$n_{pb}(f_{\langle\rangle}(E)) = B_{\uparrow}(X).$$

Recall that for the formal DNA molecule X from (a.o.) Figure 7.3 and Figure 7.5, we have $B_{\downarrow}(X) = 3$ and $B_{\uparrow}(X) = 4$. In Example 8.43, we applied $f_{\langle\rangle}$ to two (operator-)minimal DNA expressions E and E' denoting this molecule. It is easy to verify that all claims hold for these two DNA expressions. In particular, concerning Claim 1, we had already seen the resulting sequences of brackets in Example 8.37 and Example 8.40.

Proof:

1, 2 We simultaneously prove these two claims, by induction on the number p of operators occurring in E.

- If $p = 1$, then $E = \langle\uparrow \alpha\rangle$ or $E = \langle\downarrow \alpha\rangle$ for an $\mathcal{N}$-word α. Indeed, by Theorem 7.42, E is an operator-minimal DNA expression, denoting $\binom{\alpha}{-}$ or $\binom{-}{\alpha}$, respectively. In neither case, E has any $\uparrow$-arguments or $\downarrow$-arguments. Hence, by definition, in both cases,

  $$f_{\langle\rangle}(E) = \lambda.$$

  Claim 1 and Claim 2 are obviously valid now.

- Let $p \geq 1$, and suppose that for each operator-minimal $\uparrow$-expression or $\downarrow$-expression E containing at most p operators, the claims are valid (induction hypothesis).

  Now consider an arbitrary operator-minimal $\uparrow$-expression E containing $p+1$ operators. By Lemma 8.1, E does not have any $\uparrow$-arguments. Let $E_1, \ldots, E_r$ for some $r \geq 0$ be the $\downarrow$-arguments of E, in the order of their occurrence in E.

  As we observed immediately after Definition 8.42, each E_j is operator-minimal (even minimal). Consider E_j for an arbitrary j with $1 \leq j \leq r$. Obviously, E_j contains at most p operators. By the induction hypothesis, $f_{\langle\rangle}(E_j)$ is a sequence of well-nested pairs of brackets, and by Lemma 8.38(2), so is $\langle f_{\langle\rangle}(E_j)\rangle$.

  If $r = 0$, then by definition, $f_{\langle\rangle}(E) = \lambda$, which is a sequence of well-nested pairs of brackets. If $r \geq 1$ and we (iteratively) apply Lemma 8.38(1) to $\langle f_{\langle\rangle}(E_1)\rangle, \ldots, \langle f_{\langle\rangle}(E_r)\rangle$, then we also find that $f_{\langle\rangle}(E) = \langle f_{\langle\rangle}(E_1)\rangle \ldots \langle f_{\langle\rangle}(E_r)\rangle$ is a sequence of well-nested pairs of brackets. We thus have Claim 1.

  For $j = 1, \ldots, r$, by the induction hypothesis, $f_{\langle\rangle}(E_j)$ results from E_j by removing from it all letters except the brackets corresponding to inner occurrences of the operators $\uparrow$ and $\downarrow$ in E_j. Hence, because E_j is a $\downarrow$-expression, $\langle f_{\langle\rangle}(E_j)\rangle$ results from E_j by removing from it all letters except the brackets corresponding to (any occurrence of) the operators $\uparrow$ and $\downarrow$.

  Obviously, a bracket $\langle$ or $\rangle$ occurring in an argument E_j of E corresponds to an operator $\uparrow$ or $\downarrow$ in E_j, if and only if it corresponds to an inner occurrence of such an operator in E. Further, by Corollary 8.2, each argument of E that is not a $\downarrow$-expression, is either an $\mathcal{N}$-word α or an $\updownarrow$-expression $\langle\updownarrow \alpha\rangle$ for an $\mathcal{N}$-word α. Such an argument does not contain brackets corresponding to an operator $\uparrow$ or $\downarrow$. Hence, when we remove from E all letters except the brackets corresponding to inner occurrences of the operators $\uparrow$ and $\downarrow$, we remove the outermost operator $\uparrow$ of E with its brackets, the arguments of E which are not a $\downarrow$-expression, and all letters in the $\downarrow$-expressions $E_1, \ldots, E_r$ which are not a bracket $\langle$ or $\rangle$ corresponding to an operator $\uparrow$ or $\downarrow$. This leaves $\langle f_{\langle\rangle}(E_1)\rangle \ldots \langle f_{\langle\rangle}(E_r)\rangle = f_{\langle\rangle}(E)$. Consequently, Claim 2 is also valid for E.

  The proof for the case that E is an operator-minimal $\downarrow$-expression containing $p+1$ operators is analogous.

3. By Claim 1, the function n_{pb} is indeed defined for $f_{\langle\rangle}(E)$.

   (a) Assume that E is an $\uparrow$-expression. By Corollary 8.17(1), the total number of occurrences of operators $\uparrow$ and $\downarrow$ (together) in E is $1 + B_{\downarrow}(X)$. Hence, the number of inner occurrences of these operators in E is $B_{\downarrow}(X)$. Now, the equality $n_{\mathrm{pb}}(f_{\langle\rangle}(E)) = B_{\downarrow}(X)$ follows from Claim 2.

   (b) The proof of this subclaim is analogous to that of the previous one.

□

We now restrict ourselves to nick free formal DNA molecules. We use Lemma 8.44 to prove the main result of this section:

Theorem 8.45 *Let X be a nick free formal DNA molecule.*

1. *The function $f_{\langle\rangle}$ is a bijection from the operator-minimal $\uparrow$-expressions denoting X onto the sequences of $B_{\downarrow}(X)$ well-nested pairs of brackets.*
2. *The function $f_{\langle\rangle}$ is a bijection from the operator-minimal $\downarrow$-expressions denoting X onto the sequences of $B_{\uparrow}(X)$ well-nested pairs of brackets.*

Proof: By Lemma 8.44(1) and (3), the function $f_{\langle\rangle}$ is indeed a mapping from the operator-minimal $\uparrow$-expressions (or $\downarrow$-expressions) denoting X into the set of sequences of $B_{\downarrow}(X)$ ($B_{\uparrow}(X)$, respectively) well-nested pairs of brackets.

Now, we will prove that $f_{\langle\rangle}$ is a bijection, both for the operator-minimal $\uparrow$-expressions and for the operator-minimal $\downarrow$-expressions. Hence, we will prove that

$1'$ for each sequence Z of $B_{\downarrow}(X)$ well-nested pairs of brackets, there is exactly one operator-minimal $\uparrow$-expression E denoting X such that $f_{\langle\rangle}(E) = Z$, and

$2'$ for each sequence Z of $B_{\uparrow}(X)$ well-nested pairs of brackets, there is exactly one operator-minimal $\downarrow$-expression E denoting X such that $f_{\langle\rangle}(E) = Z$.

We prove Claims $1'$ and $2'$ simultaneously, by induction on $B_{\downarrow}(X)$ and $B_{\uparrow}(X)$, respectively.

- If $B_{\downarrow}(X) = 0$, then the only sequence of $B_{\downarrow}(X)$ well-nested pairs of brackets is $Z = \lambda$.

  By Lemma 7.19, there is exactly one lower block partitioning of X, namely $\mathcal{P} = Y_0 = X$. Then by Theorem 7.42 and Theorem 8.14, there is exactly one operator-minimal $\uparrow$-expression E denoting X, each of whose arguments is an $\mathcal{N}$-word α_i or an $\updownarrow$-expression $\langle\updownarrow \alpha_i\rangle$ for an $\mathcal{N}$-word α_i. For this $\uparrow$-expression E, indeed $f_{\langle\rangle}(E) = Z = \lambda$. Thus, we have proved Claim $1'$ for nick free formal DNA molecules X with $B_{\downarrow}(X) = 0$.

  The proof of Claim $2'$ for nick free formal DNA molecules X with $B_{\uparrow}(X) = 0$ is analogous.

- Let $p \geq 0$, and suppose that Claim $1'$ is valid for each nick free formal DNA molecule X with $B_{\downarrow}(X) \leq p$, and that Claim $2'$ is valid for each nick free formal DNA molecule X with $B_{\uparrow}(X) \leq p$ (induction hypothesis).

  Now consider a nick free formal DNA molecule X with $B_{\downarrow}(X) = p + 1$. Let Z be an arbitrary sequence of $p + 1$ well-nested pairs of brackets. We will prove that there exists exactly one operator-minimal $\uparrow$-expression E denoting X, such that $f_{\langle\rangle}(E) = Z$. The construction we use for this is schematically depicted in Figure 8.4.

  We first prove that there exists *at most* one operator-minimal $\uparrow$-expression E denoting X for which $f_{\langle\rangle}(E) = Z$. We do this by establishing properties of such a DNA expression, which, in the end, form a complete characterization. Therefore, assume that E is an operator-minimal $\uparrow$-expression denoting X and that $f_{\langle\rangle}(E) = Z$.

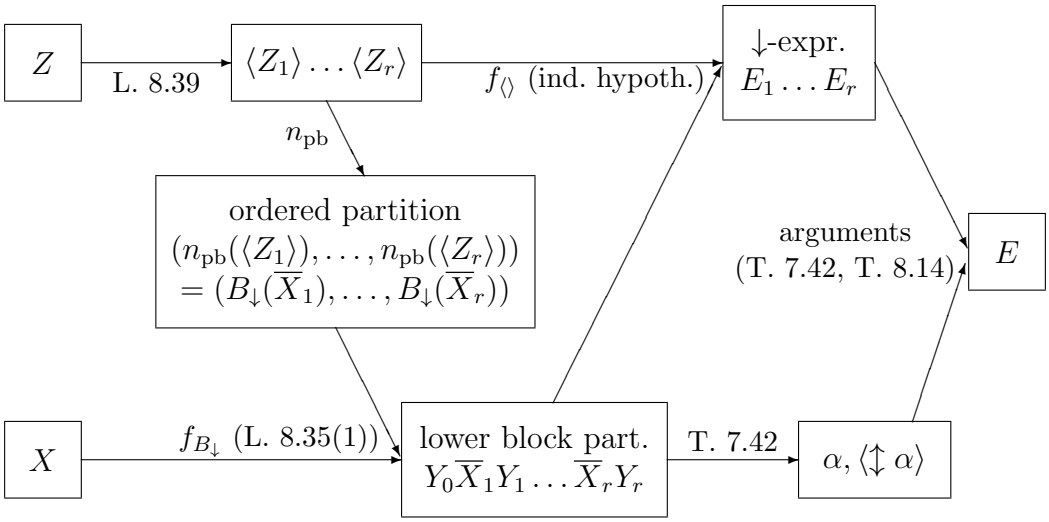


Figure 8.4: Construction of the operator-minimal ↑-expression E denoting a nick free formal DNA molecule X, such that $f_{\langle\rangle}(E) = Z$ for a given sequence Z of $B_\downarrow(X)$ well-nested pairs of brackets (see the proof of Theorem 8.45).

We analyse what it means that $f_{\langle\rangle}(E) = Z$. By Lemma 8.1, E does not have any ↑-arguments. Let $E_1, \dots, E_r$ for some $r \geq 0$ be the ↓-arguments of E, in the order of their occurrence in E. Because E is operator-minimal, $E_1, \dots, E_r$ are minimal. In particular, they are operator-minimal. By definition,

$$Z = f_{\langle\rangle}(E) = \left\langle f_{\langle\rangle}(E_1)\right\rangle \dots \left\langle f_{\langle\rangle}(E_r)\right\rangle . \tag{8.43}$$

In this equation, by Lemma 8.44(1) and Lemma 8.38(2), for $j = 1, \dots, r$, both $f_{\langle\rangle}(E_j)$ and $\left\langle f_{\langle\rangle}(E_j)\right\rangle$ are sequences of well-nested pairs of brackets.

By Lemma 8.39, there is a unique partitioning of Z as $\langle Z_1\rangle \dots \langle Z_{r'}\rangle$ for some $r' \geq 0$,[3] such that for $j = 1, \dots, r'$, Z_j is a sequence of well-nested pairs of brackets. Because (8.43) provides such a partitioning, we must have $r' = r$ and for $j = 1, \dots, r$, $f_{\langle\rangle}(E_j) = Z_j$. Now obviously, for $j = 1, \dots, r$,

$$n_{\mathrm{pb}}(\left\langle f_{\langle\rangle}(E_j)\right\rangle) = n_{\mathrm{pb}}(\langle Z_j\rangle) \geq 1.$$

Moreover,

$$\begin{aligned} n_{\mathrm{pb}}(\left\langle f_{\langle\rangle}(E_1)\right\rangle) + \dots + n_{\mathrm{pb}}(\left\langle f_{\langle\rangle}(E_r)\right\rangle) &= n_{\mathrm{pb}}(\langle Z_1\rangle) + \dots + n_{\mathrm{pb}}(\langle Z_r\rangle) \\ &= n_{\mathrm{pb}}(Z) = p + 1 = B_\downarrow(X). \end{aligned}$$

Consequently, $(n_{\mathrm{pb}}(\left\langle f_{\langle\rangle}(E_1)\right\rangle), \dots, n_{\mathrm{pb}}(\left\langle f_{\langle\rangle}(E_r)\right\rangle)) = (n_{\mathrm{pb}}(\langle Z_1\rangle), \dots, n_{\mathrm{pb}}(\langle Z_r\rangle))$ is an ordered partition of $B_\downarrow(X)$, which is uniquely determined by Z.

We proceed with the implications of E being an operator-minimal ↑-expression denoting X. By Theorem 8.14, E is based on a lower block partitioning $\mathcal{P} =$

[3] In fact, since $p + 1 \geq 1$ and hence $Z \neq \lambda$, we have $r' \geq 1$. This is, however, not important for the proof.

$Y_0\overline{X}_1Y_1\ldots\overline{X}_rY_r$ of X, as described in Theorem 7.42, such that the $\downarrow$-arguments $E_1,\ldots,E_r$ of E denote the lower blocks $\overline{X}_1,\ldots,\overline{X}_r$, respectively.

By Lemma 7.17(2b) and Lemma 8.44(3b), for $j=1,\ldots,r$,

$$B_\downarrow(\overline{X}_j) = B_\uparrow(\overline{X}_j)+1 = n_{\mathrm{pb}}(\langle f_{\langle\rangle}(E_j)\rangle).$$

Hence,

$$f_{B_\downarrow}(\mathcal{P}) = (B_\downarrow(\overline{X}_1),\ldots,B_\downarrow(\overline{X}_r)) = (n_{\mathrm{pb}}(\langle f_{\langle\rangle}(E_1)\rangle),\ldots,n_{\mathrm{pb}}(\langle f_{\langle\rangle}(E_r)\rangle)).$$

By Lemma 8.35(1), the function $f_{B_\downarrow}$ is a bijection from the lower block partitionings of X onto the ordered partitions of $B_\downarrow(X)$. This implies that the ordered partition $(n_{\mathrm{pb}}(\langle f_{\langle\rangle}(E_1)\rangle),\ldots,n_{\mathrm{pb}}(\langle f_{\langle\rangle}(E_r)\rangle))$ uniquely determines the lower block partitioning $\mathcal{P}$ of X that E is based on.

Consider an arbitrary j with $1\leq j\leq r$. We have observed that $f_{\langle\rangle}(E_j)=Z_j$ and that E_j is an operator-minimal $\downarrow$-expression denoting (the nick free submolecule) $\overline{X}_j$. By Lemma 7.23(2), $B_\uparrow(\overline{X}_j) = B_\downarrow(\overline{X}_j)-1 \leq B_\downarrow(X)-1 = p$. Now, by the induction hypothesis, Claim 2′ is valid for $\overline{X}_j$. Hence, E_j is uniquely determined by Z_j.

We have thus proved that the lower block partitioning $Y_0\overline{X}_1Y_1\ldots\overline{X}_rY_r$ of X that E is based on, is determined by Z and that, in the construction from Theorem 7.42, also the minimal $\downarrow$-expressions $E_1,\ldots,E_r$ denoting $\overline{X}_1,\ldots,\overline{X}_r$, respectively, are determined by Z. Because, in this construction, the arguments ε_i corresponding to the substrings $Y_0,Y_1,\ldots,Y_r$ ($\mathcal{N}$-words α_i and $\updownarrow$-expressions $\langle\updownarrow\alpha_i\rangle$ for $\mathcal{N}$-words α_i) are fixed, E is completely determined by Z. We conclude that there exists at most one operator-minimal $\uparrow$-expression E denoting X, such that $f_{\langle\rangle}(E)=Z$.

In the construction we described, we did not make any assumptions on Z, apart from it being a sequence of $p+1=B_\downarrow(X)$ well-nested pairs of brackets. Hence, we can really perform the construction for the sequence Z we consider. It follows from the construction that the resulting DNA expression E is indeed an operator-minimal $\uparrow$-expression denoting X, for which

$$f_{\langle\rangle}(E) = \langle f_{\langle\rangle}(E_1)\rangle\ldots\langle f_{\langle\rangle}(E_r)\rangle = \langle Z_1\rangle\ldots\langle Z_r\rangle = Z.$$

In particular, there exists an (at least one) operator-minimal $\uparrow$-expression E denoting X with $f_{\langle\rangle}(E)=Z$.

Consequently, Claim 1′ is valid for nick free formal DNA molecules X with $B_\downarrow(X)=p+1$.

The proof of Claim 2′ for nick free formal DNA molecules X with $B_\uparrow(X)=p+1$ is analogous.

□

When we combine Theorem 8.45 with Proposition 8.41, we obtain

Corollary 8.46 *Let X be a nick free formal DNA molecule.*

1. *The number of operator-minimal* $\uparrow$*-expressions denoting* X *is* $\frac{1}{p+1}\binom{2p}{p}$*, with* $p = B_{\downarrow}(X)$.

2. *The number of operator-minimal* $\downarrow$*-expressions denoting* X *is* $\frac{1}{p+1}\binom{2p}{p}$*, with* $p = B_{\uparrow}(X)$.

Recall that, when we examined the *lengths* of $\uparrow$-expressions E denoting a formal DNA molecule X, the value $B_{\downarrow}(X)$ played an important role. For example, it occurs in the lower bound for $|E|$ from Theorem 6.31(1), a lower bound that is achieved by minimal and operator-minimal $\uparrow$-expressions.

The value $B_{\downarrow}(X)$ shows up, again, in the expression for the *number* of operator-minimal $\uparrow$-expressions denoting a nick free formal DNA molecule X, which is given above.

The connection between the length of an operator-minimal $\uparrow$-expression and the number of such DNA expressions is constituted by the occurrences of operators $\uparrow$ and $\downarrow$. On the one hand, they determine the length of an operator-minimal DNA expression denoting X (given that the number of occurrences of $\updownarrow$ is fixed); on the other hand, they determine the number of pairs of brackets in the corresponding sequence of well-nested pairs of brackets. Indeed, by Corollary 8.17(1), the number of occurrences of operators $\uparrow$ and $\downarrow$ (together) in an operator-minimal $\uparrow$-expression depends (only) on $B_{\downarrow}(X)$ (cf. the proof of Lemma 8.44(3a)).

In Lemma 8.30, we could not specify values for *all* numbers of (operator-)minimal DNA expressions denoting certain types of expressible formal DNA molecules. We now can:

Corollary 8.47 *Let* X *be an expressible formal DNA molecule.*

1. *If* X *is double-complete, then*

$$\begin{aligned} n_{min\uparrow}(X) &= 0, \\ n_{opermin\uparrow}(X) &= 1, \\ n_{min\downarrow}(X) &= 0, \\ n_{opermin\downarrow}(X) &= 1, \\ n_{min\updownarrow}(X) = n_{opermin\updownarrow}(X) &= 1 \text{ and} \\ n_{min}(X) &= 1. \end{aligned}$$

2. *If* X *is nick free, contains at least one single-stranded component and* $B_{\uparrow}(X) = B_{\downarrow}(X) = p$ *for some* $p \geq 1$*, then*

$$\begin{aligned} n_{min\uparrow}(X) = n_{opermin\uparrow}(X) &= \frac{1}{p+1}\binom{2p}{p}, \\ n_{min\downarrow}(X) = n_{opermin\downarrow}(X) &= \frac{1}{p+1}\binom{2p}{p}, \\ n_{min\updownarrow}(X) = n_{opermin\updownarrow}(X) &= 0 \qquad \text{and} \\ n_{min}(X) &= \frac{2}{p+1}\binom{2p}{p}. \end{aligned}$$

3. *If X is nick free, $B_\uparrow(X) = p_1$ and $B_\downarrow(X) = p_2$ for some p_1 and p_2 with $p_1 > p_2 \geq 0$, then*

$$\begin{aligned} n_{min\uparrow}(X) = n_{opermin\uparrow}(X) &= \frac{1}{p_2+1}\binom{2p_2}{p_2}, \\ n_{min\downarrow}(X) &= 0, \\ n_{opermin\downarrow}(X) &= \frac{1}{p_1+1}\binom{2p_1}{p_1}, \\ n_{min\updownarrow}(X) = n_{opermin\updownarrow}(X) &= 0 \quad \textit{and} \\ n_{min}(X) &= \frac{1}{p_2+1}\binom{2p_2}{p_2}. \end{aligned}$$

4. *If X is nick free, $B_\uparrow(X) = p_1$ and $B_\downarrow(X) = p_2$ for some p_1 and p_2 with $p_2 > p_1 \geq 0$, then*

$$\begin{aligned} n_{min\uparrow}(X) &= 0, \\ n_{opermin\uparrow}(X) &= \frac{1}{p_2+1}\binom{2p_2}{p_2}, \\ n_{min\downarrow}(X) = n_{opermin\downarrow}(X) &= \frac{1}{p_1+1}\binom{2p_1}{p_1}, \\ n_{min\updownarrow}(X) = n_{opermin\updownarrow}(X) &= 0 \quad \textit{and} \\ n_{min}(X) &= \frac{1}{p_1+1}\binom{2p_1}{p_1}. \end{aligned}$$

5. *If X contains at least one lower nick letter, then let $Z_1 {}_\triangle Z_2 {}_\triangle \ldots {}_\triangle Z_m$ for some $m \geq 2$ be the nick free decomposition of X, and let for $h = 1, \ldots, m$, $p_h = B_\downarrow(Z_h)$.*

$$\begin{aligned} n_{min\uparrow}(X) = n_{opermin\uparrow}(X) &= \frac{1}{p_1+1}\binom{2p_1}{p_1} \times \cdots \times \frac{1}{p_m+1}\binom{2p_m}{p_m}, \\ n_{min\downarrow}(X) = n_{opermin\downarrow}(X) &= 0, \\ n_{min\updownarrow}(X) &= 0 \quad \textit{and} \\ n_{min}(X) &= \frac{1}{p_1+1}\binom{2p_1}{p_1} \times \cdots \times \frac{1}{p_m+1}\binom{2p_m}{p_m}. \end{aligned}$$

   (a) *If X does not contain any single-stranded component, then*

$$n_{opermin\updownarrow}(X) = |Z_1| \times |Z_m|.$$

   (b) *If X contains at least one single-stranded component, then*

$$n_{opermin\updownarrow}(X) = 0.$$

6. *If X contains at least one upper nick letter, then let $Z_1 {}^\triangledown Z_2 {}^\triangledown \ldots {}^\triangledown Z_m$ for some $m \geq 2$ be the nick free decomposition of X, and let for $h = 1, \ldots, m$, $p_h = B_\uparrow(Z_h)$.*

$$\begin{aligned} n_{min\uparrow}(X) = n_{opermin\uparrow}(X) &= 0, \\ n_{min\downarrow}(X) = n_{opermin\downarrow}(X) &= \frac{1}{p_1+1}\binom{2p_1}{p_1} \times \cdots \times \frac{1}{p_m+1}\binom{2p_m}{p_m}, \\ n_{min\updownarrow}(X) &= 0 \quad \textit{and} \\ n_{min}(X) &= \frac{1}{p_1+1}\binom{2p_1}{p_1} \times \cdots \times \frac{1}{p_m+1}\binom{2p_m}{p_m}. \end{aligned}$$

(a) If X does not contain any single-stranded component, then

$$n_{opermin\updownarrow}(X) = |Z_1| \times |Z_m|.$$

(b) If X contains at least one single-stranded component, then

$$n_{opermin\updownarrow}(X) = 0.$$

Note that in Claim 2, the inequality $p \geq 1$ is not restrictive. If p were equal to 0, then by Lemma 6.13(1), X would not contain any single-stranded component. Note further that in Claim 3, by Lemma 6.12(1), we must have $p_1 = p_2 + 1$. Likewise in Claim 4, $p_2 = p_1 + 1$.

Example 8.48 Let $X = \binom{\alpha_1}{-}\binom{\alpha_2}{c(\alpha_2)}\binom{-}{\alpha_3}$. Then X is nick free, contains at least one single-stranded component and $B_\uparrow(X) = B_\downarrow(X) = p = 1$. By Corollary 8.47(2), $n_{\min}(X) = \frac{2}{1+1}\binom{2}{1} = 2$. Hence, there do not exist minimal DNA expressions denoting X other than the ones given in Example 7.2.

Let X be the nick free formal DNA molecule depicted in (a.o.) Figure 7.5, for which $B_\uparrow(X) = p_1 = 4$ and $B_\downarrow(X) = p_2 = 3$. By Corollary 8.47(3), $n_{\min}(X) = \frac{1}{3+1}\binom{6}{3} = 5$.

Let X be the nick free formal DNA molecule depicted in Figure 7.6, for which $B_\uparrow(X) = B_\downarrow(X) = p = 2$. By Corollary 8.47(2), $n_{\min}(X) = \frac{2}{2+1}\binom{4}{2} = 4$. Hence, there do not exist minimal DNA expressions denoting X other than the ones we have constructed in Example 7.26.

Let X be the formal DNA molecule depicted in Figure 7.7, which contains four lower nick letters. The nick free decomposition of X is $Z_{1_\triangle}Z_{2_\triangle}Z_{3_\triangle}Z_{4_\triangle}Z_5$ for the submolecules $Z_1, \ldots, Z_5$ from (7.20). We have $p_1 = B_\downarrow(Z_1) = 1$, $p_2 = B_\downarrow(Z_2) = 2$, $p_3 = B_\downarrow(Z_3) = 0$, $p_4 = B_\downarrow(Z_4) = 0$ and $p_5 = B_\downarrow(Z_5) = 1$. By Corollary 8.47(5),

$$\begin{aligned} n_{\min}(X) &= \tfrac{1}{1+1}\binom{2}{1} \times \tfrac{1}{2+1}\binom{4}{2} \times \tfrac{1}{0+1}\binom{0}{0} \times \tfrac{1}{0+1}\binom{0}{0} \times \tfrac{1}{1+1}\binom{2}{1} \\ &= 1 \times 2 \times 1 \times 1 \times 1 = 2. \end{aligned}$$

■

Recurrence relation for the number of operator-minimal ↑-expressions and ↓-expressions

In the foregoing, we have determined the number of operator-minimal ↑-expressions (or ↓-expressions) denoting a certain nick free formal DNA molecule by establishing a bijection between such DNA expressions and sequences of well-nested pairs of brackets. We could have chosen another way to achieve the same result, and we will briefly describe this alternative now. We leave it to the reader to prove the correctness of the alternative.

By Theorem 8.14, an operator-minimal ↑-expression denoting a nick free formal DNA molecule X is based on a lower block partitioning of X, as described in Theorem 7.42. By the construction from Theorem 7.42, different lower block partitionings yield different operator-minimal ↑-expressions. Let $\mathcal{P} = Y_0\overline{X}_1Y_1 \ldots \overline{X}_rY_r$ for some $r \geq 0$ be a lower block

partitioning of X. A corresponding operator-minimal $\uparrow$-expression E has arguments ε_i for all components x'_i of a Y_j, and arguments $E_1, \ldots, E_r$ that are minimal DNA expressions denoting $\overline{X}_1, \ldots, \overline{X}_r$, respectively. By Corollary 7.31(1), $E_1, \ldots, E_r$ are $\downarrow$-expressions. The arguments ε_i are fixed, but an argument E_j may be any (operator-)minimal $\downarrow$-expression denoting $\overline{X}_j$. Because the choice of E_{j_1} with $1 \leq j_1 \leq r$ is independent of the choice of the other E_j's, the number of different operator-minimal $\uparrow$-expressions corresponding to $\mathcal{P}$ is

$$n_{\text{opermin}\downarrow}(\overline{X}_1) \times \cdots \times n_{\text{opermin}\downarrow}(\overline{X}_r).$$

Hence, the total number of operator-minimal $\uparrow$-expressions denoting X is

$$\sum_{\substack{\text{lower block part.} \\ Y_0\overline{X}_1Y_1 \ldots \overline{X}_rY_r \text{ of } X}} n_{\text{opermin}\downarrow}(\overline{X}_1) \times \cdots \times n_{\text{opermin}\downarrow}(\overline{X}_r).$$

Analogously, an operator-minimal $\downarrow$-expression denoting a nick free formal DNA molecule X' is based on an upper block partitioning of X', different upper block partitionings of X' yield different operator-minimal $\downarrow$-expressions, and the total number of operator-minimal $\downarrow$-expressions denoting X' is

$$\sum_{\substack{\text{upper block part.} \\ Y'_0\overline{X}'_1Y'_1 \ldots \overline{X}'_rY'_r \text{ of } X'}} n_{\text{opermin}\uparrow}(\overline{X}'_1) \times \cdots \times n_{\text{opermin}\uparrow}(\overline{X}'_r).$$

As we have seen in Lemma 8.35, each lower block partitioning $\mathcal{P} = Y_0\overline{X}_1Y_1 \ldots \overline{X}_rY_r$ of X can be identified with the ordered partition $(t_1, \ldots, t_r) = f_{B_\downarrow}(\mathcal{P}) = (B_\downarrow(\overline{X}_1), \ldots, B_\downarrow(\overline{X}_r))$ of $B_\downarrow(X)$, and each upper block partitioning $\mathcal{P}'$ of X' can be identified with the ordered partition $f_{B_\uparrow}(\mathcal{P}')$ of $B_\uparrow(X')$.

Now, we can prove by induction on $B_\downarrow(X)$ and $B_\uparrow(X')$ that the number of operator-minimal $\uparrow$-expressions denoting X only depends on $B_\downarrow(X)$, that the number of operator-minimal $\downarrow$-expressions denoting X' only depends on $B_\uparrow(X')$, that these numbers are equal if $B_\downarrow(X) = B_\uparrow(X') = p$, say $C(p)$, and that

$$C(p) = \begin{cases} 1 & \text{if } p = 0, \\ \displaystyle\sum_{\substack{\text{ordered partitions} \\ (t_1, \ldots, t_r) \text{ of } p}} C(t_1 - 1) \times \cdots \times C(t_r - 1) & \text{if } p \geq 1.^4 \end{cases} \tag{8.44}$$

When we recall that, by Lemma 8.39, each sequence of p well-nested pairs of brackets has a unique partitioning as $\langle Z_1 \rangle \ldots \langle Z_r \rangle$ for some $r \geq 0$ and sequences of well-nested pairs of brackets $Z_1, \ldots, Z_r$, we find that recurrence relation (8.44) is also applicable to the number of sequences of p well-nested pairs of brackets. Hence, indeed for every $p \geq 0$, $C(p)$ is equal to this number, which is the Catalan number $\frac{1}{p+1}\binom{2p}{p}$.

Recurrence relation (8.44) is also related to trees. Consider an arbitrary ordered, rooted tree with $p+1$ nodes for some $p \geq 0$. This tree consists of a root and $r \geq 0$ ordered subtrees. Each of the subtrees contains at least one node and together they contain p nodes. Hence, the respective numbers of nodes in the subtrees form an ordered partition $(t_1, \ldots, t_r)$ of p. Now it is not hard to see that the *number* of different ordered, rooted trees

[4] This summation would also give the right value 1 for $p = 0$. For the sake of clearness, however, we mention the case $p = 0$ separately.

with $p+1$ nodes satisfies recurrence relation (8.44). Indeed, the number of these trees is another well-known combinatorial interpretation of the Catalan numbers $\frac{1}{p+1}\binom{2p}{p}$ (see [Stanley, 2015, Exercise 6]).

Chapter 9

An Algorithm for Minimality

In Chapter 7, we have described how to construct a minimal DNA expression denoting a given formal DNA molecule. In Chapter 8, we have derived a characterization of minimal DNA expressions (Theorem 8.26).

Now, given an arbitrary DNA expression E, we can use the characterization to check whether or not it is minimal. If it is not, then, in order to save space, we may wish to replace it by an equivalent, minimal DNA expression, i.e., a minimal DNA expression with the same semantics. An indirect way to achieve such a minimal DNA expression consists of first determining the semantics $\mathcal{S}(E)$, and then using the applicable construction(s) from Chapter 7.

In this chapter, we follow a different, more elegant approach. We describe an algorithm to rewrite E into an equivalent, minimal DNA expression. This algorithm executes local string manipulations on E directly, based on violations of the properties in the characterization. It does not refer to the underlying semantics $\mathcal{S}(E)$, at all. Step by step, the DNA expression obtains all six properties from Lemma 8.22.

In Section 9.1, we describe the algorithm and prove its correctness. We illustrate the different steps in the algorithm by example DNA expressions, which are DNA subexpressions of a single, large DNA expression. In Section 9.2, we systematically work out the algorithm for this DNA expression as a whole. We will see that although the individual steps have small, local effects on (the structure of) the DNA expression, the total effect of the algorithm may be huge.

The description of the algorithm in Section 9.1 is not entirely complete. In particular, we sometimes say that certain arguments of a DNA expression must be considered 'in some order'. Because the actual order used does not matter for the correctness of the algorithm, we do not specify one. In addition, at other places, we consider or select certain types of arguments of a DNA expression, but we do not specify how to find these types of arguments. We fill in such implementation details and analyse the complexity of the algorithm in Section 9.3. Finally, in Section 9.4, we relate the time spent by the algorithm on actual rewriting steps to the resulting decrease of the length of the DNA expression.

9.1 The algorithm and its correctness

In this section, we describe an algorithm for rewriting an arbitrary DNA expression E into an equivalent, minimal DNA expression. This algorithm is recursive: we first construct equivalent, minimal DNA expressions for the expression-arguments of E. The resulting

expression-arguments have the six properties from Lemma 8.22. We use this in the second phase, where we construct a DNA expression E' that is equivalent to E and has these properties itself. By Theorem 8.26, E' must be minimal.

Note that this second phase is not trivial. If the arguments of a DNA expression E are minimal, then E itself may be far from minimal. For example, if E is an $\updownarrow$-expression with an expression-argument, then by Property ($\mathcal{D}_{\text{Min}}$.1), E cannot be minimal, even if this expression-argument is minimal. Similarly, if E is an $\uparrow$-expression with $\uparrow$-arguments, then by Property ($\mathcal{D}_{\text{Min}}$.2), E cannot be minimal, even if the $\uparrow$-arguments are minimal.

Another important issue is that Properties ($\mathcal{D}_{\text{Min}}$.3)–($\mathcal{D}_{\text{Min}}$.6) are more restrictive for inner occurrences of operators $\uparrow$ and $\downarrow$ than for the outermost operator of a DNA expression. For example, by Property ($\mathcal{D}_{\text{Min}}$.3), an occurrence of $\uparrow$ may only have a single $\mathcal{N}$-word-argument α, if it is the outermost operator. Also, by Properties ($\mathcal{D}_{\text{Min}}$.4) and ($\mathcal{D}_{\text{Min}}$.6), an inner occurrence of an operator $\uparrow$ must be alternating, whereas an outermost operator $\uparrow$ may have consecutive expression-arguments. Finally, by Properties ($\mathcal{D}_{\text{Min}}$.5) and ($\mathcal{D}_{\text{Min}}$.6), the first (or last) argument of an inner occurrence of $\uparrow$ cannot be a $\downarrow$-argument, whereas this is possible for an outermost operator $\uparrow$. Of course, there are analogous differences between an inner occurrence of $\downarrow$ and an outermost operator $\downarrow$.

Now suppose that E is a DNA expression and that E_i is a minimal $\uparrow$-argument of E. When we view E_i by itself, then its outermost operator $\uparrow_0$ may have, for example, consecutive expression-arguments. However, when we view E_i as an argument of E, then $\uparrow_0$ is an inner occurrence of $\uparrow$, which implies that it should be alternating.

Consequently, after we have (recursively) rewritten the expression-arguments of a DNA expression E into equivalent, minimal expression-arguments, we may still have to perform a number of rewriting steps to make E minimal itself. We can see this in Figure 9.1, where we give the pseudo-code of a recursive function `MakeMinimal`, which implements the algorithm.

The description of the function contains four instructions in a style like

$$\text{substitute } E \text{ by a minimal DNA expression } E' \text{ satisfying } E' \equiv E\text{; (proc. } \ldots\text{)}$$

These instructions will be worked out in detail in Sections 9.1.1–9.1.3, by the procedures mentioned between the brackets. We prove that both the general description of the algorithm and all procedures are correct, i.e., that they indeed produce the type of DNA expression specified, with the right semantics.

When we have an instruction of the above form, there may be many different minimal DNA expressions E' that satisfy the equivalence. Different choices may result in different outcomes of the algorithm. At this point, it does not matter which DNA expression we take. We will prove that regardless of the choice we make, the overall algorithm is correct. As we work out the procedures, however, we will see that we do not just make a random choice. For a given DNA expression E, we *systematically construct* a DNA expression E' that satisfies the requirements.

Note that an instruction of the above form bears a notion of semantics in it. The new DNA expression E' must satisfy $E' \equiv E$, i.e., its semantics must be equal to $\mathcal{S}(E)$. We use such formulations, to be able to prove the correctness of the general algorithm without knowing the 'implementation' of the procedures. Again, as we work out the procedures, we will see that we merely perform local string manipulations on the DNA expression, based on its properties as a string. Hence, the complete, detailed algorithm does not refer to the semantics of the DNA expressions involved, at all.

```
1.   MakeMinimal (E)
       // recursively rewrites an arbitrary DNA expression E
       // into an equivalent, minimal DNA expression
2.   {
3.     if (E is an ↕-expression)
4.     then if (the argument of E is a DNA expression E_1)
5.          then MakeMinimal (E_1);
                   // we proceed with the new (minimal) version of E_1
6.               if (E_1 is an ↕-expression)
7.               then substitute E by E_1;                          (D_Min.1)
8.               else    // E_1 is an ↑-expression or a ↓-expression
9.                    substitute E by a minimal DNA expression E'
                       satisfying E' ≡ E; (proc. Make↕ExprMinimal) (D_Min.1)
10.              fi
11.         fi

12.    else   // E is an ↑-expression or a ↓-expression;
              // without loss of generality, assume it is
              // an ↑-expression
13.         for all expression-arguments E_i of E (in some order)
14.         do  MakeMinimal (E_i);
15.         od
              // we proceed with the new (minimal) expression-arguments E_i
16.         for all ↓-arguments E_i of E (in some order)
17.         do  if (E_i is not alternating)
18.              then substitute E_i in E by a minimal, nick free
                        DNA expression E'_i satisfying E'_i ≡_▽ E_i;
                        (proc. Denickify)                           (D_Min.4)
19.              fi
20.         od
              // we proceed with the new expression-arguments
21.         for all ↓-arguments E_i of E (in some order)
22.         do  if (the first argument or the last argument of E_i
                    is an ↑-argument)
23.              then substitute E_i in E by a minimal ↑-expression E'_i
                       satisfying E'_i ≡ E_i; (proc. RotateToMinimal) (D_Min.5)
24.              fi
25.         od
              // we proceed with the new expression-arguments
26.         for all ↑-arguments E_i = ⟨↑ ε_{i,1} ... ε_{i,n_i}⟩ of E (in some order)
27.         do  substitute E_i in E by its arguments ε_{i,1} ... ε_{i,n_i};   (D_Min.2)
28.         od

29.         if (E has only one argument ε_1)
30.         then if (ε_1 is a DNA expression E_1)
31.              then substitute E by E_1;                          (D_Min.3)
32.              fi
33.         else   // E has at least two arguments
34.              if (E is alternating and both its first argument
                   and its last argument are ↓-arguments)
35.              then substitute E by a minimal ↓-expression E'
                       satisfying E' ≡ E; (proc. RotateToMinimal) (D_Min.6)
36.              fi
37.         fi
38.    fi
39.  }
```

Figure 9.1: Pseudo-code of the recursive function `MakeMinimal`.

We want to emphasize that (additional) recursive calls of `MakeMinimal` itself would not be appropriate to obtain the minimal DNA expressions E' or E'_i that we need in the four instructions involved. We really need specialized procedures. For each of the instructions, we explain now why this is the case.

For the substitution in line 9, we need to find a minimal DNA expression E' satisfying $E' \equiv E$. Although this is exactly what the function `MakeMinimal` is meant for, a recursive call `MakeMinimal`(E) would not work at this point. It would trigger an infinite sequence of recursive calls of the function, with the same argument E.

The minimal DNA expression E'_i that we substitute in line 18 is not equivalent to E_i. As follows from Corollary 8.11, E_i contains nicks, whereas E'_i must be nick free. Because the function `MakeMinimal` yields an *equivalent*, minimal DNA expression, it is not applicable. Apart from that, it would not make sense to call the function here, because we have just done so in line 14.

In line 23, we do not just need *any* equivalent, minimal DNA expression, but we need one of a particular type: an $\uparrow$-expression E'_i for a $\downarrow$-expression E_i. `MakeMinimal` does not make this distinction. In fact, as a result of lines 13-20, the $\downarrow$-expression E_i is minimal already. As we will see later, `MakeMinimal`(E_i) would simply yield E_i. It would never produce the desired $\uparrow$-expression.

Although the situation in line 35 looks similar, the actual problem is more serious. Just like in line 9, a call `MakeMinimal`(E) there would start an infinite sequence of recursive calls, with the same argument E.

Each substitution in the function `MakeMinimal` is justified by the violation of a particular property from Lemma 8.22. Such a violation implies that the DNA expression is not (yet) minimal. In the pseudo-code, we indicate the properties involved. We briefly discuss the relation between the different substitutions and the properties violated.

Assume that the DNA expression E is an $\updownarrow$-expression. Then `MakeMinimal` only rewrites E, if its argument is a DNA expression E_1 (in lines 5–11), i.e., not if it is an $\mathcal{N}$-word α_1. This is justified by Theorem 7.5: an $\updownarrow$-expression E with an expression-argument is not minimal. Indeed, such a DNA expression violates (at least) Property ($\mathcal{D}_{\text{Min}}$.1), and thus needs to be rewritten. On the other hand, an $\updownarrow$-expression E with an $\mathcal{N}$-word-argument is minimal already, and there is no reason to rewrite it.

There is not such a clear distinction for $\uparrow$-expressions and $\downarrow$-expressions. If E is an $\uparrow$-expression or a $\downarrow$-expression which is minimal already, then we do execute lines 13–37. However, in Theorem 9.12, we will see that, also in that case, in fact nothing happens.

We consider the action of `MakeMinimal` for an $\uparrow$-expression E. First of all, we recursively rewrite the expression-arguments E_i of E into equivalent, minimal DNA expressions. In the second for-loop, we substitute $\downarrow$-arguments of E which are not alternating. Let E_i be such a $\downarrow$-argument. Because E_i makes E violate Property ($\mathcal{D}_{\text{Min}}$.4), we indeed have reason to rewrite this expression-argument. By (the analogue for $\downarrow$-expressions of) Corollary 8.11(2) and (3), $\mathcal{S}(E_i)$ contains upper nick letters. Since these upper nick letters are removed by the outermost operator $\uparrow$ of E anyway, it does not hurt to substitute E_i by a nick free version E'_i. That is what we do in this loop.

In the third for-loop, we substitute $\downarrow$-arguments E_i for which either the first argument of the last argument is an $\uparrow$-argument. Such $\downarrow$-arguments cause a violation of Property ($\mathcal{D}_{\text{Min}}$.5). If the $\uparrow$-expression E has $\uparrow$-arguments E_i, then it violates Property ($\mathcal{D}_{\text{Min}}$.2). Therefore, in the last for-loop, we substitute such arguments.

In line 31, we have an $\uparrow$-expression E with one argument, which is an expression-

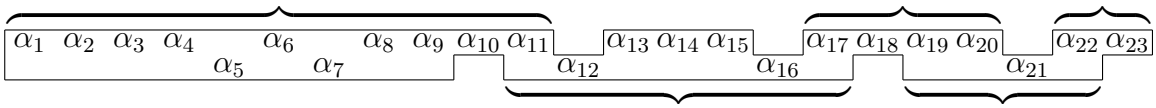


Figure 9.2: The formal DNA molecule X denoted by the DNA expression E_1^* in (9.1), with primitive upper blocks and primitive lower blocks indicated.

argument E_1. Hence, E violates Property ($\mathcal{D}_{\text{Min}}$.3). As we will see in the proof of Theorem 9.17, E_1 is nick free. This implies that the outermost operator $\uparrow$ of E does not have any effect on the semantics, and $E = \langle\uparrow E_1\rangle \equiv E_1$. Therefore, we can safely substitute E by E_1.

Finally, in line 35, we deal with a violation of Property ($\mathcal{D}_{\text{Min}}$.6).

We illustrate the different steps in the algorithm by a number of examples. All these examples are derived from the following DNA expression:

$$\begin{aligned} E_1^* = \big\langle\downarrow \big\langle\downarrow\ \big\langle\uparrow \langle\updownarrow \langle\downarrow\ \langle\updownarrow \langle\uparrow \alpha_1 \langle\updownarrow \langle\updownarrow \alpha_2\rangle\rangle \alpha_3 \langle\downarrow \langle\updownarrow \alpha_4\rangle \alpha_5\rangle\rangle\rangle\ \langle\updownarrow \alpha_6\rangle\ \alpha_7\ \big\rangle\big\rangle \\ \langle\downarrow \langle\updownarrow \alpha_8\rangle \langle\uparrow \langle\updownarrow \alpha_9\rangle \alpha_{10} \langle\updownarrow \alpha_{11}\rangle\rangle \alpha_{12} \langle\updownarrow \alpha_{13}\rangle\rangle\ \langle\updownarrow \alpha_{14}\rangle \\ \langle\downarrow \langle\updownarrow \alpha_{15}\rangle \alpha_{16} \langle\uparrow \langle\updownarrow \alpha_{17}\rangle \alpha_{18}\rangle\rangle\ \langle\uparrow \langle\updownarrow \alpha_{19}\rangle \langle\updownarrow \alpha_{20}\rangle\rangle\ \big\rangle\ \big\rangle \\ \langle\uparrow \langle\downarrow \alpha_{21}\rangle\rangle\ \langle\uparrow \langle\updownarrow \alpha_{22}\rangle \alpha_{23}\rangle\ \big\rangle\,, \end{aligned} \tag{9.1}$$

where $\alpha_1, \ldots, \alpha_{23}$ are arbitrary $\mathcal{N}$-words. It denotes the formal DNA molecule X depicted in Figure 9.2. We use the notation E_1^* to clearly distinguish the DNA expression as a whole from the parameter E of (a recursive call of) `MakeMinimal` and the expression-arguments E_i. We will use the notation E_i^* also in a more general setting, to denote the input to algorithms on DNA expressions like `MakeMinimal` and to denote the resulting output.

The DNA expression E_1^* from (9.1) is far from minimal. In fact, the only property from Lemma 8.22 that it has, is Property ($\mathcal{D}_{\text{Min}}$.6). It requires many steps to rewrite E_1^* into an equivalent, minimal DNA expression. Both for the general description of the algorithm and for each of the procedures, we select some of these steps as an illustration.

We start with examples of the substitutions that are carried out in `MakeMinimal`, as it is described in Figure 9.1. As said before, the substitutions in lines 9, 18, 23 and 35 are phrased in terms of the semantics of the DNA expressions involved, simply because we do not know yet how the procedures that are mentioned there are implemented. Therefore, in the corresponding examples, we also refer to these semantics. Later, however, when we work out the procedures and consider examples of their usage, we will see that the semantics does not play any explicit role. Hence, as desired, the algorithm merely performs string manipulations, based on syntactic properties of the DNA expressions.

Moreover, there may be more than one DNA expression E' or E_i' that satisfy the (semantic) conditions in lines 9, 18, 23 and 35. If this is the case in an example, we give all possible DNA expressions. Recall, however, that the procedures that are mentioned in these lines, systematically construct a particular DNA expression E' or E_i' for a given E or E_i.

Example 9.1 Let $E = \langle\updownarrow \langle\updownarrow \alpha_2\rangle\rangle$. E is an $\updownarrow$-expression, for which $\mathcal{S}(E) = \binom{\alpha_2}{c(\alpha_2)}$. The argument of E is the minimal $\updownarrow$-expression $E_1 = \langle\updownarrow \alpha_2\rangle$. Hence, E violates Property ($\mathcal{D}_{\text{Min}}$.1). According to line 7 of `MakeMinimal`, E is substituted by $E_1 = \langle\updownarrow \alpha_2\rangle$. Indeed, E_1 is a minimal DNA expression satisfying $E_1 \equiv E$. ■

By Theorem 7.5, each minimal $\updownarrow$-expression E_1 is of the form $\langle\updownarrow \alpha_2\rangle$ for an $\mathcal{N}$-word α_2. Hence, apart from the particular $\mathcal{N}$-word α_2, this example is the only possibility in line 7.

Example 9.2 Let

$$E = \langle\updownarrow \langle\uparrow \alpha_1 \langle\updownarrow \alpha_2\rangle \alpha_3 \langle\downarrow \langle\updownarrow \alpha_4\rangle \alpha_5\rangle\rangle\rangle\,.$$

E is an $\updownarrow$-expression, for which

$$\mathcal{S}(E) = \tbinom{\alpha_1\alpha_2\alpha_3\alpha_4 c(\alpha_5)}{c(\alpha_1\alpha_2\alpha_3\alpha_4)\alpha_5}.$$

The argument of E is a minimal $\uparrow$-expression E_1. Hence, E violates Property ($\mathcal{D}_{\mathrm{Min}}$.1). In line 9 of `MakeMinimal`, we substitute E by a minimal DNA expression E' that satisfies $E' \equiv E$. By Theorem 7.5, there exists exactly one such DNA expression: $E' = \langle\updownarrow \alpha_1\alpha_2\alpha_3\alpha_4 c(\alpha_5)\rangle$. ■

We now consider the case that E is an $\uparrow$-expression. The minimal, nick free DNA expression E_i' that we substitute for the non-alternating $\downarrow$-argument E_i of E in line 18, may be again a $\downarrow$-expression, but it may also be an $\uparrow$-expression or an $\updownarrow$-expression. We give two examples covering these three possibilities.

Example 9.3 Let

$$\begin{aligned} E = \big\langle\uparrow\ &\langle\downarrow \langle\updownarrow \alpha_1\alpha_2\alpha_3\alpha_4 c(\alpha_5)\rangle \langle\updownarrow \alpha_6 c(\alpha_7)\rangle\rangle \\ &\langle\downarrow \langle\updownarrow \alpha_8\rangle \langle\uparrow \langle\updownarrow \alpha_9\rangle \alpha_{10} \langle\updownarrow \alpha_{11}\rangle\rangle \alpha_{12} \langle\updownarrow \alpha_{13}\rangle\rangle\ \ \langle\updownarrow \alpha_{14}\rangle \\ &\langle\downarrow \langle\updownarrow \alpha_{15}\rangle \alpha_{16} \langle\uparrow \langle\updownarrow \alpha_{17}\rangle \alpha_{18}\rangle\rangle\ \ \langle\uparrow \langle\updownarrow \alpha_{19}\rangle \langle\updownarrow \alpha_{20}\rangle\rangle\ \big\rangle\,, \end{aligned}$$

for which

$$\begin{aligned} X = \mathcal{S}(E)\ \ = \ \ & \tbinom{\alpha_1\alpha_2\alpha_3\alpha_4 c(\alpha_5)\alpha_6 c(\alpha_7)}{c(\alpha_1\alpha_2\alpha_3\alpha_4)\alpha_5 c(\alpha_6)\alpha_7} \vartriangle \tbinom{\alpha_8\alpha_9}{c(\alpha_8\alpha_9)} \tbinom{\alpha_{10}}{-} \tbinom{\alpha_{11}}{c(\alpha_{11})} \tbinom{-}{\alpha_{12}} \cdot \\ & \tbinom{\alpha_{13}}{c(\alpha_{13})} \vartriangle \tbinom{\alpha_{14}}{c(\alpha_{14})} \vartriangle \tbinom{\alpha_{15}}{c(\alpha_{15})} \tbinom{-}{\alpha_{16}} \tbinom{\alpha_{17}}{c(\alpha_{17})} \tbinom{\alpha_{18}}{-} \tbinom{\alpha_{19}}{c(\alpha_{19})} \vartriangle \tbinom{\alpha_{20}}{c(\alpha_{20})}. \end{aligned}$$

All expression-arguments of E are minimal. The first argument of E is

$$E_1 = \langle\downarrow \langle\updownarrow \alpha_1\alpha_2\alpha_3\alpha_4 c(\alpha_5)\rangle \langle\updownarrow \alpha_6 c(\alpha_7)\rangle\rangle\,,$$

which is not alternating and for which

$$\mathcal{S}(E_1) = \tbinom{\alpha_1\alpha_2\alpha_3\alpha_4 c(\alpha_5)}{c(\alpha_1\alpha_2\alpha_3\alpha_4)\alpha_5} \triangledown \tbinom{\alpha_6 c(\alpha_7)}{c(\alpha_6)\alpha_7}.$$

Hence, E_1 makes E violate Property ($\mathcal{D}_{\mathrm{Min}}$.4). E_1 is not nick free. If E_1' is a nick free DNA expression satisfying $E_1' \equiv_{\triangledown} E_1$, then

$$\mathcal{S}(E_1') = \tbinom{\alpha_1\alpha_2\alpha_3\alpha_4 c(\alpha_5)\alpha_6 c(\alpha_7)}{c(\alpha_1\alpha_2\alpha_3\alpha_4)\alpha_5 c(\alpha_6)\alpha_7}.$$

By Theorem 7.5, there is exactly one minimal DNA expression with this semantics:

$$E_1' = \langle\updownarrow \alpha_1\alpha_2\alpha_3\alpha_4 c(\alpha_5)\alpha_6 c(\alpha_7)\rangle\,,$$

which is an $\updownarrow$-expression. ■

Example 9.4 Let

$$E = \big\langle\uparrow \langle\updownarrow \alpha_1\alpha_2\alpha_3\alpha_4 c(\alpha_5)\alpha_6 c(\alpha_7)\rangle \\ \langle\downarrow \langle\updownarrow \alpha_8\rangle \langle\uparrow \langle\updownarrow \alpha_9\rangle \alpha_{10} \langle\updownarrow \alpha_{11}\rangle\rangle \alpha_{12} \langle\updownarrow \alpha_{13}\rangle\rangle \ \langle\updownarrow \alpha_{14}\rangle \\ \langle\downarrow \langle\updownarrow \alpha_{15}\rangle \alpha_{16} \langle\uparrow \langle\updownarrow \alpha_{17}\rangle \alpha_{18}\rangle\rangle \ \langle\uparrow \langle\updownarrow \alpha_{19}\rangle \langle\updownarrow \alpha_{20}\rangle\rangle \big\rangle .$$

This is the result when we substitute the first $\downarrow$-argument E_1 of the $\uparrow$-expression E from Example 9.3 by the corresponding $\updownarrow$-expression E_1'. The second argument of E is

$$E_2 = \langle\downarrow \langle\updownarrow \alpha_8\rangle \langle\uparrow \langle\updownarrow \alpha_9\rangle \alpha_{10} \langle\updownarrow \alpha_{11}\rangle\rangle \alpha_{12} \langle\updownarrow \alpha_{13}\rangle\rangle ,$$

which is not alternating and for which

$$\mathcal{S}(E_2) = \tbinom{\alpha_8}{c(\alpha_8)}^{\triangledown}\tbinom{\alpha_9}{c(\alpha_9)}\tbinom{\alpha_{10}}{-}\tbinom{\alpha_{11}}{c(\alpha_{11})}\tbinom{-}{\alpha_{12}}\tbinom{\alpha_{13}}{c(\alpha_{13})}.$$

Hence, E_2 makes E violate Property ($\mathcal{D}_{\text{Min}}$.4). E_2 is not nick free. If E_2' is a nick free DNA expression satisfying $E_2' \equiv_{\triangledown} E_2$ and $X_2' = \mathcal{S}(E_2')$, then

$$X_2' = \tbinom{\alpha_8\alpha_9}{c(\alpha_8\alpha_9)}\tbinom{\alpha_{10}}{-}\tbinom{\alpha_{11}}{c(\alpha_{11})}\tbinom{-}{\alpha_{12}}\tbinom{\alpha_{13}}{c(\alpha_{13})}.$$

We have $B_{\uparrow}(X_2') = B_{\downarrow}(X_2') = 1$. By Summary 8.16(2) and the recursive construction from Theorem 7.24, there are two different minimal DNA expressions denoting X_2':

$$E_2' = \langle\uparrow \langle\updownarrow \alpha_8\alpha_9\rangle \alpha_{10} \langle\downarrow \langle\updownarrow \alpha_{11}\rangle \alpha_{12} \langle\updownarrow \alpha_{13}\rangle\rangle\rangle$$

and

$$E_2' = \langle\downarrow \langle\uparrow \langle\updownarrow \alpha_8\alpha_9\rangle \alpha_{10} \langle\updownarrow \alpha_{11}\rangle\rangle \alpha_{12} \langle\updownarrow \alpha_{13}\rangle\rangle .$$

In principle, in line 18, we may choose either of these minimal DNA expressions. If we choose the first one, then E_2' is an $\uparrow$-expression. If we choose the second one, then E_2' is a $\downarrow$-expression. ■

Note that in the second for-loop (in lines 16–20) of `MakeMinimal`, we only substitute the $\downarrow$-arguments that are not alternating. We ignore the non-alternating $\uparrow$-arguments (if these are present) there. It is only in the fourth for-loop that we substitute the $\uparrow$-arguments of the $\uparrow$-expression E (whether they are alternating or not). It would not be very useful to do this earlier in the function, because the first three for-loops may introduce new $\uparrow$-arguments.

The $\downarrow$-arguments we substitute in the third for-loop (in lines 21–25) may have been introduced *in* the second for-loop, but they may also have been arguments of E from *before* that loop. We consider examples of both possibilities now.

Example 9.5 Let

$$E = \big\langle\uparrow \langle\updownarrow \alpha_1\alpha_2\alpha_3\alpha_4 c(\alpha_5)\alpha_6 c(\alpha_7)\rangle \\ \langle\downarrow \langle\uparrow \langle\updownarrow \alpha_8\alpha_9\rangle \alpha_{10} \langle\updownarrow \alpha_{11}\rangle\rangle \alpha_{12} \langle\updownarrow \alpha_{13}\rangle\rangle \ \langle\updownarrow \alpha_{14}\rangle \\ \langle\downarrow \langle\updownarrow \alpha_{15}\rangle \alpha_{16} \langle\uparrow \langle\updownarrow \alpha_{17}\rangle \alpha_{18}\rangle\rangle \ \langle\uparrow \langle\updownarrow \alpha_{19}\rangle \langle\updownarrow \alpha_{20}\rangle\rangle \big\rangle .$$

This is the result when we substitute the second argument E_2 of the $\uparrow$-expression E from Example 9.4 by the corresponding $\downarrow$-expression E_2'. The (new) second argument of E is

$$E_2 = \langle\downarrow \langle\uparrow \langle\updownarrow \alpha_8\alpha_9\rangle \alpha_{10} \langle\updownarrow \alpha_{11}\rangle\rangle \alpha_{12} \langle\updownarrow \alpha_{13}\rangle\rangle .$$

The first argument of E_2 is an $\uparrow$-argument. Hence, E_2 makes E violate Property ($\mathcal{D}_{\text{Min}}$.5). As we have seen in Example 9.4, there is exactly one minimal $\uparrow$-expression E_2' satisfying $E_2' \equiv E_2$:

$$E_2' = \langle\uparrow \langle\updownarrow \alpha_8\alpha_9\rangle\, \alpha_{10}\, \langle\downarrow \langle\updownarrow \alpha_{11}\rangle\, \alpha_{12}\, \langle\updownarrow \alpha_{13}\rangle\rangle\rangle\,.$$

■

Example 9.6 Let

$$\begin{aligned} E = \big\langle\uparrow\ & \langle\updownarrow \alpha_1\alpha_2\alpha_3\alpha_4 c(\alpha_5)\alpha_6 c(\alpha_7)\rangle \\ & \langle\uparrow \langle\updownarrow \alpha_8\alpha_9\rangle\, \alpha_{10}\, \langle\downarrow \langle\updownarrow \alpha_{11}\rangle\, \alpha_{12}\, \langle\updownarrow \alpha_{13}\rangle\rangle\rangle\ \ \langle\updownarrow \alpha_{14}\rangle \\ & \langle\downarrow \langle\updownarrow \alpha_{15}\rangle\, \alpha_{16}\, \langle\uparrow \langle\updownarrow \alpha_{17}\rangle\, \alpha_{18}\rangle\rangle\ \ \langle\uparrow \langle\updownarrow \alpha_{19}\rangle\, \langle\updownarrow \alpha_{20}\rangle\rangle\ \big\rangle\,. \end{aligned}$$

This is the result when we substitute the second argument E_2 of the $\uparrow$-expression E from Example 9.5 by the corresponding $\uparrow$-expression E_2'. The fourth argument of E is

$$E_4 = \langle\downarrow \langle\updownarrow \alpha_{15}\rangle\, \alpha_{16}\, \langle\uparrow \langle\updownarrow \alpha_{17}\rangle\, \alpha_{18}\rangle\rangle\,,$$

for which

$$X_4 = \mathcal{S}(E_4) = \tbinom{\alpha_{15}}{c(\alpha_{15})} \tbinom{-}{\alpha_{16}} \tbinom{\alpha_{17}}{c(\alpha_{17})} \tbinom{\alpha_{18}}{-}.$$

The last argument of E_4 is an $\uparrow$-argument. Hence, E_4 makes E violate Property ($\mathcal{D}_{\text{Min}}$.5). We have $B_\uparrow(X_4) = B_\downarrow(X_4) = 1$. By Summary 8.16(2) and the recursive construction from Theorem 7.24, there is exactly one minimal $\uparrow$-expression E_4' with $\mathcal{S}(E_4') = X_4$, i.e., with $E_4' \equiv E_4$:

$$E_4' = \langle\uparrow \langle\downarrow \langle\updownarrow \alpha_{15}\rangle\, \alpha_{16}\, \langle\updownarrow \alpha_{17}\rangle\rangle\, \alpha_{18}\rangle\,.$$

■

When we substitute the argument E_4 of the $\uparrow$-expression E from the last example by the corresponding $\uparrow$-expression E_4', E does not have any $\downarrow$-argument left. This is not necessarily the case after the first three for-loops. E may still have (minimal) $\downarrow$-arguments then. These must be alternating (i.e., nick free), and by Properties ($\mathcal{D}_{\text{Min}}$.1) and ($\mathcal{D}_{\text{Min}}$.2), both the first argument and the last argument of such a $\downarrow$-argument must be either an $\mathcal{N}$-word α, or an $\updownarrow$-expression $\langle\updownarrow \alpha\rangle$ for an $\mathcal{N}$-word α.

Recall that the substitutions in the third for-loop of `MakeMinimal` were justified by violations of Property ($\mathcal{D}_{\text{Min}}$.5) by an inner occurrence of the operator $\downarrow$. Both in Example 9.5 and in Example 9.6, we have obtained an $\uparrow$-expression E_i', whose first argument or last argument is a $\downarrow$-argument. In other words: the outermost operator $\uparrow$ of E_i' (which is an inner occurrence in E) also violates Property ($\mathcal{D}_{\text{Min}}$.5). As we will see shortly, this is not really a problem. It is, however, good to realize that this is always the case:

Lemma 9.7 *Let E_i' be a minimal $\uparrow$-expression that is substituted for a $\downarrow$-argument E_i in the third for-loop of the function* `MakeMinimal`*. Then either the first argument, or the last argument of E_i' is a $\downarrow$-argument.*

Proof: As we observed before, either the $\downarrow$-argument E_i has been an argument of E from *before* the second for-loop, or it has been substituted for another $\downarrow$-argument *in* this second for-loop. In both cases, E_i is minimal.

Let $X_i = \mathcal{S}(E_i) = \mathcal{S}(E_i')$. By Summary 8.16, the fact that there exists both a minimal $\downarrow$-expression E_i and a minimal $\uparrow$-expression E_i' denoting X_i, implies that X_i is nick free, contains at least one single-stranded component and $B_\uparrow(X_i) = B_\downarrow(X_i)$. Both E_i and E_i' satisfy the construction from Theorem 7.24.

Now, when we apply Corollary 7.33(2) to E_i', we conclude that either the first argument, or the last argument of E_i' is a $\downarrow$-argument. □

The fourth for-loop (in lines 26–28) of the function `MakeMinimal` deals with violations of Property ($\mathcal{D}_{\text{Min}}$.2). However, it also resolves the violations of Properties ($\mathcal{D}_{\text{Min}}$.4) and ($\mathcal{D}_{\text{Min}}$.5) by the outermost operators of (new) $\uparrow$-arguments E_i. We proceed with an example of the substitutions carried out in that loop.

Example 9.8 Let

$$E = \langle\uparrow \langle\updownarrow \alpha_1\alpha_2\alpha_3\alpha_4 c(\alpha_5)\alpha_6 c(\alpha_7)\rangle \\ \langle\uparrow \langle\updownarrow \alpha_8\alpha_9\rangle\, \alpha_{10}\, \langle\downarrow \langle\updownarrow \alpha_{11}\rangle\, \alpha_{12}\, \langle\updownarrow \alpha_{13}\rangle\rangle\rangle\ \langle\updownarrow \alpha_{14}\rangle \\ \langle\uparrow \langle\downarrow \langle\updownarrow \alpha_{15}\rangle\, \alpha_{16}\, \langle\updownarrow \alpha_{17}\rangle\rangle\, \alpha_{18}\rangle\ \langle\uparrow \langle\updownarrow \alpha_{19}\rangle\, \langle\updownarrow \alpha_{20}\rangle\rangle\, \rangle\,.$$

This is the result when we substitute the fourth argument E_4 of the $\uparrow$-expression E from Example 9.6 by the corresponding $\uparrow$-expression E_4'. The $\uparrow$-expression E has three $\uparrow$-arguments. Hence, it violates Property ($\mathcal{D}_{\text{Min}}$.2). In lines 26–28, we substitute these three $\uparrow$-arguments by their respective arguments. The result is

$$E = \langle\uparrow \langle\updownarrow \alpha_1\alpha_2\alpha_3\alpha_4 c(\alpha_5)\alpha_6 c(\alpha_7)\rangle \\ \langle\updownarrow \alpha_8\alpha_9\rangle\ \alpha_{10}\ \langle\downarrow \langle\updownarrow \alpha_{11}\rangle\, \alpha_{12}\, \langle\updownarrow \alpha_{13}\rangle\rangle\ \langle\updownarrow \alpha_{14}\rangle \\ \langle\downarrow \langle\updownarrow \alpha_{15}\rangle\, \alpha_{16}\, \langle\updownarrow \alpha_{17}\rangle\rangle\ \alpha_{18}\ \langle\updownarrow \alpha_{19}\rangle\ \langle\updownarrow \alpha_{20}\rangle\, \rangle\,.$$

■

The function `MakeMinimal` ends with an if-then-else construction (in lines 29–37). Depending on the properties of the DNA expression E resulting from the for-loops, the if-then-else construction does or does not yield one more modification of the DNA expression. We conclude this series of examples with one example where the DNA expression remains the same, and two examples (one very simple and one more involved) where it is modified in the if-then-else construction.

Example 9.9 Let

$$E = \langle\uparrow \langle\updownarrow \alpha_1\alpha_2\alpha_3\alpha_4 c(\alpha_5)\alpha_6 c(\alpha_7)\rangle \\ \langle\updownarrow \alpha_8\alpha_9\rangle\ \alpha_{10}\ \langle\downarrow \langle\updownarrow \alpha_{11}\rangle\, \alpha_{12}\, \langle\updownarrow \alpha_{13}\rangle\rangle\ \langle\updownarrow \alpha_{14}\rangle \\ \langle\downarrow \langle\updownarrow \alpha_{15}\rangle\, \alpha_{16}\, \langle\updownarrow \alpha_{17}\rangle\rangle\ \alpha_{18}\ \langle\updownarrow \alpha_{19}\rangle\ \langle\updownarrow \alpha_{20}\rangle\, \rangle\,.$$

This is the result of Example 9.8. E has more than one argument and is not alternating. According to line 34, E is not modified any further. E denotes the formal DNA molecule X from Example 9.3. Hence, it is indeed equivalent to the original DNA expression. Moreover, it is easily verified that E has all six properties from Lemma 8.22 and thus is minimal. ■

Example 9.10 Let $E = \langle\uparrow \langle\downarrow \alpha_{21}\rangle\rangle$, for which $\mathcal{S}(E) = \binom{-}{\alpha_{21}}$. The only argument of the $\uparrow$-expression E is the $\downarrow$-expression $E_1 = \langle\downarrow \alpha_{21}\rangle$. Hence, E violates Property ($\mathcal{D}_{\text{Min}}$.3). E_1 is an alternating $\downarrow$-argument, whose only argument is the $\mathcal{N}$-word α_{21}. According to line 31, E is substituted by E_1. By Summary 8.16(4) and the construction from Theorem 7.24(2), this is the only minimal DNA expression denoting $\mathcal{S}(E)$. ■

Example 9.11 Let

$$E = \big\langle\downarrow \big\langle\uparrow \langle\updownarrow \alpha_1\alpha_2\alpha_3\alpha_4 c(\alpha_5)\alpha_6 c(\alpha_7)\alpha_8\alpha_9\rangle\ \alpha_{10} \\ \langle\downarrow \langle\updownarrow \alpha_{11}\rangle\, \alpha_{12}\, \langle\updownarrow \alpha_{13}\alpha_{14}\alpha_{15}\rangle\, \alpha_{16}\, \langle\updownarrow \alpha_{17}\rangle\rangle\ \alpha_{18}\ \langle\updownarrow \alpha_{19}\alpha_{20}\rangle\ \big\rangle \\ \alpha_{21}\ \langle\uparrow \langle\updownarrow \alpha_{22}\rangle\, \alpha_{23}\rangle\ \big\rangle,$$

which, like DNA expression E_1^* from (9.1), denotes the formal DNA molecule X from Figure 9.2. The $\downarrow$-expression E has three arguments: two minimal, alternating $\uparrow$-arguments, separated by an $\mathcal{N}$-word α_{21}. In particular, E itself is also alternating, it violates Property ($\mathcal{D}_{\text{Min}}$.6) and line 35 of the function `MakeMinimal` is applicable.

The formal DNA molecule X is nick free. As indicated in Figure 9.2, $B_\uparrow(X) = 3$ and $B_\downarrow(X) = 2$. Hence, by Summary 8.16(3) and the recursive construction from Theorem 7.24, there are two different minimal DNA expressions E' denoting X, i.e., with $E' \equiv E$:

$$E' = \big\langle\uparrow \langle\updownarrow \alpha_1\alpha_2\alpha_3\alpha_4 c(\alpha_5)\alpha_6 c(\alpha_7)\alpha_8\alpha_9\rangle\ \alpha_{10} \\ \langle\downarrow \langle\updownarrow \alpha_{11}\rangle\, \alpha_{12}\, \langle\updownarrow \alpha_{13}\alpha_{14}\alpha_{15}\rangle\, \alpha_{16}\, \langle\updownarrow \alpha_{17}\rangle\rangle\ \alpha_{18} \\ \langle\downarrow \langle\updownarrow \alpha_{19}\alpha_{20}\rangle\, \alpha_{21}\, \langle\updownarrow \alpha_{22}\rangle\rangle\ \alpha_{23}\ \big\rangle$$

and

$$E' = \big\langle\uparrow \langle\updownarrow \alpha_1\alpha_2\alpha_3\alpha_4 c(\alpha_5)\alpha_6 c(\alpha_7)\alpha_8\alpha_9\rangle\ \alpha_{10} \\ \big\langle\downarrow \langle\updownarrow \alpha_{11}\rangle\, \alpha_{12}\, \langle\updownarrow \alpha_{13}\alpha_{14}\alpha_{15}\rangle\, \alpha_{16}\ \langle\uparrow \langle\updownarrow \alpha_{17}\rangle\, \alpha_{18}\, \langle\updownarrow \alpha_{19}\alpha_{20}\rangle\rangle\ \alpha_{21}\, \langle\updownarrow \alpha_{22}\rangle\ \big\rangle \\ \alpha_{23}\ \big\rangle.$$

In principle, in line 35 of `MakeMinimal`, we may choose either of these minimal DNA expressions. ■

The recursive function `MakeMinimal` may be applied to a DNA expression E that is minimal already. Before we prove the correctness of the function in general, we examine its effect in this particular case. On page 240, we already observed that the function `MakeMinimal` does nothing to a minimal $\updownarrow$-expression. We now consider arbitrary minimal DNA expressions.

Theorem 9.12 *Let E be a minimal DNA expression. When the function* `MakeMinimal` *is applied to E, it does not perform any rewriting step.*

Hence, `MakeMinimal` leaves every minimal DNA expression unchanged. The only thing the function does for such a DNA expression, is checking some conditions and performing recursive calls for the DNA subexpressions.

Proof: By induction on the number p of operators occurring in E.

- If $p = 1$, then E is $\langle\updownarrow \alpha_1\rangle$, $\langle\uparrow \alpha_1\rangle$ or $\langle\downarrow \alpha_1\rangle$ for an $\mathcal{N}$-word α_1. Indeed, these DNA expressions are minimal.

  It is easily verified that in each of these cases, `MakeMinimal` leaves E unchanged. In particular, for an $\uparrow$-expression $E = \langle\uparrow \alpha_1\rangle$, nothing happens in the four for-loops, because E has no expression-arguments.

- Let $p \geq 1$, and suppose that `MakeMinimal` leaves all minimal DNA expressions containing at most p operators unchanged (induction hypothesis). Now let E be a minimal DNA expression that contains $p + 1$ operators.

Because, by Theorem 7.5, a minimal $\updownarrow$-expression contains only one operator, E has to be an $\uparrow$-expression or a $\downarrow$-expression. Without loss of generality, assume it is an $\uparrow$-expression.

Because E is minimal, each expression-argument of E is also minimal. Because an expression-argument E_i has at most p operators, by the induction hypothesis, the recursive calls in the first for-loop of `MakeMinimal` do not yield any rewriting step.

By Property ($\mathcal{D}_{\text{Min}}$.4), each proper $\downarrow$-subexpression of E is alternating. In particular, each $\downarrow$-argument of E is alternating. Hence, the second for-loop of `MakeMinimal` does not yield any rewriting step, either.

By Property ($\mathcal{D}_{\text{Min}}$.5), E does not have any proper $\downarrow$-subexpression, for which either the first argument or the last argument is an $\uparrow$-argument. In particular, E does not have any $\downarrow$-argument for which this is the case. Hence, the third for-loop of the function does not yield any rewriting step, either.

By Property ($\mathcal{D}_{\text{Min}}$.2), no occurrence of $\uparrow$ in E has an $\uparrow$-argument. In particular, the outermost operator $\uparrow$ of E has no $\uparrow$-argument. Hence, the fourth for-loop of the function does not yield any rewriting step, either.

We finally consider the if-then-else construction at the end of the function. If E has only one argument, then by Property ($\mathcal{D}_{\text{Min}}$.3), this is an $\mathcal{N}$-word α. Indeed, in this case, E is not rewritten.

If on the other hand, E has at least two arguments, then by Property ($\mathcal{D}_{\text{Min}}$.6), either E has consecutive expression-arguments, or its first argument is an $\mathcal{N}$-word α or an $\updownarrow$-expression $\langle\updownarrow\alpha\rangle$ for an $\mathcal{N}$-word α, or its last argument is an $\mathcal{N}$-word α or an $\updownarrow$-expression $\langle\updownarrow\alpha\rangle$ for an $\mathcal{N}$-word α. In each of the three cases, the condition in line 34 of the function becomes false, and E is not rewritten.

□

We will come back to the effect of `MakeMinimal` on minimal DNA expressions, at the end of Section 9.4. Note that for many formal DNA molecules, there exists more than one minimal DNA expression, see, e.g., Corollary 8.47. When we apply `MakeMinimal` to different equivalent, minimal DNA expressions, the outputs (which equal the inputs) are also different. This implies in particular that `MakeMinimal` does not always produce the same minimal DNA expression, when it is applied to different, equivalent DNA expressions. To state it formally:

Corollary 9.13 *Let E_1 and E_2 be equivalent DNA expressions. When we apply the function* `MakeMinimal` *to E_1 and E_2, the resulting minimal DNA expressions are not necessarily equal.*

Hence, `MakeMinimal` does not produce some kind of a 'normal form version' of its argument. We study a (minimal) normal form for DNA expressions in Chapter 10 and Chapter 11.

We now focus on a particular aspect of `MakeMinimal`, which is important for its correctness. This aspect will come back in the implementation of line 18, in procedure `Denickify`.

In lines 23 and 35 of `MakeMinimal`, we need a minimal $\uparrow$-expression E'_i or a minimal $\downarrow$-expression E' that is equivalent to a certain DNA expression. Obviously, for each DNA expression, there exist one or more equivalent, minimal DNA expressions. For certain

DNA expressions, however, there does not exist an equivalent, minimal $\uparrow$-expression or an equivalent, minimal $\downarrow$-expression, simply because all minimal DNA expressions are of another type, see Summary 8.16. We prove that under certain conditions, the desired equivalent, minimal $\uparrow$-expression or $\downarrow$-expression does exist.

Lemma 9.14 *Let E be an $\uparrow$-expression denoting a certain formal DNA molecule X.*

If E is nick free, has Properties ($\mathcal{D}_{Min}$.3)–($\mathcal{D}_{Min}$.5), and either the first argument or the last argument of E (or both arguments) is a $\downarrow$-argument, then there exists a minimal $\downarrow$-expression E' satisfying $E' \equiv E$.

Proof: Assume that E is nick free, has Properties ($\mathcal{D}_{\text{Min}}$.3)–($\mathcal{D}_{\text{Min}}$.5), and either the first argument or the last argument of E (or both arguments) is a $\downarrow$-argument.

Without loss of generality, assume that the first argument of E is a $\downarrow$-argument E_1. Let $X_1 = \mathcal{S}(E_1)$. By Property ($\mathcal{D}_{\text{Min}}$.4) and Lemma 5.8, X_1 is nick free. Hence, the semantics X of the $\uparrow$-expression E starts with $\nu^+(X_1) = X_1$.

By Property ($\mathcal{D}_{\text{Min}}$.3), E_1 has at least two arguments. By Property ($\mathcal{D}_{\text{Min}}$.5), the first argument of E_1 is either an $\mathcal{N}$-word α_1, or an $\updownarrow$-expression $\langle\updownarrow \alpha_1\rangle$ for an $\mathcal{N}$-word α_1. In the latter case, by Property ($\mathcal{D}_{\text{Min}}$.4), the second argument of E_1 is an $\mathcal{N}$-word α_2. In both cases, $X_1 = \mathcal{S}(E_1)$ has at least one single-stranded component, and the first single-stranded component of X_1 is a lower component. But then also $X = \mathcal{S}(E)$ has at least one single-stranded component, and its first single-stranded component is a lower component.

By Lemma 6.13(3b) and (3d), $B_\downarrow(X) \geq B_\uparrow(X)$. Hence, by Theorem 7.24(2), there exists a minimal $\downarrow$-expression E' denoting X, i.e., a minimal $\downarrow$-expression E' satisfying $E' \equiv E$. □

If an $\uparrow$-expression E is alternating and has Property ($\mathcal{D}_{\text{Min}}$.4), then each occurrence of $\uparrow$ or $\downarrow$ in E is alternating. By Lemma 5.8, E is nick free. But then we also have

Corollary 9.15 *Let E be an $\uparrow$-expression denoting a certain formal DNA molecule X.*

If E is alternating, has Properties ($\mathcal{D}_{Min}$.3)–($\mathcal{D}_{Min}$.5), and either the first argument or the last argument of E (or both arguments) is a $\downarrow$-argument, then there exists a minimal $\downarrow$-expression E' satisfying $E' \equiv E$.

Note that the DNA expression E in Lemma 9.14 and Corollary 9.15 is not necessarily operator-minimal. Hence, Corollary 8.11 is not applicable: the adjectives 'nick free' and 'alternating' are not equivalent, here. There exist DNA expressions E for which Lemma 9.14 is applicable, but Corollary 9.15 is not, because they are nick free but not alternating.

Example 9.16 Consider the $\uparrow$-expression

$$E = \langle\uparrow \langle\downarrow \alpha_1 \langle\updownarrow \alpha_2\rangle\rangle \langle\uparrow \alpha_3 \langle\updownarrow \alpha_4\rangle\rangle\rangle$$

for $\mathcal{N}$-words $\alpha_1, \ldots, \alpha_4$. E is nick free, but not alternating, E has Properties ($\mathcal{D}_{\text{Min}}$.3)–($\mathcal{D}_{\text{Min}}$.5), and its first argument is a $\downarrow$-argument. The formal DNA molecule denoted by E is $X = \binom{-}{\alpha_1}\binom{\alpha_2}{c(\alpha_2)}\binom{\alpha_3}{-}\binom{\alpha_4}{c(\alpha_4)}$, for which $B_\uparrow(X) = B_\downarrow(X) = 1$. It follows from Summary 8.16(2) and the construction from Theorem 7.24 that the (only) minimal $\downarrow$-expression E' denoting X is

$$E' = \langle\downarrow \alpha_1 \langle\uparrow \langle\updownarrow \alpha_2\rangle \alpha_3 \langle\updownarrow \alpha_4\rangle\rangle\rangle\,.$$

■

We now prove that the global algorithm is correct:

Theorem 9.17 *Let E_1^* be an arbitrary DNA expression, and let E_2^* be the result of applying the function* `MakeMinimal` *to E_1^*.*

1. `MakeMinimal` *is well defined.*
2. *The string E_2^* is a minimal DNA expression satisfying $E_2^* \equiv E_1^*$.*
3. *If E_1^* is an $\updownarrow$-expression, then there exists exactly one minimal DNA expression that is equivalent to E_1^*. In particular, in this case, E_2^* is independent of choices made in the procedures* `Make`$\updownarrow$`ExprMinimal`, `Denickify` *and* `RotateToMinimal`.

Note that in Claim 3, the procedures `Denickify` and `RotateToMinimal` are not irrelevant for an $\updownarrow$-expression E_1^*. If E_1^* has $\uparrow$-subexpressions or $\downarrow$-subexpressions, then the procedures may be used in the recursive call for such a DNA subexpression.

In the proof, we will see that the equivalence $E_2^* \equiv E_1^*$ in Claim 2 relies heavily on two types of observations. First, sometimes an operator occurring in a DNA expression E does not contribute to the semantics of E, at all. We can as well skip such an operator. This is the case in lines 7, 27 and 31 of `MakeMinimal`. Second, by Lemma 5.11, when we substitute an expression-argument E_i of E by an equivalent expression-argument, E remains a DNA expression with the same semantics. This is the case in lines 5, 14 and 23 of `MakeMinimal`, and the substitution in line 18 is not too different.

Proof: We first discuss some aspects of the well-definedness of `MakeMinimal`. For each DNA expression E, there exists at least one equivalent, minimal DNA expression E'. In principle, we could use the constructions mentioned in Summary 8.16 to obtain E'. Hence, the substitution in line 9 of `MakeMinimal` is well defined.

For the substitution in line 18, let us consider an *arbitrary* DNA expression E_i, with $X_i = \mathcal{S}(E_i)$. By definition, the formal DNA molecule $X_i' = \nu(X_i)$ is nick free and satisfies $X_i' \equiv_\triangledown X_i$. By Theorem 5.5, X_i' is expressible. In particular, there exists at least one minimal DNA expression E_i' denoting X_i', i.e., satisfying $E_i' \equiv_\triangledown E_i$. This holds for an arbitrary DNA expression E_i. Then it certainly holds for the $\downarrow$-expression E_i we consider in line 18. Hence, the substitution in this line is also well defined.

Now the only instructions in `MakeMinimal` that are not obviously well defined, are the substitutions in lines 23 and 35. These lines presuppose the existence of a minimal $\uparrow$-expression or a minimal $\downarrow$-expression, which is equivalent to a given DNA expression. The proof that these minimal DNA expressions indeed exist, exploits some properties of the DNA expression E which emerge in the proof of Claim 2. Therefore, we combine the proofs of Claim 1 and Claim 2.

1, 2. We prove these claims by induction on the number p of operators occurring in E_1^*.
• If $p = 1$, then E_1^* is $\langle\updownarrow \alpha_1\rangle$, $\langle\uparrow \alpha_1\rangle$ or $\langle\downarrow \alpha_1\rangle$ for an $\mathcal{N}$-word α_1. These DNA expressions are minimal already. Hence, by Theorem 9.12, `MakeMinimal` does not perform any rewriting step on E_1^*. In particular, the substitutions in lines 23 and 35 are not executed. As a result, $E_2^* = E_1^*$.

Indeed, in this case, `MakeMinimal` is well defined, and E_2^* is a minimal DNA expression satisfying $E_2^* \equiv E_1^*$.
• Let $p \geq 1$, and suppose that both claims are valid for all DNA expressions containing at most p operators (induction hypothesis). Now let E_1^* be a DNA expression that contains $p + 1$ operators.

– If E_1^* is an $\updownarrow$-expression, then its argument must be a DNA expression E_1, with p operators. By the induction hypothesis, the recursive call in line 5 yields a minimal DNA expression E_1' satisfying $E_1' \equiv E_1$. By Lemma 5.11, the resulting (overall) string $E = \langle \updownarrow E_1' \rangle$ is a DNA expression, which satisfies $E = \langle \updownarrow E_1' \rangle \equiv \langle \updownarrow E_1 \rangle = E_1^*$.

  We subsequently execute lines 6–10 of the function. In accordance with the pseudo-code, we use E_1 to denote the (new and minimal) expression-argument E_1' of E.

  If E_1 is an $\updownarrow$-expression, then by Theorem 7.5, $E_1 = \langle \updownarrow \alpha_1 \rangle$ for an $\mathcal{N}$-word α_1, and $E = \langle \updownarrow E_1 \rangle = \langle \updownarrow \langle \updownarrow \alpha_1 \rangle \rangle$. Applying the same operator $\updownarrow$ to the same argument for a second time, does not change the result. In this case, we execute line 7, yielding $E_2^* = E_1 = \langle \updownarrow \alpha_1 \rangle$. Indeed, this is a minimal DNA expression, which satisfies $E_2^* \equiv E = \langle \updownarrow \langle \updownarrow \alpha_1 \rangle \rangle \equiv E_1^*$.

  If, on the other hand, E_1 is an $\uparrow$-expression or a $\downarrow$-expression, then we execute line 9, yielding $E_2^* = E'$, where E' is a minimal DNA expression satisfying $E' \equiv E \equiv E_1^*$.

  In both subcases, the induction hypothesis is valid for the $\updownarrow$-expression E_1^*.

– If E_1^* is not an $\updownarrow$-expression, then without loss of generality, assume it is an $\uparrow$-expression. In this case, lines 13–37 of `MakeMinimal` are applicable. In accordance with the pseudo-code, we use E to denote the 'working DNA expression' in this part of the function. We prove that step by step, E becomes minimal.

  We first consider the effect of the **first for-loop** (lines 13–15). We prove that the following property is an invariant for this loop:

  $$E \text{ is an } \uparrow\text{-expression satisfying } E \equiv E_1^*. \tag{9.2}$$

  Note that, because E_1^* contains at least two operators, it has at least one expression-argument. Hence, the first for-loop has at least one iteration.

  - Initially, before the first iteration of the for-loop, E is equal to E_1^*. By assumption, the property is valid then.
  - Suppose that Property (9.2) is valid before a certain iteration of the for-loop. In this iteration, we consider an expression-argument E_i of E. E_i contains at most p operators. By the induction hypothesis, the recursive call `MakeMinimal` (E_i) in line 14 yields a minimal DNA expression E_i' satisfying $E_i' \equiv E_i$. When we apply Lemma 5.11, we find that after substituting E_i by E_i', the (overall) string E is still a DNA expression satisfying $E \equiv E_1^*$. Of course, it is still an $\uparrow$-expression.

  After the loop, all expression-arguments of E are minimal. We proceed with the **second for-loop** (lines 16–20). We prove that the following property is an invariant for this loop:

  $$E \text{ is an } \uparrow\text{-expression satisfying } E \equiv E_1^*, \text{ and each expression-argument of } E \text{ is minimal.} \tag{9.3}$$

  By Lemma 8.22, this property implies that the expression-arguments of E have Properties $(\mathcal{D}_{\text{Min}}.1)$–$(\mathcal{D}_{\text{Min}}.6)$. Because each occurrence of the operator $\updownarrow$ in the $\uparrow$-expression E must be in such an expression-argument, Property $(\mathcal{D}_{\text{Min}}.1)$ is also valid for E itself. E does not necessarily have Properties $(\mathcal{D}_{\text{Min}}.2)$–$(\mathcal{D}_{\text{Min}}.6)$. For

example, E may be any of the DNA expressions from Table 8.1, except the first two (because they do not have Property ($\mathcal{D}_{\text{Min}}$.1)) and the fifth one (the second example for Property ($\mathcal{D}_{\text{Min}}$.3)).

- Clearly, before the first iteration of the for-loop, the property is valid.
- Suppose that Property (9.3) is valid before a certain iteration of the for-loop. Let $E = \langle\uparrow \varepsilon_1 \dots \varepsilon_n\rangle$ for some $n \geq 1$ and $\mathcal{N}$-words and DNA expressions $\varepsilon_1, \dots, \varepsilon_n$, let $X = \mathcal{S}(E)$ and for $i = 1, \dots, n$, let $X_i = \mathcal{S}^+(\varepsilon_i)$. By definition,

  $$X = \nu^+(X_1)y_1\nu^+(X_2)y_2 \dots y_{n-1}\nu^+(X_n), \tag{9.4}$$

  where for $i = 1, \dots, n-1$, $y_i = \triangle$ if $R(X_i), L(X_{i+1}) \in \mathcal{A}_\pm$ and $y_i = \lambda$ otherwise.

  In the iteration, we consider a $\downarrow$-argument E_i. If it is alternating, then E is not changed, and Property (9.3) is obviously still valid at the end of the iteration.

  Now, assume that E_i is not alternating. By (the analogue for $\downarrow$-expressions of) Corollary 8.11, $X_i = \mathcal{S}(E_i)$ contains upper nick letters.

  In line 18, we substitute E_i by a minimal, nick free DNA expression E_i' satisfying $E_i' \equiv_\triangledown E_i$. Let $X_i' = \mathcal{S}(E_i')$. Then X_i' is nick free and satisfies $X_i' \equiv_\triangledown X_i$. Hence, $X_i' = \nu(X_i)$. This is equal to $\nu^+(X_i)$, because, by Lemma 5.2(1), X_i does not contain lower nick letters.

  By Lemma 5.11, after the substitution of E_i by E_i', E is still a DNA expression. In particular, it is an $\uparrow$-expression. Moreover, because $\nu^+(X_i') = \nu^+(\nu^+(X_i)) = \nu^+(X_i)$, and by Lemma 3.11, $L(X_i') = L(X_i)$ and $R(X_i') = R(X_i)$, the semantics of E is the same before and after the substitution (see (9.4)). In particular, $E \equiv E_1^*$ after the substitution.

  Clearly, because we substitute a minimal expression-argument of E by another minimal expression-argument, each expression-argument of E is minimal after the substitution, just like before the substitution.

We conclude that indeed, Property (9.3) is an invariant for the second for-loop.

We zoom in a bit more on the effect of line 18. Here, we substitute a minimal, non-alternating $\downarrow$-argument E_i of E by a minimal, nick free DNA expression E_i'. If E_i' is again a $\downarrow$-expression, then by Corollary 8.11, E_i is alternating. This implies that by every substitution, the number of non-alternating $\downarrow$-arguments of E decreases by 1. After the second for-loop, each (remaining) $\downarrow$-argument of E is alternating. When we add this property to Property (9.3), we obtain

$$\begin{array}{l} E \text{ is an } \uparrow\text{-expression satisfying } E \equiv E_1^*, \text{ each expression-argument} \\ \text{of } E \text{ is minimal and each } \downarrow\text{-argument of } E \text{ is alternating.} \end{array} \tag{9.5}$$

Now, consider an arbitrary inner occurrence $\downarrow_1$ of $\downarrow$ in E. This is either an inner occurrence in an expression-argument of E, or the outermost operator of a $\downarrow$-argument of E. If $\downarrow_1$ is an inner occurrence in an expression-argument of E, then by Property ($\mathcal{D}_{\text{Min}}$.4) of the (minimal) expression-argument, $\downarrow_1$ is alternating. If, on the other hand, $\downarrow_1$ is the outermost operator of a $\downarrow$-argument E_i of E, then by Property (9.5), it is also alternating.

Hence, as far as the inner occurrences of $\downarrow$ are concerned, E has Property ($\mathcal{D}_{\text{Min}}$.4). However, there may be inner occurrences of $\uparrow$ in E that have consecutive expression-arguments.

We prove that line 23 of `MakeMinimal` is well defined and that Property (9.5) is an invariant for the **third for-loop** (lines 21–25).

- Clearly, before the first iteration of the for-loop, the property is valid.
- Suppose that Property (9.5) is valid before a certain iteration of the for-loop. In the iteration, we consider a $\downarrow$-argument E_i of E.

  By Property (9.5), E_i is minimal and alternating. Then by Property ($\mathcal{D}_{\text{Min}}$.6), either the first argument, or the last argument of E_i (or both) is an $\mathcal{N}$-word α or an $\updownarrow$-expression $\langle\updownarrow \alpha\rangle$ for an $\mathcal{N}$-word α. It is impossible that both arguments are $\uparrow$-arguments.

  If neither the first argument, nor the last argument of E_i is an $\uparrow$-argument, then E is not changed. Obviously, in that case, Property (9.5) is still valid at the end of the iteration.

  Now, assume that either the first argument, or the last argument of E_i is an $\uparrow$-argument. Because E_i is minimal, it has Properties ($\mathcal{D}_{\text{Min}}$.1)–($\mathcal{D}_{\text{Min}}$.6). Then by Corollary 9.15, there indeed exists a minimal $\uparrow$-expression E_i' satisfying $E_i' \equiv E_i$. In particular, line 23 of `MakeMinimal` is well defined.

  By Lemma 5.11, when we substitute E_i in E by E_i', E remains an $\uparrow$-expression with the same semantics. Moreover, after the substitution, each expression-argument of E is still minimal, and each remaining $\downarrow$-argument is the same as before and thus alternating.

Indeed, Property (9.5) is an invariant for the third for-loop. Clearly, by every substitution in line 23, the number of $\downarrow$-arguments of E for which either the first argument or the last argument is an $\uparrow$-expression decreases by 1. After the loop, there are no such $\downarrow$-arguments left. Because the remaining (minimal) $\downarrow$-arguments of E have (a.o.) Properties ($\mathcal{D}_{\text{Min}}$.1) and ($\mathcal{D}_{\text{Min}}$.2), the following, extended property is valid:

$$\begin{aligned}&E \text{ is an } \uparrow\text{-expression satisfying } E \equiv E_1^*, \text{ each expression-argument}\\ &\text{of } E \text{ is minimal, each } \downarrow\text{-argument of } E \text{ is alternating, and for each}\\ &\downarrow\text{-argument of } E,\\ &\quad\blacklozenge \text{ the first argument is either an } \mathcal{N}\text{-word } \alpha \text{ or an } \updownarrow\text{-expression}\\ &\quad\ \ \langle\updownarrow \alpha\rangle \text{ for an } \mathcal{N}\text{-word } \alpha, \text{ and}\\ &\quad\blacklozenge \text{ the last argument is either an } \mathcal{N}\text{-word } \alpha \text{ or an } \updownarrow\text{-expression}\\ &\quad\ \ \langle\updownarrow \alpha\rangle \text{ for an } \mathcal{N}\text{-word } \alpha.\end{aligned} \tag{9.6}$$

Again, consider an arbitrary inner occurrence $\downarrow_1$ of $\downarrow$ in E. If it is an inner occurrence in an expression-argument of E, then by Property ($\mathcal{D}_{\text{Min}}$.5) of this (minimal) expression-argument, the first argument of $\downarrow_1$ is either an $\mathcal{N}$-word α or an $\updownarrow$-expression $\langle\updownarrow \alpha\rangle$ for an $\mathcal{N}$-word α, and the last argument of $\downarrow_1$ is either an $\mathcal{N}$-word α or an $\updownarrow$-expression $\langle\updownarrow \alpha\rangle$ for an $\mathcal{N}$-word α. If, on the other hand, $\downarrow_1$ is the outermost operator of an expression-argument of E, then the first argument and the last argument of $\downarrow_1$ have the same property by Property (9.6).

Hence, as far as the inner occurrences of $\downarrow$ are concerned, E has Property ($\mathcal{D}_{\text{Min}}$.5). However, there may be inner occurrences of $\uparrow$ in E, for which either the first argument, or the last argument (or both) is a $\downarrow$-expression. In particular, by Lemma 9.7,

this is the case for the outermost operator of each $\uparrow$-expression E'_i that we have substituted for a $\downarrow$-argument E_i of E in the third for-loop.

We prove that Property (9.6) is an invariant for the **fourth for-loop** (lines 26–28).

- Clearly, before the first iteration of the for-loop, the property is valid.
- Suppose that Property (9.6) is valid before a certain iteration of the for-loop. In the iteration, we substitute an $\uparrow$-argument E_i of E by its arguments.

  Because E was an $\uparrow$-expression before the substitution, by Lemma 5.10, it is still a DNA expression (and in particular, an $\uparrow$-expression) with the same semantics, after the substitution. The outermost operator $\uparrow$ of E_i that we have skipped, did not really contribute to the semantics of E. Further, because the expression-arguments of the minimal DNA expression E_i are also minimal, each expression-argument of E is minimal after the substitution.

  Finally, let E'_j be a new $\downarrow$-argument of E after the substitution, i.e., a $\downarrow$-argument that used to be an argument of the $\uparrow$-expression E_i we have substituted. Because E_i is minimal, by Property ($\mathcal{D}_{\text{Min}}$.4), its $\downarrow$-argument E'_j is alternating. Moreover, by Property ($\mathcal{D}_{\text{Min}}$.5) of E_i, the first argument of E'_j is either an $\mathcal{N}$-word α, or an $\updownarrow$-expression $\langle\updownarrow \alpha\rangle$ for an $\mathcal{N}$-word α, and the last argument of E'_j is either an $\mathcal{N}$-word α, or an $\updownarrow$-expression $\langle\updownarrow \alpha\rangle$ for an $\mathcal{N}$-word α.

  Consequently, each new $\downarrow$-argument of E after the substitution has the properties required by Property (9.6). All other $\downarrow$-arguments of E after the substitution also have these properties, simply because they had them before the substitution and they have not been changed.

We also zoom in a bit more on the effect of line 27. Here, we substitute a minimal $\uparrow$-argument E_i of E by its arguments. By Property ($\mathcal{D}_{\text{Min}}$.2) of E_i, none of these arguments is an $\uparrow$-expression. This implies that by every substitution, the number of $\uparrow$-arguments of E decreases by 1. After the fourth for-loop, the $\uparrow$-expression E does not have any $\uparrow$-arguments any more.

All occurrences of $\uparrow$ in E different from the outermost operator, and all occurrences of $\downarrow$ in E occur in the expression-arguments of E. These expression-arguments are minimal. Hence, by Property ($\mathcal{D}_{\text{Min}}$.2), no occurrence of $\uparrow$ in E has an $\uparrow$-argument, and no occurrence of $\downarrow$ in E has a $\downarrow$-argument. In other words, E itself has Property ($\mathcal{D}_{\text{Min}}$.2).

Earlier in the proof, we deduced from Property (9.3) that E has Property ($\mathcal{D}_{\text{Min}}$.1). As Property (9.3) is implied by Property (9.6), it is still valid and E still has Property ($\mathcal{D}_{\text{Min}}$.1).

Later, we deduced from Property (9.5) that each inner occurrence of $\downarrow$ in E is alternating. Even later, we deduced from Property (9.6) that for each inner occurrence of $\downarrow$ in E, the first argument is an $\mathcal{N}$-word α or an $\updownarrow$-expression $\langle\updownarrow \alpha\rangle$ for an $\mathcal{N}$-word α, and the last argument is an $\mathcal{N}$-word α or an $\updownarrow$-expression $\langle\updownarrow \alpha\rangle$ for an $\mathcal{N}$-word α. As both Property (9.5) and Property (9.6) are still valid, the inner occurrences of $\downarrow$ in E still have these properties.

Now, consider an inner occurrence $\uparrow_1$ of $\uparrow$ in E. Because E does not have any $\uparrow$-arguments any more, this occurrence of $\uparrow$ must be an inner occurrence in an expression-argument E_i (in fact, in a $\downarrow$-argument E_i) of E. This expression-argument

E_i is minimal. By Property ($\mathcal{D}_{\text{Min}}$.4), $\uparrow_1$ is alternating. By Property ($\mathcal{D}_{\text{Min}}$.5), the first argument of $\uparrow_1$ is either an $\mathcal{N}$-word α or an $\updownarrow$-expression $\langle\updownarrow \alpha\rangle$ for an $\mathcal{N}$-word α, and the last argument of $\uparrow_1$ is either an $\mathcal{N}$-word α or an $\updownarrow$-expression $\langle\updownarrow \alpha\rangle$ for an $\mathcal{N}$-word α.

We conclude that E also has Property ($\mathcal{D}_{\text{Min}}$.4) and Property ($\mathcal{D}_{\text{Min}}$.5).

Note that by Property ($\mathcal{D}_{\text{Min}}$.1), Property ($\mathcal{D}_{\text{Min}}$.2) and Property (9.5), the arguments of E are $\mathcal{N}$-words α, $\updownarrow$-expressions $\langle\updownarrow \alpha\rangle$ for $\mathcal{N}$-words α, and minimal, alternating $\downarrow$-expressions. In particular, all arguments of E are nick free.

We finally analyse the **if-then-else construction** in lines 29–37 of `MakeMinimal`. We prove that for every possible case, the resulting string E_2^* is a minimal DNA expression satisfying $E_2^* \equiv E_1^*$.

- Assume that E has only one argument, and that this argument is a DNA expression E_1. Then, because E_1 is nick free, the outermost operator $\uparrow$ of E has no effect. Hence, in this case, $E_2^* = E_1 \equiv \langle\uparrow E_1\rangle = E \equiv E_1^*$. Moreover, by Property (9.3), $E_2^* = E_1$ is minimal.
- Assume that E has only one argument, and that this argument is an $\mathcal{N}$-word α_1. In this case, $E_2^* = E = \langle\uparrow \alpha_1\rangle$, which is indeed a minimal DNA expression. Moreover, $E_2^* = E \equiv E_1^*$.
- Assume that E has at least two arguments, that E is alternating and both the first argument and the last argument of E are $\downarrow$-arguments. We must establish that line 35 of `MakeMinimal` is well defined.

  We first analyse the consequences of E having at least two arguments. Because the arguments of (the $\uparrow$-expression) E fit together by upper strands, none of these arguments can be a $\downarrow$-expression $\langle\downarrow \alpha\rangle$ for an $\mathcal{N}$-word α. By Property ($\mathcal{D}_{\text{Min}}$.2) of E, none of the arguments can be an $\uparrow$-expression $\langle\uparrow \alpha\rangle$ for an $\mathcal{N}$-word α, either. Hence, by Property ($\mathcal{D}_{\text{Min}}$.3) of the minimal expression-arguments of E, each occurrence of $\uparrow$ or $\downarrow$ in such an argument has at least two arguments itself. But then each occurrence of $\uparrow$ or $\downarrow$ in E has at least two arguments. Hence, in addition to Properties ($\mathcal{D}_{\text{Min}}$.1), ($\mathcal{D}_{\text{Min}}$.2), ($\mathcal{D}_{\text{Min}}$.4) and ($\mathcal{D}_{\text{Min}}$.5), E has Property ($\mathcal{D}_{\text{Min}}$.3).

  Now by Corollary 9.15, there exists a minimal $\downarrow$-expression E' satisfying $E' \equiv E$. This implies that line 35 of `MakeMinimal` is well defined. Clearly, in this case $E_2^* = E' \equiv E \equiv E_1^*$.
- Finally, assume that E has at least two arguments, and that either E is not alternating, or the first argument of E is not a $\downarrow$-argument, or the last argument of E is not a $\downarrow$-argument. Again, because E has at least two arguments, it has Property ($\mathcal{D}_{\text{Min}}$.3).

  If the first argument of E is not a $\downarrow$-argument, then it is an $\mathcal{N}$-word α or an $\updownarrow$-expression $\langle\updownarrow \alpha\rangle$ for an $\mathcal{N}$-word α. Analogously, if the last argument of E is not a $\downarrow$-argument, then it is an $\mathcal{N}$-word α or an $\updownarrow$-expression $\langle\updownarrow \alpha\rangle$ for an $\mathcal{N}$-word α.

  In every case, E has Property ($\mathcal{D}_{\text{Min}}$.6). This implies that $E_2^* = E$ has all properties from Lemma 8.22, and thus is minimal. Moreover, $E_2^* = E \equiv E_1^*$.

This completes the proof of Claims 1 and 2.

3. Assume that E_1^* is an $\updownarrow$-expression, and let $X = \mathcal{S}(E_1^*)$. By Corollary 5.7, there exist $\mathcal{N}$-words $\alpha_1, \ldots, \alpha_m$ for some $m \geq 1$ and a nick letter $y \in \{^{\triangledown}, _{\triangle}\}$, such that

$$X = \tbinom{\alpha_1}{c(\alpha_1)} y \tbinom{\alpha_2}{c(\alpha_2)} y \ldots y \tbinom{\alpha_m}{c(\alpha_m)}.$$

Without loss of generality, assume that $y = {}_{\triangle}$.

If $m = 1$, i.e., if $X = \binom{\alpha_1}{c(\alpha_1)}$, then by Theorem 7.5, there is exactly one minimal DNA expression denoting X. If $m \geq 2$, then Lemma 8.18(2) leads us to the same conclusion.

In other words, in both cases, there is exactly one minimal DNA expression E_2^* satisfying $E_2^* \equiv E_1^*$. Hence, given the $\updownarrow$-expression E_1^*, the resulting DNA expression E_2^* is fixed. It is independent of choices made in the procedures `Make`$\updownarrow$`ExprMinimal`, `Denickify` and `RotateToMinimal`.

This completes the proof of Theorem 9.17. □

9.1.1 The procedure `Make`$\updownarrow$`ExprMinimal`

Now that we have established the correctness of the global description of the algorithm, we can start working out the implementation details. In line 9 of `MakeMinimal`, we have an $\updownarrow$-expression $E = \langle\updownarrow E_1\rangle$, for which E_1 is a minimal $\uparrow$-expression or $\downarrow$-expression. We need a procedure `Make`$\updownarrow$`ExprMinimal` to rewrite E into an equivalent, minimal DNA expression E'. In Figure 9.3, we give the pseudo-code of this procedure for the case that E_1 is an $\uparrow$-expression. The code for a $\downarrow$-expression E_1 is similar. We address one difference soon.

In the original $\updownarrow$-expression $E = \langle\updownarrow E_1\rangle$, we first apply $\uparrow$ (the outermost operator of E_1) and then $\updownarrow$ (the outermost operator of E). That is, we first combine the arguments of E_1 into one molecule, and then complement the result. The idea behind the result E' of `Make`$\updownarrow$`ExprMinimal` is just the reverse: we first complement the arguments of E_1, and then combine the results into one molecule. Of course, in the first step, we do not have to complement $\updownarrow$-arguments of E_1, as these are complemented already.

As we will see in the proof of Theorem 9.20(1), initially each argument of $\widehat{E}_1 = E_1$ is either an $\mathcal{N}$-word $\alpha_{1,i}$ or an $\updownarrow$-expression $\langle\updownarrow \alpha_{1,i}\rangle$ for an $\mathcal{N}$-word $\alpha_{1,i}$, or a $\downarrow$-expression. $\widehat{E}_1$ does not have $\uparrow$-arguments.

In the first for-loop of `Make`$\updownarrow$`ExprMinimal`, we substitute the $\downarrow$-arguments $E_{1,i}$ of $\widehat{E}_1$ by $\updownarrow$-arguments $\langle\updownarrow \alpha_{E_{1,i}}\rangle$. Recall that the $\mathcal{N}$-word $\alpha_{E_{1,i}}$ is the concatenation of all $\mathcal{N}$-words (possibly complemented) occurring in $E_{1,i}$, in the order of their occurrence, see its definition on page 97. For the moment, it is not important how exactly we determine $\alpha_{E_{1,i}}$. In the proof of Lemma 9.34, where we analyse the time complexity of `Make`$\updownarrow$`ExprMinimal`, we will describe a straightforward implementation for this.

In the second for-loop of the procedure, we combine $\mathcal{N}$-word-arguments $\alpha_{1,i}$ with preceding and/or succeeding $\updownarrow$-arguments. Indeed, at that point in the procedure, these are the only types of arguments left. Clearly, if $\alpha_{1,i}$ is the first argument of $\widehat{E}_1$, then it is not preceded by an $\updownarrow$-argument $\langle\updownarrow \alpha_{1,i-1}\rangle$ for an $\mathcal{N}$-word $\alpha_{1,i-1}$. In fact, if we assume that the $\mathcal{N}$-word-arguments are maximal $\mathcal{N}$-word occurrences in $\widehat{E}_1$,[1] then this is the only case in which $\alpha_{1,i}$ is not preceded by an $\updownarrow$-argument $\langle\updownarrow \alpha_{1,i-1}\rangle$. Under the same assumption, $\alpha_{1,i}$ is not succeeded by an $\updownarrow$-argument $\langle\updownarrow \alpha_{1,i+1}\rangle$, if and only if $\alpha_{1,i}$ is the last argument

[1] This is a very natural assumption, but is not necessary for the correctness of procedure `Make`$\updownarrow$`ExprMinimal`.

```
M↕M.1.   Make↕ExprMinimal (E)
           // rewrites an ↕-expression E = ⟨↕ E_1⟩ whose argument E_1
           // is a minimal ↑-expression, into a minimal DNA expression E'
           // satisfying E' ≡ E;
           // for a ↓-expression E_1, see the remark before Example 9.18
M↕M.2.   {
M↕M.3.     Ê_1 = E_1;
M↕M.4.     for all ↓-arguments E_{1,i} of Ê_1 (in some order)
M↕M.5.     do   substitute E_{1,i} in Ê_1 by ⟨↕ α_{E_{1,i}}⟩;
M↕M.6.     od
             // arguments of Ê_1 are 𝒩-words α_{1,i} and ↕-expressions ⟨↕ α_{1,i}⟩
M↕M.7.     for all 𝒩-word-arguments α_{1,i} of Ê_1 (in some order)
M↕M.8.     do   if (α_{1,i} is preceded by an argument ⟨↕ α_{1,i-1}⟩)
M↕M.9.          then if (α_{1,i} is succeeded by an argument ⟨↕ α_{1,i+1}⟩)
M↕M.10.              then substitute ⟨↕ α_{1,i-1}⟩ α_{1,i} ⟨↕ α_{1,i+1}⟩ in Ê_1
                            by ⟨↕ α_{1,i-1}α_{1,i}α_{1,i+1}⟩;
M↕M.11.              else substitute ⟨↕ α_{1,i-1}⟩ α_{1,i} in Ê_1 by ⟨↕ α_{1,i-1}α_{1,i}⟩;
M↕M.12.              fi
M↕M.13.         else if (α_{1,i} is succeeded by an argument ⟨↕ α_{1,i+1}⟩)
M↕M.14.              then substitute α_{1,i} ⟨↕ α_{1,i+1}⟩ in Ê_1 by ⟨↕ α_{1,i}α_{1,i+1}⟩;
M↕M.15.              else substitute α_{1,i} in Ê_1 by ⟨↕ α_{1,i}⟩;
M↕M.16.              fi
M↕M.17.         fi
M↕M.18.    od
             // Ê_1 = ⟨↑ ⟨↕ α_{1,1}⟩ ... ⟨↕ α_{1,m}⟩⟩ for some m ≥ 1
             // and 𝒩-words α_{1,1}, ..., α_{1,m}
M↕M.19.    if (m == 1)
M↕M.20.    then substitute Ê_1 by ⟨↕ α_{1,1}⟩;                    (𝒟_Min.3)
M↕M.21.    fi
M↕M.22.    E' = Ê_1;
M↕M.23.  }
```

Figure 9.3: Pseudo-code of the procedure `Make↕ExprMinimal`.

of $\widehat{E}_1$. Note that in all four cases considered in this loop, the required substitution can simply be achieved by a few insertions and/or removals of brackets and operators in the DNA expression.

After the second for-loop, each argument of $\widehat{E}_1$ is an $\updownarrow$-argument $\langle\updownarrow \alpha_{1,i}\rangle$ for an $\mathcal{N}$-word $\alpha_{1,i}$. As we will see in the proof of Theorem 9.20(1), at that point, $\widehat{E}_1$ is equivalent to the original $\updownarrow$-expression E. However, it is not necessarily minimal.

If $\widehat{E}_1 = \langle\uparrow \langle\updownarrow \alpha_{1,1}\rangle\rangle$ for an $\mathcal{N}$-word $\alpha_{1,1}$, then it violates Property ($\mathcal{D}_{\text{Min}}$.3). It is not hard to prove that this will be the case, if and only if the original, minimal $\uparrow$-expression E_1 is alternating, i.e., nick free. According to line M↕M.20, in this case, we substitute $\widehat{E}_1$ by its only argument. If on the other hand $\widehat{E}_1 = \langle\uparrow \langle\updownarrow \alpha_{1,1}\rangle \dots \langle\updownarrow \alpha_{1,m}\rangle\rangle$ for some $m \geq 2$, then it is minimal already (see Lemma 8.18(2)) and we can skip line M↕M.20.

If E_1 is not an $\uparrow$-expression, but a $\downarrow$-expression, then lines M↕M.8–M↕M.17 are a bit different. In all four cases, the new $\updownarrow$-argument does not have $\alpha_{1,i}$ as (part of) its own argument but $c(\alpha_{1,i})$. For example, if the $\mathcal{N}$-word-argument $\alpha_{1,i}$ is preceded by an $\updownarrow$-argument $\langle\updownarrow \alpha_{1,i-1}\rangle$ and succeeded by an $\updownarrow$-argument $\langle\updownarrow \alpha_{1,i+1}\rangle$, then $\langle\updownarrow \alpha_{1,i-1}\rangle\, \alpha_{1,i}\, \langle\updownarrow \alpha_{1,i+1}\rangle$ must

be substituted by $\langle\updownarrow \alpha_{1,i-1}c(\alpha_{1,i})\alpha_{1,i+1}\rangle$.

We illustrate procedure `Make↕ExprMinimal` by two examples:

Example 9.18 (cf. Example 9.2) Let

$$E = \langle\updownarrow \langle\uparrow \alpha_1 \langle\updownarrow \alpha_2\rangle \alpha_3 \langle\downarrow \langle\updownarrow \alpha_4\rangle \alpha_5\rangle\rangle\rangle\,.$$

E is an $\updownarrow$-expression, for which

$$\mathcal{S}(E) = \left(\begin{smallmatrix}\alpha_1\alpha_2\alpha_3\alpha_4c(\alpha_5)\\ c(\alpha_1\alpha_2\alpha_3\alpha_4)\alpha_5\end{smallmatrix}\right).$$

The argument E_1 of E is a minimal, alternating $\uparrow$-expression. E_1 itself has one $\downarrow$-argument, $E_{1,4} = \langle\downarrow \langle\updownarrow \alpha_4\rangle \alpha_5\rangle$. In line M↕M.5, we substitute it by $\langle\updownarrow \alpha_{E_{1,4}}\rangle = \langle\updownarrow \alpha_4c(\alpha_5)\rangle$, yielding

$$\widehat{E}_1 = \langle\uparrow \alpha_1 \langle\updownarrow \alpha_2\rangle \alpha_3 \langle\updownarrow \alpha_4c(\alpha_5)\rangle\rangle\,.$$

Subsequently, in lines M↕M.7–M↕M.18, we substitute the two $\mathcal{N}$-word-arguments α_1 and α_3 of $\widehat{E}_1$ (in some order). For both possible orders, the result is

$$\widehat{E}_1 = \langle\uparrow \langle\updownarrow \alpha_1\alpha_2\alpha_3\alpha_4c(\alpha_5)\rangle\rangle\,.$$

$\widehat{E}_1$ has $m = 1$ argument left, which is an $\updownarrow$-argument. Hence, it violates Property ($\mathcal{D}_{\text{Min}}$.3). According to line M↕M.20, E' is set to this $\updownarrow$-argument: $E' = \langle\updownarrow \alpha_1\alpha_2\alpha_3\alpha_4c(\alpha_5)\rangle$. By Theorem 7.5, E' is the only minimal DNA expression with $\mathcal{S}(E') = \mathcal{S}(E)$, i.e., with $E' \equiv E$. ■

Example 9.19 (cf. Example 9.3) Let

$$E = \langle\updownarrow \langle\downarrow \langle\updownarrow \alpha_1\alpha_2\alpha_3\alpha_4c(\alpha_5)\rangle \langle\updownarrow \alpha_6\rangle \alpha_7\rangle\rangle\,.$$

E is an $\updownarrow$-expression, for which

$$\mathcal{S}(E) = \left(\begin{smallmatrix}\alpha_1\alpha_2\alpha_3\alpha_4c(\alpha_5)\\ c(\alpha_1\alpha_2\alpha_3\alpha_4)\alpha_5\end{smallmatrix}\right) {}^{\triangledown} \left(\begin{smallmatrix}\alpha_6c(\alpha_7)\\ c(\alpha_6)\alpha_7\end{smallmatrix}\right).$$

The argument E_1 of E is a minimal, non-alternating $\downarrow$-expression. E_1 does not have $\uparrow$-arguments, but it does have an $\mathcal{N}$-word-argument α_7. In line M↕M.11, we substitute $\langle\updownarrow \alpha_6\rangle \alpha_7$ by $\langle\updownarrow \alpha_6c(\alpha_7)\rangle$, yielding

$$\widehat{E}_1 = \langle\downarrow \langle\updownarrow \alpha_1\alpha_2\alpha_3\alpha_4c(\alpha_5)\rangle \langle\updownarrow \alpha_6c(\alpha_7)\rangle\rangle\,.$$

This time, $\widehat{E}_1$ has $m = 2$ arguments left. Hence, $E' = \widehat{E}_1$. By (the analogue for upper nick letters of) Lemma 8.18(2), E' is the only minimal DNA expression with $\mathcal{S}(E') = \mathcal{S}(E)$, i.e., with $E' \equiv E$. ■

We prove that procedure `Make↕ExprMinimal` is correct, not only for the two examples we considered, but for any $\updownarrow$-expression E with a minimal $\uparrow$-argument (or $\downarrow$-argument) E_1.

Theorem 9.20 *Let $E = \langle\updownarrow E_1\rangle$ be an $\updownarrow$-expression whose argument E_1 is a minimal $\uparrow$-expression, and let E' be the result of applying procedure* `Make↕ExprMinimal` *to E.*

1. *The string E' is a minimal DNA expression satisfying $E' \equiv E$.*

2. *There exists exactly one minimal DNA expression that is equivalent to E. In particular, E' is independent of the order in which $\downarrow$-arguments are considered in line M↕M.4 and independent of the order in which $\mathcal{N}$-word-arguments are considered in line M↕M.7.*

Proof:

1. E_1 is a minimal $\uparrow$-expression. By Corollary 8.2, each argument of E_1 is either an $\mathcal{N}$-word $\alpha_{1,i}$, or an $\updownarrow$-expression $\langle\updownarrow \alpha_{1,i}\rangle$ for an $\mathcal{N}$-word $\alpha_{1,i}$, or a $\downarrow$-expression $E_{1,i}$.

   We prove that the following property of our 'working DNA expression' $\widehat{E}_1$ is an invariant for the **first for-loop**:

   $$\widehat{E}_1 \text{ is a minimal } \uparrow\text{-expression satisfying } \left\langle\updownarrow \widehat{E}_1\right\rangle \equiv E. \tag{9.7}$$

   - Initially, before the first iteration of the for-loop, the property is valid, because then $\widehat{E}_1 = E_1$ and thus $\left\langle\updownarrow \widehat{E}_1\right\rangle = \langle\updownarrow E_1\rangle = E$.
   - Suppose that Property (9.7) is valid before a certain iteration of the for-loop. Let $\widehat{E}_1 = \langle\uparrow_1 \varepsilon_{1,1}, \ldots, \varepsilon_{1,n}\rangle$ for some $n \geq 1$ and $\mathcal{N}$-words and DNA expressions $\varepsilon_{1,1}, \ldots, \varepsilon_{1,n}$, before the iteration.

     In the iteration, we substitute a $\downarrow$-argument $\varepsilon_{1,i_0} = E_{1,i_0}$ of $\widehat{E}_1$ by the $\updownarrow$-expression $\langle\updownarrow \alpha_{E_{1,i_0}}\rangle$. By Lemma 4.13(2), $L(\mathcal{S}(\langle\updownarrow \alpha_{E_{1,i_0}}\rangle)), R(\mathcal{S}(\langle\updownarrow \alpha_{E_{1,i_0}}\rangle)) \in \mathcal{A}_{\pm}$. In particular, the upper strand of the new argument covers the lower strand both to the left and to the right. Because the arguments of $\uparrow_1$ fitted together by upper strands before the substitution, they certainly do so after the substitution. Hence, $\widehat{E}_1$ is still a DNA expression after the substitution. In particular, it is an $\uparrow$-expression.

     By Theorem 8.26, $\widehat{E}_1$ has Properties ($\mathcal{D}_{\text{Min}}$.1)–($\mathcal{D}_{\text{Min}}$.6) before the substitution. It is easily verified that $\widehat{E}_1$ still has these properties, and thus is minimal, after the substitution.

     We now consider the semantics of $\left\langle\updownarrow \widehat{E}_1\right\rangle$. In order for the invariant to be valid, this semantics must be the same before and after the substitution.

     For $i = 1, \ldots, n$, let $X_{1,i} = \mathcal{S}^+(\varepsilon_{1,i})$. By the definition of the semantics of an $\updownarrow$-expression and Lemma 8.7, before the substitution,

     $$\begin{aligned}\mathcal{S}(\left\langle\updownarrow \widehat{E}_1\right\rangle) &= \kappa(X_{1,1}\ y_1\ X_{1,2}\ y_2\ \ldots\ y_{i_0-2}\ X_{1,i_0-1}\ y_{i_0-1}\cdot \\ &\qquad \mathcal{S}(E_{1,i_0})\ y_{i_0}\ X_{1,i_0+1}\ y_{i_0+1}\ \ldots\ y_{n-1}\ X_{1,n}) \\ &= \kappa(X_{1,1})\ y_1\ \kappa(X_{1,2})\ y_2\ \ldots\ y_{i_0-2}\ \kappa(X_{1,i_0-1})\ y_{i_0-1}\cdot \\ &\quad \kappa(\mathcal{S}(E_{1,i_0}))\ y_{i_0}\ \kappa(X_{1,i_0+1})\ y_{i_0+1}\ \ldots\ y_{n-1}\ \kappa(X_{1,n}),\end{aligned}$$

     where for $i = 1, \ldots, n-1$, $y_i = \triangle$ if both $\varepsilon_{1,i}$ and $\varepsilon_{1,i+1}$ are expression-arguments, and $y_i = \lambda$ otherwise.

     After the substitution of E_{1,i_0} by $\langle\updownarrow \alpha_{E_{1,i_0}}\rangle$,

     $$\begin{aligned}\mathcal{S}(\left\langle\updownarrow \widehat{E}_1\right\rangle) &= \kappa(X_{1,1}\ y_1\ X_{1,2}\ y_2\ \ldots\ y_{i_0-2}\ X_{1,i_0-1}\ y'_{i_0-1}\cdot \\ &\qquad \mathcal{S}(\langle\updownarrow \alpha_{E_{1,i_0}}\rangle)\ y'_{i_0}\ X_{1,i_0+1}\ y_{i_0+1}\ \ldots\ y_{n-1}\ X_{1,n}) \\ &= \kappa(X_{1,1})\ y_1\ \kappa(X_{1,2})\ y_2\ \ldots\ y_{i_0-2}\ \kappa(X_{1,i_0-1})\ y'_{i_0-1}\cdot \\ &\quad \binom{\alpha_{E_{1,i_0}}}{c(\alpha_{E_{1,i_0}})}\ y'_{i_0}\ \kappa(X_{1,i_0+1})\ y_{i_0+1}\ \ldots\ y_{n-1}\ \kappa(X_{1,n}),\end{aligned}$$

where the y_i's are as before, and

$$y'_{i_0-1} = \begin{cases} \triangle & \text{if both } \varepsilon_{1,i_0-1} \text{ and } \langle\updownarrow \alpha_{E_{1,i_0}}\rangle \\ & \text{are expression-arguments} \\ \lambda & \text{otherwise,} \end{cases}$$

$$y'_{i_0} = \begin{cases} \triangle & \text{if both } \langle\updownarrow \alpha_{E_{1,i_0}}\rangle \text{ and } \varepsilon_{1,i_0+1} \\ & \text{are expression-arguments} \\ \lambda & \text{otherwise.} \end{cases}$$

We must prove that

$$y_{i_0-1}\ \kappa(\mathcal{S}(E_{1,i_0}))\ y_{i_0} = y'_{i_0-1}\ \binom{\alpha_{E_{1,i_0}}}{c(\alpha_{E_{1,i_0}})}\ y'_{i_0}. \tag{9.8}$$

Note that if $i_0 = 1$, then y_{i_0-1} and y'_{i_0-1} do not exist, and we have less to check. Analogously, if $i_0 = n$, then y_{i_0} and y'_{i_0} do not exist, and we have less to check. Now assume that $i_0 \geq 2$. Because clearly, the $\downarrow$-argument E_{1,i_0} is an expression-argument, $y_{i_0-1} = \triangle$, if and only if ε_{1,i_0-1} is an expression-argument. Similarly, the $\updownarrow$-argument $\langle\updownarrow \alpha_{E_{1,i_0}}\rangle$ is an expression-argument and $y'_{i_0-1} = \triangle$, if and only if ε_{1,i_0-1} is an expression-argument. This implies that $y_{i_0-1} = y'_{i_0-1}$.
Analogously, we can prove that if $i_0 \leq n-1$, then $y_{i_0} = y'_{i_0}$.
Finally, by definition, $\kappa(\mathcal{S}(E_{1,i_0}))$ is equal to $\mathcal{S}(\langle\updownarrow E_{1,i_0}\rangle)$. We can apply Lemma 5.22 to the $\updownarrow$-expression $\langle\updownarrow E_{1,i_0}\rangle$:

$$\langle\updownarrow E_{1,i_0}\rangle \ {}_{\triangledown}\!\equiv \left\langle\updownarrow \alpha_{\langle\updownarrow E_{1,i_0}\rangle}\right\rangle.$$

Because, by Lemma 8.4, E_{1,i_0} is nick free, $\langle\updownarrow E_{1,i_0}\rangle$ is also nick free and we have (strict) equivalence here. Clearly, the $\mathcal{N}$-words occurring in $\langle\updownarrow E_{1,i_0}\rangle$ and the ones occurring in E_{1,i_0} are the same, and so are their parent operators. This implies that $\alpha_{\langle\updownarrow E_{1,i_0}\rangle} = \alpha_{E_{1,i_0}}$. When we combine all ingredients, we find

$$\begin{aligned} \kappa(\mathcal{S}(E_{1,i_0})) &= \mathcal{S}(\langle\updownarrow E_{1,i_0}\rangle) = \mathcal{S}(\left\langle\updownarrow \alpha_{\langle\updownarrow E_{1,i_0}\rangle}\right\rangle) \\ &= \mathcal{S}(\langle\updownarrow \alpha_{E_{1,i_0}}\rangle) = \binom{\alpha_{E_{1,i_0}}}{c(\alpha_{E_{1,i_0}})} \end{aligned}$$

Indeed, Equality (9.8) holds.

We conclude that Property (9.7) is indeed an invariant for the first for-loop. Clearly, in every iteration of this loop, the number of $\downarrow$-arguments of $\widehat{E}_1$ decreases by 1.

After the last iteration of the first for-loop, there are no $\downarrow$-arguments left. By then, $\widehat{E}_1$ is a minimal $\uparrow$-expression satisfying $\left\langle\updownarrow \widehat{E}_1\right\rangle \equiv E$, and each argument of $\widehat{E}_1$ is either an $\mathcal{N}$-word $\alpha_{1,i}$ or an $\updownarrow$-expression $\langle\updownarrow \alpha_{1,i}\rangle$ for an $\mathcal{N}$-word $\alpha_{1,i}$. Hence, the comment after line M$\updownarrow$M.6 in Figure 9.3 is correct.

We prove that a relaxed version of this property, by which $\widehat{E}_1$ is not necessarily minimal, is an invariant for the **second for-loop** of the procedure:

$$\begin{aligned} &\widehat{E}_1 \text{ is an } \uparrow\text{-expression satisfying } \left\langle\updownarrow \widehat{E}_1\right\rangle \equiv E, \text{ and each argument} \\ &\text{of } \widehat{E}_1 \text{ is either an } \mathcal{N}\text{-word } \alpha_{1,i} \text{ or an } \updownarrow\text{-expression } \langle\updownarrow \alpha_{1,i}\rangle \text{ for an} \\ &\mathcal{N}\text{-word } \alpha_{1,i}. \end{aligned} \tag{9.9}$$

It is easily verified that an $\uparrow$-expression E, for which each argument is either an $\mathcal{N}$-word $\alpha_{1,i}$, or an $\updownarrow$-expression $\langle\updownarrow \alpha_{1,i}\rangle$ for an $\mathcal{N}$-word $\alpha_{1,i}$ has Properties ($\mathcal{D}_{\text{Min}}$.1), ($\mathcal{D}_{\text{Min}}$.2), ($\mathcal{D}_{\text{Min}}$.4), ($\mathcal{D}_{\text{Min}}$.5) and ($\mathcal{D}_{\text{Min}}$.6). Hence, E is not minimal, if and only if it violates Property ($\mathcal{D}_{\text{Min}}$.3). This is the case, if and only if $E = \langle\uparrow \langle\updownarrow \alpha_{1,1}\rangle\rangle$ for an $\mathcal{N}$-word $\alpha_{1,1}$. This case is dealt with after the second for-loop, in lines M$\updownarrow$M.19–M$\updownarrow$M.21.

A minimal DNA expression is in particular operator-minimal and a DNA expression of the form $\langle\uparrow \langle\updownarrow \alpha_{1,1}\rangle\rangle$ is also operator-minimal. Hence, whether or not our 'working DNA expression' $\widehat{E}_1$ is minimal, it is certainly operator-minimal.

As long as $\widehat{E}_1$ has $\mathcal{N}$-word-arguments, it cannot be of the form $\langle\uparrow \langle\updownarrow \alpha_{1,1}\rangle\rangle$ for an $\mathcal{N}$-word $\alpha_{1,1}$. This implies that before any iteration of the second for-loop, $\widehat{E}_1$ is minimal, after all.

The global structure of the proof that Property (9.9) is an invariant for the second for-loop, is the same as that of the proof of Property (9.7) for the first for-loop. Because the details are different, especially the ones involved with the semantics of $\left\langle\updownarrow \widehat{E}_1\right\rangle$, we give the full proof.

- Clearly, Property (9.9) is valid before the first iteration of the second for-loop.
- Suppose that Property (9.9) is valid before a certain iteration of the for-loop. Let $\widehat{E}_1 = \langle\uparrow_1 \varepsilon_{1,1} \ldots \varepsilon_{1,n}\rangle$ for some $n \geq 1$ and $\mathcal{N}$-words and $\updownarrow$-expressions $\varepsilon_{1,1}, \ldots, \varepsilon_{1,n}$, before the iteration. Each $\updownarrow$-expression is of the form $\langle\updownarrow \alpha_{1,i}\rangle$ for an $\mathcal{N}$-word $\alpha_{1,i}$.

  In the iteration, we substitute an $\mathcal{N}$-word-argument $\varepsilon_{1,i_0} = \alpha_{1,i_0}$ and possibly a preceding $\updownarrow$-argument and a succeeding $\updownarrow$-argument of $\widehat{E}_1$ by a new $\updownarrow$-argument $\langle\updownarrow \alpha\rangle$. Again, because $L(\mathcal{S}(\langle\updownarrow \alpha\rangle)), R(\mathcal{S}(\langle\updownarrow \alpha\rangle)) \in \mathcal{A}_{\pm}$ and the arguments of $\widehat{E}_1$ fitted together before the substitution, they certainly fit together after the substitution. Hence, $\widehat{E}_1$ is indeed an $\uparrow$-expression after the substitution. Moreover, because the new argument is $\langle\updownarrow \alpha\rangle$, the arguments of $\widehat{E}_1$ are still of the types occurring in Property (9.9). As we discussed above, $\widehat{E}_1$ is still operator-minimal.

  The only thing left to be verified is that the semantics of $\left\langle\updownarrow \widehat{E}_1\right\rangle$ does not change by the substitution. We assume that the $\mathcal{N}$-word-argument $\varepsilon_{1,i_0} = \alpha_{1,i_0}$ of $\widehat{E}_1$ is both preceded by an $\updownarrow$-argument $\varepsilon_{1,i_0-1} = \langle\updownarrow \alpha_{1,i_0-1}\rangle$ and succeeded by an $\updownarrow$-argument $\varepsilon_{1,i_0+1} = \langle\updownarrow \alpha_{1,i_0+1}\rangle$. If α_{1,i_0} is a maximal $\mathcal{N}$-word occurrence in $\widehat{E}_1$, then this is the case, if and only if $2 \leq i_0 \leq n-1$. The other three cases in lines M$\updownarrow$M.8–M$\updownarrow$M.17 can be checked in a similar way.

  For $i = 1, \ldots, n$, let $X_{1,i} = \mathcal{S}^+(\varepsilon_{1,i})$. Before the substitution,

$$\begin{aligned}\mathcal{S}(\left\langle\updownarrow \widehat{E}_1\right\rangle) &= \kappa(X_{1,1}\ y_1\ X_{1,2}\ y_2\ \ldots\ y_{i_0-2}\ \mathcal{S}(\langle\updownarrow \alpha_{1,i_0-1}\rangle)\ y_{i_0-1}\cdot\\ &\qquad \mathcal{S}^+(\alpha_{1,i_0})\ y_{i_0}\ \mathcal{S}(\langle\updownarrow \alpha_{1,i_0+1}\rangle)\ y_{i_0+1}\ \ldots\ y_{n-1}\ X_{1,n})\\ &= \kappa(X_{1,1})\ y_1\ \kappa(X_{1,2})\ y_2\ \ldots\ y_{i_0-2}\ \binom{\alpha_{1,i_0-1}}{c(\alpha_{1,i_0-1})}\ y_{i_0-1}\cdot\\ &\qquad \binom{\alpha_{1,i_0}}{c(\alpha_{1,i_0})}\ y_{i_0}\ \binom{\alpha_{1,i_0+1}}{c(\alpha_{1,i_0+1})}\ y_{i_0+1}\ \ldots\ y_{n-1}\ \kappa(X_{1,n}),\end{aligned}$$

  where for $i = 1, \ldots, n-1$, $y_i = \vartriangle$ if both $\varepsilon_{1,i}$ and $\varepsilon_{1,i+1}$ are expression-arguments, and $y_i = \lambda$ otherwise.

After the substitution of $\langle\updownarrow \alpha_{1,i_0-1}\rangle\, \alpha_{1,i_0}\, \langle\updownarrow \alpha_{1,i_0+1}\rangle$ by $\langle\updownarrow \alpha_{1,i_0-1}\alpha_{1,i_0}\alpha_{1,i_0+1}\rangle$,

$$\begin{aligned}\mathcal{S}(\left\langle\updownarrow \widehat{E}_1\right\rangle) &= \kappa(X_{1,1}\ y_1\ X_{1,2}\ y_2\ \ldots\ y'_{i_0-2}\cdot\\ &\qquad \mathcal{S}(\langle\updownarrow \alpha_{1,i_0-1}\alpha_{1,i_0}\alpha_{1,i_0+1}\rangle)\ y'_{i_0+1}\ \ldots\ y_{n-1}\ X_{1,n})\\ &= \kappa(X_{1,1})\ y_1\ \kappa(X_{1,2})\ y_2\ \ldots\ y'_{i_0-2}\cdot\\ &\qquad \binom{\alpha_{1,i_0-1}\alpha_{1,i_0}\alpha_{1,i_0+1}}{c(\alpha_{1,i_0-1}\alpha_{1,i_0}\alpha_{1,i_0+1})}\ y'_{i_0+1}\ \ldots\ y_{n-1}\ \kappa(X_{1,n}),\end{aligned}$$

where the y_i's are as before, and

$$y'_{i_0-2} \;=\; \begin{cases} \triangle & \text{if both } \varepsilon_{1,i_0-2} \text{ and } \langle\updownarrow \alpha_{1,i_0-1}\alpha_{1,i_0}\alpha_{1,i_0+1}\rangle\\ & \text{are expression-arguments}\\ \lambda & \text{otherwise,}\end{cases}$$

$$y'_{i_0+1} \;=\; \begin{cases} \triangle & \text{if both } \langle\updownarrow \alpha_{1,i_0-1}\alpha_{1,i_0}\alpha_{1,i_0+1}\rangle \text{ and } \varepsilon_{1,i_0+2}\\ & \text{are expression-arguments}\\ \lambda & \text{otherwise.}\end{cases}$$

We must prove that

$$\begin{aligned}&y_{i_0-2}\ \binom{\alpha_{1,i_0-1}}{c(\alpha_{1,i_0-1})}\ y_{i_0-1}\ \binom{\alpha_{1,i_0}}{c(\alpha_{1,i_0})}\ y_{i_0}\ \binom{\alpha_{1,i_0+1}}{c(\alpha_{1,i_0+1})}\ y_{i_0+1}\\ &\qquad = y'_{i_0-2}\ \binom{\alpha_{1,i_0-1}\alpha_{1,i_0}\alpha_{1,i_0+1}}{c(\alpha_{1,i_0-1}\alpha_{1,i_0}\alpha_{1,i_0+1})}\ y'_{i_0+1}.\end{aligned} \tag{9.10}$$

Clearly, if $i_0 - 1 = 1$, then neither y_{i_0-2}, nor y'_{i_0-2} exists, and we have less to check. Analogously, if $i_0 + 1 = n$, then neither y_{i_0+1}, nor y'_{i_0+1} exists, and we have less to check. We now assume that $2 \le i_0 - 1$ and $i_0 + 1 \le n - 1$.

Because $\varepsilon_{1,i_0-1} = \langle\updownarrow \alpha_{1,i_0-1}\rangle$ is an expression-argument, $\varepsilon_{1,i_0} = \alpha_{1,i_0}$ is an $\mathcal{N}$-word-argument, and $\varepsilon_{1,i_0+1} = \langle\updownarrow \alpha_{1,i_0+1}\rangle$ is again an expression-argument, we have

$y_{i_0-2} = \triangle$, if and only if ε_{1,i_0-2} is an expression-argument
$y_{i_0-1} = y_{i_0} = \lambda$, and
$y_{i_0+1} = \triangle$, if and only if ε_{1,i_0+2} is an expression-argument.

On the other hand, because obviously $\langle\updownarrow \alpha_{1,i_0-1}\alpha_{1,i_0}\alpha_{1,i_0+1}\rangle$ is an expression-argument, we have

$y'_{i_0-2} = \triangle$, if and only if ε_{1,i_0-2} is an expression-argument
$y'_{i_0+1} = \triangle$, if and only if ε_{1,i_0+2} is an expression-argument.

This implies that

$$\begin{aligned}&y_{i_0-2} = y'_{i_0-2},\\ &\binom{\alpha_{1,i_0-1}}{c(\alpha_{1,i_0-1})}\ y_{i_0-1}\ \binom{\alpha_{1,i_0}}{c(\alpha_{1,i_0})}\ y_{i_0}\ \binom{\alpha_{1,i_0+1}}{c(\alpha_{1,i_0+1})} = \binom{\alpha_{1,i_0-1}\alpha_{1,i_0}\alpha_{1,i_0+1}}{c(\alpha_{1,i_0-1}\alpha_{1,i_0}\alpha_{1,i_0+1})}, \text{ and}\\ &y_{i_0+1} = y'_{i_0+1}.\end{aligned}$$

Indeed, Equality (9.10) holds, and $\mathcal{S}(\left\langle\updownarrow \widehat{E}_1\right\rangle)$ is the same before and after the substitution.

We conclude that Property (9.9) is indeed an invariant for the second for-loop of procedure `Make↕ExprMinimal`. Clearly, in every iteration of this loop, the number of $\mathcal{N}$-word-arguments of $\widehat{E}_1$ decreases by 1.

After the last iteration of the second for-loop, there are no $\mathcal{N}$-word-arguments left. By then, each argument of $\widehat{E}_1$ is an $\updownarrow$-expression $\langle\updownarrow \alpha_{1,i}\rangle$

for an $\mathcal{N}$-word $\alpha_{1,i}$. Hence, there exist $m \geq 1$ and $\mathcal{N}$-words $\alpha_{1,1}, \ldots, \alpha_{1,m}$ such that $\widehat{E}_1 = \langle\uparrow \langle\updownarrow \alpha_{1,1}\rangle \ldots \langle\updownarrow \alpha_{1,m}\rangle\rangle$. Indeed, the comment after line M↕M.18 in Figure 9.3 is correct.

Moreover, by the invariant, $\widehat{E}_1$ satisfies $\left\langle\updownarrow \widehat{E}_1\right\rangle \equiv E$. Because $\mathcal{S}(\widehat{E}_1) = \mathcal{S}(\langle\uparrow \langle\updownarrow \alpha_{1,1}\rangle \ldots \langle\updownarrow \alpha_{1,m}\rangle\rangle)$ does not contain single-stranded components, the outermost operator $\updownarrow$ in $\left\langle\updownarrow \widehat{E}_1\right\rangle$ has no effect. This implies that $\widehat{E}_1 \equiv \left\langle\updownarrow \widehat{E}_1\right\rangle \equiv E$.

We finally examine the **if statement** in lines M↕M.19–M↕M.21. If $m = 1$, then $\widehat{E}_1 = \langle\uparrow \langle\updownarrow \alpha_{1,1}\rangle\rangle$. In this case, the outermost operator $\uparrow$ has no effect, either. Hence, $E' = \langle\updownarrow \alpha_{1,1}\rangle \equiv \widehat{E}_1 \equiv E$. Indeed, E' is a minimal DNA expression.

After the formulation of Property (9.9), we deduced that a DNA expression $\widehat{E}_1$ with that property, which is not of the form $\langle\uparrow \langle\updownarrow \alpha_{1,1}\rangle\rangle$ for an $\mathcal{N}$-word $\alpha_{1,1}$ is minimal. If $m \geq 2$, then obviously $\widehat{E}_1$ is not of this form. Hence, in this case, $E' = \widehat{E}_1$ is minimal. Moreover, $E' = \widehat{E}_1 \equiv E$.

2. In the proof of the previous claim, we did not make any assumption on the order in which $\downarrow$-arguments and $\mathcal{N}$-word-arguments of $\widehat{E}_1$ are considered in lines M↕M.4 and M↕M.7, respectively. For all possible orders, E' is a minimal DNA expression satisfying $E' \equiv E = \langle\updownarrow E_1\rangle$.

   By Theorem 9.17(3), there exists exactly one such DNA expression E' (regardless of the minimality of E_1). Then certainly, E' must be independent of the orders in which $\downarrow$-arguments and $\mathcal{N}$-word-arguments are considered.

This completes the proof of Theorem 9.20. □

9.1.2 The procedure `Denickify`

The next instruction from `MakeMinimal` we refine is the one in line 18. In Figure 9.4 we describe procedure `Denickify`. Line Dni.24 of this description will be implemented by the same procedure `RotateToMinimal` that we use for lines 23 and 35 of `MakeMinimal`. Again, all substitutions can be achieved by a few insertions and removals of brackets and operators in the DNA expression.

In Lemma 8.7, we have related the presence of consecutive expression-arguments in an operator-minimal $\uparrow$-expression E to the presence of nicks in its semantics $\mathcal{S}(E)$. In order to understand the effect of the while-loop in procedure `Denickify` (which is meant to remove nicks), it is useful to establish such a relation for a more general set of $\uparrow$-expressions.

Lemma 9.21 *Let $E = \langle\uparrow \varepsilon_1 \ldots \varepsilon_n\rangle$, where $n \geq 1$ and $\varepsilon_1, \ldots, \varepsilon_n$ are maximal $\mathcal{N}$-word occurrences and DNA expressions, be an $\uparrow$-expression denoting a certain formal DNA molecule X. For $i = 1, \ldots, n$, let $X_i = \mathcal{S}^+(\varepsilon_i)$.*

If E has Properties ($\mathcal{D}_{Min}$·2), ($\mathcal{D}_{Min}$·4) and ($\mathcal{D}_{Min}$·5), then for $i = 1, \ldots, n$, X_i is nick free, and

$$X = X_1 y_1 X_2 y_2 \ldots y_{n-1} X_n,$$

where for $i = 1, \ldots, n-1$, $y_i = \triangle$ if $R(X_i), L(X_{i+1}) \in \mathcal{A}_\pm$, and $y_i = \lambda$ otherwise.

Here, for $i = 1, \ldots, n-1$, $R(X_i), L(X_{i+1}) \in \mathcal{A}_\pm$, if and only if both ε_i and ε_{i+1} are expression-arguments. In particular, in this case, E is nick free, if and only if E is alternating.

```
Dni.1.   Denickify (E_i)
           // rewrites a minimal ↓-expression E_i which is not alternating,
           // into a minimal, nick free DNA expression E'_i
           // satisfying E'_i ≡_▽ E_i;
           // uses local rearrangements of the DNA expression for this
Dni.2.   {
Dni.3.     Ê_i = E_i;
Dni.4.     while (Ê_i is not alternating)
Dni.5.     do   select two consecutive expression-arguments
                  ε̂_{j-1} and ε̂_j of Ê_i;
Dni.6.          if (ε̂_{j-1} is an ↑-expression ⟨↑ ... ⟨↕ α_{j-1,m_{j-1}}⟩⟩)
Dni.7.          then if (ε̂_j is an ↑-expression ⟨↑ ⟨↕ α_{j,1}⟩ ...⟩)
Dni.8.                    then substitute ε̂_{j-1}ε̂_j in Ê_i
                                 by ⟨↑ ... ⟨↕ α_{j-1,m_{j-1}}α_{j,1}⟩ ...⟩;
Dni.9.                    else    // ε̂_j is an ↕-expression ⟨↕ α_{j,1}⟩
Dni.10.                        substitute ε̂_{j-1}ε̂_j in Ê_i
                                 by ⟨↑ ... ⟨↕ α_{j-1,m_{j-1}}α_{j,1}⟩⟩;
Dni.11.                   fi
Dni.12.         else    // ε̂_{j-1} is an ↕-expression ⟨↕ α_{j-1,1}⟩
Dni.13.                   if (ε̂_j is an ↑-expression ⟨↑ ⟨↕ α_{j,1}⟩ ...⟩)
Dni.14.                   then substitute ε̂_{j-1}ε̂_j in Ê_i
                                 by ⟨↑ ⟨↕ α_{j-1,1}α_{j,1}⟩ ...⟩;
Dni.15.                   else    // ε̂_j is an ↕-expression ⟨↕ α_{j,1}⟩
Dni.16.                        substitute ε̂_{j-1}ε̂_j in Ê_i
                                 by ⟨↕ α_{j-1,1}α_{j,1}⟩;
Dni.17.                   fi
Dni.18.         fi
Dni.19.    od
             // Ê_i is alternating
Dni.20.    if (Ê_i has only one argument E_{i,1} left)
Dni.21.    then substitute Ê_i by E_{i,1};                          (D_Min.3)
Dni.22.    else   // Ê_i has at least two arguments
Dni.23.         if (both the first argument and the last argument of Ê_i
                  are ↑-arguments)
Dni.24.         then substitute Ê_i by a minimal ↑-expression Ê'_i
                       satisfying Ê'_i ≡ Ê_i; (proc. RotateToMinimal) (D_Min.6)
Dni.25.         fi
Dni.26.    fi
Dni.27.    E'_i = Ê_i;
Dni.28.  }
```

Figure 9.4: Pseudo-code of the procedure `Denickify`.

Of course, there is an analogous result for ↓-expressions. The proof of this result is similar to that of Lemma 8.7. At several places in the proof, however, we have to use different arguments to conclude that a certain property is valid. For the sake of clearness, we give the full proof.

Proof: Assume that E has Properties ($\mathcal{D}_{\text{Min}}$.2), ($\mathcal{D}_{\text{Min}}$.4) and ($\mathcal{D}_{\text{Min}}$.5). By the definition

of the semantics of an $\uparrow$-expression (equation (4.2)),

$$X = \nu^+(X_1)y_1\nu^+(X_2)y_2 \dots y_{n-1}\nu^+(X_n),$$

where for $i = 1, \dots, n-1$, $y_i = \triangle$ if $R(X_i), L(X_{i+1}) \in \mathcal{A}_\pm$, and $y_i = \lambda$ otherwise. By Property ($\mathcal{D}_{\text{Min}}$.4), each occurrence of an operator $\uparrow$ or $\downarrow$ in an argument ε_i of E is alternating. Hence, by Lemma 5.8, for $i = 1, \dots, n$, $X_i = \mathcal{S}^+(\varepsilon_i)$ is nick free, and in particular, $\nu^+(X_i) = X_i$. We can thus reduce the semantics to

$$X = X_1y_1X_2y_2 \dots y_{n-1}X_n,$$

with y_i's as before. This is the first part of the claim.

Next, consider any i with $1 \leq i \leq n-1$. By Property ($\mathcal{D}_{\text{Min}}$.2), ε_i is either an $\mathcal{N}$-word α, or an $\updownarrow$-expression, or a $\downarrow$-expression.

If ε_i is an $\mathcal{N}$-word α, then $X_i = \mathcal{S}^+(\varepsilon_i) = \binom{\alpha}{_}$ and $R(X_i) \notin \mathcal{A}_\pm$.

If ε_i an $\updownarrow$-expression, then by Lemma 4.13(2), $R(X_i) = R(\mathcal{S}(\varepsilon_i)) \in \mathcal{A}_\pm$.

Finally, if ε_i is a $\downarrow$-expression, then by Property ($\mathcal{D}_{\text{Min}}$.5), the last argument of ε_i is either an $\mathcal{N}$-word α, or an $\updownarrow$-expression $\langle\updownarrow \alpha\rangle$ for an $\mathcal{N}$-word α. If it were an $\mathcal{N}$-word α, then by Lemma 4.13(4), $R(X_i) = R(\mathcal{S}(\varepsilon_i)) = R(\mathcal{S}^-(\alpha)) \in \mathcal{A}_-$. In that case, the arguments ε_i and ε_{i+1} would not fit together by upper strands, as is required by the outermost operator $\uparrow$ of E. Hence, the last argument of ε_i must be an $\updownarrow$-expression $\langle\updownarrow \alpha\rangle$ for an $\mathcal{N}$-word α. By Lemma 4.13(4), $R(X_i) = R(\mathcal{S}(\varepsilon_i)) = R(\mathcal{S}(\langle\updownarrow \alpha\rangle)) \in \mathcal{A}_\pm$.

We conclude that $R(X_i) \in \mathcal{A}_\pm$, if and only if ε_i is an expression-argument. Analogously, we find that $L(X_{i+1}) \in \mathcal{A}_\pm$, if and only if ε_{i+1} is an expression-argument. Consequently, $R(X_i), L(X_{i+1}) \in \mathcal{A}_\pm$, if and only if both ε_i and ε_{i+1} are expression-arguments. □

In the while-loop of procedure `Denickify`, the 'working DNA expression' $\widehat{E}_i$ is a $\downarrow$-expression. Moreover, as we will see in the proof of Theorem 9.24, it then has (among others) Properties ($\mathcal{D}_{\text{Min}}$.2), ($\mathcal{D}_{\text{Min}}$.4) and ($\mathcal{D}_{\text{Min}}$.5). Hence, the outermost operator $\downarrow$ introduces a nick letter between every pair of consecutive expression-arguments, and these are the only nick letters in $\mathcal{S}(\widehat{E}_i)$.

In every iteration of the while-loop, two consecutive expression-arguments of $\widehat{E}_i$ are substituted by a single expression-argument. In other words, in every iteration, one nick letter is removed from $\mathcal{S}(\widehat{E}_i)$. Step by step, $\widehat{E}_i$ becomes nick free.[2] As we will see in (the proof of) Theorem 9.24(3), the result of the while-loop is independent of the order in which we select pairs of consecutive expression-arguments.

After the while-loop, $\widehat{E}_i$ is nick free, but it is not necessarily minimal any more. The if-then-else construction at the end of the procedure ensures that the DNA expression E_i' resulting from the procedure is not only nick free, but also minimal. Lines Dni.21 and Dni.24 tackle violations of Properties ($\mathcal{D}_{\text{Min}}$.3) and ($\mathcal{D}_{\text{Min}}$.6), respectively.

We illustrate procedure `Denickify` by two examples. In the first example, $\widehat{E}_i$ is not minimal after the while-loop, and the DNA expression is modified by the if-then-else construction. In the second example, $\widehat{E}_i$ is still minimal after the while-loop. Hence, it does not have to be modified any further.

[2]We could have reached this conclusion also using Lemma 8.7 instead of Lemma 9.21. For that, however, we would have to prove that $\widehat{E}_i$ is operator-minimal in the while-loop. It is possible to do that, but it is more elegant to use Lemma 9.21.

Example 9.22 (cf. Example 9.3) Let

$$E_i = \langle\downarrow \langle\updownarrow \alpha_1\alpha_2\alpha_3\alpha_4 c(\alpha_5)\rangle \langle\updownarrow \alpha_6 c(\alpha_7)\rangle\rangle\,,$$

for which

$$\mathcal{S}(E_i) = \tbinom{\alpha_1\alpha_2\alpha_3\alpha_4 c(\alpha_5)}{c(\alpha_1\alpha_2\alpha_3\alpha_4)\alpha_5}{}^{\triangledown}\tbinom{\alpha_6 c(\alpha_7)}{c(\alpha_6)\alpha_7}.$$

E_i is minimal and not nick free. Its two arguments are expression-arguments. For this DNA expression, the while-loop has only one iteration, in which the two expression-arguments are merged according to line Dni.16. The result is:

$$\widehat{E}_i = \langle\downarrow \langle\updownarrow \alpha_1\alpha_2\alpha_3\alpha_4 c(\alpha_5)\alpha_6 c(\alpha_7)\rangle\rangle\,.$$

Indeed $\widehat{E}_i$ is nick free, but it is not minimal. It violates Property ($\mathcal{D}_{\text{Min}}$.3), as the outermost operator $\downarrow$ has only one argument $E_{i,1}$. In this case, line Dni.21 of the procedure is applicable, and the result of the procedure is

$$E'_i = E_{i,1} = \langle\updownarrow \alpha_1\alpha_2\alpha_3\alpha_4 c(\alpha_5)\alpha_6 c(\alpha_7)\rangle\,.$$

Clearly, E'_i satisfies $E'_i \equiv_{\triangledown} E_i$. Moreover, by Theorem 7.5, E'_i is minimal. ■

Example 9.23 (cf. Example 9.8) Let

$$\begin{aligned} E_i = \big\langle\uparrow\ & \langle\updownarrow \alpha_1\alpha_2\alpha_3\alpha_4 c(\alpha_5)\alpha_6 c(\alpha_7)\rangle \\ & \langle\updownarrow \alpha_8\alpha_9\rangle\ \ \alpha_{10}\ \ \langle\downarrow \langle\updownarrow \alpha_{11}\rangle\, \alpha_{12}\, \langle\updownarrow \alpha_{13}\rangle\rangle\ \ \langle\updownarrow \alpha_{14}\rangle \\ & \langle\downarrow \langle\updownarrow \alpha_{15}\rangle\, \alpha_{16}\, \langle\updownarrow \alpha_{17}\rangle\rangle\ \ \alpha_{18}\ \ \langle\updownarrow \alpha_{19}\rangle\ \ \langle\updownarrow \alpha_{20}\rangle\ \big\rangle\,, \end{aligned}$$

for which

$$\begin{aligned} \mathcal{S}(E_i) \ = \ & \tbinom{\alpha_1\alpha_2\alpha_3\alpha_4 c(\alpha_5)\alpha_6 c(\alpha_7)}{c(\alpha_1\alpha_2\alpha_3\alpha_4)\alpha_5 c(\alpha_6)\alpha_7}{}_{\triangle}\tbinom{\alpha_8\alpha_9}{c(\alpha_8\alpha_9)}\tbinom{\alpha_{10}}{-}\tbinom{\alpha_{11}}{c(\alpha_{11})}\tbinom{-}{\alpha_{12}}\cdot \\ & \tbinom{\alpha_{13}}{c(\alpha_{13})}{}_{\triangle}\tbinom{\alpha_{14}}{c(\alpha_{14})}{}_{\triangle}\tbinom{\alpha_{15}}{c(\alpha_{15})}\tbinom{-}{\alpha_{16}}\tbinom{\alpha_{17}}{c(\alpha_{17})}\tbinom{\alpha_{18}}{-}\tbinom{\alpha_{19}}{c(\alpha_{19})}{}_{\triangle}\tbinom{\alpha_{20}}{c(\alpha_{20})}. \end{aligned}$$

E_i is minimal and not nick free. It has seven expression-arguments, clustered in three groups of consecutive expression-arguments, which are separated by the $\mathcal{N}$-word-arguments α_{10} and α_{18}. There are four pairs of consecutive expression-arguments. Hence, the while-loop has four iterations. If in each iteration, we consider the leftmost (remaining) pair of consecutive expression-arguments $\widehat{\varepsilon}_{j-1}$ and $\widehat{\varepsilon}_j$, then we successively get

$$\begin{aligned} \widehat{E}_i = \big\langle\uparrow\ & \langle\updownarrow \alpha_1\alpha_2\alpha_3\alpha_4 c(\alpha_5)\alpha_6 c(\alpha_7)\alpha_8\alpha_9\rangle\ \ \alpha_{10} \\ & \langle\downarrow \langle\updownarrow \alpha_{11}\rangle\, \alpha_{12}\, \langle\updownarrow \alpha_{13}\rangle\rangle\ \ \langle\updownarrow \alpha_{14}\rangle \\ & \langle\downarrow \langle\updownarrow \alpha_{15}\rangle\, \alpha_{16}\, \langle\updownarrow \alpha_{17}\rangle\rangle\ \ \alpha_{18}\ \ \langle\updownarrow \alpha_{19}\rangle\ \ \langle\updownarrow \alpha_{20}\rangle\ \big\rangle \end{aligned}$$

(by applying line Dni.16),

$$\begin{aligned} \widehat{E}_i = \big\langle\uparrow\ & \langle\updownarrow \alpha_1\alpha_2\alpha_3\alpha_4 c(\alpha_5)\alpha_6 c(\alpha_7)\alpha_8\alpha_9\rangle\ \ \alpha_{10} \\ & \langle\downarrow \langle\updownarrow \alpha_{11}\rangle\, \alpha_{12}\, \langle\updownarrow \alpha_{13}\alpha_{14}\rangle\rangle \\ & \langle\downarrow \langle\updownarrow \alpha_{15}\rangle\, \alpha_{16}\, \langle\updownarrow \alpha_{17}\rangle\rangle\ \ \alpha_{18}\ \ \langle\updownarrow \alpha_{19}\rangle\ \ \langle\updownarrow \alpha_{20}\rangle\ \big\rangle \end{aligned}$$

(by applying line Dni.10),

$$\begin{aligned} \widehat{E}_i = \big\langle\uparrow\ & \langle\updownarrow \alpha_1\alpha_2\alpha_3\alpha_4 c(\alpha_5)\alpha_6 c(\alpha_7)\alpha_8\alpha_9\rangle\ \ \alpha_{10} \\ & \langle\downarrow \langle\updownarrow \alpha_{11}\rangle\, \alpha_{12}\, \langle\updownarrow \alpha_{13}\alpha_{14}\alpha_{15}\rangle\, \alpha_{16}\, \langle\updownarrow \alpha_{17}\rangle\rangle\ \ \alpha_{18}\ \ \langle\updownarrow \alpha_{19}\rangle\ \ \langle\updownarrow \alpha_{20}\rangle\ \big\rangle \end{aligned}$$

(by applying line Dni.8), and

$$\widehat{E}_i = \langle\uparrow\ \langle\updownarrow\ \alpha_1\alpha_2\alpha_3\alpha_4 c(\alpha_5)\alpha_6 c(\alpha_7)\alpha_8\alpha_9\rangle\ \alpha_{10} \\ \langle\downarrow\ \langle\updownarrow\ \alpha_{11}\rangle\ \alpha_{12}\ \langle\updownarrow\ \alpha_{13}\alpha_{14}\alpha_{15}\rangle\ \alpha_{16}\ \langle\updownarrow\ \alpha_{17}\rangle\rangle\ \alpha_{18}\ \langle\updownarrow\ \alpha_{19}\alpha_{20}\rangle\ \rangle$$

(by applying line Dni.16 again). The final version of $\widehat{E}_i$ would have been achieved also if we had considered consecutive expression-arguments in a different order.

$\widehat{E}_i$ is nick free now. It has more than one argument left and neither its first argument, nor its last argument is a $\downarrow$-argument. Hence, the if-then-else construction at the end of `Denickify` leaves $\widehat{E}_i$ unchanged, and $E_i' = \widehat{E}_i$. We have

$$\mathcal{S}(E_i') = \binom{\alpha_1\alpha_2\alpha_3\alpha_4 c(\alpha_5)\alpha_6 c(\alpha_7)\alpha_8\alpha_9}{c(\alpha_1\alpha_2\alpha_3\alpha_4)\alpha_5 c(\alpha_6)\alpha_7 c(\alpha_8\alpha_9)}\binom{\alpha_{10}}{-}\binom{\alpha_{11}}{c(\alpha_{11})}\binom{-}{\alpha_{12}}\cdot \\ \binom{\alpha_{13}\alpha_{14}\alpha_{15}}{c(\alpha_{13}\alpha_{14}\alpha_{15})}\binom{-}{\alpha_{16}}\binom{\alpha_{17}}{c(\alpha_{17})}\binom{\alpha_{18}}{-}\binom{\alpha_{19}\alpha_{20}}{c(\alpha_{19}\alpha_{20})}.$$

Indeed, $E_i' \equiv_\triangledown E_i$. Moreover, it is easily verified that E_i' has all six properties from Lemma 8.22 and thus is minimal. ■

We have seen two examples for which procedure `Denickify` works well. We now prove that the procedure is correct in general.

Theorem 9.24 *Let E_i be a minimal $\downarrow$-expression which is not alternating, and let E_i' be the result of applying procedure* `Denickify` *to E_i.*

1. *Procedure* `Denickify` *is well defined.*
2. *The string E_i' is a minimal, nick free DNA expression satisfying $E_i' \equiv_\triangledown E_i$.*
3. *E_i' is independent of the order in which pairs of consecutive expression-arguments $\widehat{\varepsilon}_{j-1}$ and $\widehat{\varepsilon}_j$ are selected in line Dni.5.*

Proof: The only instruction in procedure `Denickify` that is not obviously well defined, is the one in line Dni.24. This instruction presupposes the existence of a minimal $\uparrow$-expression $\widehat{E}_i'$ satisfying $\widehat{E}_i' \equiv \widehat{E}_i$.

The proof that such an $\uparrow$-expression indeed exists uses some properties of $\widehat{E}_i$ which emerge in the proof of Claim 2. Therefore, we combine the proofs of Claims 1 and 2.

1, 2. We first analyse the **while-loop** of procedure `Denickify`. We prove that the following property is an invariant of the loop:

> $\widehat{E}_i$ is a $\downarrow$-expression satisfying $\widehat{E}_i \equiv_\triangledown E_i$, $\widehat{E}_i$ has at least one expression-argument, has Properties ($\mathcal{D}_{\text{Min}}$.1), ($\mathcal{D}_{\text{Min}}$.2), ($\mathcal{D}_{\text{Min}}$.4) and ($\mathcal{D}_{\text{Min}}$.5), and each inner occurrence of $\uparrow$ or $\downarrow$ in $\widehat{E}_i$ has at least two arguments. (9.11)

Before we proceed with the proof, we mention two implications of this property. Suppose that the property is valid. Then let $\widehat{\varepsilon}_j$ be an arbitrary expression-argument of $\widehat{E}_i$. Clearly, $\widehat{\varepsilon}_j$ also has Properties ($\mathcal{D}_{\text{Min}}$.1), ($\mathcal{D}_{\text{Min}}$.2), ($\mathcal{D}_{\text{Min}}$.4) and ($\mathcal{D}_{\text{Min}}$.5). Moreover, each occurrence of $\uparrow$ or $\downarrow$ in $\widehat{\varepsilon}_j$ is an inner occurrence in $\widehat{E}_i$ and thus has at least two arguments. Hence, $\widehat{\varepsilon}_j$ has Property ($\mathcal{D}_{\text{Min}}$.3). Finally, by Property ($\mathcal{D}_{\text{Min}}$.5) of $\widehat{E}_i$, $\widehat{\varepsilon}_j$ has Property ($\mathcal{D}_{\text{Min}}$.6). This implies that $\widehat{\varepsilon}_j$ has all six properties from

Lemma 8.22, and thus is minimal. By Property ($\mathcal{D}_{\text{Min}}$.4) and Lemma 5.8, $\widehat{\varepsilon}_j$ is nick free.

Suppose that in addition, $\widehat{E}_i$ is not alternating, i.e., that it has at least two consecutive expression-arguments. Then by definition, $\widehat{E}_i$ has Property ($\mathcal{D}_{\text{Min}}$.6). Moreover, the *total* number of arguments of (the outermost operator $\downarrow$ of) $\widehat{E}_i$ is certainly at least two. This implies that $\widehat{E}_i$ also has Property ($\mathcal{D}_{\text{Min}}$.3). Consequently, in this case, $\widehat{E}_i$ itself is also minimal.

We now prove Property (9.11):

- Initially, before the first iteration of the while-loop, $\widehat{E}_i$ is equal to the minimal $\downarrow$-expression E_i. Then obviously, $\widehat{E}_i \equiv_\triangledown E_i$, and by Lemma 8.22, $\widehat{E}_i$ has Properties ($\mathcal{D}_{\text{Min}}$.1)–($\mathcal{D}_{\text{Min}}$.6). In particular, by Lemma 8.27(2), each inner occurrence of $\uparrow$ or $\downarrow$ in $\widehat{E}_i$ has at least two arguments. Finally, because $\widehat{E}_i = E_i$ is not alternating, it has at least two (consecutive) expression-arguments. Hence, Property (9.11) is valid.
- Suppose that before a certain operation of the while-loop, Property (9.11) is valid. Let $\widehat{E}_i = \langle\downarrow \widehat{\varepsilon}_1 \dots \widehat{\varepsilon}_n\rangle$ for some $n \geq 1$ and $\mathcal{N}$-words and DNA expressions $\widehat{\varepsilon}_1, \dots, \widehat{\varepsilon}_n$.

  $\widehat{E}_i$ is not alternating at the start of the iteration. Hence, as we have just observed, $\widehat{E}_i$ is minimal. In the iteration, we substitute two consecutive expression-arguments $\widehat{\varepsilon}_{j-1}$ and $\widehat{\varepsilon}_j$ of $\widehat{E}_i$ by a single expression-argument $\widehat{\varepsilon}'_j$. By Corollary 8.2, each expression-argument of $\widehat{E}_i$ is either an $\updownarrow$-expression $\langle\updownarrow \alpha\rangle$ for an $\mathcal{N}$-word α, or an $\uparrow$-expression. Hence, there are four possible combinations for the pair of expression-arguments $\widehat{\varepsilon}_{j-1}$ and $\widehat{\varepsilon}_j$. We now assume that both $\widehat{\varepsilon}_{j-1}$ and $\widehat{\varepsilon}_j$ are $\uparrow$-expressions. The proof for the other three possibilities is similar (and in fact easier).

  In order to prove that after the iteration $\widehat{E}_i$ is still a $\downarrow$-expression satisfying $\widehat{E}_i \equiv_\triangledown E_i$, we could write out the complete semantics of $\widehat{E}_i$ before and after the iteration. This would, however, give very long formulas. To avoid this, we will introduce a DNA subexpression $\langle\downarrow \widehat{\varepsilon}_{j-1}\widehat{\varepsilon}_j\rangle$.

  Let $\widehat{\varepsilon}_{j-1} = \big\langle\uparrow \widehat{\varepsilon}_{j-1,1} \dots \widehat{\varepsilon}_{j-1,m_{j-1}}\big\rangle$ and $\widehat{\varepsilon}_j = \big\langle\uparrow \widehat{\varepsilon}_{j,1} \dots \widehat{\varepsilon}_{j,m_j}\big\rangle$ for some $m_{j-1}, m_j \geq 1$ and $\mathcal{N}$-words and DNA expressions $\widehat{\varepsilon}_{j-1,1}, \dots, \widehat{\varepsilon}_{j-1,m_{j-1}}$ and $\widehat{\varepsilon}_{j,1}, \dots, \widehat{\varepsilon}_{j,m_j}$. In fact, by Property (9.11), $m_{j-1}, m_j \geq 2$. By Property ($\mathcal{D}_{\text{Min}}$.5), the first argument $\widehat{\varepsilon}_{j-1,1}$ of $\widehat{\varepsilon}_{j-1}$ is either an $\mathcal{N}$-word α or an $\updownarrow$-expression $\langle\updownarrow \alpha\rangle$ for an $\mathcal{N}$-word α. The same goes for the last argument $\widehat{\varepsilon}_{j,m_j}$ of $\widehat{\varepsilon}_j$. By Lemma 8.27(5), the last argument $\widehat{\varepsilon}_{j-1,m_{j-1}}$ of $\widehat{\varepsilon}_{j-1}$ is an $\updownarrow$-expression $\big\langle\updownarrow \alpha_{j-1,m_{j-1}}\big\rangle$ for an $\mathcal{N}$-word $\alpha_{j-1,m_{j-1}}$, and the first argument $\widehat{\varepsilon}_{j,1}$ of $\widehat{\varepsilon}_j$ is an $\updownarrow$-expression $\langle\updownarrow \alpha_{j,1}\rangle$ for an $\mathcal{N}$-word $\alpha_{j,1}$. Indeed, $\widehat{\varepsilon}_{j-1}$ and $\widehat{\varepsilon}_j$ satisfy the description of the $\uparrow$-arguments in lines Dni.6 and Dni.7 of procedure `Denickify`. By Property ($\mathcal{D}_{\text{Min}}$.4) of $\widehat{E}_i$, both $\widehat{\varepsilon}_{j-1}$ and $\widehat{\varepsilon}_j$ are alternating.

  As we argued after the formulation of Property (9.11), the $\uparrow$-arguments $\widehat{\varepsilon}_{j-1}$ and $\widehat{\varepsilon}_j$ of $\widehat{E}_i$ are minimal and nick free. Hence, by Corollary 8.10,

$$\begin{aligned}
\mathcal{S}(\widehat{\varepsilon}_{j-1}) &= \mathcal{S}^+(\widehat{\varepsilon}_{j-1,1}) \dots \mathcal{S}^+(\widehat{\varepsilon}_{j-1,m_{j-1}-1}) \tbinom{\alpha_{j-1,m_{j-1}}}{c(\alpha_{j-1,m_{j-1}})}, \\
\mathcal{S}(\widehat{\varepsilon}_j) &= \tbinom{\alpha_{j,1}}{c(\alpha_{j,1})} \mathcal{S}^+(\widehat{\varepsilon}_{j,2}) \dots \mathcal{S}^+(\widehat{\varepsilon}_{j,m_j}) \qquad \text{and}
\end{aligned}$$

$$\mathcal{S}(\langle\downarrow \widehat{\varepsilon}_{j-1}\widehat{\varepsilon}_j\rangle) = \mathcal{S}^+(\widehat{\varepsilon}_{j-1,1})\ldots\mathcal{S}^+(\widehat{\varepsilon}_{j-1,m_{j-1}-1})\tbinom{\alpha_{j-1,m_{j-1}}}{c(\alpha_{j-1,m_{j-1}})}^{\triangledown}\tbinom{\alpha_{j,1}}{c(\alpha_{j,1})}\cdot \mathcal{S}^+(\widehat{\varepsilon}_{j,2})\ldots\mathcal{S}^+(\widehat{\varepsilon}_{j,m_j}). \tag{9.12}$$

Now, let $\widehat{\varepsilon}'_j$ be the string we must substitute for $\widehat{\varepsilon}_{j-1}\widehat{\varepsilon}_j$ according to line Dni.8:

$$\widehat{\varepsilon}'_j = \langle\uparrow \widehat{\varepsilon}_{j-1,1}\ldots\widehat{\varepsilon}_{j-1,m_{j-1}-1}\langle\updownarrow \alpha_{j-1,m_{j-1}}\alpha_{j,1}\rangle\widehat{\varepsilon}_{j,2}\ldots\widehat{\varepsilon}_{j,m_j}\rangle.$$

The arguments of $\widehat{\varepsilon}'_j$ fit together by upper strands, because the arguments of $\widehat{\varepsilon}_{j-1}$ and $\widehat{\varepsilon}_j$ do so. Hence, $\widehat{\varepsilon}'_j$ is an $\uparrow$-expression. It is easily verified that $\widehat{\varepsilon}'_j$ also has Properties ($\mathcal{D}_{\text{Min}}$.1)–($\mathcal{D}_{\text{Min}}$.6), and thus is minimal. Moreover, $\widehat{\varepsilon}'_j$ is alternating, because $\widehat{\varepsilon}_{j-1}$ and $\widehat{\varepsilon}_j$ are. Hence, $\widehat{\varepsilon}'_j$ is nick free, and by Corollary 8.10,

$$\mathcal{S}(\widehat{\varepsilon}'_j) = \mathcal{S}^+(\widehat{\varepsilon}_{j-1,1})\ldots\mathcal{S}^+(\widehat{\varepsilon}_{j-1,m_{j-1}-1})\tbinom{\alpha_{j-1,m_{j-1}}\alpha_{j,1}}{c(\alpha_{j-1,m_{j-1}}\alpha_{j,1})}\cdot \mathcal{S}^+(\widehat{\varepsilon}_{j,2})\ldots\mathcal{S}^+(\widehat{\varepsilon}_{j,m_j}). \tag{9.13}$$

It follows from (9.12) and (9.13) that $\widehat{\varepsilon}'_j \equiv_{\triangledown} \langle\downarrow \widehat{\varepsilon}_{j-1}\widehat{\varepsilon}_j\rangle$. In fact, $\widehat{\varepsilon}'_j$ is nick free. whereas $\mathcal{S}(\langle\downarrow \widehat{\varepsilon}_{j-1}\widehat{\varepsilon}_j\rangle)$ contains one upper nick letter, between the semantics of $\widehat{\varepsilon}_{j-1}$ and the semantics of $\widehat{\varepsilon}_j$.

Because $L(\mathcal{S}(\widehat{\varepsilon}'_j)) = L(\mathcal{S}(\widehat{\varepsilon}_{j-1}))$ and $R(\mathcal{S}(\widehat{\varepsilon}'_j)) = R(\mathcal{S}(\widehat{\varepsilon}_j))$, the arguments of $\widehat{E}_i$ still fit together by lower strands when we substitute $\widehat{\varepsilon}_{j-1}\widehat{\varepsilon}_j$ in $\widehat{E}_i$ by $\widehat{\varepsilon}'_j$. Hence, $\widehat{E}_i$ is still a DNA expression after the substitution. In particular, it is a $\downarrow$-expression. Moreover, by Lemma 5.11, Lemma 5.10 and Property (9.11),

$$\begin{aligned}\langle\downarrow \widehat{\varepsilon}_1\ldots\widehat{\varepsilon}_{j-2}\widehat{\varepsilon}'_j\widehat{\varepsilon}_{j+1}\ldots\widehat{\varepsilon}_n\rangle &\equiv_{\triangledown} \langle\downarrow \widehat{\varepsilon}_1\ldots\widehat{\varepsilon}_{j-2}\langle\downarrow \widehat{\varepsilon}_{j-1}\widehat{\varepsilon}_j\rangle\widehat{\varepsilon}_{j+1}\ldots\widehat{\varepsilon}_n\rangle\\ &\equiv \langle\downarrow \widehat{\varepsilon}_1\ldots\widehat{\varepsilon}_{j-2}\widehat{\varepsilon}_{j-1}\widehat{\varepsilon}_j\widehat{\varepsilon}_{j+1}\ldots\widehat{\varepsilon}_n\rangle\\ &\equiv_{\triangledown} E_i.\end{aligned} \tag{9.14}$$

Hence, after the substitution, $\widehat{E}_i$ still satisfies $\widehat{E}_i \equiv_{\triangledown} E_i$. The upper nick letter between $\mathcal{S}(\widehat{\varepsilon}_{j-1})$ and $\mathcal{S}(\widehat{\varepsilon}_j)$, which was present in $\mathcal{S}(\widehat{E}_i)$ before the substitution, is no longer present after the substitution. For the rest, $\mathcal{S}(\widehat{E}_i)$ is the same before and after the substitution.

Because $\widehat{\varepsilon}'_j$ is a DNA expression, $\widehat{E}_i$ still has at least one expression-argument after the substitution.

The outermost operator $\uparrow$ of $\widehat{\varepsilon}'_j$ has $m_{j-1} + m_j - 1 \geq 3$ arguments. As we observed before, these are maximal $\mathcal{N}$-word occurrences and DNA expressions, alternately. Now, it is easily verified, that after the substitution, $\widehat{E}_i$ has Properties ($\mathcal{D}_{\text{Min}}$.1), ($\mathcal{D}_{\text{Min}}$.2), ($\mathcal{D}_{\text{Min}}$.4) and ($\mathcal{D}_{\text{Min}}$.5), and that each inner occurrence of $\uparrow$ or $\downarrow$ in $\widehat{E}_i$ has at least two arguments, because this was the case before the substitution.

We conclude that Property (9.11) is indeed an invariant of the while-loop. In every iteration of the loop, we substitute a pair of consecutive expression-arguments of $\widehat{E}_i$ by a single expression-argument. Thus, the number of pairs of consecutive expression-arguments decreases by 1. After the last iteration of the loop, $\widehat{E}_i$ is alternating. By Lemma 9.21, this implies that $\widehat{E}_i$ is nick free.

At the beginning of the proof, we deduced from Property (9.11), that each expression-argument of $\widehat{E}_i$ is minimal and nick free. Because $\widehat{E}_i$ has become alternating, it does not necessarily have Property ($\mathcal{D}_{\text{Min}}$.3) and Property ($\mathcal{D}_{\text{Min}}$.6) any more. Hence, after the last iteration of the while-loop, $\widehat{E}_i$ itself is not necessarily minimal.

This is made up for in the **if-then-else construction** following the while-loop. We prove that for every possible case, the resulting string E'_i is a minimal, nick free DNA expression satisfying $E'_i \equiv_\triangledown E_i$.

- Assume that $\widehat{E}_i$ has only one argument. By Property (9.11), this must be an expression-argument $E_{i,1}$. This argument is minimal and nick free. Because it is nick free, the outermost operator $\downarrow$ of $\widehat{E}_i$ has no effect. In this case,

  $$E'_i = E_{i,1} \equiv \langle\downarrow E_{i,1}\rangle = \widehat{E}_i \equiv_\triangledown E_i.$$

- Assume that $\widehat{E}_i$ has at least two arguments, and that both the first argument and the last argument of $\widehat{E}_i$ are $\uparrow$-arguments.

  Because $\widehat{E}_i$ has at least two arguments, by Property (9.11), each occurrence of $\uparrow$ or $\downarrow$ in $\widehat{E}_i$ has at least two arguments. This implies that, in addition to Properties ($\mathcal{D}_{\text{Min}}$.1), ($\mathcal{D}_{\text{Min}}$.2), ($\mathcal{D}_{\text{Min}}$.4) and ($\mathcal{D}_{\text{Min}}$.5), $\widehat{E}_i$ has Property ($\mathcal{D}_{\text{Min}}$.3). $\widehat{E}_i$ is nick free. Hence, by Lemma 9.14, there exists a minimal $\uparrow$-expression $\widehat{E}'_i$ satisfying $\widehat{E}'_i \equiv \widehat{E}_i$. In particular, line Dni.24 of the procedure is well defined. In this case,

  $$E'_i = \widehat{E}'_i \equiv \widehat{E}_i \equiv_\triangledown E_i.$$

- Finally, assume that $\widehat{E}_i$ has at least two arguments, and that either the first argument, or the last argument of $\widehat{E}_i$ is not an $\uparrow$-argument. Without loss of generality, assume that the first argument of $\widehat{E}_i$ is not an $\uparrow$-argument. As in the previous case, because $\widehat{E}_i$ has at least two arguments, it has Property ($\mathcal{D}_{\text{Min}}$.3).

  By Property ($\mathcal{D}_{\text{Min}}$.1) and Property ($\mathcal{D}_{\text{Min}}$.2), the first argument of $\widehat{E}_i$ must be either an $\mathcal{N}$-word α or an $\updownarrow$-expression $\langle\updownarrow \alpha\rangle$ for an $\mathcal{N}$-word α. Hence, $\widehat{E}_i$ also has Property ($\mathcal{D}_{\text{Min}}$.6).

  This implies that $\widehat{E}_i$ has all six properties from Lemma 8.22, and thus is minimal. In this case

  $$E'_i = \widehat{E}_i \equiv_\triangledown E_i,$$

  and indeed, $E'_i = \widehat{E}_i$ is nick free.

3. We prove that the DNA expression $\widehat{E}_i$ that is left after the while-loop is independent of the order in which pairs of consecutive expression-arguments $\widehat{\varepsilon}_{j-1}$ and $\widehat{\varepsilon}_j$ are selected for substitution in line Dni.5. Then the final result E'_i of the procedure is certainly independent of this order. In the remainder of this proof, we call this order the *substitution order*.

   Let $\varepsilon_1, \ldots, \varepsilon_n$ for some $n \geq 1$ be the arguments of the original $\downarrow$-expression E_i. In this sequence of arguments, we can distinguish (maximal) subsequences of expression-arguments. Such a subsequence is succeeded by a maximal $\mathcal{N}$-word occurrence, which is in turn succeeded by a (maximal) subsequence of expression-arguments, and so on. Hence, the different subsequences of expression-arguments are separated by the maximal $\mathcal{N}$-word occurrences, which are not affected by the while-loop. The substitution of pairs of consecutive expression-arguments in one subsequence of expression-arguments does not affect the expression-arguments in another subsequence. In fact, the different subsequences are rewritten independently

in the while-loop. Therefore, we only have to consider the substitution order of the expression-arguments within the same (maximal) subsequence.

Let $\varepsilon_{j_0}, \ldots, \varepsilon_{j_1}$ with $1 \leq j_0 \leq j_1 \leq n$ be a (maximal) subsequence of expression-arguments of E_i. If $j_0 = j_1$, then the subsequence does not contain any pair of consecutive expression-arguments, and the subsequence is not affected by the while-loop. Now assume that $j_0 < j_1$. In the course of the while-loop, the subsequence of expression-arguments $\varepsilon_{j_0} \ldots \varepsilon_{j_1}$ is rewritten into a single expression-argument $\widehat{\varepsilon}'_{j_1}$.

By Corollary 8.2, each expression-argument ε_j of the minimal $\downarrow$-expression E_i is either an $\updownarrow$-expression $\langle \updownarrow \alpha_{j,1} \rangle$ for an $\mathcal{N}$-word $\alpha_{j,1}$ or an $\uparrow$-expression.

We first assume that for $j = j_0, \ldots, j_1$, ε_j is an $\updownarrow$-expression $\langle \updownarrow \alpha_{j,1} \rangle$ for an $\mathcal{N}$-word $\alpha_{j,1}$. Then it is easy to prove by induction on $j_1 - j_0$ that

$$\widehat{\varepsilon}'_{j_1} = \langle \updownarrow \alpha_{j_0,1} \ldots \alpha_{j_1,1} \rangle \,,$$

regardless of the substitution order.

From now on, we assume that there is at least one ε_j with $j_0 \leq j \leq j_1$ which is an $\uparrow$-expression. We establish three properties of the resulting expression-argument $\widehat{\varepsilon}'_{j_1}$, which are independent of the substitution order. We then prove that these properties completely determine $\widehat{\varepsilon}'_{j_1}$.

- Suppose that in the while-loop in procedure `Denickify`, two consecutive expression-arguments $\widehat{\varepsilon}_{j-1}$ and $\widehat{\varepsilon}_j$ of $\widehat{E}_i$ are substituted by a single expression-argument $\widehat{\varepsilon}'_j$, and that at least one of the two expression-arguments substituted is an $\uparrow$-argument. Then it follows from a simple inspection of the code of the procedure that $\widehat{\varepsilon}'_j$ is also an $\uparrow$-argument. This implies that after any substitution, there is at least one $\uparrow$-argument left in the subsequence of arguments corresponding to $\varepsilon_{j_0} \ldots \varepsilon_{j_1}$. In particular, after the last iteration of the while-loop, when the subsequence has been reduced to a single expression-argument, this expression-argument $\widehat{\varepsilon}'_{j_1}$ is an $\uparrow$-argument.
- Suppose again that in the while-loop, two consecutive expression-arguments $\widehat{\varepsilon}_{j-1}$ and $\widehat{\varepsilon}_j$ are substituted by $\widehat{\varepsilon}'_j$, and that at least one of the two expression-arguments substituted is an $\uparrow$-argument. Then the $\mathcal{N}$-word-arguments and $\downarrow$-arguments of the new $\uparrow$-argument $\widehat{\varepsilon}'_j$ come straight from $\widehat{\varepsilon}_{j-1}$ and $\widehat{\varepsilon}_j$. There is just one $\updownarrow$-argument of $\widehat{\varepsilon}'_j$ that is new, i.e., that is a combination of two original, 'adjacent' $\updownarrow$-arguments. In particular, after any substitution, the $\downarrow$-arguments of the subsequence of arguments corresponding to $\varepsilon_{j_0}, \ldots, \varepsilon_{j_1}$ are the same as before the substitution, and they occur at corresponding positions in the subsequence, in the same order. This is still the case after the last iteration of the while-loop, when the entire subsequence $\varepsilon_{j_0} \ldots \varepsilon_{j_1}$ has been rewritten into the $\uparrow$-expression $\widehat{\varepsilon}'_{j_1}$. That is, the $\downarrow$-arguments of $\widehat{\varepsilon}'_{j_1}$ are exactly all $\downarrow$-arguments of the original expression-arguments $\varepsilon_{j_0}, \ldots, \varepsilon_{j_1}$, in the same order. This is independent of the substitution order.
- As we have seen in the proof of Claims 1 and 2, $\widehat{\varepsilon}'_{j_1}$ is minimal and nick free. Moreover, by a derivation similar to (9.14), we can prove that $\widehat{\varepsilon}'_{j_1} \equiv_{\triangledown} \langle \downarrow \varepsilon_{j_0} \ldots \varepsilon_{j_1} \rangle$. Hence, $\mathcal{S}(\widehat{\varepsilon}'_{j_1}) = X'_{j_1}$, where

  $$X'_{j_1} = \nu(\mathcal{S}(\langle \downarrow \varepsilon_{j_0} \ldots \varepsilon_{j_1} \rangle)).$$

  Again, this is independent of the substitution order.

We must prove that not just the $\downarrow$-arguments, but *all* the arguments of $\widehat{\varepsilon}'_{j_1}$ (and thus $\widehat{\varepsilon}'_{j_1}$ itself) are independent of the substitution order. For this, we make an elegant twist, back to the construction of minimal DNA expressions, as described in Chapter 7 and Chapter 8.

By Theorem 8.13, the $\downarrow$-arguments of $\widehat{\varepsilon}'_{j_1}$ define a lower block partitioning $\mathcal{P}$ of X'_{j_1}, and $\widehat{\varepsilon}'_{j_1}$ satisfies the construction from Theorem 7.24(1) based on $\mathcal{P}$.

Now, because the $\downarrow$-arguments of $\widehat{\varepsilon}'_{j_1}$ (and the order of their occurrence in $\widehat{\varepsilon}'_{j_1}$) and the semantics X'_{j_1} of $\widehat{\varepsilon}'_{j_1}$ are independent of the substitution order, so is the lower block partitioning $\mathcal{P}$. Moreover, in the construction from Theorem 7.24(1), the arguments corresponding to the lower blocks in $\mathcal{P}$ are precisely the $\downarrow$-arguments of $\widehat{\varepsilon}'_{j_1}$, which (thus) are independent of the substitution order. Because the arguments corresponding to the other parts of $\mathcal{P}$ ($\mathcal{N}$-words α and $\updownarrow$-expressions $\langle\updownarrow\alpha\rangle$ for $\mathcal{N}$-words α) are fixed by the construction, the entire $\uparrow$-expression $\widehat{\varepsilon}'_{j_1}$ is independent of the substitution order. We have thus proved the claim.

In fact, it is possible to completely specify $\widehat{\varepsilon}'_{j_1}$. For each $\updownarrow$-argument ε_j with $j_0 \leq j \leq j_1$, let $m_j = 1$. Hence, $\varepsilon_j = \langle\updownarrow\alpha_{j,1}\rangle = \langle\updownarrow\alpha_{j,m_j}\rangle$.

For each $\uparrow$-argument ε_j with $j_0 \leq j \leq j_1$, let $\varepsilon_j = \langle\uparrow\varepsilon_{j,1}\ldots\varepsilon_{j,m_j}\rangle$ for some $m_j \geq 1$ and $\mathcal{N}$-words and DNA expressions $\varepsilon_{j,1},\ldots,\varepsilon_{j,m_j}$. By Lemma 8.27(2), $m_j \geq 2$. By Lemma 8.27(5), if $j \geq j_0+1$, then $\varepsilon_{j,1} = \langle\updownarrow\alpha_{j,1}\rangle$ for an $\mathcal{N}$-word $\alpha_{j,1}$, and if $j \leq j_1-1$, then $\varepsilon_{j,m_j} = \langle\updownarrow\alpha_{j,m_j}\rangle$ for an $\mathcal{N}$-word α_{j,m_j}.

One can prove by induction on $j_1 - j_0$ that

$$\begin{aligned}\widehat{\varepsilon}'_{j_1} = \big\langle\uparrow\ & \varepsilon_{j_0,1}\ldots\varepsilon_{j_0,m_{j_0}-1}\big\langle\updownarrow\alpha_{j_0,m_{j_0}}\alpha_{j_0+1,1}\big\rangle\\ & \varepsilon_{j_0+1,2}\ldots\varepsilon_{j_0+1,m_{j_0+1}-1}\big\langle\updownarrow\alpha_{j_0+1,m_{j_0+1}}\alpha_{j_0+2,1}\big\rangle\\ & \ldots\big\langle\updownarrow\alpha_{j_1-1,m_{j_1-1}}\alpha_{j_1,1}\big\rangle\,\varepsilon_{j_1,2}\ldots\varepsilon_{j_1,m_{j_1}}\,\big\rangle\,.\end{aligned}$$

Here, if for some j with $j_0+1 \leq j \leq j_1-1$, ε_j is an $\updownarrow$-argument $\langle\updownarrow\alpha_{j,1}\rangle$ (which is the case, if and only if $m_j = 1$), then the sequence of arguments

$$\big\langle\updownarrow\alpha_{j-1,m_{j-1}}\alpha_{j,1}\big\rangle\,\varepsilon_{j,2}\ldots\varepsilon_{j,m_j-1}\big\langle\updownarrow\alpha_{j,m_j}\alpha_{j+1,1}\big\rangle$$

must be understood as

$$\big\langle\updownarrow\alpha_{j-1,m_{j-1}}\alpha_{j,1}\alpha_{j+1,1}\big\rangle\,.$$

Otherwise the $\mathcal{N}$-word $\alpha_{j,1} = \alpha_{j,m_j}$ would occur twice in $\widehat{\varepsilon}'_{j_1}$. This interpretation extends in a natural way to two or more consecutive $\updownarrow$-arguments ε_j.

This completes the proof of Theorem 9.24. □

9.1.3 The procedure `RotateToMinimal`

There are three instructions left in the recursive function `MakeMinimal` and procedure `Denickify`, which have to be worked out in detail.

In line 23 of `MakeMinimal`, we have to determine a minimal $\uparrow$-expression E'_i satisfying $E'_i \equiv E_i$. Here, E_i is a $\downarrow$-expression for which either the first argument or the last argument

```
RtM.1.   RotateToMinimal (E)
           // rewrites an alternating ↓-expression E = ⟨↓ ε_1 ... ε_n⟩
           // with Properties (D_Min.1)-(D_Min.5), for which either
           // the first argument ε_1 or the last argument ε_n (or both)
           // is an ↑-argument, into a minimal ↑-expression E'
           // satisfying E' ≡ E;
           // uses local rearrangements of the DNA expression for this
RtM.2.   {
RtM.3.     if (ε_1 is an ↑-expression ⟨↑ ε_{1,1} ... ε_{1,m_1-1} ε_{1,m_1}⟩)
RtM.4.     then if (ε_n is an ↑-expression ⟨↑ ε_{n,1} ε_{n,2} ... ε_{n,m_n}⟩)
RtM.5.          then E' = ⟨↑ ε_{1,1} ... ε_{1,m_1-1} ⟨↓ ε_{1,m_1} ε_2 ... ε_{n-1} ε_{n,1}⟩ ε_{n,2} ... ε_{n,m_n}⟩;
                                                                  (D_Min.6)
RtM.6.          else  E' = ⟨↑ ε_{1,1} ... ε_{1,m_1-1} ⟨↓ ε_{1,m_1} ε_2 ... ε_{n-1} ε_n⟩⟩;
RtM.7.          fi
RtM.8.     else    // ε_n must be an ↑-expression ⟨↑ ε_{n,1} ε_{n,2} ... ε_{n,m_n}⟩
RtM.9.          E' = ⟨↑ ⟨↓ ε_1 ε_2 ... ε_{n-1} ε_{n,1}⟩ ε_{n,2} ... ε_{n,m_n}⟩;
RtM.10.    fi
RtM.11.  }
```

Figure 9.5: Pseudo-code of the procedure `RotateToMinimal`.

is an ↑-argument. In the proof of Theorem 9.17(1) and (2), we have established that in addition, E_i is minimal and alternating. By Lemma 8.22, E_i has Properties ($\mathcal{D}_{\text{Min}}$.1)–($\mathcal{D}_{\text{Min}}$.6).

The situation in line 35 of `MakeMinimal` is not too different. We have to determine a minimal ↓-expression E' satisfying $E' \equiv E$. Here, E is an alternating ↑-expression with at least two arguments, for which both the first argument and the last argument are ↓-arguments. In the proof of Theorem 9.17(1) and (2), we have established that in addition, E has Properties ($\mathcal{D}_{\text{Min}}$.1)–($\mathcal{D}_{\text{Min}}$.5).

Finally, the situation in line Dni.24 of procedure `Denickify` is completely analogous to the previous situation: we have to determine a minimal ↑-expression $\widehat{E}'_i$ satisfying $\widehat{E}'_i \equiv \widehat{E}_i$. Here, $\widehat{E}_i$ is an alternating ↓-expression with at least two arguments, for which both the first argument and the last argument are ↑-arguments. In the proof of Theorem 9.24(1) and (2), we have established that in addition, $\widehat{E}_i$ has Properties ($\mathcal{D}_{\text{Min}}$.1)–($\mathcal{D}_{\text{Min}}$.5).

As the three cases are so similar, it is not surprising that they can be tackled by the same procedure `RotateToMinimal`. As usual, ↑-expressions and ↓-expressions are rewritten in analogous ways. In Figure 9.5, we give the procedure for ↓-expressions. In fact, the procedure is just a (nested) if-then-else statement. In all cases, the result can be achieved by a few insertions and removals of brackets and operators in the DNA expression.

The name `RotateToMinimal` is derived from the procedure's effect on the structure trees of the DNA expressions involved. In the proof of Theorem 9.27, we will see that the procedure is justified by Theorem 5.16 and Theorem 5.21. As we have depicted in Figure 5.2, Theorem 5.16 corresponds to a rotation in the structure tree. In the present situation, Theorem 5.21, which is based on Theorem 5.16, corresponds to two rotations in the structure tree.

We illustrate procedure `RotateToMinimal` by two examples.

Example 9.25 (cf. Example 9.5) Let

$$E = \langle\downarrow \langle\uparrow \langle\updownarrow \alpha_8\alpha_9\rangle \alpha_{10} \langle\updownarrow \alpha_{11}\rangle\rangle \alpha_{12} \langle\updownarrow \alpha_{13}\rangle\rangle ,$$

for which

$$\mathcal{S}(E) = \binom{\alpha_8\alpha_9}{c(\alpha_8\alpha_9)}\binom{\alpha_{10}}{-}\binom{\alpha_{11}}{c(\alpha_{11})}\binom{-}{\alpha_{12}}\binom{\alpha_{13}}{c(\alpha_{13})}.$$

The $\downarrow$-expression E is minimal and alternating, its first argument is an $\uparrow$-argument and its last argument is not an $\uparrow$-argument. According to line RtM.6,

$$E' = \langle\uparrow \langle\updownarrow \alpha_8\alpha_9\rangle\, \alpha_{10}\, \langle\downarrow \langle\updownarrow \alpha_{11}\rangle\, \alpha_{12}\, \langle\updownarrow \alpha_{13}\rangle\rangle\rangle\,.$$

Indeed, $\mathcal{S}(E') = \mathcal{S}(E)$, i.e., $E' \equiv E$. Moreover, $|E'| = |E|$, which implies that E' is minimal just like E. ■

Example 9.26 (cf. Example 9.11) Let

$$\begin{aligned} E = \big\langle\downarrow \big\langle\uparrow\ & \langle\updownarrow \alpha_1\alpha_2\alpha_3\alpha_4 c(\alpha_5)\alpha_6 c(\alpha_7)\alpha_8\alpha_9\rangle\ \ \alpha_{10} \\ & \langle\downarrow \langle\updownarrow \alpha_{11}\rangle\, \alpha_{12}\, \langle\updownarrow \alpha_{13}\alpha_{14}\alpha_{15}\rangle\, \alpha_{16}\, \langle\updownarrow \alpha_{17}\rangle\rangle\ \ \alpha_{18}\ \ \langle\updownarrow \alpha_{19}\alpha_{20}\rangle\ \big\rangle \\ & \alpha_{21}\ \ \langle\uparrow \langle\updownarrow \alpha_{22}\rangle\, \alpha_{23}\rangle\ \big\rangle\,, \end{aligned}$$

which denotes the formal DNA molecule X from Figure 9.2. The $\downarrow$-expression E is alternating, has Properties ($\mathcal{D}_{\text{Min}}$.1)–($\mathcal{D}_{\text{Min}}$.5), and both its first argument and its last argument are $\uparrow$-arguments. Hence, it violates Property ($\mathcal{D}_{\text{Min}}$.6). According to line RtM.5,

$$\begin{aligned} E' = \big\langle\uparrow\ & \langle\updownarrow \alpha_1\alpha_2\alpha_3\alpha_4 c(\alpha_5)\alpha_6 c(\alpha_7)\alpha_8\alpha_9\rangle\ \ \alpha_{10} \\ & \langle\downarrow \langle\updownarrow \alpha_{11}\rangle\, \alpha_{12}\, \langle\updownarrow \alpha_{13}\alpha_{14}\alpha_{15}\rangle\, \alpha_{16}\, \langle\updownarrow \alpha_{17}\rangle\rangle\ \ \alpha_{18} \\ & \langle\downarrow \langle\updownarrow \alpha_{19}\alpha_{20}\rangle\, \alpha_{21}\, \langle\updownarrow \alpha_{22}\rangle\rangle\ \ \alpha_{23}\ \big\rangle\,. \end{aligned}$$

E' also denotes X, i.e., $E' \equiv E$. Moreover, it is easily verified that E' has all six properties from Lemma 8.22 and thus is minimal. ■

Procedure `RotateToMinimal` is also correct:

Theorem 9.27 *Let E be an alternating $\downarrow$-expression with Properties ($\mathcal{D}_{Min}$.1)–($\mathcal{D}_{Min}$.5), for which either the first argument or the last argument (or both) is an $\uparrow$-argument.*

Then the string E' resulting from procedure `RotateToMinimal` *is a minimal $\uparrow$-expression satisfying $E' \equiv E$.*

Proof: Let $E = \langle\downarrow \varepsilon_1 \ldots \varepsilon_n\rangle$ for some $n \geq 1$ and $\mathcal{N}$-words and DNA expressions $\varepsilon_1, \ldots, \varepsilon_n$. Without loss of generality, assume that the first argument of E is an $\uparrow$-argument: $\varepsilon_1 = \langle\uparrow \varepsilon_{1,1} \ldots \varepsilon_{1,m_1-1}\varepsilon_{1,m_1}\rangle$ for some $m_1 \geq 1$ and $\mathcal{N}$-words and DNA expressions $\varepsilon_{1,1}, \ldots, \varepsilon_{1,m_1-1}, \varepsilon_{1,m_1}$. This implies that the **if-then-else construction** in lines RtM.4–RtM.7 of `RotateToMinimal` is applicable.

By Property ($\mathcal{D}_{\text{Min}}$.3), $n, m_1 \geq 2$. Because the arguments of the $\downarrow$-expression E must fit together by lower strands, the last argument ε_{1,m_1} of ε_1 cannot be an $\mathcal{N}$-word. Hence, by Property ($\mathcal{D}_{\text{Min}}$.5), it is an $\updownarrow$-expression $\langle\updownarrow \alpha\rangle$ for an $\mathcal{N}$-word α. By the same property, the first argument $\varepsilon_{1,1}$ of ε_1 is either an $\mathcal{N}$-word α or an $\updownarrow$-expression $\langle\updownarrow \alpha\rangle$ for an $\mathcal{N}$-word α.

Because E is alternating and has Property ($\mathcal{D}_{\text{Min}}$.4), each occurrence of $\uparrow$ or $\downarrow$ in E is alternating.

By Property ($\mathcal{D}_{\text{Min}}$.1) and Property ($\mathcal{D}_{\text{Min}}$.2), each argument ε_i of E is either an $\mathcal{N}$-word α, or an $\updownarrow$-expression $\langle\updownarrow \alpha\rangle$ for an $\mathcal{N}$-word α, or an $\uparrow$-expression. The string E' resulting from procedure `RotateToMinimal` depends on whether or not the last argument ε_n of E is an $\uparrow$-expression, which is tested in line RtM.4. We prove that in both cases, E' is a minimal $\uparrow$-expression satisfying $E' \equiv E$.

- Assume that ε_n is an $\uparrow$-argument, which implies in particular that E does not have Property ($\mathcal{D}_{\text{Min}}$.6) and thus is not minimal. Let $\varepsilon_n = \langle\uparrow \varepsilon_{n,1}\varepsilon_{n,2}\ldots\varepsilon_{n,m_n}\rangle$ for some $m_n \geq 1$ and $\mathcal{N}$-words and DNA expressions $\varepsilon_{n,1}, \varepsilon_{n,2}\ldots, \varepsilon_{n,m_n}$. Hence,

  $$E = \langle\downarrow \langle\uparrow \varepsilon_{1,1}\ldots\varepsilon_{1,m_1-1}\varepsilon_{1,m_1}\rangle\, \varepsilon_2\ldots\varepsilon_{n-1}\, \langle\uparrow \varepsilon_{n,1}\varepsilon_{n,2}\ldots\varepsilon_{n,m_n}\rangle\rangle\,.$$

  By Property ($\mathcal{D}_{\text{Min}}$.3), $m_n \geq 2$. When we apply Theorem 5.21(1) and (2) (with $r = 1$) to E, we find that

  $$E' = \langle\uparrow_0 \varepsilon_{1,1}\ldots\varepsilon_{1,m_1-1}\, \langle\downarrow_1 \varepsilon_{1,m_1}\varepsilon_2\ldots\varepsilon_{n-1}\varepsilon_{n,1}\rangle\, \varepsilon_{n,2}\ldots\varepsilon_{n,m_n}\rangle$$

  is a DNA expression (and in particular, an $\uparrow$-expression) satisfying $E' \equiv E$. Moreover, each occurrence of $\uparrow$ or $\downarrow$ in E' is alternating. In particular, E' has Property ($\mathcal{D}_{\text{Min}}$.4).

  As we observed before, the first argument ε_{1,m_1} of $\downarrow_1$ (which used to be the last argument of ε_1) is an $\updownarrow$-expression $\langle\updownarrow \alpha\rangle$ for an $\mathcal{N}$-word α. Analogously, the last argument $\varepsilon_{n,1}$ of $\downarrow_1$ is an $\updownarrow$-expression $\langle\updownarrow \alpha\rangle$ for an $\mathcal{N}$-word α. Clearly, $\downarrow_1$ has at least two arguments.[3] Because $m_1, m_n \geq 2$, the outermost operator $\uparrow_0$ of E' has at least three arguments. Now, it is easily verified that E' has Properties ($\mathcal{D}_{\text{Min}}$.1)–($\mathcal{D}_{\text{Min}}$.3) and ($\mathcal{D}_{\text{Min}}$.5), simply because E has these properties.

  Finally, because the first argument $\varepsilon_{1,1}$ of $\uparrow_0$ is either an $\mathcal{N}$-word α, or an $\updownarrow$-expression $\langle\updownarrow \alpha\rangle$ for an $\mathcal{N}$-word α, E' also has Property ($\mathcal{D}_{\text{Min}}$.6). We conclude that E' has all six properties from Lemma 8.22 and thus is minimal.

- Assume that ε_n is not an $\uparrow$-argument. Hence,

  $$E = \langle\downarrow \langle\uparrow \varepsilon_{1,1}\ldots\varepsilon_{1,m_1-1}\varepsilon_{1,m_1}\rangle\, \varepsilon_2\ldots\varepsilon_n\rangle\,,$$

  where ε_n is either an $\mathcal{N}$-word α or an $\updownarrow$-expression $\langle\updownarrow \alpha\rangle$ for an $\mathcal{N}$-word α. This implies that E also has Property ($\mathcal{D}_{\text{Min}}$.6), and thus is minimal itself.

  By Theorem 5.16(1) and (2),

  $$E' = \langle\uparrow \varepsilon_{1,1}\ldots\varepsilon_{1,m_1-1}\, \langle\downarrow \varepsilon_{1,m_1}\varepsilon_2\ldots\varepsilon_n\rangle\rangle$$

  is a DNA expression (and in particular, an $\uparrow$-expression) satisfying $E' \equiv E$. Each occurrence of $\uparrow$ or $\downarrow$ in E' is alternating. Because E is minimal and E' is equally long, E' is also minimal.

□

9.2 The algorithm for an example

In the previous section, we have illustrated each stage of our algorithm by some example DNA expressions. It is instructive, though, to see the effect of the algorithm as a whole for a single DNA expression. Therefore, we systematically work out the algorithm for the DNA expression E_1^* from (9.1). Step by step, we rewrite this DNA expression into an

[3] In fact, by Property ($\mathcal{D}_{\text{Min}}$.4). the two expression-arguments ε_{1,m_1} and $\varepsilon_{n,1}$ of $\downarrow_1$ must be separated by at least an $\mathcal{N}$-word-argument. Hence, the operator has at least three arguments.

equivalent, minimal DNA expression. For simplicity, whenever we have to consider certain arguments of a DNA expression 'in some order', we consider them from left to right. To visualize the effect of the algorithm on the structure of the DNA expression, we also give the structure trees of a number of the intermediate DNA expressions.

Recall that the algorithm is recursive: we first rewrite the expression-arguments of a DNA expression E into equivalent, minimal expression-arguments, and then consider E as a whole. For the structure tree of E, this means that it is reshaped in a bottom-up fashion.

Example 9.28 Let E be the DNA expression E_1^* from (9.1):

$$\begin{aligned} E = \langle\downarrow \langle\downarrow\ \langle\uparrow \langle\updownarrow \langle\downarrow\ \langle\updownarrow \langle\uparrow \alpha_1 \langle\updownarrow \langle\updownarrow \alpha_2\rangle\rangle \alpha_3 \langle\downarrow \langle\updownarrow \alpha_4\rangle \alpha_5\rangle\rangle\rangle\ \langle\updownarrow \alpha_6\rangle\ \alpha_7\ \rangle\rangle \\ \langle\downarrow \langle\updownarrow \alpha_8\rangle \langle\uparrow \langle\updownarrow \alpha_9\rangle \alpha_{10} \langle\updownarrow \alpha_{11}\rangle\rangle \alpha_{12} \langle\updownarrow \alpha_{13}\rangle\rangle\ \langle\updownarrow \alpha_{14}\rangle \\ \langle\downarrow \langle\updownarrow \alpha_{15}\rangle \alpha_{16} \langle\uparrow \langle\updownarrow \alpha_{17}\rangle \alpha_{18}\rangle\rangle\ \langle\uparrow \langle\updownarrow \alpha_{19}\rangle \langle\updownarrow \alpha_{20}\rangle\rangle\ \rangle\ \rangle \\ \langle\uparrow \langle\downarrow \alpha_{21}\rangle\rangle\ \langle\uparrow \langle\updownarrow \alpha_{22}\rangle \alpha_{23}\rangle\ \rangle\,, \end{aligned} \tag{9.15}$$

as depicted in Figure 9.6. When we apply the function `MakeMinimal` to E, we observe a cascade of recursive calls. The first time that E is actually rewritten, is when `MakeMinimal` is called for the DNA subexpression $E^s = \langle\updownarrow \langle\updownarrow \alpha_2\rangle\rangle$. This is an $\updownarrow$-expression with a minimal $\updownarrow$-argument. As we have seen in Example 9.1, by line 7 of `MakeMinimal`, E^s is simply substituted in E by its argument $\langle\updownarrow \alpha_2\rangle$, yielding

$$\begin{aligned} E = \langle\downarrow \langle\downarrow\ \langle\uparrow \langle\updownarrow \langle\downarrow\ \langle\updownarrow \langle\uparrow \alpha_1 \langle\updownarrow \alpha_2\rangle \alpha_3 \langle\downarrow \langle\updownarrow \alpha_4\rangle \alpha_5\rangle\rangle\rangle\ \langle\updownarrow \alpha_6\rangle\ \alpha_7\ \rangle\rangle \\ \langle\downarrow \langle\updownarrow \alpha_8\rangle \langle\uparrow \langle\updownarrow \alpha_9\rangle \alpha_{10} \langle\updownarrow \alpha_{11}\rangle\rangle \alpha_{12} \langle\updownarrow \alpha_{13}\rangle\rangle\ \langle\updownarrow \alpha_{14}\rangle \\ \langle\downarrow \langle\updownarrow \alpha_{15}\rangle \alpha_{16} \langle\uparrow \langle\updownarrow \alpha_{17}\rangle \alpha_{18}\rangle\rangle\ \langle\uparrow \langle\updownarrow \alpha_{19}\rangle \langle\updownarrow \alpha_{20}\rangle\rangle\ \rangle\ \rangle \\ \langle\uparrow \langle\downarrow \alpha_{21}\rangle\rangle\ \langle\uparrow \langle\updownarrow \alpha_{22}\rangle \alpha_{23}\rangle\ \rangle\,. \end{aligned}$$

We subsequently consider the DNA subexpression

$$E^s = \langle\updownarrow \langle\uparrow \alpha_1 \langle\updownarrow \alpha_2\rangle \alpha_3 \langle\downarrow \langle\updownarrow \alpha_4\rangle \alpha_5\rangle\rangle\rangle\,,$$

which is an $\updownarrow$-expression with a minimal, alternating $\uparrow$-argument. As we have seen in Example 9.18, by procedure `Make`$\updownarrow$`ExprMinimal`, E^s is substituted in E by $\langle\updownarrow \alpha_1\alpha_2\alpha_3\alpha_4 c(\alpha_5)\rangle$. This yields

$$\begin{aligned} E = \langle\downarrow \langle\downarrow\ \langle\uparrow \langle\updownarrow \langle\downarrow\ \langle\updownarrow \alpha_1\alpha_2\alpha_3\alpha_4 c(\alpha_5)\rangle\ \langle\updownarrow \alpha_6\rangle\ \alpha_7\ \rangle\rangle \\ \langle\downarrow \langle\updownarrow \alpha_8\rangle \langle\uparrow \langle\updownarrow \alpha_9\rangle \alpha_{10} \langle\updownarrow \alpha_{11}\rangle\rangle \alpha_{12} \langle\updownarrow \alpha_{13}\rangle\rangle\ \langle\updownarrow \alpha_{14}\rangle \\ \langle\downarrow \langle\updownarrow \alpha_{15}\rangle \alpha_{16} \langle\uparrow \langle\updownarrow \alpha_{17}\rangle \alpha_{18}\rangle\rangle\ \langle\uparrow \langle\updownarrow \alpha_{19}\rangle \langle\updownarrow \alpha_{20}\rangle\rangle\ \rangle\ \rangle \\ \langle\uparrow \langle\downarrow \alpha_{21}\rangle\rangle\ \langle\uparrow \langle\updownarrow \alpha_{22}\rangle \alpha_{23}\rangle\ \rangle\,, \end{aligned} \tag{9.16}$$

as depicted in Figure 9.7.

We subsequently consider the DNA subexpression

$$E^s = \langle\updownarrow \langle\downarrow\ \langle\updownarrow \alpha_1\alpha_2\alpha_3\alpha_4 c(\alpha_5)\rangle\ \langle\updownarrow \alpha_6\rangle\ \alpha_7\ \rangle\rangle\,,$$

which is an $\updownarrow$-expression, with a minimal, non-alternating $\downarrow$-argument. Hence, E^s violates Property ($\mathcal{D}_{\text{Min}}$.1). As we have seen in Example 9.19, by procedure `Make`$\updownarrow$`ExprMinimal`, E^s is substituted in E by $\langle\downarrow \langle\updownarrow \alpha_1\alpha_2\alpha_3\alpha_4 c(\alpha_5)\rangle \langle\updownarrow \alpha_6 c(\alpha_7)\rangle\rangle$. This yields

$$\begin{aligned} E = \langle\downarrow \langle\downarrow\ \langle\uparrow \langle\downarrow \langle\updownarrow \alpha_1\alpha_2\alpha_3\alpha_4 c(\alpha_5)\rangle \langle\updownarrow \alpha_6 c(\alpha_7)\rangle\rangle \\ \langle\downarrow \langle\updownarrow \alpha_8\rangle \langle\uparrow \langle\updownarrow \alpha_9\rangle \alpha_{10} \langle\updownarrow \alpha_{11}\rangle\rangle \alpha_{12} \langle\updownarrow \alpha_{13}\rangle\rangle\ \langle\updownarrow \alpha_{14}\rangle \\ \langle\downarrow \langle\updownarrow \alpha_{15}\rangle \alpha_{16} \langle\uparrow \langle\updownarrow \alpha_{17}\rangle \alpha_{18}\rangle\rangle\ \langle\uparrow \langle\updownarrow \alpha_{19}\rangle \langle\updownarrow \alpha_{20}\rangle\rangle\ \rangle\ \rangle \\ \langle\uparrow \langle\downarrow \alpha_{21}\rangle\rangle\ \langle\uparrow \langle\updownarrow \alpha_{22}\rangle \alpha_{23}\rangle\ \rangle\,, \end{aligned} \tag{9.17}$$

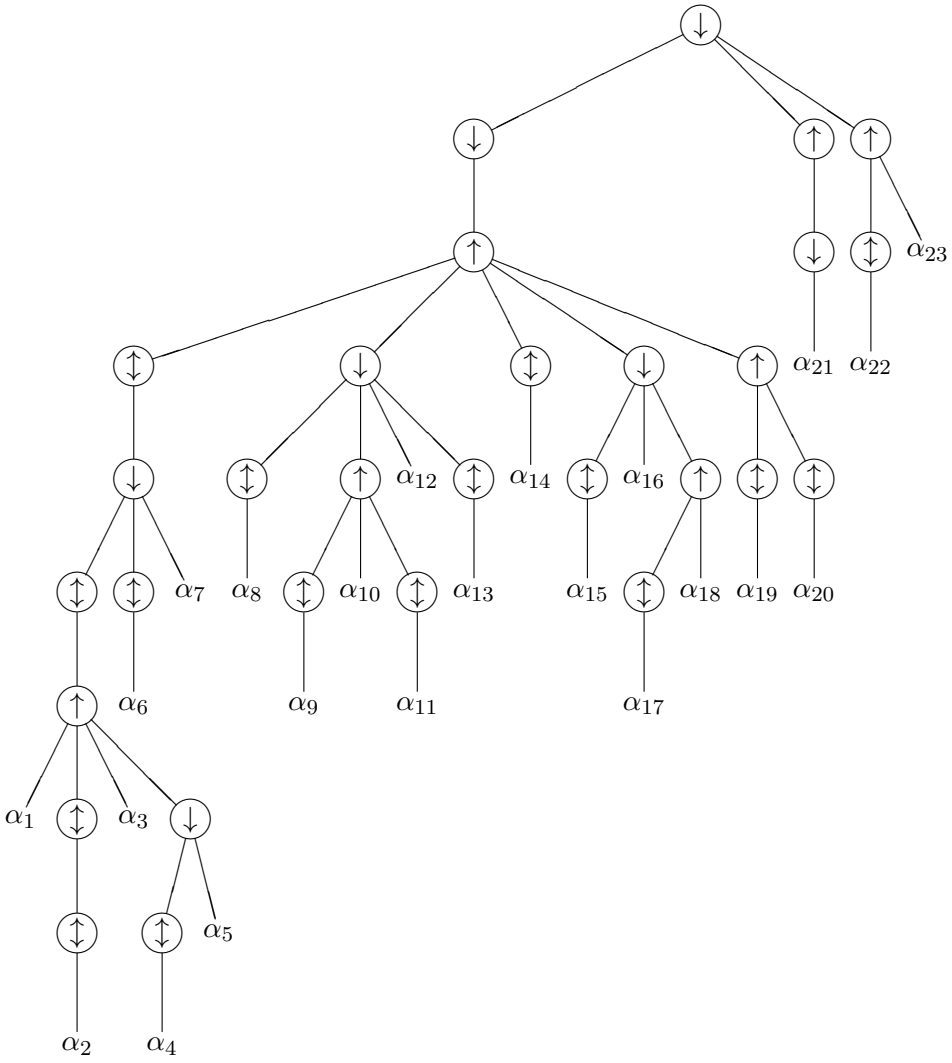


Figure 9.6: Structure tree of the example DNA expression E_1^* for the algorithm for minimality, (9.15).

as depicted in Figure 9.8.

We subsequently consider the DNA subexpression

$$E^s = \big\langle\uparrow\ \langle\downarrow\ \langle\updownarrow\ \alpha_1\alpha_2\alpha_3\alpha_4 c(\alpha_5)\rangle\ \langle\updownarrow\ \alpha_6 c(\alpha_7)\rangle\rangle \\ \langle\downarrow\ \langle\updownarrow\ \alpha_8\rangle\ \langle\uparrow\ \langle\updownarrow\ \alpha_9\rangle\ \alpha_{10}\ \langle\updownarrow\ \alpha_{11}\rangle\rangle\ \alpha_{12}\ \langle\updownarrow\ \alpha_{13}\rangle\rangle\ \ \langle\updownarrow\ \alpha_{14}\rangle \\ \langle\downarrow\ \langle\updownarrow\ \alpha_{15}\rangle\ \alpha_{16}\ \langle\uparrow\ \langle\updownarrow\ \alpha_{17}\rangle\ \alpha_{18}\rangle\rangle\ \ \langle\uparrow\ \langle\updownarrow\ \alpha_{19}\rangle\ \langle\updownarrow\ \alpha_{20}\rangle\rangle\ \big\rangle\,,$$

which is an $\uparrow$-expression with five minimal expression-arguments. In the second for-loop of `MakeMinimal`, we consider $\downarrow$-arguments of E^s that are not alternating. There are two such arguments, viz the first two arguments. As we have seen in Example 9.22, by procedure `Denickify`, the first argument

$$E_1 = \langle\downarrow\ \langle\updownarrow\ \alpha_1\alpha_2\alpha_3\alpha_4 c(\alpha_5)\rangle\ \langle\updownarrow\ \alpha_6 c(\alpha_7)\rangle\rangle$$

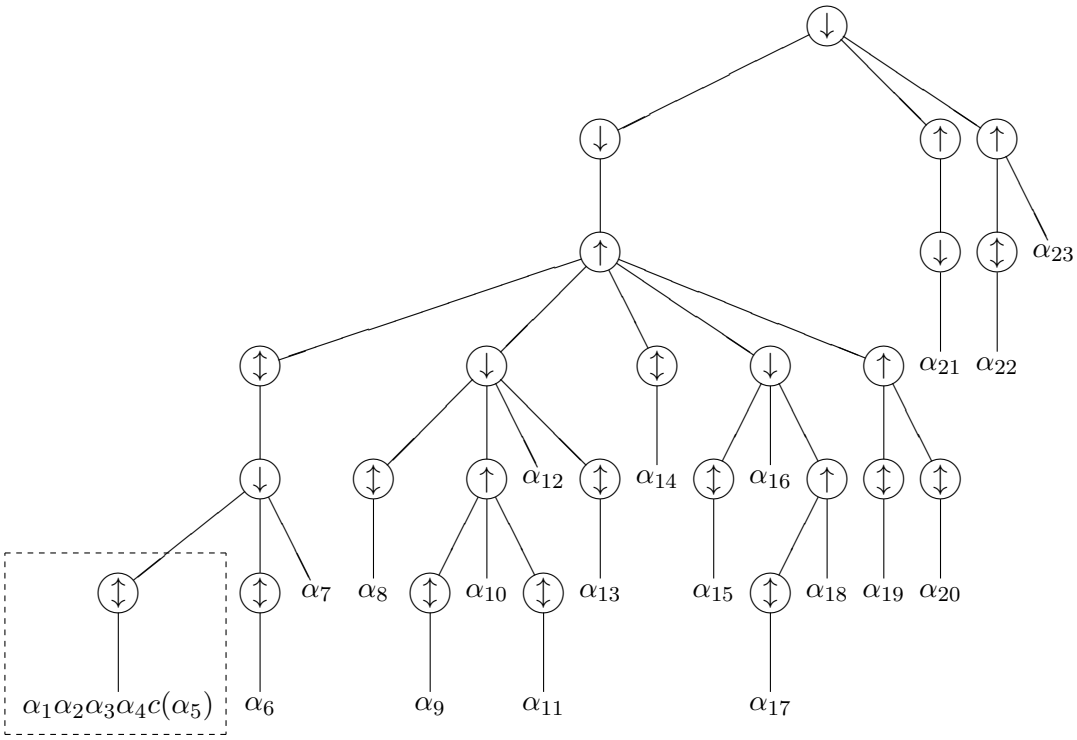


Figure 9.7: Structure tree of the example DNA expression for the algorithm for minimality, after two substitutions, (9.16). The dashed box encloses the part of the tree that has changed as compared to Figure 9.6.

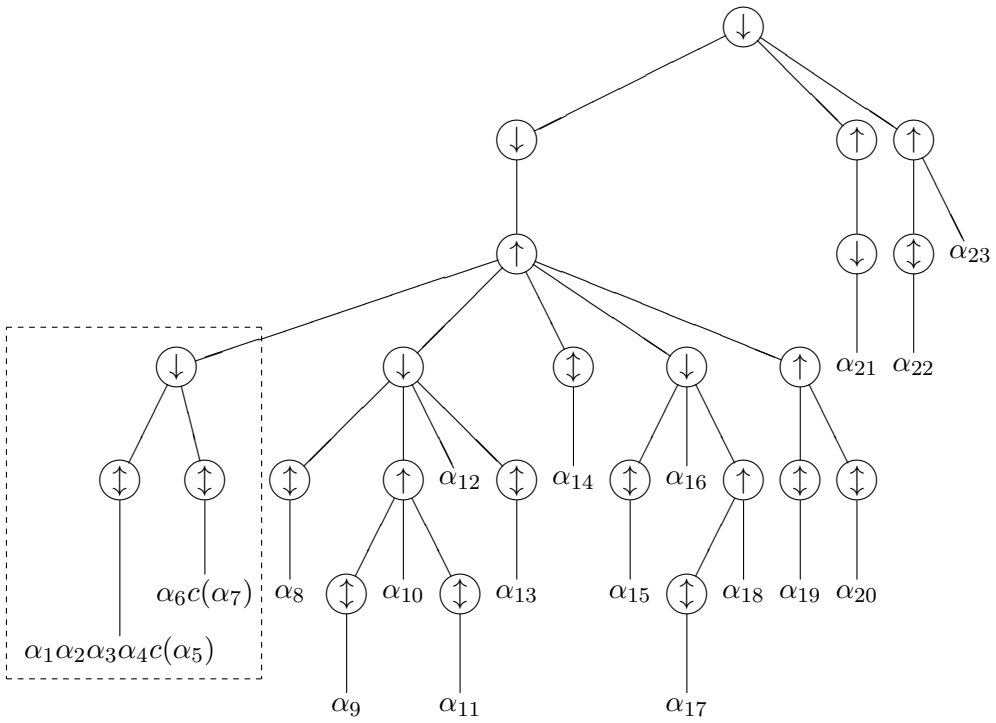


Figure 9.8: Structure tree of the example DNA expression for the algorithm for minimality, after three substitutions, (9.17). The dashed box encloses the part of the tree that has changed as compared to Figure 9.7.

is substituted in E by $\langle\updownarrow \alpha_1\alpha_2\alpha_3\alpha_4 c(\alpha_5)\alpha_6 c(\alpha_7)\rangle$, yielding

$$\begin{aligned} E = \langle\downarrow \langle\downarrow\ \langle\uparrow \langle\updownarrow \alpha_1\alpha_2\alpha_3\alpha_4 c(\alpha_5)\alpha_6 c(\alpha_7)\rangle \\ &\quad \langle\downarrow \langle\updownarrow \alpha_8\rangle \langle\uparrow \langle\updownarrow \alpha_9\rangle \alpha_{10} \langle\updownarrow \alpha_{11}\rangle\rangle \alpha_{12} \langle\updownarrow \alpha_{13}\rangle\rangle\ \langle\updownarrow \alpha_{14}\rangle \\ &\quad \langle\downarrow \langle\updownarrow \alpha_{15}\rangle \alpha_{16} \langle\uparrow \langle\updownarrow \alpha_{17}\rangle \alpha_{18}\rangle\rangle\ \langle\uparrow \langle\updownarrow \alpha_{19}\rangle \langle\updownarrow \alpha_{20}\rangle\rangle \rangle \rangle \\ &\quad \langle\uparrow \langle\downarrow \alpha_{21}\rangle\rangle\ \langle\uparrow \langle\updownarrow \alpha_{22}\rangle \alpha_{23}\rangle \rangle . \end{aligned}$$

We also apply procedure `Denickify` to the second argument

$$E_2 = \langle\downarrow \langle\updownarrow \alpha_8\rangle \langle\uparrow \langle\updownarrow \alpha_9\rangle \alpha_{10} \langle\updownarrow \alpha_{11}\rangle\rangle \alpha_{12} \langle\updownarrow \alpha_{13}\rangle\rangle$$

of E^s. The first two arguments of this minimal $\downarrow$-expression are consecutive expression-arguments. In the only iteration of the while-loop in procedure `Denickify`, these arguments are merged according to line Dni.14. The result is

$$\widehat{E}_2 = \langle\downarrow \langle\uparrow \langle\updownarrow \alpha_8\alpha_9\rangle \alpha_{10} \langle\updownarrow \alpha_{11}\rangle\rangle \alpha_{12} \langle\updownarrow \alpha_{13}\rangle\rangle .$$

This $\downarrow$-expression has more than one argument and its last argument is not an $\uparrow$-argument. Hence, $\widehat{E}_2$ is not modified any further in procedure `Denickify` (cf. Example 9.4). When we substitute E_2 in E by $\widehat{E}_2$, we obtain

$$\begin{aligned} E = \langle\downarrow \langle\downarrow\ \langle\uparrow \langle\updownarrow \alpha_1\alpha_2\alpha_3\alpha_4 c(\alpha_5)\alpha_6 c(\alpha_7)\rangle \\ &\quad \langle\downarrow \langle\uparrow \langle\updownarrow \alpha_8\alpha_9\rangle \alpha_{10} \langle\updownarrow \alpha_{11}\rangle\rangle \alpha_{12} \langle\updownarrow \alpha_{13}\rangle\rangle\ \langle\updownarrow \alpha_{14}\rangle \\ &\quad \langle\downarrow \langle\updownarrow \alpha_{15}\rangle \alpha_{16} \langle\uparrow \langle\updownarrow \alpha_{17}\rangle \alpha_{18}\rangle\rangle\ \langle\uparrow \langle\updownarrow \alpha_{19}\rangle \langle\updownarrow \alpha_{20}\rangle\rangle \rangle \rangle \\ &\quad \langle\uparrow \langle\downarrow \alpha_{21}\rangle\rangle\ \langle\uparrow \langle\updownarrow \alpha_{22}\rangle \alpha_{23}\rangle \rangle , \end{aligned} \tag{9.18}$$

as depicted in Figure 9.9. In this overall DNA expression, the DNA subexpression E^s that we consider has become

$$\begin{aligned} E^s = \langle\uparrow \langle\updownarrow \alpha_1\alpha_2\alpha_3\alpha_4 c(\alpha_5)\alpha_6 c(\alpha_7)\rangle \\ &\quad \langle\downarrow \langle\uparrow \langle\updownarrow \alpha_8\alpha_9\rangle \alpha_{10} \langle\updownarrow \alpha_{11}\rangle\rangle \alpha_{12} \langle\updownarrow \alpha_{13}\rangle\rangle\ \langle\updownarrow \alpha_{14}\rangle \\ &\quad \langle\downarrow \langle\updownarrow \alpha_{15}\rangle \alpha_{16} \langle\uparrow \langle\updownarrow \alpha_{17}\rangle \alpha_{18}\rangle\rangle\ \langle\uparrow \langle\updownarrow \alpha_{19}\rangle \langle\updownarrow \alpha_{20}\rangle\rangle \rangle . \end{aligned}$$

We proceed with the third for-loop of `MakeMinimal`, in which we consider $\downarrow$-arguments E_i of E^s, such that either the first argument, or the last argument of E_i is an $\uparrow$-argument. There are two such arguments, viz the (new) second argument and the fourth argument. As we have seen in Example 9.25, by procedure `RotateToMinimal`, the second argument

$$E_2 = \langle\downarrow \langle\uparrow \langle\updownarrow \alpha_8\alpha_9\rangle \alpha_{10} \langle\updownarrow \alpha_{11}\rangle\rangle \alpha_{12} \langle\updownarrow \alpha_{13}\rangle\rangle$$

is substituted in E by

$$\langle\uparrow \langle\updownarrow \alpha_8\alpha_9\rangle \alpha_{10} \langle\downarrow \langle\updownarrow \alpha_{11}\rangle \alpha_{12} \langle\updownarrow \alpha_{13}\rangle\rangle\rangle ,$$

yielding

$$\begin{aligned} E = \langle\downarrow \langle\downarrow\ \langle\uparrow \langle\updownarrow \alpha_1\alpha_2\alpha_3\alpha_4 c(\alpha_5)\alpha_6 c(\alpha_7)\rangle \\ &\quad \langle\uparrow \langle\updownarrow \alpha_8\alpha_9\rangle \alpha_{10} \langle\downarrow \langle\updownarrow \alpha_{11}\rangle \alpha_{12} \langle\updownarrow \alpha_{13}\rangle\rangle\rangle\ \langle\updownarrow \alpha_{14}\rangle \\ &\quad \langle\downarrow \langle\updownarrow \alpha_{15}\rangle \alpha_{16} \langle\uparrow \langle\updownarrow \alpha_{17}\rangle \alpha_{18}\rangle\rangle\ \langle\uparrow \langle\updownarrow \alpha_{19}\rangle \langle\updownarrow \alpha_{20}\rangle\rangle \rangle \rangle \\ &\quad \langle\uparrow \langle\downarrow \alpha_{21}\rangle\rangle\ \langle\uparrow \langle\updownarrow \alpha_{22}\rangle \alpha_{23}\rangle \rangle . \end{aligned}$$

We also apply procedure `RotateToMinimal` to the fourth argument

$$E_4 = \langle\downarrow \langle\updownarrow \alpha_{15}\rangle \alpha_{16} \langle\uparrow \langle\updownarrow \alpha_{17}\rangle \alpha_{18}\rangle\rangle .$$

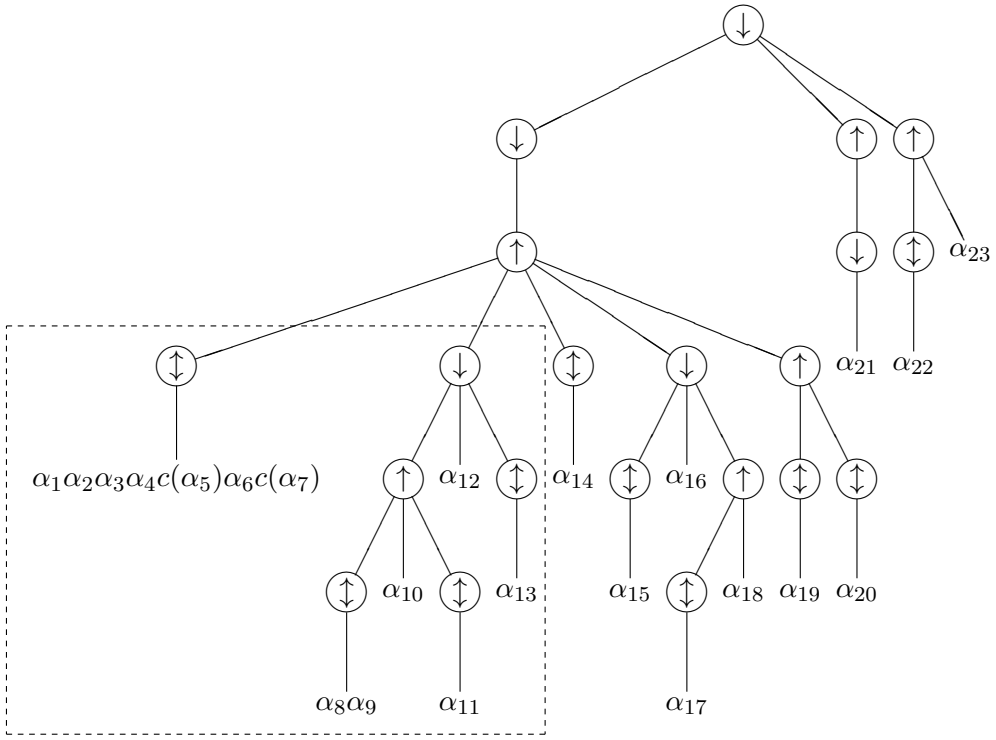


Figure 9.9: Structure tree of the example DNA expression for the algorithm for minimality, after five substitutions, (9.18). The dashed box encloses the part of the tree that has changed as compared to Figure 9.8.

The $\downarrow$-expression E_4 is minimal and alternating, its first argument is not an $\uparrow$-argument, but its last argument is an $\uparrow$-argument. According to line RtM.9, E_4 is substituted in E by

$$\langle\uparrow \langle\downarrow \langle\updownarrow \alpha_{15}\rangle\, \alpha_{16}\, \langle\updownarrow \alpha_{17}\rangle\rangle\, \alpha_{18}\rangle$$

(cf. Example 9.6). This yields

$$\begin{aligned} E = \big\langle\downarrow \big\langle\downarrow\ \big\langle\uparrow\ &\langle\updownarrow \alpha_1\alpha_2\alpha_3\alpha_4 c(\alpha_5)\alpha_6 c(\alpha_7)\rangle \\ &\langle\uparrow \langle\updownarrow \alpha_8\alpha_9\rangle\, \alpha_{10}\, \langle\downarrow \langle\updownarrow \alpha_{11}\rangle\, \alpha_{12}\, \langle\updownarrow \alpha_{13}\rangle\rangle\rangle\ \ \langle\updownarrow \alpha_{14}\rangle \\ &\langle\uparrow \langle\downarrow \langle\updownarrow \alpha_{15}\rangle\, \alpha_{16}\, \langle\updownarrow \alpha_{17}\rangle\rangle\, \alpha_{18}\rangle\ \ \langle\uparrow \langle\updownarrow \alpha_{19}\rangle\, \langle\updownarrow \alpha_{20}\rangle\rangle\ \big\rangle\ \big\rangle \\ \big\langle\uparrow \langle\downarrow \alpha_{21}\rangle\rangle\ \ &\langle\uparrow \langle\updownarrow \alpha_{22}\rangle\, \alpha_{23}\rangle\ \big\rangle\,, \end{aligned} \tag{9.19}$$

as depicted in Figure 9.10. In this overall DNA expression, the DNA subexpression E^s that we consider has become

$$\begin{aligned} E^s = \big\langle\uparrow\ &\langle\updownarrow \alpha_1\alpha_2\alpha_3\alpha_4 c(\alpha_5)\alpha_6 c(\alpha_7)\rangle \\ &\langle\uparrow \langle\updownarrow \alpha_8\alpha_9\rangle\, \alpha_{10}\, \langle\downarrow \langle\updownarrow \alpha_{11}\rangle\, \alpha_{12}\, \langle\updownarrow \alpha_{13}\rangle\rangle\rangle\ \ \langle\updownarrow \alpha_{14}\rangle \\ &\langle\uparrow \langle\downarrow \langle\updownarrow \alpha_{15}\rangle\, \alpha_{16}\, \langle\updownarrow \alpha_{17}\rangle\rangle\, \alpha_{18}\rangle\ \ \langle\uparrow \langle\updownarrow \alpha_{19}\rangle\, \langle\updownarrow \alpha_{20}\rangle\rangle\ \big\rangle\,. \end{aligned}$$

We proceed with the fourth for-loop of `MakeMinimal`. As we have seen in Example 9.8, in this loop, we substitute the three $\uparrow$-arguments of E^s by their respective arguments. The overall DNA expression resulting from these three substitutions is

$$\begin{aligned} E = \big\langle\downarrow \big\langle\downarrow\ \big\langle\uparrow\ &\langle\updownarrow \alpha_1\alpha_2\alpha_3\alpha_4 c(\alpha_5)\alpha_6 c(\alpha_7)\rangle \\ &\langle\updownarrow \alpha_8\alpha_9\rangle\ \ \alpha_{10}\ \ \langle\downarrow \langle\updownarrow \alpha_{11}\rangle\, \alpha_{12}\, \langle\updownarrow \alpha_{13}\rangle\rangle\ \ \langle\updownarrow \alpha_{14}\rangle \\ &\langle\downarrow \langle\updownarrow \alpha_{15}\rangle\, \alpha_{16}\, \langle\updownarrow \alpha_{17}\rangle\rangle\ \ \alpha_{18}\ \ \langle\updownarrow \alpha_{19}\rangle\ \ \langle\updownarrow \alpha_{20}\rangle\ \big\rangle\ \big\rangle \\ \big\langle\uparrow \langle\downarrow \alpha_{21}\rangle\rangle\ \ &\langle\uparrow \langle\updownarrow \alpha_{22}\rangle\, \alpha_{23}\rangle\ \big\rangle\,, \end{aligned} \tag{9.20}$$

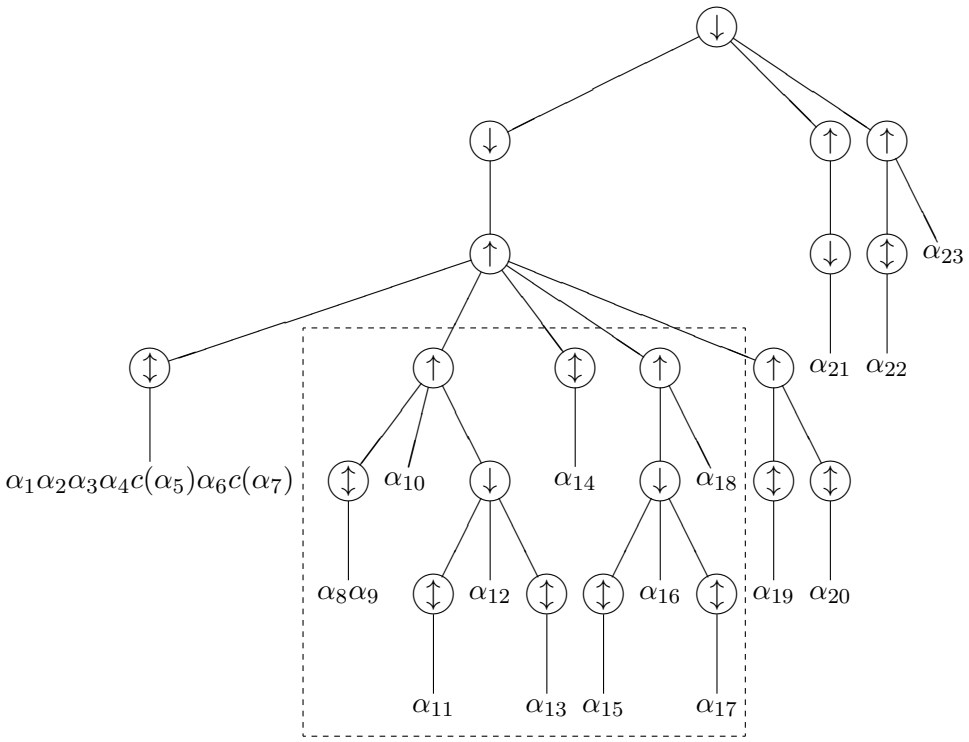


Figure 9.10: Structure tree of the example DNA expression for the algorithm for minimality, after seven substitutions, (9.19). The dashed box encloses the part of the tree that has changed as compared to Figure 9.9.

as depicted in Figure 9.11. As we have seen in Example 9.9, the DNA subexpression E^s of E that we consider is not modified any further in its own call of `MakeMinimal`.

We subsequently consider the DNA subexpression

$$\begin{aligned} E^s = \langle\downarrow\ \langle\uparrow\ &\langle\updownarrow\,\alpha_1\alpha_2\alpha_3\alpha_4c(\alpha_5)\alpha_6c(\alpha_7)\rangle \\ &\langle\updownarrow\,\alpha_8\alpha_9\rangle\ \alpha_{10}\ \langle\downarrow\,\langle\updownarrow\,\alpha_{11}\rangle\,\alpha_{12}\,\langle\updownarrow\,\alpha_{13}\rangle\rangle\ \langle\updownarrow\,\alpha_{14}\rangle \\ &\langle\downarrow\,\langle\updownarrow\,\alpha_{15}\rangle\,\alpha_{16}\,\langle\updownarrow\,\alpha_{17}\rangle\rangle\ \alpha_{18}\ \langle\updownarrow\,\alpha_{19}\rangle\ \langle\updownarrow\,\alpha_{20}\rangle\ \rangle\ \rangle\,, \end{aligned}$$

which is a $\downarrow$-expression whose only argument is a minimal $\uparrow$-argument E_1 that is not alternating. Hence, E_1 makes E^s violate Property ($\mathcal{D}_{\text{Min}}$.4). In the second for-loop of `MakeMinimal`, we make this argument nick free. In particular, as we have seen in Example 9.23, by procedure `Denickify`, E_1 is substituted in E by

$$\begin{aligned} &\langle\uparrow\ \langle\updownarrow\,\alpha_1\alpha_2\alpha_3\alpha_4c(\alpha_5)\alpha_6c(\alpha_7)\alpha_8\alpha_9\rangle\ \alpha_{10} \\ &\quad\langle\downarrow\,\langle\updownarrow\,\alpha_{11}\rangle\,\alpha_{12}\,\langle\updownarrow\,\alpha_{13}\alpha_{14}\alpha_{15}\rangle\,\alpha_{16}\,\langle\updownarrow\,\alpha_{17}\rangle\rangle\ \alpha_{18}\ \langle\updownarrow\,\alpha_{19}\alpha_{20}\rangle\ \rangle\,. \end{aligned}$$

This yields

$$\begin{aligned} E = \langle\downarrow\,\langle\downarrow\ \langle\uparrow\ &\langle\updownarrow\,\alpha_1\alpha_2\alpha_3\alpha_4c(\alpha_5)\alpha_6c(\alpha_7)\alpha_8\alpha_9\rangle\ \alpha_{10} \\ &\langle\downarrow\,\langle\updownarrow\,\alpha_{11}\rangle\,\alpha_{12}\,\langle\updownarrow\,\alpha_{13}\alpha_{14}\alpha_{15}\rangle\,\alpha_{16}\,\langle\updownarrow\,\alpha_{17}\rangle\rangle\ \alpha_{18}\ \langle\updownarrow\,\alpha_{19}\alpha_{20}\rangle\ \rangle\ \rangle \\ &\langle\uparrow\,\langle\downarrow\,\alpha_{21}\rangle\rangle\ \langle\uparrow\,\langle\updownarrow\,\alpha_{22}\rangle\,\alpha_{23}\rangle\ \rangle\,, \end{aligned} \tag{9.21}$$

as depicted in Figure 9.12. In this overall DNA expression, the DNA expression E^s that we consider has become

$$\begin{aligned} E^s = \langle\downarrow\ \langle\uparrow\ &\langle\updownarrow\,\alpha_1\alpha_2\alpha_3\alpha_4c(\alpha_5)\alpha_6c(\alpha_7)\alpha_8\alpha_9\rangle\ \alpha_{10} \\ &\langle\downarrow\,\langle\updownarrow\,\alpha_{11}\rangle\,\alpha_{12}\,\langle\updownarrow\,\alpha_{13}\alpha_{14}\alpha_{15}\rangle\,\alpha_{16}\,\langle\updownarrow\,\alpha_{17}\rangle\rangle\ \alpha_{18}\ \langle\updownarrow\,\alpha_{19}\alpha_{20}\rangle\ \rangle\ \rangle\,. \end{aligned}$$

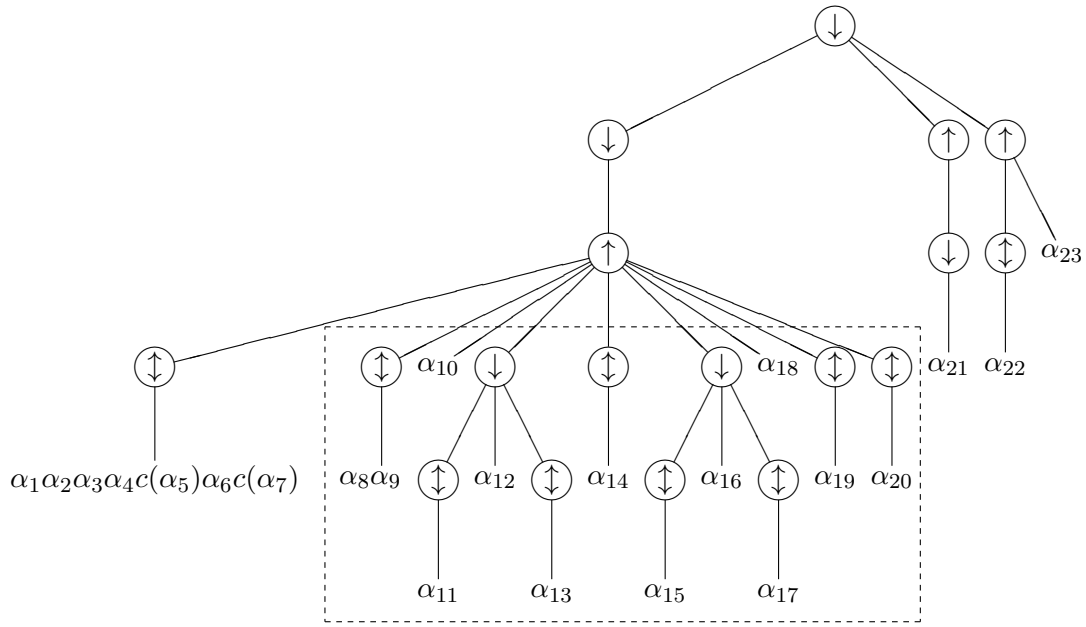


Figure 9.11: Structure tree of the example DNA expression for the algorithm for minimality, after ten substitutions, (9.20). The dashed box encloses the part of the tree that has changed as compared to Figure 9.10.

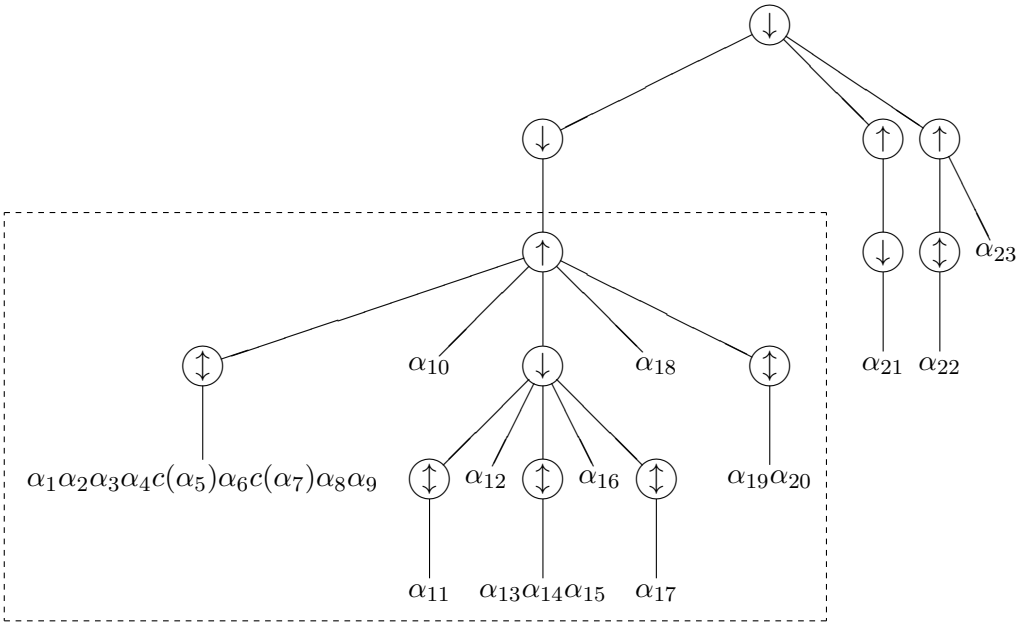


Figure 9.12: Structure tree of the example DNA expression for the algorithm for minimality, after eleven substitutions, (9.21). The dashed box encloses the part of the tree that has changed as compared to Figure 9.11.

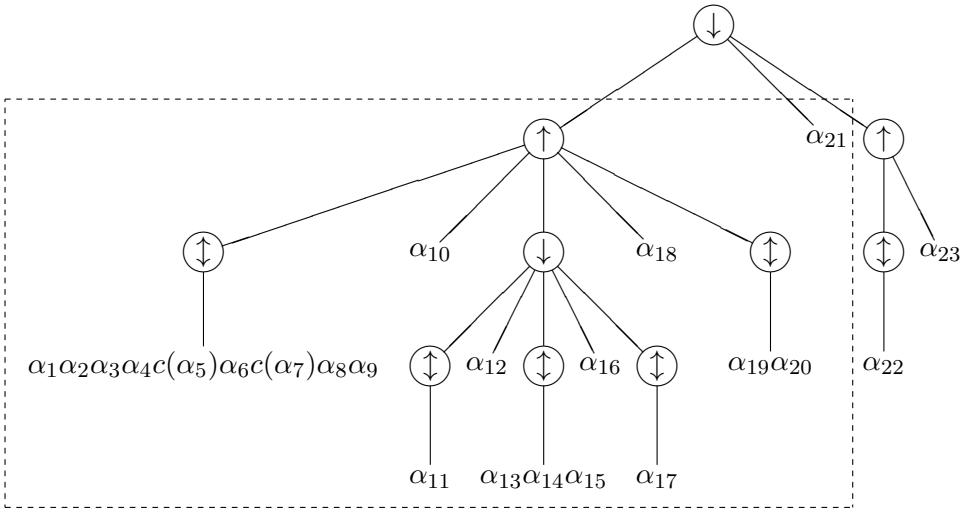


Figure 9.13: Structure tree of the example DNA expression for the algorithm for minimality, after fourteen substitutions, (9.22). The dashed box encloses the part of the tree that has changed as compared to Figure 9.12.

The third and the fourth for-loop of `MakeMinimal` do not affect E^s. In the if-then-else construction at the end of the function, we observe that E^s still has one argument, which is a DNA expression. Hence, E^s violates Property ($\mathcal{D}_{\text{Min}}$.3). According to line 31, E^s is substituted in E by this expression-argument, yielding

$$\begin{aligned} E = \langle\downarrow \langle\uparrow \langle\updownarrow \alpha_1\alpha_2\alpha_3\alpha_4 c(\alpha_5)\alpha_6 c(\alpha_7)\alpha_8\alpha_9\rangle\ \alpha_{10} \\ \langle\downarrow \langle\updownarrow \alpha_{11}\rangle\, \alpha_{12}\, \langle\updownarrow \alpha_{13}\alpha_{14}\alpha_{15}\rangle\, \alpha_{16}\, \langle\updownarrow \alpha_{17}\rangle\rangle\ \alpha_{18}\ \langle\updownarrow \alpha_{19}\alpha_{20}\rangle\ \rangle \\ \langle\uparrow \langle\downarrow \alpha_{21}\rangle\rangle\ \langle\uparrow \langle\updownarrow \alpha_{22}\rangle\, \alpha_{23}\rangle\ \rangle\,. \end{aligned}$$

We subsequently consider the DNA subexpression $E^s = \langle\uparrow \langle\downarrow \alpha_{21}\rangle\rangle$, which is an $\uparrow$-expression whose only argument is the minimal, alternating $\downarrow$-argument $\langle\downarrow \alpha_{21}\rangle$. The for-loops of `MakeMinimal` do not affect E^s. As we have seen in Example 9.10, by the if-then-else construction at the end of the function, E^s is substituted in E by its argument $\langle\downarrow \alpha_{21}\rangle$, yielding

$$\begin{aligned} E = \langle\downarrow \langle\uparrow \langle\updownarrow \alpha_1\alpha_2\alpha_3\alpha_4 c(\alpha_5)\alpha_6 c(\alpha_7)\alpha_8\alpha_9\rangle\ \alpha_{10} \\ \langle\downarrow \langle\updownarrow \alpha_{11}\rangle\, \alpha_{12}\, \langle\updownarrow \alpha_{13}\alpha_{14}\alpha_{15}\rangle\, \alpha_{16}\, \langle\updownarrow \alpha_{17}\rangle\rangle\ \alpha_{18}\ \langle\updownarrow \alpha_{19}\alpha_{20}\rangle\ \rangle \\ \langle\downarrow \alpha_{21}\rangle\ \langle\uparrow \langle\updownarrow \alpha_{22}\rangle\, \alpha_{23}\rangle\ \rangle\,. \end{aligned}$$

At last, we consider E itself, which is a $\downarrow$-expression with three minimal expression-arguments. The second and the third for-loop of `MakeMinimal` do not affect E. In the fourth for-loop, we discover that the second argument of E is the $\downarrow$-expression $\langle\downarrow \alpha_{21}\rangle$. Hence, E violates Property ($\mathcal{D}_{\text{Min}}$.2). According to line 27, we substitute $\langle\downarrow \alpha_{21}\rangle$ in E by its own argument α_{21}. This yields

$$\begin{aligned} E = \langle\downarrow \langle\uparrow \langle\updownarrow \alpha_1\alpha_2\alpha_3\alpha_4 c(\alpha_5)\alpha_6 c(\alpha_7)\alpha_8\alpha_9\rangle\ \alpha_{10} \\ \langle\downarrow \langle\updownarrow \alpha_{11}\rangle\, \alpha_{12}\, \langle\updownarrow \alpha_{13}\alpha_{14}\alpha_{15}\rangle\, \alpha_{16}\, \langle\updownarrow \alpha_{17}\rangle\rangle\ \alpha_{18}\ \langle\updownarrow \alpha_{19}\alpha_{20}\rangle\ \rangle \\ \alpha_{21}\ \langle\uparrow \langle\updownarrow \alpha_{22}\rangle\, \alpha_{23}\rangle\ \rangle\,, \end{aligned} \tag{9.22}$$

as depicted in Figure 9.13. In the if-then-else construction at the end of `MakeMinimal`, we observe that the $\downarrow$-expression E has more than one argument, is alternating and that both

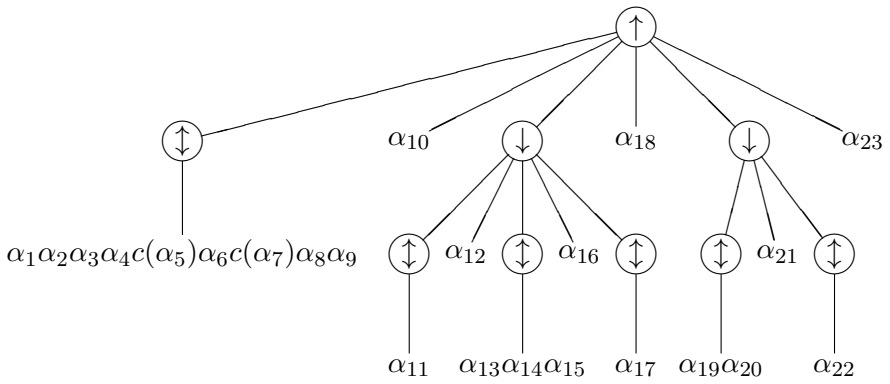


Figure 9.14: Structure tree of the example DNA expression E_2^* for the algorithm for minimality, after all fifteen substitutions, (9.23).

its first argument and its last argument are ↑-arguments. Hence, according to line 35, we apply procedure `RotateToMinimal` to E. As we have seen in Example 9.26, E is substituted by

$$\begin{aligned} E' = \big\langle\uparrow\ \langle\updownarrow\ \alpha_1\alpha_2\alpha_3\alpha_4c(\alpha_5)\alpha_6c(\alpha_7)\alpha_8\alpha_9\rangle\ \ \alpha_{10} \\ \langle\downarrow\ \langle\updownarrow\ \alpha_{11}\rangle\ \alpha_{12}\ \langle\updownarrow\ \alpha_{13}\alpha_{14}\alpha_{15}\rangle\ \alpha_{16}\ \langle\updownarrow\ \alpha_{17}\rangle\rangle\ \ \alpha_{18} \\ \langle\downarrow\ \langle\updownarrow\ \alpha_{19}\alpha_{20}\rangle\ \alpha_{21}\ \langle\updownarrow\ \alpha_{22}\rangle\rangle\ \ \alpha_{23}\ \big\rangle\,. \end{aligned} \tag{9.23}$$

This is the final result E_2^* of the algorithm, and has been depicted in Figure 9.14.

Note that we encountered the DNA expressions E from (9.22) and $E_2^* = E'$ from (9.23) also in Example 9.11. There, E_2^* turned out to be one of the two minimal DNA expressions that are equivalent to E. However, we obtained these two minimal DNA expressions from the *semantics of* E, using the recursive construction from Theorem 7.24. In the present example, we have simply performed *string manipulations on* E *itself*, as prescribed by procedure `RotateToMinimal`. ■

In the above example, when we compare the original DNA expression E_1^* and its structure tree (in Figure 9.6) to the final DNA expression E_2^* and its structure tree (in Figure 9.14), we can conclude that the latter are not only smaller, but also much better readable.

One may be surprised by some of the differences between the two structure trees. For example, in the original tree, the $\mathcal{N}$-words $\alpha_1, \ldots, \alpha_7$ are in one subtree, and the $\mathcal{N}$-words $\alpha_8, \ldots, \alpha_{13}$ are in an adjacent subtree. In the final tree, α_8 and α_9 have joined $\alpha_1, \ldots, \alpha_7$, whereas $\alpha_{10}, \ldots, \alpha_{13}$ have not made this move.

At first sight, one might think that it requires very complex steps to achieve such changes. This is, however, not the case. Every substitution performed by the algorithm corresponds to a relatively simple, local rearrangement of the structure tree.

The substitution that probably has the largest effect on the tree, is the one in line M↕M.5 of procedure `Make↕ExprMinimal`. There, we substitute a ↓-expression $E_{1,i}$ by the ↕-expression $\langle\updownarrow\ \alpha_{E_{1,i}}\rangle$. The effect on the structure tree is that the subtree corresponding to $E_{1,i}$ must be replaced by a node labelled by ↕, with a child node labelled by $\alpha_{E_{1,i}}$. This $\mathcal{N}$-word $\alpha_{E_{1,i}}$ is the concatenation of all $\mathcal{N}$-words (possibly complemented) in the leaves of the subtree of $E_{1,i}$. Since this effect is restricted to a subtree of the total structure tree, and is uniform for the entire subtree, we may also view this as a local change.

In our running example, we have used line M↕M.5 of `Make↕ExprMinimal` once, in Example 9.18. There, we substituted the (small) ↓-expression $E_{1,i} = \langle\downarrow\ \langle\updownarrow\ \alpha_4\rangle\ \alpha_5\rangle$ by

$\langle \updownarrow \alpha_{E_{1,i}} \rangle = \langle \updownarrow \alpha_4 c(\alpha_5) \rangle$. Part of the difference between the structure trees in Figure 9.6 and Figure 9.7 can be traced back to this substitution.

9.3 Detailed implementation and complexity of the algorithm

In Section 9.1, we have described an algorithm for rewriting an arbitrary DNA expression into an equivalent, minimal DNA expression, and we have proved that the algorithm is correct. However, we have not specified all details of the algorithm. We now work out these details in an implementation of the algorithm. The details concerning the data structure for the DNA expression have immediate consequences for the complexity of the algorithm. Therefore, we discuss these details in the context of an analysis of the complexity.

The recursive function `MakeMinimal` contains four successive for-loops. In each for-loop we consider (some of) the expression-arguments of E 'in some order.' Because different expression-arguments are rewritten independently, the actual orders used within the loops do not influence the result. We choose to consider the expression-arguments in the order of their occurrence in the DNA expression, like we did in Example 9.28. This is the most natural order.

The fact that different expression-arguments are rewritten independently, also implies that the aggregate effect of the four loops on a particular expression-argument E_i of E depends only on E_i itself. We can as well perform all operations (at most four) on E_i first, before proceeding to the next expression-argument. This way, the four for-loops can be replaced by a single for-loop. The conceptual advantage of this is that each expression-argument is considered only once, instead of (at most) four times.

When we modify the algorithm in the above way, and also refer to the procedures `Make↕ExprMinimal`, `Denickify` and `RotateToMinimal` more directly, we obtain the function in Figure 9.15.

One may wonder why we did not use a single for-loop in `MakeMinimal` from the very beginning. The reason is, that it is easier to formulate invariants for the four separate loops, than it would be for a single loop with all four types of substitutions. We need such invariants to prove the correctness of `MakeMinimal`, see the proof of Theorem 9.17.

We now examine the time complexity of the algorithm. In our analysis, we will frequently use the big O notation. For example, we will say that the time spent in (a specific part of) the algorithm for a given DNA expression E *is in* $\mathcal{O}(|E|)$. Recall from Section 2.1, that in this case, in order to conclude that this time really *is linear* in $|E|$, we have to establish that $|E|$ also provides a lower bound for the growth rate.

When we apply the function `MakeMinimal` to a DNA expression E, all arguments of E are examined individually. By a cascade of recursive calls of the function, the expression-arguments of E are examined up to the highest nesting level of the brackets. This way, in principle every letter of E is considered. Hence, the time required for executing the function is at least linear in the length of E.

We demonstrate that the function can indeed be executed in linear time, if we use a proper data structure to store E in. We now discuss two features of a possible, proper data structure. We later introduce two more features.

First, it is useful to store (the letters of) E in a doubly-linked list. Then letters can be inserted and removed in constant time. For example, in line $7'$ of the function,

```
1′.  MakeMinimal (E)
       // recursively rewrites an arbitrary DNA expression E
       // into an equivalent, minimal DNA expression
2′.  {
3′.    if (E is an ↕-expression)
4′.    then if (the argument of E is a DNA expression E_1)
5′.         then MakeMinimal (E_1);
                   // we proceed with the new (minimal) version of E_1
6′.              if (E_1 is an ↕-expression)
7′.              then substitute E by E_1;                          (D_Min.1)
8′.              else   // E_1 is an ↑-expression or a ↓-expression
9′.                   substitute E by the result
                        of procedure Make↕ExprMinimal;              (D_Min.1)
10′.             fi
11′.        fi

12′.   else   // E is an ↑-expression or a ↓-expression;
              // without loss of generality, assume it is
              // an ↑-expression ⟨↑ ε_1 ... ε_n⟩ for some n ≥ 1
              // and 𝒩-words and DNA expressions ε_1, ..., ε_n
13′.        for (i = 1 to n)
14′.        do   if (ε_i is a DNA expression E_i)
15′.             then MakeMinimal (E_i);

                        // we proceed with the new (minimal) version of E_i
16′.                 if (E_i is a ↓-expression which is not alternating)
17′.                 then substitute E_i in E by the result
                            of procedure Denickify;                 (D_Min.4)
18′.                 fi

19′.                 if (E_i is a ↓-expression for which the first argument
                         or the last argument is an ↑-argument)
20′.                 then substitute E_i in E by the result
                            of procedure RotateToMinimal;           (D_Min.5)
21′.                 fi

22′.                 if (E_i is an ↑-expression)
23′.                 then substitute E_i in E by its arguments;     (D_Min.2)
24′.                 fi
25′.             fi
26′.        od

27′.        if (E has only one argument ε_1)
28′.        then if (ε_1 is a DNA expression E_1)
29′.             then substitute E by E_1;                          (D_Min.3)
30′.             fi
31′.        else   // E has at least two arguments
32′.             if (E is alternating and both its first argument
                     and its last argument are ↓-arguments)
33′.             then substitute E by the result
                        of procedure RotateToMinimal;               (D_Min.6)
34′.             fi
35′.        fi
36′.   fi
37′. }
```

Figure 9.15: More detailed pseudo-code of the recursive function `MakeMinimal` (cf. Figure 9.1).

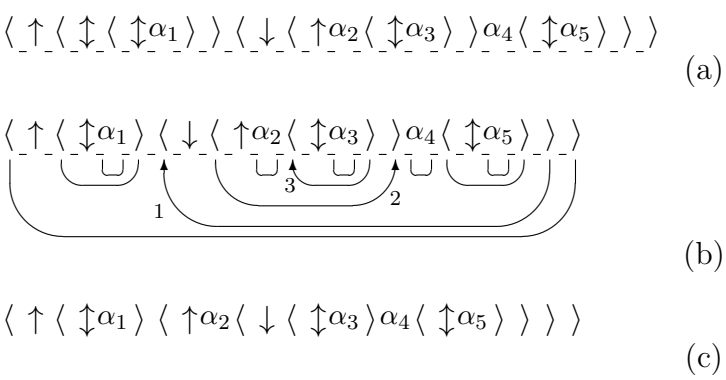


Figure 9.16: First two features of the data structure used in the implementation of the algorithm for minimality. (a) An example DNA expression, where the letters are stored in a doubly-linked list (indicated by the dashes). (b) The result after substituting the $\updownarrow$-subexpression $\langle\updownarrow\langle\updownarrow\alpha_1\rangle\rangle$ by $\langle\updownarrow\alpha_1\rangle$. Corresponding brackets are connected, and the first letter and the last letter of each maximal $\mathcal{N}$-word occurrence are connected. Note that each of the maximal $\mathcal{N}$-word occurrences $\alpha_1,\ldots,\alpha_5$ may consist of many more than one $\mathcal{N}$-letter. We can use connections 1, 2 and 3 as indicated to step through the DNA expression efficiently, for the substitution of the $\downarrow$-subexpression $\langle\downarrow\langle\uparrow\alpha_2\langle\updownarrow\alpha_3\rangle\rangle\alpha_4\langle\updownarrow\alpha_5\rangle\rangle$ by $\langle\uparrow\alpha_2\langle\downarrow\langle\updownarrow\alpha_3\rangle\alpha_4\langle\updownarrow\alpha_5\rangle\rangle\rangle$. (c) The result after this substitution.

substituting an $\updownarrow$-expression E by its (minimal) $\updownarrow$-argument E_1 corresponds to removing three redundant letters: an occurrence of $\updownarrow$, and the corresponding brackets. We have depicted this in Figure 9.16(a) and (b) for the DNA subexpression $\langle\updownarrow\langle\updownarrow\alpha_1\rangle\rangle$ of an example DNA expression.

As another example, in line 20′ of the function, substituting a $\downarrow$-argument

$$E_i = \langle\downarrow\langle\uparrow\varepsilon_{1,1}\ldots\varepsilon_{1,m_1-1}\varepsilon_{1,m_1}\rangle\,\varepsilon_2\ldots\varepsilon_n\rangle \tag{9.24}$$

for some $m_1, n \geq 1$ and $\mathcal{N}$-words and DNA expressions $\varepsilon_{1,1},\ldots,\varepsilon_{1,m_1}$ and $\varepsilon_2,\ldots,\varepsilon_n$ by an equivalent $\uparrow$-argument

$$E_i' = \langle\uparrow\varepsilon_{1,1}\ldots\varepsilon_{1,m_1-1}\,\langle\downarrow\varepsilon_{1,m_1}\varepsilon_2\ldots\varepsilon_n\rangle\rangle \tag{9.25}$$

(the result of procedure `RotateToMinimal`) corresponds to moving the operator $\downarrow$, an opening bracket and a closing bracket to new positions. Moving a letter to a new position means that we remove it from its old position and insert it at its new position.

It is important that we can easily *approach* the positions in the DNA expression where operations like removals and insertions must be performed. In particular, it is useful if we can step directly from, for example, the first letter to the last letter of an argument, and vice versa. This is the second feature of the data structure: we connect each opening bracket to the corresponding closing bracket, and for each $\mathcal{N}$-word-argument of an operator, we connect the first letter to the last letter. Moreover, if we allow $\mathcal{N}$-word-arguments of an operator that are not maximal $\mathcal{N}$-word occurrences, then we also connect the first letter of each maximal $\mathcal{N}$-word occurrence to the last letter. We establish such connections both from left to right and from right to left.

For example, with these connections it is easy to rewrite the $\downarrow$-expression E_i from (9.24) into the $\uparrow$-expression E_i' from (9.25). Let us use ε_1 to denote the first argument $\langle\uparrow\varepsilon_{1,1}\ldots\varepsilon_{1,m_1-1}\varepsilon_{1,m_1}\rangle$ of E_i. We can step directly from the end of E_i (where a closing

bracket must be inserted), via the beginning of E_i (where the operator $\downarrow$ and an opening bracket must be removed), and the end of ε_1 (where a closing bracket must be removed), to the beginning of ε_{1,m_1} (where an opening bracket and an operator $\downarrow$ must be inserted). This way, the entire substitution can be performed in constant time, independent of the length of E_i or the length of ε_1. In Figure 9.16(b) and (c), we carry out this substitution for the DNA subexpression $E_i = \langle\downarrow \langle\uparrow \alpha_2 \langle\updownarrow \alpha_3\rangle\rangle \alpha_4 \langle\updownarrow \alpha_5\rangle\rangle$ of our example DNA expression.

We can also use the connections to travel efficiently along all arguments of a given operator. In two steps we can move from the beginning of an argument, via the end of that argument, to the beginning of the next argument. This requires constant time, independent of the length of the argument.

All connections can be initialized in linear time. For any (basic) operation applied to E, the connections can be updated in constant time.[4] Hence, the overhead for maintaining the connections is linear in the time required for the function `MakeMinimal` itself.

The connections enable us to perform most instructions in the function in constant time. There are, however, two places in the function, where we may spend more than constant time. These places are line $9'$, where we apply procedure `Make↕ExprMinimal`, and line $17'$, where we apply procedure `Denickify`. Moreover, the test if E_i is not alternating in line $16'$ and the test if E is alternating in line $32'$ may be time consuming. We may have to examine all arguments of a DNA expression, before we can decide whether or not it is alternating.

If the data structure for the DNA expression E does not have more features than we have described so far, then the function `MakeMinimal` requires quadratic time for specific instances of E. We illustrate this by two examples, one for line $9'$ and one for line $17'$ of the function.

The DNA expressions we consider in these examples are not new. In Section 4.4, we used them to illustrate that a straightforward implementation of the function `ComputeSem` would require quadratic time. The careful reader will observe, that the underlying causes of the quadratic complexity in the following two examples are similar to those in Section 4.4. The main difference is, that the analysis in the examples in Section 4.4 focused at the semantics (the formal DNA molecules denoted), whereas below we stick to the syntax (the DNA expressions themselves).

In procedure `Make↕ExprMinimal`, we first substitute all $\downarrow$-arguments of the 'working DNA expression' $\widehat{E}_1$, and then substitute all $\mathcal{N}$-word-arguments, see the pseudo-code on page 256. As it is, in order to find all $\downarrow$-arguments (or $\mathcal{N}$-word-arguments) of a given DNA expression, we have to examine all arguments, and check if they are $\downarrow$-arguments ($\mathcal{N}$-word-arguments, respectively).

Example 9.29 (cf. Example 4.14) Let α be an arbitrary $\mathcal{N}$-word, and let

$$\begin{aligned} E_1 &= \langle\updownarrow \alpha\alpha\rangle \\ E_{2p} &= \langle\uparrow E_{2p-1} \langle\updownarrow \alpha\rangle \alpha\rangle && (p \geq 1) \\ E_{2p+1} &= \langle\updownarrow E_{2p}\rangle && (p \geq 1). \end{aligned}$$

Hence,

$$E_1 \;=\; \langle\updownarrow \alpha\alpha\rangle$$

[4] The substitution in line M↕M.5 of procedure `Make↕ExprMinimal` must be considered a composite operation. In the proof of Lemma 9.34, we explain how it can be implemented.

$$\begin{aligned} E_2 &= \langle\uparrow \langle\updownarrow \alpha\alpha\rangle \langle\updownarrow \alpha\rangle \alpha\rangle \\ E_3 &= \langle\updownarrow \langle\uparrow \langle\updownarrow \alpha\alpha\rangle \langle\updownarrow \alpha\rangle \alpha\rangle\rangle \\ E_4 &= \langle\uparrow \langle\updownarrow \langle\uparrow \langle\updownarrow \alpha\alpha\rangle \langle\updownarrow \alpha\rangle \alpha\rangle\rangle \langle\updownarrow \alpha\rangle \alpha\rangle \\ &\ldots \end{aligned}$$

It is easy to prove by induction on p, that for any $p \geq 1$,

- both E_{2p} and E_{2p+1} are DNA expressions,
- $$\begin{aligned} \mathcal{S}(E_{2p}) &= \underbrace{\binom{\alpha\alpha}{c(\alpha\alpha)}_{\triangle} \cdots \binom{\alpha\alpha}{c(\alpha\alpha)}_{\triangle}}_{p \text{ times}} \binom{\alpha}{c(\alpha)} \binom{\alpha}{-} \\ \mathcal{S}(E_{2p+1}) &= \underbrace{\binom{\alpha\alpha}{c(\alpha\alpha)}_{\triangle} \cdots \binom{\alpha\alpha}{c(\alpha\alpha)}_{\triangle}}_{p \text{ times}} \binom{\alpha\alpha}{c(\alpha\alpha)} \end{aligned}$$
- $|E_{2p}| = 3 \cdot 3p + (2p+2) \cdot |\alpha|$ and $|E_{2p+1}| = 3 \cdot (3p+1) + (2p+2) \cdot |\alpha|$.

In particular, the lengths of E_{2p} and E_{2p+1} are linear in p.

Moreover, by Summary 8.16(5), Theorem 7.46 and Theorem 7.42, for $p \geq 1$, the only minimal DNA expression denoting $\mathcal{S}(E_{2p})$, i.e., the only minimal DNA expression that is equivalent to E_{2p} is

$$E'_{2p} = \left\langle \uparrow \underbrace{\langle\updownarrow \alpha\alpha\rangle \ldots \langle\updownarrow \alpha\alpha\rangle}_{p \text{ times}} \langle\updownarrow \alpha\rangle \alpha \right\rangle .$$

By Lemma 8.18(2), the only minimal DNA expression denoting $\mathcal{S}(E_{2p+1})$ is

$$E'_{2p+1} = \left\langle \uparrow \underbrace{\langle\updownarrow \alpha\alpha\rangle \ldots \langle\updownarrow \alpha\alpha\rangle}_{p \text{ times}} \langle\updownarrow \alpha\alpha\rangle \right\rangle .$$

Now, let $p \geq 1$ and let us apply the function `MakeMinimal` to the $\updownarrow$-expression E_{2p+1}, with argument E_{2p}. When we call the function recursively for E_{2p}, this argument is rewritten into E'_{2p}, as that is the only minimal DNA expression that is equivalent to E_{2p}. The $\uparrow$-expression E'_{2p} has $p+2$ arguments. In procedure `Make`$\updownarrow$`ExprMinimal`, we need time that is linear in p to examine them all, to see if they are $\downarrow$-arguments or $\mathcal{N}$-word-arguments.

Likewise, at a higher level of the recursion, we have had to examine the $p+1, p, p-1, \ldots, 3$ arguments of $E'_{2(p-1)}, E'_{2(p-2)}, E'_{2(p-3)}, \ldots, E'_2$, respectively. Altogether, this takes time that is quadratic in p, and thus in the length of E_{2p+1}. ■

In every iteration of the while-loop in procedure `Denickify`, we select two consecutive expression-arguments of the 'working DNA expression' $\widehat{E}_i$, and merge them into a single new argument, see the pseudo-code on page 263. We have not specified how to select these consecutive expression-arguments. We have not even specified how to find them. As it is, we must examine all pairs of consecutive arguments of $\widehat{E}_i$ to see if both of them are expression-arguments. Without further care, this may lead to a quadratic number of steps, as we see in the next example.

Example 9.30 (cf. Example 4.15) Let α be an arbitrary $\mathcal{N}$-word, and let

$$\begin{array}{ll} E_1 = \langle\updownarrow \alpha\alpha\rangle & \\ E_{2p} = \langle\uparrow E_{2p-1}\ \alpha\ \langle\updownarrow \alpha\rangle\ \langle\updownarrow \alpha\rangle\rangle & (p \geq 1) \\ E_{2p+1} = \langle\downarrow E_{2p}\rangle & (p \geq 1). \end{array}$$

Hence,

$$\begin{array}{rcl} E_1 & = & \langle\updownarrow \alpha\alpha\rangle \\ E_2 & = & \langle\uparrow \langle\updownarrow \alpha\alpha\rangle\ \alpha\ \langle\updownarrow \alpha\rangle\ \langle\updownarrow \alpha\rangle\rangle \\ E_3 & = & \langle\downarrow \langle\uparrow \langle\updownarrow \alpha\alpha\rangle\ \alpha\ \langle\updownarrow \alpha\rangle\ \langle\updownarrow \alpha\rangle\rangle\rangle \\ E_4 & = & \langle\uparrow \langle\downarrow \langle\uparrow \langle\updownarrow \alpha\alpha\rangle\ \alpha\ \langle\updownarrow \alpha\rangle\ \langle\updownarrow \alpha\rangle\rangle\rangle\ \alpha\ \langle\updownarrow \alpha\rangle\ \langle\updownarrow \alpha\rangle\rangle \\ & \ldots & \end{array}$$

It is easy to prove by induction on p, that for any $p \geq 1$,

- both E_{2p} and E_{2p+1} are DNA expressions,
- $$\begin{array}{rcl} \mathcal{S}(E_{2p}) & = & \underbrace{\binom{\alpha\alpha}{c(\alpha\alpha)}\binom{\alpha}{-}\cdots\binom{\alpha\alpha}{c(\alpha\alpha)}\binom{\alpha}{-}}_{p \text{ times}}\binom{\alpha}{c(\alpha)}_{\triangle}\binom{\alpha}{c(\alpha)} \\ \mathcal{S}(E_{2p+1}) & = & \underbrace{\binom{\alpha\alpha}{c(\alpha\alpha)}\binom{\alpha}{-}\cdots\binom{\alpha\alpha}{c(\alpha\alpha)}\binom{\alpha}{-}}_{p \text{ times}}\binom{\alpha\alpha}{c(\alpha\alpha)} \end{array}$$
- $|E_{2p}| = 3 \cdot 4p + (3p+2) \cdot |\alpha|$ and $|E_{2p+1}| = 3 \cdot (4p+1) + (3p+2) \cdot |\alpha|$.

In particular, the lengths of E_{2p} and E_{2p+1} are linear in p.

Moreover, by Summary 8.16(5), Theorem 7.46 and Theorem 7.42, for $p \geq 1$, the only minimal DNA expression denoting $\mathcal{S}(E_{2p})$, i.e., the only minimal DNA expression that is equivalent to E_{2p} is

$$E'_{2p} = \left\langle\uparrow \underbrace{\langle\updownarrow \alpha\alpha\rangle\ \alpha \ldots \langle\updownarrow \alpha\alpha\rangle\ \alpha}_{p \text{ times}}\ \langle\updownarrow \alpha\rangle\ \langle\updownarrow \alpha\rangle\right\rangle.$$

By Lemma 8.18(1), the only minimal DNA expression denoting $\mathcal{S}(E_{2p+1})$ is

$$E'_{2p+1} = \left\langle\uparrow \underbrace{\langle\updownarrow \alpha\alpha\rangle\ \alpha \ldots \langle\updownarrow \alpha\alpha\rangle\ \alpha}_{p \text{ times}}\ \langle\updownarrow \alpha\alpha\rangle\right\rangle.$$

Now, let $p \geq 1$ and let us apply the function `MakeMinimal` to the $\downarrow$-expression E_{2p+1}, with argument E_{2p}. When we call the function recursively for E_{2p}, this argument is rewritten into E'_{2p}, as that is the only minimal DNA expression that is equivalent to E_{2p}. The $\uparrow$-expression E'_{2p} has $2p+2$ arguments. In procedure `Denickify`, we need time that is linear in p to examine them all, to see if there are consecutive expression-arguments.

Likewise, at a higher level of the recursion, we have had to examine the $2p, 2p-2, 2p-4, \ldots, 4$ arguments of $E'_{2(p-1)}, E'_{2(p-2)}, E'_{2(p-3)}, \ldots, E'_2$, respectively. Altogether, this takes time that is quadratic in p, and thus in the length of E_{2p+1}. ■

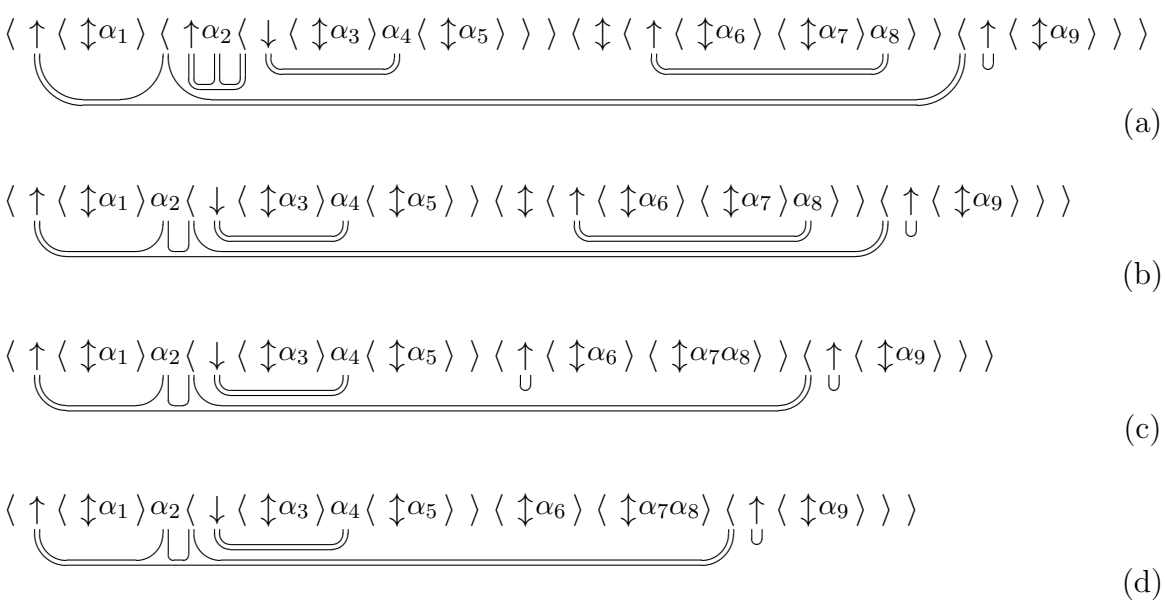


Figure 9.17: Third feature of the data structure used in the implementation of the algorithm for minimality. Each occurrence of $\uparrow$ or $\downarrow$ has a circular, doubly-linked list of its non-$\updownarrow$-arguments. (a) The lists for an example DNA expression. Note that the list is empty for the last occurrence of $\uparrow$. (b) The result after substituting the $\uparrow$-subexpression $\langle\uparrow \alpha_2 \langle\downarrow \langle\updownarrow \alpha_3\rangle \alpha_4 \langle\updownarrow \alpha_5\rangle\rangle\rangle$ by its arguments. (c) The result after using procedure `Make↕ExprMinimal` to substitute the $\updownarrow$-argument $\langle\updownarrow \langle\uparrow \langle\updownarrow \alpha_6\rangle \langle\updownarrow \alpha_7\rangle \alpha_8\rangle\rangle$ by $\langle\uparrow \langle\updownarrow \alpha_6\rangle \langle\updownarrow \alpha_7\alpha_8\rangle\rangle$. As explained in the text, we do not need to insert the resulting $\uparrow$-argument into the list of non-$\updownarrow$-arguments of the outermost operator $\uparrow$. (d) The result after substituting the $\uparrow$-subexpression $\langle\uparrow \langle\updownarrow \alpha_6\rangle \langle\updownarrow \alpha_7\alpha_8\rangle\rangle$ by its arguments.

In order to avoid the quadratic time consumption of the algorithm due to the execution of procedures `Make↕ExprMinimal` and `Denickify`, we add two more features to our data structure, one for each procedure. We first focus on a feature that is useful for `Make↕ExprMinimal`: for each occurrence of $\uparrow$ or $\downarrow$ in E we maintain a circular, doubly-linked list of its non-$\updownarrow$-arguments. In fact, the list contains (the positions of) the first letters of the non-$\updownarrow$-arguments. This is the third feature of our data structure. In Figure 9.17(a), we show the lists for all occurrences of $\uparrow$ and $\downarrow$ in an example DNA expression.

The time required to initialize the lists for all occurrences of $\uparrow$ and $\downarrow$ in E is linear in $|E|$. One can verify that for almost every operation performed on E in the course of the algorithm, the lists can easily be updated in constant time. For example, in line 23′ of `MakeMinimal`, we substitute an $\uparrow$-argument E_i of an $\uparrow$-expression by its own arguments. In principle, we can simply substitute E_i (which is a non-$\updownarrow$-argument itself) in the list of non-$\updownarrow$-arguments of its parent operator by the list of its own non-$\updownarrow$-arguments (see Figure 9.17(b)).[5] As both lists are doubly-linked lists, we can do this in constant time.

The only exception is line 9′ of the recursive function. There, we substitute an $\updownarrow$-

[5]There is a little subtlety one has to consider when implementing this: if E_i is preceded by an $\mathcal{N}$-word-argument and its own first argument $\varepsilon_{i,1}$ is also an $\mathcal{N}$-word-argument, then these $\mathcal{N}$-word-arguments merge into one maximal $\mathcal{N}$-word occurrence. Hence, instead of two $\mathcal{N}$-word-arguments in the new list of non-$\updownarrow$-arguments, one may have a single maximal $\mathcal{N}$-word occurrence. In the example from Figure 9.17(a), this would be the case if the first argument of the DNA expression were α_1 instead of $\langle\updownarrow \alpha_1\rangle$. We may have an analogous situation with the last argument of E_i and the argument succeeding E_i.

expression $E = \langle \updownarrow E_1 \rangle$, where E_1 is a minimal $\uparrow$-expression or $\downarrow$-expression, by the result of (precisely) procedure `Make↕ExprMinimal`. If, for example, E_1 is an $\uparrow$-expression, then the result may also be an $\uparrow$-expression $E' = \langle \uparrow \langle \updownarrow \alpha_1 \rangle \dots \langle \updownarrow \alpha_m \rangle \rangle$ for some $m \geq 2$ and $\mathcal{N}$-words $\alpha_1, \dots, \alpha_m$. If E is not the entire DNA expression and its parent operator is $\uparrow$ or $\downarrow$, then E' should be inserted into the list of non-$\updownarrow$-arguments of the parent operator.

There are ways to do this in constant time, but we may as well omit it. Suppose that we omit it, like we do in Figure 9.17(c). We examine what the next step of the algorithm is, after substituting E by E'.

`MakeMinimal` had been called recursively for E (as expression-argument of a larger DNA expression) in line 15′. If the parent operator of E is $\downarrow$, then according to lines 16′–18′, the next step of the algorithm is to substitute E' (which is not alternating) by the result of procedure `Denickify`, which is the nick free $\updownarrow$-expression $\langle \updownarrow \alpha_1 \dots \alpha_m \rangle$. If, on the other hand, the parent operator is $\uparrow$, then according to lines 22′–24′, the next step of the algorithm is to substitute E' by its $\updownarrow$-arguments $\langle \updownarrow \alpha_1 \rangle, \dots, \langle \updownarrow \alpha_m \rangle$, as in Figure 9.17(d). In both cases, it does not hurt that E' was not in the list of non-$\updownarrow$-arguments of its parent operator. It is substituted by only $\updownarrow$-arguments, after all.

We conclude that for every operation performed on E, we can (sufficiently) update the lists of non-$\updownarrow$-arguments in constant time. Hence, if the total number of operations performed by the algorithm is in $\mathcal{O}(|E|)$, then so is the time spent on updating these lists.

In procedure `Make↕ExprMinimal`, both the $\downarrow$-arguments and the $\mathcal{N}$-word-arguments are substituted 'in some order'. Hence, the order of the non-$\updownarrow$-arguments in the lists is not important. It is, however, natural to have the arguments in the order of their occurrence in the DNA expression. This property can easily be achieved. In fact, it is very natural to implement the initialization and updatings of the lists in such a way, that the lists always have this property.

Example 9.31 In Example 9.29, we defined a series of DNA expressions $E_1, E_2, E_3, \dots$, for which the function `MakeMinimal` spent at least quadratic time in procedure `Make↕ExprMinimal`. This complexity was based on the assumption that for $p \geq 1$, all $p+2$ arguments of the $\uparrow$-expression

$$E'_{2p} = \left\langle \uparrow \underbrace{\langle \updownarrow \alpha\alpha \rangle \dots \langle \updownarrow \alpha\alpha \rangle}_{p \text{ times}} \langle \updownarrow \alpha \rangle \, \alpha \right\rangle$$

have to be examined to see if they are $\downarrow$-arguments or $\mathcal{N}$-word-arguments. This requires time that is linear in p.

Now that we have a list of non-$\updownarrow$-arguments for each occurrence of $\uparrow$ or $\downarrow$, we can do better. We can simply traverse the list of the outermost operator $\uparrow$ of E'_{2p}. Because the only element of this list is the last argument α of E'_{2p}, this requires constant time. ■

We prove that the current features of the data structure are indeed sufficient to execute procedure `Make↕ExprMinimal` efficiently. For this proof and a later proof, we need some additional notation:

Definition 9.32 *Let E be an arbitrary DNA expression.*

- *$n_\alpha(E)$ is the number of maximal $\mathcal{N}$-word occurrences in E.*
- *$n_{\alpha\updownarrow}(E)$ is the number of maximal $\mathcal{N}$-word occurrences in E for which the parent operator is an occurrence of $\updownarrow$.*

- *$n_{\alpha\uparrow\downarrow}(E)$ is the number of maximal $\mathcal{N}$-word occurrences in E for which the parent operator is an occurrence of either $\uparrow$ or $\downarrow$.*
- *$n_{\mathcal{N}\uparrow\downarrow}(E)$ is the number of $\mathcal{N}$-letters occurring in E, for which the parent operator (of the maximal $\mathcal{N}$-word occurrence that the $\mathcal{N}$-letter is part of) is an occurrence of either $\uparrow$ or $\downarrow$.*

Let X be an arbitrary formal DNA molecule.

- *$n_{\uparrow\downarrow}(X)$ is the number of single-stranded components of X.*

Note the difference between $n_{\alpha\uparrow\downarrow}(E)$ and $n_{\mathcal{N}\uparrow\downarrow}(E)$: $n_{\alpha\uparrow\downarrow}(E)$ denotes the *number* of certain maximal $\mathcal{N}$-word occurrences, whereas $n_{\mathcal{N}\uparrow\downarrow}(E)$ denotes their *total length*. Note also that in Definition 6.10, we introduced the notation $n_{\updownarrow}(X)$, for the number of double components of a formal DNA molecule X. The notation $n_{\uparrow\downarrow}(X)$ is the natural variant for single-stranded components. Obviously, for each DNA expression E, $n_{\alpha}(E) = n_{\alpha\updownarrow}(E) + n_{\alpha\uparrow\downarrow}(E)$ and $n_{\alpha\uparrow\downarrow}(E) \leq n_{\mathcal{N}\uparrow\downarrow}(E)$.

Example 9.33 Let E be the DNA expression E_1^* from (9.1) and let $X = \mathcal{S}(E)$ (see Figure 9.2). Then

$$\begin{aligned} n_{\alpha}(E) &= 23, \\ n_{\alpha\updownarrow}(E) &= 13, \\ n_{\alpha\uparrow\downarrow}(E) &= 10, \\ n_{\mathcal{N}\uparrow\downarrow}(E) &= |\alpha_1| + |\alpha_3| + |\alpha_5| + |\alpha_7| + |\alpha_{10}| + |\alpha_{12}| + |\alpha_{16}| + |\alpha_{18}| + |\alpha_{21}| + |\alpha_{23}|, \\ n_{\uparrow\downarrow}(X) &= 6. \end{aligned}$$

■

Lemma 9.34 *Let E_1^* be an arbitrary DNA expression. The total time that the function* `MakeMinimal` *applied to E_1^* spends in procedure* `Make`$\updownarrow$`ExprMinimal` *is in $\mathcal{O}(|E_1^*|)$.*

Proof: Let $E^s = \langle \updownarrow E_1 \rangle$ be an $\updownarrow$-expression whose argument E_1 is a minimal $\uparrow$-expression, and let us apply procedure `Make`$\updownarrow$`ExprMinimal` to E^s. We first analyse the time required for this single application of the procedure. We will use the outcome of this analysis to prove the claim about the total time spent in the procedure during the execution of `MakeMinimal` for E_1^*.

The main part of procedure `Make`$\updownarrow$`ExprMinimal` consists of the two for-loops. The other instructions of the algorithm require constant time. We assume that we have a list containing the non-$\updownarrow$-arguments of E_1.

Clearly, if E_1 does not have any non-$\updownarrow$-argument, then also the for-loops require constant time. In that case, the total time spent in procedure `Make`$\updownarrow$`ExprMinimal` for E^s is constant.

Now, assume that E_1 has at least one non-$\updownarrow$-argument. We prove that the number of non-$\updownarrow$-arguments is at most $n_{\mathcal{N}\uparrow\downarrow}(E^s)$. [6]

[6] Under the natural assumption that each $\mathcal{N}$-word-argument of E_1 is a maximal $\mathcal{N}$-word occurrence, we could easily derive a tighter upper bound on the number of non-$\updownarrow$-arguments, viz $n_{\alpha\uparrow\downarrow}(E^s)$. However, this would not be sufficient as an upper bound for the total time spent in the call of procedure `Make`$\updownarrow$`ExprMinimal` for E^s. As we will see later in the proof, we have to determine the *elementwise* complement of maximal $\mathcal{N}$-word occurrences with parent operator $\downarrow$. We cannot do this in time in $\mathcal{O}(n_{\alpha\uparrow\downarrow}(E^s))$. We come back to this in Section 9.4.

By Corollary 8.2, the arguments of E_1 are $\mathcal{N}$-words $\alpha_{1,i}$, or $\updownarrow$-expressions $\langle\updownarrow \alpha_{1,i}\rangle$ for $\mathcal{N}$-words $\alpha_{1,i}$, or $\downarrow$-expressions. In particular, the non-$\updownarrow$-arguments are $\mathcal{N}$-words $\alpha_{1,i}$ and $\downarrow$-expressions. By Lemma 8.4, the expression-arguments of E_1 are nick free.

For each $\mathcal{N}$-word-argument $\alpha_{1,i}$ of E_1, the parent operator is $\uparrow$. Hence, this argument contributes its length $|\alpha_{1,i}| \geq 1$ to $n_{\mathcal{N}\uparrow\downarrow}(E^s)$. By definition, an $\updownarrow$-argument $\langle\updownarrow \alpha_{1,i}\rangle$ of E_1 does not contribute at all to $n_{\mathcal{N}\uparrow\downarrow}(E^s)$. Finally, each $\downarrow$-argument $E_{1,i}$ of E_1 contributes $n_{\mathcal{N}\uparrow\downarrow}(E_{1,i})$ to $n_{\mathcal{N}\uparrow\downarrow}(E^s)$. As all occurrences of $\mathcal{N}$-letters in E^s are in arguments of E_1, we have

$$n_{\mathcal{N}\uparrow\downarrow}(E^s) = \sum_{\mathcal{N}\text{-words } \alpha_{1,i}} |\alpha_{1,i}| + \sum_{\downarrow\text{-expr. } E_{1,i}} n_{\mathcal{N}\uparrow\downarrow}(E_{1,i}). \tag{9.26}$$

Consider a $\downarrow$-argument $E_{1,i}$ of E_1. $E_{1,i}$ is in particular a proper DNA subexpression of E_1. Hence, by Lemma 8.27(4), it has at least one $\mathcal{N}$-word-argument α. Because the parent operator of this $\mathcal{N}$-word-argument is $\downarrow$, $n_{\mathcal{N}\uparrow\downarrow}(E_{1,i}) \geq |\alpha| \geq 1$. This implies that $n_{\mathcal{N}\uparrow\downarrow}(E^s)$ is an upper bound for the number of non-$\updownarrow$-arguments of E_1.

Because there is a list of the non-$\updownarrow$-arguments of E_1, the time needed to just iterate along all $\downarrow$-arguments and $\mathcal{N}$-word-arguments is linear in the number of non-$\updownarrow$-arguments, which thus is in $\mathcal{O}(n_{\mathcal{N}\uparrow\downarrow}(E^s))$.

We now examine the operations performed for the non-$\updownarrow$-arguments.

- Let $E_{1,i}$ be a $\downarrow$-argument of E_1. In line M$\updownarrow$M.5, we substitute $E_{1,i}$ by $\langle\updownarrow \alpha_{E_{1,i}}\rangle$. For this, we first have to determine $\alpha_{E_{1,i}}$. We prove that we can do this in time that is in $\mathcal{O}(n_{\mathcal{N}\uparrow\downarrow}(E_{1,i}))$.

  We can determine $\alpha_{E_{1,i}}$ by traversing $E_{1,i}$ from left to right, skipping the operators and the brackets, and linking the maximal $\mathcal{N}$-word occurrences we encounter. Those maximal $\mathcal{N}$-word occurrences that have (an occurrence of) $\downarrow$ as their parent operator, must be complemented first, before they are added to $\alpha_{E_{1,i}}$. For the moment, however, we ignore these complementations.

  Each operator occurring in $E_{1,i}$ corresponds to a DNA subexpression of $E_{1,i}$. This DNA subexpression is a proper DNA subexpression of E_1. Hence, by Lemma 8.27(4) the total number of operators occurring in $E_{1,i}$ is limited by the number of maximal $\mathcal{N}$-word occurrences in $E_{1,i}$, which we denote by $n_\alpha(E_{1,i})$.

  As for the maximal $\mathcal{N}$-word occurrences themselves, recall that we can step directly from the beginning of a maximal $\mathcal{N}$-word occurrence to the end. Therefore, the upper bound on the number of operators implies that the total time required for traversing $E_{1,i}$ from left to right, skipping the operators and the brackets, and linking maximal $\mathcal{N}$-word occurrences is linear in $n_\alpha(E_{1,i})$.

  We now relate $n_\alpha(E_{1,i})$ (the total number of maximal $\mathcal{N}$-word occurrences in $E_{1,i}$) to $n_{\alpha\uparrow\downarrow}(E_{1,i})$ (the number of maximal $\mathcal{N}$-word occurrences with parent operator $\uparrow$ or $\downarrow$). Consider an arbitrary minimal, nick free $\uparrow$-expression or $\downarrow$-expression E, and let $X = \mathcal{S}(E)$. By Summary 8.16, X contains at least one single-stranded component and E is constructed according to Theorem 7.24. It is not difficult to prove by induction on the lower of $B_\uparrow(X)$ and $B_\downarrow(X)$, that there is a 1–1 correspondence between components of X and maximal $\mathcal{N}$-word occurrences in E. Each upper component $\binom{\alpha_i}{-}$ (or lower component $\binom{-}{\alpha_i}$ or double component $\binom{\alpha_i}{c(\alpha_i)}$) for an $\mathcal{N}$-word α_i corresponds to a maximal $\mathcal{N}$-word occurrence α_i whose parent operator is $\uparrow$ (or $\downarrow$ or $\updownarrow$, respectively).

This holds in particular for the minimal, nick free $\downarrow$-expression $E_{1,i}$. Let $X_{1,i} = \mathcal{S}(E_{1,i})$. Then

$$\begin{aligned} n_{\alpha\uparrow\downarrow}(E_{1,i}) &= n_{\uparrow\downarrow}(X_{1,i}) \geq 1, \\ n_{\alpha\updownarrow}(E_{1,i}) &= n_{\updownarrow}(X_{1,i}). \end{aligned}$$

Because, by Corollary 3.8, double components and single-stranded components alternate in $X_{1,i}$, we have

$$n_{\updownarrow}(X_{1,i}) \leq n_{\uparrow\downarrow}(X_{1,i}) + 1 \leq 2 \cdot n_{\uparrow\downarrow}(X_{1,i}).$$

Combining the above equations, we find that

$$\begin{aligned} n_{\alpha}(E_{1,i}) &= n_{\alpha\uparrow\downarrow}(E_{1,i}) + n_{\alpha\updownarrow}(E_{1,i}) \\ &= n_{\alpha\uparrow\downarrow}(E_{1,i}) + n_{\updownarrow}(X_{1,i}) \\ &\leq n_{\alpha\uparrow\downarrow}(E_{1,i}) + 2 \cdot n_{\uparrow\downarrow}(X_{1,i}) \\ &= 3 \cdot n_{\alpha\uparrow\downarrow}(E_{1,i}). \end{aligned}$$

In words: the total number of maximal $\mathcal{N}$-word occurrences in $E_{1,i}$ is at most 3 times the number of maximal $\mathcal{N}$-word occurrences in $E_{1,i}$ with parent operator $\uparrow$ or $\downarrow$. Or in other words: at least one third of the maximal $\mathcal{N}$-word occurrences in $E_{1,i}$ has parent operator $\uparrow$ or $\downarrow$. This implies that the time required for traversing $E_{1,i}$ from left to right, skipping the operators and the brackets, and linking maximal $\mathcal{N}$-word occurrences is linear in $n_{\alpha\uparrow\downarrow}(E_{1,i})$. Because $n_{\alpha\uparrow\downarrow}(E_{1,i}) \leq n_{\mathcal{N}\uparrow\downarrow}(E_{1,i})$, this time is in $\mathcal{O}(n_{\mathcal{N}\uparrow\downarrow}(E_{1,i}))$.

We finally examine the time required for complementing the maximal $\mathcal{N}$-word occurrences in $E_{1,i}$ that have $\downarrow$ as their parent operator. Clearly, these maximal $\mathcal{N}$-word occurrences contain at most $n_{\mathcal{N}\uparrow\downarrow}(E_{1,i})$ $\mathcal{N}$-letters. Moreover, it does not really cost time to *find* these maximal $\mathcal{N}$-word occurrences: we encounter all maximal $\mathcal{N}$-word occurrences anyway while traversing $E_{1,i}$ from left to right, and it is not difficult to keep track of their parent operators (a stack of operators that are currently 'active' is sufficient). This implies that the additional time needed to determine the elementwise complements of the maximal $\mathcal{N}$-word occurrences with parent operator $\downarrow$ is in $\mathcal{O}(n_{\mathcal{N}\uparrow\downarrow}(E_{1,i}))$.

We conclude that the time required for determining $\alpha_{E_{1,i}}$ is in $\mathcal{O}(n_{\mathcal{N}\uparrow\downarrow}(E_{1,i}))$. Having determined $\alpha_{E_{1,i}}$, we can substitute $E_{1,i}$ by $\langle\updownarrow \alpha_{E_{1,i}}\rangle$ in constant time. Hence, the total time requirement for line M$\updownarrow$M.5 is also in $\mathcal{O}(n_{\mathcal{N}\uparrow\downarrow}(E_{1,i}))$.

- Let $\alpha_{1,i}$ be an $\mathcal{N}$-word-argument of E_1. In lines M$\updownarrow$M.8–M$\updownarrow$M.17 of procedure `Make`$\updownarrow$`ExprMinimal`, we substitute $\alpha_{1,i}$ (possibly together with a preceding and a succeeding $\updownarrow$-argument) by a new $\updownarrow$-argument. We can do this in constant time. Hence, we can certainly do this in time that is in $\mathcal{O}(|\alpha_{1,i}|)$.

  Recall that if E_1 is not an $\uparrow$-expression, but a $\downarrow$-expression, then we must complement the $\mathcal{N}$-word-argument $\alpha_{1,i}$, before we make it (part of) the argument of $\updownarrow$. Determining the elementwise complement of $\alpha_{i,1}$ requires time that is linear in $|\alpha_{1,i}|$. Also in this case, the total time required for the substitution of $\alpha_{i,1}$ is in $\mathcal{O}(|\alpha_{1,i}|)$.

When we combine the time requirements for the operations on the two types of arguments, we find that the total time required for the operations on all non-$\updownarrow$-arguments is in

$$\sum_{\downarrow\text{-expr. } E_{1,i}} \mathcal{O}(n_{\mathcal{N}\uparrow\downarrow}(E_{1,i})) + \sum_{\mathcal{N}\text{-words } \alpha_{1,i}} \mathcal{O}(|\alpha_{1,i}|).$$

By (9.26), this is in $\mathcal{O}(n_{\mathcal{N}\uparrow\downarrow}(E^s))$. Hence, the total time spent in procedure `Make↕ExprMinimal` for the case that E_1 has at least one non-$\updownarrow$-argument is in $\mathcal{O}(n_{\mathcal{N}\uparrow\downarrow}(E^s))$.

For the general case (E_1 with or without non-$\updownarrow$-arguments), the time spent in the procedure is in $\mathcal{O}(1 + n_{\mathcal{N}\uparrow\downarrow}(E^s))$. The resulting DNA expression $E^{s\prime}$ is equal either to $\langle\updownarrow \alpha_{1,1}\rangle$ for an $\mathcal{N}$-word $\alpha_{1,1}$, or to $\langle\uparrow \langle\updownarrow \alpha_{1,1}\rangle \ldots \langle\updownarrow \alpha_{1,m}\rangle\rangle$ for some $m \geq 2$ and $\mathcal{N}$-words $\alpha_{1,1}, \ldots, \alpha_{1,m}$. In both cases, $n_{\mathcal{N}\uparrow\downarrow}(E^{s\prime}) = 0$. Now, let E be the overall 'working DNA expression' of the algorithm. When we substitute E^s in E by $E^{s\prime}$, $n_{\mathcal{N}\uparrow\downarrow}(E)$ decreases by an amount of $n_{\mathcal{N}\uparrow\downarrow}(E^s)$.

When we use function `MakeMinimal` to determine an equivalent, minimal DNA expression for the original DNA expression E_1^*, there may be several $\updownarrow$-subexpressions E^s for which we apply procedure `Make↕ExprMinimal`. Let $E_1^s, \ldots, E_r^s$ for some $r \geq 0$ be all these $\updownarrow$-subexpressions. For $h = 1, \ldots, r$, we spend $\mathcal{O}(1 + n_{\mathcal{N}\uparrow\downarrow}(E_h^s))$ time in procedure `Make↕ExprMinimal`, and as a result $n_{\mathcal{N}\uparrow\downarrow}(E)$ decreases by $n_{\mathcal{N}\uparrow\downarrow}(E_h^s)$. The total time spent in the procedure is in

$$\mathcal{O}(1 + n_{\mathcal{N}\uparrow\downarrow}(E_1^s)) + \cdots + \mathcal{O}(1 + n_{\mathcal{N}\uparrow\downarrow}(E_r^s)),$$

which is in

$$\mathcal{O}(r + n_{\mathcal{N}\uparrow\downarrow}(E_1^s) + \cdots + n_{\mathcal{N}\uparrow\downarrow}(E_r^s)).$$

In the course of the algorithm, we also perform other operations on DNA subexpressions of E. It is easily verified that none of these operations changes the parent operator of any $\mathcal{N}$-word α occurring in E. In particular, none of them increases $n_{\mathcal{N}\uparrow\downarrow}(E)$. Hence, the sum of the decreases of $n_{\mathcal{N}\uparrow\downarrow}(E)$ caused by the application of procedure `Make↕ExprMinimal` to $E_1^s, \ldots, E_r^s$ is bounded by the initial value of $n_{\mathcal{N}\uparrow\downarrow}(E)$:

$$n_{\mathcal{N}\uparrow\downarrow}(E_1^s) + \cdots + n_{\mathcal{N}\uparrow\downarrow}(E_r^s) \leq n_{\mathcal{N}\uparrow\downarrow}(E_1^*).$$

But then the total time spent in procedure `Make↕ExprMinimal` is in $\mathcal{O}(r + n_{\mathcal{N}\uparrow\downarrow}(E_1^*))$.

It follows directly from lines 5′–10′ of `MakeMinimal`, that r is bounded by the number of recursive calls of `MakeMinimal`. It is not hard to prove by induction that this number (including the call for E_1^* itself) equals the number of operators occurring in E_1^*, which is in $\mathcal{O}(|E_1^*|)$. By definition, $n_{\mathcal{N}\uparrow\downarrow}(E_1^*)$ is also in $\mathcal{O}(|E_1^*|)$. We conclude that the total time spent in procedure `Make↕ExprMinimal` while executing function `MakeMinimal` for E_1^* is in $\mathcal{O}(|E_1^*|)$.

This completes the proof of Lemma 9.34. □

By introducing lists of non-$\updownarrow$-arguments, we have managed to spend at most linear time in procedure `Make↕ExprMinimal` of our rewriting algorithm. We now add a fourth feature to our data structure to achieve the same result for procedure `Denickify`. For each occurrence of $\uparrow$ or $\downarrow$ in E, we maintain a circular, doubly-linked list of its consecutive expression-arguments. To be more precise: for each expression-argument $\widehat{\varepsilon}_j$ of the operator, which is preceded by another expression-argument $\widehat{\varepsilon}_{j-1}$, the list contains the

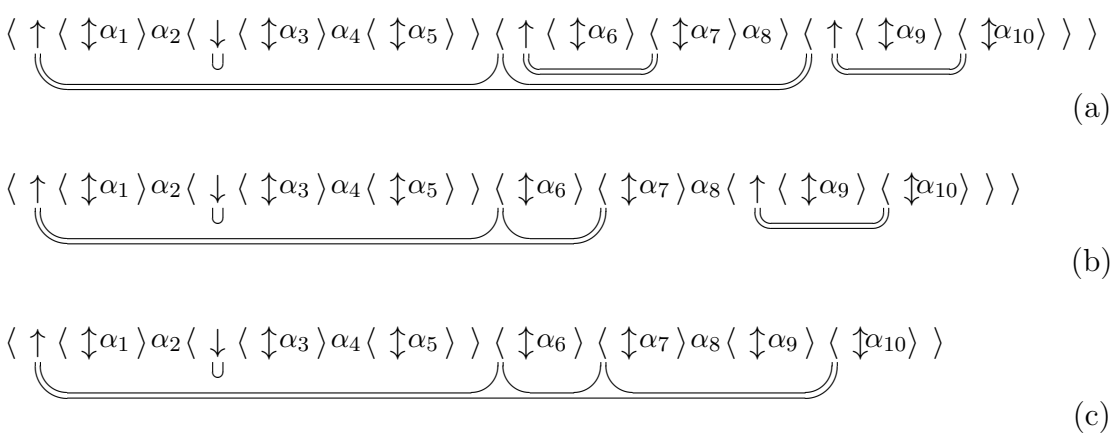


Figure 9.18: Fourth feature of the data structure used in the implementation of the algorithm for minimality. Each occurrence of $\uparrow$ or $\downarrow$ has a circular, doubly-linked list of its consecutive expression-arguments. (a) The lists for an example DNA expression. (b) The result after substituting the $\uparrow$-subexpression $\langle\uparrow \langle\updownarrow \alpha_6\rangle \langle\updownarrow \alpha_7\rangle \alpha_8\rangle$ by its arguments. (c) The result after substituting the $\uparrow$-subexpression $\langle\uparrow \langle\updownarrow \alpha_9\rangle \langle\updownarrow \alpha_{10}\rangle\rangle$ by its arguments.

position of the first letter of $\widehat{\varepsilon}_j$ (which is an opening bracket). In Figure 9.18(a), we show the lists for all occurrences of $\uparrow$ and $\downarrow$ in an example DNA expression.

The time needed to initialize these new lists for all occurrences of $\uparrow$ and $\downarrow$ in a DNA expression E is linear in $|E|$. After any basic operation performed on E, the lists can be updated in constant time.

As an example, we again consider line 23′ of `MakeMinimal`, where we substitute an $\uparrow$-argument E_i of an $\uparrow$-expression by its own arguments. Let $E_i = \langle\uparrow_1 \varepsilon_{i,1} \ldots \varepsilon_{i,n_i}\rangle$ for some $n_i \geq 1$ and $\mathcal{N}$-words and DNA expressions $\varepsilon_{i,1}, \ldots, \varepsilon_{i,n_i}$. In principle, we can just remove E_i from the list of consecutive expression-arguments of its parent operator $\uparrow_0$ (if it is in that list) and insert the list of consecutive expression-arguments of E_i into that list. The borders of E_i, however, require a special treatment.

Assume, for example, that the $\uparrow$-argument E_i is preceded by another expression-argument E_{i-1}. Then (the opening bracket of) E_i is in the list of consecutive expression-arguments of $\uparrow_0$. After the substitution, E_{i-1} is succeeded by the first argument $\varepsilon_{i,1}$ of E_i. If $\varepsilon_{i,1}$ is an $\mathcal{N}$-word, then E_{i-1} is no longer succeeded by an expression-argument. Hence, the parent operator $\uparrow_0$ loses a pair of consecutive expression-arguments. If, on the other hand, $\varepsilon_{i,1}$ is a DNA expression, then it takes over the role of E_i. Whereas $\varepsilon_{i,1}$ was not in the list of consecutive expression-arguments of $\uparrow_1$ (because it was not preceded by any argument in E_i), it must be inserted into the list of $\uparrow_0$. Similar situations may occur if E_i is succeeded by another expression-argument E_{i+1}. This description is illustrated by Figure 9.18(b).

In the above, either E_i or E_{i+1} (or both) used to be in the list of consecutive expression-arguments of $\uparrow_0$. In that case, it is natural and easy to insert the list of consecutive expression-arguments of $\uparrow_1$ into the list of $\uparrow_0$ at the position corresponding to E_i. If, however, E_i is neither preceded, nor succeeded by an expression-argument, then we should first determine this position. There are ways to do this in constant time, but we may as well omit it. In fact, we may insert the list of consecutive expression-arguments of $\uparrow_1$ anywhere into the list of $\uparrow_0$, because in procedure `Denickify`, the order in which we select pairs of consecutive expression-arguments is not specified. By Theorem 9.24(3), the result of the

procedure is completely independent of this order. In Figure 9.18(c), we have arbitrarily inserted the $\updownarrow$-argument $\langle \updownarrow \alpha_{10} \rangle$ (which is preceded by another expression-argument), at the end of the list of consecutive expression-arguments of the outermost operator $\uparrow$.

Although we have to distinguish a number of cases to update the lists of consecutive expression-arguments after substituting the $\uparrow$-argument E_i by its own arguments, we can still do this in constant time.

For the other operations we perform in the course of the algorithm, updating these lists is relatively easy. A number of operations simply consist of substituting one DNA subexpression by another, which does not really affect the occurrence of consecutive expression-arguments. As we observed in the proof of Theorem 9.27, in procedure `RotateToMinimal`, we deal with DNA expressions for which each occurrence of $\uparrow$ or $\downarrow$ is alternating. For those operators, the lists are and remain empty.

We can use the lists of consecutive expression-arguments also to check if a DNA expression E_i or E is alternating, in lines $16'$ and $32'$ of `MakeMinimal`, respectively. This is the case, if and only if the list of consecutive expression-arguments of the outermost operator is empty. We can check this in constant time.

Example 9.35 In Example 9.30, we defined a series of DNA expressions $E_1, E_2, E_3, \ldots$, for which the function `MakeMinimal` spent at least quadratic time in procedure `Denickify`. This complexity was based on the assumption that for $p \geq 1$, all $2p + 2$ arguments of the $\uparrow$-expression

$$E'_{2p} = \left\langle \uparrow \underbrace{\langle \updownarrow \alpha\alpha \rangle \, \alpha \ldots \langle \updownarrow \alpha\alpha \rangle \, \alpha}_{p \text{ times}} \langle \updownarrow \alpha \rangle \, \langle \updownarrow \alpha \rangle \right\rangle .$$

have to be examined to see if there are consecutive expression-arguments. This requires time that is linear in p.

Now that we have a list of consecutive expression-arguments for each occurrence of $\uparrow$ or $\downarrow$, we can do better. We can simply traverse the list of the outermost operator $\uparrow$ of E'_{2p}. Because the only element of this list is the last argument $\langle \updownarrow \alpha \rangle$ of E'_{2p} (which is preceded by another argument $\langle \updownarrow \alpha \rangle$), this requires constant time. ■

We now prove that the lists of consecutive expression-arguments indeed enable us to execute procedure `Denickify` efficiently.

Lemma 9.36 *Let E_1^* be an arbitrary DNA expression. The total time that the function* `MakeMinimal` *applied to E_1^* spends in procedure* `Denickify` *is in $\mathcal{O}(n_\alpha(E_1^*))$, which is in $\mathcal{O}(|E_1^*|)$.*

Proof: Let E_i be a minimal $\downarrow$-expression which is not alternating, and let us apply procedure `Denickify` to E_i. We first analyse the time required for this single application of the procedure. We will use the outcome of this analysis to prove the claim about the total time spent in the procedure during the execution of `MakeMinimal` for E_1^*.

The main part of procedure `Denickify` is formed by the while-loop. The other instructions of the procedure require constant time. We assume that we have a list containing the consecutive expression-arguments of E_i.

At the beginning of each iteration of the while-loop, we test the condition "$\widehat{E}_i$ is not alternating." This is the case, if and only if the list with the consecutive expression-arguments is not empty. We can test this in constant time.

In each iteration of the while-loop, we select two consecutive expression-arguments $\widehat{\varepsilon}_{j-1}$ and $\widehat{\varepsilon}_j$ of the 'working DNA expression' $\widehat{E}_i$, and substitute $\widehat{\varepsilon}_{j-1}\widehat{\varepsilon}_j$ in $\widehat{E}_i$ by a single expression-argument. Again, because we have a list with the consecutive expression-arguments, we can perform the selection of $\widehat{\varepsilon}_{j-1}$ and $\widehat{\varepsilon}_j$ in constant time. For the substitution, we have to distinguish four different cases. Both the distinction of these cases and the subsequent substitution can be carried out in constant time. Consequently, each iteration of the while-loop requires constant time, and the total time spent in the while-loop is linear in the number of iterations.

Let us use $n_{\text{iter}}(E_i)$ to denote this number of iterations. Then the time spent in procedure `Denickify` for E_i (outside and inside the while-loop) is in $\mathcal{O}(1 + n_{\text{iter}}(E_i))$. As E_i is not alternating, $n_{\text{iter}}(E_i) \geq 1$. Hence, the time spent in the procedure for E_i is in $\mathcal{O}(n_{\text{iter}}(E_i))$. In fact, because $n_{\text{iter}}(E_i)$ also provides a lower bound, the time is *linear* in $n_{\text{iter}}(E_i)$.

In the course of the execution of `MakeMinimal` for E_1^*, procedure `Denickify` may be applied to several different DNA subexpressions E_i. If we use $n_{\text{iter}}(E_1^*)$ to denote the total number of iterations of the while-loop in the procedure during all these applications, then the total time spent in the procedure is linear in $n_{\text{iter}}(E_1^*)$.

It is immediate from the pseudo-code of procedure `Denickify` that the substitution performed in an iteration of the while-loop leads to a decrease of the number of maximal $\mathcal{N}$-word occurrences in $\widehat{E}_i$ by 1. Of course, this corresponds to an equal decrease of the number of maximal $\mathcal{N}$-word occurrences in the overall 'working DNA expression' E, which we denote by $n_\alpha(E)$.

Now, it is easily verified that at no point in the algorithm, $n_\alpha(E)$ increases.[7] Hence, $n_{\text{iter}}(E_1^*) \leq n_\alpha(E_1^*)$. This implies that the total time spent in procedure `Denickify` is in $\mathcal{O}(n_\alpha(E_1^*))$. Obviously, $n_\alpha(E_1^*)$ is in $\mathcal{O}(|E_1^*|)$. □

By now, we know the time requirements of procedures `Make↕ExprMinimal` and `Denickify`. There is one more point we like to make before we determine the total time complexity of the function `MakeMinimal`. It is a point we also made in Section 4.4, when we analysed the recursive function `ComputeSem`.

The (only) parameter of `MakeMinimal` is a DNA expression E. When we (recursively) call the function for an expression-argument E_i of E, we do not have to explicitly copy this expression-argument as a sequence of individual characters into the actual parameter of the call. It is sufficient to make a 'call by reference', e.g., by passing the starting position of E_i (the position of its opening bracket) to the call. This implies that both the time needed to set the actual parameter and the space required to store it are constant for a single call.

Actually, we should have addressed this issue also when we analysed the time requirements of the procedures `Make↕ExprMinimal` and `Denickify`. The fact that we ignored it there, does not mean that Lemma 9.34 and Lemma 9.36 are not valid. For both procedures, as with `MakeMinimal`, the time needed to set the actual parameter (a DNA expression) can be considered constant. In the proofs of both lemmas, we can simply include this constant time in the time required by the instructions outside the loop(s). The proofs can then proceed in the same way.

We now establish the time complexity of `MakeMinimal`.

[7] If in procedure `Make↕ExprMinimal`, we allow $\mathcal{N}$-word-arguments of E_1 that are not maximal $\mathcal{N}$-word occurrences, then $n_\alpha(E)$ may temporarily increase. However, at the end of that procedure, $n_\alpha(E)$ cannot be higher than at the beginning. For example, if $E_1 = \langle\uparrow \alpha_{1,1}\alpha_{1,2}\,\langle\updownarrow \alpha_{1,3}\rangle\rangle$, then $\langle\updownarrow E_1\rangle$ may be successively rewritten into $\langle\updownarrow\langle\uparrow\langle\updownarrow \alpha_{1,1}\rangle\,\alpha_{1,2}\,\langle\updownarrow \alpha_{1,3}\rangle\rangle\rangle$, $\langle\updownarrow\langle\uparrow\langle\updownarrow \alpha_{1,1}\alpha_{1,2}\alpha_{1,3}\rangle\rangle\rangle$ and $\langle\updownarrow \alpha_{1,1}\alpha_{1,2}\alpha_{1,3}\rangle$.

Theorem 9.37 *Let E_1^* be an arbitrary DNA expression. The time required by the function* `MakeMinimal` *for E_1^* is in $\mathcal{O}(|E_1^*|)$.*

Proof: For an arbitrary DNA expression E, let us use $T_{\mathrm{MM}}(E)$ to denote the time required by `MakeMinimal` for E, except the time spent in procedures `Make`$\updownarrow$`ExprMinimal` and `Denickify`. We prove that $T_{\mathrm{MM}}(E)$ is in $\mathcal{O}(|E|)$. Then the claim follows from Lemma 9.34 and Lemma 9.36.

To analyse $T_{\mathrm{MM}}(E)$, we define three positive constants that are upper bounds for the time spent in specific parts of `MakeMinimal`:

c_1 is the maximum time required by `MakeMinimal` for an $\updownarrow$-expression E, except the time spent in recursive calls of the function and the time spent in procedure `Make`$\updownarrow$`ExprMinimal`.

Hence, c_1 is the maximum time required for setting the actual parameter E in line $1'$ and executing lines $3'$–$11'$ and $36'$ of the function, except the recursive call in line $5'$ and procedure `Make`$\updownarrow$`ExprMinimal` in line $9'$.

c_2 is the maximum time required by `MakeMinimal` for an $\uparrow$-expression E, except the time spent for each of its n arguments $\varepsilon_1, \ldots, \varepsilon_n$.

Hence, c_2 is the maximum time required for setting the actual parameter E in line $1'$ and executing lines $3'$, $12'$, $27'$–$36'$ and the initialization of the for-loop in line $13'$ of the function.

c_3 is the maximum time spent in `MakeMinimal` on an argument ε_i of an $\uparrow$-expression E, except the time spent in recursive calls of the function and the time spent in procedure `Denickify`.

Hence, c_3 is the maximum time required for executing lines $14'$–$26'$ and the iteration in line $13'$ of the function, except the recursive call in line $15'$ and procedure `Denickify` in line $17'$.

It follows from the observations made after the introduction of the first two features and the fourth feature of our data structure (on pages 287 and 297, respectively) and from the observation about passing the parameter for a (recursive) call of `MakeMinimal` (on page 298), that c_1, c_2 and c_3 are indeed constants. They do not depend on, e.g., the nesting level, the question whether or not a DNA expression is alternating, or the number of arguments of a particular DNA expression E.

Note that for most DNA expressions E, we spend less time in `MakeMinimal` than specified by the three constants. For example, the constant c_1 for $\updownarrow$-expressions E is based on the case that $E = \langle\updownarrow E_1\rangle$ for a DNA expression E_1. If, however, $E = \langle\updownarrow \alpha\rangle$ for an $\mathcal{N}$-word α, then we do not have to carry out lines $5'$–$10'$, and thus need much less time.

Now, let the constant c^* be defined by

$$c^* = \max\left\{\frac{c_1}{3}, \frac{c_2 + c_3}{3}, c_3\right\}.$$

We prove by induction on the number p of operators occurring in E, that $T_{\mathrm{MM}}(E) \leq c^* \cdot |E| - c_3$. Here, we subtract c_3, to be prepared for the additional constant time required for every argument of an $\uparrow$-expression E. [8] We will come back to this later. Although

[8] The reader who is familiar with amortized complexity may view this as a kind of amortization: a certain part of the time spent on the arguments of an $\uparrow$-expression (c_3 per argument) is accounted for by the individual arguments.

we may assume that $\mathcal{N}$-word-arguments of an $\uparrow$-expression or $\downarrow$-expression are maximal $\mathcal{N}$-word occurrences, we do not make that assumption in the proof.

- Assume that $p = 1$. Then E can only have $\mathcal{N}$-word-arguments, and we do not have recursive calls of `MakeMinimal`.

  If E is an $\updownarrow$-expression, then $E = \langle \updownarrow \alpha_1 \rangle$ for an $\mathcal{N}$-word α_1 and $T_{\mathrm{MM}}(E) \leq c_1$. Clearly, $|E| \geq 4$. We now distinguish two (overlapping) subcases. If $c_1 \geq 3c_3$, then

  $$T_{\mathrm{MM}}(E) \leq c_1 = c_1 + c_3 - c_3 \leq \frac{4}{3}c_1 - c_3 \leq 4c^* - c_3 \leq c^* \cdot |E| - c_3,$$

  where the third inequality follows from $c^* \geq \frac{c_1}{3}$. If, on the other hand, $c_1 \leq 3c_3$, then

  $$T_{\mathrm{MM}}(E) \leq c_1 \leq 3c_3 = 4c_3 - c_3 \leq 4c^* - c_3 \leq c^* \cdot |E| - c_3,$$

  where the third inequality follows from $c^* \geq c_3$.

  If E is an $\uparrow$-expression, then $E = \langle \uparrow \alpha_1 \ldots \alpha_n \rangle$ for some $n \geq 1$ and $\mathcal{N}$-words $\alpha_1, \ldots, \alpha_n$. In this case, $|E| \geq n + 3$ and

  $$T_{\mathrm{MM}}(E) \;\leq\; c_2 + n \cdot c_3 \leq 3c^* - c_3 + n \cdot c^* \leq c^* \cdot |E| - c_3,$$

  where the second inequality follows from $c^* \geq \frac{c_2 + c_3}{3}$ (which is equivalent to $c_2 \leq 3c^* - c_3$) and $c^* \geq c_3$.

  If E is a $\downarrow$-expression, then the proof is completely analogous.

- Let $p \geq 1$, and suppose that $T_{\mathrm{MM}}(E) \leq c^* \cdot |E| - c_3$ for all DNA expressions E containing at most p operators (induction hypothesis). Now let E be a DNA expression that contains $p + 1$ operators.

  If E is an $\updownarrow$-expression, then $E = \langle \updownarrow E_1 \rangle$ for a DNA expression E_1. We get a recursive call of `MakeMinimal` for E_1, in line $5'$ of the function. Hence, $T_{\mathrm{MM}}(E) \leq c_1 + T_{\mathrm{MM}}(E_1)$. Because E_1 contains p operators, we can apply the induction hypothesis to it:

  $$\begin{aligned} T_{\mathrm{MM}}(E) \;&\leq\; c_1 + T_{\mathrm{MM}}(E_1) \leq c_1 + c^* \cdot |E_1| - c_3 \\ &\leq\; c^* \cdot (|E_1| + 3) - c_3 = c^* \cdot |E| - c_3, \end{aligned}$$

  where the third inequality follows from $c^* \geq \frac{c_1}{3}$.

  If E is an $\uparrow$-expression, then $E = \langle \uparrow \varepsilon_1 \ldots \varepsilon_n \rangle$ for some $n \geq 1$ and $\mathcal{N}$-words and DNA expressions $\varepsilon_1, \ldots, \varepsilon_n$. We examine the time spent in `MakeMinimal` on an argument ε_i with $1 \leq i \leq n$. If ε_i is an $\mathcal{N}$-word α_i, then the time spent on this argument is bounded by

  $$c_3 \leq c^* \leq c^* \cdot |\alpha_i|,$$

  because obviously $|\alpha_i| \geq 1$. If, on the other hand, ε_i is a DNA expression E_i, then we have a recursive call of `MakeMinimal` for E_i, in line $15'$ of the function. Hence, the time spent on this argument is bounded by $c_3 + T_{\mathrm{MM}}(E_i)$. Because E_i contains

at most p operators, the induction hypothesis is applicable to it. This implies that the time spent on E_i is bounded by

$$c_3 + T_{\mathrm{MM}}(E_i) \leq c_3 + c^* \cdot |E_i| - c_3 = c^* \cdot |E_i|.$$

Note that here we benefit from the term $-c_3$ in the upper bound for $T_{\mathrm{MM}}(E)$.

We conclude that both if ε_i is an $\mathcal{N}$-word α_i and if it is a DNA expression E_i, we spend at most $c^* \cdot |\varepsilon_i|$ time on it, apart from procedures `Make`$\updownarrow$`ExprMinimal` and `Denickify`. Then

$$\begin{aligned} T_{\mathrm{MM}}(E) &\leq c_2 + c^* \cdot (|\varepsilon_1| + \cdots |\varepsilon_n|) \\ &\leq 3c^* - c_3 + c^* \cdot (|\varepsilon_1| + \cdots |\varepsilon_n|) = c^* \cdot |E| - c_3, \end{aligned}$$

where the second inequality follows from $c^* \geq \frac{c_2+c_3}{3}$.

If E is a $\downarrow$-expression, then the proof is completely analogous.

□

At the beginning of this section, we observed that the function `MakeMinimal` requires at least linear time. Combining this with Theorem 9.37, we obtain the following result:

Corollary 9.38 *Let E_1^* be an arbitrary DNA expression. The time required by the function* `MakeMinimal` *for E_1^* is linear in $|E_1^*|$.*

It is not hard to see that the data structure we propose to achieve this time complexity, has linear size. For each letter (symbol) in the DNA expression, we need to store (at most) a constant number of references to other letters.

For example, for the first feature of the data structure, the doubly-linked list containing the entire DNA expression, we need two references per letter: one to the preceding and one to the succeeding letter. For the other three features, the space required depends on the DNA expression at hand. It may be much less than linear, but a single, simple example suffices to demonstrate that each of these features may really require linear space.

Example 9.39 Let α be an arbitrary $\mathcal{N}$-word, and let E_p be defined by

$$E_p = \left\langle \uparrow \underbrace{\langle\uparrow \alpha\rangle \langle\uparrow \alpha\rangle \ldots \langle\uparrow \alpha\rangle}_{p \text{ times}} \right\rangle \qquad (p \geq 1).$$

It is easy to see that for any $p \geq 1$, E_p is a DNA expression, with $|E_p| = 3 + p \cdot (3 + |\alpha|) = 3 + 3p + p \cdot |\alpha|$ and $\mathcal{S}(E_p) = \underbrace{\binom{\alpha}{_}\binom{\alpha}{_}\ldots\binom{\alpha}{_}}_{p \text{ times}}$. In addition, for any $p \geq 1$,

- E_p contains $p+1$ pairs of matching brackets. Hence, the second feature of the data structure requires $p+1$ connections (in both directions) between an opening bracket and the corresponding closing bracket.
- E_p contains p occurrences of the $\mathcal{N}$-word α (in fact, maximal $\mathcal{N}$-word occurrences), each of which serves as the argument of an operator $\uparrow$. Hence, the second feature of the data structure requires p connections (in both directions) between the first letter and the last letter of such an $\mathcal{N}$-word-argument.

- the outermost operator $\uparrow$ of E_p has p arguments $\langle \uparrow \alpha \rangle$, which are, in particular, non-$\updownarrow$-arguments. Hence, the third feature of the data structure requires a circular, doubly-linked list for this operator containing these p arguments.

- E_p contains p inner occurrences of the operator $\uparrow$. Each of these inner occurrences has an $\mathcal{N}$-word-argument α, which is, in particular, a non-$\updownarrow$-argument. Hence, the third feature of the data structure requires p circular, doubly-linked lists for these operators, each containing the corresponding $\mathcal{N}$-word-argument.

- the outermost operator $\uparrow$ of E_p has p arguments $\langle \uparrow \alpha \rangle$, which are, in particular, consecutive expression-arguments. Hence, the fourth feature of the data structure requires a circular, doubly-linked list for this operator containing the last $p - 1$ arguments (each of which is the second of two consecutive expression-arguments).

Each of the specified sets of connections or doubly-linked lists requires space that is linear in p, and thus in $|E_p|$. ■

As we mentioned before the statement of Theorem 9.37 (on page 298), a single call of the function `MakeMinimal` requires constant space to pass the (only) parameter, the DNA expression E. The function is called recursively once for every DNA subexpression of E_1^*, i.e., once for every operator occurring in E_1^*. Hence, the total space required for passing the parameter for all recursive calls is at most linear in $|E_1^*|$.

We can therefore conclude:

Theorem 9.40 *Let E_1^* be an arbitrary DNA expression. The space required by the function* `MakeMinimal` *for E_1^* is linear in $|E_1^*|$.*

Hence, both the time complexity and the space complexity of the function are linear.

9.4 Decrease of length by the algorithm

In the previous section, we proved that the total time required by the function `MakeMinimal` is linear in the length of its argument E. We first observed that we need at least linear time, because we must in principle consider every letter of E. That is, we must simply read E. We subsequently introduced a proper data structure, which can be initialized in linear time. We proved that with this data structure, we can perform all rewriting steps in the function (together) in $\mathcal{O}(|E|)$ time. Our analysis did not differentiate between DNA expressions which are close to minimal and DNA expressions which are far from minimal.

Now, we choose a different approach. We ignore the time needed to read E and to initialize the data structure. We focus on the actual rewriting steps, and prove that the time they require is proportional to the improvements they produce, i.e., to the decrease of $|E|$ resulting from them. For this, however, we need to make an assumption about E, and to slightly adjust one of the rewriting steps.

So far, in the analysis of our algorithm, we allowed occurrences of operators $\uparrow$ and $\downarrow$ in E to have consecutive $\mathcal{N}$-word-arguments. That is, we did not assume $\mathcal{N}$-word-arguments to be maximal $\mathcal{N}$-word occurrences in E. We sometimes mentioned the possibility to make such an assumption, but we did not need it to prove that the algorithm is correct and that it runs in linear time. By Theorem 4.3, however, we can make the assumption without loss of generality.

The adjustment of the algorithm deals with complementing $\mathcal{N}$-words. In the proof of Lemma 9.34, we observed that in procedure `Make↕ExprMinimal`, we may have to determine the elementwise complement of an $\mathcal{N}$-word α. This requires time that is linear in $|\alpha|$. Instead of doing this, we can also *mark* the $\mathcal{N}$-word as a whole, to indicate that it has to be complemented. For example, we can label its first letter and last letter, as if we write "$c(\alpha)$". [9] That requires constant time.

Note that the issue whether or not the $\mathcal{N}$-word-arguments of an operator are maximal $\mathcal{N}$-word occurrences may have consequences for the (number of) steps to be performed in procedure `Make↕ExprMinimal`. However, the result of the procedure does not depend on it. It is not hard to verify this from the pseudo-code of the procedure directly. It also follows from Theorem 9.20(2), which states that there exists exactly one minimal DNA expression E' with the desired semantics.

For the other operations performed in the course of the algorithm, it does not matter at all whether or not $\mathcal{N}$-word-arguments are maximal $\mathcal{N}$-word occurrences. Marking $\mathcal{N}$-words instead of determining their elementwise complement certainly does not change the resulting DNA expression.

We now have

Theorem 9.41 *Let E_1^* be an arbitrary DNA expression, and let E_2^* be the result of applying the function* `MakeMinimal` *to E_1^*. Assume that for each occurrence of $\uparrow$ or $\downarrow$ in E_1^*, each $\mathcal{N}$-word-argument is a maximal $\mathcal{N}$-word occurrence, and that we simply mark the $\mathcal{N}$-words that have to be complemented. Then the time required by the rewriting steps in* `MakeMinimal` *is linear in $|E_1^*| - |E_2^*|$.*

Proof: Let E be the 'working DNA expression' of the function `MakeMinimal`. In principle, we prove that each substitution in the function corresponds to a decrease of $|E|$ that is proportional to the time required by the substitution. As we will see below, there is one exception to this rule: for the substitution in line $20'$, we need to combine the effect with the effect of another substitution.

- In line $7'$ of `MakeMinimal`, we have an $\updownarrow$-expression $E = \langle\updownarrow E_1\rangle$, where E_1 is a minimal $\updownarrow$-expression. We substitute E by E_1. This requires constant time and yields a decrease of $|E|$ by 3.

- In line $9'$ of `MakeMinimal`, we have an $\updownarrow$-expression $E = \langle\updownarrow E_1\rangle$, where E_1 is either a minimal $\uparrow$-expression, or a minimal $\downarrow$-expression. Without loss of generality, assume it is a minimal $\uparrow$-expression. We substitute E by the result of procedure `Make↕ExprMinimal`.

  This substitution is a bit more involved. For its analysis, we distinguish specific parts of procedure `Make↕ExprMinimal`, and examine how much time we spend and how much shorter E becomes in each part. To simplify the notation, we do not consider the length of E directly. Instead, we count (changes in) the number of operators occurring in E. By Lemma 6.1, this number determines $|E|$, given that (the number of $\mathcal{A}$-letters in) the semantics of E is fixed.

  - A certain part of procedure `Make↕ExprMinimal` is executed once for every DNA expression E to which the procedure is applied. This part consists of lines

[9] In this thesis, we often do write $c(\alpha)$ in a DNA expression. This is, however, only meant as a simple notation for the elementwise complement of α. In particular, $|c(\alpha)| = |\alpha|$. Indeed, the letters c, (and) are not in the alphabet $\Sigma_{\mathcal{D}}$ that our language of DNA expressions is based on (see page 43).

M↕M.1–M↕M.3, M↕M.19–M↕M.23, and the initializations of the two for-loops in lines M↕M.4 and M↕M.7.

Let c_1 and d_1 be the minimum and maximum time spent in this part of the procedure, respectively. In lines M↕M.19–M↕M.22, the final rewriting step on the (total) 'working DNA expression'

$$\left\langle \updownarrow \widehat{E}_1 \right\rangle = \langle \updownarrow \langle \uparrow \langle \updownarrow \alpha_{1,1} \rangle \dots \langle \updownarrow \alpha_{1,m} \rangle \rangle \rangle$$

is performed. As a result, we lose one (if $m \geq 2$) or two (if $m = 1$) operators.

– Another part of the procedure is executed once for each $\downarrow$-argument $E_{1,i}$ of E_1. This part consists of lines M↕M.5, M↕M.6 and the iteration of the for-loop in line M↕M.4.

In line M↕M.5, we substitute $E_{1,i}$ by $\left\langle \updownarrow \alpha_{E_{1,i}} \right\rangle$. In the proof of Lemma 9.34, we have observed that we can determine $\alpha_{E_{1,i}}$ by traversing $E_{1,i}$ from left to right, skipping operators and brackets, complementing maximal $\mathcal{N}$-word occurrences that used to have $\downarrow$ as parent operator, and linking consecutive maximal $\mathcal{N}$-word occurrences. We established that the time required for traversing, skipping and linking is linear in $n_\alpha(E_{1,i})$.

By assumption, we only *mark* maximal $\mathcal{N}$-word occurrences that have to be complemented. This implies that the time required for determining $\alpha_{E_{1,i}}$ completely (including the marking of maximal $\mathcal{N}$-word occurrences) is linear in $n_\alpha(E_{1,i})$. Consequently, the time spent in procedure `Make↕ExprMinimal` on $E_{1,i}$ is also linear in $n_\alpha(E_{1,i})$. Let c_2 and d_2 be positive constants such that $c_2 \cdot n_\alpha(E_{1,i})$ and $d_2 \cdot n_\alpha(E_{1,i})$ are a lower bound and an upper bound for this time, respectively.

We now examine how many operators we lose by substituting $E_{1,i}$ by $\left\langle \updownarrow \alpha_{E_{1,i}} \right\rangle$. Let us use p to denote the number of operators occurring in $E_{1,i}$. We derive an upper bound and a lower bound for p. As we observed in the proof of Lemma 9.34, $p \leq n_\alpha(E_{1,i})$. On the other hand, let $X_{1,i} = \mathcal{S}(E_{1,i})$. By Corollary 8.17(2),

$$p = 1 + B_\uparrow(X_{1,i}) + n_\updownarrow(X_{1,i}) \geq 1 + n_\updownarrow(X_{1,i}). \tag{9.27}$$

We also observed in the proof of Lemma 9.34 that

$$n_{\alpha\uparrow\downarrow}(E_{1,i}) = n_{\uparrow\downarrow}(X_{1,i}), \tag{9.28}$$

$$n_{\alpha\updownarrow}(E_{1,i}) = n_\updownarrow(X_{1,i}) \tag{9.29}$$

Now by Corollary 3.8, double components and single-stranded components alternate in the nick free formal DNA molecule $X_{1,i}$. By Lemma 8.27(6), either the first component or the last component of $X_{1,i}$ is a double component. Hence, $n_\updownarrow(X_{1,i}) \geq n_{\uparrow\downarrow}(X_{1,i})$. Combining this with (9.27)–(9.29), we find

$$\begin{aligned} p &\geq 1 + n_\updownarrow(X_{1,i}) \geq 1 + \frac{1}{2}(n_{\uparrow\downarrow}(X_{1,i}) + n_\updownarrow(X_{1,i})) \\ &= 1 + \frac{1}{2}(n_{\alpha\uparrow\downarrow}(E_{1,i}) + n_{\alpha\updownarrow}(E_{1,i})) = 1 + \frac{1}{2}n_\alpha(E_{1,i}). \end{aligned}$$

We can conclude that by substituting $E_{1,i}$ by $\left\langle \updownarrow \alpha_{E_{1,i}} \right\rangle$, we lose at most $n_\alpha(E_{1,i}) - 1$ and at least $\frac{1}{2}n_\alpha(E_{1,i})$ operators.

– Finally, a part of the procedure is executed once for each $\mathcal{N}$-word-argument $\alpha_{1,i}$ of E_1. This part consists of lines M↕M.8–M↕M.18 and the iteration of the for-loops in lines M↕M.4 (given that we do not have a list of ↓-arguments of E_1, but only a list of non-↕-arguments) and M↕M.7.

  Let c_3 and d_3 be the minimum and maximum time spent in this part of the procedure for a single $\mathcal{N}$-word-argument, and let $k \geq 0$ be the number of $\mathcal{N}$-word-arguments. By assumption, each $\mathcal{N}$-word-argument of E_1 is a maximal $\mathcal{N}$-word occurrence.

  We examine the result of the substitution of the $\mathcal{N}$-word-arguments in lines M↕M.8–M↕M.17. At that point in the procedure, the only other arguments of the 'working ↑-subexpression' $\widehat{E}_1$ are ↕-arguments $\langle\updownarrow \alpha_{1,i}\rangle$ for $\mathcal{N}$-words $\alpha_{1,i}$. Let us denote the number of operators occurring in $\widehat{E}_1$ by $\mathrm{Op}(\widehat{E}_1)$. We distinguish three cases.

  If an $\mathcal{N}$-word-argument $\alpha_{1,i}$ is neither preceded, nor succeeded by an ↕-argument (i.e., if it is the only argument of $\widehat{E}_1$), then $\alpha_{1,i}$ is substituted by $\langle\updownarrow \alpha_{1,i}\rangle$. In this case, $\mathrm{Op}(\widehat{E}_1)$ *increases* by 1. Otherwise, if an $\mathcal{N}$-word-argument $\alpha_{1,i}$ is not preceded or not succeeded by an ↕-argument (i.e., if $\alpha_{1,i}$ is the first or the last argument of $\widehat{E}_1$), then the corresponding substitution does not affect $\mathrm{Op}(\widehat{E}_1)$. Finally, if an $\mathcal{N}$-word-argument $\alpha_{1,i}$ is both preceded and succeeded by an ↕-argument, then the corresponding substitution yields a decrease of $\mathrm{Op}(\widehat{E}_1)$ by 1.

  Clearly, there are at most two $\mathcal{N}$-word-arguments that are the first or the last argument of $\widehat{E}_1$. Hence, if $k \geq 3$, then the substitution of the $\mathcal{N}$-word-arguments results in a decrease of $\mathrm{Op}(\widehat{E}_1)$ by at least $k-2$. In other words, in that case, we lose at least $k-2$ operators.

We now combine the effects of the different parts of procedure `Make↕ExprMinimal` to compute the overall effect for an ↕-expression E. Let $T_{\mathrm{M}\updownarrow\mathrm{M}}(E)$ be the total time spent in the procedure for E and let $\delta(E)$ be the decrease of the number of operators due to the substitutions in the procedure. Then

$$c_1 + c_2 \cdot \sum_{\downarrow\text{-arg } E_{1,i}} n_\alpha(E_{1,i}) + c_3 \cdot k \leq T_{\mathrm{M}\updownarrow\mathrm{M}}(E) \leq d_1 + d_2 \cdot \sum_{\downarrow\text{-arg } E_{1,i}} n_\alpha(E_{1,i}) + d_3 \cdot k, \qquad (9.30)$$

where k is (again) the number of $\mathcal{N}$-word-arguments of the ↑-expression E_1.

For $\delta(E)$, we distinguish three cases, which are related to the three cases for $\mathcal{N}$-word-arguments we considered above.

If E_1 does not have any expression-argument, then because its $\mathcal{N}$-word-arguments are maximal $\mathcal{N}$-word occurrences, it has only one argument, which is an $\mathcal{N}$-word $\alpha_{1,1}$. In this case,

$$\delta(E) = 2 + 0 - 1 = 1,$$

where the three terms correspond to the three parts of the procedure. If the ↑-expression E_1 has at least one expression-argument and $k \leq 2$ $\mathcal{N}$-word-arguments, then

$$1 + \frac{1}{2} \sum_{\downarrow\text{-arg } E_{1,i}} n_\alpha(E_{1,i}) + 0 \leq \delta(E) \leq 2 + \sum_{\downarrow\text{-arg } E_{1,i}} (n_\alpha(E_{1,i}) - 1) + k. \tag{9.31}$$

Finally, if the ↑-expression E_1 has at least one expression-argument and $k \geq 3$ $\mathcal{N}$-word-arguments,[10] then at least $k - 2 \geq 1$ $\mathcal{N}$-word-arguments are both preceded and succeeded by an expression-argument, and

$$1 + \frac{1}{2} \sum_{\downarrow\text{-arg } E_{1,i}} n_\alpha(E_{1,i}) + k - 2 \leq \delta(E) \leq 2 + \sum_{\downarrow\text{-arg } E_{1,i}} (n_\alpha(E_{1,i}) - 1) + k.$$

In the first case, where E_1 only has an $\mathcal{N}$-word-argument $\alpha_{1,1}$, we have $k = 1$ and no ↓-arguments $E_{1,i}$ at all. Hence, the value $\delta(E) = 1$ also satisfies (9.31).

Now, let us define the constants c^* and d^* by

$$c^* = \max\left\{\frac{2}{c_1}, \frac{1}{c_2}, \frac{1}{c_3}\right\} \qquad \text{and} \qquad d^* = \min\left\{\frac{1}{d_1 + 2d_3}, \frac{1}{2d_2}\right\}.$$

If $k \leq 2$, then

$$\begin{aligned}
\delta(E) &\geq 1 + \frac{1}{2} \sum_{\downarrow\text{-arg } E_{1,i}} n_\alpha(E_{1,i}) \\
&\geq d^* \cdot \left(d_1 + 2d_3 + d_2 \cdot \sum_{\downarrow\text{-arg } E_{1,i}} n_\alpha(E_{1,i})\right) \\
&\geq d^* \cdot \left(d_1 + d_2 \cdot \sum_{\downarrow\text{-arg } E_{1,i}} n_\alpha(E_{1,i}) + d_3 \cdot k\right) \geq d^* \cdot T_{\mathrm{M}\updownarrow\mathrm{M}}(E),
\end{aligned}$$

where the second inequality follows from $d^* \leq \frac{1}{d_1+2d_3}$ and $d^* \leq \frac{1}{2d_2}$, and the last inequality follows from (9.30). If, on the other hand, $k \geq 3$, then

$$\begin{aligned}
\delta(E) &\geq 1 + \frac{1}{2} \sum_{\downarrow\text{-arg } E_{1,i}} n_\alpha(E_{1,i}) + k - 2 \\
&\geq d^* \cdot \left(d_1 + 2d_3 + d_2 \cdot \sum_{\downarrow\text{-arg } E_{1,i}} n_\alpha(E_{1,i}) + d_3 \cdot (k-2)\right) \geq d^* \cdot T_{\mathrm{M}\updownarrow\mathrm{M}}(E),
\end{aligned}$$

where the second inequality follows from $d^* \leq \frac{1}{d_1+2d_3}$, $d^* \leq \frac{1}{2d_2}$ and $d^* \leq \frac{1}{d_1+2d_3} \leq \frac{1}{d_3}$, and the last inequality follows from (9.30).

[10] Because the $\mathcal{N}$-word-arguments are maximal $\mathcal{N}$-word occurrences, E_1 actually has at least $k - 1 \geq 2$ expression-arguments.

Further, for all cases,

$$\begin{aligned}\delta(E) &\leq 2 + \sum_{\downarrow\text{-arg } E_{1,i}} (n_\alpha(E_{1,i}) - 1) + k \\ &\leq c^* \cdot \left(c_1 + c_2 \cdot \sum_{\downarrow\text{-arg } E_{1,i}} (n_\alpha(E_{1,i}) - 1) + c_3 \cdot k \right) \\ &\leq c^* \cdot \left(c_1 + c_2 \cdot \sum_{\downarrow\text{-arg } E_{1,i}} n_\alpha(E_{1,i}) + c_3 \cdot k \right) \leq c^* \cdot T_{\mathrm{M}\updownarrow\mathrm{M}}(E),\end{aligned}$$

where the second inequality follows from $c^* \geq \frac{2}{c_1}$, $c^* \geq \frac{1}{c_2}$ and $c^* \geq \frac{1}{c_3}$, and the last inequality follows from (9.30).

We can conclude that $T_{\mathrm{M}\updownarrow\mathrm{M}}(E)$ (the time spent in procedure `Make↕ExprMinimal` for E) is linear in $\delta(E)$ (the decrease of the number of operators due to the application of the procedure to E). Thus $T_{\mathrm{M}\updownarrow\mathrm{M}}(E)$ is also linear in the corresponding decrease of $|E|$. In other, less formal words: the time we spend in procedure `Make↕ExprMinimal` is payed for with a proportional decrease of $|E|$.

Note that the above conclusion is valid for the application of the procedure as a whole. As we have seen when we analysed the substitution of $\mathcal{N}$-word-arguments of E_1, the substitution of an individual $\mathcal{N}$-word-argument does not necessarily yield a decrease of $|E|$; it may even lead to an increase of $|E|$.

- In line 17′ of `MakeMinimal`, we have an $\uparrow$-expression E, with a minimal $\downarrow$-argument E_i that is not alternating. We substitute E_i by the result of procedure `Denickify`.

  In that procedure, the 'working $\downarrow$-expression' is denoted by $\widehat{E}_i$. As in the previous case, we count (changes in) the number of operators occurring in $\widehat{E}_i$, rather than examining $|\widehat{E}_i|$ directly. We also use $\mathrm{Op}(\widehat{E}_i)$ to denote the number of operators occurring in $\widehat{E}_i$.

  Let $T_{\mathrm{Dni}}(E_i)$ be the time we spend in procedure `Denickify` for E_i, and let $\delta(E_i)$ be the number of operators we lose due to the application of the procedure to E_i.

  As we observed in the proof of Lemma 9.36, $T_{\mathrm{Dni}}(E_i)$ is linear in $n_{\mathrm{iter}}(E_i)$, the number of iterations of the while-loop in lines Dni.4–Dni.19 for E_i.

  We now examine $\delta(E_i)$. In every iteration of the loop, we substitute two consecutive expression-arguments $\widehat{\varepsilon}_{j-1}$ and $\widehat{\varepsilon}_j$ of $\widehat{E}_i$ by a single expression-argument. It is easily verified from the pseudo-code of the procedure, that this results in a decrease of $\mathrm{Op}(\widehat{E}_i)$ by 1 (if either $\widehat{\varepsilon}_{j-1}$ or $\widehat{\varepsilon}_j$ is an $\updownarrow$-expression) or 2 (if both $\widehat{\varepsilon}_{j-1}$ and $\widehat{\varepsilon}_j$ are $\uparrow$-expressions). Hence, the decrease of $\mathrm{Op}(\widehat{E}_i)$ as a result of (all iterations of) the while-loop is linear in $n_{\mathrm{iter}}(E_i)$.

  At the end of procedure `Denickify`, in lines Dni.20–Dni.26, we distinguish three cases. If $\widehat{E}_i$ has only one argument left, then we substitute $\widehat{E}_i$ by this argument. Thus, we lose one more operator. If $\widehat{E}_i$ has two or more arguments, and both its first argument and its last argument are $\uparrow$-arguments, then we substitute $\widehat{E}_i$ by an $\uparrow$-expression $\widehat{E}'_i$, which is the result of procedure `RotateToMinimal`. It is easy to see from the pseudo-code of that procedure (on page 272), that in this case, we also lose

an operator. Finally, if $\widehat{E}_i$ has two or more arguments, and either its first argument or its last argument is not an $\uparrow$-argument, then we do not rewrite $\widehat{E}_i$ any further.

We conclude that we lose either zero or one operator in lines Dni.20–Dni.26 of procedure `Denickify`. Because the original $\downarrow$-expression E_i is not alternating, $n_{\text{iter}}(E_i) \geq 1$. This implies that $\delta(E_i)$, the total number of operators we lose due to the application of procedure `Denickify` (both inside and outside the while-loop), is linear in $n_{\text{iter}}(E_i)$.

Because $T_{\text{Dni}}(E_i)$ is also linear in $n_{\text{iter}}(E_i)$, $T_{\text{Dni}}(E_i)$ is linear in $\delta(E_i)$. By Lemma 6.1, $T_{\text{Dni}}(E_i)$ is also linear in the corresponding decrease of $|E|$.

- In line $20'$ of `MakeMinimal`, we have an $\uparrow$-expression E with a $\downarrow$-argument E_i, such that either the first argument or the last argument of E_i is an $\uparrow$-argument. By the recursive call in line $15'$ of the function, and the possible application of procedure `Denickify` in line $17'$, E_i is minimal and alternating.

  We substitute E_i by the result of procedure `RotateToMinimal`, which is an equivalent, minimal $\uparrow$-expression E_i'. Because E_i and E_i' are equivalent and both of them are minimal, they are equally long. Hence, the substitution itself does not yield a decrease of $|E|$. However, in the next step of the function, in line $23'$, the $\uparrow$-expression E_i' is substituted by its arguments. Both substitutions for E_i require constant time, and the total effect of the substitutions is a decrease of $|E|$ by 3.

- In line $23'$ of `MakeMinimal`, we have an $\uparrow$-expression E with an $\uparrow$-argument E_i. Because E_i is not necessarily the product of the substitution in line $20'$ (which we considered in the previous case), we also consider this case separately.

  We substitute E_i by its arguments. This requires constant time and yields a decrease of $|E|$ by 3.

- In line $29'$ of `MakeMinimal`, we have an $\uparrow$-expression E with exactly one argument, which is a DNA expression E_1. We substitute E by E_1. This requires constant time and yields a decrease of $|E|$ by 3.

- Finally, in line $33'$ of `MakeMinimal`, we have an alternating $\uparrow$-expression E with at least two arguments, such that both the first argument and the last argument are $\downarrow$-arguments.

  We substitute E by the result of (the version for $\uparrow$-expressions of) procedure `RotateToMinimal`. This requires constant time. As was the case for the application of `RotateToMinimal` in procedure `Denickify`, it is easy to see that the substitution yields a decrease of $|E|$ by 3.

This completes the proof of Theorem 9.41. □

If a DNA expression E_1^* is minimal, then its length is equal to that of the equivalent, minimal DNA expression E_2^* produced by `MakeMinimal`. Now Theorem 9.41 implies that `MakeMinimal` spends no time on actual rewriting steps for E_1^*. In other words, E_2^* must be equal to E_1^*. Thus, Theorem 9.41 yields an alternative proof for Theorem 9.12.

This conclusion does not depend on the assumptions in Theorem 9.41, that each $\mathcal{N}$-word-argument of an operator $\uparrow$ or $\downarrow$ is a maximal $\mathcal{N}$-word occurrence and that we simply mark $\mathcal{N}$-words that need to be complemented. As we have seen in the proof of Theorem 9.41, each substitution we perform in `MakeMinimal` corresponds to a decrease of $|E|$.

This is also true if the assumptions are not satisfied, because, as we observed before the theorem, the result of a substitution is independent of the assumptions. If E is minimal already, then $|E|$ cannot decrease, and we cannot have any substitution, either.

Part III

Minimal Normal Form

Chapter 10

A Minimal Normal Form for DNA Expressions

DNA expressions can become rather complex, with arbitrarily high nesting levels of the brackets. For example, if E is a DNA expression, then so are $\langle\updownarrow E\rangle$, $\langle\updownarrow\langle\updownarrow E\rangle\rangle$, $\langle\updownarrow\langle\updownarrow\langle\updownarrow E\rangle\rangle\rangle$, etc. (see the proof of Lemma 4.23). As we have seen in the proof of Lemma 7.34, even the nesting level of a *minimal* DNA expression can get arbitrarily high. Some DNA expressions denoting a formal DNA molecule are easier to parse (for a human reader) than others for the same molecule. This holds true also for minimal DNA expressions.

A *normal form* may be useful to reduce the complexity of DNA expressions. In this thesis, a normal form is a specific set of properties, such that for each DNA expression, there is exactly one equivalent DNA expression having these properties. Now, the challenge is to find a set of properties that ensures that the DNA expression is easy to parse.

A normal form may also suit another purpose. When we want to find out if two DNA expressions E_1 and E_2 are equivalent, we can do this in a straightforward way, by computing their semantics and checking if these are the same. By Theorem 4.18, computing the semantics $\mathcal{S}(E)$ of a DNA expression E takes time that is linear in the length of E. Moreover, as we observed at the end of Section 4.4, the length of $\mathcal{S}(E)$ is at most linear in the length of E itself. Therefore, the above approach for checking equivalence takes time that is linear in the length of E_1 and E_2, which is certainly efficient.

When we have a normal form for our DNA expressions, we can also choose a different approach. Two DNA expressions E_1 and E_2 are equivalent, if and only if their normal form versions are the same. This can be decided, when we manage to rewrite E_1 and E_2 into these normal form versions. This alternative approach is more elegant, because it operates at the level of DNA expressions only. It does not refer to the semantics of the DNA expressions.

Before using a normal form for whatever purpose, we must first decide what the normal form should look like, i.e., what properties the DNA expressions in normal form should have. For this, we observe that minimal DNA expressions can be considered as the 'best' DNA expressions. They require the smallest number of letters to denote a formal DNA molecule. Moreover, we know exactly what are the minimal DNA expressions for a given formal DNA molecule (see Summary 8.16). We also have a useful characterization of minimal DNA expressions in general (see Lemma 8.22 and Theorem 8.26). Therefore, it would be desirable that normal form DNA expressions be minimal. In this chapter, we will describe a normal form which achieves this goal. Because of this, we will refer to it

as the *minimal normal form*.

We first describe the minimal normal form in a constructive way: for each expressible formal DNA molecule, we specify how to construct the corresponding DNA expression in minimal normal form. Next, we give a characterization of the normal form DNA expressions, by five simple, syntactic properties. From this characterization, we deduce that the nesting level of the brackets in these DNA expressions is bounded. We subsequently consider the structure trees of DNA expressions in minimal normal form.

Finally, we consider the language-theoretic complexity of the set of all DNA expressions in minimal normal form. We give a context-free grammar generating this set and prove that that grammar is not self-embedding. This implies that the DNA expressions in minimal normal form constitute a regular language. An advantage of this is, that we could even use a simple machine like a finite automaton to decide whether or not a string is a member of this language, i.e., whether or not it is a DNA expression in minimal normal form. This is not possible for the set of all DNA expressions and for the set of all minimal DNA expressions, as these languages are not regular, see Lemma 4.23 and Lemma 7.34.

10.1 Definition of the minimal normal form

As we have seen in Chapter 7 and Chapter 8 (and especially in Section 8.5), for many formal DNA molecules, there exists more than one minimal DNA expression. Hence, minimality alone is not sufficient to define a normal form. From among all different minimal DNA expressions denoting the same formal DNA molecule, we have to choose one to be the normal form DNA expression. We do this by explicitly fixing the choices that are made in the construction of a minimal DNA expression.

First, this construction is based on lower block partitionings and upper block partitionings of nick free (sub)molecules, see, e.g., the overview in Summary 8.16. If these partitionings are not unique, then the resulting DNA expression depends on the partitionings that we choose. Here, we make a very natural choice: we always use the *primitive* lower block partitioning or upper block partitioning.

In addition, if a formal DNA molecule X is nick free, contains at least one single-stranded component and $B_{\uparrow}(X) = B_{\downarrow}(X)$, then there exist both minimal $\uparrow$-expressions and minimal $\downarrow$-expressions. Here, our choice for an $\uparrow$-expression or a $\downarrow$-expression is determined by the first single-stranded component of X. An upper component results in an $\uparrow$-expression; a lower component results in a $\downarrow$-expression.

We thus have the following definition of the minimal normal form, where $E_{\mathrm{MinNF}}(X)$ denotes the normal form DNA expression for a formal DNA molecule X (cf. Summary 8.16):

Definition 10.1 *Let X be an expressible formal DNA molecule.*

1. *If $X = \binom{\alpha_1}{c(\alpha_1)}$ for an $\mathcal{N}$-word α_1, then $E_{MinNF}(X) = \langle \updownarrow \alpha_1 \rangle$.*
2. *If X is nick free, contains at least one single-stranded component and $B_{\uparrow}(X) = B_{\downarrow}(X)$, then*
   (a) *if the first single-stranded component of X is an upper component, then $E_{MinNF}(X)$ is the minimal $\uparrow$-expression denoting X based on the primitive lower block partitioning of X, as described in Theorem 7.24(1);*

(b) *if the first single-stranded component of X is a lower component, then $E_{MinNF}(X)$ is the minimal $\downarrow$-expression denoting X based on the primitive upper block partitioning of X, as described in Theorem 7.24(2).*

3. *If X is nick free and $B_{\uparrow}(X) > B_{\downarrow}(X)$, then $E_{MinNF}(X)$ is the minimal $\uparrow$-expression denoting X based on the primitive lower block partitioning of X, as described in Theorem 7.24(1).*

4. *If X is nick free and $B_{\downarrow}(X) > B_{\uparrow}(X)$, then $E_{MinNF}(X)$ is the minimal $\downarrow$-expression denoting X based on the primitive upper block partitioning of X, as described in Theorem 7.24(2).*

5. *If X contains at least one lower nick letter, then let $Z_1 {}_{\triangle}Z_2 {}_{\triangle} \ldots {}_{\triangle}Z_m$ for some $m \geq 2$ be the nick free decomposition of X. For $h = 1, \ldots, m$, let E_h be the operator-minimal $\uparrow$-expression denoting Z_h based on the primitive lower block partitioning of Z_h, as described in Theorem 7.42. $E_{MinNF}(X)$ is the minimal $\uparrow$-expression denoting X based on $E_1, \ldots, E_m$, as described in Theorem 7.46.*

6. *If X contains at least one upper nick letter, then let $Z_1 {}^{\triangledown}Z_2 {}^{\triangledown} \ldots {}^{\triangledown}Z_m$ for some $m \geq 2$ be the nick free decomposition of X. For $h = 1, \ldots, m$, let E_h be the operator-minimal $\downarrow$-expression denoting Z_h based on the primitive upper block partitioning of Z_h, analogous to the description in Theorem 7.42. $E_{MinNF}(X)$ is the minimal $\downarrow$-expression denoting X based on $E_1, \ldots, E_m$, analogous to the description in Theorem 7.46.*

Example 10.2 Consider the nick free formal DNA molecule X from Figure 7.5, for which $B_{\uparrow}(X) = 4$ and $B_{\downarrow}(X) = 3$. In Example 7.25, we have used Theorem 7.24 to construct two minimal DNA expressions denoting X. They were based on the lower block partitionings of X shown in Figure 7.3(a3) and (a4). By Case 3 of Definition 10.1, $E_{\mathrm{MinNF}}(X)$ is based on the primitive lower block partitioning $Y_0\overline{X}_1Y_1\overline{X}_2Y_2\overline{X}_3Y_3$ of X, which is depicted in Figure 7.3(a1).

A minimal DNA expression E_1 denoting the (primitive) lower block $\overline{X}_1$, for which $B_{\downarrow}(\overline{X}_1) = 1 > B_{\uparrow}(\overline{X}_1) = 0$, is constructed according to the description in Theorem 7.24(2). By Lemma 7.19, the only upper block partitioning of $\overline{X}_1$ is $\mathcal{P}_1 = \overline{X}_1$. Hence,

$$E_1 = \langle\downarrow \langle\updownarrow \alpha_2\rangle \alpha_3 \langle\updownarrow \alpha_4\rangle \alpha_5 \langle\updownarrow \alpha_6\rangle\rangle\,.$$

In the same way, we can construct the minimal DNA expressions E_2 and E_3 denoting the (primitive) lower blocks $\overline{X}_2$ and $\overline{X}_3$, respectively:

$$E_2 = \langle\downarrow \langle\updownarrow \alpha_8\rangle \alpha_9 \langle\updownarrow \alpha_{10}\rangle\rangle \quad \text{and}$$
$$E_3 = \langle\downarrow \langle\updownarrow \alpha_{14}\rangle \alpha_{15} \langle\updownarrow \alpha_{16}\rangle\rangle\,.$$

Consequently,

$$\begin{aligned} E_{\mathrm{MinNF}}(X) &= \langle\uparrow \alpha_1\ E_1\ \alpha_7\ E_2\ \alpha_{11} \langle\updownarrow \alpha_{12}\rangle \alpha_{13}\ E_3\ \alpha_{17} \langle\updownarrow \alpha_{18}\rangle\ \rangle \\ &= \langle\uparrow \alpha_1\ \langle\downarrow \langle\updownarrow \alpha_2\rangle \alpha_3 \langle\updownarrow \alpha_4\rangle \alpha_5 \langle\updownarrow \alpha_6\rangle\rangle\ \alpha_7\ \langle\downarrow \langle\updownarrow \alpha_8\rangle \alpha_9 \langle\updownarrow \alpha_{10}\rangle\rangle \\ &\qquad \alpha_{11} \langle\updownarrow \alpha_{12}\rangle \alpha_{13}\ \langle\downarrow \langle\updownarrow \alpha_{14}\rangle \alpha_{15} \langle\updownarrow \alpha_{16}\rangle\rangle\ \alpha_{17} \langle\updownarrow \alpha_{18}\rangle\ \rangle\,. \end{aligned} \tag{10.1}$$

■

Example 10.3 Consider the nick free formal DNA molecule X from Figure 7.6, for which $B_{\uparrow}(X) = B_{\downarrow}(X) = 2$ and whose first single-stranded component is an upper component. In Example 7.26 we have used Theorem 7.24 to construct four minimal DNA expressions denoting X, one for each upper block partitioning and each lower block partitioning of X. By Case 2a of Definition 10.1, $E_{\mathrm{MinNF}}(X)$ is the $\uparrow$-expression based on the primitive lower block partitioning of X. This partitioning is depicted in Figure 7.6(a). Hence,

$$\begin{aligned} E_{\mathrm{MinNF}}(X) &= E_a \\ &= \langle\uparrow \alpha_1 \ \langle\downarrow \langle\updownarrow \alpha_2\rangle \alpha_3 \langle\updownarrow \alpha_4\rangle\rangle \ \alpha_5 \langle\updownarrow \alpha_6\rangle \alpha_7 \ \langle\downarrow \langle\updownarrow \alpha_8\rangle \alpha_9 \langle\updownarrow \alpha_{10}\rangle\rangle\rangle\,. \end{aligned} \tag{10.2}$$

(see (7.9)). ■

Example 10.4 Consider the formal DNA molecule X from Figure 7.7, which contains four lower nick letters. The nick free decomposition of X is $Z_{1_\triangle} Z_{2_\triangle} Z_{3_\triangle} Z_{4_\triangle} Z_5$ for the submolecules $Z_1, \ldots, Z_5$ from (7.20).

In Example 7.47, we have constructed a minimal DNA expression E denoting X. We observed that each of the submolecules Z_1, Z_3, Z_4, Z_5 has exactly one lower block partitioning, which must be the *primitive* lower block partitioning then. For Z_2, there exist two lower block partitionings, each of which corresponds to an operator-minimal $\uparrow$-expression denoting Z_2. In the construction, we used the $\uparrow$-expression E_2'' which was based on the primitive lower block partitioning of Z_2.

Thus, each of the operator-minimal $\uparrow$-expressions used in the construction of E was based on the primitive lower block partitioning of the corresponding nick free submolecule. Consequently, $E_{\mathrm{MinNF}}(X)$ is equal to the minimal DNA expression E from (7.28):

$$\begin{aligned} E_{\mathrm{MinNF}}(X) = \langle\uparrow \quad & \alpha_1 \langle\downarrow \langle\updownarrow \alpha_2\rangle \alpha_3 \langle\updownarrow \alpha_4\rangle\rangle \\ & \langle\downarrow \langle\updownarrow \alpha_5\rangle \alpha_6 \langle\updownarrow \alpha_7\rangle\rangle \alpha_8 \langle\downarrow \langle\updownarrow \alpha_9\rangle \alpha_{10} \langle\updownarrow \alpha_{11}\rangle\rangle \\ & \langle\updownarrow \alpha_{12}\rangle \alpha_{13} \langle\updownarrow \alpha_{14}\rangle \alpha_{15} \langle\updownarrow \alpha_{16}\rangle \quad \langle\updownarrow \alpha_{17}\rangle \\ & \langle\downarrow \langle\updownarrow \alpha_{18}\rangle \alpha_{19} \langle\updownarrow \alpha_{20}\rangle\rangle \alpha_{21} \langle\updownarrow \alpha_{22}\rangle \quad \rangle\,. \end{aligned} \tag{10.3}$$

■

At several places, Definition 10.1 refers to the construction of (operator-)minimal DNA expressions, as described in Theorem 7.24 and Theorem 7.42. These constructions involve minimal DNA expressions E_j denoting lower blocks or upper blocks $\overline{X}_j$. In Definition 10.1, we do not consider the choice of these E_j's. Therefore, one may wonder if, for certain formal DNA molecules X, there might exist different minimal DNA expressions E_j denoting a particular lower block or upper block occurring in the construction of $E_{\mathrm{MinNF}}(X)$. If so, then $E_{\mathrm{MinNF}}(X)$ would not be uniquely determined, and the minimal normal form would not be well defined.

This situation does, however, not occur, just like it did not occur in the examples above. Because in the constructions from Theorem 7.24 and Theorem 7.42, we use *primitive* lower block partitionings (or *primitive* upper block partitionings), the lower blocks (upper blocks) $\overline{X}_j$ are *primitive* lower blocks (*primitive* upper blocks). Hence, by Lemma 8.18(1), the minimal DNA expressions E_j denoting the $\overline{X}_j$'s are unique.

We want the normal form DNA expressions to be minimal. By Theorem 7.5, if $X = \binom{\alpha_1}{c(\alpha_1)}$ for an $\mathcal{N}$-word α_1, then $E_{\mathrm{MinNF}}(X) = \langle\updownarrow \alpha_1\rangle$ is the only minimal DNA expression denoting X. For all other types of expressible formal DNA molecules, the minimality of $E_{\mathrm{MinNF}}(X)$ follows immediately from the definition.

We thus have

Lemma 10.5 *For each expressible formal DNA molecule X,*

1. *$E_{MinNF}(X)$ is well defined, and*
2. *$E_{MinNF}(X)$ is a minimal DNA expression denoting X.*

10.2 Characterization of the minimal normal form

For a given DNA expression E, we can decide if it is in minimal normal form by first determining its semantics $X = \mathcal{S}(E)$, then constructing the minimal normal form DNA expression E' denoting X according to Definition 10.1, and finally comparing E to E'. In this section, we will describe a more elegant way to achieve the same goal.

In Section 8.3, we proved that minimal DNA expressions can be characterized by six simple properties. This characterizations makes it easy to decide whether or not a given DNA expression is minimal. We now do something similar for DNA expressions in minimal normal form. We derive a characterization of these DNA expressions, consisting of five properties of (the arguments of) the operators occurring in them. Then in order to decide whether or not a DNA expression is in minimal normal form, we only have to check these properties.

We first prove that each DNA expression in minimal normal form has these properties. After that, we prove that each DNA expression with these five properties is indeed in minimal normal form.

Lemma 10.6 *Let E be a DNA expression in minimal normal form.*

($\mathcal{D}_{\mathbf{MinNF}}$.1) *Each occurrence of the operator $\updownarrow$ in E has as its argument an $\mathcal{N}$-word α (i.e., not a DNA expression).*

($\mathcal{D}_{\mathbf{MinNF}}$.2) *No occurrence of the operator $\uparrow$ in E has an $\uparrow$-argument, and no occurrence of the operator $\downarrow$ in E has a $\downarrow$-argument.*

($\mathcal{D}_{\mathbf{MinNF}}$.3) *Unless $E = \langle\uparrow \alpha\rangle$ or $E = \langle\downarrow \alpha\rangle$ for an $\mathcal{N}$-word α, each occurrence of an operator $\uparrow$ or $\downarrow$ in E has at least two arguments.*

($\mathcal{D}_{\mathbf{MinNF}}$.4) *For each inner occurrence of an operator $\uparrow$ or $\downarrow$ in E, the arguments are maximal $\mathcal{N}$-word occurrences α and $\updownarrow$-expressions $\langle\updownarrow \alpha\rangle$ for $\mathcal{N}$-words α, alternately.*

($\mathcal{D}_{\mathbf{MinNF}}$.5) *If the outermost operator of E is $\uparrow$ or $\downarrow$, then*

- *either its first argument is an $\mathcal{N}$-word α or an $\updownarrow$-expression $\langle\updownarrow \alpha\rangle$ for an $\mathcal{N}$-word α,*
- *or it has two consecutive expression-arguments.*

Note that Properties ($\mathcal{D}_{\text{MinNF}}$.1), ($\mathcal{D}_{\text{MinNF}}$.2) and ($\mathcal{D}_{\text{MinNF}}$.3) are equal to Properties ($\mathcal{D}_{\text{Min}}$.1), ($\mathcal{D}_{\text{Min}}$.2) and ($\mathcal{D}_{\text{Min}}$.3) of minimal DNA expressions in general.

Property ($\mathcal{D}_{\text{MinNF}}$.4) includes Properties ($\mathcal{D}_{\text{Min}}$.4) and ($\mathcal{D}_{\text{Min}}$.5). It is stronger, however, than these two properties together. As we will see in the proof, this is due to the choice for *primitive* lower block partitionings and *primitive* upper block partitionings in the definition of the minimal normal form.

Finally, Property ($\mathcal{D}_{\text{MinNF}}$.5) is a stronger version of Property ($\mathcal{D}_{\text{Min}}$.6). We will see in the proof that the difference between the two properties is caused by the second choice

Properties	E $X = \mathcal{S}(E)$ $E_{\mathrm{MinNF}}(X)$
$(\mathcal{D}_{\mathrm{MinNF}}.4)$	$\langle\downarrow \langle\uparrow \alpha_1 \langle\downarrow \langle\updownarrow \alpha_2\rangle \alpha_3 \langle\updownarrow \alpha_4\rangle\rangle \alpha_5 \langle\updownarrow \alpha_6\rangle\rangle \langle\updownarrow \alpha_7\rangle\rangle$ $\binom{\alpha_1}{-}\binom{\alpha_2}{c(\alpha_2)}\binom{-}{\alpha_3}\binom{\alpha_4}{c(\alpha_4)}\binom{\alpha_5}{-}\binom{\alpha_6}{c(\alpha_6)}^{\triangledown}\binom{\alpha_7}{c(\alpha_7)}$ $\langle\downarrow \langle\uparrow \alpha_1 \langle\updownarrow \alpha_2\rangle\rangle \alpha_3 \langle\uparrow \langle\updownarrow \alpha_4\rangle \alpha_5 \langle\updownarrow \alpha_6\rangle\rangle \langle\updownarrow \alpha_7\rangle\rangle$
$(\mathcal{D}_{\mathrm{MinNF}}.4)$	$\langle\uparrow \alpha_1 \langle\downarrow \langle\updownarrow \alpha_2\rangle \alpha_3 \langle\uparrow \langle\updownarrow \alpha_4\rangle \alpha_5 \langle\updownarrow \alpha_6\rangle\rangle \alpha_7\rangle\rangle$ $\binom{\alpha_1}{-}\binom{\alpha_2}{c(\alpha_2)}\binom{-}{\alpha_3}\binom{\alpha_4}{c(\alpha_4)}\binom{\alpha_5}{-}\binom{\alpha_6}{c(\alpha_6)}\binom{-}{\alpha_7}$ $\langle\uparrow \alpha_1 \langle\downarrow \langle\updownarrow \alpha_2\rangle \alpha_3 \langle\updownarrow \alpha_4\rangle\rangle \alpha_5 \langle\downarrow \langle\updownarrow \alpha_6\rangle \alpha_7\rangle\rangle$
$(\mathcal{D}_{\mathrm{MinNF}}.5)$	$\langle\downarrow \langle\uparrow \alpha_1 \langle\updownarrow \alpha_2\rangle\rangle \alpha_3\rangle$ $\binom{\alpha_1}{-}\binom{\alpha_2}{c(\alpha_2)}\binom{-}{\alpha_3}$ $\langle\uparrow \alpha_1 \langle\downarrow \langle\updownarrow \alpha_2\rangle \alpha_3\rangle\rangle$
$(\mathcal{D}_{\mathrm{MinNF}}.5)$	$\langle\downarrow \langle\uparrow \alpha_1 \langle\updownarrow \alpha_2\rangle\rangle \alpha_3 \langle\uparrow \langle\updownarrow \alpha_4\rangle \alpha_5 \langle\updownarrow \alpha_6\rangle\rangle \alpha_7\rangle$ $\binom{\alpha_1}{-}\binom{\alpha_2}{c(\alpha_2)}\binom{-}{\alpha_3}\binom{\alpha_4}{c(\alpha_4)}\binom{\alpha_5}{-}\binom{\alpha_6}{c(\alpha_6)}\binom{-}{\alpha_7}$ $\langle\uparrow \alpha_1 \langle\downarrow \langle\updownarrow \alpha_2\rangle \alpha_3 \langle\updownarrow \alpha_4\rangle\rangle \alpha_5 \langle\downarrow \langle\updownarrow \alpha_6\rangle \alpha_7\rangle\rangle$
$(\mathcal{D}_{\mathrm{MinNF}}.4)$, $(\mathcal{D}_{\mathrm{MinNF}}.5)$	$\langle\downarrow \langle\uparrow \alpha_1 \langle\downarrow \langle\updownarrow \alpha_2\rangle \alpha_3 \langle\updownarrow \alpha_4\rangle\rangle \alpha_5 \langle\updownarrow \alpha_6\rangle\rangle \alpha_7\rangle$ $\binom{\alpha_1}{-}\binom{\alpha_2}{c(\alpha_2)}\binom{-}{\alpha_3}\binom{\alpha_4}{c(\alpha_4)}\binom{\alpha_5}{-}\binom{\alpha_6}{c(\alpha_6)}\binom{-}{\alpha_7}$ $\langle\uparrow \alpha_1 \langle\downarrow \langle\updownarrow \alpha_2\rangle \alpha_3 \langle\updownarrow \alpha_4\rangle\rangle \alpha_5 \langle\downarrow \langle\updownarrow \alpha_6\rangle \alpha_7\rangle\rangle$
$(\mathcal{D}_{\mathrm{MinNF}}.4)$, $(\mathcal{D}_{\mathrm{MinNF}}.5)$	$\langle\downarrow \langle\uparrow \alpha_1 \langle\downarrow \langle\updownarrow \alpha_2\rangle \alpha_3 \langle\uparrow \langle\updownarrow \alpha_4\rangle \alpha_5 \langle\updownarrow \alpha_6\rangle\rangle \alpha_7 \langle\updownarrow \alpha_8\rangle\rangle \alpha_9 \langle\updownarrow \alpha_{10}\rangle\rangle \alpha_{11}\rangle$ $\binom{\alpha_1}{-}\binom{\alpha_2}{c(\alpha_2)}\binom{-}{\alpha_3}\binom{\alpha_4}{c(\alpha_4)}\binom{\alpha_5}{-}\binom{\alpha_6}{c(\alpha_6)}\binom{-}{\alpha_7}\binom{\alpha_8}{c(\alpha_8)}\binom{\alpha_9}{-}\binom{\alpha_{10}}{c(\alpha_{10})}\binom{-}{\alpha_{11}}$ $\langle\uparrow \alpha_1 \langle\downarrow \langle\updownarrow \alpha_2\rangle \alpha_3 \langle\updownarrow \alpha_4\rangle\rangle \alpha_5 \langle\downarrow \langle\updownarrow \alpha_6\rangle \alpha_7 \langle\updownarrow \alpha_8\rangle\rangle \alpha_9 \langle\downarrow \langle\updownarrow \alpha_{10}\rangle \alpha_{11}\rangle\rangle$

Table 10.1: Examples of minimal DNA expressions which do not have all properties from Lemma 10.6. The first column mentions the properties that are not valid. Each entry in the second column contains a corresponding DNA expression E, the formal DNA molecule X denoted by E, and the DNA expression in minimal normal form $E_{\mathrm{MinNF}}(X)$. As usual, the α_i's occurring represent (arbitrary) $\mathcal{N}$-words.
The DNA expressions in the third case are the ones from Example 7.2. The second, the fourth and the fifth case deal with the same formal DNA molecule, which is similar to the molecule from Example 7.26 (but slightly smaller). In fact, the DNA expressions E in these three cases resemble the minimal DNA expressions E_b, E_c and E_d from this example, respectively.

we make in the definition of the minimal normal form: if $B_{\uparrow}(X) = B_{\downarrow}(X) \geq 1$ for a nick free formal DNA molecule X, then the first single-stranded component of X determines whether $E_{\mathrm{MinNF}}(X)$ is an $\uparrow$-expression or a $\downarrow$-expression.

It is easily verified that the DNA expressions in minimal normal form from (10.1), (10.2) and (10.3) have all five properties. In Table 10.1, we give some examples of minimal DNA expressions which are not in minimal normal form. Such DNA expressions do have Properties $(\mathcal{D}_{\mathrm{MinNF}}.1)$–$(\mathcal{D}_{\mathrm{MinNF}}.3)$, simply because all minimal DNA expressions have these properties. However, they lack Properties $(\mathcal{D}_{\mathrm{MinNF}}.4)$ and/or $(\mathcal{D}_{\mathrm{MinNF}}.5)$. We also give the semantics of the DNA expressions, and the corresponding DNA expressions in minimal normal form.

Proof of Lemma 10.6: Let $X = \mathcal{S}(E)$, i.e., X is the formal DNA molecule for which $E = E_{\mathrm{MinNF}}(X)$.

By Lemma 10.5(2), E is minimal. Hence, Properties ($\mathcal{D}_{\mathrm{MinNF}}$.1), ($\mathcal{D}_{\mathrm{MinNF}}$.2) and ($\mathcal{D}_{\mathrm{MinNF}}$.3) are valid for E, simply because, by Lemma 8.22, they are valid for any minimal DNA expression. The other two properties require some specific considerations.

($\mathcal{D}_{\mathrm{MinNF}}$.4) If E is an $\updownarrow$-expression, then by Property ($\mathcal{D}_{\mathrm{MinNF}}$.1), $E = \langle\updownarrow \alpha\rangle$ for an $\mathcal{N}$-word α. In this case, E does not contain any occurrence of $\uparrow$ or $\downarrow$, which implies that E trivially has Property ($\mathcal{D}_{\mathrm{MinNF}}$.4).

Assume that E is an $\uparrow$-expression. Inner occurrences of $\uparrow$ and $\downarrow$ in E occur in the arguments of E.

If X is nick free, then either Case 2a or Case 3 of Definition 10.1 is applicable to $E = E_{\mathrm{MinNF}}(X)$. In both cases, E is based on the primitive lower block partitioning $\mathcal{P}$ of X, as described in Theorem 7.24(1).

By the construction from Theorem 7.24(1), the arguments of E are $\mathcal{N}$-words α_i, $\updownarrow$-expressions $\langle\updownarrow \alpha_i\rangle$ and minimal DNA expressions E_j denoting the lower blocks $\overline{X}_j$ occurring in $\mathcal{P}$. Obviously, $\mathcal{N}$-words α_i and $\updownarrow$-expressions $\langle\updownarrow \alpha_i\rangle$ for $\mathcal{N}$-words α_i do not contain occurrences of $\uparrow$ and $\downarrow$. Hence, the inner occurrences of $\uparrow$ and $\downarrow$ in E are the occurrences of these operators in the arguments E_j. As we argued before Lemma 10.5, the lower blocks $\overline{X}_j$ occurring in $\mathcal{P}$ are primitive lower blocks of X. Hence, by Lemma 8.18(1), each E_j is a $\downarrow$-expression, whose arguments are maximal $\mathcal{N}$-word occurrences α and $\updownarrow$-expressions $\langle\updownarrow \alpha\rangle$ for $\mathcal{N}$-words α, alternately. Clearly, the only occurrence of an operator $\uparrow$ or $\downarrow$ in E_j is its outermost operator $\downarrow$. Indeed, its arguments are maximal $\mathcal{N}$-word occurrences α and $\updownarrow$-expressions $\langle\updownarrow \alpha\rangle$ for $\mathcal{N}$-words α, alternately.

If X contains lower nick letters, then Case 5 of Definition 10.1 is applicable to $E = E_{\mathrm{MinNF}}(X)$. Let $Z_1{}_{\triangle}Z_2{}_{\triangle}\ldots{}_{\triangle}Z_m$ for some $m \geq 2$ be the nick free decomposition of X. E is based on operator-minimal $\uparrow$-expressions $E_1, \ldots, E_m$ denoting $Z_1, \ldots, Z_m$, respectively, as described in Theorem 7.46. The arguments of E are precisely the arguments of $E_1, \ldots, E_m$.

For $h = 1, \ldots, m$, the operator-minimal $\uparrow$-expression E_h is based on the primitive lower block partitioning of Z_h, as described in Theorem 7.42. Because the constructions from Theorem 7.24(1) and Theorem 7.42 are in fact identical, we can proceed in exactly the same way as in the case that X is nick free. We conclude that also now, the only occurrences of operators $\uparrow$ or $\downarrow$ in an argument of E are the outermost operators $\downarrow$ of the $\downarrow$-arguments of E, and that the arguments of such an occurrence of $\downarrow$ are $\mathcal{N}$-words α and $\updownarrow$-expressions $\langle\updownarrow \alpha\rangle$ for $\mathcal{N}$-words α, alternately.

Both if X is nick free and if it contains lower nick letters, we find that $E = E_{\mathrm{MinNF}}(X)$ has Property ($\mathcal{D}_{\mathrm{MinNF}}$.4).

The proof for the case that E is a $\downarrow$-expression is analogous.

($\mathcal{D}_{\mathrm{MinNF}}$.5) Assume that the outermost operator of E is $\uparrow$. Then in particular, by Theorem 7.5, X is not double-complete.

Assume further that E does not have two consecutive expression-arguments. By Property ($\mathcal{D}_{\mathrm{MinNF}}$.4) and Lemma 5.8, X is nick free. Hence, either Case 2a or Case 3 of Definition 10.1 is applicable to $E = E_{\mathrm{MinNF}}(X)$. In both cases, E is based on the primitive lower block partitioning of X, as described in Theorem 7.24(1).

If Case 2a is applicable, then by definition the first single-stranded component of X is an upper component. If, on the other hand, Case 3 is applicable, then $B_{\uparrow}(X) > B_{\downarrow}(X)$ and by Lemma 6.13(3), both the first single-stranded component and the last single-stranded component of X are upper components. In both cases, the first single-stranded component of X is an upper component.

By Lemma 7.11(3), the maximal upper prefix Y_0 of X is not empty, Hence, in the construction from Theorem 7.24(1), the first argument of E corresponds to the first component of Y_0, and thus is either an $\mathcal{N}$-word α or an $\updownarrow$-expression $\langle\updownarrow \alpha\rangle$ for an $\mathcal{N}$-word α (cf. the proof of Property ($\mathcal{D}_{\text{Min}}$.6) in Lemma 8.22).

We conclude that E has Property ($\mathcal{D}_{\text{MinNF}}$.5).

The proof for the case that the outermost operator of E is $\downarrow$ is analogous.

□

Let us use $\mathcal{D}_{\text{MinNF}}$ to denote the set of DNA expressions with Properties ($\mathcal{D}_{\text{MinNF}}$.1)–($\mathcal{D}_{\text{MinNF}}$.5).

Lemma 10.7 *Each DNA expression $E \in \mathcal{D}_{MinNF}$ is in minimal normal form.*

Proof: Let E be an arbitrary DNA expression in $\mathcal{D}_{\text{MinNF}}$, i.e., E has Properties ($\mathcal{D}_{\text{MinNF}}$.1)–($\mathcal{D}_{\text{MinNF}}$.5).

Properties ($\mathcal{D}_{\text{Min}}$.1), ($\mathcal{D}_{\text{Min}}$.2) and ($\mathcal{D}_{\text{Min}}$.3) from Lemma 8.22 are identical to Properties ($\mathcal{D}_{\text{MinNF}}$.1), ($\mathcal{D}_{\text{MinNF}}$.2) and ($\mathcal{D}_{\text{MinNF}}$.3), respectively. Both Property ($\mathcal{D}_{\text{Min}}$.4) and Property ($\mathcal{D}_{\text{Min}}$.5) follow immediately from Property ($\mathcal{D}_{\text{MinNF}}$.4), because they are weaker versions of this property. Finally, Property ($\mathcal{D}_{\text{Min}}$.6) follows immediately from Property ($\mathcal{D}_{\text{MinNF}}$.5). Thus, E is in $\mathcal{D}_{\text{Min}}$ and by Theorem 8.26, E is minimal.

Let $X = \mathcal{S}(E)$. We distinguish several cases.

1. If X is $\binom{\alpha_1}{c(\alpha_1)}$ for an $\mathcal{N}$-word α_1, then by Theorem 7.5, $E = \langle\updownarrow \alpha_1\rangle$. Indeed, $E = E_{\text{MinNF}}(X)$ (see Case 1 of Definition 10.1).

2. If X is nick free, contains at least one single-stranded component and $B_{\uparrow}(X) = B_{\downarrow}(X)$, then by Summary 8.16(2), E is either an $\uparrow$-expression based on a lower block partitioning of X as described in Theorem 7.24(1), or a $\downarrow$-expression based on an upper block partitioning of X as described in Theorem 7.24(2).

   We have to prove that the first single-stranded component of X determines if E is a an $\uparrow$-expression or a $\downarrow$-expression, and that the lower (or upper) block partitioning that E is based on, is indeed the primitive lower (upper, respectively) block partitioning of X.

   By Lemma 6.13(3), either the first single-stranded component of X is an upper component and the last single-stranded component of X is a lower component, or the other way round: the first single-stranded component of X is a lower component and the last single-stranded component of X is an upper component.

   (a) Assume that the first single-stranded component of X is an upper component (and hence, that the last single-stranded component of X is a lower component). First, we determine if E is an $\uparrow$-expression or a $\downarrow$-expression. Subsequently, we consider the partitioning of X that is used in the construction from Theorem 7.24.

• By Lemma 7.27, the arguments of E are $\mathcal{N}$-words and DNA expressions, alternately. Hence, by Property ($\mathcal{D}_{\text{MinNF}}$.5), the first argument of E is an $\mathcal{N}$-word α or an $\updownarrow$-expression $\langle\updownarrow\alpha\rangle$ for an $\mathcal{N}$-word α. In the latter case, by Property ($\mathcal{D}_{\text{MinNF}}$.3), E has at least two arguments, and the second argument of E is an $\mathcal{N}$-word α. In both cases, if E were a $\downarrow$-expression, then the first single-stranded component of X would be a lower component. Because the first single-stranded component of X is an upper component, E must be an $\uparrow$-expression.

• E satisfies the construction from Theorem 7.24(1). Let $\mathcal{P} = Y_0\overline{X}_1Y_1 \ldots \overline{X}_rY_r$ for some $r \geq 0$ be the lower block partitioning that E is based on. By construction and by Corollary 7.31(1), the $\downarrow$-arguments of E are precisely the minimal DNA expressions E_j denoting the lower blocks $\overline{X}_j$ occurring in $\mathcal{P}$.

Let E_j with $1 \leq j \leq r$ be an arbitrary $\downarrow$-argument of E. By Property ($\mathcal{D}_{\text{MinNF}}$.4), the arguments of E_j are $\mathcal{N}$-words α and $\updownarrow$-expressions $\langle\updownarrow\alpha\rangle$ for $\mathcal{N}$-words α. Hence, $\overline{X}_j = \mathcal{S}(E_j)$ consists only of lower components and double components.

By definition, the lower block $\overline{X}_j$ is an alternating sequence of primitive lower blocks and maximal upper sequences of X, which both starts and ends with a primitive lower block. By Lemma 7.12(4b), there cannot be any maximal upper sequence in this sequence, because $\overline{X}_j$ does not contain upper components. Hence, $\overline{X}_j$ starts and ends with the same primitive lower block. In other words, $\overline{X}_j$ is equal to a primitive lower block of X.

As the $\downarrow$-argument E_j was arbitrary, each lower block $\overline{X}_j$ occurring in $\mathcal{P}$ is equal to a primitive lower block of X. By the definition of a lower block partitioning, each primitive lower block of X is contained in one of the $\overline{X}_j$'s. Hence, the lower blocks $\overline{X}_j$ occurring in $\mathcal{P}$ are precisely all primitive lower blocks of X. This implies that $\mathcal{P}$ is the *primitive* lower block partitioning of X.

We conclude that $E = E_{\text{MinNF}}(X)$ (see Case 2a of Definition 10.1).

(b) Analogously, if we assume that the first single-stranded component of X is a lower component, then we find that E is a $\downarrow$-expression, which is based on the primitive upper block partitioning of X as described in Theorem 7.24(2). Hence, also in this case, $E = E_{\text{MinNF}}(X)$ (see Case 2b of Definition 10.1).

3. If X is nick free and $B_\uparrow(X) > B_\downarrow(X)$, then by Summary 8.16(3), E is an $\uparrow$-expression which is based on a lower block partitioning of X, as described in Theorem 7.24(1). Now, we can prove that this lower block partitioning actually is the *primitive* lower block partitioning of X, in the same way that we did in (the second part of) the proof for Case 2a.

   This implies that $E = E_{\text{MinNF}}(X)$ (see Case 3 of Definition 10.1).

4. The case that X is nick free and $B_\downarrow(X) > B_\uparrow(X)$ is analogous to the previous case. Hence, also in this case, $E = E_{\text{MinNF}}(X)$ (see Case 4 of Definition 10.1).

5. If X contains at least one lower nick letter, then let $Z_1{}_\triangle Z_2{}_\triangle \ldots {}_\triangle Z_m$ for some $m \geq 2$ be the nick free decomposition of X. By Summary 8.16(5), E is an $\uparrow$-expression which is based on operator-minimal $\uparrow$-expressions $E_1, \ldots, E_m$ denoting $Z_1, \ldots, Z_m$, respectively, as described in Theorem 7.46. For $h = 1, \ldots, m$, E_h is in turn based on a lower block partitioning $\mathcal{P}_h$ of Z_h, as described in Theorem 7.42.

Because the arguments of E are precisely the arguments of $E_1, \ldots, E_m$, and the constructions from Theorem 7.24(1) and Theorem 7.42 are in fact identical, we can proceed in the same way as in (the second part of) the proof for Case 2a: for $h = 1, \ldots, m$, the $\downarrow$-arguments of E_h correspond to the lower blocks occurring in $\mathcal{P}_h$. Because the arguments of these $\downarrow$-arguments are $\mathcal{N}$-words α and $\updownarrow$-expressions $\langle\updownarrow \alpha\rangle$ for $\mathcal{N}$-words α, the lower blocks occurring in $\mathcal{P}_h$ are precisely the primitive lower blocks of Z_h. Hence, for $h = 1, \ldots, m$, E_h is based on the *primitive* lower block partitioning of Z_h.

We conclude that $E = E_{\mathrm{MinNF}}(X)$ (see Case 5 of Definition 10.1).

6. The case that X contains at least one upper nick letter is analogous to the previous case. Hence, also in this case, we find that $E = E_{\mathrm{MinNF}}(X)$ (see Case 6 of Definition 10.1).

□

When we combine Lemma 10.6 and Lemma 10.7, we obtain

Theorem 10.8 *A DNA expression E is in minimal normal form if and only if $E \in \mathcal{D}_{MinNF}$.*

We can use the properties from Lemma 10.6 to prove other properties of normal form DNA expressions.

Lemma 10.9 *Let E be a DNA expression in minimal normal form.*

1. *If E is an $\uparrow$-expression, then E does not have any inner occurrence of $\uparrow$, and the only occurrences of $\downarrow$ in E are the operators governing $\downarrow$-arguments of E.*
2. *If E is a $\downarrow$-expression, then E does not have any inner occurrence of $\downarrow$, and the only occurrences of $\uparrow$ in E are the operators governing $\uparrow$-arguments of E.*

Proof:

1. Assume that E is an $\uparrow$-expression. Then by definition, each occurrence of $\downarrow$ in E is an inner occurrence.

   Suppose that $\uparrow_1$ is an inner occurrence of $\uparrow$ in E, and let E_1 be the DNA subexpression of E governed by $\uparrow_1$. By Properties ($\mathcal{D}_{\mathrm{MinNF}}$.1) and ($\mathcal{D}_{\mathrm{MinNF}}$.2), the parent operator of E_1 is not $\updownarrow$ or $\uparrow$. Hence, it must be $\downarrow$. This, however, contradicts Property ($\mathcal{D}_{\mathrm{MinNF}}$.4), as each occurrence of $\downarrow$ in E is an inner occurrence.

   Let $\downarrow_1$ be an arbitrary (inner) occurrence of $\downarrow$ in E, and let E_1 be the DNA subexpression of E governed by $\downarrow_1$. By Properties ($\mathcal{D}_{\mathrm{MinNF}}$.1) and ($\mathcal{D}_{\mathrm{MinNF}}$.2), the parent operator of E_1 is not $\updownarrow$ or $\downarrow$. Hence, it must be $\uparrow$. By the above, the only occurrence of $\uparrow$ in E is the outermost operator. Thus, E_1 is a $\downarrow$-argument of (the outermost operator of) E.

2. The proof of this claim is analogous to that of the previous claim.

□

As mentioned in the introduction to this chapter (on page 313), the nesting level of a DNA expression, and even of a minimal DNA expression, can get arbitrarily high. For DNA expressions in minimal normal form, however, the nesting level is bounded:

Lemma 10.10 *Let E be a DNA expression in minimal normal form. The maximal nesting level of E is at most 3.*

One of the objectives of a normal form was to reduce the complexity of a DNA expression (to a human reader). When we consider the maximal nesting level of a DNA expression as a measure for its complexity, the minimal normal form indeed achieves this objective.

Proof: If E only has $\mathcal{N}$-word-arguments, then by definition, the maximal nesting level of E is 1.

Now assume that E has at least one expression-argument. By Property ($\mathcal{D}_{\text{MinNF}}$.1), each $\updownarrow$-argument of E is equal to $\langle\updownarrow \alpha\rangle$ for an $\mathcal{N}$-word α. The maximal nesting level of such an argument is 1. Let E_1 be an arbitrary $\uparrow$-argument or $\downarrow$-argument of E. By Property ($\mathcal{D}_{\text{MinNF}}$.4), the only arguments of E_1 are maximal $\mathcal{N}$-word occurrences α and $\updownarrow$-expressions $\langle\updownarrow \alpha\rangle$ for $\mathcal{N}$-words α. In fact, by Property ($\mathcal{D}_{\text{MinNF}}$.3), at least one of these arguments is an $\updownarrow$-expression $\langle\updownarrow \alpha\rangle$. Then by Lemma 4.7(2), the maximal nesting level of E_1 is 2.

When we subsequently apply the same lemma to E itself, we conclude that its maximal nesting level is at most 3. □

Note that there also exist DNA expressions with maximal nesting level at most 3 that are not in minimal normal form.

Example 10.11 Let

$$\begin{aligned} E_1 &= \langle\updownarrow \langle\updownarrow \alpha_1\rangle\rangle\,, \\ E_2 &= \langle\downarrow \langle\updownarrow \alpha_1\rangle\, \alpha_2\, \langle\downarrow \langle\updownarrow \alpha_3\rangle\, \alpha_4\rangle\rangle\,, \\ E_3 &= \langle\uparrow \langle\downarrow \alpha_1\, \langle\updownarrow \alpha_2\rangle\rangle\, \alpha_3\, \langle\updownarrow \alpha_4\rangle\rangle \end{aligned}$$

for $\mathcal{N}$-words α_1, α_2, α_3 and α_4. E_1 has maximal nesting level 2, E_2 and E_3 have maximal nesting level 3. However, none of these DNA expressions is in minimal normal form. E_1 lacks Property ($\mathcal{D}_{\text{MinNF}}$.1), E_2 lacks Property ($\mathcal{D}_{\text{MinNF}}$.2), and E_3 lacks Property ($\mathcal{D}_{\text{MinNF}}$.5). E_1 and E_2 are not even minimal. ■

The property of the minimal normal form that is mainly responsible for the bounded nesting level, is Property ($\mathcal{D}_{\text{MinNF}}$.4). The following result on minimal DNA expressions makes this explicit:

Lemma 10.12 *Let E be a minimal DNA expression. Then E has maximal nesting level at most 3, if and only if E has Property ($\mathcal{D}_{MinNF}$.4).*

Proof: Because E is minimal, it has Properties ($\mathcal{D}_{\text{MinNF}}$.1)–($\mathcal{D}_{\text{MinNF}}$.3), simply because these properties are equal to Properties ($\mathcal{D}_{\text{Min}}$.1)–($\mathcal{D}_{\text{Min}}$.3) of minimal DNA expressions (see Lemma 8.22 and Lemma 10.6). E also has Properties ($\mathcal{D}_{\text{Min}}$.4)–($\mathcal{D}_{\text{Min}}$.6), but not necessarily Properties ($\mathcal{D}_{\text{MinNF}}$.4) and ($\mathcal{D}_{\text{MinNF}}$.5).

$\Longleftarrow$ If E has Property ($\mathcal{D}_{\text{MinNF}}$.4), then we can repeat the argumentation from the proof of Lemma 10.10 and conclude that the maximal nesting level of E is at most 3. Note that Property ($\mathcal{D}_{\text{MinNF}}$.5) is not used at all in the proof of Lemma 10.10. Therefore, it does not matter whether E has that property (and thus is in minimal normal form) or not.

$\Longrightarrow$ If E does not have Property ($\mathcal{D}_{\text{MinNF}}$.4), then there must be an inner occurrence of $\uparrow$ or $\downarrow$ in E whose arguments are not maximal $\mathcal{N}$-word occurrences α and $\updownarrow$-expressions

$\langle \updownarrow \alpha \rangle$, alternately. Without loss of generality, assume that this is an inner occurrence $\uparrow_1$ of $\uparrow$, and let E_1 be the DNA subexpression of E governed by $\uparrow_1$. By Property ($\mathcal{D}_{\text{Min}}$.4), $\uparrow_1$ is alternating, i.e., its arguments are maximal $\mathcal{N}$-word occurrences and DNA expressions, alternately. This implies that $\uparrow_1$ must have an expression-argument E_2 that is not of the form $\langle \updownarrow \alpha \rangle$. By Properties ($\mathcal{D}_{\text{Min}}$.1)and ($\mathcal{D}_{\text{Min}}$.2), E_2 must be a $\downarrow$-expression. By Lemma 8.27(6), E_2 has at least one argument $\langle \updownarrow \alpha \rangle$ for an $\mathcal{N}$-word α. But then we have the following situation:

$$E = \langle \ldots \langle \uparrow_1 \ldots \langle \downarrow \ldots \langle \updownarrow \alpha \rangle \ldots \rangle \ldots \rangle \ldots \rangle$$

In particular, the maximal nesting level of E is at least 4. □

10.3 The structure tree of a DNA expression in minimal normal form

Many properties of DNA expressions can be directly translated into properties of the corresponding structure trees, as defined in Section 4.6 (see Section 8.4). This is in particular true for DNA expressions in minimal normal form. Let t be the structure tree of a DNA expression E. We say that t is in minimal normal form, if and only if E is in minimal normal form. We then have

Theorem 10.8 (and Lemma 10.6) t is in minimal normal form if and only if

- **($\mathcal{D}_{\text{MinNF}}$.1)** each node labelled by $\updownarrow$ in t has a (single) child labelled by an $\mathcal{N}$-word α, and
- **($\mathcal{D}_{\text{MinNF}}$.2)** no node labelled by $\uparrow$ in t has a child labelled by $\uparrow$, and no node labelled by $\downarrow$ in t has a child labelled by $\downarrow$, and
- **($\mathcal{D}_{\text{MinNF}}$.3)** unless $E = \langle \uparrow \alpha \rangle$ or $E = \langle \downarrow \alpha \rangle$ for an $\mathcal{N}$-word α, each node labelled by $\uparrow$ or $\downarrow$ in t has at least two children, and
- **($\mathcal{D}_{\text{MinNF}}$.4)** for each non-root labelled by either $\uparrow$ or $\downarrow$ in t, the children are labelled by an $\mathcal{N}$-word α or by the operator $\updownarrow$, alternately, and
- **($\mathcal{D}_{\text{MinNF}}$.5)** if the root of t is labelled by either $\uparrow$ or $\downarrow$, then either its first child is labelled by an $\mathcal{N}$-word α or the operator $\updownarrow$, or it has two consecutive children labelled by an operator.

Lemma 10.9 If t is in minimal normal form, then

1. if the root of t is labelled by $\uparrow$, then t does not have any non-roots labelled by $\uparrow$, and the only nodes labelled by $\downarrow$ are children of the root;
2. if the root of t is labelled by $\downarrow$, then t does not have any non-roots labelled by $\downarrow$, and the only nodes labelled by $\uparrow$ are children of the root.

Lemma 10.10 and Lemma 4.30 If t is in minimal normal form, then the height of t is at most 4.

As we observed in Example 10.4, the minimal DNA expression from (7.28) is in minimal normal form. Hence, the corresponding structure tree, which we have shown in Figure 8.3(c), is also in minimal normal form. Indeed, it has all properties listed above.

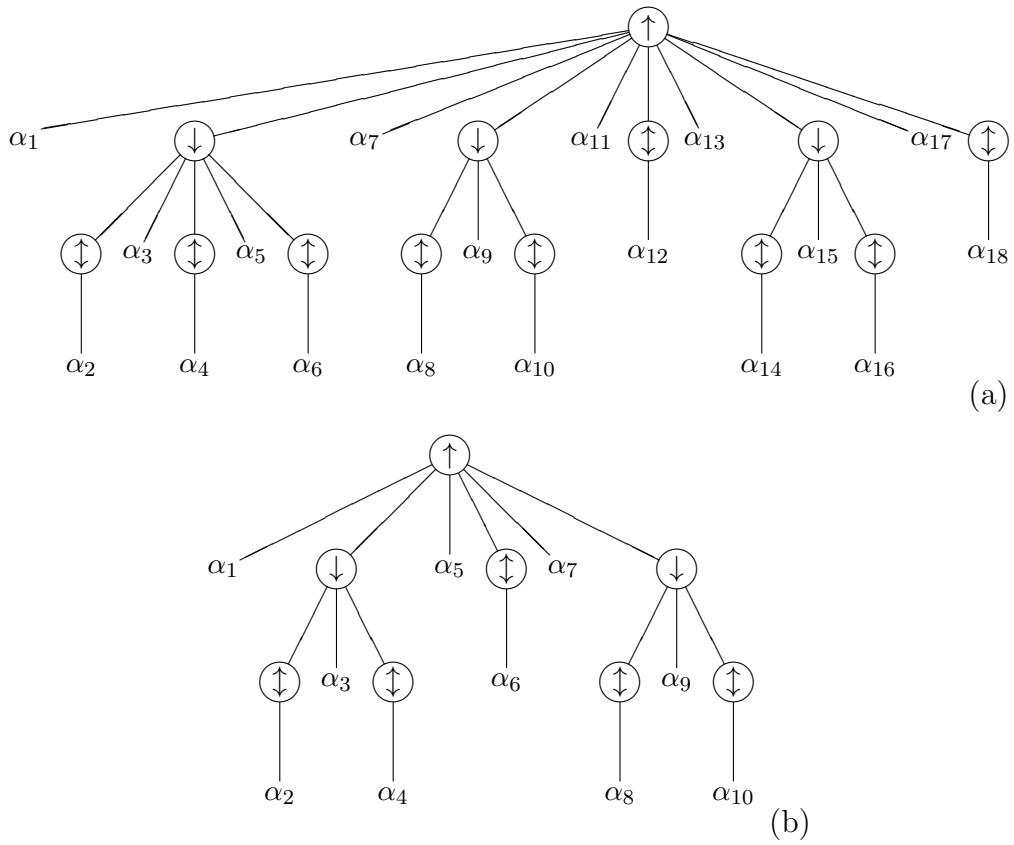


Figure 10.1: Two structure trees that are in minimal normal form. (a) The structure tree of the DNA expression $E_{\text{MinNF}}(X)$ from (10.1), denoting the nick free formal DNA molecule from (a.o.) Figure 7.5. (b) The structure tree of the DNA expression $E_{\text{MinNF}}(X) = E_a$ from (10.2), denoting the nick free formal DNA molecule from Figure 7.6.

On the other hand, although the DNA expression E from (7.7) is minimal, it is not in minimal normal form. Consequently, the corresponding structure tree in Figure 8.3(a) is not in minimal normal form, either. It does not satisfy (the tree-version of) Property ($\mathcal{D}_{\text{MinNF}}$.4). In the tree, the second child of the root, which is labelled by $\downarrow$, has a child which is labelled by $\uparrow$. Consequently, the height of the tree is greater than 4.

Likewise, the minimal DNA expression E_d from (7.12) and the corresponding structure tree in Figure 8.3(b) are not in minimal normal form. They violate both Property ($\mathcal{D}_{\text{MinNF}}$.4) and Property ($\mathcal{D}_{\text{MinNF}}$.5). Again, the height of the tree is greater than 4.

In Example 10.2 and Example 10.3, we have given the DNA expressions in minimal normal form for the last two cases. The corresponding structure trees are depicted in Figure 10.1.

10.4 Regularity of $\mathcal{D}_{\text{MinNF}}$

Neither the language $\mathcal{D}$ of all DNA expressions, nor the language $\mathcal{D}_{\text{Min}}$ containing only the minimal DNA expressions is regular (see Lemma 4.23 and Lemma 7.34). The proofs of these results were based on the fact that in a DNA expression, every opening bracket must be matched by a closing bracket, while there exist DNA expressions (even minimal

DNA expressions) with arbitrarily high nesting levels of the brackets.

Of course, in a DNA expression in minimal normal form, the opening brackets and the closing brackets must still match. However, by Lemma 10.10, the nesting level of the brackets in such a DNA expression is limited. We cannot get arbitrarily high nesting levels. This suggests that the language $\mathcal{D}_{\text{MinNF}}$ of DNA expressions in minimal normal form is regular, and that indeed turns out to be the case.

There are different ways to demonstrate this. One would be to give a right-linear grammar and to prove that it generates $\mathcal{D}_{\text{MinNF}}$. In this thesis, we follow another strategy. We describe a context-free grammar G_2 and prove that it generates $\mathcal{D}_{\text{MinNF}}$. This context-free grammar is not right-linear. However, as we will see, because the grammar is not self-embedding, the language generated by it (i.e., $\mathcal{D}_{\text{MinNF}}$) is regular, after all.

The new grammar G_2 is derived from the grammar G_1 that generates $\mathcal{D}$, the language of all DNA expressions (see Section 4.5). For example, like G_1, it has non-terminal symbols E, U and L (with some subscripts), which represent certain DNA expressions, sequences of arguments for the operator $\uparrow$ and sequences of arguments for the operator $\downarrow$, respectively.

However, due to the characteristic properties of the minimal normal form, Properties ($\mathcal{D}_{\text{MinNF}}$.1)–($\mathcal{D}_{\text{MinNF}}$.5), there exist important differences between the two grammars. Before we describe G_2 formally, we explain some of the differences. Most of these differences give rise to the use of different non-terminal symbols. Because of the symmetry between $\uparrow$-expressions and $\downarrow$-expressions, we sometimes restrict the explanation to $\uparrow$-expressions.

In our explanations, we often refer to the five properties of the minimal normal form. However, in order for a string to be a DNA expression in minimal normal form, it has to be a DNA expression in the first place. Therefore, we also sometimes refer to properties of DNA expressions in general. In particular, we refer to the fact that the arguments of the operator $\uparrow$ must fit together by upper strands.

- By Property ($\mathcal{D}_{\text{MinNF}}$.4), the arguments of an inner occurrence of $\uparrow$ or $\downarrow$ are $\mathcal{N}$-words α and $\updownarrow$-expressions $\langle\updownarrow \alpha\rangle$ for $\mathcal{N}$-words α, alternately. This is not necessarily true for the arguments of an outermost operator $\uparrow$ or $\downarrow$. Those arguments satisfy different (in particular, weaker) conditions.

  This difference is reflected by the notation we use for sequences of arguments of $\uparrow$ and $\downarrow$. For the outermost operator, we use U and L (with some subscript), respectively, as in G_1. For an inner occurrence, we introduce a new non-terminal symbol A (with some subscripts). We use this symbol both for inner occurrences of $\uparrow$ and for inner occurrences of $\downarrow$, because the arguments of these inner occurrences satisfy the same conditions.

- An outermost operator $\uparrow_0$ may have $\downarrow$-arguments. However, if the *first* argument is a $\downarrow$-argument, then by Property ($\mathcal{D}_{\text{MinNF}}$.5), $\uparrow_0$ must have two consecutive expression-arguments. Hence, in this case, the second and later arguments of $\uparrow_0$ have to satisfy an additional condition.

  We use the new non-terminal symbol $\widehat{U}$ (with some subscript) to denote a sequence of arguments of $\uparrow_0$ that must contain two consecutive expression-arguments.

  Note that a sequence of arguments represented by U (with some subscript) may also contain consecutive expression-arguments, but it does not have to.

- By the above, there is an essential difference between the sequence of *all* arguments of an outermost operator $\uparrow$, and a proper suffix of this sequence. A $\downarrow$-argument which

is the first of all arguments has other consequences for the rest of the sequence than a $\downarrow$-argument which is the first of a proper suffix.

Moreover, by Property ($\mathcal{D}_{\mathrm{MinNF}}$.3), the sequence of all arguments cannot be just one DNA expression, whereas a proper suffix of this sequence may be a DNA expression.

This difference is reflected by the subscript of the non-terminal symbol U. We use $U_\star$ to represent the sequence of all arguments of an outermost operator $\uparrow$, and we use U with other subscripts for proper suffices of this sequence.

- In the grammar G_1, a non-terminal symbol U (with some subscripts) represents an arbitrary suffix of the sequence of arguments of an operator $\uparrow$. This may be a proper suffix, but it may also be the entire sequence of arguments. The first subscript of U denotes whether or not one strand of this suffix of arguments must cover the other strand to the left.

  Now, consider a non-terminal symbol U or $\widehat{U}$ (with some subscript) in G_2, which is not equal to $U_\star$. By the above, this non-terminal symbol represents a *proper* suffix of the sequence of arguments of an outermost operator $\uparrow$, i.e., a subsequence of arguments which is preceded by at least one other argument.

  Because, by definition, the arguments of $\uparrow$ must fit together by upper strands, the upper strand of this subsequence of arguments must (always) cover the lower strand to the left. It is no use indicating this explicitly by means of a particular subscript.

- Consider again a non-terminal symbol U (with some subscripts) in the grammar G_1. The second subscript denotes whether or not one strand of the suffix of arguments represented by the symbol must cover the other strand to the right. This is useful for inner occurrences of $\uparrow$. If, for example, the $\uparrow$-expression is an argument of an operator $\downarrow$ and it is not the last argument, then the lower strand must cover the upper strand to the right.

  Now, consider any non-terminal symbol U or $\widehat{U}$ in G_2. As mentioned before, this symbol is used only to represent a suffix of the sequence of arguments of an *outermost* operator $\uparrow$. It does not matter if one strand of this suffix of arguments strictly covers the other strand to the right. There are no restrictions of the right-hand side of the strands, at all. Hence, we do not need a particular subscript to indicate such restrictions, either.

- In the grammar G_2, a non-terminal symbol E with a subscript $+$ represents a DNA expression that is the argument of an outermost operator $\uparrow$. By Property ($\mathcal{D}_{\mathrm{MinNF}}$.2), this DNA expression cannot be an $\uparrow$-expression. Hence, it can only be rewritten into either an $\updownarrow$-expression or a $\downarrow$-expression.

- In the grammar G_1, as soon as we introduce an operator $\uparrow$ or $\downarrow$, we give it a non-empty sequence of arguments, represented by a non-terminal symbol U or L (with some subscripts), respectively.

  Now, let E be a DNA expression in minimal normal form. If E contains inner occurrences of $\uparrow$ or $\downarrow$, then by Property ($\mathcal{D}_{\mathrm{MinNF}}$.3), each inner occurrence of $\uparrow$ or $\downarrow$ in E has at least two arguments. We do not introduce a special non-terminal symbol to represent "at least two arguments". Instead, as soon as we introduce an inner occurrence of $\uparrow$ or $\downarrow$, we give it one argument *plus* a non-empty sequence

of arguments. As we explained before, this non-empty sequence of arguments is represented by the non-terminal symbol A (with some subscripts).

- By Property ($\mathcal{D}_{\mathrm{MinNF}}$.4), for each inner occurrence of $\uparrow$ or $\downarrow$, the arguments are maximal $\mathcal{N}$-word occurrences α or $\updownarrow$-expressions $\langle\updownarrow \alpha\rangle$ for $\mathcal{N}$-words α, alternately. As we just explained, the non-terminal symbols A (with some subscripts) represent proper suffices of such sequences of arguments.

  To enforce the alternation of the arguments, we provide A with a subscript (in fact, its first subscript) α or $\updownarrow$. If it is α, then the suffix of the sequence of arguments must start with an $\mathcal{N}$-word α. If it is $\updownarrow$, then it must start with an $\updownarrow$-expression. The actual value of the subscript depends on the argument (an $\updownarrow$-expression or an $\mathcal{N}$-word) preceding the suffix.

  Of course, we might allow consecutive $\mathcal{N}$-word-arguments in the sequence of arguments, because they can be considered as a single $\mathcal{N}$-word. However, since we have to avoid consecutive $\updownarrow$-arguments, anyway, it is more elegant to also avoid consecutive $\mathcal{N}$-word-arguments.

  Moreover, allowing consecutive $\mathcal{N}$-word-arguments would introduce ambiguity in the grammar. If it were possible to have $\mathcal{N}$-words at consecutive positions in the sequence of arguments, then an $\mathcal{N}$-word-argument of length 2 or more could be derived in different ways: both from a single non-terminal symbol α and from two (or more) consecutive non-terminal symbols α.

- By Lemma 8.27(1b) and Property ($\mathcal{D}_{\mathrm{MinNF}}$.4), a proper $\downarrow$-subexpression is an argument of an outermost operator $\uparrow$. If it is not the last argument, then by Lemma 8.27(5b), the last argument of the $\downarrow$-subexpression is an $\updownarrow$-expression $\langle\updownarrow \alpha\rangle$ for an $\mathcal{N}$-word α. It cannot be an $\mathcal{N}$-word α.

  To represent such restrictions, we provide the non-terminal A with a second subscript, which is either $\star$ or $\updownarrow$. It it is $\star$, then the last argument in the sequence of arguments may be either an $\mathcal{N}$-word or an $\updownarrow$-expression. If it is $\updownarrow$, then the last argument must be an $\updownarrow$-expression.

- We do not only avoid consecutive $\mathcal{N}$-word-arguments for inner occurrences of $\uparrow$ or $\downarrow$. For consistency, we do the same for an outermost operator $\uparrow$ or $\downarrow$.

  For this, consider a non-terminal symbol U or $\widehat{U}$ which is not equal to $U_\star$. We provide this non-terminal, which represents a proper suffix of the sequence of arguments of an outermost operator $\uparrow$, with a subscript α or E. If the subscript is α, then the suffix must start with an $\mathcal{N}$-word α. If it is E, then the suffix must start with a DNA expression.[1]

Now, formally, G_2 is a 4-tuple $(V_2, \Sigma_2, P_2, S_2)$, where

$$\begin{aligned} V_2 &= \{E_\star, U_\star, U_\alpha, U_E, \widehat{U}_E, E_{+,+}, E_{+,\star}, L_\star, L_\alpha, L_E, \widehat{L}_E, E_{-,-}, E_{-,\star}, \\ &\qquad A_{\alpha,\updownarrow}, A_{\updownarrow,\updownarrow}, A_{\alpha,\star}, A_{\updownarrow,\star}, \alpha\} \\ \Sigma_2 &= \{\mathrm{A}, \mathrm{C}, \mathrm{G}, \mathrm{T}, \uparrow, \downarrow, \updownarrow, \langle\ ,\ \rangle\}, \\ S_2 &= E_\star \end{aligned}$$

[1] Actually, we have skipped the non-terminal symbol $\widehat{U}_\alpha$, because there would be only one production for this symbol: $\widehat{U}_\alpha \longrightarrow \alpha\widehat{U}_E$. We have substituted this production in the right-hand side of two of the productions for $U_\star$ and one production for $\widehat{U}_E$.

and P_2 is the following set of productions:

1. $E_\star \longrightarrow \langle \updownarrow \alpha\rangle \mid \langle \uparrow U_\star\rangle \mid \langle \downarrow L_\star\rangle$
2. $U_\star \longrightarrow \alpha \mid \alpha U_E \mid \langle \updownarrow \alpha\rangle U_\alpha \mid \langle \updownarrow \alpha\rangle U_E$
3. $U_\star \longrightarrow \left\langle \downarrow \alpha A_{\updownarrow,\updownarrow}\right\rangle \alpha \widehat{U}_E \mid \left\langle \downarrow \alpha A_{\updownarrow,\updownarrow}\right\rangle U_E \mid \left\langle \downarrow \langle \updownarrow \alpha\rangle A_{\alpha,\updownarrow}\right\rangle \alpha \widehat{U}_E \mid \left\langle \downarrow \langle \updownarrow \alpha\rangle A_{\alpha,\updownarrow}\right\rangle U_E$
4. $U_\alpha \longrightarrow \alpha \mid \alpha U_E$
5. $U_E \longrightarrow E_{+,\star} \mid E_{+,+}U_\alpha \mid E_{+,+}U_E$
6. $\widehat{U}_E \longrightarrow E_{+,+}\alpha \widehat{U}_E \mid E_{+,+}U_E$
7. $E_{+,+} \longrightarrow \langle \updownarrow \alpha\rangle \mid \left\langle \downarrow \langle \updownarrow \alpha\rangle A_{\alpha,\updownarrow}\right\rangle$
8. $E_{+,\star} \longrightarrow \langle \updownarrow \alpha\rangle \mid \langle \downarrow \langle \updownarrow \alpha\rangle A_{\alpha,\star}\rangle$
9. $L_\star \longrightarrow \alpha \mid \alpha L_E \mid \langle \updownarrow \alpha\rangle L_\alpha \mid \langle \updownarrow \alpha\rangle L_E$
10. $L_\star \longrightarrow \left\langle \uparrow \alpha A_{\updownarrow,\updownarrow}\right\rangle \alpha \widehat{L}_E \mid \left\langle \uparrow \alpha A_{\updownarrow,\updownarrow}\right\rangle L_E \mid \left\langle \uparrow \langle \updownarrow \alpha\rangle A_{\alpha,\updownarrow}\right\rangle \alpha \widehat{L}_E \mid \left\langle \uparrow \langle \updownarrow \alpha\rangle A_{\alpha,\updownarrow}\right\rangle L_E$
11. $L_\alpha \longrightarrow \alpha \mid \alpha L_E$
12. $L_E \longrightarrow E_{-,\star} \mid E_{-,-}L_\alpha \mid E_{-,-}L_E$
13. $\widehat{L}_E \longrightarrow E_{-,-}\alpha \widehat{L}_E \mid E_{-,-}L_E$
14. $E_{-,-} \longrightarrow \langle \updownarrow \alpha\rangle \mid \left\langle \uparrow \langle \updownarrow \alpha\rangle A_{\alpha,\updownarrow}\right\rangle$
15. $E_{-,\star} \longrightarrow \langle \updownarrow \alpha\rangle \mid \langle \uparrow \langle \updownarrow \alpha\rangle A_{\alpha,\star}\rangle$
16. $A_{\alpha,\updownarrow} \longrightarrow \alpha \langle \updownarrow \alpha\rangle \mid \alpha \langle \updownarrow \alpha\rangle A_{\alpha,\updownarrow}$
17. $A_{\updownarrow,\updownarrow} \longrightarrow \langle \updownarrow \alpha\rangle \mid \langle \updownarrow \alpha\rangle A_{\alpha,\updownarrow}$
18. $A_{\alpha,\star} \longrightarrow \alpha \mid \alpha A_{\updownarrow,\star}$
19. $A_{\updownarrow,\star} \longrightarrow \langle \updownarrow \alpha\rangle \mid \langle \updownarrow \alpha\rangle A_{\alpha,\star}$
20. $\alpha \longrightarrow \mathrm{A} \mid \mathrm{C} \mid \mathrm{G} \mid \mathrm{T} \mid \mathrm{A}\alpha \mid \mathrm{C}\alpha \mid \mathrm{G}\alpha \mid \mathrm{T}\alpha$

As is the case with most of the DNA expressions in this thesis, the DNA expressions we consider in this section are expressed in terms of general $\mathcal{N}$-words α_i, and not in terms of the actual $\mathcal{N}$-letters A, C, G and T. Therefore, in the examples below, we do not use the productions from line 20 of the list, for (rewriting) α. When we speak of a leftmost derivation in G_2, we mean that in every derivation step, we rewrite the leftmost non-terminal symbol unequal to α (with some subscript i). In other words, we treat α as a terminal symbol, ignoring the productions in line 20.

Example 10.13 Consider the nick free formal DNA molecule $X = \binom{\alpha_1}{-}\binom{\alpha_2}{c(\alpha_2)}\binom{-}{\alpha_3}$, for which $B_\uparrow(X) = B_\downarrow(X) = 1$ and the first single-stranded component is an upper component. By Case 2a of Definition 10.1,

$$E_{\mathrm{MinNF}}(X) = \langle \uparrow \alpha_1 \langle \downarrow \langle \updownarrow \alpha_2\rangle \alpha_3\rangle\rangle\,, \tag{10.4}$$

which is DNA expression E from Example 7.2 (see also Table 10.1). The following, leftmost derivation in G_2 yields $E_{\mathrm{MinNF}}(X)$:

$$\begin{aligned} E_\star &\overset{1,2}{\Longrightarrow} \langle \uparrow U_\star\rangle \\ &\overset{2,2}{\Longrightarrow} \langle \uparrow \alpha_1 U_E\rangle \end{aligned}$$

$$
\begin{aligned}
&\overset{5,1}{\Longrightarrow} && \langle\uparrow \alpha_1 E_{+,\star}\rangle \\
&\overset{8,2}{\Longrightarrow} && \langle\uparrow \alpha_1 \langle\downarrow \langle\updownarrow \alpha_2\rangle A_{\alpha,\star}\rangle\rangle \\
&\overset{18,1}{\Longrightarrow} && \langle\uparrow \alpha_1 \langle\downarrow \langle\updownarrow \alpha_2\rangle \alpha_3\rangle\rangle \\
&= && E_{\mathrm{MinNF}}(X).
\end{aligned}
$$

As in earlier derivations, numbers i, j above an arrow indicate that we have used production (i, j). ■

Example 10.14 Consider the minimal normal form DNA expression $E_{\mathrm{MinNF}}(X)$ from (10.1), which denotes the nick free formal DNA molecule X from (a.o.) Figure 7.3 and Figure 7.5. The following, leftmost derivation in G_2 yields $E_{\mathrm{MinNF}}(X)$:

$$
\begin{aligned}
E_\star &\overset{1,2}{\Longrightarrow} && \langle\uparrow U_\star\,\rangle \\
&\overset{2,2}{\Longrightarrow} && \langle\uparrow \alpha_1 U_E\,\rangle \\
&\overset{5,2}{\Longrightarrow} && \langle\uparrow \alpha_1 E_{+,+} U_\alpha\,\rangle \\
&\overset{7,2}{\Longrightarrow} && \langle\uparrow \alpha_1 \langle\downarrow \langle\updownarrow \alpha_2\rangle A_{\alpha,\updownarrow}\,\rangle U_\alpha\,\rangle \\
&\overset{16,2}{\Longrightarrow} && \langle\uparrow \alpha_1 \langle\downarrow \langle\updownarrow \alpha_2\rangle \alpha_3 \langle\updownarrow \alpha_4\rangle A_{\alpha,\updownarrow}\,\rangle U_\alpha\,\rangle \\
&\overset{16,1}{\Longrightarrow} && \langle\uparrow \alpha_1 \langle\downarrow \langle\updownarrow \alpha_2\rangle \alpha_3 \langle\updownarrow \alpha_4\rangle \alpha_5 \langle\updownarrow \alpha_6\rangle\rangle U_\alpha\,\rangle \\
&\overset{4,2}{\Longrightarrow} && \langle\uparrow \alpha_1 \langle\downarrow \langle\updownarrow \alpha_2\rangle \alpha_3 \langle\updownarrow \alpha_4\rangle \alpha_5 \langle\updownarrow \alpha_6\rangle\rangle \alpha_7 U_E\,\rangle \\
&\overset{5,2}{\Longrightarrow} && \ldots \overset{7,2}{\Longrightarrow} \ldots \overset{16,1}{\Longrightarrow} \ldots \overset{4,2}{\Longrightarrow} \ldots \\
&\overset{5,2}{\Longrightarrow} && \langle\uparrow \alpha_1 \langle\downarrow \langle\updownarrow \alpha_2\rangle \alpha_3 \langle\updownarrow \alpha_4\rangle \alpha_5 \langle\updownarrow \alpha_6\rangle\rangle \alpha_7 \langle\downarrow \langle\updownarrow \alpha_8\rangle \alpha_9 \langle\updownarrow \alpha_{10}\rangle\rangle \alpha_{11} E_{+,+} U_\alpha\,\rangle \\
&\overset{7,1}{\Longrightarrow} && \langle\uparrow \alpha_1 \langle\downarrow \langle\updownarrow \alpha_2\rangle \alpha_3 \langle\updownarrow \alpha_4\rangle \alpha_5 \langle\updownarrow \alpha_6\rangle\rangle \alpha_7 \langle\downarrow \langle\updownarrow \alpha_8\rangle \alpha_9 \langle\updownarrow \alpha_{10}\rangle\rangle \\
&&& \quad \alpha_{11} \langle\updownarrow \alpha_{12}\rangle U_\alpha\,\rangle \\
&\overset{4,2}{\Longrightarrow} && \ldots \overset{5,2}{\Longrightarrow} \ldots \overset{7,2}{\Longrightarrow} \ldots \overset{16,1}{\Longrightarrow} \ldots \\
&\overset{4,2}{\Longrightarrow} && \langle\uparrow \alpha_1 \langle\downarrow \langle\updownarrow \alpha_2\rangle \alpha_3 \langle\updownarrow \alpha_4\rangle \alpha_5 \langle\updownarrow \alpha_6\rangle\rangle \alpha_7 \langle\downarrow \langle\updownarrow \alpha_8\rangle \alpha_9 \langle\updownarrow \alpha_{10}\rangle\rangle \\
&&& \quad \alpha_{11} \langle\updownarrow \alpha_{12}\rangle \alpha_{13} \langle\downarrow \langle\updownarrow \alpha_{14}\rangle \alpha_{15} \langle\updownarrow \alpha_{16}\rangle\rangle \alpha_{17} U_E\,\rangle \\
&\overset{5,1}{\Longrightarrow} && \langle\uparrow \alpha_1 \langle\downarrow \langle\updownarrow \alpha_2\rangle \alpha_3 \langle\updownarrow \alpha_4\rangle \alpha_5 \langle\updownarrow \alpha_6\rangle\rangle \alpha_7 \langle\downarrow \langle\updownarrow \alpha_8\rangle \alpha_9 \langle\updownarrow \alpha_{10}\rangle\rangle \\
&&& \quad \alpha_{11} \langle\updownarrow \alpha_{12}\rangle \alpha_{13} \langle\downarrow \langle\updownarrow \alpha_{14}\rangle \alpha_{15} \langle\updownarrow \alpha_{16}\rangle\rangle \alpha_{17} E_{+,\star}\,\rangle \\
&\overset{8,1}{\Longrightarrow} && E_{\mathrm{MinNF}}(X).
\end{aligned}
$$

■

Example 10.15 Consider the minimal normal form DNA expression $E_{\mathrm{MinNF}}(X)$ from (10.3). This DNA expression denotes the formal DNA molecule X from Figure 7.7, which contains four lower nick letters. The following, leftmost derivation in G_2 yields $E_{\mathrm{MinNF}}(X)$:

$$
\begin{aligned}
E_\star &\overset{1,2}{\Longrightarrow} && \langle\uparrow U_\star\,\rangle \\
&\overset{2,2}{\Longrightarrow} && \langle\uparrow \alpha_1 U_E\,\rangle \\
&\overset{5,3}{\Longrightarrow} && \langle\uparrow \alpha_1 E_{+,+} U_E\,\rangle
\end{aligned}
$$

$$
\begin{array}{ll}
\overset{7,2}{\Longrightarrow} & \langle\uparrow \alpha_1 \langle\downarrow \langle\updownarrow \alpha_2\rangle A_{\alpha,\updownarrow} \rangle U_E \rangle \\
\overset{16,1}{\Longrightarrow} & \langle\uparrow \alpha_1 \langle\downarrow \langle\updownarrow \alpha_2\rangle \alpha_3 \langle\updownarrow \alpha_4\rangle\rangle U_E \rangle \\
\overset{5,2}{\Longrightarrow} & \langle\uparrow \alpha_1 \langle\downarrow \langle\updownarrow \alpha_2\rangle \alpha_3 \langle\updownarrow \alpha_4\rangle\rangle E_{+,+}U_\alpha \rangle \\
\overset{7,2}{\Longrightarrow} & \langle\uparrow \alpha_1 \langle\downarrow \langle\updownarrow \alpha_2\rangle \alpha_3 \langle\updownarrow \alpha_4\rangle\rangle \langle\downarrow \langle\updownarrow \alpha_5\rangle A_{\alpha,\updownarrow} \rangle U_\alpha \rangle \\
\overset{16,1}{\Longrightarrow} & \langle\uparrow \alpha_1 \langle\downarrow \langle\updownarrow \alpha_2\rangle \alpha_3 \langle\updownarrow \alpha_4\rangle\rangle \langle\downarrow \langle\updownarrow \alpha_5\rangle \alpha_6 \langle\updownarrow \alpha_7\rangle\rangle U_\alpha \rangle \\
\overset{4,2}{\Longrightarrow} & \langle\uparrow \alpha_1 \langle\downarrow \langle\updownarrow \alpha_2\rangle \alpha_3 \langle\updownarrow \alpha_4\rangle\rangle \langle\downarrow \langle\updownarrow \alpha_5\rangle \alpha_6 \langle\updownarrow \alpha_7\rangle\rangle \alpha_8 U_E \rangle \\
\overset{5,3}{\Longrightarrow} & \ldots \quad \overset{7,2}{\Longrightarrow} \quad \ldots \quad \overset{16,1}{\Longrightarrow} \quad \ldots \\
\overset{5,2}{\Longrightarrow} & \langle\uparrow \alpha_1 \langle\downarrow \langle\updownarrow \alpha_2\rangle \alpha_3 \langle\updownarrow \alpha_4\rangle\rangle \langle\downarrow \langle\updownarrow \alpha_5\rangle \alpha_6 \langle\updownarrow \alpha_7\rangle\rangle \alpha_8 \langle\downarrow \langle\updownarrow \alpha_9\rangle \alpha_{10} \langle\updownarrow \alpha_{11}\rangle\rangle \\
 & \quad E_{+,+}U_\alpha \rangle \\
\overset{7,1}{\Longrightarrow} & \langle\uparrow \alpha_1 \langle\downarrow \langle\updownarrow \alpha_2\rangle \alpha_3 \langle\updownarrow \alpha_4\rangle\rangle \langle\downarrow \langle\updownarrow \alpha_5\rangle \alpha_6 \langle\updownarrow \alpha_7\rangle\rangle \alpha_8 \langle\downarrow \langle\updownarrow \alpha_9\rangle \alpha_{10} \langle\updownarrow \alpha_{11}\rangle\rangle \\
 & \quad \langle\updownarrow \alpha_{12}\rangle U_\alpha \rangle \\
\overset{4,2}{\Longrightarrow} & \ldots \quad \overset{5,2}{\Longrightarrow} \quad \ldots \quad \overset{7,1}{\Longrightarrow} \quad \ldots \quad \overset{4,2}{\Longrightarrow} \quad \ldots \\
\overset{5,3}{\Longrightarrow} & \langle\uparrow \alpha_1 \langle\downarrow \langle\updownarrow \alpha_2\rangle \alpha_3 \langle\updownarrow \alpha_4\rangle\rangle \langle\downarrow \langle\updownarrow \alpha_5\rangle \alpha_6 \langle\updownarrow \alpha_7\rangle\rangle \alpha_8 \langle\downarrow \langle\updownarrow \alpha_9\rangle \alpha_{10} \langle\updownarrow \alpha_{11}\rangle\rangle \\
 & \quad \langle\updownarrow \alpha_{12}\rangle \alpha_{13} \langle\updownarrow \alpha_{14}\rangle \alpha_{15} E_{+,+}U_E \rangle \\
\overset{7,1}{\Longrightarrow} & \ldots \quad \overset{5,3}{\Longrightarrow} \quad \ldots \quad \overset{7,1}{\Longrightarrow} \quad \ldots \\
\overset{5,2}{\Longrightarrow} & \langle\uparrow \alpha_1 \langle\downarrow \langle\updownarrow \alpha_2\rangle \alpha_3 \langle\updownarrow \alpha_4\rangle\rangle \langle\downarrow \langle\updownarrow \alpha_5\rangle \alpha_6 \langle\updownarrow \alpha_7\rangle\rangle \alpha_8 \langle\downarrow \langle\updownarrow \alpha_9\rangle \alpha_{10} \langle\updownarrow \alpha_{11}\rangle\rangle \\
 & \quad \langle\updownarrow \alpha_{12}\rangle \alpha_{13} \langle\updownarrow \alpha_{14}\rangle \alpha_{15} \langle\updownarrow \alpha_{16}\rangle \langle\updownarrow \alpha_{17}\rangle E_{+,+}U_\alpha \rangle \\
\overset{7,2}{\Longrightarrow} & \ldots \quad \overset{16,1}{\Longrightarrow} \quad \ldots \\
\overset{4,2}{\Longrightarrow} & \langle\uparrow \alpha_1 \langle\downarrow \langle\updownarrow \alpha_2\rangle \alpha_3 \langle\updownarrow \alpha_4\rangle\rangle \langle\downarrow \langle\updownarrow \alpha_5\rangle \alpha_6 \langle\updownarrow \alpha_7\rangle\rangle \alpha_8 \langle\downarrow \langle\updownarrow \alpha_9\rangle \alpha_{10} \langle\updownarrow \alpha_{11}\rangle\rangle \\
 & \quad \langle\updownarrow \alpha_{12}\rangle \alpha_{13} \langle\updownarrow \alpha_{14}\rangle \alpha_{15} \langle\updownarrow \alpha_{16}\rangle \langle\updownarrow \alpha_{17}\rangle \langle\downarrow \langle\updownarrow \alpha_{18}\rangle \alpha_{19} \langle\updownarrow \alpha_{20}\rangle\rangle \alpha_{21} U_E \rangle \\
\overset{5,1}{\Longrightarrow} & \langle\uparrow \alpha_1 \langle\downarrow \langle\updownarrow \alpha_2\rangle \alpha_3 \langle\updownarrow \alpha_4\rangle\rangle \langle\downarrow \langle\updownarrow \alpha_5\rangle \alpha_6 \langle\updownarrow \alpha_7\rangle\rangle \alpha_8 \langle\downarrow \langle\updownarrow \alpha_9\rangle \alpha_{10} \langle\updownarrow \alpha_{11}\rangle\rangle \\
 & \quad \langle\updownarrow \alpha_{12}\rangle \alpha_{13} \langle\updownarrow \alpha_{14}\rangle \alpha_{15} \langle\updownarrow \alpha_{16}\rangle \langle\updownarrow \alpha_{17}\rangle \langle\downarrow \langle\updownarrow \alpha_{18}\rangle \alpha_{19} \langle\updownarrow \alpha_{20}\rangle\rangle \alpha_{21} E_{+,\star} \rangle \\
\overset{8,1}{\Longrightarrow} & E_{\mathrm{MinNF}}(X).
\end{array}
$$

■

Example 10.16 Consider the formal DNA molecule

$$X = \binom{\alpha_1}{-}\binom{\alpha_2}{c(\alpha_2)}\binom{-}{\alpha_3}\binom{\alpha_4}{c(\alpha_4)}^{\triangledown}\binom{\alpha_5}{c(\alpha_5)},$$

which contains one upper nick letter. The nick free decomposition of X is $Z_1{}^{\triangledown}Z_2$, where

$$Z_1 = \binom{\alpha_1}{-}\binom{\alpha_2}{c(\alpha_2)}\binom{-}{\alpha_3}\binom{\alpha_4}{c(\alpha_4)}$$

and $Z_2 = \binom{\alpha_5}{c(\alpha_5)}$. The primitive upper block partitioning of Z_1 is $Y_0\overline{X}_1Y_1$, where $Y_0 = \lambda$, $\overline{X}_1 = \binom{\alpha_1}{-}\binom{\alpha_2}{c(\alpha_2)}$ and $Y_1 = \binom{-}{\alpha_3}\binom{\alpha_4}{c(\alpha_4)}$. By (the analogue for $\downarrow$-expressions of) Theorem 7.42, we can use this primitive upper block partitioning to construct an operator-minimal $\downarrow$-expression E_1 denoting Z_1:

$$E_1 = \langle\downarrow \langle\uparrow \alpha_1 \langle\updownarrow \alpha_2\rangle\rangle \alpha_3 \langle\updownarrow \alpha_4\rangle\rangle .$$

It is not hard to see that the only operator-minimal $\downarrow$-expression denoting Z_2 is $E_2 = \langle\downarrow \langle\updownarrow \alpha_5\rangle\rangle$. By Case 6 of Definition 10.1, $E_{\mathrm{MinNF}}(X)$ is the $\downarrow$-expression based on E_1 and E_2 analogous to the description in Theorem 7.46:

$$E_{\mathrm{MinNF}}(X) = \langle\downarrow \langle\uparrow \alpha_1 \langle\updownarrow \alpha_2\rangle\rangle \alpha_3 \langle\updownarrow \alpha_4\rangle \langle\updownarrow \alpha_5\rangle\rangle\,.$$

The following, leftmost derivation in G_2 yields $E_{\mathrm{MinNF}}(X)$:

$$\begin{aligned}
E_\star &\overset{1,3}{\Longrightarrow} \langle\downarrow L_\star\rangle \\
&\overset{10,1}{\Longrightarrow} \left\langle\downarrow \langle\uparrow \alpha_1 A_{\updownarrow,\updownarrow}\rangle \alpha_3 \widehat{L}_E\right\rangle \\
&\overset{17,1}{\Longrightarrow} \left\langle\downarrow \langle\uparrow \alpha_1 \langle\updownarrow \alpha_2\rangle\rangle \alpha_3 \widehat{L}_E\right\rangle \\
&\overset{13,2}{\Longrightarrow} \langle\downarrow \langle\uparrow \alpha_1 \langle\updownarrow \alpha_2\rangle\rangle \alpha_3 E_{-,-} L_E\rangle \\
&\overset{14,1}{\Longrightarrow} \langle\downarrow \langle\uparrow \alpha_1 \langle\updownarrow \alpha_2\rangle\rangle \alpha_3 \langle\updownarrow \alpha_4\rangle\, L_E\rangle \\
&\overset{12,1}{\Longrightarrow} \langle\downarrow \langle\uparrow \alpha_1 \langle\updownarrow \alpha_2\rangle\rangle \alpha_3 \langle\updownarrow \alpha_4\rangle\, E_{-,\star}\rangle \\
&\overset{15,1}{\Longrightarrow} \langle\downarrow \langle\uparrow \alpha_1 \langle\updownarrow \alpha_2\rangle\rangle \alpha_3 \langle\updownarrow \alpha_4\rangle \langle\updownarrow \alpha_5\rangle\rangle \\
&= E_{\mathrm{MinNF}}(X).
\end{aligned}$$

■

The DNA expressions in minimal normal form from the four examples above can indeed be derived in the context-free grammar G_2. We now consider the situation in general. We prove that the language generated by G_2 is exactly the language $\mathcal{D}_{\mathrm{MinNF}}$ of DNA expressions in minimal normal form. Step by step, we analyse which languages can be derived from certain non-terminal symbols or after applying a certain production. Starting from 'low-level' non-terminal symbols, which generate a relatively simple language, we work up to (the productions for rewriting) the start symbol $E_\star$ of the grammar.

Some of the languages consist of sequences $\varepsilon_1 \ldots \varepsilon_n$ with $n \geq 1$, which have certain properties. In fact, these sequences consist of arguments of the operator $\uparrow$. To simplify the description of the languages, we introduce a notation for three (possible) properties of such sequences:

(LU.1) for $i = 1, \ldots, n$, ε_i is either a maximal $\mathcal{N}$-word occurrence α (in $\varepsilon_1 \ldots \varepsilon_n$), or an $\updownarrow$-expression $\langle\updownarrow \alpha\rangle$ for an $\mathcal{N}$-word α, or a $\downarrow$-expression with two or more arguments which form an alternating sequence of $\mathcal{N}$-words α and $\updownarrow$-expressions $\langle\updownarrow \alpha\rangle$.

(LU.2) $\varepsilon_1, \ldots, \varepsilon_n$ fit together by upper strands.

(LU.3) $L(\mathcal{S}^+(\varepsilon_1)) \in \mathcal{A}_\pm \cup \mathcal{A}_+$.

Lemma 10.17 *In the context-free grammar G_2,*

1. *$\mathcal{L}(\alpha)$ is the set of all $\mathcal{N}$-words.*
2. *$\mathcal{L}(\langle\updownarrow \alpha\rangle)$ is the set of all $\updownarrow$-expressions in minimal normal form.*
3. *$\mathcal{L}(A_{\alpha,\updownarrow})$ is the set of all (non-empty and finite) alternating sequences of $\mathcal{N}$-words α and $\updownarrow$-expressions $\langle\updownarrow \alpha\rangle$ for $\mathcal{N}$-words α, which start with an $\mathcal{N}$-word α and end with an $\updownarrow$-expression $\langle\updownarrow \alpha\rangle$.*

4. $\mathcal{L}(A_{\updownarrow,\updownarrow})$ *is the set of all (non-empty and finite) alternating sequences of* $\mathcal{N}$*-words* α *and* $\updownarrow$*-expressions* $\langle\updownarrow\alpha\rangle$ *for* $\mathcal{N}$*-words* α*, which start with an* $\updownarrow$*-expression* $\langle\updownarrow\alpha\rangle$ *and end with an* $\updownarrow$*-expression* $\langle\updownarrow\alpha\rangle$*.*

5. $\mathcal{L}(A_{\alpha,\star})$ *is the set of all (non-empty and finite) alternating sequences of* $\mathcal{N}$*-words* α *and* $\updownarrow$*-expressions* $\langle\updownarrow\alpha\rangle$ *for* $\mathcal{N}$*-words* α*, which start with an* $\mathcal{N}$*-word* α*.*

6. $\mathcal{L}(A_{\updownarrow,\star})$ *is the set of all (non-empty and finite) alternating sequences of* $\mathcal{N}$*-words* α *and* $\updownarrow$*-expressions* $\langle\updownarrow\alpha\rangle$ *for* $\mathcal{N}$*-words* α*, which start with an* $\updownarrow$*-expression* $\langle\updownarrow\alpha\rangle$*.*

7. $\mathcal{L}(\langle\downarrow\alpha A_{\updownarrow,\updownarrow}\rangle)$ *is the set of all* $\downarrow$*-expressions* $\langle\downarrow\varepsilon_{1,1}\ldots\varepsilon_{1,m}\rangle$ *with* $m\geq 2$*, such that* $\varepsilon_{1,1},\ldots,\varepsilon_{1,m}$ *form an alternating sequence of* $\mathcal{N}$*-words* α *and* $\updownarrow$*-expressions* $\langle\updownarrow\alpha\rangle$ *for* $\mathcal{N}$*-words* α*, which starts with an* $\mathcal{N}$*-word* $\varepsilon_{1,1}=\alpha$ *and ends with an* $\updownarrow$*-expression* $\varepsilon_{1,m}=\langle\updownarrow\alpha\rangle$*.*

8. $\mathcal{L}(\langle\downarrow\langle\updownarrow\alpha\rangle A_{\alpha,\updownarrow}\rangle)$ *is the set of all* $\downarrow$*-expressions* $\langle\downarrow\varepsilon_{1,1}\ldots\varepsilon_{1,m}\rangle$ *with* $m\geq 2$*, such that* $\varepsilon_{1,1},\ldots,\varepsilon_{1,m}$ *form an alternating sequence of* $\mathcal{N}$*-words* α *and* $\updownarrow$*-expressions* $\langle\updownarrow\alpha\rangle$ *for* $\mathcal{N}$*-words* α*, which starts with an* $\updownarrow$*-expression* $\varepsilon_{1,1}=\langle\updownarrow\alpha\rangle$ *and ends with an* $\updownarrow$*-expression* $\varepsilon_{1,m}=\langle\updownarrow\alpha\rangle$*.*

9. $\mathcal{L}(\langle\downarrow\langle\updownarrow\alpha\rangle A_{\alpha,\star}\rangle)$ *is the set of all* $\downarrow$*-expressions* $\langle\downarrow\varepsilon_{1,1}\ldots\varepsilon_{1,m}\rangle$ *with* $m\geq 2$*, such that* $\varepsilon_{1,1},\ldots,\varepsilon_{1,m}$ *form an alternating sequence of* $\mathcal{N}$*-words* α *and* $\updownarrow$*-expressions* $\langle\updownarrow\alpha\rangle$ *for* $\mathcal{N}$*-words* α*, which starts with an* $\updownarrow$*-expression* $\varepsilon_{1,1}=\langle\updownarrow\alpha\rangle$*.*

10. $\mathcal{L}(E_{+,+})$ *is the union of*

    - *the set of all* $\updownarrow$*-expressions* $\langle\updownarrow\alpha\rangle$ *for* $\mathcal{N}$*-words* α*, and*
    - *the set of all* $\downarrow$*-expressions* $\langle\downarrow\varepsilon_{1,1}\ldots\varepsilon_{1,m}\rangle$ *with* $m\geq 2$*, such that* $\varepsilon_{1,1},\ldots,\varepsilon_{1,m}$ *form an alternating sequence of* $\mathcal{N}$*-words* α *and* $\updownarrow$*-expressions* $\langle\updownarrow\alpha\rangle$ *for* $\mathcal{N}$*-words* α*, which starts with an* $\updownarrow$*-expression* $\varepsilon_{1,1}=\langle\updownarrow\alpha\rangle$ *and ends with an* $\updownarrow$*-expression* $\varepsilon_{1,m}=\langle\updownarrow\alpha\rangle$*.*

11. $\mathcal{L}(E_{+,\star})$ *is the union of*

    - *the set of all* $\updownarrow$*-expressions* $\langle\updownarrow\alpha\rangle$ *for* $\mathcal{N}$*-words* α*, and*
    - *the set of all* $\downarrow$*-expressions* $\langle\downarrow\varepsilon_{1,1}\ldots\varepsilon_{1,m}\rangle$ *with* $m\geq 2$*, such that* $\varepsilon_{1,1},\ldots,\varepsilon_{1,m}$ *form an alternating sequence of* $\mathcal{N}$*-words* α *and* $\updownarrow$*-expressions* $\langle\updownarrow\alpha\rangle$ *for* $\mathcal{N}$*-words* α*, which starts with an* $\updownarrow$*-expression* $\varepsilon_{1,1}=\langle\updownarrow\alpha\rangle$*.*

12. $\mathcal{L}(U_\alpha)$ *is the set of all sequences of arguments* $\varepsilon_1\ldots\varepsilon_n$ *with* $n\geq 1$ *and Properties (LU.1)–(LU.3), such that (in addition)*

    - ε_1 *is a maximal* $\mathcal{N}$*-word occurrence* α *(in* $\varepsilon_1\ldots\varepsilon_n$*).*

13. $\mathcal{L}(U_E)$ *is the set of all sequences of arguments* $\varepsilon_1\ldots\varepsilon_n$ *with* $n\geq 1$ *and Properties (LU.1)–(LU.3), such that (in addition)*

    - ε_1 *is a DNA expression.*

14. $\mathcal{L}(\widehat{U}_E)$ *is the set of all sequences of arguments* $\varepsilon_1\ldots\varepsilon_n$ *with* $n\geq 1$ *and Properties (LU.1)–(LU.3), such that (in addition)*

    - ε_1 *is a DNA expression, and*

- *there exists i with $1 \leq i \leq n-1$, such that both ε_i and ε_{i+1} are DNA expressions.*

15. *$\mathcal{L}(U_\star)$ is the set of all sequences of arguments $\varepsilon_1 \ldots \varepsilon_n$ with $n \geq 1$ and Properties (LU.1) and (LU.2), such that (in addition)*
    - *if $n = 1$, then ε_1 is an $\mathcal{N}$-word α, and*
    - *if ε_1 is a $\downarrow$-expression, then there exists i with $1 \leq i \leq n-1$, such that both ε_i and ε_{i+1} are DNA expressions.*
16. *$\mathcal{L}(\langle\uparrow U_\star\rangle)$ is the set of all $\uparrow$-expressions in minimal normal form.*
17. *$\mathcal{L}(\langle\downarrow L_\star\rangle)$ is the set of all $\downarrow$-expressions in minimal normal form.*

Note that the number m in Claims 8 and 10 is always odd, and thus at least 3, because the alternating sequence in the claims both starts and ends with an $\updownarrow$-expression $\langle\updownarrow \alpha\rangle$. Note also that the number n in Claim 14 is in fact at least 2, because there are two consecutive ε_i's which are DNA expressions.

Finally, it is worth noting that some of the languages described are (proper) subsets of other languages. We mention a few of the relations between the languages:

$$\mathcal{L}(A_{\alpha,\updownarrow}) \subset \mathcal{L}(A_{\alpha,\star})$$
$$\mathcal{L}(A_{\updownarrow,\updownarrow}) \subset \mathcal{L}(A_{\updownarrow,\star})$$
$$\mathcal{L}(E_{+,+}) \subset \mathcal{L}(E_{+,\star})$$
$$\mathcal{L}(\widehat{U}_E) \subset \mathcal{L}(U_E)$$
$$\mathcal{L}(U_\alpha) \subset \mathcal{L}(U_\star)$$

These relations fit in with the intuitive meanings of the non-terminal symbols involved. For example, the subscript $\star$ denotes the absence of a particular restriction.

Proof:

1 This claim follows immediately from the productions for (rewriting) α.

2 This claim follows immediately from the observation that the $\updownarrow$-expressions in minimal normal form are (exactly) all DNA expressions of the form $\langle\updownarrow \alpha\rangle$ for an $\mathcal{N}$-word α (see Definition 10.1).

3 This claim follows immediately from the productions for (rewriting) $A_{\alpha,\updownarrow}$.

4 This claim follows immediately from the productions for (rewriting) $A_{\updownarrow,\updownarrow}$ and the previous claim.

5, 6 These claims (simultaneously) follow immediately from the productions for (rewriting) $A_{\alpha,\star}$ and $A_{\updownarrow,\star}$.

7 This claim follows immediately from Claim 4 and the fact that the elements of an alternating sequence of $\mathcal{N}$-words α and $\updownarrow$-expressions $\langle\updownarrow \alpha\rangle$ fit together by lower strands (so that each element of $\mathcal{L}(\langle\downarrow \langle\updownarrow \alpha\rangle A_{\alpha,\updownarrow}\rangle)$ is indeed a DNA expression).

8 This claim follows immediately from Claim 3 and the fact that the elements of an alternating sequence of $\mathcal{N}$-words α and $\updownarrow$-expressions $\langle\updownarrow \alpha\rangle$ fit together by lower strands.

9 This claim follows immediately from Claim 5 and the fact that the elements of an alternating sequence of $\mathcal{N}$-words α and $\updownarrow$-expressions $\langle\updownarrow\alpha\rangle$ fit together by lower strands.

10 This claim follows immediately from the productions for (rewriting) $E_{+,+}$ and Claim 8.

11 This claim follows immediately from the productions for (rewriting) $E_{+,\star}$ and Claim 9.

12, 13 We first prove that each element of $\mathcal{L}(U_\alpha)$ or $\mathcal{L}(U_E)$ is a sequence as described in the respective claims. Let X be an arbitrary element of $\mathcal{L}(U_\alpha) \cup \mathcal{L}(U_E)$.

It follows immediately from the productions for (rewriting) U_α and U_E that X is a sequence $\varepsilon_1 \dots \varepsilon_n$ for some $n \geq 1$, such that for $i = 1, \dots, n$, ε_i is either an $\mathcal{N}$-word α, or an element of $\mathcal{L}(E_{+,+})$, or an element of $\mathcal{L}(E_{+,\star})$.

By Claims 10 and 11, each element of $\mathcal{L}(E_{+,+})$ or $\mathcal{L}(E_{+,\star})$ is either an $\updownarrow$-expression $\langle\updownarrow\alpha\rangle$ for an $\mathcal{N}$-word α, or a $\downarrow$-expression with two or more arguments which form an alternating sequence of $\mathcal{N}$-words α and $\updownarrow$-expressions $\langle\updownarrow\alpha\rangle$. This implies in particular that if $X \in \mathcal{L}(U_E)$, then ε_1 is a DNA expression (which is the additional property in Claim 13).

To complete Property (LU.1), we must establish that each $\mathcal{N}$-word ε_i is a maximal $\mathcal{N}$-word occurrence in X. For this, it is sufficient to show that no $\mathcal{N}$-word ε_i is succeeded by another $\mathcal{N}$-word. Therefore, assume that ε_i with $1 \leq i \leq n-1$ is an $\mathcal{N}$-word α. This $\mathcal{N}$-word has been introduced into X by the application of the production $U_\alpha \longrightarrow \alpha U_E$. Hence, $\varepsilon_i = \alpha$ is succeeded by an element of $\mathcal{L}(U_E)$, which starts with a DNA expression.

In particular, if $X \in \mathcal{L}(U_\alpha)$, then X starts with a maximal $\mathcal{N}$-word occurrence ε_1 (which is the additional property in Claim 12).

We proceed with Properties (LU.2) and (LU.3). By definition, for each $\mathcal{N}$-word α,

$$L(\mathcal{S}^+(\alpha)) = L(\left(\begin{smallmatrix}\alpha\\ _\end{smallmatrix}\right)) \in \mathcal{A}_+,\ R(\mathcal{S}^+(\alpha)) = R(\left(\begin{smallmatrix}\alpha\\ _\end{smallmatrix}\right)) \in \mathcal{A}_+,$$
$$L(\mathcal{S}(\langle\updownarrow\alpha\rangle)) = L(\left(\begin{smallmatrix}\alpha\\ c(\alpha)\end{smallmatrix}\right)) \in \mathcal{A}_\pm \text{ and } R(\mathcal{S}(\langle\updownarrow\alpha\rangle)) = R(\left(\begin{smallmatrix}\alpha\\ c(\alpha)\end{smallmatrix}\right)) \in \mathcal{A}_\pm.$$

Hence, if ε_i with $1 \leq i \leq n$ is a maximal $\mathcal{N}$-word occurrence α or an $\updownarrow$-expression $\langle\updownarrow\alpha\rangle$ for an $\mathcal{N}$-word α, then $L(\mathcal{S}^+(\varepsilon_i)), R(\mathcal{S}^+(\varepsilon_i)) \in \mathcal{A}_\pm \cup \mathcal{A}_+$.

We now consider a $\downarrow$-expression ε_i. If $\varepsilon_i \in \mathcal{L}(E_{+,+})$, then by Claim 10, the first argument of ε_i is an $\updownarrow$-expression $\langle\updownarrow\alpha_{i,1}\rangle$ for an $\mathcal{N}$-word $\alpha_{i,1}$, and the last argument of ε_i is an $\updownarrow$-expression $\langle\updownarrow\alpha_{i,m}\rangle$ for an $\mathcal{N}$-word $\alpha_{i,m}$. Now by Lemma 4.13(4), $L(\mathcal{S}(\varepsilon_i)) = L(\mathcal{S}(\langle\updownarrow\alpha_{i,1}\rangle)) \in \mathcal{A}_\pm$ and $R(\mathcal{S}(\varepsilon_i)) = R(\mathcal{S}(\langle\updownarrow\alpha_{i,m}\rangle)) \in \mathcal{A}_\pm$.

If, on the other hand, $\varepsilon_i \in \mathcal{L}(E_{+,\star})$, then we must have $i = n$. By Claim 11, the first argument of the $\downarrow$-expression ε_i is an $\updownarrow$-expression $\langle\updownarrow\alpha_{i,1}\rangle$ for an $\mathcal{N}$-word $\alpha_{i,1}$. Hence, $L(\mathcal{S}(\varepsilon_i)) = L(\mathcal{S}(\langle\updownarrow\alpha_{i,1}\rangle)) \in \mathcal{A}_\pm$.

We conclude that for $i = 1, \dots, n-1$, $L(\mathcal{S}^+(\varepsilon_i)), R(\mathcal{S}^+(\varepsilon_i)) \in \mathcal{A}_\pm \cup \mathcal{A}_+$ and that $L(\mathcal{S}^+(\varepsilon_n)) \in \mathcal{A}_\pm \cup \mathcal{A}_+$. This implies that X has Properties (LU.2) and (LU.3).

We also have to prove that each sequence $\varepsilon_1 \dots \varepsilon_n$ as described in the claims is an element of $\mathcal{L}(U_\alpha)$ or $\mathcal{L}(U_E)$, respectively. Let X be such a sequence.

We first analyse the $\downarrow$-expressions occurring in the sequence. Therefore, let ε_i with $1 \leq i \leq n$ be a $\downarrow$-expression. By Property (LU.1), ε_i has $m \geq 2$ arguments which form an alternating sequence of $\mathcal{N}$-words α and $\updownarrow$-expressions $\langle\updownarrow \alpha\rangle$ for $\mathcal{N}$-words α.

If the first argument of ε_i were an $\mathcal{N}$-word $\alpha_{i,1}$, then by Lemma 4.13(4), $L(\mathcal{S}(\varepsilon_i)) = L(\mathcal{S}^-(\alpha_{i,1})) \in \mathcal{A}_-$. This would contradict Property (LU.2) (if $i \geq 2$) or Property (LU.3) (if $i = 1$). Hence, the first argument of ε_i is an $\updownarrow$-expression $\langle\updownarrow \alpha_{i,1}\rangle$ for an $\mathcal{N}$-word $\alpha_{i,1}$.

If the last argument of ε_i is an $\mathcal{N}$-word $\alpha_{i,m}$, then $R(\mathcal{S}(\varepsilon_i)) = R(\mathcal{S}^-(\alpha_{i,m})) \in \mathcal{A}_-$. If $i \leq n-1$, then this would contradict Property (LU.2). Hence, in that case, the last argument of ε_i is an $\updownarrow$-expression $\langle\updownarrow \alpha_{i,m}\rangle$ for an $\mathcal{N}$-word $\alpha_{i,m}$.

Now, by Claim 10, if $1 \leq i \leq n-1$, then the $\downarrow$-expression ε_i is an element of $\mathcal{L}(E_{+,+})$. By Claim 11, if $i = n$, then ε_i is an element of $\mathcal{L}(E_{+,\star})$.

By Claim 10 and Claim 11, $\mathcal{L}(E_{+,+})$ and $\mathcal{L}(E_{+,\star})$ also contain all $\updownarrow$-expressions of the form $\langle\updownarrow \alpha\rangle$ for an $\mathcal{N}$-word α. We can thus conclude that if an element ε_i of the sequence X is a DNA expression (a $\downarrow$-expression or an $\updownarrow$-expression), then $\varepsilon_i \in \mathcal{L}(E_{+,+})$ if $1 \leq i \leq n-1$, and $\varepsilon_i \in \mathcal{L}(E_{+,\star})$ if $i = n$.

We finally observe that if an element ε_i of the sequence X is a maximal $\mathcal{N}$-word occurrence α, then either it is the last element of the sequence, or it is succeeded by a DNA expression. If, on the other hand, ε_i is a DNA expression, then either it is the last element of the sequence, or it is succeeded by a maximal $\mathcal{N}$-word occurrence or it is succeeded by another DNA expression. These possibilities can exactly be realized by the productions for U_α and U_E, respectively.

We can thus conclude that $X \in \mathcal{L}(U_\alpha)$ if ε_1 is an $\mathcal{N}$-word α and that $X \in \mathcal{L}(U_E)$ if ε_1 is a DNA expression.

14 Let X be an arbitrary element of $\mathcal{L}(\widehat{U}_E)$. We can prove that X is a sequence $\varepsilon_1 \ldots \varepsilon_n$ for some $n \geq 1$, which has Properties (LU.1)–(LU.3) and for which ε_1 is a DNA expression, like we did in the proof of Claims 12 and 13. Next, we observe that in the derivation of X from $\widehat{U}_E$, we must have applied the production $\widehat{U}_E \longrightarrow E_{+,+}U_E$ (exactly) once. By Claim 10, $E_{+,+}$ is rewritten into a DNA expression ε_i with $i \geq 1$, and by Claim 13, U_E is rewritten into a sequence $\varepsilon_{i+1} \ldots \varepsilon_n$ with $n \geq i+1$, for which ε_{i+1} is a DNA expression. Indeed, the sequence $\varepsilon_1 \ldots \varepsilon_n$ contains two consecutive elements ε_i and ε_{i+1} that are DNA expressions.

On the other hand, let $X = \varepsilon_1 \ldots \varepsilon_n$ be a sequence as described in the claim. We have to prove that $X \in \mathcal{L}(\widehat{U}_E)$. Also for this, we can start in the same way as in the proof of Claims 12 and 13. Thus, we find that if ε_i with $1 \leq i \leq n$ is a DNA expression, then $\varepsilon_i \in \mathcal{L}(E_{+,+})$ if $1 \leq i \leq n-1$, and $\varepsilon_i \in \mathcal{L}(E_{+,\star})$ if $i = n$.

In addition, let i_0 be the smallest value of i for which both ε_i and ε_{i+1} are DNA expressions. Then $\varepsilon_1, \ldots, \varepsilon_{i_0}$ are maximal $\mathcal{N}$-word occurrences and DNA expressions, alternately. Because, by assumption, both ε_1 and ε_{i_0} are DNA expressions. i_0 must be odd.

Now, when we start a derivation from $\widehat{U}_E$, first apply the production $\widehat{U}_E \longrightarrow E_{+,+}\alpha\widehat{U}_E$ $\frac{i_0-1}{2}$ times, and subsequently apply the production $\widehat{U}_E \longrightarrow E_{+,+}U_E$ once,

we obtain

$$\underbrace{E_{+,+}\alpha \ldots E_{+,+}\alpha}_{\frac{i_0-1}{2} \text{ times}} E_{+,+}U_E.$$

It follows from the foregoing that the $\frac{i_0-1}{2}$ pairs $E_{+,+}\alpha$ can be rewritten into $\varepsilon_1 \ldots \varepsilon_{i_0-1}$ and that the subsequent occurrence of $E_{+,+}$ can be rewritten into the DNA expression ε_{i_0}. Finally, by Claim 13, U_E can be rewritten into the sequence $\varepsilon_{i_0+1} \ldots \varepsilon_n$ which starts with the DNA expression ε_{i_0+1}.

15 We first prove that each element of $\mathcal{L}(U_\star)$ is a sequence $\varepsilon_1 \ldots \varepsilon_n$ with the properties from the claim. Therefore, let X be an arbitrary element of $\mathcal{L}(U_\star)$.

It follows immediately from the productions for (rewriting) $U_\star$ and Claims 7, 8, 12, 13 and 14, that X is a sequence $\varepsilon_1 \ldots \varepsilon_n$ with $n \geq 1$, which has Property (LU.1).

If ε_1 is an element of $\mathcal{L}(\langle\downarrow \alpha A_{\updownarrow,\updownarrow}\rangle) \cup \mathcal{L}(\langle\downarrow \langle\updownarrow \alpha\rangle A_{\alpha,\updownarrow}\rangle)$, then by Claims 7 and 8, ε_1 is a $\downarrow$-expression, whose last argument is an $\updownarrow$-expression $\langle\updownarrow \alpha\rangle$ for an $\mathcal{N}$-word α. Hence, by Lemma 4.13(4), $R(\mathcal{S}(\varepsilon_1)) = R(\mathcal{S}(\langle\updownarrow \alpha\rangle)) \in \mathcal{A}_\pm$. Now, Property (LU.2) follows from the productions for (rewriting) $U_\star$ and Claims 12, 13 and 14,

We finally consider the additional properties in the claim. If $n = 1$, then we must have applied the production $U_\star \longrightarrow \alpha$ in the first step of the derivation of X from $U_\star$. This implies that ε_1 is an $\mathcal{N}$-word α. If, on the other hand, ε_1 is a $\downarrow$-expression, then we must have applied one of the productions from line 3. It follows from these productions and Claims 13 and 14 that in that case, there exists i with $1 \leq i \leq n-1$, such that both ε_i and ε_{i+1} are DNA expressions.

We now prove that each sequence $\varepsilon_1 \ldots \varepsilon_n$ as described in the claim is an element of $\mathcal{L}(U_\star)$. Let X be such a sequence. We distinguish a number of cases, based on ε_1 and (possibly) subsequent ε_i's.

• If ε_1 is an $\mathcal{N}$-word α, then we may have $n = 1$. In that case, $X = \alpha$, which is derived from $U_\star$ by the application of production $U_\star \longrightarrow \alpha$.

If, on the other hand, $n \geq 2$, then the sequence $\varepsilon_2 \ldots \varepsilon_n$ has Properties (LU.1)–(LU.3) (with subscripts increased by 1), and ε_2 is a DNA expression. By Claim 13, the sequence $\varepsilon_2 \ldots \varepsilon_n$ is an element of $\mathcal{L}(U_E)$. Hence, $X = \alpha\varepsilon_2 \ldots \varepsilon_n \in \mathcal{L}(\alpha U_E)$.

• If ε_1 is an $\updownarrow$-expression $\langle\updownarrow \alpha\rangle$ for an $\mathcal{N}$-word α, then we must have $n \geq 2$. The sequence $\varepsilon_2 \ldots \varepsilon_n$ has Properties (LU.1)–(LU.3) (with subscripts increased by 1).

Now, if ε_2 is an $\mathcal{N}$-word α, then by Claim 12, the sequence $\varepsilon_2 \ldots \varepsilon_n$ is an element of $\mathcal{L}(U_\alpha)$ and $X = \langle\updownarrow \alpha\rangle\, \varepsilon_2 \ldots \varepsilon_n \in \mathcal{L}(\langle\updownarrow \alpha\rangle\, U_\alpha)$. If, on the other hand, ε_2 is a DNA expression, then by Claim 13, the sequence $\varepsilon_2 \ldots \varepsilon_n$ is an element of $\mathcal{L}(U_E)$ and $X = \langle\updownarrow \alpha\rangle\, \varepsilon_2 \ldots \varepsilon_n \in \mathcal{L}(\langle\updownarrow \alpha\rangle\, U_E)$.

• If ε_1 is a $\downarrow$-expression, then we must again have $n \geq 2$. The $\downarrow$-expression ε_1 has $m \geq 2$ arguments $\varepsilon_{1,1}, \ldots, \varepsilon_{1,m}$, which form an alternating sequence of $\mathcal{N}$-words α and $\updownarrow$-expressions $\langle\updownarrow \alpha\rangle$. Moreover, because ε_1 prefits ε_2 by upper strands, the last argument $\varepsilon_{1,m}$ of ε_1 must be an $\updownarrow$-expression.

Now, if the first argument $\varepsilon_{1,1}$ of ε_1 is an $\mathcal{N}$-word α, then by Claim 7, $\varepsilon_1 \in \mathcal{L}(\langle\downarrow \alpha A_{\updownarrow,\updownarrow}\rangle)$. If, on the other hand, $\varepsilon_{1,1}$ is an $\updownarrow$-expression, then by Claim 8, $\varepsilon_1 \in \mathcal{L}(\langle\downarrow \langle\updownarrow \alpha\rangle A_{\alpha,\updownarrow}\rangle)$.

In both cases, there exists i with $1 \leq i \leq n-1$ such that both ε_i and ε_{i+1} are DNA expressions.

If ε_2 is an $\mathcal{N}$-word α, then we do not have two consecutive DNA expressions, yet. Hence, n must be at least 3 (in fact, at least 4) and $\varepsilon_3 \dots \varepsilon_n$ is a sequence with Properties (LU.1)–(LU.3) (with subscripts increased by 2), such that (in addition) ε_3 is a DNA expression (because it succeeds the maximal $\mathcal{N}$-word occurrence ε_2) and there exists i with $3 \leq i \leq n-1$ for which both ε_i and ε_{i+1} are DNA expressions. By Claim 14, the sequence $\varepsilon_3 \dots \varepsilon_n$ is an element of $\mathcal{L}(\widehat{U}_E)$. Hence, either $X = \varepsilon_1\alpha\varepsilon_3 \dots \varepsilon_n \in \mathcal{L}(\langle\downarrow \alpha A_{\updownarrow,\updownarrow}\rangle\, \alpha\widehat{U}_E)$, or $X = \varepsilon_1\alpha\varepsilon_3 \dots \varepsilon_n \in \mathcal{L}(\langle\downarrow \langle\updownarrow \alpha\rangle\, A_{\alpha,\updownarrow}\rangle\, \alpha\widehat{U}_E)$.

If, on the other hand, ε_2 is a DNA expression, then ε_1 and ε_2 are two consecutive DNA expressions. There does not necessarily exist i with $2 \leq i \leq n-1$ for which both ε_i and ε_{i+1} are DNA expressions. The sequence $\varepsilon_2 \dots \varepsilon_n$ has Properties (LU.1)–(LU.3) and ε_2 is a DNA expression. By Claim 13, $\varepsilon_2 \dots \varepsilon_n$ is an element of $\mathcal{L}(U_E)$. Hence, either $X = \varepsilon_1\varepsilon_2 \dots \varepsilon_n \in \mathcal{L}(\langle\downarrow \alpha A_{\updownarrow,\updownarrow}\rangle\, U_E)$, or $X = \varepsilon_1\varepsilon_2 \dots \varepsilon_n \in \mathcal{L}(\langle\downarrow \langle\updownarrow \alpha\rangle\, A_{\alpha,\updownarrow}\rangle\, U_E)$.

16 Let X be an arbitrary element of $\mathcal{L}(\langle\uparrow U_\star\rangle)$. By the previous claim, $X = \langle\uparrow \varepsilon_1 \dots \varepsilon_n\rangle$ for $n \geq 1$ $\mathcal{N}$-words and DNA expressions $\varepsilon_1, \dots, \varepsilon_n$ with some special properties. By Property (LU.2), the arguments $\varepsilon_1, \dots, \varepsilon_n$ fit together by upper strands. Hence, X is indeed an $\uparrow$-expression. Each of Properties ($\mathcal{D}_{\text{MinNF}}$.1)–($\mathcal{D}_{\text{MinNF}}$.5) follows easily from the properties listed in the previous claim. By Theorem 10.8, X is in minimal normal form.

On the other hand, let X be an arbitrary $\uparrow$-expression in minimal normal form. Then $X = \langle\uparrow \varepsilon_1 \dots \varepsilon_n\rangle$ for $n \geq 1$ maximal $\mathcal{N}$-word occurrences and DNA expressions $\varepsilon_1, \dots, \varepsilon_n$ that fit together by upper strands, and X has Properties ($\mathcal{D}_{\text{MinNF}}$.1)–($\mathcal{D}_{\text{MinNF}}$.5). It is easily verified that the sequence $\varepsilon_1 \dots \varepsilon_n$ have the properties listed in the previous claim. Hence, $\varepsilon_1 \dots \varepsilon_n$ is an element of $\mathcal{L}(U_\star)$ and $X = \langle\uparrow \varepsilon_1 \dots \varepsilon_n\rangle \in \mathcal{L}(\langle\uparrow U_\star\rangle)$.

17 The proof of this claim is analogous to that of the previous claim (with analogous auxiliary claims).

□

In the first part of the proof of Claims 12 and 13, we showed that each element X of $\mathcal{L}(U_\alpha)$ or $\mathcal{L}(U_E)$ has Properties (LU.1)–(LU.3). It is worth noting that such an element X has even stronger properties. Also the 'lower analogues' of Properties (LU.2) and (LU.3) are valid: $\varepsilon_1, \dots, \varepsilon_n$ fit together by lower strands and $L(\mathcal{S}^-(\varepsilon_1)) \in \mathcal{A}_\pm \cup \mathcal{A}_-$.

The lower analogue of Property (LU.1), however, is not necessarily valid. (Some of) the elements ε_i of the sequences in $\mathcal{L}(U_\alpha)$ and $\mathcal{L}(U_E)$ may be $\downarrow$-expressions. Consequently, the languages $\mathcal{L}(L_\alpha)$ and $\mathcal{L}(L_E)$, which contain sequences $\varepsilon_1 \dots \varepsilon_n$ with the lower analogues of Properties (LU.1)–(LU.3), are really different from $\mathcal{L}(U_\alpha)$ and $\mathcal{L}(U_E)$, respectively.

A corollary of Lemma 10.17(2), (16) and (17) is

Theorem 10.18 *$\mathcal{L}(G_2) = \mathcal{L}_{G_2}(E_\star)$ is the language $\mathcal{D}_{MinNF}$ of all DNA expressions in minimal normal form.*

Because G_2 is a context-free grammar, we know that $\mathcal{D}_{\text{MinNF}}$ is a context-free language. We use Proposition 2.6 to prove that it is even a regular language. In order to apply

Level	Non-terminal symbols
1	$E_\star$
2	$U_\star, U_\alpha, U_E, \widehat{U}_E, L_\star, L_\alpha, L_E, \widehat{L}_E$
3	$E_{+,+}, E_{+,\star}, E_{-,-}, E_{-,\star}$
4	$A_{\alpha,\updownarrow}A_{\updownarrow,\updownarrow}, A_{\alpha,\star}, A_{\updownarrow,\star}$
5	α

Table 10.2: Intuitive levels of non-terminal symbols in the context-free grammar G_2. Note that the higher the level is, the 'simpler' the non-terminal symbols are.

Proposition 2.6 to G_2 directly, we have to establish that G_2 is not self-embedding, i.e., that none of its non-terminal symbols is self-embedding.

Note that it is not really surprising that this property is valid. Intuitively, we can distinguish 'levels' of non-terminal symbols in G_2, as indicated in Table 10.2. When we rewrite a non-terminal symbol, the result consists of terminal symbols, non-terminal symbols at a higher level and at most one non-terminal symbol at the same level, which then is the rightmost letter of the result. Hence, if a non-terminal symbol expands at its own level, then it does so 'in a right-linear way'.

The levels of the non-terminal symbols, as listed in Table 10.2, do not correspond perfectly to the nesting levels in the DNA expressions that can be derived in G_2. For example, the elements of $\{U_\star, U_\alpha, U_E, \widehat{U}_E, L_\star, L_\alpha, L_E, \widehat{L}_E\}$ and the elements of $\{E_{+,+}, E_{+,\star}, E_{-,-}, E_{-,\star}\}$ are at different levels in the table. However, as we discussed at the beginning of this section, each of these elements corresponds to a sequence of arguments or a single argument of the *outermost* operator, i.e., at nesting level 1 of the DNA expression. As another example, the symbol α is at level 5 in Table 10.2, whereas an $\mathcal{N}$-word α may occur at different levels in a DNA expression (in minimal normal form).

On the other hand, there definitely is a relation between the levels in the table and the nesting levels of a DNA expression. To see this, recall that a DNA subexpression induces a temporary increase of the nesting level of the DNA expression. Indeed, for each production $A \longrightarrow Z$ in P_2 that introduces a new DNA subexpression (i.e., for which the right-hand side Z contains a pair of matching brackets), the non-terminal symbols occurring inside the brackets of the DNA subexpression are at the higher level than the original non-terminal symbol A. Hence, the levels of the non-terminal symbols in G_2 'follow the direction' of the nesting levels of the DNA expression.

Our intuition about the levels of the non-terminal symbols is expressed formally in the following result:

Lemma 10.19 *Let A be an arbitrary non-terminal symbol in V_2 and let X be a string over $V_2 \cup \Sigma_2$ that can be derived from A in one or more derivation steps.*

1. *Assume that $A = \alpha$. If X contains a non-terminal symbol B, then $B = \alpha$ and B is the last letter of X.*

2. *Assume that $A \in \{A_{\alpha,\updownarrow}, A_{\updownarrow,\updownarrow}, A_{\alpha,\star}, A_{\updownarrow,\star}\}$. If X contains a non-terminal symbol B, then*

   - *either $B = \alpha$,*
   - *or $B \in \{A_{\alpha,\updownarrow}, A_{\updownarrow,\updownarrow}, A_{\alpha,\star}, A_{\updownarrow,\star}\}$ and B is the last letter of X.*

3. *Assume that* $A \in \{E_{+,+}, E_{+,\star}, E_{-,-}, E_{-,\star}\}$. *If* X *contains a non-terminal symbol* B, *then* $B \in \{A_{\alpha,\updownarrow}, A_{\updownarrow,\updownarrow}, A_{\alpha,\star}, A_{\updownarrow,\star}, \alpha\}$.

4. *Assume that* $A \in \{U_\star, U_\alpha, U_E, \widehat{U}_E\}$. *If* X *contains a non-terminal symbol* B, *then*

   - *either* $B \in \{E_{+,+}, E_{+,\star}, A_{\alpha,\updownarrow}, A_{\updownarrow,\updownarrow}, A_{\alpha,\star}, A_{\updownarrow,\star}, \alpha\}$,
   - *or* $B \in \{U_\star, U_\alpha, U_E, \widehat{U}_E\}$ *and* B *is the last letter of* X.

5. *Assume that* $A \in \{L_\star, L_\alpha, L_E, \widehat{L}_E\}$. *If* X *contains a non-terminal symbol* B, *then*

   - *either* $B \in \{E_{-,-}, E_{-,\star}, A_{\alpha,\updownarrow}, A_{\updownarrow,\updownarrow}, A_{\alpha,\star}, A_{\updownarrow,\star}, \alpha\}$,
   - *or* $B \in \{L_\star, L_\alpha, L_E, \widehat{L}_E\}$ *and* B *is the last letter of* X.

6. *Assume that* $A = E_\star$. *If* X *contains a non-terminal symbol* B, *then* $B \neq E_\star$.

Proof:

1. This claim follows immediately from the productions for (rewriting) α.
2. This claim follows immediately from the productions for (rewriting) the non-terminals in $\{A_{\alpha,\updownarrow}, A_{\updownarrow,\updownarrow}, A_{\alpha,\star}, A_{\updownarrow,\star}\}$ and the previous claim.
3. This claim follows immediately from the productions for (rewriting) the non-terminals in $\{E_{+,+}, E_{+,\star}, E_{-,-}, E_{-,\star}\}$ and from the previous two claims.
4. This claim follows immediately from the productions for (rewriting) the non-terminals in $\{U_\star, U_\alpha, U_E, \widehat{U}_E\}$ and from the previous three claims.
5. The proof of this claim is analogous to that of the previous claim.
6. This claim is obvious, because the non-terminal symbol $E_\star$ does not occur in the right-hand side of any production in G_2.

□

It follows immediately from Lemma 10.19 that none of the non-terminal symbols in G_2 is self-embedding, and thus that G_2 is not self-embedding. By Theorem 10.18, $\mathcal{L}(G_2)$ equals the language of all DNA expressions in minimal normal form. Hence, by Proposition 2.6,

Theorem 10.20 *The language* $\mathcal{D}_{MinNF}$ *of DNA expressions in minimal normal form is regular.*

Chapter 11

Algorithms for the Minimal Normal Form

At the beginning of Chapter 10, we introduced the (minimal) normal form, a.o., as a means to check equivalence. Two DNA expressions E_1 and E_2 are equivalent, if and only if their normal form versions are equal.

To utilize this property, we need an algorithm that, for a given DNA expression, computes the equivalent DNA expression in minimal normal form. With such an algorithm, we can compute the normal form versions of E_1 and E_2. If these are equal, then the original DNA expressions E_1 and E_2 are equivalent. If not, then E_1 and E_2 are not equivalent.

In order to obtain the normal form version of a given DNA expression E_1^*, we may first compute its semantics $X_1 = \mathcal{S}(E_1^*)$, and then use Definition 10.1 to construct $E_{\mathrm{MinNF}}(X_1)$. However, if we do this for E_1 and E_2, to decide if they are equivalent, then we make a useless detour. We can as well omit the second step, the construction of the DNA expression in minimal normal form from the semantics, and base our decision on $\mathcal{S}(E_1)$ and $\mathcal{S}(E_2)$ directly. Apart from that, of course, it would be more elegant if we did not need the semantics, at all, to get from one DNA expression (E_1^*) to another ($E_{\mathrm{MinNF}}(X_1)$).

In this chapter, we discuss two approaches to rewrite an arbitrary DNA expression E_1^* into its normal form equivalent, without referring to $\mathcal{S}(E_1^*)$. The first approach is inspired by the efficient recursive function `MakeMinimal`, which we used in Chapter 9 to rewrite a given DNA expression into an equivalent, minimal DNA expression. Unfortunately, the resulting recursive function for the minimal normal form turns out be be less efficient: it uses at least quadratic time in the worst case, whereas the complexity of `MakeMinimal` was linear. We subsequently describe an alternative, two-step algorithm, and prove that it is correct and uses only linear time and space.

Note that the recursive function `MakeMinimal` itself is not sufficient to produce some kind of a normal form. By Corollary 9.13, this function does not necessarily yield the same output for different equivalent inputs, which is required for a normal form. However, as we see in Section 11.2, it will be useful as the first step of the two-step algorithm.

11.1 Recursive algorithm for the minimal normal form

In Chapter 9, we have described a recursive function `MakeMinimal`, which rewrites a given DNA expression E into an equivalent, minimal DNA expression. For an expression-argument E_i of E, the function first performs a recursive call. If necessary, the result is subject to some local rearrangements, to make E minimal itself. We proved that, with

```
1.   MakeMinimalNF (E)
       // recursively rewrites an arbitrary DNA expression E
       // into an equivalent DNA expression in minimal normal form
2.   {
3.     if (E is an ↕-expression)
4.     then if (the argument of E is a DNA expression E1)
5.          then MakeMinimalNF (E1);
6.               substitute E by a DNA expression E' in minimal normal form
                   satisfying E' ≡ E;
7.          fi

8.     else    // E is an ↑-expression or a ↓-expression
9.          for all expression-arguments Ei of E (in some order)
10.         do   MakeMinimalNF (Ei);
11.         od
12.         substitute E by a DNA expression E' in minimal normal form
              satisfying E' ≡ E;
13.    fi
14.  }
```

Figure 11.1: Set-up of a recursive function `MakeMinimalNF`.

a proper data structure, this function requires time and space that are linear in $|E|$ (see Corollary 9.38 and Theorem 9.40).

We now want to rewrite a given DNA expression into the equivalent DNA expression in minimal normal form. Our first attempt is again a recursive function, which we call `MakeMinimalNF`. When we apply this function to a DNA expression E, we first (recursively) rewrite the expression-arguments of E into the minimal normal form. After that, we deal with the DNA expression as a whole. Just like we did in `MakeMinimal`, we consider $\updownarrow$-expressions on the one hand, and $\uparrow$-expressions and $\downarrow$-expressions on the other hand, separately. Figure 11.1 displays the global set-up of `MakeMinimalNF`.

In lines 6 and 12, we substitute a DNA expression E whose arguments are in minimal normal form by an equivalent DNA expression E' which is in minimal normal form itself. We have not specified how to find this DNA expression E'. It is, however, clear, that we should not implement those lines by a recursive call `MakeMinimalNF`(E), as that would start an infinite series of recursive calls of `MakeMinimalNF`, with the same argument E. Instead, analogous to our implementation of `MakeMinimal`, we should try to devise procedures consisting of local rearrangements at the string level, which make sure that a DNA expression with normal form arguments becomes in normal form itself.

Note that indeed, the structure of `MakeMinimalNF` is equal to that of `MakeMinimal`(see Figure 9.1). The main difference between the description of `MakeMinimal` and that of `MakeMinimalNF` is that the former has more detail. Both lines 6–10 and lines 16–37 of `MakeMinimal` are an implementation of the general statement *'substitute E by a minimal DNA expression E' satisfying $E' \equiv E$'* (cf. lines 6 and 12 of `MakeMinimalNF`).

Although we have not specified the details of lines 6 and 12, it is not difficult to prove that the set-up of `MakeMinimalNF` is correct.

Theorem 11.1 *Let E_1^* be an arbitrary DNA expression, and let E_2^* be the result of applying the function* `MakeMinimalNF` *to E_1^*.*

1. `MakeMinimalNF` *is well defined.*

2. The string E_2^* is a DNA expression in minimal normal form satisfying $E_2^* \equiv E_1^*$.

Proof:

1. Clearly, for every DNA expression E, there exists an equivalent DNA expression E' which is minimal normal form. This implies that lines 6 and 12 of `MakeMinimalNF` are well defined. Hence, the entire recursive function is well defined.

2. The proof of this claim is straightforward by induction on the number p of operators occurring in E_1^*.

   If $E_1^* = \langle\updownarrow \alpha_1\rangle$ for an $\mathcal{N}$-word α_1, then `MakeMinimalNF` leaves E_1^* unchanged. By Case 1 of Definition 10.1, $E_2^* = E_1^* = \langle\updownarrow \alpha_1\rangle = E_{\mathrm{MinNF}}(X)$ for $X = \binom{\alpha_1}{c(\alpha_1)}$. Indeed, E_2^* is in minimal normal form, and obviously, $E_2^* \equiv E_1^*$.

   In all other cases ($E_1^* = \langle\updownarrow E_1\rangle$ for a DNA expression E_1, or E_1^* is an $\uparrow$-expression or a $\downarrow$-expression), suppose that the recursive calls in lines 5 and 10 of `MakeMinimalNF` yield DNA expressions that are equivalent to the expression-arguments E_i of $E = E_1^*$. Then Lemma 5.11 and lines 6 and 12 of `MakeMinimalNF` ensure that E_2^* is in minimal normal form and equivalent to E_1^*. We leave the details to the reader.

□

In the above proof of Claim 2, we did not use the fact that the expression-arguments resulting from the recursive calls in lines 5 and 10 of `MakeMinimalNF` are in minimal normal form. This fact may, however, be exploited in an actual implementation of lines 6 and 12.

Regardless of the actual implementations of lines 6 and 12, we can draw another important conclusion: the recursive approach of `MakeMinimalNF` is not as efficient as that of `MakeMinimal`. We demonstrate this by examining its complexity for DNA expressions of a specific type.

Example 11.2 Let α be an arbitrary $\mathcal{N}$-word, and let

$$\begin{array}{ll} E_1 = \langle\downarrow \langle\updownarrow \alpha\rangle\, \alpha\, \langle\updownarrow \alpha\rangle\rangle\,, & \\ E_{2p} = \langle\uparrow \langle\updownarrow \alpha\rangle\, \alpha\ E_{2p-1}\ \alpha\, \langle\updownarrow \alpha\rangle\rangle & (p \geq 1), \\ E_{2p+1} = \langle\downarrow \langle\updownarrow \alpha\rangle\, \alpha\ E_{2p}\ \alpha\, \langle\updownarrow \alpha\rangle\rangle & (p \geq 1). \end{array}$$

Hence,

$$\begin{aligned} E_1 &= \langle\downarrow \langle\updownarrow \alpha\rangle\, \alpha\, \langle\updownarrow \alpha\rangle\rangle\,, \\ E_2 &= \langle\uparrow \langle\updownarrow \alpha\rangle\, \alpha\, \langle\downarrow \langle\updownarrow \alpha\rangle\, \alpha\, \langle\updownarrow \alpha\rangle\rangle\, \alpha\, \langle\updownarrow \alpha\rangle\rangle\,, \\ E_3 &= \langle\downarrow \langle\updownarrow \alpha\rangle\, \alpha\, \langle\uparrow \langle\updownarrow \alpha\rangle\, \alpha\, \langle\downarrow \langle\updownarrow \alpha\rangle\, \alpha\, \langle\updownarrow \alpha\rangle\rangle\, \alpha\, \langle\updownarrow \alpha\rangle\rangle\, \alpha\, \langle\updownarrow \alpha\rangle\rangle\,, \\ E_4 &= \langle\uparrow \langle\updownarrow \alpha\rangle\, \alpha\, \langle\downarrow \langle\updownarrow \alpha\rangle\, \alpha\, \langle\uparrow \langle\updownarrow \alpha\rangle\, \alpha\, \langle\downarrow \langle\updownarrow \alpha\rangle\, \alpha\, \langle\updownarrow \alpha\rangle\rangle\, \alpha\, \langle\updownarrow \alpha\rangle\rangle\, \alpha\, \langle\updownarrow \alpha\rangle\rangle\, \alpha\, \langle\updownarrow \alpha\rangle\rangle\,, \\ &\text{etc.} \end{aligned}$$

It is easy to prove by induction on p, that for any $p \geq 1$,

- both E_{2p} and E_{2p+1} are DNA expressions,

•

$$\begin{aligned}
\mathcal{S}(E_{2p}) &= \binom{\alpha}{c(\alpha)}\binom{\alpha}{-}\underbrace{\binom{\alpha}{c(\alpha)}\binom{-}{\alpha}\binom{\alpha}{c(\alpha)}\binom{\alpha}{-}\dots\binom{\alpha}{c(\alpha)}\binom{-}{\alpha}\binom{\alpha}{c(\alpha)}\binom{\alpha}{-}}_{p-1 \text{ times}}\cdot \\
&\quad \binom{\alpha}{c(\alpha)}\binom{-}{\alpha}\binom{\alpha}{c(\alpha)}\cdot \\
&\quad \underbrace{\binom{\alpha}{-}\binom{\alpha}{c(\alpha)}\binom{-}{\alpha}\binom{\alpha}{c(\alpha)}\dots\binom{\alpha}{-}\binom{\alpha}{c(\alpha)}\binom{-}{\alpha}\binom{\alpha}{c(\alpha)}}_{p-1 \text{ times}}\binom{\alpha}{-}\binom{\alpha}{c(\alpha)} \\
&= \binom{\alpha}{c(\alpha)}\binom{\alpha}{-}\underbrace{\binom{\alpha}{c(\alpha)}\binom{-}{\alpha}\binom{\alpha}{c(\alpha)}\binom{\alpha}{-}\dots\binom{\alpha}{c(\alpha)}\binom{-}{\alpha}\binom{\alpha}{c(\alpha)}\binom{\alpha}{-}}_{2p-1 \text{ times}}\binom{\alpha}{c(\alpha)}, \\
\mathcal{S}(E_{2p+1}) &= \underbrace{\binom{\alpha}{c(\alpha)}\binom{-}{\alpha}\binom{\alpha}{c(\alpha)}\binom{\alpha}{-}\dots\binom{\alpha}{c(\alpha)}\binom{-}{\alpha}\binom{\alpha}{c(\alpha)}\binom{\alpha}{-}}_{p \text{ times}}\cdot \\
&\quad \binom{\alpha}{c(\alpha)}\binom{-}{\alpha}\binom{\alpha}{c(\alpha)}\cdot \\
&\quad \underbrace{\binom{\alpha}{-}\binom{\alpha}{c(\alpha)}\binom{-}{\alpha}\binom{\alpha}{c(\alpha)}\dots\binom{\alpha}{-}\binom{\alpha}{c(\alpha)}\binom{-}{\alpha}\binom{\alpha}{c(\alpha)}}_{p \text{ times}} \\
&= \binom{\alpha}{c(\alpha)}\binom{-}{\alpha}\underbrace{\binom{\alpha}{c(\alpha)}\binom{\alpha}{-}\binom{\alpha}{c(\alpha)}\binom{-}{\alpha}\dots\binom{\alpha}{c(\alpha)}\binom{\alpha}{-}\binom{\alpha}{c(\alpha)}\binom{-}{\alpha}}_{2p \text{ times}}\binom{\alpha}{c(\alpha)},
\end{aligned}$$

•

$$\begin{aligned}
B_{\uparrow}(\mathcal{S}(E_{2p})) &= B_{\downarrow}(\mathcal{S}(E_{2p})) + 1 = 2p, \\
B_{\downarrow}(\mathcal{S}(E_{2p+1})) &= B_{\uparrow}(\mathcal{S}(E_{2p+1})) + 1 = 2p + 1,
\end{aligned}$$

- $n_{\updownarrow}(\mathcal{S}(E_q)) = 2q$, both if $q = 2p$ and if $q = 2p + 1$,
- $|E_q| = 3 \cdot 3q + (4q - 1) \cdot |\alpha|$, both if $q = 2p$ and if $q = 2p + 1$.

In particular, E_{2p} and E_{2p+1} are nick free, and their lengths are linear in p. Moreover, both E_{2p} and E_{2p+1} are minimal, because they achieve the minimal lengths mentioned in Summary 8.16(3) and (4), respectively. However, for $q \geq 3$, E_q is not in minimal normal form, because it violates Property ($\mathcal{D}_{\text{MinNF}}$.4).

By Definition 10.1(3) and (4) and the construction from Theorem 7.24, the corresponding DNA expressions in minimal normal form are

$$\begin{aligned}
E'_{2p} &= E_{\text{MinNF}}(\mathcal{S}(E_{2p})) \\
&= \left\langle \uparrow \langle\updownarrow \alpha\rangle\, \alpha\, \underbrace{\langle\downarrow \langle\updownarrow \alpha\rangle\, \alpha\, \langle\updownarrow \alpha\rangle\rangle\, \alpha \dots \langle\downarrow \langle\updownarrow \alpha\rangle\, \alpha\, \langle\updownarrow \alpha\rangle\rangle\, \alpha}_{2p-1 \text{ times}}\, \langle\updownarrow \alpha\rangle \right\rangle, \qquad (11.1) \\
E'_{2p+1} &= E_{\text{MinNF}}(\mathcal{S}(E_{2p+1})) \\
&= \left\langle \downarrow \langle\updownarrow \alpha\rangle\, \alpha\, \underbrace{\langle\uparrow \langle\updownarrow \alpha\rangle\, \alpha\, \langle\updownarrow \alpha\rangle\rangle\, \alpha \dots \langle\uparrow \langle\updownarrow \alpha\rangle\, \alpha\, \langle\updownarrow \alpha\rangle\rangle\, \alpha}_{2p \text{ times}}\, \langle\updownarrow \alpha\rangle \right\rangle. \qquad (11.2)
\end{aligned}$$

Now, let $p \geq 1$ and let us apply the function `MakeMinimalNF` to the $\downarrow$-expression E_{2p+1}, with the $\uparrow$-expression E_{2p} as one of its arguments. When we call the function recursively for E_{2p}, this argument is rewritten into the $\uparrow$-expression E'_{2p}. The other two expression-arguments $\langle \updownarrow \alpha \rangle$ of E_{2p+1} are already in minimal normal form. In order to rewrite the result

$$\big\langle \downarrow \langle \updownarrow \alpha \rangle \, \alpha \; E'_{2p} \; \alpha \, \langle \updownarrow \alpha \rangle \big\rangle \tag{11.3}$$

into the corresponding DNA expression in minimal normal form E'_{2p+1}, we must remove the $2p-1$ occurrences of $\downarrow$ in E'_{2p}, add $2p-1$ occurrences of $\uparrow$ at other positions in the DNA expression, and also rearrange the brackets. Regardless of the actual implementation of such a rearrangement, it requires time that is at least linear in p.

Likewise, at a higher level of the recursion, we have had to rearrange $2p-2, 2p-3, 2p-4, \ldots, 1$ occurrences of operators in $E'_{2p-1}, E'_{2p-2}, E'_{2p-3}, \ldots, E'_2$, respectively. Altogether, this takes time that is at least quadratic in p, and thus in the length of E_{2p+1}.

The analysis for the $\uparrow$-expression E_{2p} is completely analogous. ■

It is instructive to examine the operation of the recursive function `MakeMinimalNF` on the structure trees of the DNA expressions from the above example. We have depicted this in Figure 11.2 and Figure 11.3 for the $\downarrow$-expression E_5.

Since there exist DNA expressions E for which `MakeMinimalNF` requires time that is at least quadratic in $|E|$, we can conclude:

Theorem 11.3 *The worst case time complexity of the recursive function* `MakeMinimalNF` *is at least quadratic.*

Alternative implementation

We need to mention that Theorem 11.3 was actually based on an implicit, but natural assumption about the way that line 12 of `MakeMinimalNF` may be implemented. In particular, in Example 11.2, when we observe that the DNA expression from (11.3) must be rewritten into E'_{2p+1}, we assume that the requisite rewriting steps are really carried out, in the current version of the DNA expression E.

There is, however, an alternative implementation possible, in which `MakeMinimalNF` maintains two DNA expressions E' and $\widehat{E'}$ instead of just one. E' and $\widehat{E'}$ are operator-minimal DNA expressions, as defined in Section 7.2, or are based on such DNA expressions. In the case of a nick free DNA molecule X, one of these two DNA expressions is the operator-minimal $\uparrow$-expression based on the primitive lower block partitioning of X, and the other is the operator-minimal $\downarrow$-expression based on the primitive upper block partitioning of X. Only one of these two DNA expressions, say E', is in minimal normal form.

Let E be an $\uparrow$-expression or $\downarrow$-expression. For each expression-argument E_i of E, the recursive call in line 10 of `MakeMinimalNF` should produce an operator-minimal $\uparrow$-expression and an operator-minimal $\downarrow$-expression, which are both equivalent to E_i. Now, suppose that in line 12, we discover that E', the DNA expression in minimal normal form satisfying $E' \equiv E$, should be an $\uparrow$-expression. Then we can efficiently construct E' from its operator-minimal $\uparrow$-arguments. In addition, we construct an operator-minimal $\downarrow$-expression $\widehat{E'}$ satisfying $\widehat{E'} \equiv E$ from the operator-minimal $\downarrow$-arguments.

Example 11.4 Consider the $\downarrow$-expression $E = E_{2p+1}$ for some $p \geq 1$, with semantics $X = \mathcal{S}(E_{2p+1})$, as described in Example 11.2. The recursive call of `MakeMinimalNF` for argument

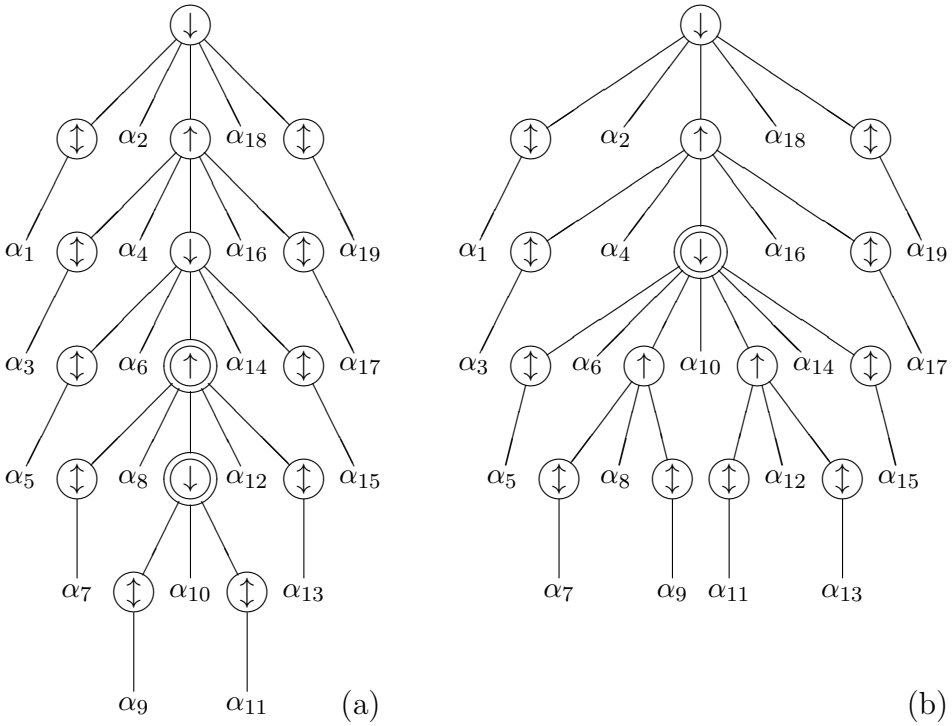


Figure 11.2: Structure trees of the DNA expressions that we successively obtain, when we apply the recursive function `MakeMinimalNF` to the $\downarrow$-expression E_5 from Example 11.2. To make the structure trees easier to compare, we have added subscripts to the occurring $\mathcal{N}$-words. (a) Structure tree of the original DNA expression. The nodes in the backbone of the tree correspond in top-down order to E_5, E_4, E_3, E_2 and E_1, respectively. Note that E_1 and E_2 are already in the minimal normal form. The corresponding two nodes are marked with an extra circle. (b) Structure tree after rewriting the DNA subexpression E_3 into the minimal normal form equivalent E'_3. The node corresponding to E'_3 is marked with an extra circle. (Continued in Figure 11.3)

E_{2p} of E_{2p+1} yields two operator-minimal DNA expressions denoting $X_{2p} = \mathcal{S}(E_{2p})$. The operator-minimal $\uparrow$-expression based on the primitive lower block partitioning of X_{2p} is the $\uparrow$-expression E'_{2p} from (11.1), which is in minimal normal form. The operator-minimal $\downarrow$-expression based on the primitive upper block partitioning of X_{2p} is

$$\widehat{E'_{2p}} = \left\langle \downarrow \underbrace{\langle\uparrow \langle\updownarrow \alpha\rangle \alpha \langle\updownarrow \alpha\rangle\rangle \alpha \ldots \langle\uparrow \langle\updownarrow \alpha\rangle \alpha \langle\updownarrow \alpha\rangle\rangle \alpha}_{2p-1 \text{ times}} \langle\uparrow \langle\updownarrow \alpha\rangle \alpha \langle\updownarrow \alpha\rangle\rangle \right\rangle .$$

Now, the DNA expression E' in minimal normal form, satisfying $E' \equiv E = E_{2p+1}$, is the $\downarrow$-expression E'_{2p+1} from (11.2). This DNA expression can be constructed in constant time from $\widehat{E'_{2p}}$. In addition, we use E'_{2p} to construct the equivalent, operator-minimal $\uparrow$-expression

$$\widehat{E'} = \left\langle \uparrow \underbrace{\langle\downarrow \langle\updownarrow \alpha\rangle \alpha \langle\updownarrow \alpha\rangle\rangle \alpha \ldots \langle\downarrow \langle\updownarrow \alpha\rangle \alpha \langle\updownarrow \alpha\rangle\rangle \alpha}_{2p \text{ times}} \langle\downarrow \langle\updownarrow \alpha\rangle \alpha \langle\updownarrow \alpha\rangle\rangle \right\rangle .$$

The construction of $\widehat{E'}$ can also be done in constant time. ■

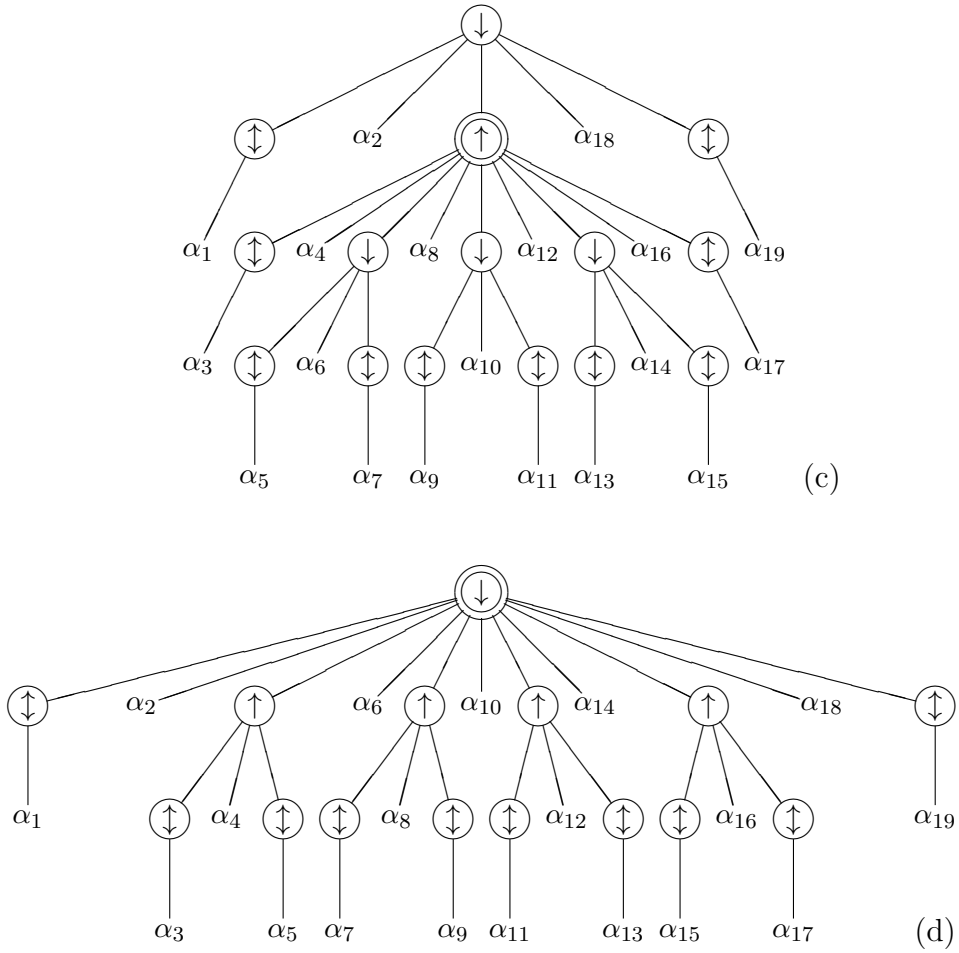


Figure 11.3: Structure trees of the DNA expressions that we successively obtain, when we apply the recursive function `MakeMinimalNF` to the ↓-expression E_5 from Example 11.2 (continuation of Figure 11.2). (c) Structure tree after rewriting the DNA subexpression E_4 into the minimal normal form equivalent E_4'. The node corresponding to E_4' is marked with an extra circle. (d) Structure tree of the final result of the function, the minimal normal form equivalent E_5' of E_5 itself. For consistency, the root node (corresponding to E_5') is marked with an extra circle.

If E denotes a formal DNA molecule X denoting nick letters, then without loss of generality, assume that these are lower nick letters. We know from Lemma 5.2(1) that we cannot find an operator-minimal ↓-expression denoting X. In that case, we consider the substrings Z_h occurring in the nick free decomposition of X. For each Z_h, we maintain both the operator-minimal ↑-expression denoting Z_h based on the primitive lower block partitioning, and the operator-minimal ↓-expression denoting Z_h based on the primitive upper block partitioning. Now, the operator-minimal ↑-expressions are used to construct the ↑-expression E' in minimal normal form satisfying $E' \equiv E$. The operator-minimal ↓-expressions are used to construct a second, equivalent ↑-expression $\widehat{E'}$.

There are many more details about this alternative implementation of `MakeMinimalNF` that should be worked out, before one can conclude that its time complexity is really linear, as desired. We believe it is possible, but do not do this in this thesis. Instead, we describe a completely different algorithm, which maintains only one DNA expression,

```
1.   NormalizeMinimal (E_2^*)
       // rewrites an arbitrary minimal DNA expression E_2^*
       // into a DNA expression E_3^* in minimal normal form satisfying E_3^* ≡ E_2^*;
       // uses local rearrangements of the DNA expression for this
2.   {
3.     E = E_2^*;
4.     if (E is an ↕-expression)
5.     then E_3^* = E;
6.     else    // E is an ↑-expression or a ↓-expression;
               // without loss of generality, assume it is an ↑-expression
7.          if (E is alternating and its first argument is a ↓-argument)
8.          then substitute E by the result of procedure RotateToMinimal;
                                                                  (D_MinNF.5)
9.          fi
              // E is an ↑-expression or a ↓-expression;
              // without loss of generality, assume it is an ↑-expression
10.         while (E has inner occurrences of ↑)
11.         do   select a ↓-subexpression Ê of E
                   which has at least one ↑-argument E_i;
                      // Ê = ⟨↓ ε_1 ... ε_{i-1} E_i ε_{i+1} ... ε_n⟩
                      // and E_i = ⟨↑ ε_{i,1} ε_{i,2} ... ε_{i,m-1} ε_{i,m}⟩
12.              substitute Ê in E
                   by ⟨↓ ε_1 ... ε_{i-1} ε_{i,1}⟩ ε_{i,2} ... ε_{i,m-1} ⟨↓ ε_{i,m} ε_{i+1} ... ε_n⟩;   (D_MinNF.4)
13.         od
14.         E_3^* = E;
15.    fi
16.  }
```

Figure 11.4: Pseudo-code of the algorithm `NormalizeMinimal`.

performs rewriting steps directly in that DNA expression, and has linear complexity, after all.

11.2 Two-step algorithm for the minimal normal form

As we have seen in Section 11.1, a natural implementation of the direct, recursive function `MakeMinimalNF` might produce an equivalent DNA expression in minimal normal form for its argument E, but would not really be efficient. We now propose another, two-step algorithm. Given an arbitrary DNA expression E_1^*, we first use the function `MakeMinimal` to construct an equivalent, minimal DNA expression E_2^*. This DNA expression is not necessarily in minimal normal form. We subsequently rewrite E_2^* into the minimal normal form.

In Figure 11.4, we give pseudo-code for the algorithm `NormalizeMinimal`, which performs this second step. Both substitutions occurring in this pseudo-code can be achieved by local rearrangements of brackets and operators in the DNA expression.

As usual, in `NormalizeMinimal`, we consider $\updownarrow$-expressions on the one hand, and $\uparrow$- and $\downarrow$-expressions on the other hand, separately. If the minimal DNA expression E_2^* is an $\updownarrow$-expression, then by Theorem 7.5, there is no other minimal DNA expression with the same semantics. Hence, E_2^* must be in minimal normal form already. It does not have to be rewritten. This explains line 5.

Now, let us assume that $E = E_2^*$ is an $\uparrow$-expression. In lines 7–9, we consider the case that E is alternating and its first argument is a $\downarrow$-expression. In this case, as indicated in the code, E violates Property ($\mathcal{D}_{\text{MinNF}}$.5). We correct this by applying a procedure that we also used in the implementation of `MakeMinimal`, namely `RotateToMinimal`, see Figure 9.5.

In the subsequent while-loop, we deal with inner occurrences of $\uparrow$ in the $\uparrow$-expression E. As we have seen in the proof of Lemma 10.9(1), such inner occurrences correspond to violations of Property ($\mathcal{D}_{\text{MinNF}}$.4). When we perform the substitution in line 12, we get rid of one inner occurrence of $\uparrow$.

In Lemma 10.10, we have established an upper bound on the nesting level of the brackets in a DNA expression in minimal normal form. In fact, due to the substitution in line 12, the nesting level decreases by 2 at the location of the substitution. We can also use the terms from Definition 10.1: the substitution in line 12 corresponds to breaking a large lower block into two smaller lower blocks.

Note that Properties ($\mathcal{D}_{\text{MinNF}}$.1)–($\mathcal{D}_{\text{MinNF}}$.3) are not mentioned in the pseudo-code. This is natural, as they equal Properties ($\mathcal{D}_{\text{Min}}$.1)–($\mathcal{D}_{\text{Min}}$.3) of minimal DNA expressions, and the input of `NormalizeMinimal` is supposed to be minimal.

We illustrate the algorithm by an example. In this example, we also show (or refer back to) the structure trees of the DNA expressions we obtain in the course of the algorithm.

Example 11.5 In Example 7.26, we have constructed four minimal DNA expressions for the formal DNA molecule X depicted in Figure 7.6. Let

$$E = E_c = \langle\downarrow \langle\uparrow \alpha_1 \langle\updownarrow \alpha_2\rangle\rangle \alpha_3 \langle\uparrow \langle\updownarrow \alpha_4\rangle \alpha_5 \langle\updownarrow \alpha_6\rangle \alpha_7 \langle\updownarrow \alpha_8\rangle\rangle \alpha_9 \langle\updownarrow \alpha_{10}\rangle\rangle \tag{11.4}$$

(see (7.11)), which has been depicted in Figure 11.5(a). The fact that E is minimal implies (1) that, by Theorem 9.12, it is not affected by the recursive function `MakeMinimal`, and (2) that we can apply the algorithm `NormalizeMinimal` to it.

E is an alternating $\downarrow$-expression. Because its first argument is the $\uparrow$-expression $E_1 = \langle\uparrow \alpha_1 \langle\updownarrow \alpha_2\rangle\rangle$, E violates Property ($\mathcal{D}_{\text{MinNF}}$.5). According to (the analogue for $\downarrow$-expressions of) line 8 of algorithm `NormalizeMinimal` and line RtM.6 of procedure `RotateToMinimal`, E is substituted by

$$E = \langle\uparrow \alpha_1 \langle\downarrow \langle\updownarrow \alpha_2\rangle \alpha_3 \langle\uparrow \langle\updownarrow \alpha_4\rangle \alpha_5 \langle\updownarrow \alpha_6\rangle \alpha_7 \langle\updownarrow \alpha_8\rangle\rangle \alpha_9 \langle\updownarrow \alpha_{10}\rangle\rangle\rangle\,. \tag{11.5}$$

This is the minimal DNA expression E_b from (7.10). It has been depicted in Figure 11.5(b). Because the $\uparrow$-expression E has an inner occurrence of $\uparrow$, we enter the while-loop. We select the $\downarrow$-subexpression

$$\widehat{E} = \langle\downarrow \langle\updownarrow \alpha_2\rangle \alpha_3 \langle\uparrow \langle\updownarrow \alpha_4\rangle \alpha_5 \langle\updownarrow \alpha_6\rangle \alpha_7 \langle\updownarrow \alpha_8\rangle\rangle \alpha_9 \langle\updownarrow \alpha_{10}\rangle\rangle\,,$$

(the second argument of E), whose third argument is the $\uparrow$-expression $E_3 = \langle\uparrow \langle\updownarrow \alpha_4\rangle \alpha_5 \langle\updownarrow \alpha_6\rangle \alpha_7 \langle\updownarrow \alpha_8\rangle\rangle$. Because the outermost operator $\downarrow$ of $\widehat{E}$ is an inner occurrence in E, it violates Property ($\mathcal{D}_{\text{MinNF}}$.4). According to line 12 of algorithm `NormalizeMinimal`, $\widehat{E}$ is substituted in E by the sequence of arguments

$$\langle\downarrow \langle\updownarrow \alpha_2\rangle \alpha_3 \langle\updownarrow \alpha_4\rangle\rangle \quad \alpha_5 \langle\updownarrow \alpha_6\rangle \alpha_7 \quad \langle\downarrow \langle\updownarrow \alpha_8\rangle \alpha_9 \langle\updownarrow \alpha_{10}\rangle\rangle\,,$$

yielding

$$E = \langle\uparrow \alpha_1 \langle\downarrow \langle\updownarrow \alpha_2\rangle \alpha_3 \langle\updownarrow \alpha_4\rangle\rangle \alpha_5 \langle\updownarrow \alpha_6\rangle \alpha_7 \langle\downarrow \langle\updownarrow \alpha_8\rangle \alpha_9 \langle\updownarrow \alpha_{10}\rangle\rangle\rangle\,. \tag{11.6}$$

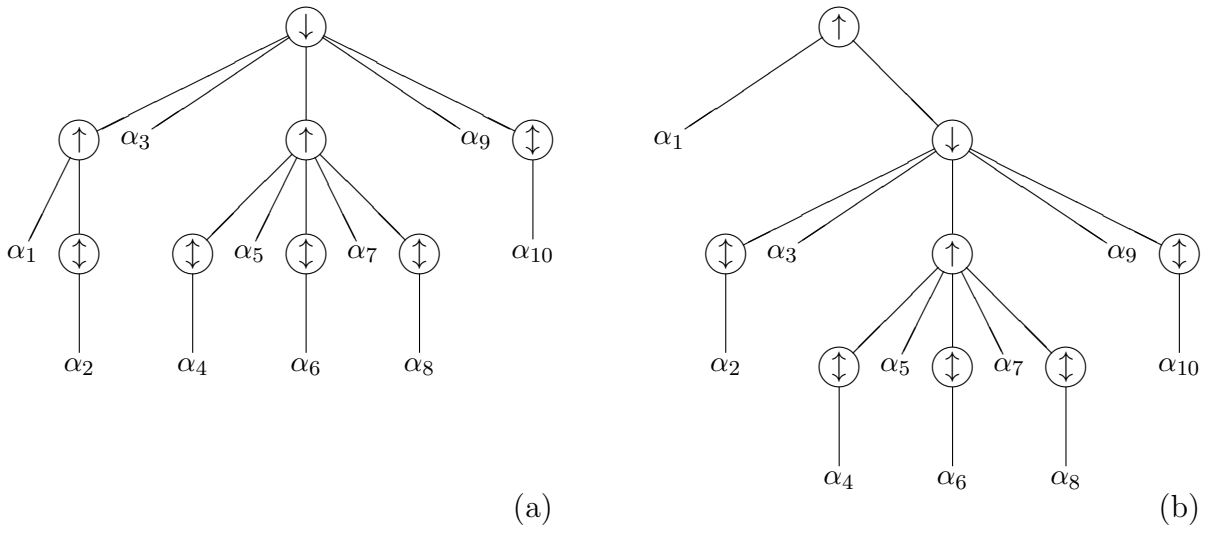


Figure 11.5: Structure trees of the first two minimal DNA expressions occurring in Example 11.5, denoting the formal DNA molecule from Figure 7.6. (a) The structure tree of E_c from (11.4). (b) The structure tree of E_b from (11.5).

After the substitution, E has no inner occurrences of $\uparrow$ any more, and we exit the while-loop. We do not rewrite the DNA expression any further. Indeed, E has all five properties from Lemma 10.6, and thus is in minimal normal form. It equals $E_{\text{MinNF}}(X) = E_a$ from (7.9) and (10.2), which has been depicted in Figure 10.1(b). ■

In the above example, the while-loop in lines 10–13 of `NormalizeMinimal` has only one iteration. In general, there may be more iterations. We will see an example of this in Section 11.3.

It is possible that for a given DNA expression E_1^*, the result E_2^* of function `MakeMinimal` is already in minimal normal form. One can verify that this is, e.g., the case for the DNA expression from Example 9.28. In such a case, `NormalizeMinimal` obviously will not find violations of Properties ($\mathcal{D}_{\text{MinNF}}$.4) and ($\mathcal{D}_{\text{MinNF}}$.5) in E_2^*, and will leave E_2^* unchanged.

When we introduced algorithm `NormalizeMinimal`, we already mentioned the relation between inner occurrences of $\uparrow$ in an $\uparrow$-expression E (and inner occurrences of $\downarrow$ in a $\downarrow$-expression E) and violations of Property ($\mathcal{D}_{\text{MinNF}}$.4). This property deals (a.o.) with the arguments of *arbitrary* inner occurrences of $\downarrow$ in E, i.e., the arguments of *arbitrary* proper $\downarrow$-subexpressions of E. We now focus on the arguments of (direct) $\downarrow$-arguments of an $\uparrow$-expression E.

Lemma 11.6 *Let E be a minimal $\uparrow$-expression. Then E has an inner occurrence of $\uparrow$, if and only if E has a $\downarrow$-argument with at least one $\uparrow$-argument.*

Proof: Obviously, if E has a $\downarrow$-argument with at least one $\uparrow$-argument, then E has an inner occurrence of $\uparrow$.

Now assume that E has an inner occurrence $\uparrow_1$ of $\uparrow$. Then $\uparrow_1$ occurs in an argument $\widehat{\varepsilon}$ of E. By Corollary 8.2, $\widehat{\varepsilon}$ is either an $\mathcal{N}$-word α, or an $\updownarrow$-expression $\langle\updownarrow \alpha\rangle$ for an $\mathcal{N}$-word α, or a $\downarrow$-expression. Because the first two types of arguments do not contain occurrences of $\uparrow$, $\widehat{\varepsilon}$ must be a $\downarrow$-expression $\widehat{E}$.

Inside $\widehat{E}$, $\uparrow_1$ occurs in an argument ε_i of $\widehat{E}$. Because E is minimal, so is $\widehat{E}$. Hence, by Corollary 8.2, ε_i is either an $\mathcal{N}$-word α, or an $\updownarrow$-expression $\langle\updownarrow\alpha\rangle$ for an $\mathcal{N}$-word α, or an $\uparrow$-expression. Because ε_i contains $\uparrow_1$, it must be an $\uparrow$-expression E_i. We conclude that E has a $\downarrow$-argument $\widehat{E}$ with at least one $\uparrow$-argument E_i.

Note that $\uparrow_1$ may be the outermost operator of E_i, but it may also be an inner occurrence in E_i. This is not important for the proof. □

We prove that algorithm `NormalizeMinimal` is correct.

Theorem 11.7 *Let E_2^* be an arbitrary minimal DNA expression, and let E_3^* be the result of applying algorithm* `NormalizeMinimal` *to E_2^*.*

1. *Algorithm* `NormalizeMinimal` *is well defined.*
2. *Algorithm* `NormalizeMinimal` *terminates.*
3. *The string E_3^* is a DNA expression in minimal normal form satisfying $E_3^* \equiv E_2^*$.*
4. *E_3^* is independent of the order in which $\downarrow$-subexpressions $\widehat{E}$ with at least one $\uparrow$-argument E_i are selected in line 11.*

Proof: We combine the proofs of Claims 1 and 3, because both of them (partly) rely on an invariant of the while-loop in algorithm `NormalizeMinimal`.

1, 3. The only instructions that are not obviously well defined, are the ones in lines 8, 11 and 12. Before we can apply procedure `RotateToMinimal` to E in line 8, we must verify that E satisfies the preconditions of the procedure. In line 11, we select a $\downarrow$-subexpression $\widehat{E}$ that has at least one $\uparrow$-argument. Of course, this is only possible, if E has at least one such $\downarrow$-subexpression. Finally, the substitution in line 12 is only well defined if $m \geq 2$.

We first consider the case that E_2^* is an $\updownarrow$-expression. Because E_2^* is minimal, by Theorem 7.5, $E_2^* = \langle\updownarrow\alpha_1\rangle$ for an $\mathcal{N}$-word α_1. By Case 1 of Definition 10.1, E_2^* is in minimal normal form, already. In this case, by line 5 of `NormalizeMinimal`, $E_3^* = E = E_2^*$. Obviously, E_3^* satisfies $E_3^* \equiv E_2^*$.

Now assume that E_2^* is an $\uparrow$-expression or a $\downarrow$-expression. We enter the else-branch in line 6 with $E = E_2^*$. Because E is minimal, it has Properties ($\mathcal{D}_{\text{Min}}$.1)–($\mathcal{D}_{\text{Min}}$.6) from Lemma 8.22. E also has Properties ($\mathcal{D}_{\text{MinNF}}$.1)–($\mathcal{D}_{\text{MinNF}}$.3) from Lemma 10.6, because these properties are equal to Properties ($\mathcal{D}_{\text{Min}}$.1)–($\mathcal{D}_{\text{Min}}$.3). E does, however, not necessarily have Properties ($\mathcal{D}_{\text{MinNF}}$.4) and ($\mathcal{D}_{\text{MinNF}}$.5).

Without loss of generality, we assume that E is an $\uparrow$-expression. By Corollary 8.2, the first argument of E is either an $\mathcal{N}$-word α, or an $\updownarrow$-expression $\langle\updownarrow\alpha\rangle$ for an $\mathcal{N}$-word α, or a $\downarrow$-argument.

If the first argument of E is an $\mathcal{N}$-word α or an $\updownarrow$-expression $\langle\updownarrow\alpha\rangle$ for an $\mathcal{N}$-word α, or E has two consecutive expression-arguments, then E has Property ($\mathcal{D}_{\text{MinNF}}$.5) and we skip line 8 of `NormalizeMinimal`.

If on the other hand, the first argument of E is a $\downarrow$-argument and E is alternating, then E does not have Property ($\mathcal{D}_{\text{MinNF}}$.5) and we do execute line 8. Indeed, E satisfies all conditions of (the analogue for $\uparrow$-expressions of) procedure `RotateToMinimal`. By Property ($\mathcal{D}_{\text{Min}}$.6), the last argument of E cannot be another $\downarrow$-argument. Hence,

in `RotateToMinimal`, we execute line RtM.6. The result is a minimal $\downarrow$-expression E', which satisfies $E' \equiv E$ and whose last argument is an $\uparrow$-argument. As we have seen in the proof of Theorem 9.27, the first argument $\varepsilon_{1,1}$ of E' is either an $\mathcal{N}$-word α or an $\updownarrow$-expression $\langle\updownarrow \alpha\rangle$ for an $\mathcal{N}$-word α. Hence, E' has Property ($\mathcal{D}_{\text{MinNF}}$.5).

In both cases, after the if-then construction of lines 7–9, E is a minimal $\uparrow$-expression or $\downarrow$-expression with Property ($\mathcal{D}_{\text{MinNF}}$.5), which satisfies $E \equiv E_2^*$. Without loss of generality, we again assume that E is an $\uparrow$-expression. We thus have

$$E \text{ is a minimal } \uparrow\text{-expression with Property } (\mathcal{D}_{\text{MinNF}}.5), \text{ satisfying } E \equiv E_2^*. \tag{11.7}$$

Before we prove that this property is an invariant for the while-loop in `NormalizeMinimal`, we examine some implications. As we observed before, because E is minimal, it also has Properties ($\mathcal{D}_{\text{MinNF}}$.1)–($\mathcal{D}_{\text{MinNF}}$.3). Hence, Property (11.7) and Theorem 10.8 imply that E is in minimal normal form, if and only if E has Property ($\mathcal{D}_{\text{MinNF}}$.4).

Now suppose that E has at least one inner occurrence of $\uparrow$. Because E is minimal, we can apply Lemma 11.6 and conclude that E has a $\downarrow$-argument with at least one $\uparrow$-argument. Then there certainly exists a $\downarrow$-subexpression $\widehat{E}$ of E with at least one $\uparrow$-argument. Hence, line 11 of `NormalizeMinimal` is well defined.[1] Moreover, the outermost operator $\downarrow$ of $\widehat{E}$ (which is an inner occurrence in E) makes E violate Property ($\mathcal{D}_{\text{MinNF}}$.4).

Suppose, on the other hand, that E has no inner occurrence of $\uparrow$. Let $\downarrow_1$ be an inner occurrence of $\downarrow$ in E. Because E is minimal, so is the DNA subexpression of E governed by $\downarrow_1$. By Corollary 8.2, the arguments of $\downarrow_1$ are $\mathcal{N}$-words α, $\updownarrow$-expressions $\langle\updownarrow \alpha\rangle$ for $\mathcal{N}$-words α, or $\uparrow$-expressions. The last type of arguments, however, is not possible, because $\uparrow$-arguments would correspond to inner occurrences of $\uparrow$. Now by Property ($\mathcal{D}_{\text{Min}}$.4) of E, the arguments of $\downarrow_1$ are maximal $\mathcal{N}$-word occurrences α and $\updownarrow$-expressions $\langle\updownarrow \alpha\rangle$ for $\mathcal{N}$-words α, alternately. This implies that E has Property ($\mathcal{D}_{\text{MinNF}}$.4).

We conclude that (under the assumption that Property (11.7) is valid) E has no inner occurrences of $\uparrow$, if and only if E has Property ($\mathcal{D}_{\text{MinNF}}$.4), which is the case if and only if E is in minimal normal form.

We now prove that Property (11.7) is indeed an invariant for the while-loop.

- Clearly, before the first iteration of the while-loop, Property (11.7) is valid.
- Suppose that Property (11.7) is valid before a certain iteration of the while-loop.

  When we enter the iteration, E has at least one inner occurrence of $\uparrow$. As we just observed, there indeed exists at least one $\downarrow$-subexpression of E with an $\uparrow$-argument. Let $\widehat{E}$ be the $\downarrow$-subexpression of E that we select in line 11, say

  $$\widehat{E} = \langle\downarrow \varepsilon_1 \ldots \varepsilon_{i-1} \langle\uparrow \varepsilon_{i,1}\varepsilon_{i,2} \ldots \varepsilon_{i,m-1}\varepsilon_{i,m}\rangle \varepsilon_{i+1} \ldots \varepsilon_n\rangle$$

  for some $m, n \geq 1$ and $\mathcal{N}$-words and DNA expressions $\varepsilon_1, \ldots, \varepsilon_{i-1}, \varepsilon_{i+1}, \ldots, \varepsilon_n$, and $\varepsilon_{i,1}, \varepsilon_{i,2}, \ldots, \varepsilon_{i,m-1}, \varepsilon_{i,m}$.

[1] There may also be $\downarrow$-subexpressions $\widehat{E}$ of E with an $\uparrow$-argument, which are not arguments of E. They occur *in* arguments of E. In line 11, we may also select such a $\downarrow$-subexpression.

We zoom in on the $\uparrow$-argument $E_i = \langle\uparrow \varepsilon_{i,1}\varepsilon_{i,2}\ldots\varepsilon_{i,m-1}\varepsilon_{i,m}\rangle$. E_i is the argument of the $\downarrow$-expression $\widehat{E}$, which is in turn a proper DNA subexpression of the minimal $\uparrow$-expression E. By Lemma 8.27(7), $m \geq 3$ and both $\varepsilon_{i,1}$ and $\varepsilon_{i,m}$ are $\updownarrow$-expressions. Then certainly $m \geq 2$, which implies that the substitution in line 12 is well defined. By Property ($\mathcal{D}_{\text{Min}}$.4), $\varepsilon_{i,1}, \varepsilon_{i,2}, \ldots, \varepsilon_{i,m-1}, \varepsilon_{i,m}$ form an alternating sequence of maximal $\mathcal{N}$-word occurrences and DNA expressions. In particular, $\varepsilon_{i,2}$ and $\varepsilon_{i,m-1}$ are $\mathcal{N}$-words.

We now consider $\widehat{E}$ itself. As we just mentioned, $\widehat{E}$ is a proper DNA subexpression of E. By Property ($\mathcal{D}_{\text{Min}}$.5), E_i cannot be the first or the last argument of $\widehat{E}$, so $2 \leq i \leq n-1$. By Property ($\mathcal{D}_{\text{Min}}$.4), each occurrence of $\uparrow$ or $\downarrow$ in $\widehat{E}$ is alternating. Now when we apply Theorem 5.19(1) and (2) to $\widehat{E}$ (with $r = 1$), we find that

$$\widehat{E}' = \langle\uparrow \langle\downarrow \varepsilon_1 \ldots \varepsilon_{i-1}\varepsilon_{i,1}\rangle\, \varepsilon_{i,2}\ldots\varepsilon_{i,m-1}\, \langle\downarrow \varepsilon_{i,m}\varepsilon_{i+1}\ldots\varepsilon_n\rangle\rangle$$

is a DNA expression satisfying $\widehat{E}' \equiv \widehat{E}$. [2]

By Lemma 8.27(1b), the parent operator of $\widehat{E}$ in E is an occurrence $\uparrow_0$ of $\uparrow$. Let $\widehat{E}$ be the j^{th} argument of $\uparrow_0$, and let E_0 be the DNA subexpression of E governed by $\uparrow_0$:

$$E_0 = \left\langle\uparrow_0 \widehat{\varepsilon}_1 \ldots \widehat{\varepsilon}_{j-1}\widehat{E}\widehat{\varepsilon}_{j+1}\ldots\widehat{\varepsilon}_l\right\rangle \tag{11.8}$$

for some $l \geq 1$ and $\mathcal{N}$-words and DNA expressions $\widehat{\varepsilon}_1, \ldots, \widehat{\varepsilon}_{j-1}, \widehat{\varepsilon}_{j+1}, \ldots, \widehat{\varepsilon}_l$. Note that E_0 may be equal to E, but that is not important for the moment. By Lemma 5.11 and Lemma 5.10,

$$\begin{aligned}
E_0 &\equiv \left\langle\uparrow_0\ \widehat{\varepsilon}_1 \ldots \widehat{\varepsilon}_{j-1}\widehat{E}'\widehat{\varepsilon}_{j+1}\ldots\widehat{\varepsilon}_l\right\rangle \\
&= \Big\langle\uparrow_0\ \widehat{\varepsilon}_1 \ldots \widehat{\varepsilon}_{j-1} \\
&\qquad \langle\uparrow \langle\downarrow \varepsilon_1 \ldots \varepsilon_{i-1}\varepsilon_{i,1}\rangle\, \varepsilon_{i,2}\ldots\varepsilon_{i,m-1}\, \langle\downarrow \varepsilon_{i,m}\varepsilon_{i+1}\ldots\varepsilon_n\rangle\rangle \\
&\qquad \widehat{\varepsilon}_{j+1}\ldots\widehat{\varepsilon}_l\Big\rangle \\
&\equiv \Big\langle\uparrow_0\ \widehat{\varepsilon}_1 \ldots \widehat{\varepsilon}_{j-1} \\
&\qquad \langle\downarrow \varepsilon_1 \ldots \varepsilon_{i-1}\varepsilon_{i,1}\rangle\, \varepsilon_{i,2}\ldots\varepsilon_{i,m-1}\, \langle\downarrow \varepsilon_{i,m}\varepsilon_{i+1}\ldots\varepsilon_n\rangle \\
&\qquad \widehat{\varepsilon}_{j+1}\ldots\widehat{\varepsilon}_l\Big\rangle .
\end{aligned} \tag{11.9}$$

Hence, when we substitute $\widehat{E}$ in E_0 (and thus in E) by

$$\langle\downarrow \varepsilon_1 \ldots \varepsilon_{i-1}\varepsilon_{i,1}\rangle\, \varepsilon_{i,2}\ldots\varepsilon_{i,m-1}\, \langle\downarrow \varepsilon_{i,m}\varepsilon_{i+1}\ldots\varepsilon_n\rangle\,, \tag{11.10}$$

like we do in line 12 of `NormalizeMinimal`, we obtain an equivalent $\uparrow$-expression. After the substitution, E still satisfies $E \equiv E_2^*$. Moreover, it is easily verified that after the substitution, E has the same length as before the substitution. This implies that E is still minimal.

We finally verify that E also has Property ($\mathcal{D}_{\text{MinNF}}$.5) after the substitution. If E_0 was a proper DNA subexpression of E, then the substitution has no effect on the number of arguments and the types of arguments of E. Hence, E has

[2] The substitution in line 12 of `NormalizeMinimal` is almost the reverse of line RtM.5 of procedure `RotateToMinimal`. This explains why we use the same type of arguments to prove that the operations do not affect the semantics of the DNA expression, see the proof of Theorem 9.27.

Property ($\mathcal{D}_{\mathrm{MinNF}}$.5) after the substitution, because it had this property before the substitution.

Now assume that E_0 happened to be E itself. The $\downarrow$-argument $\widehat{E}$ of E has been substituted by the sequence of arguments in (11.10). This is an alternating sequence of $\mathcal{N}$-words and DNA expressions, which both starts and ends with a $\downarrow$-expression. It is easily verified that E was alternating before the substitution of $\widehat{E}$, if and only if E is alternating after the substitution.

By Property ($\mathcal{D}_{\mathrm{MinNF}}$.5), before the substitution, either the first argument of $E = E_0$ was an $\mathcal{N}$-word α or an $\updownarrow$-expression $\langle\updownarrow \alpha\rangle$ for an $\mathcal{N}$-word α, or E was not alternating. In the former case, it follows from (11.8) and (11.9) that $j \geq 2$ and the first argument $\widehat{\varepsilon}_1$ of E is not affected by the substitution. It is still α or $\langle\updownarrow \alpha\rangle$ after the substitution. In the latter case, as we just observed, E is not alternating after the substitution, either. In both cases, E also has Property ($\mathcal{D}_{\mathrm{MinNF}}$.5) after the substitution.

Indeed, Property (11.7) is an invariant of the while-loop. After the last iteration of the loop, E has no inner occurrences of $\uparrow$ any more, which implies that E is in minimal normal form. By the invariant, E satisfies $E \equiv E_2^*$. This carries over to E_3^*.

2. In every iteration of the while-loop, we substitute a $\downarrow$-subexpression

$$\widehat{E} = \langle\downarrow \varepsilon_1 \ldots \varepsilon_{i-1} \langle\uparrow \varepsilon_{i,1}\varepsilon_{i,2} \ldots \varepsilon_{i,m-1}\varepsilon_{i,m}\rangle \varepsilon_{i+1} \ldots \varepsilon_n\rangle$$

of E by the sequence of arguments

$$\langle\downarrow \varepsilon_1 \ldots \varepsilon_{i-1}\varepsilon_{i,1}\rangle \varepsilon_{i,2} \ldots \varepsilon_{i,m-1} \langle\downarrow \varepsilon_{i,m}\varepsilon_{i+1} \ldots \varepsilon_n\rangle .$$

This way, we decrease the number of inner occurrences of $\uparrow$ in E by 1. Because, obviously, this number cannot become negative, the number of iterations of the while-loop is bounded, and algorithm `NormalizeMinimal` terminates.

4. By Claim 3, E_3^* is a DNA expression in minimal normal form satisfying $E_3^* \equiv E_2^*$, i.e., with $\mathcal{S}(E_3^*) = \mathcal{S}(E_2^*)$. By definition, there is only one DNA expression in minimal normal form with this semantics. Then E_3^* is certainly independent of the order in which $\downarrow$-subexpressions $\widehat{E}$ with at least one $\uparrow$-argument E_i are selected in line 11.

This completes the proof of Theorem 11.7. □

11.3 Implementation and complexity of the algorithm

In the description of algorithm `NormalizeMinimal` in Figure 11.4, we have not specified all details of the while-loop. In particular, in line 11, we have not specified *how* to select a $\downarrow$-subexpression $\widehat{E}$ of E with at least one $\uparrow$-argument E_i. To make it possible to analyse the algorithm's complexity, we now make the description more precise. In fact, we completely rewrite the while-loop. However, the purpose of the loop (to achieve Property ($\mathcal{D}_{\mathrm{MinNF}}$.4)) and the types of substitutions performed in the loop remain the same.

We also describe three features of a data structure to store the DNA expression in. We prove that with this data structure, the algorithm can be carried out in linear time.

In the proof of Theorem 11.7(1) and (3), we have established that during the while-loop of `NormalizeMinimal`, the $\uparrow$-expression E is minimal. Hence, by Lemma 11.6, the condition

while (E has inner occurrences of $\uparrow$)

in line 10 of Figure 11.4 is equivalent to

while (E has a $\downarrow$-argument with at least one $\uparrow$-argument).

If E has such a $\downarrow$-argument $\widehat{E}$, then that is, in particular, a $\downarrow$-subexpression of E with at least one $\uparrow$-argument. Hence, in line 11, we can simply select this $\downarrow$-argument.

A natural implementation of the while-loop would then consist of iterating over all $\downarrow$-arguments of E, and selecting the ones that have at least one $\uparrow$-argument. Note, however, that the substitution in line 12 introduces new arguments $\langle\downarrow \varepsilon_1 \ldots \varepsilon_{i-1}\varepsilon_{i,1}\rangle, \varepsilon_{i,2}, \ldots, \varepsilon_{i,m-1}, \langle\downarrow \varepsilon_{i,m}\varepsilon_{i+1} \ldots \varepsilon_n\rangle$ for E. These may include new $\downarrow$-arguments with at least one $\uparrow$-argument, which also have to substituted. This is accounted for in algorithm `NormalizeMinimal2`, which is given in Figure 11.6. The while-loop in `NormalizeMinimal2` considers all arguments $\widehat{\varepsilon}$ of E from left to right. A boolean `stop` indicates whether or not the last argument of E has been considered.

As an illustration, we revisit the DNA expressions from Example 11.2, for which the recursive function `MakeMinimalNF` turned out to use quadratic time.

Example 11.8 Let α be an arbitrary $\mathcal{N}$-word, and let

$$\begin{array}{ll} E_1 = \langle\downarrow \langle\updownarrow \alpha\rangle\, \alpha\, \langle\updownarrow \alpha\rangle\rangle\,, & \\ E_{2p} = \langle\uparrow \langle\updownarrow \alpha\rangle\, \alpha\ E_{2p-1}\ \alpha\, \langle\updownarrow \alpha\rangle\rangle & (p \geq 1), \\ E_{2p+1} = \langle\downarrow \langle\updownarrow \alpha\rangle\, \alpha\ E_{2p}\ \alpha\, \langle\updownarrow \alpha\rangle\rangle & (p \geq 1). \end{array}$$

As we observed in Example 11.2, for $p \geq 1$, both E_{2p} and E_{2p+1} are minimal. The starting DNA expression E_1 is also minimal. The fact that for each $q \geq 1$, E_q is minimal, implies (1) that, by Theorem 9.12, E_q is not affected by the recursive function `MakeMinimal`, and (2) that we can apply the algorithm `NormalizeMinimal2` to it.

For $q \geq 1$, E_q is alternating but its first argument is $\langle\updownarrow \alpha\rangle$. Hence, lines 7–9 of the algorithm are not applicable. We examine the effect of the while-loop on an $\uparrow$-expression E_{2p} for $p \geq 2$:

$$\begin{array}{rcl} E = E_{2p} & = & \langle\uparrow \langle\updownarrow \alpha\rangle\, \alpha\ E_{2p-1}\ \alpha\, \langle\updownarrow \alpha\rangle\rangle \\ & = & \big\langle\uparrow \langle\updownarrow \alpha\rangle\, \alpha \big\langle\downarrow \langle\updownarrow \alpha\rangle\, \alpha \big\langle\uparrow \langle\updownarrow \alpha\rangle\, \alpha\ E_{2(p-1)-1}\ \alpha\, \langle\updownarrow \alpha\rangle\big\rangle\, \alpha\, \langle\updownarrow \alpha\rangle\big\rangle\, \alpha\, \langle\updownarrow \alpha\rangle\big\rangle\,. \end{array}$$

The third argument of E_{2p} is the $\downarrow$-expression E_{2p-1}, which has in turn as an argument the $\uparrow$-expression $E_{2(p-1)}$. The outermost operator $\downarrow$ of E_{2p-1} violates Property ($\mathcal{D}_{\text{MinNF}}$.4). According to line 14 of `NormalizeMinimal2`, E_{2p-1} is substituted in E by the sequence of arguments

$$\langle\downarrow \langle\updownarrow \alpha\rangle\, \alpha\, \langle\updownarrow \alpha\rangle\rangle\, \alpha\ E_{2(p-1)-1}\ \alpha\, \langle\downarrow \langle\updownarrow \alpha\rangle\, \alpha\, \langle\updownarrow \alpha\rangle\rangle\,,$$

yielding

$$E \;=\; \big\langle\uparrow \langle\updownarrow \alpha\rangle\, \alpha\, \langle\downarrow \langle\updownarrow \alpha\rangle\, \alpha\, \langle\updownarrow \alpha\rangle\rangle\, \alpha\ E_{2(p-1)-1}\ \alpha\, \langle\downarrow \langle\updownarrow \alpha\rangle\, \alpha\, \langle\updownarrow \alpha\rangle\rangle\, \alpha\, \langle\updownarrow \alpha\rangle\big\rangle\,.$$

After the substitution, the algorithm proceeds with the (new) fourth argument of E, which is an $\mathcal{N}$-word α. The fifth argument of E is the $\downarrow$-expression $E_{2(p-1)-1}$. If $p \geq 3$, then

```
1.   NormalizeMinimal2 (E_2^*)
       // rewrites an arbitrary minimal DNA expression E_2^* into
       // a DNA expression E_3^* in minimal normal form satisfying E_3^* ≡ E_2^*;
       // uses local rearrangements of the DNA expression for this
2.   {
3.     E = E_2^*;
4.     if (E is an ↕-expression)
5.     then E_3^* = E;
6.     else    // E is an ↑-expression or a ↓-expression;
               // without loss of generality, assume it is an ↑-expression
7.          if (E is alternating and its first argument is a ↓-argument)
8.          then substitute E by the result of procedure RotateToMinimal;
                                                                  (D_MinNF.5)
9.          fi
              // E is an ↑-expression or a ↓-expression;
              // without loss of generality, assume it is an ↑-expression
10.         ε̂ = first argument of E;
11.         stop = false;
12.         while (not stop)
13.         do   if (ε̂ is a ↓-expression with at least one ↑-argument)
                   // let ε̂ = ⟨↓ ε_1 ... ε_{i-1} E_i ε_{i+1} ... ε_n⟩,
                   // where E_i = ⟨↑ ε_{i,1} ε_{i,2} ... ε_{i,m-1} ε_{i,m}⟩
                   // is the first ↑-argument of ε̂
14.                then substitute ε̂ in E
                          by ⟨↓ ε_1 ... ε_{i-1} ε_{i,1}⟩ ε_{i,2} ... ε_{i,m-1} ⟨↓ ε_{i,m} ε_{i+1} ... ε_n⟩;
                                                                  (D_MinNF.4)
15.                     ε̂ = ε_{i,2};
16.                else  if (ε̂ is not the last argument of E)
17.                     then ε̂ = next argument of E;
18.                     else  stop = true;
19.                     fi
20.                fi
21.         od
22.         E_3^* = E;
23.    fi
24.  }
```

Figure 11.6: Pseudo-code of the algorithm `NormalizeMinimal2`, which is a more detailed version of the algorithm `NormalizeMinimal` from Figure 11.4.

this ↓-expression has as an argument the ↑-expression $E_{2(p-2)}$. The outermost operator ↓ of $E_{2(p-1)-1}$ violates Property ($\mathcal{D}_{\text{MinNF}}$.4). According to line 14, $E_{2(p-1)-1}$ is substituted in E by the sequence of arguments

$$\langle\downarrow \langle\updownarrow \alpha\rangle\, \alpha\, \langle\updownarrow \alpha\rangle\rangle\, \alpha\; E_{2(p-2)-1}\; \alpha\, \langle\downarrow \langle\updownarrow \alpha\rangle\, \alpha\, \langle\updownarrow \alpha\rangle\rangle\,.$$

In $p-1$ substitutions, we obtain the DNA expression E'_{2p} from (11.1), which is in minimal normal form. For each substitution, we perform a constant amount of work: remove one occurrence of ↑, add one occurrence of ↓ and rearrange two brackets. Hence, the total amount of work (and time) to rewrite E_{2p} into E'_{2p} is linear in p, and thus linear in $|E_{2p}|$.

The effect of the while-loop on the ↓-expressions E_{2p+1} is analogous. ■

Indeed, for the ↑-expressions E_{2p} with $p \geq 3$ in the example, the substitution of a ↓-argument in line 14 of `NormalizeMinimal2` introduces a new ↓-argument with an ↑-argument, which is in turn substituted. It is not hard to prove by induction, that the

maximal nesting level of the brackets in E_{2p} is $2p + 1$. Due to the substitution in line 14, the nesting level decreases by 2. Successive substitutions bring down the nesting level of the brackets to at most 3.

In Figure 11.7, we have depicted the effect of algorithm `NormalizeMinimal2` for DNA expression E_6 from Example 11.8. In two steps (iterations of the while-loop), we transform the original, tall tree in Figure 11.7(a) into the relatively flat tree in Figure 11.7(c). In each step, the height of the tree decreases by 2: from 8 via 6 to 4. With the recursive function `MakeMinimalNF`, we would need four steps to achieve the same result. Each step would yield a decrease of only 1 (cf. Figures 11.2 and 11.3).

The difference in complexity between (a natural implementation of) `MakeMinimalNF` and `NormalizeMinimal2` is not just this factor of 2. If it were just this factor of 2, then `NormalizeMinimal2` would also require at least quadratic time, while the algorithm was meant to be more efficient. There is, however, a relation with this factor. In the first rewriting step of `MakeMinimalNF` for E_6, we rewrite the $\downarrow$-subexpression E_3 into E_3'. For this, we substitute an inner occurrence of $\downarrow$ by an inner occurrence of $\uparrow$. In the second step, we substitute two inner occurrences of $\uparrow$ (including the one we just introduced) by two inner occurrence of $\downarrow$, and so on. In `NormalizeMinimal2`, we somehow make two steps at a time. Thus, we no longer introduce operators in one step that we have to remove in the next step. This is what really reduces the complexity for the DNA expressions from Example 11.2 and Example 11.8. In Theorem 11.11, we will consider the complexity of `NormalizeMinimal2` for *arbitrary* minimal DNA expressions.

Note that there is another difference between the operation of `MakeMinimalNF` and that of `NormalizeMinimal2`, besides the fact that `NormalizeMinimal2` takes two steps at a time. Due to its recursive set-up, `MakeMinimalNF` rewrites a DNA expression from the inside outwards (bottom-up in the tree). `NormalizeMinimal2`, on the other hand, rewrites a DNA expression from the outside inwards (top-down in the tree).

We prove that `NormalizeMinimal2` is correct. It does not suffice to just refer to Theorem 11.7, where we established the correctness of `NormalizeMinimal`, because the while-loop in the algorithm has significantly changed. In particular, it is not immediately clear that by the end of the loop, the DNA expression has no more inner occurrences of $\uparrow$. We can, however, reuse some elements of the argumentation.

Theorem 11.9 *Let E_2^* be an arbitrary minimal DNA expression.*

1. *Algorithm* `NormalizeMinimal2` *is well defined.*
2. *Algorithm* `NormalizeMinimal2` *terminates.*
3. *The string E_3^* resulting from algorithm* `NormalizeMinimal2` *is a DNA expression in minimal normal form satisfying $E_3^* \equiv E_2^*$.*

Proof: We combine the proofs of Claims 1 and 3, because both of them (partly) rely on an invariant of the while-loop.

1, 3. The only differences between algorithm `NormalizeMinimal` and algorithm `NormalizeMinimal2` are in the while-loop. Hence, to prove Claims 1 and 3, it suffices to analyse this loop in `NormalizeMinimal2`.

The only instructions in the loop that are not obviously well defined, are the ones in lines 14 and 17. The substitution in line 14 requires m, the number of arguments

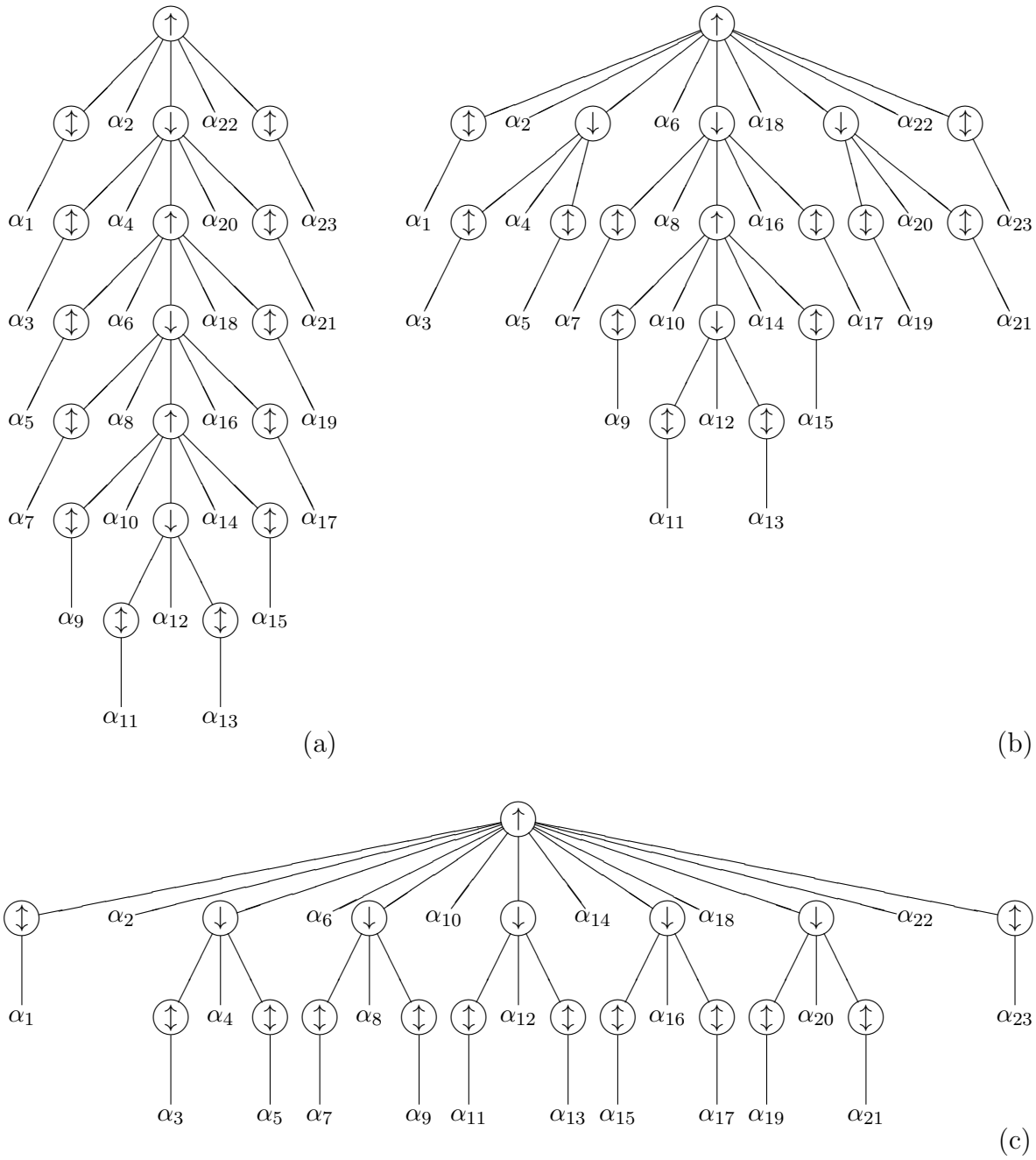


Figure 11.7: Structure trees of the three DNA expressions we successively obtain, when we apply algorithm `NormalizeMinimal2` to the ↑-expression E_6 from Example 11.8. To make the structure trees easier to compare, we have added subscripts to the occurring $\mathcal{N}$-words. (a) Structure tree of the original DNA expression E_6. The nodes in the backbone of the tree correspond in top-down order to E_6, E_5, E_4, E_3, E_2 and E_1, respectively. The third argument of E_6 is the ↓-expression E_5, which has in turn the ↑-expression E_4 as an argument. (b) Structure tree of the DNA expression after substituting E_5, according to line 14 of the algorithm. The fifth argument of the DNA expression is the ↓-expression E_3, which has in turn the ↑-expression E_2 as an argument. (c) Structure tree of the DNA expression after substituting E_3, according to line 14 of the algorithm. This is the final result of the algorithm.

of E_i, to be at least 2. The assignment in line 17 is only well defined if $\widehat{\varepsilon}$ is (still) an argument of E. We use an invariant of the while-loop to verify both requirements.

Before the first iteration of the loop, E has the same properties as in `NormalizeMinimal`. By Property (11.7) from the proof of Theorem 11.7, E is a minimal $\uparrow$-expression with Property ($\mathcal{D}_{\text{MinNF}}$.5), satisfying $E \equiv E_2^*$. We prove that the following, extended property is an invariant of the while-loop in `NormalizeMinimal2`:

$$\begin{array}{l} E \text{ is a minimal } \uparrow\text{-expression with Property } (\mathcal{D}_{\text{MinNF}}.5), \text{ satisfying} \\ E \equiv E_2^*, \widehat{\varepsilon} \text{ is an argument of } E \text{ and the arguments of } E \text{ to the left} \\ \text{of } \widehat{\varepsilon} \text{ do not contain any occurrence of } \uparrow. \end{array} \tag{11.11}$$

The fact that, according to this property, $\widehat{\varepsilon}$ is an argument of E, implies that line 17 of the algorithm is well defined.

- Initially, before the first iteration of the while-loop, $\widehat{\varepsilon}$ is the first argument of E. Hence, there are no arguments to the left of $\widehat{\varepsilon}$. This makes Property (11.11) valid.
- Suppose that Property (11.11) is valid before a certain iteration of the while-loop. In the iteration, we consider the argument $\widehat{\varepsilon}$ of E.

  We first examine the case that $\widehat{\varepsilon}$ is a $\downarrow$-expression with at least one $\uparrow$-argument. Let

  $$\widehat{\varepsilon} = \langle\downarrow \varepsilon_1 \dots \varepsilon_{i-1} \langle\uparrow \varepsilon_{i,1}\varepsilon_{i,2} \dots \varepsilon_{i,m-1}\varepsilon_{i,m}\rangle \varepsilon_{i+1} \dots \varepsilon_n\rangle$$

  for some $m, n \geq 1$ and $\mathcal{N}$-words and DNA expressions $\varepsilon_1, \dots, \varepsilon_{i-1}, \varepsilon_{i+1}, \dots, \varepsilon_n$, and $\varepsilon_{i,1}, \varepsilon_{i,2}, \dots, \varepsilon_{i,m-1}, \varepsilon_{i,m}$, where $E_i = \langle\uparrow \varepsilon_{i,1}\varepsilon_{i,2} \dots \varepsilon_{i,m-1}\varepsilon_{i,m}\rangle$ is the *first* $\uparrow$-argument of $\widehat{\varepsilon}$. Because E is minimal, so are its DNA subexpressions $\widehat{\varepsilon}$ and E_i.

  We zoom in on the $\uparrow$-argument E_i of $\widehat{\varepsilon}$. By Lemma 8.27(7), $m \geq 3$ and the first argument $\varepsilon_{i,1}$ is an $\updownarrow$-expression $\langle\updownarrow \alpha\rangle$ for an $\mathcal{N}$-word α. Then certainly $m \geq 2$, and the substitution in line 14 is well defined.

  We now consider $\widehat{\varepsilon}$ itself. By Corollary 8.2, each argument of $\widehat{\varepsilon}$ is either an $\mathcal{N}$-word α, or an $\updownarrow$-expression $\langle\updownarrow \alpha\rangle$ for an $\mathcal{N}$-word α, or an $\uparrow$-expression. Because E_i is the first $\uparrow$-argument of $\widehat{\varepsilon}$, the arguments $\varepsilon_1, \dots, \varepsilon_{i-1}$ are $\mathcal{N}$-words α or $\updownarrow$-expressions $\langle\updownarrow \alpha\rangle$.

  In line 14 of `NormalizeMinimal2`, we substitute $\widehat{\varepsilon}$ in E by the sequence of arguments

  $$\langle\downarrow \varepsilon_1 \dots \varepsilon_{i-1}\varepsilon_{i,1}\rangle \varepsilon_{i,2} \dots \varepsilon_{i,m-1} \langle\downarrow \varepsilon_{i,m}\varepsilon_{i+1} \dots \varepsilon_n\rangle .$$

  This substitution is of exactly the same type as the substitution in line 12 of `NormalizeMinimal`. Hence, we can reuse part of the proof of Theorem 11.7(1) and (3), and conclude that after the substitution, E is still a minimal $\uparrow$-expression with Property ($\mathcal{D}_{\text{MinNF}}$.5), satisfying $E \equiv E_2^*$.

  In line 15 of `NormalizeMinimal2`, we set $\widehat{\varepsilon}$ to $\varepsilon_{i,2}$, which is indeed an argument of E after the substitution. It follows from the above that the new argument $\langle\downarrow \varepsilon_1 \dots \varepsilon_{i-1}\varepsilon_{i,1}\rangle$ of E, which precedes $\widehat{\varepsilon} = \varepsilon_{i,2}$, does not contain any occurrence of $\uparrow$. Hence, Property (11.11) is also valid at the end of the iteration. This concludes the analysis for the case that $\widehat{\varepsilon}$ is a $\downarrow$-expression with at least one $\uparrow$-argument.

We subsequently examine the (simpler) case that $\widehat{\varepsilon}$ is not such a $\downarrow$-expression. Because in this case, E is not modified, it is still a minimal $\uparrow$-expression with Property ($\mathcal{D}_{\mathrm{MinNF}}$.5), satisfying $E \equiv E_2^*$, at the end of the iteration.

We first consider the subcase that $\widehat{\varepsilon}$ is a $\downarrow$-expression without $\uparrow$-arguments. Because E is minimal, so is $\widehat{\varepsilon}$. By Corollary 8.2, each argument of $\widehat{\varepsilon}$ is either an $\mathcal{N}$-word α or an $\updownarrow$-expression $\langle\updownarrow \alpha\rangle$ for an $\mathcal{N}$-word α. We also consider the subcase that $\widehat{\varepsilon}$ is not a $\downarrow$-expression, at all. By Corollary 8.2 (applied to E), $\widehat{\varepsilon}$ is either an $\mathcal{N}$-word α, or an $\updownarrow$-expression $\langle\updownarrow \alpha\rangle$ for an $\mathcal{N}$-word α.

In both subcases, $\widehat{\varepsilon}$ does not contain any occurrence of $\uparrow$. Hence, if $\widehat{\varepsilon}$ is not the last argument of E and it is set to the next argument of E (in line 17), then Property (11.11) is again valid. If, on the other hand, $\widehat{\varepsilon}$ is the last argument of E and $\widehat{\varepsilon}$ remains the same, then certainly Property (11.11) remains valid. The variable `stop` is set to true. Apparently, in this case, none of the arguments of E contains an occurrence of $\uparrow$.

In all cases, Property (11.11) is also valid at the end of the iteration.

Indeed, Property (11.11) is an invariant of the while-loop. After the last iteration of the loop, the variable `stop` is true. This implies that *in* the last iteration, $\widehat{\varepsilon}$ was not a $\downarrow$-expression with at least one $\uparrow$-argument, and $\widehat{\varepsilon}$ was the last argument of E. As we have just observed, at that point, none of the arguments of E contains an occurrence of $\uparrow$ any more. In other words, the $\uparrow$-expression E does not contain any inner occurrence of $\uparrow$ any more.

We can again reuse part of the proof of Theorem 11.7(1) and (3), and conclude that E is in minimal normal form. By the invariant, E satisfies $E \equiv E_2^*$. This carries over to E_3^*.

2. We prove that the number of iterations of the while-loop is bounded. By Property (11.11), during the while-loop, E is a minimal DNA expression, which satisfies $E_2^* \equiv E$. This implies in particular that the length $|E|$ of E is constant. Further, during the loop, $\widehat{\varepsilon}$ is an argument of E.

   Initially, before the first iteration of the loop, $\widehat{\varepsilon}$ is the first argument of E. Hence, there is no argument of E to the left of $\widehat{\varepsilon}$.

   As we have seen in the proof of Claims 1 and 3, in every iteration of the loop, either the number of arguments of E to the left of $\widehat{\varepsilon}$ increases by 1, or the variable `stop` is set to true. The latter occurs only once, in the final iteration. Clearly, the number of arguments to the left of $\widehat{\varepsilon}$ is bounded by the length of E. Because this length is constant, the number of arguments to the left of $\widehat{\varepsilon}$ can only increase a bounded number of times.

   Consequently, the number of iterations of the while-loop is bounded.

This completes the proof of Theorem 11.9. □

Recall that in Section 11.2, we have introduced algorithm `NormalizeMinimal` as the second step of a two-step algorithm. The purpose of this two-step algorithm is to rewrite arbitrary DNA expressions into the minimal normal form, and the first step consists of applying the recursive function `MakeMinimal`. In Section 9.1 and Section 9.3, we have proved the correctness of `MakeMinimal` and worked out the implementation details of this function. By

now, we have also proved the correctness of `NormalizeMinimal2`, which is an implementation of `NormalizeMinimal`. This implies that the total, two-step algorithm is correct:

Corollary 11.10 *Let E_1^* be an arbitrary DNA expression, let E_2^* be the result of applying the recursive function* `MakeMinimal` *to E_1^*, and let E_3^* be the result of applying algorithm* `NormalizeMinimal2` *to E_2^*. Then E_3^* is a DNA expression in minimal normal form satisfying $E_3^* \equiv E_1^*$.*

We proceed by examining the complexity of algorithm `NormalizeMinimal2`. During the while-loop of the algorithm, we traverse the DNA expression from left to right. Therefore, we may expect the time complexity to be linear. We prove that this is indeed the case.

In Section 9.3, we used a data structure with four specific features to prove that the recursive function `MakeMinimal` requires linear time. For `NormalizeMinimal2`, we use three of these features: the first, the second and the fourth feature.

First, we store the letters that a DNA expression E consists of in a doubly-linked list. Then we can insert letters at a given position, or remove letters from a given position in constant time.

Second, for each DNA subexpression of E, we connect the first letter (the opening bracket) to the last letter (the closing bracket). In addition, for each $\mathcal{N}$-word-argument of an operator, we connect the first letter to the last letter. Both types of connections are two-way: we can step directly from the first letter to the last letter and vice versa. These connections enable us to move from one end to the other end of a DNA subexpression or an $\mathcal{N}$-word-argument in constant time.[3]

Finally, for each operator $\uparrow$ or $\downarrow$ in E, we maintain a circular, doubly-linked list of its consecutive expression-arguments. This feature is not really crucial in the proof of the linear time complexity. We use it only in line 7 of `NormalizeMinimal2`, to check if E is alternating. As this test is performed only once, it would not harm if we had to traverse the entire DNA expression for this. That would cost only linear time. However, since we have already defined the lists of consecutive expression-arguments, we can as well use them again here. They allow us to do the test in line 7 in constant time, because E is alternating, if and only if the list of consecutive expression-arguments of its outermost operator is empty.

Note that for each inner occurrence of $\uparrow$ or $\downarrow$ in E, the list of consecutive expression-arguments is empty. Because E is minimal, it has all properties from Lemma 8.22. By Property ($\mathcal{D}_{\text{Min}}$.4), each inner occurrence of $\uparrow$ or $\downarrow$ in E is alternating.

Examples of the three features of the data structure and their usage are given in Section 9.3. In particular, Figure 9.16 and Figure 9.18 show all connections and lists for some example DNA expressions.

For a given DNA expression E, the connections can be initialized in linear time. For every basic operation (substitution) that is applied to E in the course of algorithm `NormalizeMinimal2`, the connections can be updated in constant time.

We finally observe that, unlike, e.g., the function `MakeMinimal`, algorithm `NormalizeMinimal2` is not recursive. When we apply it to a minimal DNA expression E_2^*, we do not have a cascade of calls of the algorithm, for different arguments. This implies that

[3] In Section 9.3, we described an additional type of connection. If the $\mathcal{N}$-word-arguments of an operator were not necessarily maximal $\mathcal{N}$-word occurrences, then we also connected the first letter and the last letter of every maximal $\mathcal{N}$-word occurrence in E. We do not need such connections now.

the time (and space) required for passing the parameter of `NormalizeMinimal2` is not an issue in the analysis of its complexity. [4]

We now have

Theorem 11.11 *Let E_2^* be an arbitrary minimal DNA expression. The time required by algorithm* `NormalizeMinimal2` *for E_2^* is linear in $|E_2^*|$.*

Proof: First, we observe that algorithm `NormalizeMinimal2` requires at least linear time in the worst case. Initializing the desired data structure already costs linear time, but even after that, it may take linear time to just read and check the DNA expression. For example, let α be an arbitrary $\mathcal{N}$-word, let $p \geq 1$, and let E_2^* be an $\uparrow$-expression with $2p$ arguments: the $\mathcal{N}$-word α, an $\updownarrow$-expression $\langle \updownarrow \alpha \rangle$, the $\mathcal{N}$-word α, an $\updownarrow$-expression $\langle \updownarrow \alpha \rangle$, etc. Hence,

$$E_2^* = \left\langle \uparrow \underbrace{\alpha \langle \updownarrow \alpha \rangle \ldots \alpha \langle \updownarrow \alpha \rangle}_{p \text{ times}} \right\rangle .$$

It is easily verified that E_2^* is in minimal normal form already, and that $|E_2^*| = 3 + p \cdot (3 + 2 \cdot |\alpha|)$, which is linear in p. The while-loop in `NormalizeMinimal2` has $2p$ iterations. In every iteration, we check one argument $\widehat{\varepsilon}$ of $E = E_2^*$, and move on to the next argument, without changing anything. This takes time which is linear in p and thus in the length $|E_2^*|$ of E_2^*.

It may be even more convincing to revisit Example 11.8. For the DNA expressions E_q considered there, `NormalizeMinimal2` performs a linear number of *rewriting steps*. Together, these steps require time that is linear in $|E_q|$.

We now prove that algorithm `NormalizeMinimal2` also requires *at most* linear time in the worst case. For an arbitrary minimal DNA expression E_2^*, let us use $T_{\mathrm{NM2}}(E_2^*)$ to denote the time required by algorithm `NormalizeMinimal2` for E_2^*.

We first zoom in on line 13 of `NormalizeMinimal2`, where we check if $\widehat{\varepsilon}$ is a $\downarrow$-expression with at least one $\uparrow$-argument. If so, then we need the first $\uparrow$-argument E_i as the centre for the substitution in line 14.

It is easy to decide if $\widehat{\varepsilon}$ is a $\downarrow$-expression. If this is the case, then we can check if it has an $\uparrow$-argument and (if necessary) determine E_i, by simply examining the arguments of $\widehat{\varepsilon}$ from left to right. Of course, we can stop this iteration, as soon as we encounter an $\uparrow$-argument, which then is E_i.

With this implementation of line 13 in mind, we define four constants, which are upper bounds on the time spent in specific parts of the algorithm:

c_1 is the maximum time required by `NormalizeMinimal2` for an $\updownarrow$-expression E_2^*.

Hence, c_1 is the maximum time required for executing lines 3–5 and 23 of the algorithm.

c_2 is the maximum time required by `NormalizeMinimal2` for an $\uparrow$-expression E_2^*, except the time spent in (the iterations of) the while-loop.

Hence, c_2 is the maximum time required for executing lines 3, 4, 6–11, 22, 23 and the first test of the condition of the while-loop in line 12 of the algorithm.

[4] In fact, we could easily establish that the time (and space) required for passing the parameter for a single application of the algorithm is constant, just like we have done for a call of `MakeMinimal` in Section 9.3.

c_3 is the maximum time required by `NormalizeMinimal2` for one iteration of the while-loop, except the time spent for examining the arguments of a $\downarrow$-expression $\widehat{\varepsilon}$, as described above.

Hence, c_3 is the maximum time required for executing part of line 13, lines 14–21 and one test of the condition of the while-loop in line 12 of the algorithm.

c_4 is the maximum time required by `NormalizeMinimal2` for examining one argument of a $\downarrow$-expression $\widehat{\varepsilon}$, in line 13 of the algorithm, as described above.

It follows from the observations made at the description of the three features of the data structure, that c_1, c_2, c_3 and c_4 are indeed constants.

Note that for a particular DNA expression, the time required by a part of the algorithm may be much less than specified by the corresponding constant. For example, if an argument $\widehat{\varepsilon}$ in line 13 is not a $\downarrow$-expression, then we certainly do not have to perform the substitution in line 14. We execute lines 16–19 instead, which probably costs less time. Then the total time required for this iteration of the while-loop is less than c_3.

Let the constant c^* be defined by

$$c^* = \max\left\{\frac{c_1}{4}, \frac{c_2}{3}, c_3, c_4\right\}.$$

If E_2^* is an $\updownarrow$-expression, then by Theorem 7.5, $E_2^* = \langle\updownarrow \alpha_1\rangle$ for an $\mathcal{N}$-word α_1. Because an $\mathcal{N}$-word has at least length 1, $|E_2^*| = 3 + |\alpha_1| \geq 4$. In this case,

$$T_{\mathrm{NM2}}(E_2^*) \leq c_1 \leq \frac{c_1}{4} \cdot |E_2^*| \leq c^* \cdot |E_2^*|,$$

where the last inequality follows from $c^* \geq \frac{c_1}{4}$.

From now on, we assume that E_2^* is an $\uparrow$-expression or a $\downarrow$-expression. We first analyse the effect of the while-loop on the 'working DNA expression' E. By Theorem 11.9(2), the number of iterations of the loop is finite, say it is N. As we have established in the proof of Theorem 11.9(1) and (3), throughout the while-loop, E is a minimal $\uparrow$-expression and $\widehat{\varepsilon}$ is an argument of E.

In the first iteration (in fact, *at the beginning of* the first iteration), $\widehat{\varepsilon}$ is the first argument of the $\uparrow$-expression E. As we have also seen in the proof of Theorem 11.9(1) and (3), in every iteration of the loop except the last one, the number of arguments of E to the left of $\widehat{\varepsilon}$ increases by 1. In fact, in the j^{th} iteration (with $1 \leq j \leq N-1$), we append an argument $\widehat{\varepsilon}_j$ to the sequence of arguments to the left of $\widehat{\varepsilon}$. In the last iteration, $\widehat{\varepsilon}$ is the last argument of E, and E is not modified any further.

Hence, in the successive iterations, E has the following shapes: $\langle\uparrow \widehat{\varepsilon}\ldots\rangle$, $\langle\uparrow \widehat{\varepsilon}_1\widehat{\varepsilon}\ldots\rangle$, $\langle\uparrow \widehat{\varepsilon}_1\widehat{\varepsilon}_2\widehat{\varepsilon}\ldots\rangle$, $\ldots$, $\langle\uparrow \widehat{\varepsilon}_1\widehat{\varepsilon}_2\ldots\widehat{\varepsilon}_{N-1}\widehat{\varepsilon}\rangle$. When we define $\widehat{\varepsilon}_N$ as the argument $\widehat{\varepsilon}$ in the last iteration, the DNA expression E_3^* resulting from algorithm `NormalizeMinimal2` equals $\langle\uparrow \widehat{\varepsilon}_1\widehat{\varepsilon}_2\ldots\widehat{\varepsilon}_{N-1}\widehat{\varepsilon}_N\rangle$.

We examine the time spent in the j^{th} iteration of the while-loop. Let us use T_j to denote this time.

- If, in this iteration, $\widehat{\varepsilon}$ is not a $\downarrow$-expression, then the iteration costs at most c_3 time and $\widehat{\varepsilon}_j = \widehat{\varepsilon}$. As $|\widehat{\varepsilon}_j| \geq 1$ (note that $\widehat{\varepsilon}_j = \widehat{\varepsilon}$ may an $\mathcal{N}$-word of length 1), we have

  $$T_j \leq c_3 \leq c_3 \cdot |\widehat{\varepsilon}_j| \leq c^* \cdot |\widehat{\varepsilon}_j|,$$

  where the last inequality follows from $c^* \geq c_3$.

- If $\widehat{\varepsilon}$ is a $\downarrow$-expression $\langle\downarrow \varepsilon_1 \dots \varepsilon_n\rangle$ for some $n \geq 1$ and $\mathcal{N}$-words and DNA expressions $\varepsilon_1, \dots, \varepsilon_n$, then we consider two subcases. If $\widehat{\varepsilon}$ does not have any $\uparrow$-argument, then we spend at most $c_4 \cdot n$ time on examining the n arguments of $\widehat{\varepsilon}$, which implies that

$$T_j \leq c_3 + c_4 \cdot n. \tag{11.12}$$

In this case, $\widehat{\varepsilon}_j = \widehat{\varepsilon} = \langle\downarrow \varepsilon_1 \dots \varepsilon_n\rangle$. By Lemma 8.27(6), $\widehat{\varepsilon}_j$ has at least one argument $\langle\updownarrow \alpha\rangle$ for an $\mathcal{N}$-word α. Hence, $\widehat{\varepsilon}_j$ contains at least two occurrences of operators (its outermost operator $\downarrow$ and $\updownarrow$), each of which is accompanied by its own opening bracket and closing bracket. This implies that $|\widehat{\varepsilon}_j| \geq 6 + n$, which is equivalent to $n \leq |\widehat{\varepsilon}_j| - 6$. When we combine this with (11.12), we obtain

$$T_j \leq c_3 + c_4 \cdot n \leq c_3 + c_4 \cdot (|\widehat{\varepsilon}_j| - 6) = c_4 \cdot |\widehat{\varepsilon}_j| + c_3 - 6c_4.$$

If, on the other hand, $\widehat{\varepsilon}$ does have an $\uparrow$-argument, then let $\varepsilon_i = E_i = \langle\uparrow \varepsilon_{i,1}\varepsilon_{i,2} \dots \varepsilon_{i,m-1}\varepsilon_{i,m}\rangle$ for some $m \geq 1$ and $\mathcal{N}$-words and DNA expressions $\varepsilon_{i,1}, \varepsilon_{i,2}, \dots, \varepsilon_{i,m-1}$, $\varepsilon_{i,m}$ be the first $\uparrow$-argument of $\widehat{\varepsilon}$. In order to find E_i, we have to examine i arguments of $\widehat{\varepsilon}$, This costs at most $c_4 \cdot i$ time, which implies that

$$T_j \leq c_3 + c_4 \cdot i. \tag{11.13}$$

In this case, $\widehat{\varepsilon}_j = \langle\downarrow \varepsilon_1 \dots \varepsilon_{i-1}\varepsilon_{i,1}\rangle$, which is a $\downarrow$-expression with i arguments. We can now proceed in the same way as in the previous subcase, and find that $i \leq |\widehat{\varepsilon}_j| - 6$. When we combine this with (11.13), we obtain

$$T_j \leq c_3 + c_4 \cdot i \leq c_3 + c_4 \cdot (|\widehat{\varepsilon}_j| - 6) = c_4 \cdot |\widehat{\varepsilon}_j| + c_3 - 6c_4.$$

In both subcases ($\widehat{\varepsilon}$ without or with an $\uparrow$-argument), we find that $T_j \leq c_4 \cdot |\widehat{\varepsilon}_j| + c_3 - 6c_4$.

Now, if $c_3 \leq 6c_4$, then

$$T_j \leq c_4 \cdot |\widehat{\varepsilon}_j| \leq c^* \cdot |\widehat{\varepsilon}_j|,$$

where the last inequality follows from $c^* \geq c_4$. If, on the other hand $c_3 > 6c_4$, which is equivalent to $c_4 < \frac{c_3}{6}$, then

$$T_j \leq c_3 + c_4 \cdot (|\widehat{\varepsilon}_j| - 6) < c_3 + \frac{c_3}{6} \cdot (|\widehat{\varepsilon}_j| - 6) = \frac{c_3}{6} \cdot |\widehat{\varepsilon}_j| < c_3 \cdot |\widehat{\varepsilon}_j| \leq c^* \cdot |\widehat{\varepsilon}_j|,$$

where the last inequality follows from $c^* \geq c_3$.

In each case, we have obtained that $T_j \leq c^* \cdot |\widehat{\varepsilon}_j|$. Now, it is not difficult to derive an upper bound on $T_{\mathrm{NM2}}(E_2^*)$:

$$\begin{aligned} T_{\mathrm{NM2}}(E_2^*) &\leq c_2 + T_1 + \dots + T_N \\ &\leq c^* \cdot 3 + c^* \cdot |\widehat{\varepsilon}_1| + \dots + c^* \cdot |\widehat{\varepsilon}_N| \\ &= c^* \cdot |\langle\uparrow \widehat{\varepsilon}_1 \dots \widehat{\varepsilon}_N\rangle| = c^* \cdot |E_3^*| = c^* \cdot |E_2^*|, \end{aligned}$$

where the second inequality follows from $c^* \geq \frac{c_2}{3}$, and the last equality follows from the fact that E_2^* and E_3^* are equivalent, minimal DNA expressions.

Indeed, the time required by `NormalizeMinimal2` is at most linear in the length $|E_2^*|$ of E_2^*. This completes the proof of Theorem 11.11. □

As part of Theorem 9.40, we established that the data structure we propose to carry out the recursive function `MakeMinimal` efficiently, has linear size. For `NormalizeMinimal2`, we only use part of this data structure: three of the four features. We obviously do not need more than linear space for this.

For each DNA expression, the first feature, the doubly-linked list containing the DNA expression, does require linear space. For the second feature and the fourth feature of the data structure, the space requirements depend on the DNA expression. We cannot reuse Example 9.39 to demonstrate that there exist inputs to `NormalizeMinimal2` for which these two features also require linear space. The DNA expressions E_p from that example are not minimal, which is required for `NormalizeMinimal2`. It is, however, not difficult to find an example that suits the current context.

Example 11.12 Let α be an arbitrary $\mathcal{N}$-word, and let E_p be defined by

$$E_p = \left\langle \uparrow \underbrace{\langle \updownarrow \alpha\rangle \langle \updownarrow \alpha\rangle \ldots \langle \updownarrow \alpha\rangle}_{p \text{ times}} \right\rangle \qquad (p \geq 2).$$

It is easy to see that for any $p \geq 2$, E_p is a minimal DNA expression, with $|E_p| = 3 + p \cdot (3 + |\alpha|) = 3 + 3p + p \cdot |\alpha|$ and $\mathcal{S}(E_p) = \binom{\alpha}{c(\alpha)} \underbrace{\triangle \binom{\alpha}{c(\alpha)} \triangle \ldots \binom{\alpha}{c(\alpha)}}_{p-1 \text{ times}}$. In fact, by Lemma 8.18(2), E_p is the *only* minimal DNA expression with this semantics, which implies in particular that E_p is in minimal normal form already. In addition, for any $p \geq 2$,

- E_p contains $p+1$ pairs of matching brackets. Hence, the second feature of the data structure requires $p+1$ connections (in both directions) between an opening bracket and the corresponding closing bracket.
- E_p contains p occurrences of the $\mathcal{N}$-word α (in fact, maximal $\mathcal{N}$-word occurrences), each of which serves as the argument of an operator $\updownarrow$. Hence, the second feature of the data structure requires p connections (in both directions) between the first letter and the last letter of such an $\mathcal{N}$-word-argument.
- the outermost operator $\uparrow$ of E_p has p arguments $\langle \updownarrow \alpha\rangle$, which are, in particular, consecutive expression-arguments. Hence, the fourth feature of the data structure requires a circular, doubly-linked list for this operator containing the last $p-1$ arguments (each of which is the second of two consecutive expression-arguments).

Both specified sets of connections require space that is linear in p, and thus in $|E_p|$. The same goes for the doubly-linked list.[5] ■

[5] The outermost (and only) operator $\uparrow$ in the DNA expressions E_p from this example does not have any non-$\updownarrow$-arguments. Hence, these DNA expressions would not be suitable to demonstrate that the third feature of the data structure we use to perform `MakeMinimal` efficiently, can really require linear space. For the sake of completeness, we like to mention that there do exist *minimal* DNA expressions for which *all four* features of the data structure require linear space. We leave it to the reader to verify that the

We conclude

Theorem 11.13 *Let E_2^* be an arbitrary minimal DNA expression. The space required by algorithm* `NormalizeMinimal2` *for E_2^* is linear in $|E_2^*|$.*

Hence, both the time complexity and the space complexity of `NormalizeMinimal2` are linear. By Corollary 9.38 and Theorem 9.40, the same holds for the time complexity and the space complexity of the recursive function `MakeMinimal`.

We can combine these complexities to find the complexity of the total two-step algorithm to rewrite a given DNA expression E_1^* into the normal form. We only need to realize that the output of the first step, the function `MakeMinimal`, is a *minimal* DNA expression E_2^* that is equivalent to E_1^*, which implies that $|E_2^*| \leq |E_1^*|$. Then the time and space required by the second step, algorithm `NormalizeMinimal2`, which are linear in $|E_2^*|$, are at most linear in $|E_1^*|$. We thus have

Theorem 11.14 *Let E_1^* be an arbitrary DNA expression. Both the time and the space required by the two-step algorithm to rewrite E_1^* into the minimal normal form are linear in $|E_1^*|$.*

Hence, the two-step algorithm is better than (a natural implementation of) the single-pass recursive function `MakeMinimalNF`. That function would also yield the normal form version of its input, but, by Theorem 11.3, would require at least quadratic time in the worst case.

DNA expressions

$$E_p = \left\langle \uparrow \underbrace{\alpha \langle\updownarrow \alpha\rangle \langle\updownarrow \alpha\rangle \ldots \alpha \langle\updownarrow \alpha\rangle \langle\updownarrow \alpha\rangle}_{p \text{ times}} \right\rangle \qquad (p \geq 1)$$

are an example of this.

Chapter 12

Conclusions and Directions for Future Research

In this thesis, we have introduced DNA expressions as a formal notation for DNA molecules that may contain nicks and gaps. However, there are (formal) DNA molecules that cannot be represented by our expressions, the ones with nicks in both strands. For each expressible formal DNA molecule, there are infinitely many DNA expressions denoting it. We have rigorously analysed the ones with minimal length, the minimal DNA expressions.

For this, we first derived lower bounds on the length of a DNA expression denoting a given formal DNA molecule. We subsequently described how to construct DNA expressions that achieve the lower bounds, and thus are minimal. We also proved that there do not exist minimal DNA expressions other than those obtained from the constructions. Minimal DNA expressions are characterized by six syntactic properties, which can easily be verified. This can be used to decide whether or not a given DNA expression is minimal.

As a combinatorial intermezzo, we determined the number of minimal DNA expressions denoting a given molecule. For almost all types of expressible formal DNA molecules, the number of minimal DNA expressions can be expressed in terms of the Catalan numbers.

We then described a recursive algorithm, which rewrites a given DNA expression into an equivalent, minimal DNA expression. This is useful, e.g., to save space for storing a description of the DNA molecule denoted. In the algorithm, step by step, the DNA expression acquires the six properties that characterize minimal DNA expressions.

We finally introduced a (minimal) normal form for DNA expressions: a well-defined set of properties such that for each expressible formal DNA molecule X, there is a unique DNA expression denoting X and satisfying those properties. Or actually, we *defined* the normal form DNA expressions as specific minimal DNA expressions, and we *proved* that these DNA expressions are characterized by five syntactic properties. We described a two-step algorithm, which computes the normal form version of a given DNA expression. This can be used, e.g., to decide if two DNA expressions are equivalent. The algorithm first rewrites its input into an equivalent, minimal DNA expression, using the recursive algorithm for minimality mentioned above. After that, it performs some additional rewriting steps to acquire the (remaining) properties of the minimal normal form.

Both the algorithm for minimality and the algorithm for the minimal normal form are elegant, because they do not refer to the semantics of the DNA expression involved. They consist of (local) string manipulations on the DNA expression itself. We proved that both algorithms are correct (they do what they are supposed to do), and that they require linear time and space.

Although the analysis of DNA expressions in this thesis is quite elaborate, one could think of some more aspects to examine. For example, one might consider the detection of submolecules: given two DNA expressions E_1 and E_2, is $\mathcal{S}(E_1)$ a formal DNA submolecule of $\mathcal{S}(E_2)$? Of course, this can easily be decided by computing $\mathcal{S}(E_1)$ and $\mathcal{S}(E_2)$, but can it also be done by local rearrangements at the level of the DNA expressions? Perhaps, the minimal normal form may be helpful for this.

The formal DNA molecules, which form the semantic basis of our notation, are a formalization of the double-word notation for DNA molecules. As mentioned in Section 2.2, a rotation of a double word by an angle of 180° yields another double word representing the same DNA molecule (see Figure 2.10). As it is, the two representations of the molecule have their own DNA expression in minimal normal form. One could argue that only one of the two should be the 'true' normal form representation of the molecule. In that case, the definition of the normal form should be adjusted.

An idea we have not worked out in detail in this thesis, is the alternative implementation of the recursive function `MakeMinimalNF`. We only described it in global terms at the end of Section 11.1. In the process of rewriting a given DNA expression into the minimal normal form, this implementation maintains two DNA expressions, which are (based on) operator-minimal $\uparrow$-expressions and operator-minimal $\downarrow$-expressions, respectively. It would be interesting to work out the details of this implementation, and verify that its complexity is linear, as opposed to that of a natural implementation of `MakeMinimalNF`.

As we have remarked in Section 4.1, the set of operators $\{\uparrow, \downarrow, \updownarrow\}$ that we consider is one of many possible choices. It could be an important research line for the future to investigate other notations for DNA molecules.

One could think of an extension of the current notation, such that all formal DNA molecules (also the ones with nicks in both strands) can be denoted. Perhaps, this can be achieved by a 'guard-operator', which protects a DNA submolecule from the effects of other operators. In particular, it may prevent a nick in one strand from being sealed by an operator introducing nicks in the other strand.

Another extension could be an operator that makes two formal DNA molecules with complementary sticky ends anneal. It would also be desirable to define operators that make it possible to denote DNA molecules with a variety of other 'imperfections' than nicks and gaps, such as, e.g., hairpin loops and circular strands, see Section 2.2.

Rather than extending the present notation, one could also consider to start with a completely new set of operators. As mentioned in Section 1.2, a possible motivation for considering formal notations for DNA molecules is to provide a formal calculus for the processing of DNA molecules. Such a calculus would have to contain operators that correspond to various biochemical operations on DNA molecules, as well as expressions for sorts of DNA molecules resulting from applications of these operators. This way, the expressions not only *denote* DNA molecules, but they also implicitly describe how to synthesize them from the basic elements A, C, G and T.

From the mathematical point of view, the set of operators acting on expressions denoting DNA molecules does not have to correspond exactly to the biochemical operations. However, one should be able to express such operations by suitable compositions of mathematical operations.

The DNA expressions from this thesis can be considered as a first step towards a formal calculus for DNA processing, including descriptions for more complex DNA molecules. To finally achieve that goal, there are many more steps to be taken.

Samenvatting

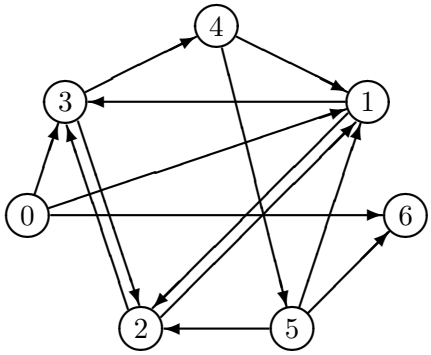
Veel eigenschappen van mensen, dieren en planten worden (gedeeltelijk) bepaald door hun genen. Voorbeelden van zulke eigenschappen bij de mens zijn bijvoorbeeld het geslacht, de kleur van de ogen, en de aanleg voor bepaalde ziektes. Genetische informatie is opgeslagen in DNA-moleculen, en een gen is een deel van een DNA-molecuul.

Iemands DNA is te vinden in vrijwel elke cel van het lichaam. DNA-moleculen kunnen echter ook los van een cel en los van een lichaam bestaan. Ze kunnen in een laboratorium gemaakt en verwerkt worden, en zijn in dat opzicht niet anders dan andere moleculen waar in laboratoria mee gewerkt wordt.

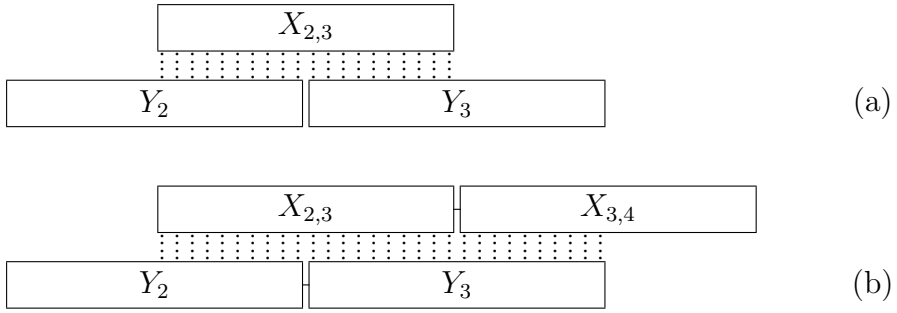
Hoewel onderzoek naar DNA van nature door moleculair biologen wordt uitgevoerd, zijn in de loop van de tijd ook informatici geïnteresseerd geraakt in het onderwerp. In het vakgebied *natural computing* worden algoritmes onderzocht, die geïnspireerd zijn door de natuur. Ook worden processen die voorkomen in de natuur, geïnterpreteerd en geanalyseerd als berekeningen.

Een tak van het veelomvattende gebied natural computing is *DNA computing*. Hier wordt onderzocht hoe DNA-moleculen gebruikt kunnen worden om berekeningen uit te voeren. Een concreet voorbeeld betreft een experiment van Leonard Adleman, die met behulp van DNA in een laboratorium een kleine instantie van het gerichte Hamiltonpad probleem oploste. Bij dit probleem (een variant van het handelsreizigersprobleem) is de vraag, of er in een gegeven gerichte graaf een pad bestaat van een gegeven beginknoop naar een gegeven eindknoop dat elke knoop precies één keer bezoekt. Bijvoorbeeld, in de graaf in Figuur 12.1 is er zo'n pad van knoop 0, via achtereenvolgens de knopen 1, 2, 3, 4 en 5 naar knoop 6.

Om het experiment van Adleman te kunnen beschrijven (en ook om het onderwerp van dit proefschrift te kunnen beschrijven), vertellen we eerst iets meer over DNA. DNA-moleculen zijn opgebouwd uit *nucleotiden*. Een belangrijk onderdeel van een nucleotide



Figuur 12.1: Graaf waarvoor Adleman het gerichte Hamiltonpad probleem oploste met behulp van DNA-moleculen.



Figuur 12.2: Losse DNA-strengen die samen een dubbelstrengs DNA-molecuul vormen dat overeenkomt met een pad in de graaf in Adlemans experiment. (a) Er ontstaan waterstofbruggen tussen strengen die knoop 2, pijl $(2, 3)$ en knoop 3 coderen. (b) Resultaat nadat ook een streng voor de pijl $(3, 4)$ in de graaf is vastgemaakt.

is de zogenaamde *base*. Er zijn vier verschillende mogelijke basen: adenine, cytosine, guanine en thymine, aangeduid met hun beginletters A, C, G en T. Omdat een nucleotide gekarakteriseerd wordt door zijn base, worden de vier letters ook wel gebruikt om een complete nucleotide aan te duiden.

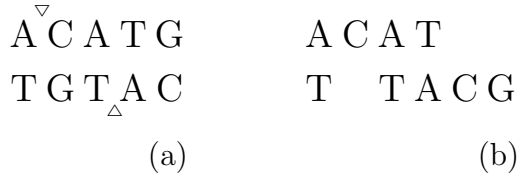
Nucleotiden kunnen met stevige *esterbindingen* aan elkaar gekoppeld worden en zo een lange streng vormen, bijvoorbeeld ACATG. Daarnaast kunnen twee basen (en daarmee: twee nucleotiden) met behulp van *waterstofbruggen* met elkaar verbonden worden. Hierdoor ontstaan zogeheten *basenparen*. Om precies te zijn: A kan met twee waterstofbruggen met T verbonden worden, en C kan met drie waterstofbruggen met G verbonden worden. Omdat deze waterstofbruggen zo specifiek zijn, worden A en T elkaars complement genoemd, net als C en G. Wanneer we twee DNA-strengen hebben met complementaire basen, kunnen ze een dubbelstrengs DNA-molecuul vormen, bijvoorbeeld $\begin{matrix}\text{ACATG}\\ \text{TGTAC}\end{matrix}$.[1]

De waterstofbruggen zijn veel zwakker dan de esterbindingen. Daardoor is het mogelijk om de strengen van een dubbelstrengs DNA-molecuul van elkaar te scheiden. In een organisme als de mens is dit ook van groot belang, bijvoorbeeld bij een proces als celdeling, waarbij iedere cel een eigen kopie van het DNA moet krijgen.

Adleman nu codeerde elke knoop in de graaf van Figuur 12.1 met een specifieke DNA-streng van twintig nucleotiden. Daarnaast codeerde hij ook de aanwezige pijlen in de graaf met DNA-strengen. Figuur 12.2 illustreert hoe hij dat precies deed. Laat Y_2 de streng van knoop 2 zijn en Y_3 de streng van knoop 3. Dan codeerde Adleman de pijl van knoop 2 naar knoop 3 met een streng $X_{2,3}$, die bestond uit het complement van de tweede helft van Y_2 en het complement van de eerste helft van Y_3. Met behulp van $X_{2,3}$ kunnen de strengen Y_2 en Y_3 aan elkaar gekoppeld worden. Na de aanhechting van $X_{3,4}$ kan vervolgens ook Y_4 (de streng voor knoop 4) erachter gehangen worden, enzovoort.

Adleman deed nu flinke hoeveelheden van de DNA-strengen $Y_0, Y_1, \ldots, Y_6$ en $X_{0,1}, X_{0,3}$, $X_{0,6}, \ldots, X_{5,6}$ (overeenkomend met alle knopen en pijlen in zijn graaf) bij elkaar, en liet ze onder de juiste condities met elkaar reageren. Na verloop van tijd onderzocht hij het resultaat. Daarbij stelde hij met de nodige biomoleculaire trucs vast dat er tussen alle gevormde dubbelstrengs DNA-moleculen, ook een molecuul was dat een pad codeerde van knoop 0 naar knoop 6, dat elke knoop precies één keer bevatte. Probleem opgelost.

[1] De strengen moeten ook tegengestelde *oriëntaties* hebben, maar daar gaan we in deze samenvatting verder niet op in.



Figuur 12.3: Twee afwijkingen van het standaard dubbelstrengs DNA-molecuul. (a) Een molecuul met twee nicks. (b) Een molecuul met twee gaps.

Hoewel er diverse kanttekeningen bij het experiment van Adleman zijn te plaatsen, toonde het aan dat je DNA-moleculen in principe kunt gebruiken om berekeningen uit te voeren. Bij deze en andere berekeningen met DNA is het belangrijk dat je de moleculen waarmee je begint en die er tijdens de berekening ontstaan precies kunt beschrijven. Ook als het moleculen met 'afwijkingen' betreft.

Een mogelijke afwijking is dat er een esterbinding ontbreekt tussen twee naast elkaar gelegen nucleotiden in dezelfde streng. De nucleotiden worden slechts bij elkaar gehouden doordat hun complementen (en de complementen van aangrenzende nucleotiden) in de andere streng met esterbindingen aan elkaar zitten. Zo'n ontbrekende esterbinding wordt een *nick* genoemd. Het molecuul in Figuur 12.3(a) kent twee nicks: een nick in de bovenste streng tussen de eerste A en de C, aangeduid met $^{\triangledown}$, en een nick in de onderste streng tussen de tweede T en de A, aangeduid met $_{\triangle}$.

Een andere mogelijke afwijking is dat een DNA-molecuul deels dubbelstrengs is en deels enkelstrengs: niet alle nucleotiden in de twee strengen zijn voorzien van hun complement, er zitten *gaps* in de strengen. Figuur 12.3(b) toont een voorbeeld van een DNA-molecuul met twee gaps. Er zijn nog vele andere afwijkingen van DNA-moleculen denkbaar, maar in dit proefschrift concentreren we ons op een formele notatie voor moleculen met nicks en gaps.

We onderzoeken DNA-expressies – expressies om moleculen te beschrijven die nicks en gaps kunnen bevatten. De feitelijke beschrijving van de resultaten hiervan begint na een inleiding tot het onderwerp (Hoofdstuk 1) en een hoofdstuk met benodigde voorkennis (Hoofdstuk 2), en valt uiteen in drie delen.

In Deel I kijken we naar DNA-expressies in het algemeen. Allereerst definiëren we in Hoofdstuk 3 *formele DNA-moleculen* – een formalisatie van DNA-moleculen. Deze vormen de semantische basis van onze expressies. Elke DNA-expressie zal als *semantiek* (formele betekenis) een formeel DNA-molecuul hebben.

In Hoofdstuk 4 introduceren we dan de DNA-expressies. Deze expressies zijn gebaseerd op de $\mathcal{N}$*-letters* A, C, G en T en drie operatoren $\uparrow$, $\downarrow$ en $\updownarrow$. De eenvoudigste DNA-expressies ontstaan wanneer we een operator toepassen op een $\mathcal{N}$*-woord*, dat wil zeggen: op een niet-lege string van $\mathcal{N}$-letters. We krijgen dan bijvoorbeeld $\langle\uparrow \mathrm{ACATG}\rangle$, $\langle\downarrow \mathrm{TGTAC}\rangle$ of $\langle\updownarrow \mathrm{ACATG}\rangle$ – de haakjes $\langle$ en $\rangle$ leggen vast tot hoever de operatoren effect hebben. Vervolgens kunnen we de operatoren ook toepassen op andere DNA-expressies. Ten slotte kunnen we de operatoren $\uparrow$ en $\downarrow$ toepassen op combinaties van $\mathcal{N}$-woorden en DNA-expressies.

Voor een DNA-expressie E noteren we de semantiek als $\mathcal{S}(E)$. Er geldt bijvoorbeeld:

$$\mathcal{S}(\langle\uparrow \mathrm{ACATG}\rangle) = \binom{\mathrm{ACATG}}{_} \quad \text{een bovenste DNA-streng,}$$

$$\mathcal{S}(\langle\downarrow \mathrm{TGTAC}\rangle) = \binom{_}{\mathrm{TGTAC}} \quad \text{een onderste DNA-streng,}$$

$$\begin{aligned}
\mathcal{S}(\langle\updownarrow \mathrm{ACATG}\rangle) &= \binom{\mathrm{ACATG}}{\mathrm{TGTAC}} && \text{een dubbelstrengs molecuul,}\\
\mathcal{S}(\langle\uparrow \langle\updownarrow \mathrm{A}\rangle \,\mathrm{C}\, \langle\downarrow \langle\updownarrow \mathrm{AT}\rangle \,\mathrm{CG}\rangle\rangle) &= \binom{\mathrm{A}}{\mathrm{T}}\binom{\mathrm{C}}{-}\binom{\mathrm{AT}}{\mathrm{TA}}\binom{-}{\mathrm{CG}} && \text{molecuul met twee gaps,}\\
\mathcal{S}(\langle\downarrow \langle\uparrow \mathrm{C}\, \langle\updownarrow \mathrm{AT}\rangle\rangle \langle\updownarrow \mathrm{G}\rangle\rangle) &= \binom{\mathrm{C}}{-}\binom{\mathrm{AT}}{\mathrm{TA}}^{\triangledown}\binom{\mathrm{G}}{\mathrm{C}} && \text{molecuul met gap en nick.}
\end{aligned}$$

Verschillende DNA-expressies kunnen hetzelfde formele DNA-molecuul beschrijven. Zulke expressies worden *equivalent* genoemd.

In Hoofdstuk 5 leiden we een aantal algemene resultaten over DNA-expressies af. We stellen bijvoorbeeld vast dat elk formeel DNA-molecuul X beschreven kan worden door een DNA-expressie, behalve als X nicks in zowel de bovenste als de onderste streng bevat. Verder bewijzen we een aantal resultaten over DNA-expressies die (bijna) equivalent zijn.

Deel II van dit proefschrift gaat over *minimale* DNA-expressies. Equivalente DNA-expressies kunnen namelijk verschillende lengtes hebben. Minimale DNA-expressies zijn de kortste expressies uit elke klasse van equivalente DNA-expressies. Anders gezegd: de DNA-expressies met minimale lengte voor een bepaald formeel DNA-molecuul.

In Hoofdstuk 6 bepalen we ondergrenzen voor de lengte van DNA-expressies E. Deze ondergrenzen worden uitgedrukt als functies van de semantiek $\mathcal{S}(E)$. We maken hierbij onderscheid tussen verschillende soorten DNA-expressies: expressies met $\uparrow$, expressies met $\downarrow$ en expressies met $\updownarrow$. Wanneer een DNA-expressie de betreffende ondergrens bereikt, weten we zeker dat er geen kortere DNA-expressie van hetzelfde soort voor hetzelfde molecuul bestaat.

Vervolgens beschrijven we in Hoofdstuk 7 hoe je voor een gegeven formeel DNA-molecuul een minimale DNA-expressie construeert. We doen dit voor alle mogelijke formele DNA-moleculen waarvoor DNA-expressies bestaan: eerst moleculen zonder gaps en nicks, vervolgens moleculen met gaps maar zonder nicks, en ten slotte moleculen met nicks (in een van de strengen).

In Hoofdstuk 8 tonen we aan dat elke minimale DNA-expressie is opgebouwd volgens een van de genoemde constructies; er bestaan dus geen andere. Om te kunnen zeggen of een DNA-expressie minimaal is, hoeven we niet expliciet haar lengte te controleren, of na te gaan of ze voldoet aan een van de constructies. We laten namelijk zien dat de minimale DNA-expressies gekarakteriseerd worden door zes *syntactische* eigenschappen, eigenschappen die je kunt controleren door puur naar de expressie als string te kijken, zonder de semantiek te bepalen. Hoewel er dus vaste constructies zijn voor minimale DNA-expressies, laten die constructies wel ruimte voor keuzes. Het gevolg daarvan is dat er voor veel formele DNA-moleculen meer dan één minimale DNA-expressie bestaat. Voor een gegeven molecuul berekenen we het aantal verschillende minimale DNA-expressies.

Wanneer een DNA-expressie E niet minimaal is, kun je benieuwd zijn naar een equivalente, minimale DNA-expressie. Je zou dan eerst de semantiek $\mathcal{S}(E)$ kunnen bepalen, en vervolgens de geëigende constructie kunnen toepassen die een minimale DNA-expressie oplevert. In dit proefschrift pakken we het anders aan. We beschrijven in Hoofdstuk 9 een recursief algoritme dat een gegeven DNA-expressie omschrijft naar een equivalent, minimaal exemplaar. Het algoritme past de oorspronkelijke DNA-expressie, met lokale transformaties, zó aan dat ze stap voor stap de zes eigenschappen krijgt die minimale DNA-expressies karakteriseren. Daarmee wordt de expressie minimaal. We tonen aan dat het algoritme lineaire tijd en lineair geheugen vereist, en dus efficiënt is. Daarnaast is het algoritme elegant, omdat het volledig op stringniveau opereert – het maakt geen gebruik van de semantiek, ook al zorgen we er natuurlijk wel over dat het resultaat equivalent is aan de oorspronkelijke DNA-expressie.

resultaat	korte beschrijving
Definitie 3.2 (p. 35)	formele DNA-moleculen
Definitie 4.1 (p. 47)	DNA-expressies
Stelling 5.5 (p. 81)	te beschrijven formele DNA-moleculen
Stelling 6.31 (p. 134)	ondergrens voor lengte DNA-expressies
Stelling 7.5 (p. 138)	minimale $\updownarrow$-expressies
Stelling 7.24 (p. 158)	constructie minimale, nickvrije $\uparrow$-expressies en $\downarrow$-expressies
Stelling 7.46 (p. 177)	constructie minimale $\uparrow$-expressies (en $\downarrow$-expressies) met nicks
Lemma 8.22 (p. 205), Stelling 8.26 (p. 211)	karakterisatie minimale DNA-expressies
Gevolg 8.47 (p. 232)	aantal minimale DNA-expressies
Figuur 9.15 (p. 285)	algoritme voor minimaliteit
Definitie 10.1 (p. 314)	minimale normaalvorm
Lemma 10.6 (p. 317), Stelling 10.8 (p. 322)	karakterisatie minimale normaalvorm
Figuur 11.6 (p. 356)	algoritme voor minimale normaalvorm

Tabel 12.1: Overzicht van belangrijkste resultaten uit het proefschrift.

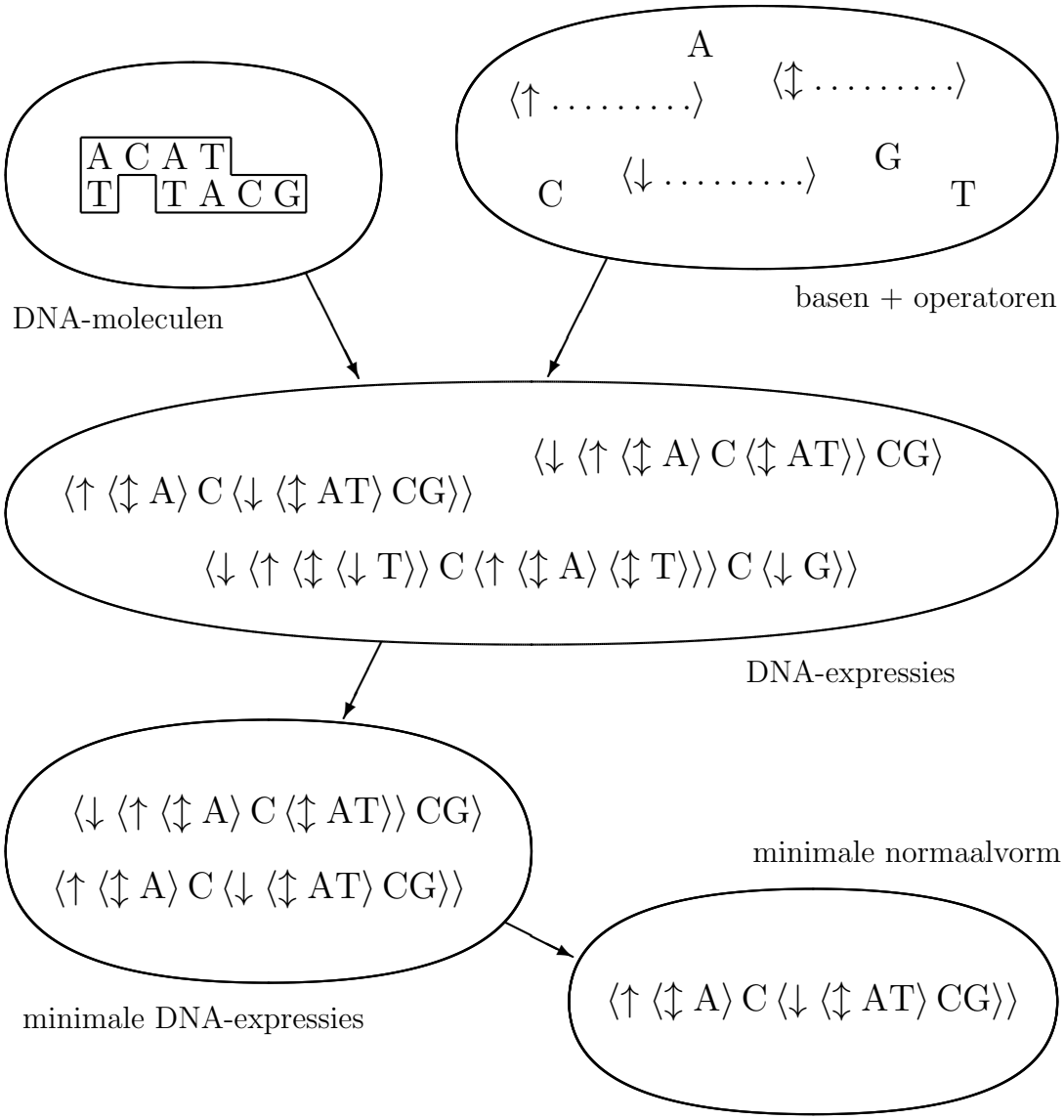
Deel III van dit proefschrift gaat over een *normaalvorm* voor DNA-expressies, een verzameling eigenschappen, zodat er voor elk molecuul precies één DNA-expressie bestaat die die eigenschappen heeft. We definiëren de normaalvorm in Hoofdstuk 10. Zoals gezegd, kunnen er in de constructie van een minimale DNA-expressie voor een gegeven formeel DNA-molecuul vaak keuzes gemaakt worden. We maken nu heel specifieke keuzes, zodat er precies één DNA-expressie overblijft. Dit noemen we de DNA-expressie in normaalvorm, en omdat het dus een minimale DNA-expressie is, spreken we van de *minimale normaalvorm*. Vervolgens tonen we aan dat alle DNA-expressies in minimale normaalvorm gekarakteriseerd worden door vijf syntactische eigenschappen.

Wanneer je voor een gegeven DNA-expressie E de equivalente DNA-expressie in minimale normaalvorm wil hebben, kun je de semantiek $\mathcal{S}(E)$ bepalen, en daarna de bijbehorende DNA-expressie in normaalvorm construeren. Opnieuw kiezen we in dit proefschrift een andere benadering, die geen gebruik maakt van de semantiek. In Hoofdstuk 11 beschrijven we een tweestapsalgoritme dat een gegeven DNA-expressie rechtstreeks omschrijft naar de equivalente DNA-expressie in minimale normaalvorm. De eerste stap bestaat eruit dat we de oorspronkelijke expressie met het recursieve algoritme uit Hoofdstuk 9 omschrijven in een equivalente, minimale DNA-expressie. In de tweede stap voeren we op het resultaat van de eerste stap een aantal lokale transformaties uit, die ervoor zorgen dat de expressie alle vijf de eigenschappen van de minimale normaalvorm krijgt. Ook dit tweestapsalgoritme vereist lineaire tijd en lineair geheugen.

We kunnen dit algoritme ook gebruiken om te bepalen of twee willekeurige DNA-expressies equivalent zijn. We schrijven dan eerst, met behulp van het algoritme, beide DNA-expressies om naar de minimale normaalvorm. Als dit twee identieke DNA-expressies oplevert (en alleen dan), zijn de oorspronkelijke DNA-expressies equivalent.

Aan het eind van dit proefschrift, in Hoofdstuk 12, trekken we de conclusies uit het onderzoek, en doen we enkele suggesties voor nader onderzoek.

Tabel 12.1 bevat een overzicht van de belangrijkste resultaten uit dit proefschrift. De inhoud van het proefschrift is ook schematisch weergegeven in Figuur 12.4. We kunnen



Figuur 12.4: Schematische weergave van de inhoud van het proefschrift.

deze figuur als volgt lezen. Om (formele) DNA-moleculen te beschrijven, gebruiken we letters voor de basen en operatoren $\uparrow$, $\downarrow$ en $\updownarrow$. Dit resulteert in DNA-expressies. Elk formeel DNA-molecuul kan beschreven worden door oneindig veel DNA-expressies. Sommige van deze DNA-expressies zijn korter dan andere. We richten ons op degene met minimale lengte, de minimale DNA-expressies. Er kunnen voor hetzelfde DNA-molecuul meerdere minimale DNA-expressies bestaan. Slechts één daarvan is in minimale normaalvorm.

Over de Auteur

Rudy van Vliet werd op 14 april 1969 geboren in Aarlanderveen, gemeente Alphen aan den Rijn. Hij behaalde in 1987 het diploma Atheneum b aan het Christelijk Lyceum in Alphen aan den Rijn. Daarna begon hij met de studies Wiskunde en Informatica aan (toen nog) de Rijksuniversiteit Leiden. In 1993 rondde hij de studie Informatica af, cum laude, met een afstudeerproject over neurale netwerken voor het meervoudige handelsreizigersprobleem, na een stage bij het Koninklijke Shell Exploratie en Productie Laboratorium in Rijswijk. In 1996 studeerde hij af in de wiskunde, eveneens cum laude, met een afstudeerproject over Hadamard matrices van het Williamson type.

Op 1 januari 1997 begon Rudy als promovendus in de Theoretische Informatica groep van professor Grzegorz Rozenberg bij het Leiden Institute of Advanced Computer Science, het informatica-instituut van de Leidse universiteit. Naast zijn onderzoek fungeerde hij als assistent bij verschillende colleges. Na afloop van zijn AIO-contract deed hij op ad-hoc-basis diverse onderwijsklussen bij LIACS. Tussen 2006 en 2010 had hij een kleine aanstelling als docent aan de Hogeschool Leiden. Vervolgens ging hij weer als docent bij LIACS aan de slag, en als zodanig is hij op dit moment nog werkzaam.

Tijdens zijn studietijd nam Rudy drie maal met een team deel aan de ACM Scholastic Programming Contest (tegenwoordig bekend als International Collegiate Programming Contest, ICPC), een internationale programmeerwedstrijd voor studententeams. In 1990 en 1992 resulteerde dat in deelname aan de World Finals, waar respectievelijk een zevende en een twaalfde plaats werd behaald. Toen de Benelus Algorithm Programming Contest, de regionale voorronde van de ICPC, in 2006, 2010 en 2015 in Leiden werd georganiseerd, hielp hij als lid (en de eerste twee keer voorzitter) van de jury mee bij het bedenken en uitwerken van de opgaven. Bijkomend resultaat van het jurylidmaatschap waren een wetenschappelijk artikel en twee stellingen bij dit proefschrift.

In zijn vrije tijd is Rudy actief voor Fietsersbond en kerk en houdt hij hobbymatig een aantal schapen. Tussen de bedrijven door voltooide hij dit proefschrift.

Dankwoord

Er zijn twee mensen die ik bij naam wil noemen voor hun (directe en indirecte) bijdrage aan dit proefschrift. Allereerst professor Grzegorz Rozenberg, voor zijn inspanningen in het najaar van 1996 om een promotieplaats voor mij te bewerkstelligen. Daarnaast natuurlijk Hendrik Jan Hoogeboom, mijn directe begeleider bij het promotieonderzoek. Hij was het die in januari 2004 voorstelde om eens een artikel over mijn onderzoek naar de conferentie DNA 10 te sturen. Daar werd ik op dat moment helemaal niet vrolijk van, maar als we dat niet hadden doorgezet, dan was er nu misschien wel een mooi boekje, maar waarschijnlijk geen proefschrift geweest. Behalve bij mijn onderzoek heb ik in de eerste jaren van mijn promotietraject ook in het onderwijs met Hendrik Jan mogen samenwerken, als assistent bij het college Datastructuren. Dat deze samenwerking soepel verliep, kwam wellicht mede doordat we een zelfde werkwijze hanteren: wanneer er bijvoorbeeld tentamens moeten worden verzonnen, huldigen wij beide de opvatting dat je daarmee het beste zo laat mogelijk kunt beginnen, omdat je dan het meest gericht, en daarmee het efficiëntst werkt. De strip van Casper en Hobbes voor in dit proefschrift is daar ook niet toevallig terechtgekomen: dezelfde strip sierde jarenlang de home page van Hendrik Jan bij LIACS.

Behalve deze twee personen wil ik ook mijn kamergenoten (door de jaren heen te veel om allemaal op te noemen), de mede-promovendi van de Theoretische Informatica Club van rond de eeuwwisseling, en de andere (oud-)collega's bij LIACS bedanken voor het creëren van een plezierige omgeving. Ik voel me thuis op de eerste verdieping van het Snellius.

Ten slotte een woord van dank voor LIACS als geheel, voor de ruime gelegenheid om te ontdekken dat ik onderwijs geven eigenlijk best leuk vind, iets wat ik negentien jaar geleden absoluut niet had gedacht. Het onderwijs heeft de voortgang van mijn promotie niet bevorderd, maar uiteindelijk dit proefschrift niet in de weg gestaan.

Hora est.

Bibliography

L.M. Adleman: Molecular computation of solutions to combinatorial problems, *Science* **266** (1994), 1021–1024. (Cited on pages 2, 27 and 29.)

L.M. Adleman: Computing with DNA, *Scientific American* **279**(2) (August 1998), 54–61. (Cited on page 27.)

D. Boneh, C. Dunworth, R.J. Lipton: Breaking DES using a molecular computer, *DNA Based Computers – Proceedings of a DIMACS Workshop, April 4, 1995, Princeton University*, DIMACS Series in Discrete Mathematics and Theoretical Computer Science **27** (R.J. Lipton, E.B. Baum, eds), American Mathematical Society (1996), 37–66. (Cited on pages 2, 3 and 29.)

J. Chen, N. Jonoska, G. Rozenberg (eds): *Nanotechnology: Science and Computation*, Natural Computing Series, Springer (2006). (Cited on page 2.)

N. Chomsky: A note on phrase structure grammars, *Information and Control*, **2**(4) (1959), 393–395. (Cited on page 17.)

D.I.A. Cohen: *Basic Techniques of Combinatorial Theory*, John Wiley & Sons (1978). (Cited on page 226.)

T.H. Cormen, C.E. Leiserson, R.L. Rivest: *Introduction to Algorithms*, The MIT Press and McGraw-Hill (1990). (Cited on page 88.)

R. Deaton, R.C. Murphy, M. Garzon, D.R. Franceschetti, S.E. Stevens, Jr.: Good encodings for DNA-based solutions to combinatorial problems, *DNA Based Computers II – DIMACS Workshop, June 10–12, 1996, Princeton University*, DIMACS Series in Discrete Mathematics and Theoretical Computer Science **44** (L.F. Landweber, E.B. Baum, eds), American Mathematical Society (1999), 247–258. (Cited on page 2.)

R. Deaton, R.C. Murphy, J.A. Rose, M. Garzon, D.R. Franceschetti, S.E. Stevens, Jr.: A DNA based implementation of an evolutionary search for good encodings for DNA computation, *Proceedings of 1997 IEEE International Conference on Evolutionary Computation (ICEC '97)*, IEEE (1997), 267–271. (Cited on page 29.)

A. Ehrenfeucht, T. Harju, I. Petre, D.M. Prescott, G. Rozenberg: *Computation in Living Cells – Gene Assembly in Ciliates*, Springer (2004). (Cited on page 2.)

W. Feller: *An Introduction to Probability Theory and Its Applications*, Vol. 1, 3rd ed., John Wiley & Sons (1968). (Cited on page 226.)

H. Gu, J. Chao, S.-J. Xiao, N.C. Seeman: A proximity-based programmable DNA nanoscale assembly line, *Nature* **465** (2010), 202–205. (Cited on page 2.)

M. Hall, Jr.: *Combinatorial Theory*, Blaisdell Publishing Company (1967). (Cited on page 221.)

J. Hartmanis: The structural complexity column: On the weight of computations, *Bulletin of European Association for Theoretical Computer Science* **55** (1995), 136–138. (Cited on page 29.)

T. Head: Formal language theory and DNA: an analysis of the generative capacity of specific recombinant behaviors, *Bulletin of Mathematical Biology* **49**(6) (1987), 737–759. (Cited on pages 1 and 25.)

T. Head, Gh. Păun, D. Pixton: Language theory and molecular genetics: generative mechanisms suggested by DNA recombination, *Handbook of Formal Languages* (G. Rozenberg, A. Salomaa, eds), Vol. 2, Springer (1997), 295–360. (Cited on pages 1 and 26.)

J.H. Holland: *Adaptation in Natural and Artificial Systems*, The University of Michigan Press (1975). (Cited on page 1.)

L. Kari: DNA computing: arrival of biological mathematics, *The Mathematical Intelligencer* **19**(2) (1997), 9–22. (Cited on page 2.)

L. Kari, S. Konstantinidis, P. Sosík: On properties of bond-free DNA languages, *Theoretical Computer Science* **334** (2005), 131–159. (Cited on pages 2 and 29.)

L. Kari, Gh. Păun, A. Salomaa: The power of restricted splicing with rules from a regular language, *Journal of Universal Computer Science* **2**(4) (1996), 224–240. (Cited on page 1.)

L. Kari, S. Seki, P. Sosik: DNA computing – Foundations and implications, *Handbook of Natural Computing* (G. Rozenberg, T. Bäck, J.N. Kok, eds), Vol. 3, Springer (2012), 1073–1127. (Cited on page 2.)

Z. Li: Algebraic properties of DNA operations, *Proceedings of the Fourth International Meeting on DNA Based Computers, University of Pennsylvania, Philadelphia, USA, June 15–19, 1998*, *BioSystems* **52** (L. Kari, H. Rubin, D.H. Wood, eds) (1999), 55–61. (Cited on page 3.)

J.F.J. Laros, A. Blavier, J.T. den Dunnen, P.E.M. Taschner: A formalized description of the standard human variant nomenclature in Extended Backus-Naur Form, *BMC Bioinformatics* **12**(Suppl 4):S5 (2011). (Cited on page 3.)

R.J. Lipton: DNA solution of hard computational problems, *Science* **268** (1995), 542–545. (Cited on page 29.)

S. Murata, S. Kobayashi (eds): *DNA Computing and Molecular Programming – 20th International Conference, DNA 20, Kyoto, Japan, September 22–26, 2014 – Proceedings*, Lecture Notes in Computer Science **8727**, Springer (2014). (Cited on page 2.)

Gh. Păun, G. Rozenberg, A. Salomaa: *DNA Computing – New Computing Paradigms*, Springer (1998). (Cited on page 2.)

A. Phillips, P. Yin (eds): *DNA Computing and Molecular Programming – 21th International Conference, DNA 21, Boston and Cambridge, MA, USA, August 17–21, 2015 – Proceedings*, Lecture Notes in Computer Science **9211**, Springer (2015). (Cited on page 2.)

D.M. Prescott: The DNA of ciliated protozoa, *Microbiological Reviews* **58**(2) (1994), 233–267. (Cited on page 25.)

E. Rivas, S.R. Eddy: The language of RNA: a formal grammar that includes pseudoknots, *Bioinformatics* **16**(4) (2000), 334–340. (Cited on page 3.)

P.W.K. Rothemund: Folding DNA to create nanoscale shapes and patterns, *Nature* **440** (2006), 297–302. (Cited on page 2.)

G. Rozenberg, T. Bäck, J.N. Kok (eds): *Handbook of Natural Computing*, Springer (2012). (Cited on page 1.)

J.L. Schroeder, F.R. Blattner: Formal description of a DNA oriented computer language, *Nucleic Acids Research* **10**(1) (1982), 69–84. (Cited on page 2.)

D.B. Searls: The linguistics of DNA, *American Scientist* **80** (1992), 579–591. (Cited on page 3.)

R.P. Stanley: *Catalan Numbers*, Cambridge University Press (2015). (Cited on pages 226 and 236.)

J.D. Watson, F.H.C. Crick: Molecular structure of nucleic acids, *Nature* **171** (1953), 737–738. (Cited on page 19.)

D. Whitley, A.M. Sutton: Genetic algorithms – A survey of models and methods, *Handbook of Natural Computing* (G. Rozenberg, T. Bäck, J.N. Kok, eds), Vol. 2, Springer (2012), 637–671. (Cited on page 1.)

E. Winfree: DNA computing by self-assembly, *The Bridge* **33**(4) (2003), 31–38. (Cited on page 2.)

D.Y. Zhang, G. Seelig: Dynamic DNA nanotechnology using strand-displacement reactions, *Nature Chemistry* **3** (2011), 103–113. (Cited on page 2.)

List of Symbols

symbol	introduced on page
$\longrightarrow$	13
$\Longrightarrow$	14
$\overset{i,j}{\Longrightarrow}$	70
$\sqsubseteq$	39
$\overline{\sqsubseteq}$	39
$\sqsubset$	39
$\equiv$	77
$\overline{\underset{\triangledown}{\equiv}}$	77
${}_{\triangledown}\!\equiv$	77
$\equiv_{\triangledown}$	77
$\uparrow$	3, 43, 44
$\downarrow$	3, 43, 45
$\updownarrow$	3, 43, 45
$\triangledown$	34
$\triangle$	34
$\#_a(X)$	9
$\#_{a,b}(X)$	9
1_K	11
α, α_i	33
α_E	97, 255
a, a_i	33
$\mathcal{A}$	34
$\lvert X\rvert_{\mathcal{A}}$	101
$\mathcal{A}_{\pm}$	34
$\mathcal{A}_{+}$	34
$\mathcal{A}_{-}$	34
$\mathcal{A}_{\triangledown\triangle}$	34
$B_{\uparrow}(X)$	110
$B_{\downarrow}(X)$	110
$c(a)$	33
C_p	226
$\delta(E)$	305
$\mathcal{D}$	43, 48
$\mathcal{D}_{\mathrm{Min}}$	209
$(\mathcal{D}_{\mathrm{Min}}.1)$–$(\mathcal{D}_{\mathrm{Min}}.6)$	205
$\mathcal{D}_{\mathrm{MinNF}}$	320
$(\mathcal{D}_{\mathrm{MinNF}}.1)$–$(\mathcal{D}_{\mathrm{MinNF}}.5)$	317
$\varepsilon, \varepsilon_i$	43
E, E_i	43
$E(\alpha_1, \ldots, \alpha_k)$	51, 97
E_i^*	61, 241
$E_{\mathrm{MinNF}}(X)$	314
$\mathrm{Exp}^{+}(\varepsilon)$	48
$\mathrm{Exp}^{-}(\varepsilon)$	48
$f_{\langle\rangle}(E)$	226
$f_{B_{\uparrow}}(\mathcal{P})$	221
$f_{B_{\downarrow}}(\mathcal{P})$	221
$\mathcal{F}$	35
G	13
G_1	67
G_2	326, 328
(i, j)	15
$\kappa(X)$	40
$\mathcal{K}$	10
λ	9
$L(X)$	9
$\mathcal{L}(G)$	14
$\mathcal{L}(X)$	14
$\mathcal{L}_G(X)$	14
$\nu(X)$	40
$\nu^{+}(X)$	40
$\nu^{-}(X)$	40
$n_{\uparrow\downarrow}(X)$	292
$n_{\updownarrow}(X)$	110
$n_{\alpha}(E)$	291
$n_{\alpha\uparrow\downarrow}(E)$	292
$n_{\alpha\updownarrow}(E)$	291
$n_{\mathrm{imus}}(X)$	142
$n_{\mathrm{iter}}(E)$	298
$n_{\mathrm{min}}(X)$	217
$n_{\mathrm{min}\uparrow}(X)$	217
$n_{\mathrm{min}\downarrow}(X)$	217
$n_{\mathrm{min}\updownarrow}(X)$	217
$n_{\mathrm{mus}}(X)$	142
$n_{\mathcal{N}\uparrow\downarrow}(E)$	292
$n_{\mathrm{opermin}\uparrow}(X)$	217
$n_{\mathrm{opermin}\downarrow}(X)$	217
$n_{\mathrm{opermin}\updownarrow}(X)$	217
$n_{\mathrm{pb}}(Z)$	224
$\mathcal{N}$	33
$\mathcal{O}(\lvert E\rvert)$	19
$\mathrm{Op}(E)$	305
P	13

$\mathcal{P}$ 153
$R(X)$ 9
xRy 18
Σ 9, 13
$\Sigma_{\mathcal{D}}$ 43
Σ^* 10
Σ^+ 10
S 13
$\mathcal{S}(E)$ 44
$\mathcal{S}^+(\varepsilon)$ 47
$\mathcal{S}^-(\varepsilon)$ 47
$(t_1, \ldots, t_r)$ 220
$T_{\mathrm{CS}}(E)$ 62
$T_{\mathrm{Dni}}(E)$ 307
$T_{\mathrm{M}\updownarrow\mathrm{M}}(E)$ 305
$T_{\mathrm{MM}}(E)$ 299
$T_{\mathrm{NM2}}(E)$ 362
V 13
$|X|$ 9
$X_1 \cdot X_2$ 9
(X_1, X_2) 10
$\overline{X}_j$ 151
$(X^s)^k$ 10

Index

term — introduced on page

↑-argument . . . 50
↑-block
  primitive ~ . . . 104
    ~ of ↑-component . . . 107
↑-component . . . 103
  maximal series of ~s . . . 104
↑-expression . . . 48
  alternating ~ . . . 51
↑-subexpression . . . 50
↓-argument . . . 50
↓-block
  primitive ~ . . . 105
    ~ of ↓-component . . . 107
↓-component . . . 103
  maximal series of ~s . . . 104
↓-expression . . . 48
  alternating ~ . . . 51
↓-subexpression . . . 50
↕-argument . . . 50
  non-~ . . . 51, 290
↕-expression . . . 48
↕-subexpression . . . 50
1′-site . . . 19
3′-end . . . 20
3′-overhang . . . 26
3′-site . . . 19
5′-end . . . 20
5′-overhang . . . 25
5′-site . . . 19
adenine . . . 19
Adleman, Leonard . . . 2, 27
$\mathcal{A}$-letter . . . 34
  double ~ . . . 34
  lower ~ . . . 34
  upper ~ . . . 34
algebraic structure . . . 11
algorithm . . . 18
  ~ for minimality . . . 4, 237
  ~ for semantics . . . 4, 58
  evolutionary ~ . . . 1
  genetic ~ . . . 1
  recursive ~
    for minimal normal form . . 5, 342
  two-step ~
    for minimal normal form . . 4, 348
alphabet . . . 9
alternating
  ~ ↑- or ↓-expression . . . 51
  ~ occurrence of ↑ or ↓ . . . 51
ambiguous grammar . . . 3, 15
ancestor
  ~ of node . . . 12
  ~ operator . . . 50
antisymmetric binary relation . . . 18
apply
  ~ operator . . . 43
  ~ production . . . 13
argument
  ↑-~ . . . 50
  ↓-~ . . . 50
  ↕-~ . . . 50
  ~ of DNA expression . . . 50
  ~ of operator . . . 43
  expression-~ . . . 50
  $\mathcal{N}$-word-~ . . . 50
assembly line
  DNA-based ~ . . . 2
associative operation . . . 9
$\mathcal{A}$-word . . . 34
  double ~ . . . 34
  lower ~ . . . 34
  upper ~ . . . 34
ballot sequence . . . 226
base . . . 19
  ~ pair . . . 21, 34
    complete ~ . . . 34
big O notation . . . 19
binary
  ~ relation . . . 18
    antisymmetric ~ . . . 18
    reflexive ~ . . . 18
    symmetric ~ . . . 18
    transitive ~ . . . 18

~ tree 88
block 141
lower ~ 151
~ partitioning 152, 154
primitive ~ 140
~ partitioning 140
primitive ↑-~ 104
~ of ↑-component 107
primitive ↓-~ 105
~ of ↓-component 107
primitive ~ 105
upper ~ 151
~ partitioning 153
primitive ~ 140
~ partitioning 141
brackets
sequence of well-nested
pairs of ~ 224
bulge 24
calculus for processing
of DNA molecules 2
Catalan numbers 226, 226, 236
characterization
~ of minimal
~ DNA expressions .. 4, 204, 211
~ normal form 5, 317, 322
~ of operator-minimal
DNA expressions 214
child of node 12
chromosome 19
ciliate 25
circular DNA molecule 25
closing bracket of operator 43
commutative composition
of functions 41
complement 21
~ function 33
Watson-Crick ~ 3, 21
complete base pair 34
complexity
exponential ~ 19
linear ~ 18
polynomial ~ 19
quadratic ~ 19
space ~ 18
~ of `ComputeSem` 65
~ of `MakeMinimal` 302
~ of `NormalizeMinimal2` 366
~ of two-step algorithm
for minimal normal form ... 366
time ~ 18
~ of `ComputeSem` 59, 62
~ of `MakeMinimal` 284, 299
~ of `MakeMinimalNF` 345
~ of `NormalizeMinimal2` 362
~ of two-step algorithm
for minimal normal form ... 366
component
~ of formal DNA molecule 37
↑-~ 103
↓-~ 103
double ~ 37
lower ~ 37
non-double ~ 37
single-stranded ~ 37
upper ~ 37
tree of types of ~s 37, 104
composition of functions
commutative ~ 41
computer
DNA-based ~ 2
`ComputeSem` 58, 63
space complexity of ~ 65
time complexity of ~ 59, 62
concatenation
~ of DNA expressions 56
~ of formal DNA molecules 40
~ of strings 9
~ of $\mathcal{N}$-words 33
conference on DNA computing ... 2, 29
contain
substring ~s other substring 10
context-free
~ grammar 13, 224
ambiguous ~ 15
start symbol of ~ 13
~ for $\mathcal{D}$ 4, 67
~ for $\mathcal{D}_{\text{MinNF}}$ 326
derivation in ~ 3, 14
leftmost ~ 15
language generated by ~ 14
language of ~ 14
non-terminal symbol of ~ 13
productions of ~ 13
self-embedding ~ 17, 339
sentential forms of ~ 13
start symbol of ~ 13
terminal symbol of ~ 13
unambiguous ~ 16
~ language 14
$\mathcal{D}$ is ~ 67, 73
counting function 103, 110

covalent bond 20
cover to the left/right 39
  strictly ~ 39
Crick, Francis 19
cytosine 19
data structure
  for `MakeMinimal` 284
  for `NormalizeMinimal2` 361
decomposition of formal
    DNA molecule 36
  nick free ~ 172
`Denickify` 263
deoxyribonucleic acid 19
depth first search walk 75
derivation 3, 14
  ~ tree 15
    yield of ~ 15
  leftmost ~ 15
descendant of node 12
directed
  ~ graph 11
  ~ Hamiltonian path problem . 2, 27
disjoint substrings 10
distance between nodes in tree 12
DNA-based
  assembly line 2
  computer 2
DNA computing 1, 25
  conference on ~ 2, 29
DNA expression 3, 43, 47
  argument of ~ 50
  concatenation of ~s 56
  ~ of tree 75
  equivalent ~s 3, 77
  length of ~ 101
  level of ~ 51
  minimal ~ 3, 137, 138, 158, 178
    algorithm for ~ 4, 237
    characterization of ~ . 4, 204, 211
    number of ~s ... 4, 183, 217, 232
  operator-minimal ~
    ..... 173, 175, 200, 345
    characterization of ~ 214
    number of ~s 220, 232
  semantics of ~ 44, 47
  structure tree of ~ 73
DNA mismatch 24
DNA molecule 1, 19
  calculus for processing of ~s 2
  circular ~ 25
  double-stranded ~ 3, 21
  formal ~ 3, 35
  perfect ~ 3, 23
  single-stranded ~ 3, 20
DNA origami 2
DNA subexpression 50
  proper ~ 50
DNA submolecule
  formal ~ 36
double
  ~ $\mathcal{A}$-letter 34
  ~ $\mathcal{A}$-word 34
  ~ complete formal
    DNA molecule 106
  ~ component of formal
    DNA molecule 37
  ~ helix 23
  ~-stranded DNA molecule ... 3, 21
  ~ word 3, 22
EcoRI 25
edge 11
empty string 9
end
  $3'$-~ 20
  $5'$-~ 20
  sticky ~ 24
endomorphism 11
enzyme
  restriction ~ 25
equivalence relation 18
equivalent DNA expressions 3, 77
  ~ modulo nicks 77
  ~ post-modulo nicks 77
  ~ pre-modulo nicks 77
  strictly ~ 77
evolutionary algorithm 1
exponential complexity 19
expressible formal
    DNA molecule 79, 81
expression
  $\uparrow$-~ 48
  $\downarrow$-~ 48
  $\updownarrow$-~ 48
  ~-argument 50
extension of relation 18
fit together 39
formal
  ~ DNA molecule 3, 35
    component of ~ 37
    concatenation of ~s 40
    decomposition of ~ 36
    double-complete ~ 106

expressible ~ 79, 81
lower strand of ~ 35
upper strand of ~ 35
~ DNA submolecule 36
Franklin, Rosalind 19
gap 3, 23
gene 19
genetic algorithm 1
govern 50
grammar 3, 13
context-free ~ 13, 224
ambiguous ~ 15
~ for $\mathcal{D}$ 4, 67
~ for $\mathcal{D}_{\text{MinNF}}$ 326
derivation in ~ 3, 14
leftmost ~ 15
language generated by ~ 14
language of ~ 14
non-terminal symbol of ~ 13
productions of ~ 13
self-embedding ~ 17, 339
sentential forms of ~ 13
start symbol of ~ 13
terminal symbol of ~ 13
unambiguous ~ 16
right-linear ~ 16
~ for $\mathcal{F}$ 36
graph 11
directed ~ 11
undirected ~ 11
guanine 19
hairpin loop 25
Hamiltonian path 27
directed ~ problem 2, 27
Head, Tom 1, 25
height
~ of rooted tree 13
~ of structure tree 74
~ in minimal normal form ... 324
helix
double ~ 23
homomorphism 11
hydrogen bond 21
hydroxyl group 19
identity of set with operation 11
include
substring ~s other substring 10
inner occurrence of operator 50
internal
~ maximal lower sequence 141
~ maximal upper sequence 140
~ node 12
intersecting substrings 10
invariants for ν^+, ν^-, ν 115
inverse relation 18
labelled
ordered, rooted, node-~ tree 13
language 1, 10
context-free ~ 14
$\mathcal{D}$ is ~ 67, 73
~ generated by
context-free grammar 14
~ of context-free grammar 14
regular ~ 16
$\mathcal{D}$ is no ~ 67
$\mathcal{D}_{\text{Min}}$ is no ~ 171
$\mathcal{D}_{\text{MinNF}}$ is ~ 5, 326, 340
$\mathcal{F}$ is ~ 36
pumping lemma for ~s
..... 17, 67, 171
leaf in tree 12
leftmost derivation 15
length
of DNA expression 101
lower bound on ~ 4, 101, 134
upper bound on ~ 101
of string 9
upper bound on ~ of
minimal DNA expression 168
letter 9
level
~ of DNA expression 51
~ of rooted tree 13
nesting ~ 51
maximal ~ 52
ligase 3, 20, 25
linear complexity 18
linked list 62, 284, 361
loop
hairpin ~ 25
lower
~ $\mathcal{A}$-letter 34
~ $\mathcal{A}$-word 34
~ block 151
~ partitioning 152, 154
primitive ~ 140
~ partitioning 140
~ bound
~ on length of
DNA expression 4, 101, 134
tight ~ 134
~ on number of

occurrences of operators ... 131
~ component of formal DNA molecule ... 37
~ nick letter ... 34
~ semantics of argument ... 48
~ strand ... 3, 35
maximal ~
~ prefix ... 141
~ sequence ... 141
internal ~ ... 141
~ suffix ... 141
`Make↕ExprMinimal` ... 256
`MakeMinimal` ... 239, 285
data structure for ~ ... 284
space complexity of ~ ... 302
time complexity of ~ ... 284, 299
`MakeMinimalNF` ... 342
time complexity of ~ ... 345
maximal
~ lower
~ prefix ... 141
~ sequence ... 141
internal ~ ... 141
~ suffix ... 141
~ $\mathcal{N}$-word occurrence ... 49
~ nesting level of DNA expression ... 52
~ in minimal normal form ... 323
~ series
~ of ↑-components ... 104
~ of ↓-components ... 104
~ upper
~ prefix ... 140
~ sequence ... 140
internal ~ ... 140
~ suffix ... 140
minimal
~ DNA expression ... 3, 137, 138, 158, 178
algorithm for ~ ... 4, 237
characterization of ~ ... 4, 204, 211
number of ~s ... 4, 183, 217, 232
operator-~ ... 173, 175, 200, 345
characterization of ~ ... 214
number of ~s ... 220, 232
upper bound on length of ~ ... 168
~ normal form ... 4, 314
characterization of ~ ... 5, 317, 322
recursive algorithm for ~ ... 5, 342
structure tree in ~ ... 324
two-step algorithm for ~ ... 4, 348
~ structure tree ... 215
mismatch
DNA ~ ... 24
modulo
equivalent ~ nicks ... 77
natural computing ... 1
nesting level of DNA expression ... 51
maximal ~ ... 52
~ in minimal normal form ... 323
nick ... 3, 23
equivalent modulo ~s ... 77
equivalent post-modulo ~s ... 77
equivalent pre-modulo ~s ... 77
~ free ... 35
~ decomposition ... 172
~ letter ... 34
lower ~ ... 34
upper ~ ... 34
$\mathcal{N}$-letter ... 33
node ... 11
ancestor of ~ ... 12
child of ~ ... 12
descendant of ~ ... 12
distance between ~s ... 12
internal ~ ... 12
ordered, rooted, ~-labelled tree ... 13
parent of ~ ... 12
non-↕-argument ... 51, 290
non-double component ... 37
non-root ... 12
non-terminal symbol ... 13
self-embedding ~ ... 17, 339
normal form ... 4, 247, 313
minimal ~ ... 4, 314
characterization of ~ ... 5, 317, 322
recursive algorithm for ~ ... 5, 342
structure tree in ~ ... 324
two-step algorithm for ~ ... 4, 348
`NormalizeMinimal` ... 348
`NormalizeMinimal2` ... 356
data structure for ~ ... 361
space complexity of ~ ... 366
time complexity of ~ ... 362
notation
big O ~ ... 19
simplified ~ ... 36
nucleobase ... 19
nucleotide ... 3, 19
number
~ of occurrences of operators

lower bound on ~ 131
~ in (operator-)minimal DNA expressions 198, 204
~ of minimal DNA expressions 4, 183, 217, 232
~ of operator-minimal ↑-/↓-expressions 220, 232
$\mathcal{N}$-word 33
concatenation of ~s 33
maximal ~ occurrence 49
~-argument 50
occurrence
alternating ~ of ↑ or ↓ 51
inner ~ of operator 50
maximal $\mathcal{N}$-word ~ 49
number of ~s of operators
lower bound on ~ 131
~ in (operator-)minimal DNA expressions 198, 204
~ of substring 10
preceding ~ 10
opening bracket of operator 43
operator 3, 43
ancestor ~ 50
apply ~ 43
argument of ~ 43
choice of ~s 46
closing bracket of ~ 43
inner occurrence of ~ 50
number of occurrences of ~s
lower bound on ~ 131
~ in (operator-)minimal DNA expressions 198, 204
opening bracket of ~ 43
~-minimal DNA expression 173, 175, 200, 345
characterization of ~ 214
number of ~s 220, 232
outermost ~ 50
parent ~ 50
scope of ~ 43
opposite orientation 21
ordered
~ partition 220
~, rooted
~, node-labelled tree 13
~ tree 13
orientation 3, 20
opposite ~ 21
origami
DNA ~ 2
outermost operator 50
overhang
3′-~ 26
5′-~ 25
overlapping substrings 10
overview of main results thesis 5
palindrome
(the $\mathcal{N}$-word) 33
(the DNA molecule) 22
parent
~ of node 12
~ operator 50
parse tree 15
partial order 18
partition
ordered ~ 220
partitioning
lower block ~ 152, 154
primitive ~ 140
upper block ~ 153
primitive ~ 141
perfect DNA molecule 3, 23
phosphate group 19
phosphodiester bond 20
polynomial complexity 19
post-modulo
equivalent ~ nicks 77
preceding occurrence of substring ... 10
prefit 39
~ by lower strands 39
~ by upper strands 39
prefix
maximal lower ~ 141
maximal upper ~ 140
~ of string 10
proper ~ 10
pre-modulo
equivalent ~ nicks 77
preorder walk 75
primitive
~ ↑-block 104
~ of ↑-component 107
~ ↓-block 105
~ of ↓-component 107
~ block 105
~ lower block 140
~ partitioning 140
~ upper block 140
~ partitioning 141
production 13
apply ~ 13

proper
~ DNA subexpression 50
~ prefix 10
~ substring 10
~ suffix 10
pumping lemma for regular languages 17, 67, 171
quadratic complexity 19
recursive function
~ `MakeMinimal` 239, 285
~ `MakeMinimalNF` 342
refinement 18
reflexive binary relation 18
regular language 16
$\mathcal{D}$ is no ~ 67
$\mathcal{D}_{\text{Min}}$ is no ~ 171
$\mathcal{D}_{\text{MinNF}}$ is ~ 5, 326, 340
$\mathcal{F}$ is ~ 36
pumping lemma for ~s . 17, 67, 171
relation
binary ~ 18
antisymmetric ~ 18
reflexive ~ 18
symmetric ~ 18
transitive ~ 18
equivalence ~ 18
inverse ~ 18
restriction enzyme 25
rewriting rule 13
right-linear
~ grammar 16
~ for $\mathcal{F}$ 36
RNA molecule 3, 22
root of tree 12
rooted
ordered, ~, node-labelled tree .. 13
~ tree 12
heigth of ~ 13
level of ~ 13
ordered, ~ 13
`RotateToMinimal` 272
rotation in tree 88
scope of operator 43
secondary structure of DNA/RNA ... 3
self-assembly 28
self-embedding
~ context-free grammar ... 17, 339
~ non-terminal symbol 17, 339
semantics
lower ~ of argument 48
~ of DNA expression 44, 47
algorithm for ~ 4, 58
upper ~ of argument 48
sentential form 13
sequence of well-nested pairs of brackets 224
simplified notation 36
single
~ strand 20
~-stranded component of formal DNA molecule 37
~-stranded DNA molecule ... 3, 20
$1'$-site 19
$3'$-site 19
$5'$-site 19
space complexity 18
~ of two-step algorithm for minimal normal form 366
~ of `ComputeSem` 65
~ of `MakeMinimal` 302
~ of `NormalizeMinimal2` 366
splicing system 25
start symbol 13
sticky end 24
strand 3, 20
lower ~ 3, 35
single ~ 20
upper ~ 3, 35
strictly
~ cover to the left/right 39
~ equivalent DNA expressions .. 77
string 1, 9
empty ~ 9
length of ~ 9
structure tree 73
height of ~ 74
~ in minimal normal form ... 324
minimal ~ 215
~ in minimal normal form 324
subexpression
$\uparrow$-~ 50
$\downarrow$-~ 50
$\updownarrow$-~ 50
substring 10
disjoint ~s 10
intersecting ~s 10
occurrence of ~ 10
overlapping ~s 10
proper ~ 10
~ contains other ~ 10
~ includes other ~ 10
subtree rooted in node 12

subword
maximal ~ 49
suffix
maximal lower ~ 141
maximal upper ~ 140
~ of string 10
proper ~ 10
sugar 19
symbol 9
symmetric binary relation 18
terminal symbol 13
non-~ 13
self-embedding ~ 17, 339
thymine 19
tight lower bound 134
time complexity 18
~ of two-step algorithm
for minimal normal form 366
~ of `ComputeSem` 59, 62
~ of `MakeMinimal` 284, 299
~ of `MakeMinimalNF` 345
~ of `NormalizeMinimal2` 362
transitive binary relation 18
tree 11
binary ~ 88
derivation ~ 15
yield of ~ 15
DNA expression of ~ 75
ordered, rooted, node-labelled ~ . 13
parse ~ 15
rooted ~ 12
height of ~ 13
leaf in ~ 12
level of ~ 13
ordered, ~ 13
root of ~ 12
rotation in ~ 88
structure ~ 73
minimal ~ 215
~ in minimal normal form . . . 324
~ of types
of components 37, 104
two-step algorithm for
minimal normal form 348
space complexity of ~ 366
time complexity of ~ 366
type
tree of ~s of components . . 37, 104
unambiguous
context-free grammar 16
undirected graph 11
upper
maximal ~
~ prefix 140
~ sequence 140
internal ~ 140
~ suffix 140
~ $\mathcal{A}$-letter 34
~ $\mathcal{A}$-word 34
~ block 151
~ partitioning 153
primitive ~ 140
~ partitioning 141
~ bound on length of
~ DNA expression 101
~ minimal DNA expression . 168
~ component of formal
DNA molecule 37
~ nick letter 34
~ semantics of argument 48
~ strand 3, 35
variable in context-free grammar 13
vertex 11
Watson, James 19
Watson-Crick complementarity . . . 3, 21
well-nested
sequence of ~ pairs
of brackets 224
Wilkins, Maurice 19
yield of derivation tree 15

Titles in the IPA Dissertation Series since 2009

M.H.G. Verhoef. *Modeling and Validating Distributed Embedded Real-Time Control Systems.* Faculty of Science, Mathematics and Computer Science, RU. 2009-01

M. de Mol. *Reasoning about Functional Programs: Sparkle, a proof assistant for Clean.* Faculty of Science, Mathematics and Computer Science, RU. 2009-02

M. Lormans. *Managing Requirements Evolution.* Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2009-03

M.P.W.J. van Osch. *Automated Model-based Testing of Hybrid Systems.* Faculty of Mathematics and Computer Science, TU/e. 2009-04

H. Sozer. *Architecting Fault-Tolerant Software Systems.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2009-05

M.J. van Weerdenburg. *Efficient Rewriting Techniques.* Faculty of Mathematics and Computer Science, TU/e. 2009-06

H.H. Hansen. *Coalgebraic Modelling: Applications in Automata Theory and Modal Logic.* Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2009-07

A. Mesbah. *Analysis and Testing of Ajax-based Single-page Web Applications.* Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2009-08

A.L. Rodriguez Yakushev. *Towards Getting Generic Programming Ready for Prime Time.* Faculty of Science, UU. 2009-9

K.R. Olmos Joffré. *Strategies for Context Sensitive Program Transformation.* Faculty of Science, UU. 2009-10

J.A.G.M. van den Berg. *Reasoning about Java programs in PVS using JML.* Faculty of Science, Mathematics and Computer Science, RU. 2009-11

M.G. Khatib. *MEMS-Based Storage Devices. Integration in Energy-Constrained Mobile Systems.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2009-12

S.G.M. Cornelissen. *Evaluating Dynamic Analysis Techniques for Program Comprehension.* Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2009-13

D. Bolzoni. *Revisiting Anomaly-based Network Intrusion Detection Systems.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2009-14

H.L. Jonker. *Security Matters: Privacy in Voting and Fairness in Digital Exchange.* Faculty of Mathematics and Computer Science, TU/e. 2009-15

M.R. Czenko. *TuLiP - Reshaping Trust Management.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2009-16

T. Chen. *Clocks, Dice and Processes.* Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2009-17

C. Kaliszyk. *Correctness and Availability: Building Computer Algebra on top of Proof Assistants and making Proof Assistants available over the Web.* Faculty of Science, Mathematics and Computer Science, RU. 2009-18

R.S.S. O'Connor. *Incompleteness & Completeness: Formalizing Logic and Analysis in Type Theory.* Faculty of Science, Mathematics and Computer Science, RU. 2009-19

B. Ploeger. *Improved Verification Methods for Concurrent Systems.* Faculty of Mathematics and Computer Science, TU/e. 2009-20

T. Han. *Diagnosis, Synthesis and Analysis of Probabilistic Models.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2009-21

R. Li. *Mixed-Integer Evolution Strategies for Parameter Optimization and Their Applications to Medical Image Analysis.* Faculty of Mathematics and Natural Sciences, UL. 2009-22

J.H.P. Kwisthout. *The Computational Complexity of Probabilistic Networks.* Faculty of Science, UU. 2009-23

T.K. Cocx. *Algorithmic Tools for Data-Oriented Law Enforcement.* Faculty of Mathematics and Natural Sciences, UL. 2009-24

A.I. Baars. *Embedded Compilers.* Faculty of Science, UU. 2009-25

M.A.C. Dekker. *Flexible Access Control for Dynamic Collaborative Environments.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2009-26

J.F.J. Laros. *Metrics and Visualisation for Crime Analysis and Genomics.* Faculty of Mathematics and Natural Sciences, UL. 2009-27

C.J. Boogerd. *Focusing Automatic Code Inspections.* Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2010-01

M.R. Neuhäußer. *Model Checking Nondeterministic and Randomly Timed Systems.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2010-02

J. Endrullis. *Termination and Productivity.* Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2010-03

T. Staijen. *Graph-Based Specification and Verification for Aspect-Oriented Languages.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2010-04

Y. Wang. *Epistemic Modelling and Protocol Dynamics.* Faculty of Science, UvA. 2010-05

J.K. Berendsen. *Abstraction, Prices and Probability in Model Checking Timed Automata.* Faculty of Science, Mathematics and Computer Science, RU. 2010-06

A. Nugroho. *The Effects of UML Modeling on the Quality of Software.* Faculty of Mathematics and Natural Sciences, UL. 2010-07

A. Silva. *Kleene Coalgebra.* Faculty of Science, Mathematics and Computer Science, RU. 2010-08

J.S. de Bruin. *Service-Oriented Discovery of Knowledge - Foundations, Implementations and Applications.* Faculty of Mathematics and Natural Sciences, UL. 2010-09

D. Costa. *Formal Models for Component Connectors.* Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2010-10

M.M. Jaghoori. *Time at Your Service: Schedulability Analysis of Real-Time and Distributed Services.* Faculty of Mathematics and Natural Sciences, UL. 2010-11

R. Bakhshi. *Gossiping Models: Formal Analysis of Epidemic Protocols.* Faculty of Sciences, Department of Computer Science, VUA. 2011-01

B.J. Arnoldus. *An Illumination of the Template Enigma: Software Code Generation with Templates.* Faculty of Mathematics and Computer Science, TU/e. 2011-02

E. Zambon. *Towards Optimal IT Availability Planning: Methods and Tools.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2011-03

L. Astefanoaei. *An Executable Theory of Multi-Agent Systems Refinement.* Faculty of Mathematics and Natural Sciences, UL. 2011-04

J. Proença. *Synchronous coordination of distributed components.* Faculty of Mathematics and Natural Sciences, UL. 2011-05

A. Moralı. *IT Architecture-Based Confidentiality Risk Assessment in Networks of Organizations.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2011-06

M. van der Bijl. *On changing models in Model-Based Testing.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2011-07

C. Krause. *Reconfigurable Component Connectors.* Faculty of Mathematics and Natural Sciences, UL. 2011-08

M.E. Andrés. *Quantitative Analysis of Information Leakage in Probabilistic and Nondeterministic Systems.* Faculty of Science, Mathematics and Computer Science, RU. 2011-09

M. Atif. *Formal Modeling and Verification of Distributed Failure Detectors.* Faculty of Mathematics and Computer Science, TU/e. 2011-10

P.J.A. van Tilburg. *From Computability to Executability – A process-theoretic view on automata theory.* Faculty of Mathematics and Computer Science, TU/e. 2011-11

Z. Protic. *Configuration management for models: Generic methods for model comparison and model co-evolution.* Faculty of Mathematics and Computer Science, TU/e. 2011-12

S. Georgievska. *Probability and Hiding in Concurrent Processes.* Faculty of Mathematics and Computer Science, TU/e. 2011-13

S. Malakuti. *Event Composition Model: Achieving Naturalness in Runtime Enforcement.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2011-14

M. Raffelsieper. *Cell Libraries and Verification.* Faculty of Mathematics and Computer Science, TU/e. 2011-15

C.P. Tsirogiannis. *Analysis of Flow and Visibility on Triangulated Terrains.* Faculty of Mathematics and Computer Science, TU/e. 2011-16

Y.-J. Moon. *Stochastic Models for Quality of Service of Component Connectors.* Faculty of Mathematics and Natural Sciences, UL. 2011-17

R. Middelkoop. *Capturing and Exploiting Abstract Views of States in OO Verification.* Faculty of Mathematics and Computer Science, TU/e. 2011-18

M.F. van Amstel. *Assessing and Improving the Quality of Model Transformations.* Faculty of Mathematics and Computer Science, TU/e. 2011-19

A.N. Tamalet. *Towards Correct Programs in Practice.* Faculty of Science, Mathematics and Computer Science, RU. 2011-20

H.J.S. Basten. *Ambiguity Detection for Programming Language Grammars.* Faculty of Science, UvA. 2011-21

M. Izadi. *Model Checking of Component Connectors.* Faculty of Mathematics and Natural Sciences, UL. 2011-22

L.C.L. Kats. *Building Blocks for Language Workbenches.* Faculty of Electrical

Engineering, Mathematics, and Computer Science, TUD. 2011-23

S. Kemper. *Modelling and Analysis of Real-Time Coordination Patterns.* Faculty of Mathematics and Natural Sciences, UL. 2011-24

J. Wang. *Spiking Neural P Systems.* Faculty of Mathematics and Natural Sciences, UL. 2011-25

A. Khosravi. *Optimal Geometric Data Structures.* Faculty of Mathematics and Computer Science, TU/e. 2012-01

A. Middelkoop. *Inference of Program Properties with Attribute Grammars, Revisited.* Faculty of Science, UU. 2012-02

Z. Hemel. *Methods and Techniques for the Design and Implementation of Domain-Specific Languages.* Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2012-03

T. Dimkov. *Alignment of Organizational Security Policies: Theory and Practice.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2012-04

S. Sedghi. *Towards Provably Secure Efficiently Searchable Encryption.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2012-05

F. Heidarian Dehkordi. *Studies on Verification of Wireless Sensor Networks and Abstraction Learning for System Inference.* Faculty of Science, Mathematics and Computer Science, RU. 2012-06

K. Verbeek. *Algorithms for Cartographic Visualization.* Faculty of Mathematics and Computer Science, TU/e. 2012-07

D.E. Nadales Agut. *A Compositional Interchange Format for Hybrid Systems: Design and Implementation.* Faculty of Mechanical Engineering, TU/e. 2012-08

H. Rahmani. *Analysis of Protein-Protein Interaction Networks by Means of Annotated Graph Mining Algorithms.* Faculty of Mathematics and Natural Sciences, UL. 2012-09

S.D. Vermolen. *Software Language Evolution.* Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2012-10

L.J.P. Engelen. *From Napkin Sketches to Reliable Software.* Faculty of Mathematics and Computer Science, TU/e. 2012-11

F.P.M. Stappers. *Bridging Formal Models – An Engineering Perspective.* Faculty of Mathematics and Computer Science, TU/e. 2012-12

W. Heijstek. *Software Architecture Design in Global and Model-Centric Software Development.* Faculty of Mathematics and Natural Sciences, UL. 2012-13

C. Kop. *Higher Order Termination.* Faculty of Sciences, Department of Computer Science, VUA. 2012-14

A. Osaiweran. *Formal Development of Control Software in the Medical Systems Domain.* Faculty of Mathematics and Computer Science, TU/e. 2012-15

W. Kuijper. *Compositional Synthesis of Safety Controllers.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2012-16

H. Beohar. *Refinement of Communication and States in Models of Embedded Systems.* Faculty of Mathematics and Computer Science, TU/e. 2013-01

G. Igna. *Performance Analysis of Real-Time Task Systems using Timed Automata.* Faculty of Science, Mathematics and Computer Science, RU. 2013-02

E. Zambon. *Abstract Graph Transformation – Theory and Practice.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2013-03

B. Lijnse. *TOP to the Rescue – Task-Oriented Programming for Incident Response Applications.* Faculty of Science, Mathematics and Computer Science, RU. 2013-04

G.T. de Koning Gans. *Outsmarting Smart Cards*. Faculty of Science, Mathematics and Computer Science, RU. 2013-05

M.S. Greiler. *Test Suite Comprehension for Modular and Dynamic Systems*. Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2013-06

L.E. Mamane. *Interactive mathematical documents: creation and presentation*. Faculty of Science, Mathematics and Computer Science, RU. 2013-07

M.M.H.P. van den Heuvel. *Composition and synchronization of real-time components upon one processor*. Faculty of Mathematics and Computer Science, TU/e. 2013-08

J. Businge. *Co-evolution of the Eclipse Framework and its Third-party Plug-ins*. Faculty of Mathematics and Computer Science, TU/e. 2013-09

S. van der Burg. *A Reference Architecture for Distributed Software Deployment*. Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2013-10

J.J.A. Keiren. *Advanced Reduction Techniques for Model Checking*. Faculty of Mathematics and Computer Science, TU/e. 2013-11

D.H.P. Gerrits. *Pushing and Pulling: Computing push plans for disk-shaped robots, and dynamic labelings for moving points*. Faculty of Mathematics and Computer Science, TU/e. 2013-12

M. Timmer. *Efficient Modelling, Generation and Analysis of Markov Automata*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2013-13

M.J.M. Roeloffzen. *Kinetic Data Structures in the Black-Box Model*. Faculty of Mathematics and Computer Science, TU/e. 2013-14

L. Lensink. *Applying Formal Methods in Software Development*. Faculty of Science, Mathematics and Computer Science, RU. 2013-15

C. Tankink. *Documentation and Formal Mathematics — Web Technology meets Proof Assistants*. Faculty of Science, Mathematics and Computer Science, RU. 2013-16

C. de Gouw. *Combining Monitoring with Run-time Assertion Checking*. Faculty of Mathematics and Natural Sciences, UL. 2013-17

J. van den Bos. *Gathering Evidence: Model-Driven Software Engineering in Automated Digital Forensics*. Faculty of Science, UvA. 2014-01

D. Hadziosmanovic. *The Process Matters: Cyber Security in Industrial Control Systems*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2014-02

A.J.P. Jeckmans. *Cryptographically-Enhanced Privacy for Recommender Systems*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2014-03

C.-P. Bezemer. *Performance Optimization of Multi-Tenant Software Systems*. Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2014-04

T.M. Ngo. *Qualitative and Quantitative Information Flow Analysis for Multi-threaded Programs*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2014-05

A.W. Laarman. *Scalable Multi-Core Model Checking*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2014-06

J. Winter. *Coalgebraic Characterizations of Automata-Theoretic Classes*. Faculty of Science, Mathematics and Computer Science, RU. 2014-07

W. Meulemans. *Similarity Measures and Algorithms for Cartographic Schemat-*

ization. Faculty of Mathematics and Computer Science, TU/e. 2014-08

A.F.E. Belinfante. *JTorX: Exploring Model-Based Testing*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2014-09

A.P. van der Meer. *Domain Specific Languages and their Type Systems*. Faculty of Mathematics and Computer Science, TU/e. 2014-10

B.N. Vasilescu. *Social Aspects of Collaboration in Online Software Communities*. Faculty of Mathematics and Computer Science, TU/e. 2014-11

F.D. Aarts. *Tomte: Bridging the Gap between Active Learning and Real-World Systems*. Faculty of Science, Mathematics and Computer Science, RU. 2014-12

N. Noroozi. *Improving Input-Output Conformance Testing Theories*. Faculty of Mathematics and Computer Science, TU/e. 2014-13

M. Helvensteijn. *Abstract Delta Modeling: Software Product Lines and Beyond*. Faculty of Mathematics and Natural Sciences, UL. 2014-14

P. Vullers. *Efficient Implementations of Attribute-based Credentials on Smart Cards*. Faculty of Science, Mathematics and Computer Science, RU. 2014-15

F.W. Takes. *Algorithms for Analyzing and Mining Real-World Graphs*. Faculty of Mathematics and Natural Sciences, UL. 2014-16

M.P. Schraagen. *Aspects of Record Linkage*. Faculty of Mathematics and Natural Sciences, UL. 2014-17

G. Alpár. *Attribute-Based Identity Management: Bridging the Cryptographic Design of ABCs with the Real World*. Faculty of Science, Mathematics and Computer Science, RU. 2015-01

A.J. van der Ploeg. *Efficient Abstractions for Visualization and Interaction*. Faculty of Science, UvA. 2015-02

R.J.M. Theunissen. *Supervisory Control in Health Care Systems*. Faculty of Mechanical Engineering, TU/e. 2015-03

T.V. Bui. *A Software Architecture for Body Area Sensor Networks: Flexibility and Trustworthiness*. Faculty of Mathematics and Computer Science, TU/e. 2015-04

A. Guzzi. *Supporting Developers' Teamwork from within the IDE*. Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2015-05

T. Espinha. *Web Service Growing Pains: Understanding Services and Their Clients*. Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2015-06

S. Dietzel. *Resilient In-network Aggregation for Vehicular Networks*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2015-07

E. Costante. *Privacy throughout the Data Cycle*. Faculty of Mathematics and Computer Science, TU/e. 2015-08

S. Cranen. *Getting the point — Obtaining and understanding fixpoints in model checking*. Faculty of Mathematics and Computer Science, TU/e. 2015-09

R. Verdult. *The (in)security of proprietary cryptography*. Faculty of Science, Mathematics and Computer Science, RU. 2015-10

J.E.J. de Ruiter. *Lessons learned in the analysis of the EMV and TLS security protocols*. Faculty of Science, Mathematics and Computer Science, RU. 2015-11

Y. Dajsuren. *On the Design of an Architecture Framework and Quality Evaluation for Automotive Software Systems*. Faculty of Mathematics and Computer Science, TU/e. 2015-12

J. Bransen. *On the Incremental Evaluation of Higher-Order Attribute Grammars*. Faculty of Science, UU. 2015-13

S. Picek. *Applications of Evolutionary Computation to Cryptology.* Faculty of Science, Mathematics and Computer Science, RU. 2015-14

C. Chen. *Automated Fault Localization for Service-Oriented Software Systems.* Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2015-15

S. te Brinke. *Developing Energy-Aware Software.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2015-16

R.W.J. Kersten. *Software Analysis Methods for Resource-Sensitive Systems.* Faculty of Science, Mathematics and Computer Science, RU. 2015-17

J.C. Rot. *Enhanced coinduction.* Faculty of Mathematics and Natural Sciences, UL. 2015-18

M. Stolikj. *Building Blocks for the Internet of Things.* Faculty of Mathematics and Computer Science, TU/e. 2015-19

D. Gebler. *Robust SOS Specifications of Probabilistic Processes.* Faculty of Sciences, Department of Computer Science, VUA. 2015-20

M. Zaharieva-Stojanovski. *Closer to Reliable Software: Verifying functional behaviour of concurrent programs.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2015-21

R.J. Krebbers. *The C standard formalized in Coq.* Faculty of Science, Mathematics and Computer Science, RU. 2015-22

R. van Vliet. *DNA Expressions – A Formal Notation for DNA.* Faculty of Mathematics and Natural Sciences, UL. 2015-23