Technical Report 2011-03 July 2011

Universiteit Leiden

Universiteit Leiden

Leiden Institute of Advanced Computer Science

All about a Minimal Normal Form for DNA Expressions

Rudy van Vliet

`rvvliet@liacs.nl`

Leiden Institute of Advanced Computer Science
Leiden University
Niels Bohrweg 1
2333 CA Leiden
The Netherlands

Technical Report 2011-03 July 2011

Universiteit Leiden

Universiteit Leiden

Leiden Institute of Advanced Computer Science

All about a Minimal Normal Form for DNA Expressions

Rudy van Vliet

rvvliet@liacs.nl

Leiden Institute of Advanced Computer Science
Leiden University
Niels Bohrweg 1
2333 CA Leiden
The Netherlands

Contents

Preface **V**

Abstract **VII**

1 Introduction **1**

2 Terminology and Notation **3**
- 2.1 Strings, $\mathcal{N}$-words, trees, grammars and complexity . . . 3
- 2.2 Formal DNA molecules . . . 9
- 2.3 Properties, relations and functions of formal DNA molecules . . . 12
- 2.4 Operators and DNA expressions . . . 13
- 2.5 Nesting level of the brackets . . . 18
- 2.6 The functions L and R for arguments of DNA expressions . . . 18
- 2.7 A context-free grammar for $\mathcal{D}$ . . . 19
- 2.8 The structure tree of a DNA expression . . . 23
- 2.9 Equivalent DNA expressions . . . 25

3 Basic Results on DNA Expressions **27**
- 3.1 Expressible formal DNA molecules . . . 27
- 3.2 Nick free DNA expressions . . . 28
- 3.3 Some equivalences . . . 28

4 The Length of a DNA Expression **37**
- 4.1 (Blocks of) components of a formal DNA molecule . . . 37
- 4.2 Lower bounds for the length of a DNA expression . . . 39

5 The Construction of Minimal DNA Expressions **41**
- 5.1 Minimal DNA expressions for a nick free formal DNA molecule . . . 41
- 5.2 Minimal DNA expressions for a formal DNA molecule with nick letters 52

6 All Minimal DNA Expressions **56**
- 6.1 Reverse construction of a minimal DNA expression . . . 56
- 6.2 Characterization of minimal DNA expressions . . . 61
- 6.3 The structure tree of a minimal DNA expression . . . 63
- 6.4 The number of (operator-)minimal DNA expressions . . . 65

7 An Algorithm for Minimality **68**
- 7.1 The algorithm and its correctness . . . 68
- 7.2 The algorithm for an example . . . 105
- 7.3 Detailed implementation and complexity of the algorithm . . . 114

7.4 Decrease of length by the algorithm 132

8 A Minimal Normal Form for DNA Expressions 140

8.1 Definition of the minimal normal form 141

8.2 Characterization of the minimal normal form 143

8.3 The structure tree of a DNA expression in minimal normal form 150

8.4 Regularity of $\boldsymbol{\mathcal{D}}_{\text{MinNF}}$ 151

9 Algorithms for the Minimal Normal Form 167

9.1 Recursive algorithm for the minimal normal form 167

9.2 Two-step algorithm for the minimal normal form 171

9.3 Implementation and complexity of the algorithm 179

10 Conclusions and directions for future research 192

Bibliography 193

List of Symbols 195

Index 197

Preface

In the summer of 2011, Rudy van Vliet and Hendrik Jan Hoogeboom prepared a paper entitled “A minimal normal form for DNA expressions”, and submitted it to the scientific journal *Fundamenta Informaticae*. As the title suggests, this paper presented a minimal normal form for DNA expressions. Moreover, it described an algorithm to rewrite an arbitrary DNA expression into the normal form. This is a two-step algorithm: it first rewrites the DNA expression into an equivalent, minimal DNA expression, and then rewrites the result of that into the normal form.

In the summer of 2012, after the paper had been reviewed by the journal, it was splitted into two papers, allowing for more detailed proofs of the results. The first paper, entitled “Making DNA expressions minimal”, describes the first step of the two-step algorithm, i.e., the algorithm to rewrite an arbitrary algorithm into an equivalent, minimal DNA expression. The second paper, entitled “A minimal normal form for DNA expressions”, describes the minimal normal form and an algorithm to rewrite an arbitrary minimal DNA expression into the normal form. The two new papers are self-contained. They were submitted together, as a diptych, to Fundamenta Informaticae, and were accepted for publication.

The interested reader of the papers may wish to see more details. Therefore, we compiled this report. In contains even more detailed proofs of the results from the papers (including auxiliary results, again with proofs), more examples illustrating the text and a section with a related topic that is not covered in the papers (§ 7.4).

The following table may serve as a quick reference list from definitions, examples, results, table and figures in the papers to their equivalents in this report:

In paper 1	In paper 2	In report
Definition 1		Definition 2.5
DNA expression		Definition 2.11
Theorem 2		Theorem 3.3
Theorem 3	Theorem 1	Theorem 5.3
Definition 4	Definition 2a	Definition 4.3, Definition 5.4
Definition 5	$B_\downarrow(X)$, $B_\uparrow(X)$	Definition 4.5
Definition 6	Definition 2b	Definition 5.8, Definition 5.9
	Example 3	Example 5.14
Theorem 7	Theorem 4	Theorem 5.12
Example 8	Example 5	Example 5.14
Example 9		Lemma 6.14(2)
Theorem 10	Theorem 6	Lemma 6.15, Theorem 6.16
Lemma 11	Lemma 7	Lemma 7.21
Example 12		not in this report
Lemma 13		Theorem 7.20

In paper 1	In paper 2	In report
Lemma 14		Theorem 7.24
Lemma 15		Theorem 7.27
Theorem 16		Theorem 7.17
Example 17		not in this report
Lemma 18		Lemma 7.34
Lemma 19		Lemma 7.36
Theorem 20		Theorem 7.37, Corollary 7.38, Theorem 7.40
	Definition 8	Definition 8.1
	Example 9	Example 8.3
	Theorem 10	Lemma 8.6, Lemma 8.7, Theorem 8.8
	language of minimal normal form is regular	§ 8.4
	Example 11	Example 9.2
	Example 12	Example 9.4
	Example 13	Example 9.7
	Theorem 14	Theorem 9.8
	Theorem 15	Theorem 9.10, Theorem 9.12
	Theorem 16	Theorem 9.13

In paper 1	In paper 2	In report
Figure 1	Figure 1	Figure 5.4
Table 1		Table 6.1
Figure 2		Figure 7.1, Figure 7.15
Figure 3		Figure 7.3
Figure 4		Figure 7.4
Figure 5	Figure 4	Figure 7.5
Figure 6		Figure 7.16, Figure 7.17, Figure 7.18
	Figure 2	Figure 9.1
	Figure 3	Figure 9.4, Figure 9.6

This report was first published in July 2011. This preface is the only part of the report that has been adjusted since then.

Rudy van Vliet
October 2012

Abstract

DNA expressions consitute a formal language/notation for DNA molecules that may contain nicks and gaps. Different DNA expressions may denote the same DNA molecule. We define a (minimal) normal form for this language and describe an algorithm to rewrite a given DNA expression into the normal form.

Chapter 1

Introduction

In the past two decades, DNA computing has become a flourishing research area. Since [Head, 1987] and [Adleman, 1994], 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. Nowadays, research groups from all over the world contribute to the field, see, e.g., [Deaton & Suyama, 2009] and [Sakakibara & Mi, 2011]. 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], [Reif, 2003], [Rothemund, 2006] and [Chen et al., 2006]. The basic concepts of DNA computing are described in [Paun et al., 1998].

Despite the growing interest in DNA computing, not much attention is paid in literature to formal ways to denote the DNA molecules – exceptions are [Boneh et al., 1996] and [Li, 1999]. Formal notations can, however, be useful, e.g., to precisely denote molecules and to compactly describe the computations carried out using them.

In [Van Vliet, 2004], [Van Vliet et al., 2005] and [Van Vliet et al., 2006], we have introduced *DNA expressions* as a formal notation for DNA molecules that may contain nicks (missing phosphodiester bonds between adjacent nucleotides in the same strand) and gaps (missing nucleotides in one of the strands). Different DNA expressions may denote the same DNA molecule. Such DNA expressions are called *equivalent*. In these three publications, it is also explained how to construct *minimal* DNA expressions: the shortest possible DNA expressions denoting a given molecule.

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. In this report, we present 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. We also 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. This approach is elegant, because it operates at the level of DNA expressions only, rather than to refer to the denoted DNA molecules.

The report is organized as follows. In Chapters 2–6, we recall a number of definitions and results which we have published before and which we need for the normal form and the algorithms. In particular, in Chapter 2, we introduce the concepts of a formal DNA molecule and a DNA expression. Chapter 3 contains some results on DNA expressions in general. Chapter 4 deals with (lower bounds on) the length of a DNA expression.

In Chapter 5, we describe how to construct minimal DNA expressions. In Chapter 6, we find out that there do not exist minimal DNA expressions other than the ones constructed in Chapter 5.

For every known definition or result in Chapters 2–6, we mention the corresponding definition or result in the earlier publications. We do not repeat the proofs for the old results, as they can simply be looked up, especially in [Van Vliet, 2004]. In addition, these five chapters contain some new, related results. For those, we do provide the proofs.

Because the contents of Chapters 2–6 are meant mainly as background material, we have not put much effort in presenting it as a nice, fluent story. This is different for Chapters 7–9, which describe the normal form and the algorithms.

In Chapter 7, we present an algorithm to rewrite an arbitrary DNA expression into an equivalent, minimal DNA expression. By itself, this is not sufficient to yield a normal form. For many DNA molecules, there exist many (equivalent) minimal DNA expressions. Depending on the input, the algorithm may yield each of these. However, the algorithm can function as a first step towards a true normal form.

Such normal form is introduced in Chapter 8. As the DNA expressions that satisfy the normal form are minimal, it is called a *minimal normal form*. In Chapter 9, we describe an algorithm for constructing this normal form. It first uses the algorithm from Chapter 7 to construct a minimal DNA expression, and then rewrites the result into the minimal normal form. This turns out to be more efficient than an alternative, direct algorithm.

Finally, in Chapter 10, we draw conclusions and suggest directions for future research.

Chapter 2

Terminology and Notation

2.1 Strings, $\mathcal{N}$-words, trees, grammars 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 $x_i \in \Sigma$ for $i = 1, 2, \dots, r$, the *length* of X is r and it is denoted by $|X|$. 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 by X_1X_2; sometimes, however, we will write $X_1 \cdot X_2$.

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^*.

Let $\mathcal{N} = \{\text{A, C, G, T}\}$ be the alphabet of nucleotides. The elements of $\mathcal{N}$ are called *$\mathcal{N}$-letters*. We reserve the symbol a (possibly with a subscript) to denote $\mathcal{N}$-letters. A *non-empty* string over $\mathcal{N}$ is called an *$\mathcal{N}$-word*. Clearly, 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.

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 $X = X_1X^sX_2$. 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 there exists a (possibly empty) string X_2 such that $X = X^sX_2$, then X^s is a *prefix* of X; if there exists a (possibly empty) string X_1 such that $X = X_1X^s$, 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.

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 $X = x_1x_2 \dots x_r$ 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 $X^s = a$ for a letter a from the alphabet Σ, then the number of occurrences of X^s in X is denoted by $\#_a(X)$. Obviously, when $X = x_1 x_2 \ldots x_r$ with $x_1, x_2, \ldots, x_r \in \Sigma$, $\#_a(X)$ is the number of x_i's that are equal to a. Sometimes, we are not so much interested in the number of occurrences of *one* 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 $\#_{a,b}(X)$.

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

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. For example, in the string $X = \text{ACATGAT}$ over the alphabet $\mathcal{N}$, the third occurrence of λ (the occurrence $(\text{AC}, \text{ATGAT})$) intersects with the (only) occurrence of CAT. In the remainder of this report, however, we will not come across intersections of λ with other strings. Occurrrences 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 $|Z_1| + |Z^s| \leq |Y_1| + |Y^s|$.

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 $X = \text{ACATGAT}$ over $\mathcal{N}$, the (only occurrence of the) substring $Y^s = \text{ATGA}$ 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.

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

Functions on strings

Let Σ be an alphabet. A function h from Σ^* to a set K with an operation $\circ$ is called a *homomorphism* if $h(X_1 X_2) = h(X_1) \circ h(X_2)$ for all $X_1, X_2 \in \Sigma^*$. Hence, to specify h if suffices to give its values for the letters from Σ.

The empty string λ is the *identity* 1_{Σ^*} of Σ^*, i.e., the element satisfying $X \circ 1_{\Sigma^*} = 1_{\Sigma^*} \circ X = X$ for all $X \in \Sigma^*$. It follows from the definition of a homomorphism that $h(\lambda) = 1_K$, where 1_K is the identity of K.

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.

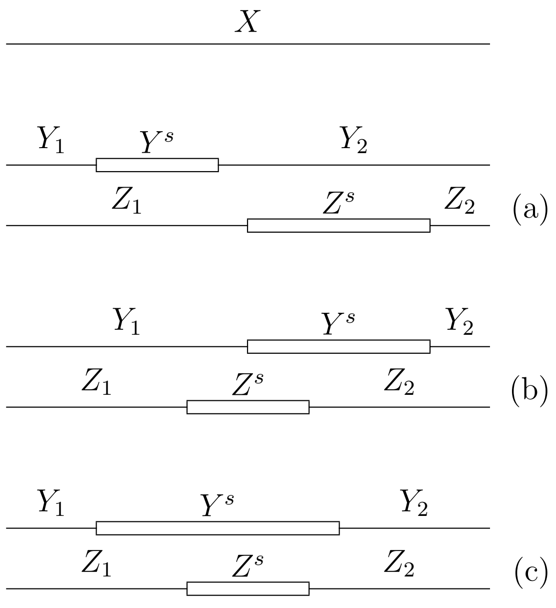


Figure 2.1: Examples of disjoint and intersecting occurrences (Y_1, Y_2) of Y^s and (Z_1, Z_2) of Z^s in a string X. (a) The occurrences are disjoint: $|Y_1| + |Y^s| \leq |Z_1|$. (b) The occurrences overlap: $|Z_1| < |Y_1| < |Z_1| + |Z^s| < |Y_1| + |Y^s|$. (c) The occurrence of Y^s contains the occurrence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

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 is concatenation), then h is called an *endomorphism*.

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

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

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

Directed trees

A *tree* is a non-empty graph such that for all nodes X and Y in the graph, there is exactly one path between X and Y. In particular, a tree is connected. Figure 2.2(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 2.2(a) is 3.

A *directed 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 that is not the root of the tree. Let X be a non-root in a directed tree. 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' 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 arcs connecting these nodes. A *leaf* in a directed tree is a node without descendants. Nodes that do have descendants are called *internal nodes*. We thus have two ways to

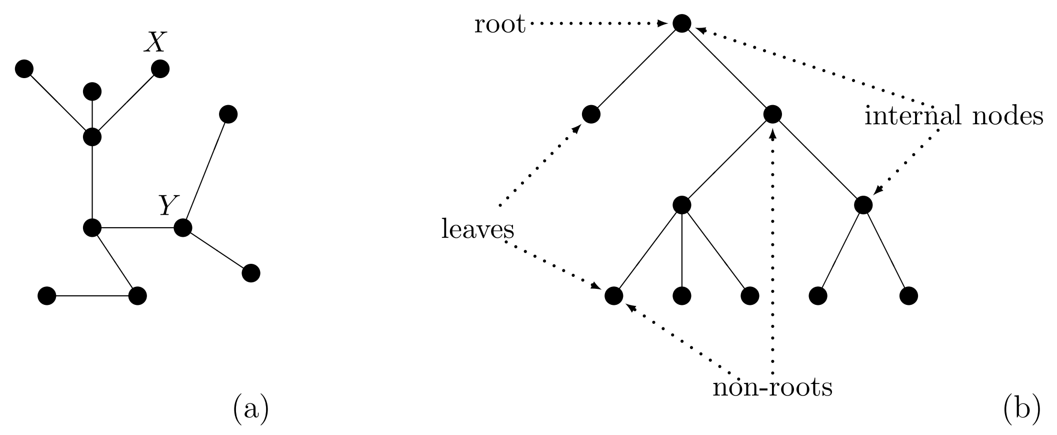


Figure 2.2: Examples of trees. (a) A tree with ten nodes. (b) A directed tree with ten nodes, in which the root and some non-roots, internal nodes and leaves have been indicated.

partition the nodes in a directed tree: either in a root and non-roots, or in leaves and internal nodes.

Usually, in a picture of a directed 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 2.2(b). In this example we have also indicated the root and some of the non-roots, internal nodes and leaves.

A *level* of a directed 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 directed tree is the maximal non-empty level of the tree. Obviously, this maximal level only contains leaves. For example, the height of the tree depicted in Figure 2.2(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 directed 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 directed tree is *ordered* if for each internal node X, the children of X are linearly ordered ('from left to right'). Finally, an *ordered, directed, node-labelled tree* is an ordered directed tree with labels at the nodes.

Grammars

A *grammar* is a set of rules that describe how the elements (strings) of a certain language can be derived from a certain initial symbol. We are in particular interested in context-free grammars and right-linear grammars.

A *context-free grammar* is a 4-tuple $G = (\Sigma, \Delta, P, S)$, where Σ is the total alphabet (the set of all symbols that may occur in an intermediate or final string in the grammar), Δ is the alphabet of *terminal symbols* (the set of symbols that may occur in the elements of the language described), P is a finite set of *productions* (rewriting rules for elements from $\Sigma \setminus \Delta$) and S is the *axiom* (the initial symbol). The elements of $\Sigma \setminus \Delta$ are called *non-terminal symbols*. Every production is of the form $A \longrightarrow Z$, where $A \in \Sigma \setminus \Delta$ and $Z \in \Sigma^*$. It allows for rewriting the non-terminal symbol A into the string Z over Σ (which may contain both terminal and non-terminal symbols).

Let (X_1, X_2) be an occurrence of the non-terminal symbol A in a string X over Σ. Hence, $X = X_1AX_2$. 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_1ZX_2.

A string that can be obtained from the axiom S by applying zero or more productions from P, is called a *sentential form*. In particular, the string S (containing only the axiom) 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 Δ that can be obtained from the axiom 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 Σ. 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 axiom S of the grammar, then we often simply speak of a *derivation of* Y (and not mention S).

For arbitrary strings X over Σ, the language $\mathcal{L}_G(X)$ is the set of all strings over Δ that can be derived in G from X: $\mathcal{L}_G(X) = \{Y \in \Delta^* \mid \text{there exists a derivation of } Y \text{ in } G \text{ from } X\}$. 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 $G = (\{S, A, B, a, b\}, \{a, b\}, P, S)$, where

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

A possible derivation in G is

$$\begin{array}{lll} S & \Longrightarrow & ASB \\ & \Longrightarrow & aSB \\ & \Longrightarrow & aASBB \\ & \Longrightarrow & aaSBB \\ & \Longrightarrow & aaBB \\ & \Longrightarrow & aabB \\ & \Longrightarrow & aabb. \end{array} \tag{2.1}$$

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

In this derivation, we successively applied the second, the third, the second, the third, the first, the fourth and once more the fourth 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

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

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 will often 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.

Example 2.3 Let G be the context-free grammar from Example 2.2. Another derivation of the string *aabb* in G is

$$\begin{array}{rcl} S & \Longrightarrow & ASB \\ & \Longrightarrow & AASBB \\ & \Longrightarrow & AASBb \\ & \Longrightarrow & aASBb \\ & \Longrightarrow & aASbb \\ & \Longrightarrow & aaSbb \\ & \Longrightarrow & aabb. \end{array} \tag{2.2}$$

■

If, in each step of a derivation, we expand the leftmost non-terminal symbol, then the derivation is called the *leftmost derivation*. Derivation (2.1) of *aabb* in our example context-free grammar is the leftmost derivation,

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 $A, B \in \Sigma \setminus \Delta$ and $a \in \Delta$. Hence, a production $A \longrightarrow aB$ allows for rewriting the non-terminal symbol A into a terminal symbol a followed by a non-terminal B.

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

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 Δ 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 Δ, 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. Clearly, a right-linear grammar is not self-embedding. Hence, 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 2.4 *A language $\mathcal{K}$ is regular, if and only if it can be generated by a context-free grammar that is not self-embedding.*

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 report, we describe 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 compexity* (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.

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 $|E|$ of E. 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 growth rate.

2.2 Formal DNA molecules

Every symbol in the upper strand of a double-stranded DNA molecule corresponds to a symbol in the lower strand. If there are no gaps, then two such corresponding symbols denote a base pair – two complementary nucleotides that are connected through a hydrogen bond. In the formal semantics of our DNA expressions, a pair of corresponding elements in the upper strand and the lower strand is denoted by a composite symbol $x = \binom{x^+}{x^-}$. Here x^+ stands for the nucleotide in the upper strand and x^- stands for the nucleotide in the lower strand. If we happen to have a gap in either of the strands, the missing nucleotide is denoted by $-$. Hence, $x^+, x^- \in \mathcal{N} \cup \{-\}$. 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 12 out of the 25 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 of these 12 composite symbols is denoted by $\mathcal{A}$.

For the 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 called *double $\mathcal{A}$-letters*, the elements of $\mathcal{A}_+$ are called *upper $\mathcal{A}$-letters*, and the elements of $\mathcal{A}_-$ are called *lower $\mathcal{A}$-letters.* Consequently, a non-empty string over $\mathcal{A}$ is called an *$\mathcal{A}$-word*, a non-empty string over $\mathcal{A}_\pm$ is called a *double $\mathcal{A}$-word*, a non-empty string over $\mathcal{A}_+$ is called an *upper $\mathcal{A}$-word*, and a non-empty string over $\mathcal{A}_-$ is called a *lower $\mathcal{A}$-word*.

We also need symbols to denote nicks. There are three possibilities for the connection structure of two adjacent base pairs in a double stranded DNA molecule: there can be a nick in the upper strand, there can be a nick in the lower strand, or there can be no nick at all between the base pairs. Note that there cannot be both a nick in the upper strand and a nick in the lower strand 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 the default; it is not denoted explicitly. A nick in the upper strand is denoted by $^\triangledown$ and a nick in the lower strand by $_\vartriangle$. We call $^\triangledown$ and $_\vartriangle$ the *nick letters* – $^\triangledown$ is the *upper* nick letter, and $_\vartriangle$ the *lower* nick letter.

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

Definition 2.5 (See [Van Vliet, 2004, Definition 2.1], [Van Vliet et al., 2005, Definition 1], [Van Vliet et al., 2006, Definition 1]) *A formal DNA molecule is a string $X = x_1 x_2 \ldots x_r$ with $r \geq 1$ and for $i = 1, \ldots, r$, $x_i \in \mathcal{A}_{\triangledown\vartriangle}$, satisfying*

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

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

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

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

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.

If a formal DNA molecule does not contain upper nick letters, then we say that its *upper strand is nick free*. Similarly, if a formal DNA molecule does not contain lower nick letters, then its *lower strand is nick free*. If a formal DNA molecule does not contain nick letters at all, then the molecule 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 2.5). With this in mind, it is easy to construct a right-linear grammar that generates the language $\mathcal{F}$. We thus have:

Lemma 2.6 *The language $\mathcal{F}$ of formal DNA molecules is regular.*

Components of a formal DNA molecule

Let $X = x_1 \dots x_r$ be a formal DNA molecule, with $x_i \in \mathcal{A}_{\triangledown\vartriangle}$ 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 2.7 *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 $L(X^s), R(X^s) \in \mathcal{A}$.*

Definition 2.8 (See [Van Vliet, 2004, Definition 2.3], [Van Vliet et al., 2005, Definition 2], [Van Vliet et al., 2006, page 130]) *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 $\mathcal{A}_{\triangledown\vartriangle}$ 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.*

Hence, the decomposition of X cannot be simplified any further. For the ease of notation, we will in general write $x'_1 \dots x'_k$ instead of $x'_1, \dots, x'_k$.

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.

Corollary 2.9 (See [Van Vliet, 2004, Corollary 2.5]) *Let X be a nick free formal DNA molecule 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 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.*

2.3 Properties, relations and functions of formal DNA molecules

Properties

Let $X = x_1 \ldots x_r$ be a formal DNA molecule, with $x_i \in \mathcal{A}_{\triangledown\vartriangle}$ for $i = 1, \ldots, r$. 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 2.5), x_r^+ is well defined. Intuitively, the upper strand extends at least as far to the right as the lower strand then.

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.

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 $L(X_2) \notin \mathcal{A}_-$; we also say that X_1 is an *upper prefit* for X_2 then. 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* (to be a *lower prefit* for X_2) 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 $X_1 \underline{\sqsubset} X_2$, we say that X_1 *prefits* X_2 or that X_1 is a *prefit* for X_2, and write then $X_1 \sqsubset X_2$.

If X_1 prefits X_2 (by upper/lower strands), then, from the perspective of X_2, we say that X_2 *postfits* X_1 (by upper/lower strands), or that X_2 is an (upper/lower) *postfit* for X_1.

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).

Functions

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

$$\nu^+(x) = \begin{cases} x & \text{if } x \in \mathcal{A} \cup \{{}_{\vartriangle}\} \\ \lambda & \text{if } x = {}^{\triangledown} \end{cases} \tag{2.3}$$

$$\nu^-(x) = \begin{cases} x & \text{if } x \in \mathcal{A} \cup \{{}^{\triangledown}\} \\ \lambda & \text{if } x = {}_{\vartriangle} \end{cases} \tag{2.4}$$

$$\nu(x) = \begin{cases} x & \text{if } x \in \mathcal{A} \\ \lambda & \text{if } x \in \{{}^{\triangledown}, {}_{\vartriangle}\} \end{cases} \tag{2.5}$$

$$\kappa(x) = \begin{cases} x & \text{if } x \in \mathcal{A}_{\pm} \cup \{^{\triangledown}, {}_{\vartriangle}\} \\ \binom{a}{c(a)} & \text{if } x = \binom{a}{-} \text{ for } a \in \mathcal{N} \\ \binom{c(a)}{a} & \text{if } x = \binom{-}{a} \text{ for } a \in \mathcal{N} \end{cases} \tag{2.6}$$

It is easy to see (by inspecting the effect of the functions on the symbols from $\mathcal{A}_{\triangledown\vartriangle}$), that applying the same function more than one time, does not change the result:

$$h(h(X)) = h(X) \text{ for each } h \in \{\nu^+, \nu^-, \nu, \kappa\} \text{ and } X \in \mathcal{A}^*_{\triangledown\vartriangle}. \tag{2.7}$$

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

Lemma 2.10 (See [Van Vliet, 2004, Lemma 2.7]) *For each formal DNA molecule X,*

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

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

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

2.4 Operators and DNA expressions

The formal DNA molecules constitute the foundation of our DNA language. They allow us to define the elements of the DNA language: the 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 report 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 the DNA language into the set of formal DNA molecules. When we precisely define the DNA expressions, we will also describe the corresponding semantics.

$$\mathcal{S}\left(\left\langle\uparrow \begin{array}{l}\text{C}\\\text{G}\end{array}\ \text{AT}\ \begin{array}{l}\text{G}\overset{\triangledown}{}\text{C}\\\text{CG}\end{array}\right\rangle\right) = \begin{array}{l}\text{CATGC}\\\text{G}\quad\text{CG}\end{array} \qquad \mathcal{S}\left(\left\langle\uparrow \begin{array}{l}\text{A}\\\text{T}\end{array}\ \begin{array}{l}\text{T}\\\text{A}\end{array}\right\rangle\right) = \begin{array}{l}\text{AT}\\\text{T}\underset{\vartriangle}{}\text{A}\end{array} \qquad \text{(a)}$$

$$\mathcal{S}\left(\left\langle\downarrow \text{T}\ \begin{array}{l}\text{CATGC}\\\text{G}\quad\text{CG}\end{array}\ \begin{array}{l}\text{AT}\\\text{T}\underset{\vartriangle}{}\text{A}\end{array}\right\rangle\right) = \begin{array}{l}\quad\text{CATGC}\overset{\triangledown}{}\text{AT}\\\text{TG}\quad\text{CGTA}\end{array} \qquad \text{(b)}$$

$$\mathcal{S}\left(\left\langle\updownarrow \begin{array}{l}\quad\text{CATGC}\overset{\triangledown}{}\text{AT}\\\text{TG}\quad\text{CGTA}\end{array}\right\rangle\right) = \begin{array}{l}\text{ACATGC}\overset{\triangledown}{}\text{AT}\\\text{TGTACGTA}\end{array} \qquad \text{(c)}$$

Figure 2.3: (See [Van Vliet, 2004, Figure 2.5], [Van Vliet et al., 2005, Figure 1], [Van Vliet et al., 2006, Figure 1]) Examples of the effects of the three operators. (a) The effect of the operator $\uparrow$. (b) The effect of the operator $\downarrow$. (c) The effect of the operator $\updownarrow$.

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 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 (semantical) 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 2.3(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 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 2.3(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 2.3(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 effect of this operator is illustrated in Figure 2.3(c).

Definition 2.11 (See [Van Vliet, 2004, Definition 2.8 and Definition 2.9], [Van Vliet et al., 2005, pages 378-380], [Van Vliet et al., 2006, pages 131-133]) *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

$$\mathcal{S}^+(\varepsilon) = \begin{cases} \binom{\alpha}{-} & \text{if } \varepsilon \text{ is an } \mathcal{N}\text{-word } \alpha \\ \mathcal{S}(\varepsilon) & \text{if } \varepsilon \text{ is a DNA expression} \end{cases} . \tag{2.8}$$

  Further,

$$\mathcal{S}(\langle\uparrow \varepsilon_1\varepsilon_2\ldots\varepsilon_n\rangle) = \nu^+(\mathcal{S}^+(\varepsilon_1))y_1\nu^+(\mathcal{S}^+(\varepsilon_2))y_2\ldots y_{n-1}\nu^+(\mathcal{S}^+(\varepsilon_n)) \tag{2.9}$$

  with

$$y_i = \begin{cases} \vartriangle & \text{if } \mathcal{S}^+(\varepsilon_i)\underline{\sqsubset}\mathcal{S}^+(\varepsilon_{i+1}), \text{ i.e., if both } R(\mathcal{S}^+(\varepsilon_i)) \in \mathcal{A}_\pm \\ & \text{and } L(\mathcal{S}^+(\varepsilon_{i+1})) \in \mathcal{A}_\pm \\ \lambda & \text{otherwise, i.e., if either } R(\mathcal{S}^+(\varepsilon_i)) \in \mathcal{A}_+ \\ & \text{or } L(\mathcal{S}^+(\varepsilon_{i+1})) \in \mathcal{A}_+ \text{ (or both)} \end{cases}$$
$$(i = 1, 2, \ldots, n-1). \tag{2.10}$$

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

$$\mathcal{S}^-(\varepsilon) = \begin{cases} \binom{-}{\alpha} & \text{if } \varepsilon \text{ is an } \mathcal{N}\text{-word } \alpha \\ \mathcal{S}(\varepsilon) & \text{if } \varepsilon \text{ is a DNA expression} \end{cases} . \tag{2.11}$$

  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.

Example 2.12 (See [Van Vliet, 2004, Equation (2.17)]) (Cf. [Van Vliet et al., 2005, Equation (4)], [Van Vliet et al., 2006, Equation (4)]) 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 2.3(b). ■

We call a DNA expression of the form $\langle \uparrow \varepsilon_1 \dots \varepsilon_n \rangle$ an $\uparrow$-*expression*, one of the form $\langle \downarrow \varepsilon_1 \dots \varepsilon_n \rangle$ a $\downarrow$-*expression*, and one of the form $\langle \updownarrow \varepsilon_1 \rangle$ an $\updownarrow$-*expression*. Hence, the DNA expression in Example 2.12 is a $\downarrow$-expression.

Theorem 2.13 (See [Van Vliet, 2004, Theorem 2.10]) *Let $E = \langle \uparrow \varepsilon_1 \dots \varepsilon_{i_0-1} \varepsilon_{i_0} \dots \varepsilon_{j_0} \varepsilon_{j_0+1} \dots \varepsilon_n \rangle$ be a DNA expression where for $i = 1, \dots, i_0 - 1, j_0 + 1, \dots, n$, ε_i is either an $\mathcal{N}$-word or a DNA expression, and for $i = i_0, \dots, j_0$, $\varepsilon_i = \alpha_i$ is an $\mathcal{N}$-word. 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 $\alpha_{i_0}, \dots, \alpha_{j_0}$.*

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., 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'.

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 interchange the terms. 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 2.12 are T, $\langle \uparrow \langle \updownarrow \mathrm{C} \rangle \, \mathrm{AT} \langle \downarrow \langle \updownarrow \mathrm{G} \rangle \langle \updownarrow \mathrm{C} \rangle \rangle \rangle$ and $\langle \uparrow \langle \updownarrow \mathrm{A} \rangle \langle \updownarrow \mathrm{T} \rangle \rangle$.

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 an index. 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 2.12, 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 $\langle\updownarrow \text{G}\rangle$ 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 2.12 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 report, 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*.

We say that an $\uparrow$-expression or a $\downarrow$-expression 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. Examples of alternating DNA expressions are

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

Both E_1 and E_2 have exactly 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 2.12 is not alternating, because both its second argument $\langle\uparrow \langle\updownarrow \text{C}\rangle\, \text{AT}\, \langle\downarrow \langle\updownarrow \text{G}\rangle \langle\updownarrow \text{C}\rangle\rangle\rangle$ and its third argument $\langle\uparrow \langle\updownarrow \text{A}\rangle \langle\updownarrow \text{T}\rangle\rangle$ 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 occur 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 $\mathcal{W}_{\mathcal{A}}(\alpha_i)$ For example, if $E = E(\alpha_1, \alpha_2) = \langle \updownarrow \langle \downarrow \alpha_1 \langle \updownarrow \alpha_2 \rangle\rangle\rangle$, then $\mathcal{S}(E) = \binom{c(\alpha_1)\alpha_2}{\alpha_1 c(\alpha_2)}$.

2.5 Nesting level of the brackets

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.

The maximal nesting level of a DNA expression is of particular interest. For example, the maximal nesting level of the DNA expression from Example 2.12 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 2.14 *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.

2.6 The functions L and R for arguments of DNA expressions

An important requirement on the arguments $\varepsilon_1, \ldots, \varepsilon_n$ of an $\uparrow$-expression (or $\downarrow$-expression) is that they must fit together by upper strands (lower strands, respectively). The requirement for $\uparrow$-expressions can be expressed formally in terms of $R(\mathcal{S}^+(\varepsilon_i))$ and $L(\mathcal{S}^+(\varepsilon_{i+1}))$ for $i = 1, \ldots, n-1$. 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. 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 $\downarrow$ 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 $\mathcal{S}^+(\varepsilon_i) = \mathcal{S}^-(\varepsilon_i) = \mathcal{S}(E_i)$. Hence, 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 that $L(\mathcal{S}^+(\varepsilon_i))$, $R(\mathcal{S}^+(\varepsilon_i))$, $L(\mathcal{S}^-(\varepsilon_i))$ and $R(\mathcal{S}^-(\varepsilon_i))$ are an element of:

Lemma 2.15 (See [Van Vliet, 2004, Lemma 2.16]) *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 $\updownarrow$-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 $\uparrow$-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 $\downarrow$-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*

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

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

As we have established in Lemma 2.6, 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 2.16 *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. Using induction, one can easily prove that for arbitrary $l \geq 1$, $E_l = \left(\langle \updownarrow\right)^l \alpha \left(\,\rangle\right)^l$ is a DNA expression, with $\mathcal{S}(E_l) = \binom{\alpha}{c(\alpha)}$. By the pumping lemma for regular languages, 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 $G_1 = (\Sigma_1, \Delta_1, P_1, S_1)$, 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 the same meaning, however, 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 Δ_1 of terminal symbols is equal to $\Sigma_{\mathcal{D}}$: $\Delta_1 = \{\mathrm{A}, \mathrm{C}, \mathrm{G}, \mathrm{T}, \uparrow, \downarrow, \updownarrow, \langle\, , \rangle\}$.

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 2.15(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 $E_{\star,\star}$ of $\updownarrow$ occurring in the latter production represents an arbitrary DNA expression, without restrictions on the strands.

By Lemma 2.15(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 2.17 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 2.18 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

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

The description by nine separate lines, however, makes it easier to refer to a particular production, as we do in the following example.

Example 2.19 The DNA expression from Example 2.12 is the result of many different derivations in G_1. The leftmost derivation is

$$\begin{array}{lcl}
E_{\star,\star} & \overset{1,4}{\Longrightarrow} & \langle\downarrow L_{\star,\star}\rangle \\
& \overset{19,3}{\Longrightarrow} & \langle\downarrow \alpha L_{-,\star}\rangle \\
& \overset{28,4}{\Longrightarrow} & \langle\downarrow \mathrm{T} L_{-,\star}\rangle \\
& \overset{25,4}{\Longrightarrow} & \langle\downarrow \mathrm{T} E_{-,-}L_{-,\star}\rangle \\
& \overset{9,3}{\Longrightarrow} & \langle\downarrow \mathrm{T}\,\langle\uparrow U_{-,-}\rangle\, L_{-,\star}\rangle \\
& \overset{18,2}{\Longrightarrow} & \langle\downarrow \mathrm{T}\,\langle\uparrow E_{-,+}U_{+,-}\rangle\, L_{-,\star}\rangle \\
& \overset{8,1}{\Longrightarrow} & \langle\downarrow \mathrm{T}\,\langle\uparrow \langle\updownarrow \alpha\rangle\, U_{+,-}\rangle\, L_{-,\star}\rangle \\
& \overset{28,2}{\Longrightarrow} & \langle\downarrow \mathrm{T}\,\langle\uparrow \langle\updownarrow \mathrm{C}\rangle\, U_{+,-}\rangle\, L_{-,\star}\rangle \\
& \overset{15,2}{\Longrightarrow} & \langle\downarrow \mathrm{T}\,\langle\uparrow \langle\updownarrow \mathrm{C}\rangle\, \alpha U_{+,-}\rangle\, L_{-,\star}\rangle \\
& \overset{28,5}{\Longrightarrow} & \langle\downarrow \mathrm{T}\,\langle\uparrow \langle\updownarrow \mathrm{C}\rangle\, \mathrm{A}\alpha U_{+,-}\rangle\, L_{-,\star}\rangle
\end{array}$$

$$
\begin{aligned}
&\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\\
&\overset{28,4}{\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 \mathrm{T}\rangle\rangle\rangle\,.
\end{aligned}
$$

Here, numbers i, j above an arrow $\Longrightarrow$ indicate that we have used production (i, j) for the corresponding derivation step. ■

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

Theorem 2.20 $\mathcal{L}(G_1) = \mathcal{L}_{G_1}(E_{\star,\star})$ *is the language* $\mathcal{D}$ *of all DNA expressions.*

and

Corollary 2.21 *The language* $\mathcal{D}$ *of DNA expressions is context-free.*

2.8 The structure tree of a DNA expression

Let E be an arbitrary DNA expression. We define *the structure tree of E* as follows. For each $\mathcal{N}$-word α and 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.

In the structure tree we draw arcs from (nodes labelled by) operators to their arguments. By definition, these arguments are $\mathcal{N}$-words and DNA subexpressions of E. Indeed, for every occurrence of an $\mathcal{N}$-word or a DNA subexpression of E, there is a corresponding node. Hence, the arcs are well defined.

Clearly, the node labelled by an operator is the parent of (the nodes corresponding to) its arguments. These arguments are the children of the operator. If an operator

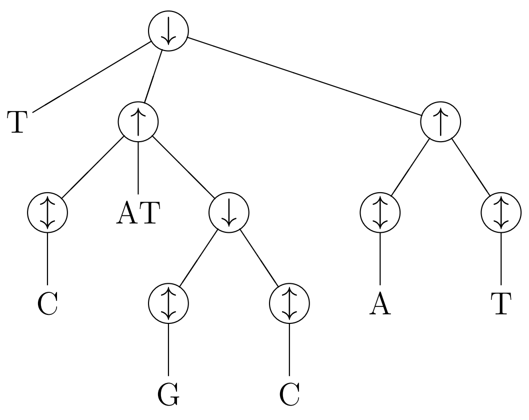


Figure 2.4: The structure tree of the DNA expression from Example 2.12.

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.

Because every $\mathcal{N}$-word and every proper DNA subexpression of E has exactly one parent operator, we indeed obtain a tree. The leaves of the tree are labelled by the $\mathcal{N}$-words α occurring in E, and the internal nodes by the operators. The node labelled by the outermost operator of E is the root of the tree. It corresponds to the entire DNA expression. As an example, in Figure 2.4 we have drawn the structure tree of the DNA expression from Example 2.12.

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

Lemma 2.22 *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 § 2.5, the maximal nesting level of the DNA expression from Example 2.12 is 4. Indeed, the height of the corresponding structure tree in Figure 2.4 is $4+1=5$.

Proof: By induction on the number p of operators occurring in E.

- If $p = 1$, then E is equal to $\langle\uparrow \alpha\rangle$, $\langle\downarrow \alpha\rangle$ or $\langle\updownarrow \alpha\rangle$ for an $\mathcal{N}$-word α. By Lemma 2.14(1), the maximal nesting level of E is $l = 1$. The structure tree t of E consists of a root, labelled by an operator, with one child node, labelled by α. Indeed, the height of t is $2 = l+1$.

- 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.

  Let $E_1, \ldots, E_r$ for some $r \geq 0$ be the expression-arguments of E. Because E contains $p+1 \geq 2$ operators, we must have $r \geq 1$. Each E_j contains at most p operators. For $j = 1, \ldots, r$, let l_j be the maximal nesting level of E_j.

  The structure tree t of E has subtrees corresponding to the arguments of E. A subtree corresponding to an $\mathcal{N}$-word-argument consists of one node labelled by the $\mathcal{N}$-word concerned. Such a subtree has height 1. By the induction hypothesis, the subtree corresponding to an argument E_j has height l_j+1. Hence, by

Lemma 2.1(2), the height of t is

$$\begin{aligned}&\max_{\text{arguments } \varepsilon_i \text{ of } E}(\text{height of the subtree of } t \text{ corresponding to } \varepsilon_i) + 1\\ &= \max\left(\max_{\mathcal{N}\text{-word-arguments of } E} 1,\ \max_{j=1}^{r}(l_j+1)\right) + 1 = \max_{j=1}^{r}(l_j+1)+1.\end{aligned}$$

By Lemma 2.14(2), this equals $l+1$.

□

2.9 Equivalent DNA expressions

Different DNA expressions may correspond to the same DNA molecule. It is, for example, easy to verify that the DNA expressions $\langle\uparrow \alpha\rangle$ and $\langle\uparrow \langle\uparrow \alpha\rangle\rangle$ have the same semantics. 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, we give a number of definitions. Before that, however, we recall some general notions.

A *binary relation* R on a set X is a subset of $X \times X = \{(x, y) \mid x, y \in X\}$. If $(x, y) \in R$, we also write xRy; if $(x, y) \notin R$, we may write $x \not R y$. 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
- *transitive* if for every $x, y, z \in X$, (xRy and yRz) implies xRz

If a relation R is reflexive, symmetric and transitive, R is called an *equivalence relation*;

We return to the world of DNA. We define four binary relations on $\mathcal{D}$.

Definition 2.23 (See [Van Vliet, 2004, Definition 2.17], [Van Vliet et al., 2005, page 380], [Van Vliet et al., 2006, page 134]) *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 2.24 (See [Van Vliet, 2004, Definition 2.18]) *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 $E_1 \overline{\overline{\triangledown}} E_2$ 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.

We further define a variant of this last relation.

Definition 2.25 (See [Van Vliet, 2004, Definition 2.19]) *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 $\mathcal{S}(E_1) = X_1c_1 \ldots c_{r-1}X_r$ and $\mathcal{S}(E_2) = X_1 \ldots X_r$. We write $E_1 \ {}_{\triangledown}{\equiv}\ E_2$ then.*

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.

Chapter 3

Basic Results on DNA Expressions

3.1 Expressible formal DNA molecules

Many formal DNA molecules can be denoted by DNA expressions. We call such formal DNA molecules *expressible*.

Lemma 3.1 (See [Van Vliet, 2004, Lemma 3.1]) *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 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 $i = 1, \ldots, n$, the lower 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).*

In an analogous way we prove

Lemma 3.2 (See [Van Vliet, 2004, Lemma 3.2]) *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).*

Theorem 3.3 (See [Van Vliet, 2004, Theorem 3.5], [Van Vliet et al., 2005, Theorem 4], [Van Vliet et al., 2006, Theorem 2]) *A formal DNA molecule X is expressible, if and only if X does not contain both upper nick letters and lower nick letters.*

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

Corollary 3.4 (Cf. [Van Vliet, 2004, Corollary 2.6]) *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 $y \in \{^\triangledown, {}_\vartriangle\}$, such that*

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

3.2 Nick free DNA expressions

Lemma 3.5 *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} (or both) 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.

3.3 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,

$$\langle \updownarrow \alpha \rangle \equiv \langle \updownarrow \langle \uparrow \alpha \rangle \rangle \equiv \langle \updownarrow \langle \downarrow c(\alpha) \rangle \rangle . \tag{3.1}$$

Lemma 3.6 (See [Van Vliet, 2004, Lemma 3.6]) *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*

$$\langle \uparrow \varepsilon_1 \ldots \varepsilon_{i_0-1} \langle \uparrow \varepsilon_{i_0} \ldots \varepsilon_{j_0} \rangle \varepsilon_{j_0+1} \ldots \varepsilon_n \rangle \equiv \langle \uparrow \varepsilon_1 \ldots \varepsilon_n \rangle \tag{3.2}$$

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

The following equivalence is clear from the definition of the operator $\updownarrow$ (see Definition 2.11) and from property (2.7):

$$\langle \updownarrow \langle \updownarrow \varepsilon \rangle \rangle \equiv \langle \updownarrow \varepsilon \rangle \tag{3.3}$$

for every $\mathcal{N}$-word or DNA expression ε.

Lemma 3.7 (See [Van Vliet, 2004, Lemma 3.7]) *Let E be a DNA expression and let E^s be (an occurrence of) a DNA subexpression in E. Let $E^{s\prime}$ be a DNA expression such that $E^s \underset{\triangledown}{\equiv} E^{s\prime}$.*

When we substitute (the occurrence of) E^s in E by $E^{s\prime}$, the resulting string E' is again a DNA expression, and $E \underset{\triangledown}{\equiv} E'$.

Lemma 3.8 (See [Van Vliet, 2004, Lemma 3.10]) *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 $E \equiv_\triangledown \langle\uparrow \langle\updownarrow \varepsilon_1\rangle \dots \langle\updownarrow \varepsilon_n\rangle\rangle$.*

Corollary 3.9 (See [Van Vliet, 2004, Corollary 3.11]) *For all $\mathcal{N}$-words $\alpha_1, \dots, \alpha_n$ with $n \geq 1$, we have*

$$\langle\uparrow \langle\updownarrow \alpha_1\rangle \dots \langle\updownarrow \alpha_n\rangle\rangle \;{}_\triangledown\!\equiv \langle\updownarrow \alpha_1 \dots \alpha_n\rangle\,.$$

Theorem 3.10 (See [Van Vliet, 2004, Theorem 3.12]) *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* $i = 1, \dots, n-2$,
- $\mathcal{S}^+(\varepsilon_{n-1}) \overline{\sqsubset} \mathcal{S}(E_{n,1})$,
- $\mathcal{S}(E_{n,1}) \underline{\sqsubset} \mathcal{S}^-(\varepsilon_{n,2})$ *and*
- $\mathcal{S}^-(\varepsilon_{n,i}) \underline{\sqsubset} \mathcal{S}^-(\varepsilon_{n,i+1})$ *for* $i = 2, \dots, m-1$.

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 $E \underset{\triangledown}{\equiv} E'$.*
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 $E \equiv E'$.*

What we actually do in Theorem 3.10, 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. 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 3.1.

Proof:

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 $\varepsilon_1, \dots, \varepsilon_{n-1}$ 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 $\varepsilon_{n,2}, \dots \varepsilon_{n,m}$ are $\mathcal{N}$-words and DNA expressions, alternately.

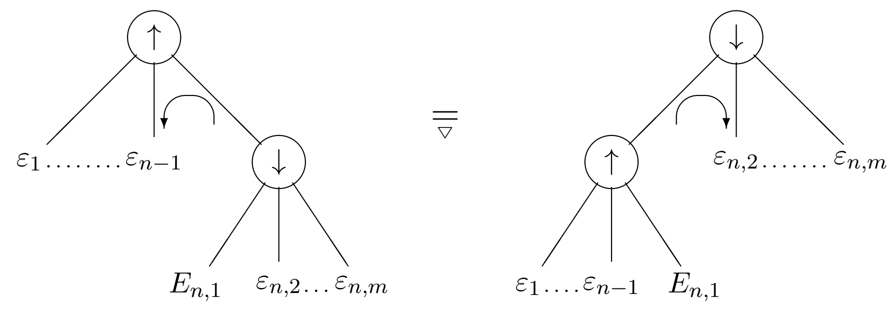

$\uparrow$
$\varepsilon_1 \ldots\ldots \varepsilon_{n-1}$
$\downarrow$
$E_{n,1}$
$\varepsilon_{n,2} \ldots \varepsilon_{n,m}$
$\equiv_{\triangledown}$
$\downarrow$
$\uparrow$
$\varepsilon_{n,2} \ldots\ldots \varepsilon_{n,m}$
$\varepsilon_1 \ldots \varepsilon_{n-1}$
$E_{n,1}$

Figure 3.1: (See [Van Vliet, 2004, Figure 3.2]) Analogue of Theorem 3.10(1) for structure trees of DNA expressions.

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, $E \equiv_{\triangledown} E'$. By Lemma 3.5, 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 $E \equiv E'$.

□

Theorem 3.11 *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 $j = 1, \ldots, r$, $\varepsilon_{i_j} = \left\langle \downarrow \varepsilon_{i_j,1} \ldots \varepsilon_{i_j,m_j} \right\rangle$ 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 $E \equiv_{\triangledown} E'$.

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 $\langle\uparrow \varepsilon_1 \ldots \varepsilon_{i_1-1}\varepsilon_{i_1,1}\rangle$, $\big\langle\uparrow \varepsilon_{i_j,m_j}\varepsilon_{i_j+1} \ldots \varepsilon_{i_{j+1}-1}\varepsilon_{i_{j+1},1}\big\rangle$ 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.

Proof: 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 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 3.6, Theorem 3.10(1) (together with Lemma 3.7) and once more Theorem 3.10(1):

$$\begin{array}{rcl} E & = & \langle\uparrow \varepsilon_1 \ldots \varepsilon_{i_1-1}\,\langle\downarrow \varepsilon_{i_1,1}\varepsilon_{i_1,2} \ldots \varepsilon_{i_1,m_1-1}\varepsilon_{i_1,m_1}\rangle\,\varepsilon_{i_1+1} \ldots \varepsilon_n\rangle \\ & \equiv & \langle\uparrow \varepsilon_1 \ldots \varepsilon_{i_1-1}\,\langle\uparrow\,\langle\downarrow \varepsilon_{i_1,1}\varepsilon_{i_1,2} \ldots \varepsilon_{i_1,m_1-1}\varepsilon_{i_1,m_1}\rangle\,\varepsilon_{i_1+1} \ldots \varepsilon_n\rangle\,\rangle \\ & \underset{\triangledown}{=} & \langle\uparrow \varepsilon_1 \ldots \varepsilon_{i_1-1}\,\langle\downarrow \varepsilon_{i_1,1}\varepsilon_{i_1,2} \ldots \varepsilon_{i_1,m_1-1}\,\langle\uparrow \varepsilon_{i_1,m_1}\varepsilon_{i_1+1} \ldots \varepsilon_n\rangle\rangle\,\rangle \\ & \underset{\triangledown}{=} & \langle\downarrow\,\langle\uparrow \varepsilon_1 \ldots \varepsilon_{i_1-1}\varepsilon_{i_1,1}\rangle\,\varepsilon_{i_1,2} \ldots \varepsilon_{i_1,m_1-1}\,\langle\uparrow \varepsilon_{i_1,m_1}\varepsilon_{i_1+1} \ldots \varepsilon_n\rangle\,\rangle \\ & = & E'. \end{array}$$

     Indeed, E' is a DNA expression satisfying $E \underset{\triangledown}{=} E'$.

   - 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,

$$\begin{array}{rcl} E & = & \big\langle\uparrow \varepsilon_1 \ldots \varepsilon_{i_1-1}\,\langle\downarrow \varepsilon_{i_1,1}\varepsilon_{i_1,2} \ldots \varepsilon_{i_1,m_1-1}\varepsilon_{i_1,m_1}\rangle\,\varepsilon_{i_1+1} \ldots \varepsilon_{i_\rho-1} \\ & & \quad \big\langle\downarrow \varepsilon_{i_\rho,1}\varepsilon_{i_\rho,2} \ldots \varepsilon_{i_\rho,m_\rho-1}\varepsilon_{i_\rho,m_\rho}\big\rangle\,\varepsilon_{i_\rho+1} \ldots \varepsilon_{i_{\rho+1}-1} \\ & & \quad \big\langle\downarrow \varepsilon_{i_{\rho+1},1}\varepsilon_{i_{\rho+1},2} \ldots \varepsilon_{i_{\rho+1},m_{\rho+1}-1}\varepsilon_{i_{\rho+1},m_{\rho+1}}\big\rangle\,\varepsilon_{i_{\rho+1}+1} \ldots \varepsilon_n\,\big\rangle\,. \end{array}$$

     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

$$\begin{array}{rcl} E & = & \big\langle\uparrow \varepsilon_1 \ldots \varepsilon_{i_1-1}\varepsilon_{i_1}\varepsilon_{i_1+1} \ldots \varepsilon_{i_\rho-1}\varepsilon_{i_\rho}\varepsilon_{i_\rho+1} \ldots \varepsilon_{i_{\rho+1}-1} \\ & & \quad \big\langle\downarrow \varepsilon_{i_{\rho+1},1}\varepsilon_{i_{\rho+1},2} \ldots \varepsilon_{i_{\rho+1},m_{\rho+1}-1}\varepsilon_{i_{\rho+1},m_{\rho+1}}\big\rangle\,\varepsilon_{i_{\rho+1}+1} \ldots \varepsilon_n\,\big\rangle\,. \end{array}$$

     We apply the induction hypothesis to E and the $\downarrow$-argument $\varepsilon_{i_{\rho+1}}$:

$$\begin{array}{rcl} E & \underset{\triangledown}{=} & \big\langle\downarrow\,\big\langle\uparrow \varepsilon_1 \ldots \varepsilon_{i_1-1}\varepsilon_{i_1}\varepsilon_{i_1+1} \ldots \varepsilon_{i_\rho-1}\varepsilon_{i_\rho}\varepsilon_{i_\rho+1} \ldots \varepsilon_{i_{\rho+1}-1}\varepsilon_{i_{\rho+1},1}\big\rangle \\ & & \quad \varepsilon_{i_{\rho+1},2} \ldots \varepsilon_{i_{\rho+1},m_{\rho+1}-1}\,\big\langle\uparrow \varepsilon_{i_{\rho+1},m_{\rho+1}}\varepsilon_{i_{\rho+1}+1} \ldots \varepsilon_n\big\rangle\,\big\rangle \\ & = & \big\langle\downarrow\,\big\langle\uparrow \varepsilon_1 \ldots \varepsilon_{i_1-1}\,\langle\downarrow \varepsilon_{i_1,1}\varepsilon_{i_1,2} \ldots \varepsilon_{i_1,m_1-1}\varepsilon_{i_1,m_1}\rangle\,\varepsilon_{i_1+1} \ldots \varepsilon_{i_\rho-1} \\ & & \qquad \big\langle\downarrow \varepsilon_{i_\rho,1}\varepsilon_{i_\rho,2} \ldots \varepsilon_{i_\rho,m_\rho-1}\varepsilon_{i_\rho,m_\rho}\big\rangle\,\varepsilon_{i_\rho+1} \ldots \varepsilon_{i_{\rho+1}-1}\varepsilon_{i_{\rho+1},1}\,\big\rangle \\ & & \quad \varepsilon_{i_{\rho+1},2} \ldots \varepsilon_{i_{\rho+1},m_{\rho+1}-1}\,\big\langle\uparrow \varepsilon_{i_{\rho+1},m_{\rho+1}}\varepsilon_{i_{\rho+1}+1} \ldots \varepsilon_n\big\rangle\,\big\rangle\,. \end{array}$$

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 3.7 and subsequently use Lemma 3.6, we find

$$\begin{aligned} E \;&\underset{\triangledown}{=}\; \langle\downarrow \langle\downarrow \langle\uparrow \varepsilon_1 \ldots \varepsilon_{i_1-1}\varepsilon_{i_1,1}\rangle\, \varepsilon_{i_1,2} \ldots \varepsilon_{i_1,m_1-1} \\ &\qquad\quad \langle\uparrow \varepsilon_{i_1,m_1}\varepsilon_{i_1+1} \ldots\rangle \ldots \langle\uparrow \ldots \varepsilon_{i_\rho-1}\varepsilon_{i_\rho,1}\rangle \\ &\qquad\quad \varepsilon_{i_\rho,2} \ldots \varepsilon_{i_\rho,m_\rho-1} \langle\uparrow \varepsilon_{i_\rho,m_\rho}\varepsilon_{i_\rho+1} \ldots \varepsilon_{i_{\rho+1}-1}\varepsilon_{i_{\rho+1},1}\rangle\ \rangle \\ &\qquad \varepsilon_{i_{\rho+1},2} \ldots \varepsilon_{i_{\rho+1},m_{\rho+1}-1} \langle\uparrow \varepsilon_{i_{\rho+1},m_{\rho+1}}\varepsilon_{i_{\rho+1}+1} \ldots \varepsilon_n\rangle\ \rangle \\ &\equiv\; \langle\downarrow \langle\uparrow \varepsilon_1 \ldots \varepsilon_{i_1-1}\varepsilon_{i_1,1}\rangle\, \varepsilon_{i_1,2} \ldots \varepsilon_{i_1,m_1-1} \\ &\qquad \langle\uparrow \varepsilon_{i_1,m_1}\varepsilon_{i_1+1} \ldots\rangle \ldots \langle\uparrow \ldots \varepsilon_{i_\rho-1}\varepsilon_{i_\rho,1}\rangle \\ &\qquad \varepsilon_{i_\rho,2} \ldots \varepsilon_{i_\rho,m_\rho-1} \langle\uparrow \varepsilon_{i_\rho,m_\rho}\varepsilon_{i_\rho+1} \ldots \varepsilon_{i_{\rho+1}-1}\varepsilon_{i_{\rho+1},1}\rangle \\ &\qquad \varepsilon_{i_{\rho+1},2} \ldots \varepsilon_{i_{\rho+1},m_{\rho+1}-1} \langle\uparrow \varepsilon_{i_{\rho+1},m_{\rho+1}}\varepsilon_{i_{\rho+1}+1} \ldots \varepsilon_n\rangle\ \rangle \\ &=\; E'. \end{aligned}$$

We conclude again that E' is a DNA expression satisfying $E \underset{\triangledown}{=} E'$.

2. In the inductive proof of the previous claim, we did not only use Theorem 3.10(1), but also Lemma 3.6 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 3.10(2). We would also need to consider the effects of Lemma 3.6. Instead of doing that, we give a direct proof, which resembles the proof of Theorem 3.10(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 $\langle\uparrow \varepsilon_1 \dots \varepsilon_{i_1-1}\varepsilon_{i_1,1}\rangle$ of E'. The first i_1-1 arguments $\varepsilon_1, \dots, \varepsilon_{i_1-1}$ 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 $\langle\uparrow \varepsilon_{i_r,m_r}\varepsilon_{i_r+1}\dots\varepsilon_n\rangle$ of E' are $\mathcal{N}$-words and DNA expressions, alternately. Finally, for $j = 1, \dots, r-1$, the arguments of the $\uparrow$-argument $\left\langle\uparrow \varepsilon_{i_j,m_j}\varepsilon_{i_j+1}\dots\varepsilon_{i_{j+1}-1}\varepsilon_{i_{j+1},1}\right\rangle$ of E' are the DNA expression ε_{i_j,m_j}, an alternating sequence of $\mathcal{N}$-words and DNA expressions $\varepsilon_{i_j+1}, \dots, \varepsilon_{i_{j+1}-1}$ (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, $E \overset{=}{\triangledown} E'$. By Lemma 3.5, however, both E and E' are nick free. This implies that E and E' are (strictly) equivalent: $E \equiv E'$.

□

Theorem 3.11 can be reversed. That is, we can also start from E' and conclude that E is a DNA expression satisfying $E \overset{=}{\triangledown} E'$ (or even $E \equiv E'$):

Theorem 3.12 *Let $E' = \langle\downarrow \varepsilon_1 \dots \varepsilon_n\rangle$ for some $n \geq 1$ and $\mathcal{N}$-words and DNA expressions $\varepsilon_1, \dots, \varepsilon_n$ be a DNA expression. Let $\varepsilon_{i_1}, \dots, \varepsilon_{i_r}, \varepsilon_{i_{r+1}}$ for some $r \geq 1$ and $1 = i_1 < \dots < i_r < i_{r+1} = n$ be $\uparrow$-arguments of E' that have at least two arguments themselves. Hence, for $j = 1, \dots, r, r+1$, $\varepsilon_{i_j} = \left\langle\uparrow \varepsilon_{i_j,1}\dots\varepsilon_{i_j,m_j}\right\rangle$ for some $m_j \geq 2$ and $\mathcal{N}$-words and DNA expressions $\varepsilon_{i_j,1}, \dots, \varepsilon_{i_j,m_j}$, and*

$$\begin{aligned} E' = \Big\langle\downarrow\ & \langle\uparrow \varepsilon_{1,1}\dots\varepsilon_{1,m_1-1}\varepsilon_{1,m_1}\rangle\, \varepsilon_2\dots\varepsilon_{i_2-1}\, \langle\uparrow \varepsilon_{i_2,1}\varepsilon_{i_2,2}\dots\varepsilon_{i_2,m_2-1}\varepsilon_{i_2,m_2}\rangle \\ & \varepsilon_{i_2+1}\dots\varepsilon_{i_r-1}\, \langle\uparrow \varepsilon_{i_r,1}\varepsilon_{i_r,2}\dots\varepsilon_{i_r,m_r-1}\varepsilon_{i_r,m_r}\rangle \\ & \varepsilon_{i_r+1}\dots\varepsilon_{n-1}\, \left\langle\uparrow \varepsilon_{n,1}\varepsilon_{n,2}\dots\varepsilon_{n,m_{r+1}}\right\rangle\ \Big\rangle\,. \end{aligned}$$

1. *The string*

$$\begin{aligned} E = \Big\langle\uparrow\ & \varepsilon_{1,1}\dots\varepsilon_{1,m_1-1}\, \langle\downarrow \varepsilon_{1,m_1}\varepsilon_2\dots\varepsilon_{i_2-1}\varepsilon_{i_2,1}\rangle\, \varepsilon_{i_2,2}\dots\varepsilon_{i_2,m_2-1} \\ & \langle\downarrow \varepsilon_{i_2,m_2}\varepsilon_{i_2+1}\dots\rangle \dots \langle\downarrow \dots\varepsilon_{i_r-1}\varepsilon_{i_r,1}\rangle\, \varepsilon_{i_r,2}\dots\varepsilon_{i_r,m_r-1} \\ & \langle\downarrow \varepsilon_{i_r,m_r}\varepsilon_{i_r+1}\dots\varepsilon_{n-1}\varepsilon_{n,1}\rangle\, \varepsilon_{n,2}\dots\varepsilon_{n,m_{r+1}}\ \Big\rangle \end{aligned}$$

   is a DNA expression satisfying $E \overset{=}{\triangledown} E'$.

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 $n \geq 2$, 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 3.11(1). Instead, we give a proof that makes use of Theorem 3.11(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 3.10(1) two times (the second time in combination with Lemma 3.7) and subsequently apply Lemma 3.6, we find

$$\begin{aligned}
E' \underset{\triangledown}{=} & \left\langle\uparrow \varepsilon_{1,1} \ldots \varepsilon_{1,m_1-1} \left\langle\downarrow \varepsilon_{1,m_1}\varepsilon_2 \ldots \varepsilon_{i_2-1} \left\langle\uparrow \varepsilon_{i_2,1}\varepsilon_{i_2,2} \ldots \varepsilon_{i_2,m_2-1}\varepsilon_{i_2,m_2}\right\rangle \right.\right. \\
& \qquad \varepsilon_{i_2+1} \ldots \varepsilon_{i_r-1} \left\langle\uparrow \varepsilon_{i_r,1}\varepsilon_{i_r,2} \ldots \varepsilon_{i_r,m_r-1}\varepsilon_{i_r,m_r}\right\rangle \\
& \qquad \left.\left. \varepsilon_{i_r+1} \ldots \varepsilon_{n-1} \left\langle\uparrow \varepsilon_{n,1}\varepsilon_{n,2} \ldots \varepsilon_{n,m_{r+1}}\right\rangle \right\rangle \right\rangle \\
\underset{\triangledown}{=} & \left\langle\uparrow \varepsilon_{1,1} \ldots \varepsilon_{1,m_1-1} \left\langle\uparrow \left\langle\downarrow \varepsilon_{1,m_1}\varepsilon_2 \ldots \varepsilon_{i_2-1} \left\langle\uparrow \varepsilon_{i_2,1}\varepsilon_{i_2,2} \ldots \varepsilon_{i_2,m_2-1}\varepsilon_{i_2,m_2}\right\rangle \right.\right.\right. \\
& \qquad \varepsilon_{i_2+1} \ldots \varepsilon_{i_r-1} \left\langle\uparrow \varepsilon_{i_r,1}\varepsilon_{i_r,2} \ldots \varepsilon_{i_r,m_r-1}\varepsilon_{i_r,m_r}\right\rangle \\
& \qquad \left.\left.\left. \varepsilon_{i_r+1} \ldots \varepsilon_{n-1}\varepsilon_{n,1} \right\rangle \varepsilon_{n,2} \ldots \varepsilon_{n,m_{r+1}} \right\rangle \right\rangle \\
\equiv & \left\langle\uparrow \varepsilon_{1,1} \ldots \varepsilon_{1,m_1-1} \left\langle\downarrow \varepsilon_{1,m_1}\varepsilon_2 \ldots \varepsilon_{i_2-1} \left\langle\uparrow \varepsilon_{i_2,1}\varepsilon_{i_2,2} \ldots \varepsilon_{i_2,m_2-1}\varepsilon_{i_2,m_2}\right\rangle \right.\right. \\
& \qquad \varepsilon_{i_2+1} \ldots \varepsilon_{i_r-1} \left\langle\uparrow \varepsilon_{i_r,1}\varepsilon_{i_r,2} \ldots \varepsilon_{i_r,m_r-1}\varepsilon_{i_r,m_r}\right\rangle \\
& \qquad \left.\left. \varepsilon_{i_r+1} \ldots \varepsilon_{n-1}\varepsilon_{n,1} \right\rangle \varepsilon_{n,2} \ldots \varepsilon_{n,m_{r+1}} \right\rangle .
\end{aligned}$$

   Let us use E'' to denote the resulting DNA expression, and let us use E_1 to denote the $\downarrow$-argument

$$\begin{aligned}
& \left\langle\downarrow \varepsilon_{1,m_1}\varepsilon_2 \ldots \varepsilon_{i_2-1} \left\langle\uparrow \varepsilon_{i_2,1}\varepsilon_{i_2,2} \ldots \varepsilon_{i_2,m_2-1}\varepsilon_{i_2,m_2}\right\rangle \right. \\
& \quad \varepsilon_{i_2+1} \ldots \varepsilon_{i_r-1} \left\langle\uparrow \varepsilon_{i_r,1}\varepsilon_{i_r,2} \ldots \varepsilon_{i_r,m_r-1}\varepsilon_{i_r,m_r}\right\rangle \\
& \quad \left. \varepsilon_{i_r+1} \ldots \varepsilon_{n-1}\varepsilon_{n,1} \right\rangle
\end{aligned}$$

   of E''.

   If $r = 1$, then $i_2 = i_{r+1} = n$, E_1 reduces to $\langle\downarrow \varepsilon_{1,m_1}\varepsilon_2 \ldots \varepsilon_{i_2-1}\varepsilon_{n,1}\rangle$, and

$$E'' = \left\langle\uparrow \varepsilon_{1,1} \ldots \varepsilon_{1,m_1-1} \left\langle\downarrow \varepsilon_{1,m_1}\varepsilon_2 \ldots \varepsilon_{i_2-1}\varepsilon_{n,1}\right\rangle \varepsilon_{n,2} \ldots \varepsilon_{n,m_{r+1}}\right\rangle ,$$

   which equals the string E from the claim. In this case, indeed, E is an $\uparrow$-expression satisfying $E \underset{\triangledown}{=} E'$.

   If, on the other hand, $r \geq 2$, then E_1 has at least one $\uparrow$-argument $\langle\uparrow \varepsilon_{i_2,1}\varepsilon_{i_2,2} \ldots \varepsilon_{i_2,m_2-1}\varepsilon_{i_2,m_2}\rangle$, which is neither the first argument, nor the last argument of E_1 and which has at least two arguments itself. Hence, we can apply Theorem 3.11(1) (in combination with Lemma 3.7) to E_1 and subsequently apply Lemma 3.6:

$$\begin{aligned}
E' \underset{\triangledown}{=} & \left\langle\uparrow \varepsilon_{1,1} \ldots \varepsilon_{1,m_1-1} \left\langle\uparrow \left\langle\downarrow \varepsilon_{1,m_1}\varepsilon_2 \ldots \varepsilon_{i_2-1}\varepsilon_{i_2,1}\right\rangle \varepsilon_{i_2,2} \ldots \varepsilon_{i_2,m_2-1} \right.\right. \\
& \qquad \left\langle\downarrow \varepsilon_{i_2,m_2}\varepsilon_{i_2+1} \ldots\right\rangle \ldots \left\langle\downarrow \ldots \varepsilon_{i_r-1}\varepsilon_{i_r,1}\right\rangle \varepsilon_{i_r,2} \ldots \varepsilon_{i_r,m_r-1} \\
& \qquad \left.\left. \left\langle\downarrow \varepsilon_{i_r,m_r}\varepsilon_{i_r+1} \ldots \varepsilon_{n-1}\varepsilon_{n,1}\right\rangle \right\rangle \varepsilon_{n,2} \ldots \varepsilon_{n,m_{r+1}} \right\rangle \\
\equiv & \left\langle\uparrow \varepsilon_{1,1} \ldots \varepsilon_{1,m_1-1} \left\langle\downarrow \varepsilon_{1,m_1}\varepsilon_2 \ldots \varepsilon_{i_2-1}\varepsilon_{i_2,1}\right\rangle \varepsilon_{i_2,2} \ldots \varepsilon_{i_2,m_2-1} \right. \\
& \quad \left\langle\downarrow \varepsilon_{i_2,m_2}\varepsilon_{i_2+1} \ldots\right\rangle \ldots \left\langle\downarrow \ldots \varepsilon_{i_r-1}\varepsilon_{i_r,1}\right\rangle \varepsilon_{i_r,2} \ldots \varepsilon_{i_r,m_r-1} \\
& \quad \left. \left\langle\downarrow \varepsilon_{i_r,m_r}\varepsilon_{i_r+1} \ldots \varepsilon_{n-1}\varepsilon_{n,1}\right\rangle \varepsilon_{n,2} \ldots \varepsilon_{n,m_{r+1}} \right\rangle \\
= & \ E.
\end{aligned}$$

   We conclude that also in this case, E is an $\uparrow$-expression satisfying $E \underset{\triangledown}{=} E'$.

2. The proof of this claim is similar to that of Theorem 3.11(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 $E = E(\alpha_1, \dots, \alpha_k)$ for some $k \geq 1$ be an arbitrary DNA expression. We define the $\mathcal{N}$-word α_E as the concatenation of the $\mathcal{N}$-words $\alpha'_1, \dots, \alpha'_k$, where

$$\alpha'_i = \begin{cases} \alpha_i & \text{if the parent operator of } \alpha_i \text{ in } E \text{ is } \updownarrow \text{ or } \uparrow \\ c(\alpha_i) & \text{if the parent operator of } \alpha_i \text{ in } E \text{ is } \downarrow \end{cases} \qquad (i = 1, \dots, k).$$

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 3.13 *Let $E = E(\alpha_1, \dots, \alpha_k)$ for some $k \geq 1$ be an $\updownarrow$-expression. Then $E \mathrel{_\triangledown\!\equiv} \langle \updownarrow \alpha_E \rangle$,*

Proof: 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 $E \mathrel{_\triangledown\!\equiv} \langle \updownarrow \alpha_E \rangle$.

- If $p = 2$, then the argument of the (outermost) operator $\updownarrow$ in 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 (3.3), $E = \langle \updownarrow \langle \updownarrow \alpha_1 \rangle \rangle \equiv \langle \updownarrow \alpha_1 \rangle = \langle \updownarrow \alpha_E \rangle$ with $\alpha_E = \alpha_1$;
  - $E_1 = \langle \uparrow \alpha_1 \rangle$, but then, by (3.1), $E = \langle \updownarrow \langle \uparrow \alpha_1 \rangle \rangle \equiv \langle \updownarrow \alpha_1 \rangle = \langle \updownarrow \alpha_E \rangle$ with $\alpha_E = \alpha_1$;
  - $E_1 = \langle \downarrow \alpha_1 \rangle$, but then, by (3.1), $E = \langle \updownarrow \langle \downarrow \alpha_1 \rangle \rangle \equiv \langle \updownarrow c(\alpha_1) \rangle = \langle \updownarrow \alpha_E \rangle$ 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 (3.3). 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 $E = \langle \updownarrow \langle \uparrow \varepsilon_1 \dots \varepsilon_n \rangle \rangle$ for some $n \geq 1$ and $\mathcal{N}$-words and DNA expressions $\varepsilon_1, \dots, \varepsilon_n$. For $i = 1, \dots, n$, let

$$\varepsilon_i' = \begin{cases} \langle \uparrow \alpha \rangle & \text{if } \varepsilon_i \text{ is an } \mathcal{N}\text{-word } \alpha \\ \varepsilon_i & \text{if } \varepsilon_i \text{ is a DNA expression} \end{cases} .$$

Then by Lemma 3.6 and Lemma 3.7,

$$E = \langle \updownarrow \langle \uparrow \varepsilon_1 \dots \varepsilon_n \rangle \rangle \equiv \langle \updownarrow \langle \uparrow \varepsilon_1' \dots \varepsilon_n' \rangle \rangle .$$

Because every ε_i' is a DNA expression, we can apply Lemma 3.8:

$$\langle \updownarrow \langle \uparrow \varepsilon_1' \dots \varepsilon_n' \rangle \rangle \equiv_{\triangledown} \langle \uparrow \langle \updownarrow \varepsilon_1' \rangle \dots \langle \updownarrow \varepsilon_n' \rangle \rangle .$$

Now consider an argument $\langle \updownarrow \varepsilon_i' \rangle$ with $1 \leq i \leq n$. If ε_i is an $\mathcal{N}$-word α, then $\varepsilon_i' = \langle \uparrow \alpha \rangle$ and $\langle \updownarrow \varepsilon_i' \rangle = \langle \updownarrow \langle \uparrow \alpha \rangle \rangle$, which contains $2 \leq p$ operators. If, on the other hand, ε_i is a DNA expression, then $\varepsilon_i' = \varepsilon_i$ and $\langle \updownarrow \varepsilon_i' \rangle = \langle \updownarrow \varepsilon_i \rangle$. This $\updownarrow$-expression contains at most p operators.

In both cases, by the induction hypothesis, $\langle \updownarrow \varepsilon_i' \rangle \,{}_{\triangledown}\!\equiv \left\langle \updownarrow \alpha_{\varepsilon_i'} \right\rangle$. Now, by Lemma 3.7 and Corollary 3.9,

$$\langle \uparrow \langle \updownarrow \varepsilon_1' \rangle \dots \langle \updownarrow \varepsilon_n' \rangle \rangle \,{}_{\triangledown}\!\equiv \left\langle \uparrow \left\langle \updownarrow \alpha_{\varepsilon_1'} \right\rangle \dots \left\langle \updownarrow \alpha_{\varepsilon_n'} \right\rangle \right\rangle \,{}_{\triangledown}\!\equiv \left\langle \updownarrow \alpha_{\varepsilon_1'} \dots \alpha_{\varepsilon_n'} \right\rangle .$$

Indeed, $\alpha_{\varepsilon_1'} \dots \alpha_{\varepsilon_n'}$ 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: $\alpha_{\varepsilon_1'} \dots \alpha_{\varepsilon_n'} = \alpha_E$.

When we combine all equivalences (pre-/post-modulo nicks), we conclude that $E \mathrel{\overset{=}{\triangledown}} \langle \updownarrow \alpha_E \rangle$. 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. This case can be dealt with completely analogously to the previous case, using the '$\downarrow$-versions' of Lemma 3.6, Lemma 3.8 and Corollary 3.9.

□

Chapter 4

The Length of 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.

There is a simple relation between the length of a DNA expression E denoting a formal DNA molecule X and $|X|_{\mathcal{A}}$.

Lemma 4.1 (See [Van Vliet, 2004, Lemma 4.1], [Van Vliet et al., 2005, Lemma 5], [Van Vliet et al., 2006, Lemma 3]) *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 4.1 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.

4.1 (Blocks of) components of a formal DNA molecule

Definition 4.2 (See [Van Vliet, 2004, Definition 4.2], [Van Vliet et al., 2005, page 381], [Van Vliet et al., 2006, page 134]) *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.*

Definition 4.3 (Cf. [Van Vliet, 2004, Definition 4.3 and Definition 4.33], [Van Vliet et al., 2005, page 381]) (See [Van Vliet et al., 2006, Definition 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 $\uparrow$-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

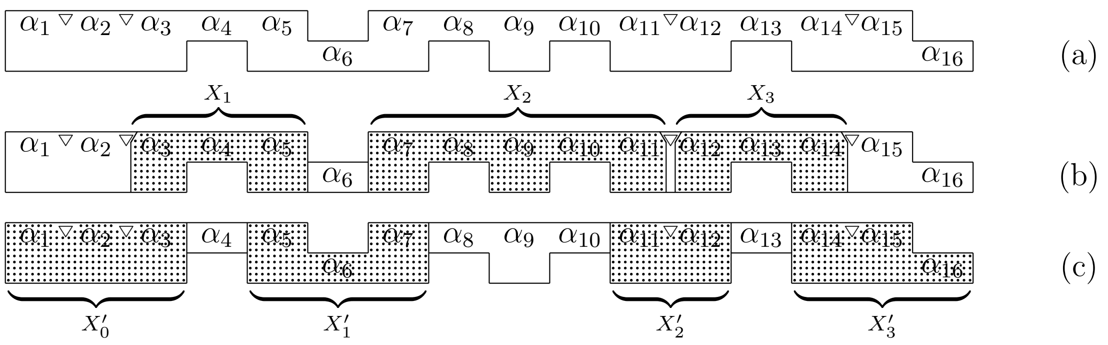

$\alpha_1 \triangledown \alpha_2 \triangledown \alpha_3$ α_4 α_5 α_6 α_7 α_8 α_9 α_{10} $\alpha_{11} \triangledown \alpha_{12}$ α_{13} $\alpha_{14} \triangledown \alpha_{15}$ α_{16}
(a)
X_1
X_2
X_3
(b)
(c)
X'_0
X'_1
X'_2
X'_3

Figure 4.1: Primitive $\uparrow$-blocks and primitive $\downarrow$-blocks. (a) An example formal DNA molecule X that contains (upper) nick letters. (b) The primitive $\uparrow$-blocks of X. Note that the upper nick letters are not part of these blocks. (c) The primitive $\downarrow$-blocks of X.

- *X_1 contains at least one non-double component,*
- *each non-double component of X_1 is an $\uparrow$-component,*
-
  - *either $a_0 = 1$ (hence Y_1 is empty),*
  - *or $a_0 \geq 2$ and x'_{a_0-1} is a $\downarrow$-component,*

  and
-
  - *either $a_1 = k$ (hence Y_2 is empty),*
  - *or $a_1 \leq k-1$ and x'_{a_1+1} is a $\downarrow$-component.*

A primitive $\uparrow$-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 $\uparrow$-block by the substring X_1 itself.

The definition of a primitive $\downarrow$-block is completely analogous to that of a primitive $\uparrow$-block. We may use the term *primitive block* to refer to either a primitive $\uparrow$-block, or a primitive $\downarrow$-block.

In Figure 4.1, we have indicated the primitive $\uparrow$-blocks and the primitive $\downarrow$-blocks of a certain formal DNA molecule containing upper nick letters.

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.

Lemma 4.4 (Cf. [Van Vliet, 2004, Lemma 4.7]) *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.*

It is easily verified that all claims are valid for the formal DNA molecule depicted in Figure 4.1. For this molecule, the alternating sequence is $X'_0, X_1, X'_1, X_2, X'_2, X_3, X'_3$.

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 4.5 (Cf. [Van Vliet, 2004, Definition 4.8], [Van Vliet et al., 2005, Definition 6]) (See [Van Vliet et al., 2006, Definition 5]) *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.*

Lemma 4.6 (See [Van Vliet, 2004, Lemma 4.10, Lemma 4.25 and Lemma 4.12]) *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 $B_{\uparrow}(X) = B_{\downarrow}(X) - 1$.*

4.2 Lower bounds for the length of a DNA expression

Theorem 4.7 (See [Van Vliet, 2004, Corollary 4.19], [Van Vliet et al., 2005, Theorem 8], [Van Vliet et al., 2006, Theorem 8]) *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*

$$\begin{aligned} |E| &\geq 3 \cdot B_{\uparrow}(X) + 3 \cdot n_{\updownarrow}(X) + |X|_{\mathcal{A}} \quad \textit{and} \\ |E| &\geq 3 \cdot B_{\downarrow}(X) + 3 \cdot n_{\updownarrow}(X) + |X|_{\mathcal{A}}. \end{aligned} \tag{4.1}$$

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,* $|E| \geq 3 + 3 \cdot n_{\updownarrow}(X) + |X|_{\mathcal{A}}$.

Chapter 5

The Construction of Minimal DNA Expressions

Definition 5.1 (See [Van Vliet, 2004, Definition 4.20], [Van Vliet et al., 2005, page 382], [Van Vliet et al., 2006, page 140]) *A DNA expression E is minimal if for every DNA expression E' with $E' \equiv E$, $|E'| \geq |E|$.*

Example 5.2 (See [Van Vliet, 2004, page 58]) 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 4.7(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. ■

5.1 Minimal DNA expressions for a nick free formal DNA molecule

Theorem 5.3 (See [Van Vliet, 2004, Theorem 4.23], [Van Vliet et al., 2005, Theorem 9(1)], [Van Vliet et al., 2006, Theorem 9]) *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 $\mathcal{S}(E) = \binom{\alpha_1}{c(\alpha_1)}$.

In § 4.1, we defined the primitive $\uparrow$-blocks and primitive $\downarrow$-blocks of a formal DNA molecule. Here, these notions appear 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.

Definition 5.4 (See [Van Vliet, 2004, Definition 4.47, Definition 4.26 and Definition 4.43]) *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 $X = Y_0 X_1 Y_1 \ldots X_{r_0} Y_{r_0}$.*

- *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 $(Y_0X_1Y_1\dots X_j, X_{j+1}Y_{j+1}\dots X_{r_0}Y_{r_0})$ of a substring Y_j with $0 \le j \le r_0$ and $Y_j \neq \lambda$.*
- *The maximal upper prefix of X is the occurrence $(\lambda, X_1Y_1\dots X_{r_0}Y_{r_0})$ of Y_0.*
- *The maximal upper suffix of X is the occurrence $(Y_0X_1Y_1\dots X_{r_0}, \lambda)$ of Y_{r_0}.*
- *An internal maximal upper sequence of X is the occurrence $(Y_0X_1Y_1\dots X_j, X_{j+1}Y_{j+1}\dots X_{r_0}Y_{r_0})$ of a substring Y_j with $1 \le j \le r_0 - 1$.*

Hence, if $r_0 \ge 1$, 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 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.

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.

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.

Lemma 5.5 (See [Van Vliet, 2004, Lemma 4.10(2) and Lemma 4.46(2)]) *Let X be a nick free formal DNA molecule.*

1. *If $B_\downarrow(X) = 0$, then $n_{imus}(X) = 0$.*
2. *If $B_\downarrow(X) \ge 1$, then $n_{imus}(X) = B_\downarrow(X) - 1$.*

Lemma 5.6 (See [Van Vliet, 2004, Lemma 4.31(2)-(3), Lemma 4.50, Lemma 4.44, Lemma 4.45 and Lemma 4.49(2)]) *Let X be a nick free formal DNA molecule*

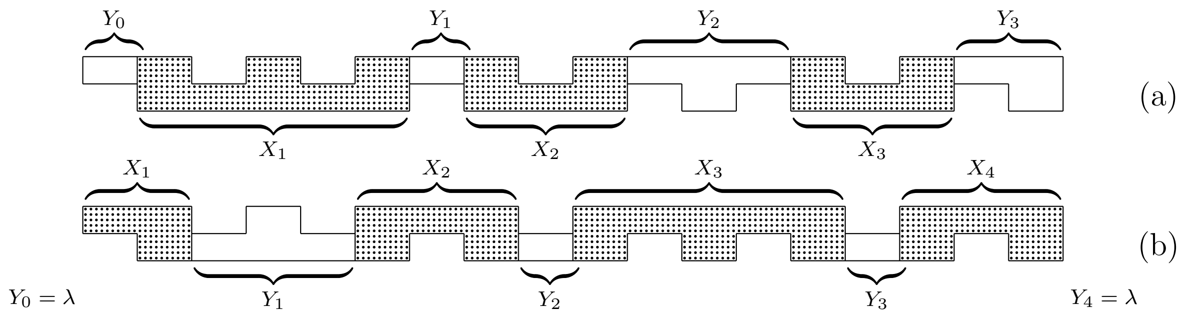


Figure 5.1: (Cf. [Van Vliet, 2004, Figure 4.5 and Figure 4.6]) (See [Van Vliet et al., 2006, Figure 4(a)]) Two partitionings of a nick free formal DNA molecule X. (a) The primitive lower block partitioning of X. X_1, X_2, X_3 are the primitive lower blocks of X, Y_0, Y_1, Y_2, Y_3 are the maximal upper 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.

and let $Y_0 X_1 Y_1 \dots X_{r_0} Y_{r_0}$ 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 4.4(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 4.4(1) ends with a primitive lower block.*

(d) *The last single-stranded component of X is a lower component.*

Lemma 5.7 (See [Van Vliet, 2004, Lemma 4.27, Definition 4.26, Lemma 4.30 and Lemma 4.31(1)]) *Let X be a nick free formal DNA molecule and let $x'_1 \dots x'_k$ be the decomposition of X.*

1. *Let $Y = x'_{b_0} \dots x'_{b_1}$ 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.*

Definition 5.8 (Cf. [Van Vliet, 2004, Lemma 4.51(3a)]) (See [Van Vliet et al., 2006, Definition 10]) *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.*

A lower block is an occurrence $(Y_0X_1Y_1 \dots Y_{j_1-1}, Y_{j_2}X_{j_2+1} \dots X_{r_0}Y_{r_0})$ of a substring $X_{j_1}Y_{j_1} \dots X_{j_2}$ 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.

Definition 5.9 (Cf. [Van Vliet, 2004, Definition 4.47]) (See [Van Vliet et al., 2006, Definition 10]) *Let X be a nick free formal DNA molecule.*

A lower block partitioning of X is a sequence $Y_0, \overline{X}_1, Y_1, \dots, \overline{X}_r, Y_r$ for some $r \geq 0$ such that

- *$X = Y_0\overline{X}_1Y_1 \dots \overline{X}_rY_r$, and*
- *for $j = 1, \dots, r$, $\overline{X}_j$ is a lower block of X, and*

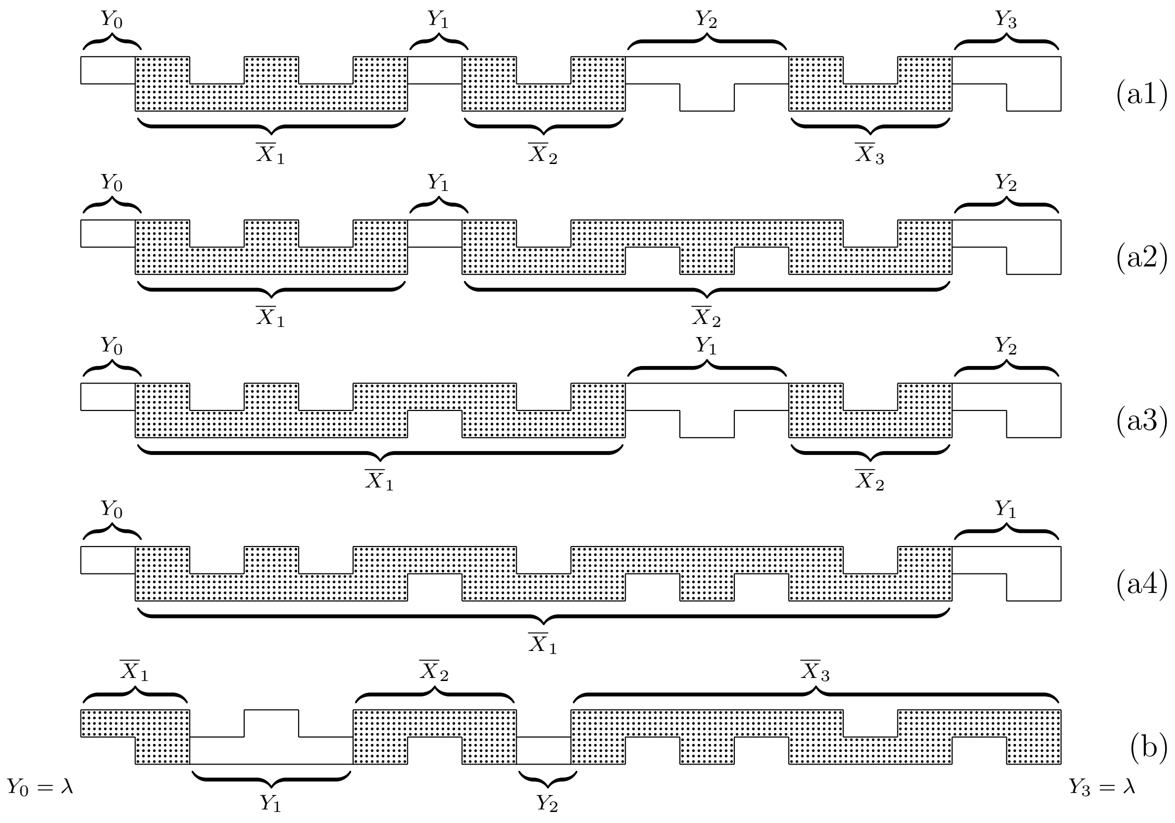


Figure 5.2: (See [Van Vliet, 2004, Figure 4.7], [Van Vliet et al., 2005, Figure 3], [Van Vliet et al., 2006, Figure 4]) Different partitionings of the formal DNA molecule X from Figure 5.1, for which $B_\uparrow(X) = 4$ and $B_\downarrow(X) = 3$. (a1) (Once more) the primitive lower block partitioning 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.

- *for each primitive lower block X_1 of X, there is a j with $1 \le j \le r$, such that X_1 is contained in $\overline{X}_j$.*

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.

Lemma 5.10 (See [Van Vliet, 2004, Lemma 4.10(2) and Lemma 4.50]) *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.*

Lemma 5.11 (See [Van Vliet, 2004, Lemma 4.48]) *Let X be a nick free formal DNA molecule. Then the number of different lower block partitionings of X is $2^{n_{imus}(X)}$.*

Theorem 5.12 (See [Van Vliet, 2004, Theorem 4.53], [Van Vliet et al., 2006, Theorem 13]) (Cf. [Van Vliet et al., 2005, Theorem 9(2)-(4)]) *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*
   $$\varepsilon_i = \begin{cases} \alpha_i & \text{if } x'_i = \binom{\alpha_i}{-} \text{ for an } \mathcal{N}\text{-word } \alpha_i \\ \langle\updownarrow \alpha_i\rangle & \text{if } x'_i = \binom{\alpha_i}{c(\alpha_i)} \text{ for an } \mathcal{N}\text{-word } \alpha_i; \end{cases} \quad \text{and} \tag{5.1}$$
   - *let*
   $$E = \langle\uparrow \varepsilon_{a_0} \dots \varepsilon_{b_0} E_1 \varepsilon_{a_1} \dots \varepsilon_{b_1} \ \dots \ E_r \varepsilon_{a_r} \dots \varepsilon_{b_r}\rangle . \tag{5.2}$$

   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*
   $$|E| = 3 + 3 \cdot B_\downarrow(X) + 3 \cdot n_\updownarrow(X) + |X|_\mathcal{A}. \tag{5.3}$$

2. *If $B_\downarrow(X) \geq B_\uparrow(X)$, then*
   - *let $\mathcal{P} = Y_0\overline{X}_1Y_1 \dots \overline{X}_rY_r$ for some $r \geq 0$ be an arbitrary upper 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*
   $$\varepsilon_i = \begin{cases} \alpha_i & \text{if } x'_i = \binom{-}{\alpha_i} \text{ for an } \mathcal{N}\text{-word } \alpha_i \\ \langle\updownarrow \alpha_i\rangle & \text{if } x'_i = \binom{\alpha_i}{c(\alpha_i)} \text{ for an } \mathcal{N}\text{-word } \alpha_i; \end{cases} \quad \text{and}$$
   - *let*
   $$E = \langle\downarrow \varepsilon_{a_0} \dots \varepsilon_{b_0} E_1 \varepsilon_{a_1} \dots \varepsilon_{b_1} \ \dots \ E_r \varepsilon_{a_r} \dots \varepsilon_{b_r}\rangle . \tag{5.4}$$

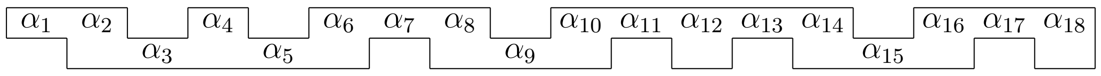

α_1 α_2 α_3 α_4 α_5 α_6 α_7 α_8 α_9 α_{10} α_{11} α_{12} α_{13} α_{14} α_{15} α_{16} α_{17} α_{18}

Figure 5.3: (See [Van Vliet, 2004, Figure 4.8], [Van Vliet et al., 2005, Figure 3], [Van Vliet et al., 2006, Figure 4]) The formal DNA molecule from Figure 5.1 with occurring $\mathcal{N}$-words indicated.

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

$$|E| = 3 + 3 \cdot B_{\uparrow}(X) + 3 \cdot n_{\updownarrow}(X) + |X|_{\mathcal{A}}. \tag{5.5}$$

Example 5.13 (See [Van Vliet, 2004, pages 81-82], [Van Vliet et al., 2005, pages 383-384], [Van Vliet et al., 2006, pages 143-144]) In Figure 5.3, we have specified names for the components of the formal DNA molecule from (a.o.) Figure 5.1 and Figure 5.2. For this formal DNA molecule X, we have $B_{\uparrow}(X) = 4$ and $B_{\downarrow}(X) = 3$. Hence, by Theorem 5.12(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 5.11, four different lower block partitionings of X. We will consider two of them, the ones depicted in Figure 5.2(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 5.12(2) to $\overline{X}_1$ and Theorem 5.12(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 5.12(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 5.12(1), a minimal DNA expression denoting X is

$$\begin{aligned}
E = \langle\uparrow \quad \alpha_1 \quad & \langle\downarrow \langle\updownarrow \alpha_2\rangle\, \alpha_3 \,\langle\updownarrow \alpha_4\rangle\, \alpha_5 \,\langle\uparrow \langle\updownarrow \alpha_6\rangle\, \alpha_7 \,\langle\updownarrow \alpha_8\rangle\rangle\, \alpha_9 \,\langle\updownarrow \alpha_{10}\rangle\rangle \\
& \alpha_{11} \,\langle\updownarrow \alpha_{12}\rangle\, \alpha_{13} \quad \langle\downarrow \langle\updownarrow \alpha_{14}\rangle\, \alpha_{15} \,\langle\updownarrow \alpha_{16}\rangle\rangle \quad \alpha_{17} \,\langle\updownarrow \alpha_{18}\rangle \quad \rangle\,.
\end{aligned} \tag{5.6}$$

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 5.2(a4), $r = 1$ and

$$\begin{aligned} Y_0 &= \binom{\alpha_1}{-}, \\ \overline{X}_1 &= \binom{\alpha_2}{c(\alpha_2)}\binom{-}{\alpha_3}\binom{\alpha_4}{c(\alpha_4)}\binom{-}{\alpha_5}\binom{\alpha_6}{c(\alpha_6)}\binom{\alpha_7}{-}\binom{\alpha_8}{c(\alpha_8)}\binom{-}{\alpha_9} \\ &\quad \cdot \binom{\alpha_{10}}{c(\alpha_{10})}\binom{\alpha_{11}}{-}\binom{\alpha_{12}}{c(\alpha_{12})}\binom{\alpha_{13}}{-}\binom{\alpha_{14}}{c(\alpha_{14})}\binom{-}{\alpha_{15}}\binom{\alpha_{16}}{c(\alpha_{16})}, \\ Y_1 &= \binom{\alpha_{17}}{-}\binom{\alpha_{18}}{c(\alpha_{18})}. \end{aligned}$$

We now have $B_{\downarrow}(\overline{X}_1) = 3$ and $B_{\uparrow}(\overline{X}_1) = 2$. By Theorem 5.12(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 5.12(1). The resulting minimal DNA-expression for $\overline{X}_1$ is

$$\begin{aligned} E_1 = \langle\downarrow\ &\langle\updownarrow \alpha_2\rangle\, \alpha_3\, \langle\updownarrow \alpha_4\rangle\, \alpha_5\ \ \langle\uparrow \langle\updownarrow \alpha_6\rangle\, \alpha_7\, \langle\updownarrow \alpha_8\rangle\rangle\ \ \alpha_9 \\ &\langle\uparrow \langle\updownarrow \alpha_{10}\rangle\, \alpha_{11}\, \langle\updownarrow \alpha_{12}\rangle\, \alpha_{13}\, \langle\updownarrow \alpha_{14}\rangle\rangle\ \ \alpha_{15}\, \langle\updownarrow \alpha_{16}\rangle\ \rangle \end{aligned}$$

and the corresponding minimal DNA-expression denoting X is

$$\begin{aligned} E = \langle\uparrow\ \ \alpha_1\ \ \langle\downarrow\ &\langle\updownarrow \alpha_2\rangle\, \alpha_3\, \langle\updownarrow \alpha_4\rangle\, \alpha_5\ \ \langle\uparrow \langle\updownarrow \alpha_6\rangle\, \alpha_7\, \langle\updownarrow \alpha_8\rangle\rangle\ \ \alpha_9 \\ &\langle\uparrow \langle\updownarrow \alpha_{10}\rangle\, \alpha_{11}\, \langle\updownarrow \alpha_{12}\rangle\, \alpha_{13}\, \langle\updownarrow \alpha_{14}\rangle\rangle\ \ \alpha_{15}\, \langle\updownarrow \alpha_{16}\rangle\ \rangle\ \ \ \alpha_{17}\, \langle\updownarrow \alpha_{18}\rangle\ \ \ \rangle\,. \end{aligned}$$

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 5.14 Consider the nick free formal DNA molecule

$$X = \binom{\alpha_1}{-}\binom{\alpha_2}{c(\alpha_2)}\binom{-}{\alpha_3}\binom{\alpha_4}{c(\alpha_4)}\binom{\alpha_5}{-}\binom{\alpha_6}{c(\alpha_6)}\binom{\alpha_7}{-}\binom{\alpha_8}{c(\alpha_8)}\binom{-}{\alpha_9}\binom{\alpha_{10}}{c(\alpha_{10})}, \tag{5.7}$$

for which $B_{\uparrow}(X) = B_{\downarrow}(X) = 2$. By Theorem 5.12, we can construct minimal $\uparrow$-expressions (based on lower block partitionings) and minimal $\downarrow$-expressions (based on upper block partitionings) for X. By Lemma 5.11 and Lemma 5.5(2), X has two lower block partitionings and two upper block partitionings. We have depicted them in Figure 5.4. We carry out the construction for the upper block partitioning $Y_0\overline{X}_1Y_1$ from Figure 5.4(d). Here $Y_0 = \lambda$,

$$\overline{X}_1 = \binom{\alpha_1}{-}\binom{\alpha_2}{c(\alpha_2)}\binom{-}{\alpha_3}\binom{\alpha_4}{c(\alpha_4)}\binom{\alpha_5}{-}\binom{\alpha_6}{c(\alpha_6)}\binom{\alpha_7}{-}\binom{\alpha_8}{c(\alpha_8)}$$

and $Y_1 = \binom{-}{\alpha_9}\binom{\alpha_{10}}{c(\alpha_{10})}$. By Theorem 5.12(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 $\overline{X}_1$.

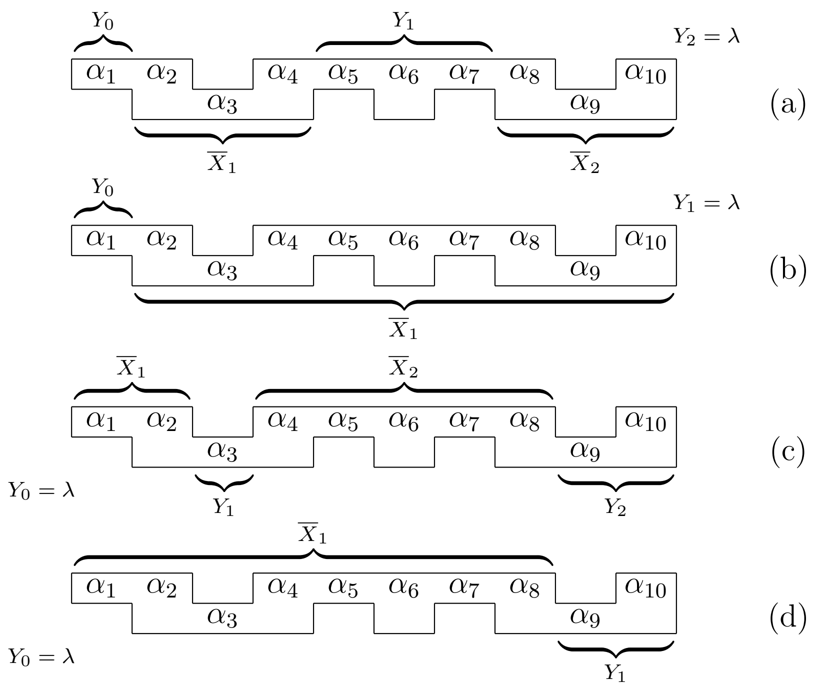

Y_0
Y_1
$Y_2 = \lambda$
α_1 α_2 α_3 α_4 α_5 α_6 α_7 α_8 α_9 α_{10}
$\overline{X}_1$
$\overline{X}_2$
(a)
$Y_1 = \lambda$
(b)
$Y_0 = \lambda$
(c)
(d)

Figure 5.4: Partitionings of the formal DNA molecule X from Example 5.14. (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.

As $B_\uparrow(\overline{X}_1) = 2 > B_\downarrow(\overline{X}_1) = 1$, we can (recursively) apply Theorem 5.12(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 5.4:

$$\begin{aligned}
E_a &= \langle\uparrow \alpha_1 \quad \langle\downarrow \langle\updownarrow \alpha_2\rangle\, \alpha_3 \langle\updownarrow \alpha_4\rangle\rangle \quad \alpha_5 \langle\updownarrow \alpha_6\rangle\, \alpha_7 \quad \langle\downarrow \langle\updownarrow \alpha_8\rangle\, \alpha_9 \langle\updownarrow \alpha_{10}\rangle\rangle\rangle\, , && (5.8)\\
E_b &= \langle\uparrow \alpha_1 \quad \langle\downarrow \langle\updownarrow \alpha_2\rangle\, \alpha_3 \langle\uparrow \langle\updownarrow \alpha_4\rangle\, \alpha_5 \langle\updownarrow \alpha_6\rangle\, \alpha_7 \langle\updownarrow \alpha_8\rangle\rangle\, \alpha_9 \langle\updownarrow \alpha_{10}\rangle\rangle\rangle\, , && (5.9)\\
E_c &= \langle\downarrow \langle\uparrow \alpha_1 \langle\updownarrow \alpha_2\rangle\rangle \quad \alpha_3 \quad \langle\uparrow \langle\updownarrow \alpha_4\rangle\, \alpha_5 \langle\updownarrow \alpha_6\rangle\, \alpha_7 \langle\updownarrow \alpha_8\rangle\rangle \quad \alpha_9 \langle\updownarrow \alpha_{10}\rangle\rangle\, , && (5.10)\\
E_d &= \langle\downarrow \langle\uparrow \alpha_1 \langle\downarrow \langle\updownarrow \alpha_2\rangle\, \alpha_3 \langle\updownarrow \alpha_4\rangle\rangle\, \alpha_5 \langle\updownarrow \alpha_6\rangle\, \alpha_7 \langle\updownarrow \alpha_8\rangle\rangle \quad \alpha_9 \langle\updownarrow \alpha_{10}\rangle\rangle\, . && (5.11)
\end{aligned}$$

All these minimal DNA expressions have length

$$\begin{aligned}
|E| &= 24 + |X|_{\mathcal{A}} = 3 + 3\cdot 2 + 3 \cdot 5 + |X|_{\mathcal{A}}\\
&= 3 + 3\cdot B_\downarrow(X) + 3\cdot n_\updownarrow(X) + |X|_{\mathcal{A}} = 3 + 3\cdot B_\uparrow(X) + 3\cdot n_\updownarrow(X) + |X|_{\mathcal{A}}.
\end{aligned}$$

■

Lemma 5.15 (See [Van Vliet, 2004, Lemma 4.57]) *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 5.12 (equation (5.2) or equation (5.4)).*

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.

Lemma 5.16 (See [Van Vliet, 2004, Lemma 4.58], [Van Vliet et al., 2005, Theorem 9(3)-(4)]) *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.*

Corollary 5.17 (See [Van Vliet, 2004, Corollary 4.59]) *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 5.12(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 5.12(2) is an $\uparrow$-expression.

Note that this result is trivially valid if X is double-complete. In that case, by Lemma 5.10, 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 5.18 *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 5.12(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 5.12(2).

Proof:

1. $\Longrightarrow$ Assume that the first single-stranded component of X is a lower component. Then by Lemma 5.6(3a) and (3d), the maximal upper prefix Y_0 of X is empty. By the construction from Theorem 5.12(1) and Corollary 5.17(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 5.12(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 5.6(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 4.6(3):

Corollary 5.19 *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 5.12(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 5.12(2).

By Theorem 5.3, we know that for a double-complete formal DNA molecule, there is exactly one minimal DNA expression. Theorem 5.12, 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 5.20 *The language of all minimal DNA expressions is not regular.*

Note that by Lemma 2.16, 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 $\mathcal{P}_l = Y_0\overline{X}_1Y_1$ 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

$$E_l = \left(\langle\uparrow \langle\updownarrow \alpha\rangle\, \alpha \,\langle\downarrow \langle\updownarrow \alpha\rangle\, \alpha\right)^l \langle\uparrow \langle\updownarrow \alpha\rangle\, \alpha\, \langle\updownarrow \alpha\rangle\rangle \left(\alpha\, \langle\updownarrow \alpha\rangle\, \rangle\, \alpha\, \langle\updownarrow \alpha\rangle\, \rangle\right)^l$$

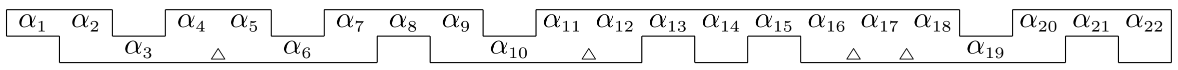

α_1 α_2 α_3 α_4 α_5 $\triangle$ α_6 α_7 α_8 α_9 α_{10} α_{11} $\triangle$ α_{12} α_{13} α_{14} α_{15} α_{16} $\triangle$ α_{17} $\triangle$ α_{18} α_{19} α_{20} α_{21} α_{22}

Figure 5.5: (See [Van Vliet, 2004, Figure 4.9]) (Cf. [Van Vliet et al., 2005, Figure 4], [Van Vliet et al., 2006, Figure 7]) A formal DNA molecule containing lower nick letters.

is a minimal DNA expression denoting X_l based on $\mathcal{P}_l$ as described in Theorem 5.12. It follows from the pumping lemma for regular languages, that a language requiring brackets to match and containing such DNA expressions is not regular. □

5.2 Minimal DNA expressions for a formal DNA molecule with nick letters

Definition 5.21 (See [Van Vliet, 2004, Definition 4.61], [Van Vliet et al., 2005, page 384], [Van Vliet et al., 2006, page 146]) *Let X be a formal DNA molecule. The nick free decomposition of X is the sequence $Z_1, y_1, Z_2, y_2, \dots, y_{m-1}, Z_m$ for some $m \geq 1$ such that*

- *$X = Z_1 y_1 Z_2 y_2 \dots y_{m-1} Z_m$, and*
- *for $h = 1, \dots, m$, Z_h is nick free, and*
- *for $h = 1, \dots, m-1$, $y_h \in \{^\triangledown, {}_\triangle\}$.*

Example 5.22 (See [Van Vliet, 2004, page 90]) (Cf. [Van Vliet et al., 2005, pages 384], [Van Vliet et al., 2006, Figure 7]) Consider the formal DNA molecule X depicted in Figure 5.5. 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

$$\begin{aligned}
Z_1 &= \binom{\alpha_1}{-}\binom{\alpha_2}{c(\alpha_2)}\binom{-}{\alpha_3}\binom{\alpha_4}{c(\alpha_4)}, \\
Z_2 &= \binom{\alpha_5}{c(\alpha_5)}\binom{-}{\alpha_6}\binom{\alpha_7}{c(\alpha_7)}\binom{\alpha_8}{-}\binom{\alpha_9}{c(\alpha_9)}\binom{-}{\alpha_{10}}\binom{\alpha_{11}}{c(\alpha_{11})}, \\
Z_3 &= \binom{\alpha_{12}}{c(\alpha_{12})}\binom{\alpha_{13}}{-}\binom{\alpha_{14}}{c(\alpha_{14})}\binom{\alpha_{15}}{-}\binom{\alpha_{16}}{c(\alpha_{16})}, \\
Z_4 &= \binom{\alpha_{17}}{c(\alpha_{17})}, \\
Z_5 &= \binom{\alpha_{18}}{c(\alpha_{18})}\binom{-}{\alpha_{19}}\binom{\alpha_{20}}{c(\alpha_{20})}\binom{\alpha_{21}}{-}\binom{\alpha_{22}}{c(\alpha_{22})}.
\end{aligned} \tag{5.12}$$

■

Definition 5.23 (See [Van Vliet, 2004, Definition 4.63], [Van Vliet et al., 2005, page 384], [Van Vliet et al., 2006, page 146]) *A DNA expression E is operator-minimal if for every DNA expression E' with the same outermost operator as E and with $E' \equiv E$, $|E'| \geq |E|$.*

Example 5.24 (See [Van Vliet, 2004, page 91], [Van Vliet et al., 2005, page 384-385], [Van Vliet et al., 2006, page 146]) We continue with the formal DNA molecule X from Example 5.22, which is depicted in Figure 5.5. The second formal DNA submolecule occurring in the nick free decomposition of X is

$$Z_2 = \binom{\alpha_5}{c(\alpha_5)}\binom{-}{\alpha_6}\binom{\alpha_7}{c(\alpha_7)}\binom{\alpha_8}{-}\binom{\alpha_9}{c(\alpha_9)}\binom{-}{\alpha_{10}}\binom{\alpha_{11}}{c(\alpha_{11})} \tag{5.13}$$

(see (5.12)). We have $B_\uparrow(Z_2) = 1$ and $B_\downarrow(Z_2) = 2$.

By Lemma 5.16(2), each minimal DNA expression E_2 denoting Z_2 is a $\downarrow$-expression. When we apply Theorem 5.12 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 4.7(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 4.1, 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

$$E'_2 = \langle\uparrow E_2\rangle = \langle\uparrow \langle\downarrow \langle\updownarrow \alpha_5\rangle\, \alpha_6 \langle\uparrow \langle\updownarrow \alpha_7\rangle\, \alpha_8 \langle\updownarrow \alpha_9\rangle\rangle\, \alpha_{10} \langle\updownarrow \alpha_{11}\rangle\rangle\rangle\,, \tag{5.14}$$

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

$$E''_2 = \langle\uparrow \langle\downarrow \langle\updownarrow \alpha_5\rangle\, \alpha_6 \langle\updownarrow \alpha_7\rangle\rangle\, \alpha_8 \langle\downarrow \langle\updownarrow \alpha_9\rangle\, \alpha_{10} \langle\updownarrow \alpha_{11}\rangle\rangle\rangle\,. \tag{5.15}$$

■

Lemma 5.25 (See [Van Vliet, 2004, Lemma 4.64]) *If a DNA expression E is operator-minimal, then each proper DNA subexpression of E is minimal.*

Theorem 5.26 (See [Van Vliet, 2004, Theorem 4.65], [Van Vliet et al., 2005, page 385], [Van Vliet et al., 2006, page 146]) *Let X be a nick free formal DNA molecule and let $x'_1 \dots x'_k$ for some $k \geq 1$ be the decomposition of X.*

- *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*

$$\varepsilon_i = \begin{cases} \alpha_i & \text{if } x'_i = \binom{\alpha_i}{-} \text{ for an } \mathcal{N}\text{-word } \alpha_i \\ \langle\updownarrow \alpha_i\rangle & \text{if } x'_i = \binom{\alpha_i}{c(\alpha_i)} \text{ for an } \mathcal{N}\text{-word } \alpha_i; \end{cases} \qquad \textit{and}$$

- *let*

$$E = \langle \uparrow \varepsilon_{a_0} \dots \varepsilon_{b_0} E_1 \varepsilon_{a_1} \dots \varepsilon_{b_1} \ \dots \ E_r \varepsilon_{a_r} \dots \varepsilon_{b_r} \rangle . \tag{5.16}$$

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

$$|E| = 3 + 3 \cdot B_{\downarrow}(X) + 3 \cdot n_{\updownarrow}(X) + |X|_{\mathcal{A}}. \tag{5.17}$$

Example 5.27 (See [Van Vliet, 2004, page 93]) Indeed, the two operator-minimal $\uparrow$-expressions E_2' and E_2'' we have given in Example 5.24, which denote the formal DNA molecule Z_2 from (5.13), can be constructed according to the description in Theorem 5.26. 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 5.11, 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

$$\begin{aligned} \overline{X}_1'' &= \binom{\alpha_5}{c(\alpha_5)}\binom{-}{\alpha_6}\binom{\alpha_7}{c(\alpha_7)}, \\ Y_1'' &= \binom{\alpha_8}{-}, \\ \overline{X}_2'' &= \binom{\alpha_9}{c(\alpha_9)}\binom{-}{\alpha_{10}}\binom{\alpha_{11}}{c(\alpha_{11})}. \end{aligned}$$

The DNA expression E_2' from (5.14) corresponds to $\mathcal{P}'$ and the DNA expression E_2'' from (5.15) corresponds to $\mathcal{P}''$. ■

Theorem 5.28 (See [Van Vliet, 2004, Theorem 4.67], [Van Vliet et al., 2005, Theorem 10], [Van Vliet et al., 2006, Theorem 14]) *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 $Z_1 {}_\triangle Z_2 {}_\triangle \dots {}_\triangle Z_m$ for some $m \geq 2$ be the nick free decomposition of X;*
- *for $h = 1, \dots, m$, 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 $E_h = \langle \uparrow \varepsilon_{h,1} \dots \varepsilon_{h,n_h} \rangle$ for some $n_h \geq 1$ and $\mathcal{N}$-words and DNA expressions $\varepsilon_{h,1}, \dots, \varepsilon_{h,n_h}$, then $\widehat{E}_h = \varepsilon_{h,1} \dots \varepsilon_{h,n_h}$); and*
- *let $E = \left\langle \uparrow \widehat{E}_1 \dots \widehat{E}_m \right\rangle$.*

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

(b) E is a minimal DNA expression denoting X, and

$$|E| = 3 + 3 \cdot B_{\downarrow}(X) + 3 \cdot n_{\updownarrow}(X) + |X|_{\mathcal{A}}. \tag{5.18}$$

Example 5.29 (See [Van Vliet, 2004, pages 94-95]) (Cf. [Van Vliet et al., 2005, pages 385-386], [Van Vliet et al., 2006, pages 147-148]) In Example 5.22, we have established that the nick free decomposition for the formal DNA molecule from Figure 5.5 is $Z_1 {}_\triangle Z_2 {}_\triangle Z_3 {}_\triangle Z_4 {}_\triangle Z_5$, where $Z_1, \ldots, Z_5$ are given in (5.12). 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 5.26 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 5.27, the DNA expressions E_2' and E_2'' from (5.14) and (5.15) 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

$$\begin{aligned}
E = \langle\uparrow \quad & \alpha_1 \langle\downarrow \langle\updownarrow \alpha_2\rangle\, \alpha_3 \langle\updownarrow \alpha_4\rangle\rangle \quad \langle\downarrow \langle\updownarrow \alpha_5\rangle\, \alpha_6 \langle\updownarrow \alpha_7\rangle\rangle\, \alpha_8 \langle\downarrow \langle\updownarrow \alpha_9\rangle\, \alpha_{10} \langle\updownarrow \alpha_{11}\rangle\rangle \\
& \langle\updownarrow \alpha_{12}\rangle\, \alpha_{13} \langle\updownarrow \alpha_{14}\rangle\, \alpha_{15} \langle\updownarrow \alpha_{16}\rangle \quad \langle\updownarrow \alpha_{17}\rangle \\
& \langle\downarrow \langle\updownarrow \alpha_{18}\rangle\, \alpha_{19} \langle\updownarrow \alpha_{20}\rangle\rangle\, \alpha_{21} \langle\updownarrow \alpha_{22}\rangle \quad \rangle\,.
\end{aligned} \tag{5.19}$$

Indeed,

$$|E| = 54 + |X|_{\mathcal{A}} = 3 + 3 \cdot 4 + 3 \cdot 13 + |X|_{\mathcal{A}} = 3 + 3 \cdot B_{\downarrow}(X) + 3 \cdot n_{\updownarrow}(X) + |X|_{\mathcal{A}}.$$

■

Chapter 6

All Minimal DNA Expressions

6.1 Reverse construction of a minimal DNA expression

Lemma 6.1 (See [Van Vliet, 2004, Lemma 4.69]) *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.

Corollary 6.2 (See [Van Vliet, 2004, Corollary 4.70]) *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.

Lemma 6.3 (See [Van Vliet, 2004, Lemma 4.71]) *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.

Corollary 6.4 (Cf. [Van Vliet, 2004, Corollary 4.72]) *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 5.25, E_1 is minimal. Hence, if E_1 is an $\updownarrow$-subexpression of E, then by Theorem 5.3, $E_1 = \langle\updownarrow \alpha_1\rangle$ 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 5.3, and Lemma 6.1, 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 6.3, 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. $\square$

Lemma 6.5 (See [Van Vliet, 2004, Lemma 4.74]) *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. $$\sum_{\downarrow\text{-expr. } \varepsilon_i} B_\downarrow(X_i) = B_\downarrow(X_1) + \dots + B_\downarrow(X_n) = B_\downarrow(X).$$

Lemma 6.6 (See [Van Vliet, 2004, Corollary 4.73(1)]) *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. For $i = 1, \dots, n$, let $X_i = \mathcal{S}^+(\varepsilon_i)$.*

Then

$$X = X_1 y_1 X_2 y_2 \dots y_{n-1} X_n,$$

where for $i = 1, \dots, n-1$, $y_i = \vartriangle$ if $R(X_i), L(X_{i+1}) \in \mathcal{A}_\pm$, and $y_i = \lambda$ otherwise.

Here, for $i = 1, \dots, n-1$, $R(X_i), L(X_{i+1}) \in \mathcal{A}_\pm$, if and only if both ε_i and ε_{i+1} are expression-arguments.

Proof of second part of the claim: Consider any i with $1 \leq i \leq n-1$. By Corollary 6.2, 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. By Lemma 6.3, $X_i = \mathcal{S}^+(\varepsilon_i)$ is nick free.

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 6.5(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 2.9(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. □

Corollary 6.7 (See [Van Vliet, 2004, Corollary 4.73(2)]) *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

$$X = X_1 X_2 \dots X_n, \tag{6.1}$$

and the arguments $\varepsilon_1, \dots, \varepsilon_n$ are maximal $\mathcal{N}$-word occurrences and DNA expressions, alternately.

Corollary 6.8 (Cf. [Van Vliet, 2004, Lemma 4.76(3)]) *Let E be an operator-minimal $\uparrow$-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 $\uparrow$ of) E is alternating.*
4. *Each occurrence of $\uparrow$ or $\downarrow$ in E is alternating.*

Proof: The equivalence of statements 1 and 2 follows from Lemma 3.1(1). We now prove that statements 3 and 4 are also equivalent to statement 1.

1 $\Longrightarrow$ 3 This implication follows directly from Corollary 6.7.

3 $\Longleftrightarrow$ 4 Consider an arbitrary *inner* occurrence of $\uparrow$ or $\downarrow$ in E, and let E^s be the (proper) DNA subexpression of E governed by it. By Lemma 5.25, E^s is minimal, and by Corollary 6.4, E^s is nick free. Hence, we can apply Corollary 6.7 to E^s, and conclude that the occurrence of $\uparrow$ or $\downarrow$ that we consider is alternating.

This implies that the outermost operator $\uparrow$ of E is alternating, if and only if *each* occurrence of $\uparrow$ or $\downarrow$ in E is alternating.

4 $\Longrightarrow$ 1 This implication follows directly from Lemma 3.5.

$\square$

Theorem 6.9 (See [Van Vliet, 2004, Theorem 4.77]) (Cf. [Van Vliet et al., 2005, page 384], [Van Vliet et al., 2006, page 143]) *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 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 a minimal DNA expression denoting X and based on $\mathcal{P}$, given in Theorem 5.12(1).*

Theorem 6.10 (See [Van Vliet, 2004, Theorem 4.78]) *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 $\varepsilon_{i_1}, \varepsilon_{i_2}, \ldots, \varepsilon_{i_r}$, with $0 \leq r \leq n$ and $i_1 < i_2 < \cdots < i_r$, be all $\downarrow$-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 an operator-minimal DNA expression denoting X and based on $\mathcal{P}$, given in Theorem 5.26.*

Theorem 6.11 (See [Van Vliet, 2004, Theorem 4.79]) (Cf. [Van Vliet et al., 2005, Theorem 10], [Van Vliet et al., 2006, Theorem 14]) *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 formal DNA molecule X which contains at least one nick letter. Let $Z_1 {}_\triangle Z_2 {}_\triangle \ldots {}_\triangle Z_m$ 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 5.28. Hence, there exist indices $i_0, i_1, \ldots, i_m$, such that

- *$i_0 = 0 < i_1 < i_2 < \cdots < i_{m-1} < i_m = n$, and*
- *for $h = 1, \ldots, m$, $\langle \uparrow \varepsilon_{i_{h-1}+1} \ldots \varepsilon_{i_h} \rangle$ is an operator-minimal $\uparrow$-expression denoting Z_h.*

Summary 6.12 (See [Van Vliet, 2004, Corollary 4.80]) *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 5.3).*

   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 5.12(1), and $\downarrow$-expressions based on an upper block partitioning of X as described in Theorem 5.12(2) (see also Theorem 6.9).*

   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 5.12(1) (see also Theorem 6.9).*

   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 5.12(2) (see also Theorem 6.9).*

   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 5.28 (see also Theorem 6.11).*

   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 5.26 (see also Theorem 6.10).

   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 5.28 (see also Theorem 6.11).*

   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 5.26 (see also Theorem 6.10).

   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 4.7.

Corollary 6.13 (See [Van Vliet, 2004, Corollary 4.81]) *Let E be a DNA expression, and let $X = \mathcal{S}(E)$.*

1. *If E is an operator-minimal $\uparrow$-expression, then*

$$\begin{aligned} \#_{\uparrow,\downarrow}(E) &= 1 + B_{\downarrow}(X) \quad \text{and} \\ \#_{\updownarrow}(E) &= n_{\updownarrow}(X). \end{aligned}$$

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*

$$\begin{aligned}\#_{\uparrow,\downarrow}(E) &= 0 \quad \textit{and}\\ \#_{\updownarrow}(E) &= 1.\end{aligned}$$

Lemma 6.14 (See [Van Vliet, 2004, pages 110-111]) *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 $\binom{\alpha_1}{c(\alpha_1)} \triangle \binom{\alpha_2}{c(\alpha_2)} \triangle \cdots \triangle \binom{\alpha_m}{c(\alpha_m)}$ 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*

$$E = \langle \uparrow \langle \updownarrow \alpha_1 \rangle \langle \updownarrow \alpha_2 \rangle \ldots \langle \updownarrow \alpha_m \rangle \rangle .$$

6.2 Characterization of minimal DNA expressions

Lemma 6.15 (See [Van Vliet, 2004, Definition 4.97 and Theorem 4.100], [Van Vliet et al., 2005, Theorem 12]) *Let E be a minimal DNA expression.*

($\mathcal{D}_{\mathbf{Min}}$.1) *Each occurrence of the operator $\updownarrow$ in E has as its argument an $\mathcal{N}$-word α (i.e., not a DNA expression).*

($\mathcal{D}_{\mathbf{Min}}$.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.*

($\mathcal{D}_{\mathbf{Min}}$.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.*

($\mathcal{D}_{\mathbf{Min}}$.4) *Each inner occurrence of an operator $\uparrow$ or $\downarrow$ in E is alternating.*

($\mathcal{D}_{\mathbf{Min}}$.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.*

($\mathcal{D}_{\mathbf{Min}}$.6) *If the outermost operator of E is $\uparrow$ or $\downarrow$, then*

Prop.	E	$X = \mathcal{S}(E)$	E^*
$(\mathcal{D}_{\mathrm{Min}}.1)$	$\langle\updownarrow \langle\updownarrow \alpha_1\rangle\rangle$	$\binom{\alpha_1}{c(\alpha_1)}$	$\langle\updownarrow \alpha_1\rangle$
$(\mathcal{D}_{\mathrm{Min}}.1)$	$\langle\uparrow \alpha_1 \langle\updownarrow \langle\uparrow \alpha_2 \langle\updownarrow \alpha_3\rangle\rangle\rangle\rangle$	$\binom{\alpha_1}{-}\binom{\alpha_2\alpha_3}{c(\alpha_2\alpha_3)}$	$\langle\uparrow \alpha_1 \langle\updownarrow \alpha_2\alpha_3\rangle\rangle$
$(\mathcal{D}_{\mathrm{Min}}.2)$	$\langle\uparrow \alpha_1 \langle\uparrow \langle\updownarrow \alpha_2\rangle \alpha_3\rangle\rangle$	$\binom{\alpha_1}{-}\binom{\alpha_2}{c(\alpha_2)}\binom{\alpha_3}{-}$	$\langle\uparrow \alpha_1 \langle\updownarrow \alpha_2\rangle \alpha_3\rangle$
$(\mathcal{D}_{\mathrm{Min}}.3)$	$\langle\uparrow \langle\updownarrow \alpha_1\rangle\rangle$	$\binom{\alpha_1}{c(\alpha_1)}$	$\langle\updownarrow \alpha_1\rangle$
$(\mathcal{D}_{\mathrm{Min}}.3)$	$\langle\downarrow \alpha_1 \langle\uparrow \langle\updownarrow \alpha_2\rangle\rangle\rangle$	$\binom{-}{\alpha_1}\binom{\alpha_2}{c(\alpha_2)}$	$\langle\downarrow \alpha_1 \langle\updownarrow \alpha_2\rangle\rangle$
$(\mathcal{D}_{\mathrm{Min}}.4)$	$\langle\uparrow \alpha_1 \langle\downarrow \langle\updownarrow \alpha_2\rangle \langle\updownarrow \alpha_3\rangle\rangle\rangle$	$\binom{\alpha_1}{-}\binom{\alpha_2\alpha_3}{c(\alpha_2\alpha_3)}$	$\langle\uparrow \alpha_1 \langle\updownarrow \alpha_2\alpha_3\rangle\rangle$
$(\mathcal{D}_{\mathrm{Min}}.5)$	$\langle\uparrow \alpha_1 \langle\downarrow \langle\uparrow \alpha_2 \langle\updownarrow \alpha_3\rangle\rangle \alpha_4\rangle\rangle$	$\binom{\alpha_1\alpha_2}{-}\binom{\alpha_3}{c(\alpha_3)}\binom{-}{\alpha_4}$	$\langle\uparrow \alpha_1\alpha_2 \langle\downarrow \langle\updownarrow \alpha_3\rangle \alpha_4\rangle\rangle$
$(\mathcal{D}_{\mathrm{Min}}.5)$	$\langle\uparrow \langle\downarrow \alpha_1 \langle\uparrow \langle\updownarrow \alpha_2\rangle \alpha_3\rangle\rangle \alpha_4\rangle$	$\binom{-}{\alpha_1}\binom{\alpha_2}{c(\alpha_2)}\binom{\alpha_3\alpha_4}{-}$	$\langle\uparrow \langle\downarrow \alpha_1 \langle\updownarrow \alpha_2\rangle\rangle \alpha_3\alpha_4\rangle$
$(\mathcal{D}_{\mathrm{Min}}.6)$	$\langle\uparrow \langle\downarrow \alpha_1 \langle\updownarrow \alpha_2\rangle\rangle \alpha_3 \langle\downarrow \langle\updownarrow \alpha_4\rangle \alpha_5\rangle\rangle$	$\binom{-}{\alpha_1}\binom{\alpha_2}{c(\alpha_2)}\binom{\alpha_3}{-}\binom{\alpha_4}{c(\alpha_4)}\binom{-}{\alpha_5}$	$\langle\downarrow \alpha_1 \langle\uparrow \langle\updownarrow \alpha_2\rangle \alpha_3 \langle\updownarrow \alpha_4\rangle\rangle \alpha_5\rangle$

Table 6.1: (See [Van Vliet, 2004, Table 4.3]) Examples of DNA expressions with all six properties from Lemma 6.15 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.

- *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.*

We use $\mathcal{D}_{\mathrm{Min}}$ to denote the set of DNA expressions that have Properties $(\mathcal{D}_{\mathrm{Min}}.1)$–$(\mathcal{D}_{\mathrm{Min}}.6)$.

Theorem 6.16 (See [Van Vliet, 2004, Theorem 4.100], [Van Vliet et al., 2005, Theorem 12]) *A DNA expression E is minimal if and only if $E \in \mathcal{D}_{Min}$.*

Lemma 6.17 (See [Van Vliet, 2004, Lemma 4.99]) *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.*

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*.*

Proof:

3. Assume that E is nick free. If E is an $\updownarrow$-expression, then the claim follows from Property ($\mathcal{D}_{\text{Min}}$.1).

   Now assume that E is an $\uparrow$-expression. By Corollary 6.8(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 $\downarrow$-expression E is analogous.

□

6.3 The structure tree of a minimal DNA expression

As we observed in § 2.8, each DNA expression has a unique representation as an ordered, directed, 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 § 6.2, we have given a characterization of minimal DNA expressions by six properties of the operators occurring in them. These properties can be directly translated into properties characterizing minimal structure trees.

Let t be the structure tree of a DNA expression E.

Theorem 6.16 (and Lemma 6.15) t is minimal if and only if

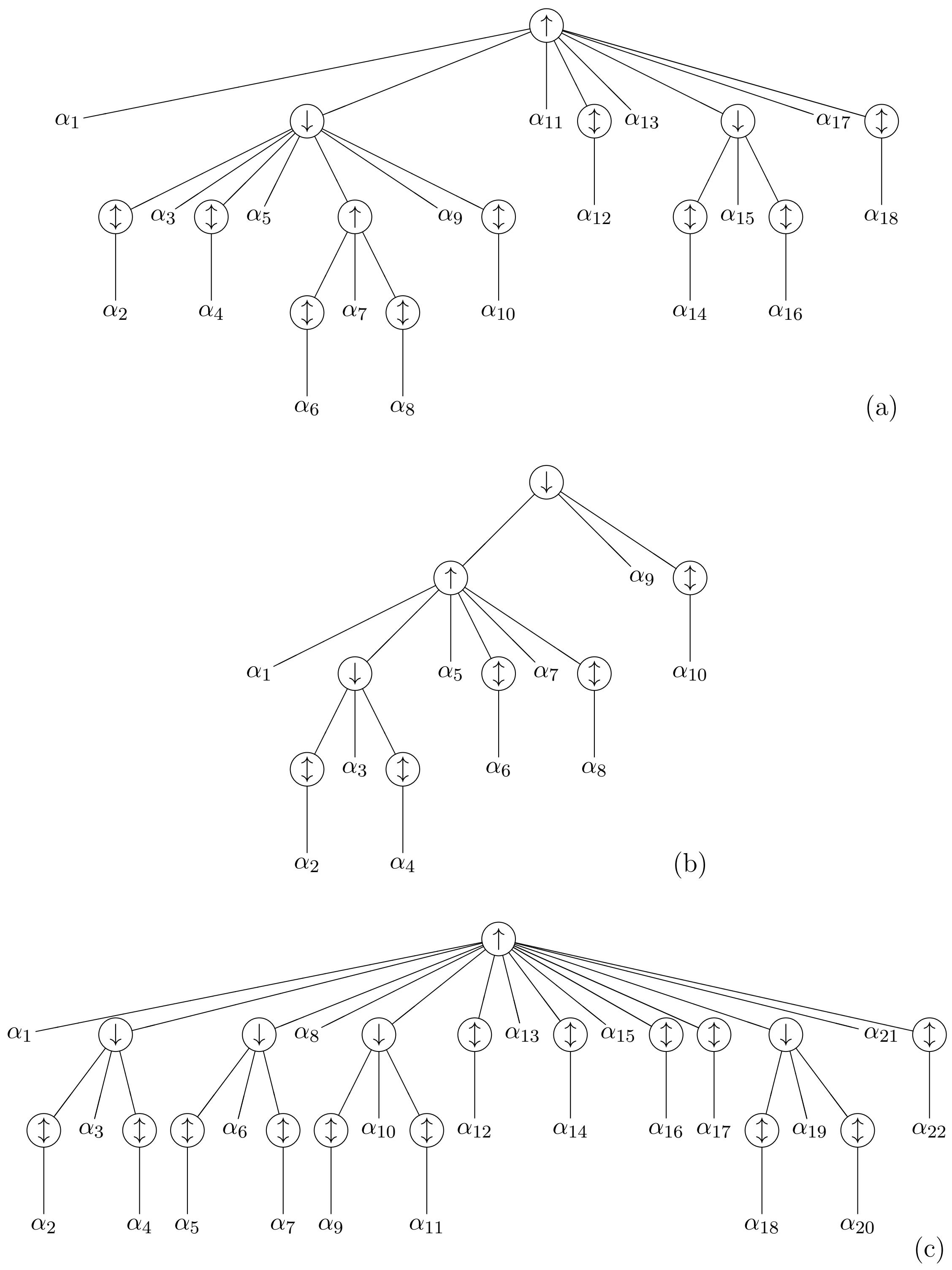
($\mathcal{D}_{\text{Min}}$.1) each node labelled by $\updownarrow$ in t has a (single) child labelled by an $\mathcal{N}$-word α, and

($\mathcal{D}_{\text{Min}}$.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

($\mathcal{D}_{\text{Min}}$.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

($\mathcal{D}_{\text{Min}}$.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

($\mathcal{D}_{\text{Min}}$.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



$\uparrow$
α_1 $\downarrow$ α_{11} $\updownarrow$ α_{13} $\downarrow$ α_{17} $\updownarrow$
$\updownarrow$ α_3 $\updownarrow$ α_5 $\uparrow$ α_9 $\updownarrow$ α_{12} $\updownarrow$ α_{15} $\updownarrow$ α_{18}
α_2 α_4 $\updownarrow$ α_7 $\updownarrow$ α_{10} α_{14} α_{16}
α_6 α_8
(a)
$\downarrow$
$\uparrow$ α_9 $\updownarrow$
α_1 $\downarrow$ α_5 $\updownarrow$ α_7 $\updownarrow$ α_{10}
$\updownarrow$ α_3 $\updownarrow$ α_6 α_8
α_2 α_4
(b)
$\uparrow$
α_1 $\downarrow$ $\downarrow$ α_8 $\downarrow$ $\updownarrow$ α_{13} $\updownarrow$ α_{15} $\updownarrow$ $\updownarrow$ $\downarrow$ α_{21} $\updownarrow$
$\updownarrow$ α_3 $\updownarrow$ $\updownarrow$ α_6 $\updownarrow$ $\updownarrow$ α_{10} $\updownarrow$ α_{12} α_{14} α_{16} α_{17} $\updownarrow$ α_{19} $\updownarrow$ α_{22}
α_2 α_4 α_5 α_7 α_9 α_{11} α_{18} α_{20}
(c)

Figure 6.1: (See [Van Vliet, 2004, Figure 4.10(b)-(c)]) Three minimal structure trees. (a) The structure tree of the minimal DNA expression from Equation (5.6), denoting the nick free formal DNA molecule from (a.o.) Figure 5.3. (b) The structure tree of the minimal DNA expression E_d from Equation (5.11), denoting the nick free formal DNA molecule from Figure 5.4. (c) The structure tree of the minimal DNA expression from Equation (5.19), denoting the formal DNA molecule from Figure 5.5, which contains four lower nick letters.

($\mathcal{D}_{\mathbf{Min}}$.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.

In Figure 6.1, we have drawn the structure trees of three minimal DNA expressions we have constructed in the course of Chapter 5. One can verify that these structure trees exhibit all properties we have just listed.

6.4 The number of (operator-)minimal DNA expressions

Definition 6.18 (See [Van Vliet, 2004, Definition 4.82]) *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.*

Corollary 6.19 (See [Van Vliet, 2004, Corollary 4.96], [Van Vliet et al., 2005, Theorem 11]) *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*

$$\begin{aligned}
n_{min\uparrow}(X) = n_{opermin\uparrow}(X) &= \frac{1}{p_2+1}\binom{2p_2}{p_2}, \\
n_{min\downarrow}(X) &= 0, \\
n_{opermin\downarrow}(X) &= \frac{1}{p_1+1}\binom{2p_1}{p_1}, \\
n_{min\updownarrow}(X) = n_{opermin\updownarrow}(X) &= 0 \qquad \textit{and} \\
n_{min}(X) &= \frac{1}{p_2+1}\binom{2p_2}{p_2}.
\end{aligned}$$

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*

$$\begin{aligned}
n_{min\uparrow}(X) &= 0, \\
n_{opermin\uparrow}(X) &= \frac{1}{p_2+1}\binom{2p_2}{p_2}, \\
n_{min\downarrow}(X) = n_{opermin\downarrow}(X) &= \frac{1}{p_1+1}\binom{2p_1}{p_1}, \\
n_{min\updownarrow}(X) = n_{opermin\updownarrow}(X) &= 0 \qquad \textit{and} \\
n_{min}(X) &= \frac{1}{p_1+1}\binom{2p_1}{p_1}.
\end{aligned}$$

5. *If X contains at least one lower nick letter, then let $Z_{1\vartriangle}Z_{2\vartriangle}\ldots{}_\vartriangle Z_m$ for some $m \geq 2$ be the nick free decomposition of X, and let for $h = 1, \ldots, m$, $p_h = B_\downarrow(Z_h)$.*

$$\begin{aligned}
n_{min\uparrow}(X) = n_{opermin\uparrow}(X) &= \frac{1}{p_1+1}\binom{2p_1}{p_1} \times \cdots \times \frac{1}{p_m+1}\binom{2p_m}{p_m}, \\
n_{min\downarrow}(X) = n_{opermin\downarrow}(X) &= 0, \\
n_{min\updownarrow}(X) &= 0 \qquad \textit{and} \\
n_{min}(X) &= \frac{1}{p_1+1}\binom{2p_1}{p_1} \times \cdots \times \frac{1}{p_m+1}\binom{2p_m}{p_m}.
\end{aligned}$$

   (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 $h = 1, \ldots, m$, $p_h = B_\uparrow(Z_h)$.*

$$\begin{aligned}
n_{min\uparrow}(X) = n_{opermin\uparrow}(X) &= 0, \\
n_{min\downarrow}(X) = n_{opermin\downarrow}(X) &= \frac{1}{p_1+1}\binom{2p_1}{p_1} \times \cdots \times \frac{1}{p_m+1}\binom{2p_m}{p_m}, \\
n_{min\updownarrow}(X) &= 0 \qquad \textit{and} \\
n_{min}(X) &= \frac{1}{p_1+1}\binom{2p_1}{p_1} \times \cdots \times \frac{1}{p_m+1}\binom{2p_m}{p_m}.
\end{aligned}$$

(a) If X does not contain any single-stranded component, then

$$n_{opermin\updownarrow}(X) \quad = \quad |Z_1| \times |Z_m|.$$

(b) If X contains at least one single-stranded component, then

$$n_{opermin\updownarrow}(X) \quad = \quad 0.$$

Chapter 7

An Algorithm for Minimality

In Chapter 5, we have described how to construct a minimal DNA expression denoting a given formal DNA molecule. In Chapter 6, we have given a characterization of minimal DNA expressions (Theorem 6.16).

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 5.

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 6.15.

In § 7.1, we describe the algorithm and prove that it is correct. We illustrate the different steps in the algorithm by example DNA expressions, which are DNA subexpressions of a single, large DNA expression. In § 7.2, we systematically work out the algorithm for this DNA expression as a whole.

The description of the algorithm in § 7.1 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 details and analyse the complexity of the algorithm in § 7.3. Finally, in § 7.4, we relate the time spent by the algorithm on actual rewriting steps to the resulting decrease of $|E|$.

7.1 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 6.15. We use that in the second phase, where we construct a DNA expression E' that is equivalent to E and has these properties itself. By Theorem 6.16, 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, wheras 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 7.1, 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

> substitute E by a minimal DNA expression E' satisfying $E' \equiv E$; (procedure ...)

These instructions will be worked out in detail later, 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 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 $\updownarrow$-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 $\updownarrow$-expression)
7.               then substitute $E$ by $E_1$;                    ($\mathcal{D}_{\mathtt{Min}}$.1)
8.               else   // $E_1$ is an $\uparrow$-expression or a $\downarrow$-expression
9.                    substitute $E$ by a minimal DNA expression $E'$
                        satisfying $E' \equiv E$; (procedure Make$\updownarrow$ExprMinimal) ($\mathcal{D}_{\mathtt{Min}}$.1)
10.              fi
11.         fi

12.    else    // $E$ is an $\uparrow$-expression or a $\downarrow$-expression;
               // without loss of generality, assume it is
               // an $\uparrow$-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 $\downarrow$-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 $E_i' \equiv_{\triangledown} E_i$;
                        (procedure Denickify)                    ($\mathcal{D}_{\mathtt{Min}}$.4)
19.              fi
20.         od
              // we proceed with the new expression-arguments
21.         for all $\downarrow$-arguments $E_i$ of $E$ (in some order)
22.         do   if (the first argument or the last argument of $E_i$
                     is an $\uparrow$-argument)
23.              then substitute $E_i$ in $E$ by a minimal $\uparrow$-expression $E_i'$
                        satisfying $E_i' \equiv E_i$; (procedure RotateToMinimal) ($\mathcal{D}_{\mathtt{Min}}$.5)
24.              fi
25.         od
              // we proceed with the new expression-arguments
26.         for all $\uparrow$-arguments $E_i = \langle\uparrow \varepsilon_{i,1} \dots \varepsilon_{i,n_i}\rangle$ of $E$ (in some order)
27.         do   substitute $E_i$ in $E$ by its arguments $\varepsilon_{i,1} \dots \varepsilon_{i,n_i}$; ($\mathcal{D}_{\mathtt{Min}}$.2)
28.         od

29.         if ($E$ has only one argument $\varepsilon_1$)
30.         then if ($\varepsilon_1$ is a DNA expression $E_1$)
31.              then substitute $E$ by $E_1$;                    ($\mathcal{D}_{\mathtt{Min}}$.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 $\downarrow$-arguments)
35.              then substitute $E$ by a minimal $\downarrow$-expression $E'$
                        satisfying $E' \equiv E$; (procedure RotateToMinimal) ($\mathcal{D}_{\mathtt{Min}}$.6)
36.              fi
37.         fi
38.    fi
39.  }
```

Figure 7.1: 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 6.8, 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 6.15. 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.

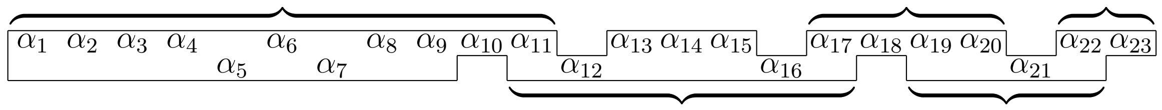
Assume that the DNA expression E is an $\updownarrow$-expression. Then `MakeMinimal` only rewrites E, if its argument is a DNA expression E_1 (in lines 5–11), i.e., not if it is an $\mathcal{N}$-word α_1. This is justified by Theorem 5.3: an $\updownarrow$-expression E with an expression-argument is 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 7.12, 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 6.8(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-



α_1 α_2 α_3 α_4 α_5 α_6 α_7 α_8 α_9 α_{10} α_{11}
α_{12}
α_{13} α_{14} α_{15}
α_{16}
α_{17} α_{18} α_{19} α_{20}
α_{21}
α_{22} α_{23}

Figure 7.2: The formal DNA molecule X denoted by the DNA expression E_1^* in (7.1), 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 7.17, 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:

$$\begin{aligned} E_1^* = \langle\downarrow \langle\downarrow\ &\langle\uparrow \langle\updownarrow \langle\downarrow\ \langle\updownarrow \langle\uparrow \alpha_1 \langle\updownarrow \langle\updownarrow \alpha_2\rangle\rangle\, \alpha_3 \langle\downarrow \langle\updownarrow \alpha_4\rangle\, \alpha_5\rangle\rangle\rangle\ \langle\updownarrow \alpha_6\rangle\ \alpha_7\ \rangle\rangle \\ &\langle\downarrow \langle\updownarrow \alpha_8\rangle \langle\uparrow \langle\updownarrow \alpha_9\rangle\, \alpha_{10} \langle\updownarrow \alpha_{11}\rangle\rangle\, \alpha_{12} \langle\updownarrow \alpha_{13}\rangle\rangle\ \langle\updownarrow \alpha_{14}\rangle \\ &\langle\downarrow \langle\updownarrow \alpha_{15}\rangle\, \alpha_{16} \langle\uparrow \langle\updownarrow \alpha_{17}\rangle\, \alpha_{18}\rangle\rangle\ \langle\uparrow \langle\updownarrow \alpha_{19}\rangle \langle\updownarrow \alpha_{20}\rangle\rangle\ \rangle\ \rangle \\ \langle\uparrow \langle\downarrow \alpha_{21}\rangle\rangle\ &\langle\uparrow \langle\updownarrow \alpha_{22}\rangle\, \alpha_{23}\rangle\ \rangle, \end{aligned} \tag{7.1}$$

where $\alpha_1, \ldots, \alpha_{23}$ are arbitrary $\mathcal{N}$-words. It denotes the formal DNA molecule X depicted in Figure 7.2. 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.

It requires a number of steps to rewrite the DNA expression E_1^* from (7.1) 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 7.1. 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 the procedures that are mentioned there, yet. 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 (semantical) 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 7.1 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 E_1 of E is the minimal $\updownarrow$-expression $\langle\updownarrow \alpha_2\rangle$. 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 $E_1 \equiv E$. ■

In the proof of Theorem 7.17, we will see that, apart from the particular $\mathcal{N}$-word α_2, this example is the only possibility in line 7.

Example 7.2 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

$$\mathcal{S}(E) = \binom{\alpha_1\alpha_2\alpha_3\alpha_4 c(\alpha_5)}{c(\alpha_1\alpha_2\alpha_3\alpha_4)\alpha_5}.$$

The argument of E is a minimal $\uparrow$-expression E_1. Hence, E violates Property ($\mathcal{D}_{\text{Min}}$.1). In line 9 of `MakeMinimal`, we substitute E by a minimal DNA expression E' that satisfies $E' \equiv E$. By Theorem 5.3, there exists exactly one such DNA expression: $E' = \langle \updownarrow \alpha_1\alpha_2\alpha_3\alpha_4 c(\alpha_5)\rangle$. ■

The minimal, nick free DNA expression E'_i that we substitute for the non-alternating $\downarrow$-argument E_i in line 18, may be again a $\downarrow$-expression, but it may also be an $\uparrow$-expression or an $\updownarrow$-expression. We consider two examples covering these three possibilities.

Example 7.3 Let

$$\begin{aligned} E = \langle \uparrow\ & \langle \downarrow \langle \updownarrow \alpha_1\alpha_2\alpha_3\alpha_4 c(\alpha_5)\rangle \langle \updownarrow \alpha_6 c(\alpha_7)\rangle\rangle \\ & \langle \downarrow \langle \updownarrow \alpha_8 \rangle \langle \uparrow \langle \updownarrow \alpha_9 \rangle\, \alpha_{10} \langle \updownarrow \alpha_{11} \rangle\rangle\, \alpha_{12} \langle \updownarrow \alpha_{13} \rangle\rangle\ \ \langle \updownarrow \alpha_{14} \rangle \\ & \langle \downarrow \langle \updownarrow \alpha_{15} \rangle\, \alpha_{16} \langle \uparrow \langle \updownarrow \alpha_{17} \rangle\, \alpha_{18} \rangle\rangle\ \ \langle \uparrow \langle \updownarrow \alpha_{19} \rangle \langle \updownarrow \alpha_{20} \rangle\rangle\ \rangle\, , \end{aligned}$$

for which

$$\begin{aligned} X = \mathcal{S}(E) \quad = \quad & \binom{\alpha_1\alpha_2\alpha_3\alpha_4 c(\alpha_5)\alpha_6 c(\alpha_7)}{c(\alpha_1\alpha_2\alpha_3\alpha_4)\alpha_5 c(\alpha_6)\alpha_7} {\scriptstyle\triangle} \binom{\alpha_8\alpha_9}{c(\alpha_8\alpha_9)} \binom{\alpha_{10}}{-} \binom{\alpha_{11}}{c(\alpha_{11})} \binom{-}{\alpha_{12}} \cdot \\ & \binom{\alpha_{13}}{c(\alpha_{13})} {\scriptstyle\triangle} \binom{\alpha_{14}}{c(\alpha_{14})} {\scriptstyle\triangle} \binom{\alpha_{15}}{c(\alpha_{15})} \binom{-}{\alpha_{16}} \binom{\alpha_{17}}{c(\alpha_{17})} \binom{\alpha_{18}}{-} \binom{\alpha_{19}}{c(\alpha_{19})} {\scriptstyle\triangle} \binom{\alpha_{20}}{c(\alpha_{20})}. \end{aligned}$$

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

$$\mathcal{S}(E_1) = \binom{\alpha_1\alpha_2\alpha_3\alpha_4 c(\alpha_5)}{c(\alpha_1\alpha_2\alpha_3\alpha_4)\alpha_5} {\scriptstyle\triangledown} \binom{\alpha_6 c(\alpha_7)}{c(\alpha_6)\alpha_7}.$$

Hence, E_1 makes E violate Property ($\mathcal{D}_{\text{Min}}$.4). E_1 is not nick free. If E'_1 is a nick free DNA expression satisfying $E'_1 \equiv_{\triangledown} E_1$, then

$$\mathcal{S}(E'_1) = \binom{\alpha_1\alpha_2\alpha_3\alpha_4 c(\alpha_5)\alpha_6 c(\alpha_7)}{c(\alpha_1\alpha_2\alpha_3\alpha_4)\alpha_5 c(\alpha_6)\alpha_7}.$$

By Theorem 5.3, 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 7.4 Let

$$\begin{aligned} E = \langle \uparrow\ & \langle \updownarrow \alpha_1\alpha_2\alpha_3\alpha_4 c(\alpha_5)\alpha_6 c(\alpha_7)\rangle \\ & \langle \downarrow \langle \updownarrow \alpha_8 \rangle \langle \uparrow \langle \updownarrow \alpha_9 \rangle\, \alpha_{10} \langle \updownarrow \alpha_{11} \rangle\rangle\, \alpha_{12} \langle \updownarrow \alpha_{13} \rangle\rangle\ \ \langle \updownarrow \alpha_{14} \rangle \\ & \langle \downarrow \langle \updownarrow \alpha_{15} \rangle\, \alpha_{16} \langle \uparrow \langle \updownarrow \alpha_{17} \rangle\, \alpha_{18} \rangle\rangle\ \ \langle \uparrow \langle \updownarrow \alpha_{19} \rangle \langle \updownarrow \alpha_{20} \rangle\rangle\ \rangle\, . \end{aligned}$$

This is the result when we substitute the first $\downarrow$-argument E_1 of the $\uparrow$-expression E from Example 7.3 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

$$\mathcal{S}(E_2) = \binom{\alpha_8}{c(\alpha_8)}^{\triangledown}\binom{\alpha_9}{c(\alpha_9)}\binom{\alpha_{10}}{-}\binom{\alpha_{11}}{c(\alpha_{11})}\binom{-}{\alpha_{12}}\binom{\alpha_{13}}{c(\alpha_{13})}.$$

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

$$X'_2 = \binom{\alpha_8\alpha_9}{c(\alpha_8\alpha_9)}\binom{\alpha_{10}}{-}\binom{\alpha_{11}}{c(\alpha_{11})}\binom{-}{\alpha_{12}}\binom{\alpha_{13}}{c(\alpha_{13})}.$$

We have $B_{\uparrow}(X'_2) = B_{\downarrow}(X'_2) = 1$. By Summary 6.12(2) and the recursive construction from Theorem 5.12, there are two diffent 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 7.5 Let

$$\begin{aligned}E = \langle\uparrow\ &\langle\updownarrow \alpha_1\alpha_2\alpha_3\alpha_4 c(\alpha_5)\alpha_6 c(\alpha_7)\rangle\\ &\langle\downarrow \langle\uparrow \langle\updownarrow \alpha_8\alpha_9\rangle \alpha_{10} \langle\updownarrow \alpha_{11}\rangle\rangle \alpha_{12} \langle\updownarrow \alpha_{13}\rangle\rangle\ \langle\updownarrow \alpha_{14}\rangle\\ &\langle\downarrow \langle\updownarrow \alpha_{15}\rangle \alpha_{16} \langle\uparrow \langle\updownarrow \alpha_{17}\rangle \alpha_{18}\rangle\rangle\ \langle\uparrow \langle\updownarrow \alpha_{19}\rangle \langle\updownarrow \alpha_{20}\rangle\rangle\ \rangle\,.\end{aligned}$$

This is the result when we substitute the second argument E_2 of the $\uparrow$-expression E from Example 7.4 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 7.4, 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 7.6 Let

$$E = \langle \uparrow \langle \updownarrow \alpha_1\alpha_2\alpha_3\alpha_4 c(\alpha_5)\alpha_6 c(\alpha_7)\rangle \\ \langle \uparrow \langle \updownarrow \alpha_8\alpha_9\rangle \, \alpha_{10} \, \langle \downarrow \langle \updownarrow \alpha_{11}\rangle \, \alpha_{12} \, \langle \updownarrow \alpha_{13}\rangle\rangle\rangle \;\; \langle \updownarrow \alpha_{14}\rangle \\ \langle \downarrow \langle \updownarrow \alpha_{15}\rangle \, \alpha_{16} \, \langle \uparrow \langle \updownarrow \alpha_{17}\rangle \, \alpha_{18}\rangle\rangle \;\; \langle \uparrow \langle \updownarrow \alpha_{19}\rangle \langle \updownarrow \alpha_{20}\rangle\rangle \; \rangle \, .$$

This is the result when we substitute the second argument E_2 of the $\uparrow$-expression E from Example 7.5 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

$$X_4 = \mathcal{S}(E_4) = \binom{\alpha_{15}}{c(\alpha_{15})}\binom{-}{\alpha_{16}}\binom{\alpha_{17}}{c(\alpha_{17})}\binom{\alpha_{18}}{-}.$$

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 6.12(2) and the recursive construction from Theorem 5.12, 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 7.5 and in Example 7.6, 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 7.7 *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 6.12, 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 5.12.

Now, when we apply Corollary 5.19(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 7.8 Let

$$\begin{aligned} E = \langle\uparrow\ &\langle\updownarrow\ \alpha_1\alpha_2\alpha_3\alpha_4 c(\alpha_5)\alpha_6 c(\alpha_7)\rangle \\ &\langle\uparrow\ \langle\updownarrow\ \alpha_8\alpha_9\rangle\ \alpha_{10}\ \langle\downarrow\ \langle\updownarrow\ \alpha_{11}\rangle\ \alpha_{12}\ \langle\updownarrow\ \alpha_{13}\rangle\rangle\rangle\ \ \langle\updownarrow\ \alpha_{14}\rangle \\ &\langle\uparrow\ \langle\downarrow\ \langle\updownarrow\ \alpha_{15}\rangle\ \alpha_{16}\ \langle\updownarrow\ \alpha_{17}\rangle\rangle\ \alpha_{18}\rangle\ \ \langle\uparrow\ \langle\updownarrow\ \alpha_{19}\rangle\ \langle\updownarrow\ \alpha_{20}\rangle\rangle\ \rangle\,. \end{aligned}$$

This is the result when we substitute the fourth argument E_4 of the $\uparrow$-expression E from Example 7.6 by the corresponding $\uparrow$-expression E_4'. The $\uparrow$-expression E has three $\uparrow$-arguments. Hence, it violates Property ($\mathcal{D}_{\rm Min}$.2). In lines 26–28, we substitute these three $\uparrow$-arguments by their respective arguments. The result is

$$\begin{aligned} E = \langle\uparrow\ &\langle\updownarrow\ \alpha_1\alpha_2\alpha_3\alpha_4 c(\alpha_5)\alpha_6 c(\alpha_7)\rangle \\ &\langle\updownarrow\ \alpha_8\alpha_9\rangle\ \ \alpha_{10}\ \ \langle\downarrow\ \langle\updownarrow\ \alpha_{11}\rangle\ \alpha_{12}\ \langle\updownarrow\ \alpha_{13}\rangle\rangle\ \ \langle\updownarrow\ \alpha_{14}\rangle \\ &\langle\downarrow\ \langle\updownarrow\ \alpha_{15}\rangle\ \alpha_{16}\ \langle\updownarrow\ \alpha_{17}\rangle\rangle\ \ \alpha_{18}\ \ \langle\updownarrow\ \alpha_{19}\rangle\ \ \langle\updownarrow\ \alpha_{20}\rangle\ \rangle\,. \end{aligned}$$

■

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 7.9 Let

$$\begin{aligned} E = \langle\uparrow\ &\langle\updownarrow\ \alpha_1\alpha_2\alpha_3\alpha_4 c(\alpha_5)\alpha_6 c(\alpha_7)\rangle \\ &\langle\updownarrow\ \alpha_8\alpha_9\rangle\ \ \alpha_{10}\ \ \langle\downarrow\ \langle\updownarrow\ \alpha_{11}\rangle\ \alpha_{12}\ \langle\updownarrow\ \alpha_{13}\rangle\rangle\ \ \langle\updownarrow\ \alpha_{14}\rangle \\ &\langle\downarrow\ \langle\updownarrow\ \alpha_{15}\rangle\ \alpha_{16}\ \langle\updownarrow\ \alpha_{17}\rangle\rangle\ \ \alpha_{18}\ \ \langle\updownarrow\ \alpha_{19}\rangle\ \ \langle\updownarrow\ \alpha_{20}\rangle\ \rangle\,. \end{aligned}$$

This is the result of Example 7.8. 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 7.3. Hence, it is indeed equivalent to the original DNA expression. Moreover, it is easily verified that E has all six properties from Lemma 6.15 and thus is minimal. ■

Example 7.10 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 ($\mathcal{D}_{\rm Min}$.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 6.12(4) and the construction from Theorem 5.12(2), this is the only minimal DNA expression denoting $\mathcal{S}(E)$. ■

Example 7.11 Let

$$\begin{aligned} E = \langle\downarrow\ &\langle\uparrow\ \langle\updownarrow\ \alpha_1\alpha_2\alpha_3\alpha_4 c(\alpha_5)\alpha_6 c(\alpha_7)\alpha_8\alpha_9\rangle\ \ \alpha_{10} \\ &\qquad \langle\downarrow\ \langle\updownarrow\ \alpha_{11}\rangle\ \alpha_{12}\ \langle\updownarrow\ \alpha_{13}\alpha_{14}\alpha_{15}\rangle\ \alpha_{16}\ \langle\updownarrow\ \alpha_{17}\rangle\rangle\ \ \alpha_{18}\ \ \langle\updownarrow\ \alpha_{19}\alpha_{20}\rangle\ \rangle \\ &\alpha_{21}\ \ \langle\uparrow\ \langle\updownarrow\ \alpha_{22}\rangle\ \alpha_{23}\rangle\ \rangle\,, \end{aligned}$$

which, like DNA expression E_1^* from (7.1), denotes the formal DNA molecule X from Figure 7.2. The $\downarrow$-expression E has three arguments: two minimal, alternating $\uparrow$-arguments, separated by an $\mathcal{N}$-word α_{21}. Because E itself is also alternating, it violates Property ($\mathcal{D}_{\rm Min}$.6) and line 35 of the function `MakeMinimal` is applicable.

The formal DNA molecule X is nick free. As indicated in Figure 7.2, $B_\uparrow(X) = 3$ and $B_\downarrow(X) = 2$. Hence, by Summary 6.12(3) and the recursive construction from

Theorem 5.12, there are two different minimal DNA expressions E' denoting X, i.e., with $E' \equiv E$:

$$\begin{aligned} E' = \langle\uparrow\ &\langle\updownarrow\ \alpha_1\alpha_2\alpha_3\alpha_4 c(\alpha_5)\alpha_6 c(\alpha_7)\alpha_8\alpha_9\rangle\ \alpha_{10} \\ &\langle\downarrow\ \langle\updownarrow\ \alpha_{11}\rangle\ \alpha_{12}\ \langle\updownarrow\ \alpha_{13}\alpha_{14}\alpha_{15}\rangle\ \alpha_{16}\ \langle\updownarrow\ \alpha_{17}\rangle\rangle\ \alpha_{18} \\ &\langle\downarrow\ \langle\updownarrow\ \alpha_{19}\alpha_{20}\rangle\ \alpha_{21}\ \langle\updownarrow\ \alpha_{22}\rangle\rangle\ \alpha_{23}\ \rangle \end{aligned}$$

and

$$\begin{aligned} E' = \langle\uparrow\ &\langle\updownarrow\ \alpha_1\alpha_2\alpha_3\alpha_4 c(\alpha_5)\alpha_6 c(\alpha_7)\alpha_8\alpha_9\rangle\ \alpha_{10} \\ &\langle\downarrow\ \langle\updownarrow\ \alpha_{11}\rangle\ \alpha_{12}\ \langle\updownarrow\ \alpha_{13}\alpha_{14}\alpha_{15}\rangle\ \alpha_{16}\ \ \langle\uparrow\ \langle\updownarrow\ \alpha_{17}\rangle\ \alpha_{18}\ \langle\updownarrow\ \alpha_{19}\alpha_{20}\rangle\rangle\ \ \alpha_{21}\ \langle\updownarrow\ \alpha_{22}\rangle\ \rangle \\ &\alpha_{23}\ \rangle\,. \end{aligned}$$

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 71, we already observed that the function `MakeMinimal` does nothing to a minimal $\updownarrow$-expression. We now consider arbitrary minimal DNA expressions.

Theorem 7.12 *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.

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 5.3, 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 have no effect on E.

  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` has no effect on E, 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 has no effect on E, 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 has no effect on E, 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 § 7.4. Note that for many formal DNA molecules, there exists more than one minimal DNA expression, see, e.g., Corollary 6.19). 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 7.13 *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.*

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. We prove that under certain conditions, the desired equivalent, minimal $\uparrow$-expression or $\downarrow$-expression does exist.

Lemma 7.14 *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 3.5, 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 4.6(3b) and (3d), $B_{\downarrow}(X) \geq B_{\uparrow}(X)$. Hence, by Theorem 5.12(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 ($\mathcal{D}_{\mathrm{Min}}$.4), then each occurrence of $\uparrow$ or $\downarrow$ in E is alternating. By Lemma 3.5, E is nick free. But then we also have

Corollary 7.15 *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 7.14 and Corollary 7.15 is not necessarily operator-minimal. Hence, Corollary 6.8 is not applicable: the adjectives 'nick free' and 'alternating' are not equivalent, here. There exist DNA expressions E for which Lemma 7.14 is applicable, but Corollary 7.15 is not, because they are nick free but not alternating.

Example 7.16 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 ($\mathcal{D}_{\mathrm{Min}}$.3)–($\mathcal{D}_{\mathrm{Min}}$.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 6.12(2) and the construction from Theorem 5.12 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 7.17 *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 E_2^* is independent of choices made in the procedures* `Make↕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 3.7, 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 6.12 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 3.3, 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 7.12, `MakeMinimal` does not perform any rewriting step on E^*_1. The only thing the function does, is checking some conditions. In particular, we do not have recursive calls of `MakeMinimal` and 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 3.7, 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 5.3, $E_1 = \langle \updownarrow \alpha_1 \rangle$ 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 $E' \equiv E \equiv E_1^*$.

– 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. We prove that the following property is an invariant for this loop:

$$E \text{ is an } \uparrow\text{-expression satisfying } E \equiv E_1^*. \tag{7.2}$$

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 (7.2) 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 3.7, 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. We prove that the following property is an invariant for this loop:

$$E \text{ is an } \uparrow\text{-expression satisfying } E \equiv E_1^*, \text{ and each expression-argument of } E \text{ is minimal.} \tag{7.3}$$

By Lemma 6.15, this property implies that the expression-arguments of E have Properties ($\mathcal{D}_{\text{Min}}$.1)–($\mathcal{D}_{\text{Min}}$.6). Because each occurrrence 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 6.1, 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 (7.3) 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,

$$X = \nu^+(X_1)y_1\nu^+(X_2)y_2 \dots y_{n-1}\nu^+(X_n), \tag{7.4}$$

where for $i = 1, \dots, n-1$, $y_i = \vartriangle$ 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 (7.3) 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 6.8, $X_i = \mathcal{S}(E_i)$ 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 3.2(1), X_i does not contain lower nick letters.

By Lemma 3.7, 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 2.10, $L(X_i') = L(X_i)$ and $R(X_i') = R(X_i)$, the semantics of E is the same before and after the substitution (see (7.4)). 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 (7.3) 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 6.8, 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 (7.3), we obtain

E is an $\uparrow$-expression satisfying $E \equiv E_1^*$, each expression-argument of E is minimal and each $\downarrow$-argument of E is alternating. (7.5)

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 (7.5), 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 (7.5) is an invariant for the third for-loop.

- Clearly, before the first iteration of the for-loop, the property is valid.
- Suppose that Property (7.5) is valid before a certain iteration of the for-loop. In the iteration, we consider a $\downarrow$-argument E_i of E.

  By Property (7.5), 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 (7.5) 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 7.15, 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 3.7, 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 (7.5) 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:

> E is an $\uparrow$-expression satisfying $E \equiv E_1^*$, each expression-argument of E is minimal, each $\downarrow$-argument of E is alternating, and for each $\downarrow$-argument of 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 (7.6)
> - the last argument is either an $\mathcal{N}$-word α or an $\updownarrow$-expression $\langle \updownarrow \alpha \rangle$ for an $\mathcal{N}$-word α.

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 (7.6).

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 7.7, 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 (7.6) is an invariant for the fourth for-loop.

- Clearly, before the first iteration of the for-loop, the property is valid.
- Suppose that Property (7.6) 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 3.6, 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 ($\mathcal{D}_{\mathrm{Min}}$.4), its $\downarrow$-argument E'_j is alternating. Moreover, by Property ($\mathcal{D}_{\mathrm{Min}}$.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 (7.6). 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 ($\mathcal{D}_{\mathrm{Min}}$.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, anymore.

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 ($\mathcal{D}_{\mathrm{Min}}$.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 ($\mathcal{D}_{\mathrm{Min}}$.2).

Earlier in the proof, we deduced from Property (7.3) that E has Property ($\mathcal{D}_{\mathrm{Min}}$.1). As Property (7.3) is still valid, E still has Property ($\mathcal{D}_{\mathrm{Min}}$.1).

Later, we deduced from Property (7.5) that each inner occurrence of $\downarrow$ in E is alternating. Even later, we deduced from Property (7.6) 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 (7.5) and Property (7.6) 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 anymore, 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 ($\mathcal{D}_{\mathrm{Min}}$.4), $\uparrow_1$ is alternating. By Property ($\mathcal{D}_{\mathrm{Min}}$.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 ($\mathcal{D}_{\mathrm{Min}}$.4) and Property ($\mathcal{D}_{\mathrm{Min}}$.5).

Note that by Property ($\mathcal{D}_{\mathrm{Min}}$.1), Property ($\mathcal{D}_{\mathrm{Min}}$.2) and Property (7.5), 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 (7.3), $E_2^* = E_1$ 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 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 7.15, 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 6.15, and thus is minimal. Moreover, $E_2^* = E \equiv E_1^*$.

3. Assume that E_1^* is an $\updownarrow$-expression, and let $X = \mathcal{S}(E_1^*)$. By Corollary 3.4, there exist $\mathcal{N}$-words $\alpha_1, \dots, \alpha_m$ for some $m \geq 1$ and a nick letter $y \in \{{}^{\triangledown}, {}_{\vartriangle}\}$, such that

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

Without loss of generality, assume that $y = {}_{\vartriangle}$.

If $m = 1$, i.e., if $X = \binom{\alpha_1}{c(\alpha_1)}$, then by Theorem 5.3, there is exactly one minimal DNA expression denoting X. If $m \geq 2$, then Lemma 6.14(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`.

```
M$\updownarrow$M.1.   Make$\updownarrow$ExprMinimal ($E$)
                        // rewrites an $\updownarrow$-expression $E = \langle \updownarrow E_1 \rangle$ whose argument $E_1$
                        // is a minimal $\uparrow$-expression, into a minimal DNA expression $E'$
                        // satisfying $E' \equiv E$
M$\updownarrow$M.2.   {
M$\updownarrow$M.3.       $\widehat{E}_1 = E_1$;
M$\updownarrow$M.4.       for all $\downarrow$-arguments $E_{1,i}$ of $\widehat{E}_1$ (in some order)
M$\updownarrow$M.5.       do  substitute $E_{1,i}$ in $\widehat{E}_1$ by $\langle \updownarrow \alpha_{E_{1,i}} \rangle$;
M$\updownarrow$M.6.       od
                          // arguments of $\widehat{E}_1$ are $\mathcal{N}$-words $\alpha_{1,i}$ and $\updownarrow$-expressions $\langle \updownarrow \alpha_{1,i} \rangle$
M$\updownarrow$M.7.       for all $\mathcal{N}$-word-arguments $\alpha_{1,i}$ of $\widehat{E}_1$ (in some order)
M$\updownarrow$M.8.       do  if ($\alpha_{1,i}$ is preceded by an argument $\langle \updownarrow \alpha_{1,i-1} \rangle$)
M$\updownarrow$M.9.           then if ($\alpha_{1,i}$ is succeeded by an argument $\langle \updownarrow \alpha_{1,i+1} \rangle$)
M$\updownarrow$M.10.               then substitute $\langle \updownarrow \alpha_{1,i-1} \rangle \, \alpha_{1,i} \, \langle \updownarrow \alpha_{1,i+1} \rangle$ in $\widehat{E}_1$
                                           by $\langle \updownarrow \alpha_{1,i-1}\alpha_{1,i}\alpha_{1,i+1} \rangle$;
M$\updownarrow$M.11.               else substitute $\langle \updownarrow \alpha_{1,i-1} \rangle \, \alpha_{1,i}$ in $\widehat{E}_1$ by $\langle \updownarrow \alpha_{1,i-1}\alpha_{1,i} \rangle$;
M$\updownarrow$M.12.               fi
M$\updownarrow$M.13.           else if ($\alpha_{1,i}$ is succeeded by an argument $\langle \updownarrow \alpha_{1,i+1} \rangle$)
M$\updownarrow$M.14.               then substitute $\alpha_{1,i} \, \langle \updownarrow \alpha_{1,i+1} \rangle$ in $\widehat{E}_1$ by $\langle \updownarrow \alpha_{1,i}\alpha_{1,i+1} \rangle$;
M$\updownarrow$M.15.               else substitute $\alpha_{1,i}$ in $\widehat{E}_1$ by $\langle \updownarrow \alpha_{1,i} \rangle$;
M$\updownarrow$M.16.               fi
M$\updownarrow$M.17.           fi
M$\updownarrow$M.18.      od
                          // $\widehat{E}_1 = \langle \uparrow \langle \updownarrow \alpha_{1,1} \rangle \dots \langle \updownarrow \alpha_{1,m} \rangle \rangle$ for $m \geq 1$
                          // and $\mathcal{N}$-words $\alpha_{1,1}, \dots, \alpha_{1,m}$
M$\updownarrow$M.19.      if ($m == 1$)
M$\updownarrow$M.20.      then substitute $\widehat{E}_1$ by $\langle \updownarrow \alpha_{1,1} \rangle$;                    ($\mathcal{D}_{\texttt{Min}}$.3)
M$\updownarrow$M.21.      fi
M$\updownarrow$M.22.      $E' = \widehat{E}_1$;
M$\updownarrow$M.23.  }
```

Figure 7.3: Pseudo-code of the procedure Make$\updownarrow$ExprMinimal.

This completes the proof of Theorem 7.17. □

Now that we have established the correctness of the global description of the algorithm, we can start working out the details. In Figure 7.3, we give the pseudo-code of procedure Make$\updownarrow$ExprMinimal (for line 9 of MakeMinimal), 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.

As we will see in the proof of Theorem 7.20(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. For the moment, it is not important how exactly we determine $\alpha_{E_{1,i}}$. In the proof of Lemma 7.34, 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 try to 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 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 7.20(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$\updownarrow$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 6.14(2)) and we can skip line M$\updownarrow$M.20.

If E_1 is not an $\uparrow$-expression, but a $\downarrow$-expression, then lines M$\updownarrow$M.8–M$\updownarrow$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 7.18 (cf. Example 7.2) 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

$$\mathcal{S}(E) = \binom{\alpha_1\alpha_2\alpha_3\alpha_4 c(\alpha_5)}{c(\alpha_1\alpha_2\alpha_3\alpha_4)\alpha_5}.$$

The argument E_1 of E is a minimal, alternating $\uparrow$-expression. E_1 itself has one $\downarrow$-argument, $E_{1,i} = \langle\downarrow \langle\updownarrow \alpha_4\rangle\, \alpha_5\rangle$. In line M$\updownarrow$M.5, we substitute it by $\left\langle\updownarrow \alpha_{E_{1,i}}\right\rangle = \langle\updownarrow \alpha_4 c(\alpha_5)\rangle$, yielding

$$\widehat{E}_1 = \langle\uparrow \alpha_1 \langle\updownarrow \alpha_2\rangle\, \alpha_3 \langle\updownarrow \alpha_4 c(\alpha_5)\rangle\rangle\,.$$

Subsequently, in lines M$\updownarrow$M.7–M$\updownarrow$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

$$\widehat{E}_1 = \langle\uparrow \langle\updownarrow \alpha_1\alpha_2\alpha_3\alpha_4 c(\alpha_5)\rangle\rangle\,.$$

$\widehat{E}_1$ has $m = 1$ argument left, which is an $\updownarrow$-argument. Hence, it violates Property ($\mathcal{D}_{\text{Min}}$.3). According to line M$\updownarrow$M.20, E' is set to this $\updownarrow$-argument: $E' = \langle\updownarrow \alpha_1\alpha_2\alpha_3\alpha_4$ $c(\alpha_5)\rangle$. By Theorem 5.3, E' is the only minimal DNA expression with $\mathcal{S}(E') = \mathcal{S}(E)$, i.e., with $E' \equiv E$. ■

[1]This is a very natural assumption, but is not necessary for the correctness of procedure `Make↕ExprMinimal`.

Example 7.19 (cf. Example 7.3) Let

$$E = \langle \updownarrow \langle \downarrow \langle \updownarrow \alpha_1\alpha_2\alpha_3\alpha_4 c(\alpha_5)\rangle \langle \updownarrow \alpha_6\rangle \alpha_7\rangle\rangle \,.$$

E is an $\updownarrow$-expression, for which

$$\mathcal{S}(E) = \binom{\alpha_1\alpha_2\alpha_3\alpha_4 c(\alpha_5)}{c(\alpha_1\alpha_2\alpha_3\alpha_4)\alpha_5} \triangledown \binom{\alpha_6 c(\alpha_7)}{c(\alpha_6)\alpha_7}.$$

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$\updownarrow$M.11, we substitute $\langle \updownarrow \alpha_6\rangle \alpha_7$ by $\langle \updownarrow \alpha_6 c(\alpha_7)\rangle$, yielding

$$\widehat{E}_1 = \langle \downarrow \langle \updownarrow \alpha_1\alpha_2\alpha_3\alpha_4 c(\alpha_5)\rangle \langle \updownarrow \alpha_6 c(\alpha_7)\rangle\rangle \,.$$

This time, $\widehat{E}_1$ has $m = 2$ arguments left. Hence, $E' = \widehat{E}_1$. By (the analogue for upper nick letters of) Lemma 6.14(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 7.20 *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. *E' is independent of the order in which $\downarrow$-arguments are considered in line M$\updownarrow$M.4 and independent of the order in which $\mathcal{N}$-word-arguments are considered in line M$\updownarrow$M.7.*

Proof:

1. E_1 is a minimal $\uparrow$-expression. By Corollary 6.2, 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:

   $$\widehat{E}_1 \text{ is a minimal } \uparrow\text{-expression satisfying } \left\langle \updownarrow \widehat{E}_1 \right\rangle \equiv E. \tag{7.7}$$

   - 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 (7.7) 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 $\left\langle \updownarrow \alpha_{E_{1,i_0}} \right\rangle$. By Lemma 2.15(2), $L(\mathcal{S}(\left\langle \updownarrow \alpha_{E_{1,i_0}} \right\rangle)), R(\mathcal{S}(\left\langle \updownarrow \alpha_{E_{1,i_0}} \right\rangle)) \in \mathcal{A}_{\pm}$. 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 6.16, $\widehat{E}_1$ 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 6.6, 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 $i = 1, \ldots, n-1$, $y_i = \vartriangle$ 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

$$y'_{i_0-1} = \begin{cases} \vartriangle & \text{if both } \varepsilon_{1,i_0-1} \text{ and } \langle \updownarrow \alpha_{E_{1,i_0}} \rangle \\ & \text{are expression-arguments} \\ \lambda & \text{otherwise,} \end{cases}$$

$$y'_{i_0} = \begin{cases} \vartriangle & \text{if both } \langle \updownarrow \alpha_{E_{1,i_0}} \rangle \text{ and } \varepsilon_{1,i_0+1} \\ & \text{are expression-arguments} \\ \lambda & \text{otherwise.} \end{cases}$$

We must prove that

$$y_{i_0-1}\ \kappa(\mathcal{S}(E_{1,i_0}))\ y_{i_0} = y'_{i_0-1} \binom{\alpha_{E_{1,i_0}}}{c(\alpha_{E_{1,i_0}})}\ y'_{i_0}. \tag{7.8}$$

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, $y_{i_0-1} = \vartriangle$, 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 $y'_{i_0-1} = \vartriangle$, 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 3.13 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 6.3, 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 (7.8) holds.

We conclude that Property (7.7) 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 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 7.3 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:

$\widehat{E}_1$ is an $\uparrow$-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 $\alpha_{1,i}$. (7.9)

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 (7.9) is an invariant for the second for-loop, is the same as that of the proof of Property (7.7) 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 (7.9) is valid before the first iteration of the second for-loop.

- Suppose that Property (7.9) 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 (7.9). 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

$$y'_{i_0-2} = \begin{cases} \vartriangle & \text{if both } \varepsilon_{1,i_0-2} \text{ and } \langle \updownarrow \alpha_{1,i_0-1}\alpha_{1,i_0}\alpha_{1,i_0+1} \rangle \\ & \text{are expression-arguments} \\ \lambda & \text{otherwise,} \end{cases}$$

$$y'_{i_0+1} = \begin{cases} \vartriangle & \text{if both } \langle \updownarrow \alpha_{1,i_0-1}\alpha_{1,i_0}\alpha_{1,i_0+1} \rangle \text{ and } \varepsilon_{1,i_0+2} \\ & \text{are expression-arguments} \\ \lambda & \text{otherwise.} \end{cases}$$

  We must prove that

$$\begin{aligned} y_{i_0-2} \binom{\alpha_{1,i_0-1}}{c(\alpha_{1,i_0-1})}\ y_{i_0-1} \binom{\alpha_{1,i_0}}{c(\alpha_{1,i_0})}\ y_{i_0} \binom{\alpha_{1,i_0+1}}{c(\alpha_{1,i_0+1})}\ y_{i_0+1} \\ = y'_{i_0-2} \binom{\alpha_{1,i_0-1}\alpha_{1,i_0}\alpha_{1,i_0+1}}{c(\alpha_{1,i_0-1}\alpha_{1,i_0}\alpha_{1,i_0+1})}\ y'_{i_0+1}.\end{aligned} \tag{7.10}$$

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 $2 \leq i_0 - 1$ and $i_0 + 1 \leq n - 1$.

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

$$\begin{aligned} &y_{i_0-2} = \triangle, \text{ if and only if } \varepsilon_{1,i_0-2} \text{ is an expression-argument} \\ &y_{i_0-1} = y_{i_0} = \lambda, \text{ and} \\ &y_{i_0+1} = \triangle, \text{ if and only if } \varepsilon_{1,i_0+2} \text{ is an expression-argument.} \end{aligned}$$

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

$$\begin{aligned} &y'_{i_0-2} = \triangle, \text{ if and only if } \varepsilon_{1,i_0-2} \text{ is an expression-argument} \\ &y'_{i_0+1} = \triangle, \text{ if and only if } \varepsilon_{1,i_0+2} \text{ is an expression-argument.} \end{aligned}$$

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 (7.10) holds, and $\mathcal{S}(\left\langle \updownarrow \widehat{E}_1 \right\rangle)$ is the same before and after the substitution.

We conclude that Property (7.9) 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 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 7.3 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 $\widehat{E}_1 \equiv \left\langle \updownarrow \widehat{E}_1 \right\rangle \equiv E$.

Now, 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 (7.9), 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 $E' \equiv E = \langle \updownarrow E_1 \rangle$.

As we have seen in the proof of Theorem 7.17(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 7.20. $\square$

The next instruction from `MakeMinimal` we refine is the one in line 18. In Figure 7.4 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 6.6, 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`, it is useful to establish such a relation for a more general set of $\uparrow$-expressions.

Lemma 7.21 *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 $i = 1, \ldots, n-1$, $y_i = \vartriangle$ 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.

Of course, there is an analogous result for $\downarrow$-expressions. The proof of this result is similar to that of Lemma 6.6. 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 (2.9)),

$$X = \nu^+(X_1) y_1 \nu^+(X_2) y_2 \ldots y_{n-1} \nu^+(X_n),$$

where for $i = 1, \ldots, n-1$, $y_i = \vartriangle$ 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 3.5, for $i = 1, \ldots, n$, $X_i = \mathcal{S}^+(\varepsilon_i)$ is nick free, and in particular, $\nu^+(X_i) = X_i$. We can thus reduce the semantics to

$$X = X_1 y_1 X_2 y_2 \ldots y_{n-1} X_n,$$

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 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 2.15(2), $R(X_i) = R(\mathcal{S}(\varepsilon_i)) \in \mathcal{A}_\pm$.

```
Dni.1.   Denickify ($E_i$)
           // rewrites a minimal $\downarrow$-expression $E_i$ which is not alternating,
           // into a minimal, nick free DNA expression $E'_i$
           // satisfying $E'_i \equiv_\triangledown E_i$;
           // uses local rearrangements of the DNA expression for this
Dni.2.   {
Dni.3.     $\widehat{E}_i = E_i$;
Dni.4.     while ($\widehat{E}_i$ is not alternating)
Dni.5.     do  select two consecutive expression-arguments $\widehat{\varepsilon}_{j-1}, \widehat{\varepsilon}_j$ of $\widehat{E}_i$;
Dni.6.         if ($\widehat{\varepsilon}_{j-1}$ is an $\uparrow$-expression $\langle\uparrow \ldots \langle\updownarrow \alpha_{j-1,m_{j-1}}\rangle\rangle$)
Dni.7.         then if ($\widehat{\varepsilon}_j$ is an $\uparrow$-expression $\langle\uparrow \langle\updownarrow \alpha_{j,1}\rangle \ldots\rangle$)
Dni.8.              then substitute $\widehat{\varepsilon}_{j-1}\widehat{\varepsilon}_j$ in $\widehat{E}_i$
                             by $\langle\uparrow \ldots \langle\updownarrow \alpha_{j-1,m_{j-1}}\alpha_{j,1}\rangle \ldots\rangle$;
Dni.9.              else   // $\widehat{\varepsilon}_j$ is an $\updownarrow$-expression $\langle\updownarrow \alpha_{j,1}\rangle$
Dni.10.                  substitute $\widehat{\varepsilon}_{j-1}\widehat{\varepsilon}_j$ in $\widehat{E}_i$
                             by $\langle\uparrow \ldots \langle\updownarrow \alpha_{j-1,m_{j-1}}\alpha_{j,1}\rangle\rangle$;
Dni.11.             fi
Dni.12.        else   // $\widehat{\varepsilon}_{j-1}$ is an $\updownarrow$-expression $\langle\updownarrow \alpha_{j-1,1}\rangle$
Dni.13.             if ($\widehat{\varepsilon}_j$ is an $\uparrow$-expression $\langle\uparrow \langle\updownarrow \alpha_{j,1}\rangle \ldots\rangle$)
Dni.14.             then substitute $\widehat{\varepsilon}_{j-1}\widehat{\varepsilon}_j$ in $\widehat{E}_i$
                             by $\langle\uparrow \langle\updownarrow \alpha_{j-1,1}\alpha_{j,1}\rangle \ldots\rangle$;
Dni.15.             else   // $\widehat{\varepsilon}_j$ is an $\updownarrow$-expression $\langle\updownarrow \alpha_{j,1}\rangle$
Dni.16.                  substitute $\widehat{\varepsilon}_{j-1}\widehat{\varepsilon}_j$ in $\widehat{E}_i$
                             by $\langle\updownarrow \alpha_{j-1,1}\alpha_{j,1}\rangle$;
Dni.17.             fi
Dni.18.        fi
Dni.19.    od
             // $\widehat{E}_i$ is alternating
Dni.20.    if ($\widehat{E}_i$ has only one argument $E_{i,1}$ left)
Dni.21.    then substitute $\widehat{E}_i$ by $E_{i,1}$;                            ($\mathcal{D}_{\mathtt{Min}}$.3)
Dni.22.    else   // $\widehat{E}_i$ has at least two arguments
Dni.23.        if (both the first argument and the last argument of $\widehat{E}_i$
                 are $\uparrow$-arguments)
Dni.24.        then substitute $\widehat{E}_i$ by a minimal $\uparrow$-expression $\widehat{E}'_i$
                       satisfying $\widehat{E}'_i \equiv \widehat{E}_i$;
                       (procedure RotateToMinimal)                  ($\mathcal{D}_{\mathtt{Min}}$.6)
Dni.25.        fi
Dni.26.    fi
Dni.27.    $E'_i = \widehat{E}_i$;
Dni.28.  }
```

Figure 7.4: Pseudo-code of the procedure `Denickify`.

Finally, if ε_i is a $\downarrow$-expression, then by Property ($\mathcal{D}_{\mathrm{Min}}$.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 2.15(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 2.15(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 7.24, 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 7.24(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 anymore. 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.

Example 7.22 (cf. Example 7.3) 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

$$\mathcal{S}(E_i) = \binom{\alpha_1\alpha_2\alpha_3\alpha_4 c(\alpha_5)}{c(\alpha_1\alpha_2\alpha_3\alpha_4)\alpha_5} \triangledown \binom{\alpha_6 c(\alpha_7)}{c(\alpha_6)\alpha_7}.$$

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 5.3, E'_i is minimal. ■

Example 7.23 (cf. Example 7.8) Let

$$\begin{aligned} E_i = \langle\uparrow\ & \langle\updownarrow \alpha_1\alpha_2\alpha_3\alpha_4 c(\alpha_5)\alpha_6 c(\alpha_7)\rangle \\ & \langle\updownarrow \alpha_8\alpha_9\rangle\ \alpha_{10}\ \langle\downarrow \langle\updownarrow \alpha_{11}\rangle\, \alpha_{12}\, \langle\updownarrow \alpha_{13}\rangle\rangle\ \langle\updownarrow \alpha_{14}\rangle \\ & \langle\downarrow \langle\updownarrow \alpha_{15}\rangle\, \alpha_{16}\, \langle\updownarrow \alpha_{17}\rangle\rangle\ \alpha_{18}\ \langle\updownarrow \alpha_{19}\rangle\ \langle\updownarrow \alpha_{20}\rangle\ \rangle\,, \end{aligned}$$

[2] We could have reached this conclusion also using Lemma 6.6 instead of Lemma 7.21. 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 7.21.

for which

$$\begin{aligned}\mathcal{S}(E_i) \;=\; & \binom{\alpha_1\alpha_2\alpha_3\alpha_4 c(\alpha_5)\alpha_6 c(\alpha_7)}{c(\alpha_1\alpha_2\alpha_3\alpha_4)\alpha_5 c(\alpha_6)\alpha_7} \triangle \binom{\alpha_8\alpha_9}{c(\alpha_8\alpha_9)} \binom{\alpha_{10}}{-} \binom{\alpha_{11}}{c(\alpha_{11})} \binom{-}{\alpha_{12}} \cdot \\ & \binom{\alpha_{13}}{c(\alpha_{13})} \triangle \binom{\alpha_{14}}{c(\alpha_{14})} \triangle \binom{\alpha_{15}}{c(\alpha_{15})} \binom{-}{\alpha_{16}} \binom{\alpha_{17}}{c(\alpha_{17})} \binom{\alpha_{18}}{-} \binom{\alpha_{19}}{c(\alpha_{19})} \triangle \binom{\alpha_{20}}{c(\alpha_{20})}.\end{aligned}$$

E_i is minimal and not nick free. It has seven expression-arguments, clustered in three groups of consecutive expression-arguments. 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

$$\begin{aligned}\widehat{E}_i = \langle\uparrow\ & \langle\updownarrow\ \alpha_1\alpha_2\alpha_3\alpha_4 c(\alpha_5)\alpha_6 c(\alpha_7)\alpha_8\alpha_9\rangle\ \alpha_{10} \\ & \langle\downarrow\ \langle\updownarrow\ \alpha_{11}\rangle\ \alpha_{12}\ \langle\updownarrow\ \alpha_{13}\rangle\rangle\ \ \langle\updownarrow\ \alpha_{14}\rangle \\ & \langle\downarrow\ \langle\updownarrow\ \alpha_{15}\rangle\ \alpha_{16}\ \langle\updownarrow\ \alpha_{17}\rangle\rangle\ \ \alpha_{18}\ \ \langle\updownarrow\ \alpha_{19}\rangle\ \ \langle\updownarrow\ \alpha_{20}\rangle\ \rangle\end{aligned}$$

(by applying line Dni.16),

$$\begin{aligned}\widehat{E}_i = \langle\uparrow\ & \langle\updownarrow\ \alpha_1\alpha_2\alpha_3\alpha_4 c(\alpha_5)\alpha_6 c(\alpha_7)\alpha_8\alpha_9\rangle\ \alpha_{10} \\ & \langle\downarrow\ \langle\updownarrow\ \alpha_{11}\rangle\ \alpha_{12}\ \langle\updownarrow\ \alpha_{13}\alpha_{14}\rangle\rangle \\ & \langle\downarrow\ \langle\updownarrow\ \alpha_{15}\rangle\ \alpha_{16}\ \langle\updownarrow\ \alpha_{17}\rangle\rangle\ \ \alpha_{18}\ \ \langle\updownarrow\ \alpha_{19}\rangle\ \ \langle\updownarrow\ \alpha_{20}\rangle\ \rangle\end{aligned}$$

(by applying line Dni.10),

$$\begin{aligned}\widehat{E}_i = \langle\uparrow\ & \langle\updownarrow\ \alpha_1\alpha_2\alpha_3\alpha_4 c(\alpha_5)\alpha_6 c(\alpha_7)\alpha_8\alpha_9\rangle\ \alpha_{10} \\ & \langle\downarrow\ \langle\updownarrow\ \alpha_{11}\rangle\ \alpha_{12}\ \langle\updownarrow\ \alpha_{13}\alpha_{14}\alpha_{15}\rangle\ \alpha_{16}\ \langle\updownarrow\ \alpha_{17}\rangle\rangle\ \ \alpha_{18}\ \ \langle\updownarrow\ \alpha_{19}\rangle\ \ \langle\updownarrow\ \alpha_{20}\rangle\ \rangle\end{aligned}$$

(by applying line Dni.8), and

$$\begin{aligned}\widehat{E}_i = \langle\uparrow\ & \langle\updownarrow\ \alpha_1\alpha_2\alpha_3\alpha_4 c(\alpha_5)\alpha_6 c(\alpha_7)\alpha_8\alpha_9\rangle\ \alpha_{10} \\ & \langle\downarrow\ \langle\updownarrow\ \alpha_{11}\rangle\ \alpha_{12}\ \langle\updownarrow\ \alpha_{13}\alpha_{14}\alpha_{15}\rangle\ \alpha_{16}\ \langle\updownarrow\ \alpha_{17}\rangle\rangle\ \ \alpha_{18}\ \ \langle\updownarrow\ \alpha_{19}\alpha_{20}\rangle\ \rangle\end{aligned}$$

(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

$$\begin{aligned}\mathcal{S}(E'_i) \;=\; & \binom{\alpha_1\alpha_2\alpha_3\alpha_4 c(\alpha_5)\alpha_6 c(\alpha_7)\alpha_8\alpha_9}{c(\alpha_1\alpha_2\alpha_3\alpha_4)\alpha_5 c(\alpha_6)\alpha_7 c(\alpha_8\alpha_9)} \binom{\alpha_{10}}{-} \binom{\alpha_{11}}{c(\alpha_{11})} \binom{-}{\alpha_{12}} \cdot \\ & \binom{\alpha_{13}\alpha_{14}\alpha_{15}}{c(\alpha_{13}\alpha_{14}\alpha_{15})} \binom{-}{\alpha_{16}} \binom{\alpha_{17}}{c(\alpha_{17})} \binom{\alpha_{18}}{-} \binom{\alpha_{19}\alpha_{20}}{c(\alpha_{19}\alpha_{20})}.\end{aligned}$$

Indeed, $E'_i \equiv_\triangledown E_i$. Moreover, it is easily verified that E'_i has all six properties from Lemma 6.15 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 7.24 *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 $(\mathcal{D}_{\mathrm{Min}}.1)$, $(\mathcal{D}_{\mathrm{Min}}.2)$, $(\mathcal{D}_{\mathrm{Min}}.4)$ and $(\mathcal{D}_{\mathrm{Min}}.5)$, and each inner occurrence of $\uparrow$ or $\downarrow$ in $\widehat{E}_i$ has at least two arguments. (7.11)

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 $(\mathcal{D}_{\mathrm{Min}}.1)$, $(\mathcal{D}_{\mathrm{Min}}.2)$, $(\mathcal{D}_{\mathrm{Min}}.4)$ and $(\mathcal{D}_{\mathrm{Min}}.5)$. 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 $(\mathcal{D}_{\mathrm{Min}}.3)$. Finally, by Property $(\mathcal{D}_{\mathrm{Min}}.5)$ of $\widehat{E}_i$, $\widehat{\varepsilon}_j$ has Property $(\mathcal{D}_{\mathrm{Min}}.6)$. This implies that $\widehat{\varepsilon}_j$ has all six properties from Lemma 6.15, and thus is minimal. By Property $(\mathcal{D}_{\mathrm{Min}}.4)$ and Lemma 3.5, $\widehat{\varepsilon}_j$ 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 $(\mathcal{D}_{\mathrm{Min}}.6)$. 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 $(\mathcal{D}_{\mathrm{Min}}.3)$. Consequently, in this case, $\widehat{E}_i$ itself is also minimal.

- 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 6.15, $\widehat{E}_i$ has Properties $(\mathcal{D}_{\mathrm{Min}}.1)$–$(\mathcal{D}_{\mathrm{Min}}.6)$. In particular, by Lemma 6.17(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 (7.11) is valid.
- Suppose that before a certain operation of the while-loop, Property (7.11) 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 6.2, 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).

Let $\widehat{\varepsilon}_{j-1} = \left\langle \uparrow \widehat{\varepsilon}_{j-1,1} \dots \widehat{\varepsilon}_{j-1,m_{j-1}} \right\rangle$ and $\widehat{\varepsilon}_j = \left\langle \uparrow \widehat{\varepsilon}_{j,1} \dots \widehat{\varepsilon}_{j,m_j} \right\rangle$ 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 (7.11), $m_{j-1}, m_j \geq 2$. 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 6.17(5), the last argument $\widehat{\varepsilon}_{j-1,m_{j-1}}$ of $\widehat{\varepsilon}_{j-1}$ is an $\updownarrow$-expression $\left\langle \updownarrow \alpha_{j-1,m_{j-1}} \right\rangle$ 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 (7.11), the $\uparrow$-arguments $\widehat{\varepsilon}_{j-1}$ and $\widehat{\varepsilon}_j$ of $\widehat{E}_i$ are minimal and nick free. Hence, by Corollary 6.7,

$$\begin{aligned}
\mathcal{S}(\widehat{\varepsilon}_{j-1}) &= \mathcal{S}^+(\widehat{\varepsilon}_{j-1,1}) \dots \mathcal{S}^+(\widehat{\varepsilon}_{j-1,m_{j-1}-1}) \binom{\alpha_{j-1,m_{j-1}}}{c(\alpha_{j-1,m_{j-1}})}, \\
\mathcal{S}(\widehat{\varepsilon}_j) &= \binom{\alpha_{j,1}}{c(\alpha_{j,1})} \mathcal{S}^+(\widehat{\varepsilon}_{j,2}) \dots \mathcal{S}^+(\widehat{\varepsilon}_{j,m_j}) \qquad \text{and} \\
\mathcal{S}(\langle \downarrow \widehat{\varepsilon}_{j-1} \widehat{\varepsilon}_j \rangle) &= \mathcal{S}^+(\widehat{\varepsilon}_{j-1,1}) \dots \mathcal{S}^+(\widehat{\varepsilon}_{j-1,m_{j-1}-1}) \binom{\alpha_{j-1,m_{j-1}}}{c(\alpha_{j-1,m_{j-1}})}^{\triangledown} \\
&\quad \binom{\alpha_{j,1}}{c(\alpha_{j,1})} \mathcal{S}^+(\widehat{\varepsilon}_{j,2}) \dots \mathcal{S}^+(\widehat{\varepsilon}_{j,m_j}). \qquad (7.12)
\end{aligned}$$

Now, let

$$\widehat{\varepsilon}'_j = \left\langle \uparrow \widehat{\varepsilon}_{j-1,1} \dots \widehat{\varepsilon}_{j-1,m_{j-1}-1} \left\langle \updownarrow \alpha_{j-1,m_{j-1}} \alpha_{j,1} \right\rangle \widehat{\varepsilon}_{j,2} \dots \widehat{\varepsilon}_{j,m_j} \right\rangle.$$

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 6.7,

$$\begin{aligned}
\mathcal{S}(\widehat{\varepsilon}'_j) &= \mathcal{S}^+(\widehat{\varepsilon}_{j-1,1}) \dots \mathcal{S}^+(\widehat{\varepsilon}_{j-1,m_{j-1}-1}) \binom{\alpha_{j-1,m_{j-1}} \alpha_{j,1}}{c(\alpha_{j-1,m_{j-1}} \alpha_{j,1})} \cdot \\
&\quad \mathcal{S}^+(\widehat{\varepsilon}_{j,2}) \dots \mathcal{S}^+(\widehat{\varepsilon}_{j,m_j}). \qquad (7.13)
\end{aligned}$$

It follows from (7.12) and (7.13) 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 3.7, Lemma 3.6 and Property (7.11),

$$\begin{aligned}
\langle \downarrow \widehat{\varepsilon}_1 \dots \widehat{\varepsilon}_{j-2} \widehat{\varepsilon}'_j \widehat{\varepsilon}_{j+1} \dots \widehat{\varepsilon}_n \rangle &\equiv_{\triangledown} \langle \downarrow \widehat{\varepsilon}_1 \dots \widehat{\varepsilon}_{j-2} \langle \downarrow \widehat{\varepsilon}_{j-1} \widehat{\varepsilon}_j \rangle \widehat{\varepsilon}_{j+1} \dots \widehat{\varepsilon}_n \rangle \\
&\equiv \langle \downarrow \widehat{\varepsilon}_1 \dots \widehat{\varepsilon}_{j-2} \widehat{\varepsilon}_{j-1} \widehat{\varepsilon}_j \widehat{\varepsilon}_{j+1} \dots \widehat{\varepsilon}_n \rangle \\
&\equiv_{\triangledown} E_i. \qquad (7.14)
\end{aligned}$$

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 (7.11) 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 7.21, this implies that $\widehat{E}_i$ is nick free.

At the beginning of the proof, we deduced from Property (7.11), 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), anymore. 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 (7.11), 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 (7.11), 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 7.14, 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 6.15, 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 $\varepsilon_1, \dots, \varepsilon_n$ 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 $\varepsilon_{j_0}, \dots, \varepsilon_{j_1}$ 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 $\varepsilon_{j_0} \dots \varepsilon_{j_1}$ is rewritten into a single expression-argument $\widehat{\varepsilon}'_{j_1}$.

   By Corollary 6.2, 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 $j = j_0, \dots, j_1$, 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

$$\widehat{\varepsilon}'_{j_1} = \langle \updownarrow \alpha_{j_0,1} \dots \alpha_{j_1,1} \rangle \,,$$

   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 $\varepsilon_{j_0} \dots \varepsilon_{j_1}$. 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.

   Moreover, when we substitute $\widehat{\varepsilon}_{j-1}$ and $\widehat{\varepsilon}_j$ by $\widehat{\varepsilon}'_j$, the $\mathcal{N}$-word-arguments and $\downarrow$-arguments of the $\uparrow$-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 (7.14), 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.

By Theorem 6.9, 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 5.12(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 5.12(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 6.17(2), $m_j \geq 2$. By Lemma 6.17(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.

```
RtM.1.   RotateToMinimal ($E$)
           // rewrites an alternating $\downarrow$-expression $E = \langle\downarrow \varepsilon_1 \ldots \varepsilon_n\rangle$
           // with Properties ($\mathcal{D}_{\mathtt{Min}}$.1)-($\mathcal{D}_{\mathtt{Min}}$.5), for which either
           // the first argument $\varepsilon_1$ or the last argument $\varepsilon_n$ (or both)
           // is an $\uparrow$-argument, into a minimal $\uparrow$-expression $E'$
           // satisfying $E' \equiv E$;
           // uses local rearrangements of the DNA expression for this
RtM.2.   {
RtM.3.     if ($\varepsilon_1$ is an $\uparrow$-expression $\langle\uparrow \varepsilon_{1,1} \ldots \varepsilon_{1,m_1-1}\varepsilon_{1,m_1}\rangle$)
RtM.4.     then if ($\varepsilon_n$ is an $\uparrow$-expression $\langle\uparrow \varepsilon_{n,1}\varepsilon_{n,2} \ldots \varepsilon_{n,m_n}\rangle$)
RtM.5.          then $E' = \langle\uparrow \varepsilon_{1,1} \ldots \varepsilon_{1,m_1-1} \langle\downarrow \varepsilon_{1,m_1}\varepsilon_2 \ldots \varepsilon_{n-1}\varepsilon_{n,1}\rangle \varepsilon_{n,2} \ldots \varepsilon_{n,m_n}\rangle$;
                                                                         ($\mathcal{D}_{\mathtt{Min}}$.6)
RtM.6.          else  $E' = \langle\uparrow \varepsilon_{1,1} \ldots \varepsilon_{1,m_1-1} \langle\downarrow \varepsilon_{1,m_1}\varepsilon_2 \ldots \varepsilon_{n-1}\varepsilon_n\rangle\rangle$;
RtM.7.          fi
RtM.8.     else   // $\varepsilon_n$ must be an $\uparrow$-expression $\langle\uparrow \varepsilon_{n,1}\varepsilon_{n,2} \ldots \varepsilon_{n,m_n}\rangle$
RtM.9.          $E' = \langle\uparrow \langle\downarrow \varepsilon_1\varepsilon_2 \ldots \varepsilon_{n-1}\varepsilon_{n,1}\rangle \varepsilon_{n,2} \ldots \varepsilon_{n,m_n}\rangle$;
RtM.10.    fi
RtM.11.  }
```

Figure 7.5: Pseudo-code of the procedure `RotateToMinimal`.

This completes the proof of Theorem 7.24. □

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 is an $\uparrow$-argument. In the proof of Theorem 7.17(1) and (2), we have established that in addition, E_i is minimal and alternating. By Lemma 6.15, E_i has Properties ($\mathcal{D}_{\mathrm{Min}}$.1)–($\mathcal{D}_{\mathrm{Min}}$.6).

The situation in line 35 of `MakeMinimal` is not too different. We have to determine a minimal $\downarrow$-expression E' satisfying $E' \equiv E$. Here, E is an alternating $\uparrow$-expression with at least two arguments, for which both the first argument and the last argument are $\downarrow$-arguments. In the proof of Theorem 7.17(1) and (2), we have established that in addition, E has Properties ($\mathcal{D}_{\mathrm{Min}}$.1)–($\mathcal{D}_{\mathrm{Min}}$.5).

Finally, the situation in line Dni.24 of procedure `Denickify` is completely analogous to the previous situation: we have to determine a minimal $\uparrow$-expression $\widehat{E}_i'$ satisfying $\widehat{E}_i' \equiv \widehat{E}_i$. Here, $\widehat{E}_i$ is an alternating $\downarrow$-expression with at least two arguments, for which both the first argument and the last argument are $\uparrow$-arguments. In the proof of Theorem 7.24(1) and (2), we have established that in addition, $\widehat{E}_i$ has Properties ($\mathcal{D}_{\mathrm{Min}}$.1)–($\mathcal{D}_{\mathrm{Min}}$.5).

As the three cases are so similar, it is not surprising that they can be tackled by the same procedure `RotateToMinimal`. As usual, $\uparrow$-expressions and $\downarrow$-expressions are rewritten in analogous ways. In Figure 7.5, we give the procedure for $\downarrow$-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 7.27, we will see that the procedure is justified by Theorem 3.10 and Theorem 3.12. As we have depicted in Figure 3.1, Theorem 3.10 corresponds to a rotation in the structure tree. In the present situation, Theorem 3.12, which is based on Theorem 3.10, corresponds to two rotations in the structure tree.

We illustrate procedure `RotateToMinimal` by two examples.

Example 7.25 (cf. Example 7.5) 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 7.26 (cf. Example 7.11) 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 7.2. 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 6.15 and thus is minimal. ■

Procedure `RotateToMinimal` is also correct:

Theorem 7.27 *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 $\varepsilon_{1,1}, \ldots, \varepsilon_{1,m_1-1}, \varepsilon_{1,m_1}$.

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. 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 $\varepsilon_n = \langle\uparrow \varepsilon_{n,1}\varepsilon_{n,2}\dots\varepsilon_{n,m_n}\rangle$ for some $m_n \geq 1$ and $\mathcal{N}$-words and DNA expressions $\varepsilon_{n,1}, \varepsilon_{n,2}\dots, \varepsilon_{n,m_n}$. Hence,

  $$E = \langle\downarrow \langle\uparrow \varepsilon_{1,1}\dots\varepsilon_{1,m_1-1}\varepsilon_{1,m_1}\rangle\, \varepsilon_2\dots\varepsilon_{n-1}\, \langle\uparrow \varepsilon_{n,1}\varepsilon_{n,2}\dots\varepsilon_{n,m_n}\rangle\rangle\,.$$

  By Property ($\mathcal{D}_{\text{Min}}$.3), $m_n \geq 2$. When we apply Theorem 3.12(1) and (2) (with $r = 1$) to E, we find that

  $$E' = \langle\uparrow_0 \varepsilon_{1,1}\dots\varepsilon_{1,m_1-1}\, \langle\downarrow_1 \varepsilon_{1,m_1}\varepsilon_2\dots\varepsilon_{n-1}\varepsilon_{n,1}\rangle\, \varepsilon_{n,2}\dots\varepsilon_{n,m_n}\rangle$$

  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 6.15 and thus is minimal.

- Assume that ε_n is not an $\uparrow$-argument. Hence,

  $$E = \langle\downarrow \langle\uparrow \varepsilon_{1,1}\dots\varepsilon_{1,m_1-1}\varepsilon_{1,m_1}\rangle\, \varepsilon_2\dots\varepsilon_n\rangle\,,$$

  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 3.10(1) and (2),

  $$E' = \langle\uparrow \varepsilon_{1,1}\dots\varepsilon_{1,m_1-1}\, \langle\downarrow \varepsilon_{1,m_1}\varepsilon_2\dots\varepsilon_n\rangle\rangle$$

  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.

□

[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.

7.2 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 (7.1). Step by step, we rewrite this DNA expression into an 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 7.28 Let E be the DNA expression E_1^* from (7.1):

$$\begin{aligned} E = \langle\downarrow \langle\downarrow\ \langle\uparrow \langle\updownarrow \langle\downarrow\ \langle\updownarrow \langle\uparrow \alpha_1 \langle\updownarrow \langle\updownarrow \alpha_2\rangle\rangle\, \alpha_3 \langle\downarrow \langle\updownarrow \alpha_4\rangle\, \alpha_5\rangle\rangle\rangle\ \langle\updownarrow \alpha_6\rangle\ \alpha_7\ \rangle\rangle \\ \langle\downarrow \langle\updownarrow \alpha_8\rangle \langle\uparrow \langle\updownarrow \alpha_9\rangle\, \alpha_{10} \langle\updownarrow \alpha_{11}\rangle\rangle\, \alpha_{12} \langle\updownarrow \alpha_{13}\rangle\rangle\ \langle\updownarrow \alpha_{14}\rangle \\ \langle\downarrow \langle\updownarrow \alpha_{15}\rangle\, \alpha_{16} \langle\uparrow \langle\updownarrow \alpha_{17}\rangle\, \alpha_{18}\rangle\rangle\ \langle\uparrow \langle\updownarrow \alpha_{19}\rangle \langle\updownarrow \alpha_{20}\rangle\rangle\ \rangle\ \rangle \\ \langle\uparrow \langle\downarrow \alpha_{21}\rangle\rangle\ \langle\uparrow \langle\updownarrow \alpha_{22}\rangle\, \alpha_{23}\rangle\ \rangle\,, \end{aligned} \tag{7.15}$$

as depicted in Figure 7.6. 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 7.1, 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 7.18, by procedure `Make↕ExprMinimal`, E^s is substituted in E by $\langle\updownarrow \alpha_1\alpha_2\alpha_3\alpha_4\, c(\alpha_5)\rangle$. This yields

$$\begin{aligned} E = \langle\downarrow \langle\downarrow\ \langle\uparrow \langle\updownarrow \langle\downarrow\ \langle\updownarrow \alpha_1\alpha_2\alpha_3\alpha_4 c(\alpha_5)\rangle\ \langle\updownarrow \alpha_6\rangle\ \alpha_7\ \rangle\rangle \\ \langle\downarrow \langle\updownarrow \alpha_8\rangle \langle\uparrow \langle\updownarrow \alpha_9\rangle\, \alpha_{10} \langle\updownarrow \alpha_{11}\rangle\rangle\, \alpha_{12} \langle\updownarrow \alpha_{13}\rangle\rangle\ \langle\updownarrow \alpha_{14}\rangle \\ \langle\downarrow \langle\updownarrow \alpha_{15}\rangle\, \alpha_{16} \langle\uparrow \langle\updownarrow \alpha_{17}\rangle\, \alpha_{18}\rangle\rangle\ \langle\uparrow \langle\updownarrow \alpha_{19}\rangle \langle\updownarrow \alpha_{20}\rangle\rangle\ \rangle\ \rangle \\ \langle\uparrow \langle\downarrow \alpha_{21}\rangle\rangle\ \langle\uparrow \langle\updownarrow \alpha_{22}\rangle\, \alpha_{23}\rangle\ \rangle\,, \end{aligned} \tag{7.16}$$

as depicted in Figure 7.7.

We subsequently consider the DNA subexpression

$$E^s = \langle\updownarrow \langle\downarrow\ \langle\updownarrow \alpha_1\alpha_2\alpha_3\alpha_4 c(\alpha_5)\rangle\ \langle\updownarrow \alpha_6\rangle\ \alpha_7\ \rangle\rangle\,,$$

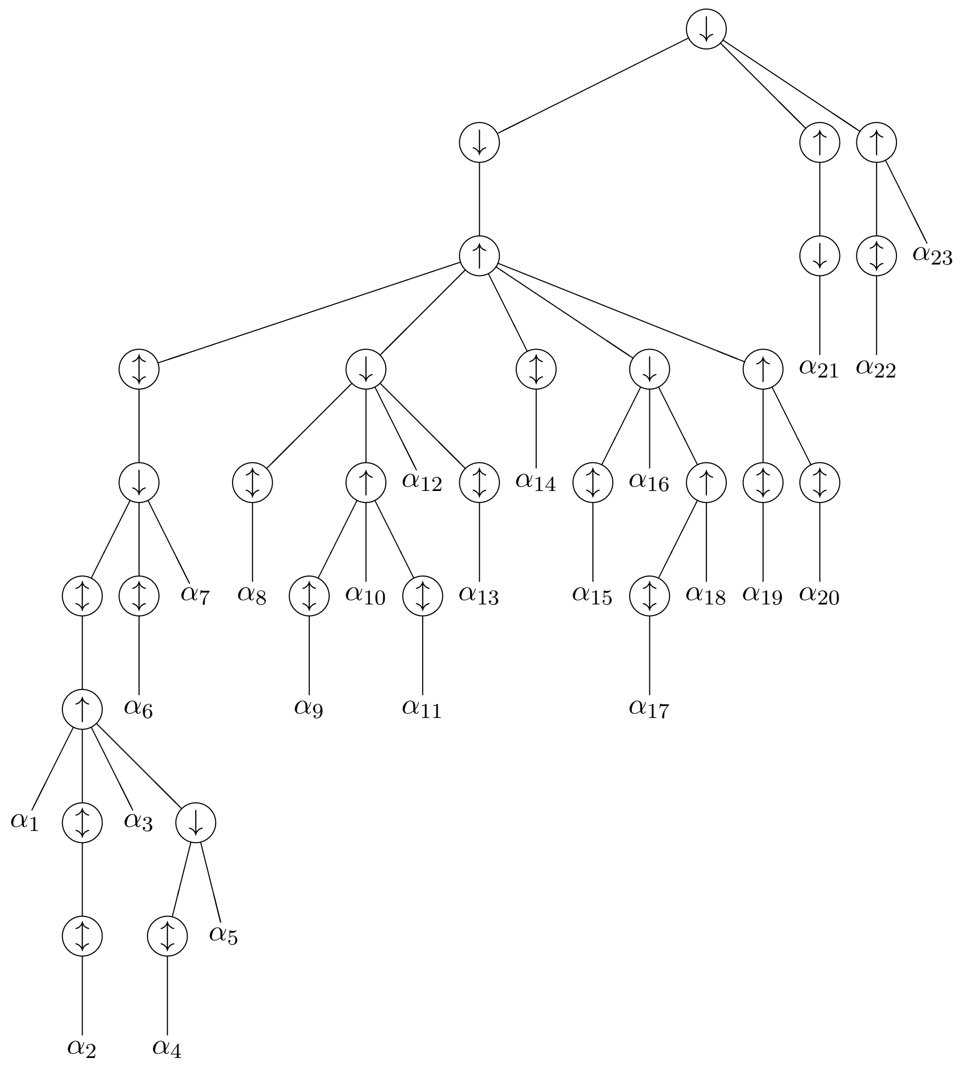


Figure 7.6: Structure tree of the example DNA expression E_1^* for the algorithm for minimality, (7.15).

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 7.19, by procedure `Make`$\updownarrow$`Expr-Minimal`, 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

$$\begin{aligned} E = \big\langle\downarrow \big\langle\downarrow \ &\langle\uparrow \langle\downarrow \langle\updownarrow \alpha_1\alpha_2\alpha_3\alpha_4 c(\alpha_5)\rangle \langle\updownarrow \alpha_6 c(\alpha_7)\rangle\rangle \\ &\quad \langle\downarrow \langle\updownarrow \alpha_8\rangle \langle\uparrow \langle\updownarrow \alpha_9\rangle \alpha_{10} \langle\updownarrow \alpha_{11}\rangle\rangle \alpha_{12} \langle\updownarrow \alpha_{13}\rangle\rangle \ \langle\updownarrow \alpha_{14}\rangle \\ &\quad \langle\downarrow \langle\updownarrow \alpha_{15}\rangle \alpha_{16} \langle\uparrow \langle\updownarrow \alpha_{17}\rangle \alpha_{18}\rangle\rangle \ \langle\uparrow \langle\updownarrow \alpha_{19}\rangle \langle\updownarrow \alpha_{20}\rangle\rangle \ \rangle \ \big\rangle \\ &\langle\uparrow \langle\downarrow \alpha_{21}\rangle\rangle \ \langle\uparrow \langle\updownarrow \alpha_{22}\rangle \alpha_{23}\rangle \ \big\rangle, \end{aligned} \tag{7.17}$$

as depicted in Figure 7.8.

We subsequently consider the DNA subexpression

$$\begin{aligned} E^s = \big\langle\uparrow \ &\langle\downarrow \langle\updownarrow \alpha_1\alpha_2\alpha_3\alpha_4 c(\alpha_5)\rangle \langle\updownarrow \alpha_6 c(\alpha_7)\rangle\rangle \\ &\langle\downarrow \langle\updownarrow \alpha_8\rangle \langle\uparrow \langle\updownarrow \alpha_9\rangle \alpha_{10} \langle\updownarrow \alpha_{11}\rangle\rangle \alpha_{12} \langle\updownarrow \alpha_{13}\rangle\rangle \ \langle\updownarrow \alpha_{14}\rangle \\ &\langle\downarrow \langle\updownarrow \alpha_{15}\rangle \alpha_{16} \langle\uparrow \langle\updownarrow \alpha_{17}\rangle \alpha_{18}\rangle\rangle \ \langle\uparrow \langle\updownarrow \alpha_{19}\rangle \langle\updownarrow \alpha_{20}\rangle\rangle \ \big\rangle, \end{aligned}$$

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

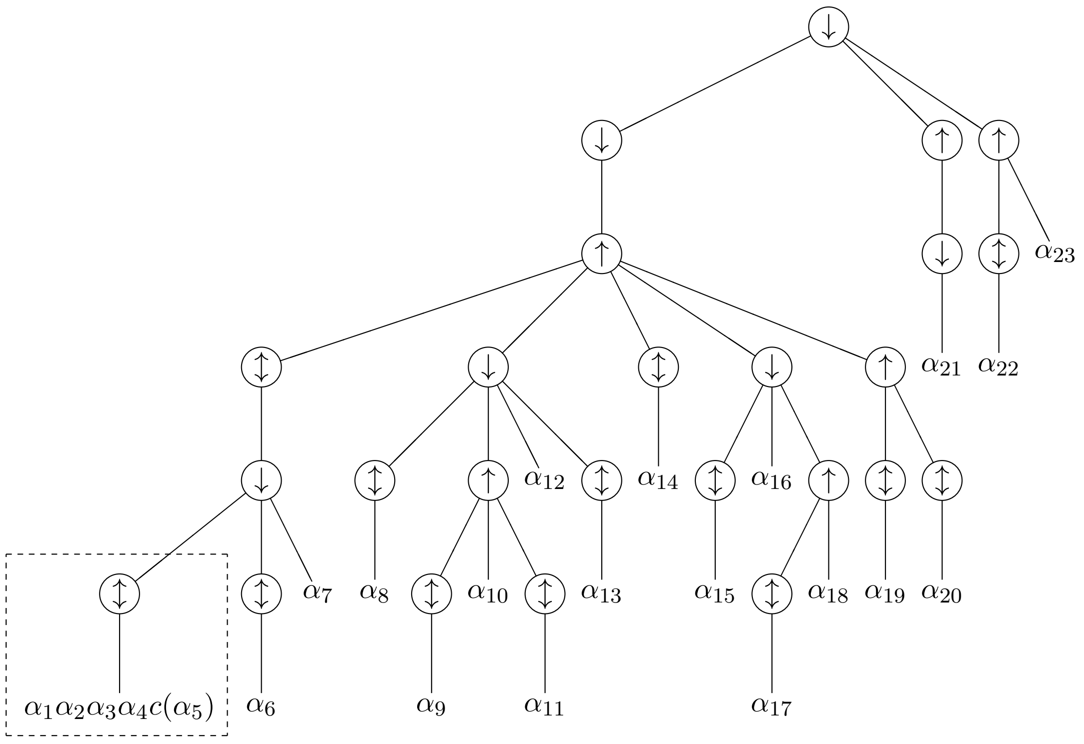


Figure 7.7: Structure tree of the example DNA expression for the algorithm for minimality, after two substitutions, (7.16). The dashed box encloses the part of the tree that has changed as compared to Figure 7.6.

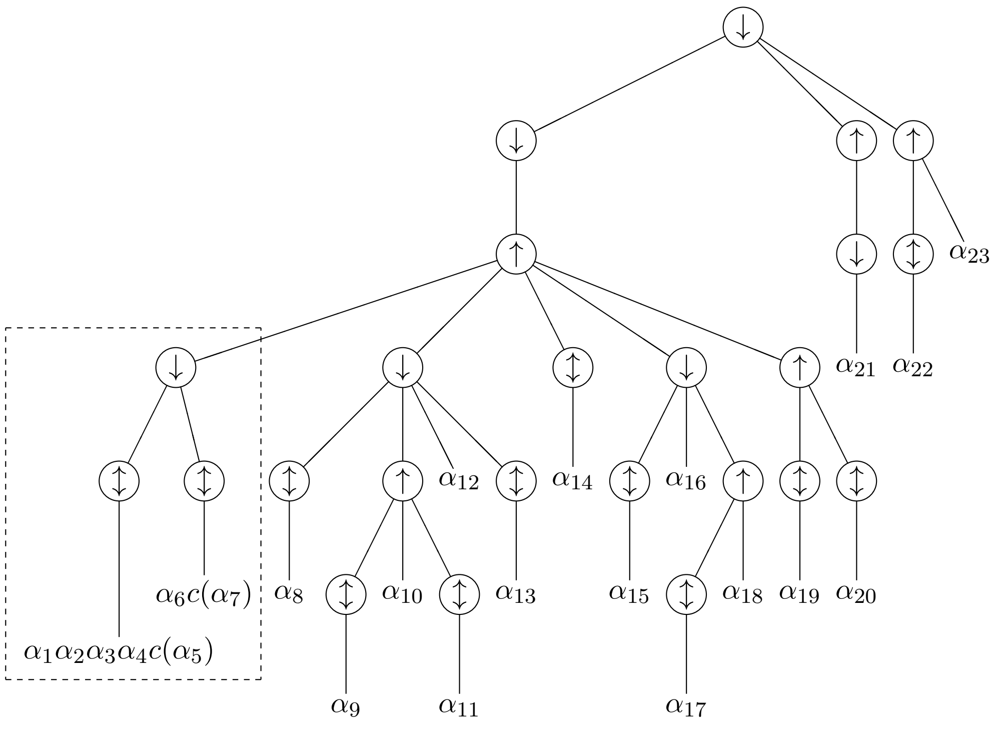


Figure 7.8: Structure tree of the example DNA expression for the algorithm for minimality, after three substitutions, (7.17). The dashed box encloses the part of the tree that has changed as compared to Figure 7.7.

are two such arguments, viz the first two arguments. As we have seen in Example 7.22, by procedure `Denickify`, the first argument

$$E_1 = \langle\downarrow \langle\updownarrow \alpha_1\alpha_2\alpha_3\alpha_4 c(\alpha_5)\rangle \langle\updownarrow \alpha_6 c(\alpha_7)\rangle\rangle$$

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

$$\begin{aligned} E = \big\langle\downarrow \big\langle\downarrow\ \big\langle\uparrow\ &\langle\updownarrow \alpha_1\alpha_2\alpha_3\alpha_4 c(\alpha_5)\alpha_6 c(\alpha_7)\rangle \\ &\langle\downarrow \langle\updownarrow \alpha_8\rangle \langle\uparrow \langle\updownarrow \alpha_9\rangle\, \alpha_{10}\, \langle\updownarrow \alpha_{11}\rangle\rangle\, \alpha_{12}\, \langle\updownarrow \alpha_{13}\rangle\rangle\ \ \langle\updownarrow \alpha_{14}\rangle \\ &\langle\downarrow \langle\updownarrow \alpha_{15}\rangle\, \alpha_{16}\, \langle\uparrow \langle\updownarrow \alpha_{17}\rangle\, \alpha_{18}\rangle\rangle\ \ \langle\uparrow \langle\updownarrow \alpha_{19}\rangle \langle\updownarrow \alpha_{20}\rangle\rangle\ \big\rangle\ \big\rangle \\ \langle\uparrow \langle\downarrow \alpha_{21}\rangle\rangle\ \ &\langle\uparrow \langle\updownarrow \alpha_{22}\rangle\, \alpha_{23}\rangle\ \big\rangle\,. \end{aligned}$$

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 7.4). When we substitute E_2 in E by $\widehat{E}_2$, we obtain

$$\begin{aligned} E = \big\langle\downarrow \big\langle\downarrow\ \big\langle\uparrow\ &\langle\updownarrow \alpha_1\alpha_2\alpha_3\alpha_4 c(\alpha_5)\alpha_6 c(\alpha_7)\rangle \\ &\langle\downarrow \langle\uparrow \langle\updownarrow \alpha_8\alpha_9\rangle\, \alpha_{10}\, \langle\updownarrow \alpha_{11}\rangle\rangle\, \alpha_{12}\, \langle\updownarrow \alpha_{13}\rangle\rangle\ \ \langle\updownarrow \alpha_{14}\rangle \\ &\langle\downarrow \langle\updownarrow \alpha_{15}\rangle\, \alpha_{16}\, \langle\uparrow \langle\updownarrow \alpha_{17}\rangle\, \alpha_{18}\rangle\rangle\ \ \langle\uparrow \langle\updownarrow \alpha_{19}\rangle \langle\updownarrow \alpha_{20}\rangle\rangle\ \big\rangle\ \big\rangle \\ \langle\uparrow \langle\downarrow \alpha_{21}\rangle\rangle\ \ &\langle\uparrow \langle\updownarrow \alpha_{22}\rangle\, \alpha_{23}\rangle\ \big\rangle\,, \end{aligned} \tag{7.18}$$

as depicted in Figure 7.9. In this overall DNA expression, the DNA subexpression E^s that we consider has become

$$\begin{aligned} E^s = \big\langle\uparrow\ &\langle\updownarrow \alpha_1\alpha_2\alpha_3\alpha_4 c(\alpha_5)\alpha_6 c(\alpha_7)\rangle \\ &\langle\downarrow \langle\uparrow \langle\updownarrow \alpha_8\alpha_9\rangle\, \alpha_{10}\, \langle\updownarrow \alpha_{11}\rangle\rangle\, \alpha_{12}\, \langle\updownarrow \alpha_{13}\rangle\rangle\ \ \langle\updownarrow \alpha_{14}\rangle \\ &\langle\downarrow \langle\updownarrow \alpha_{15}\rangle\, \alpha_{16}\, \langle\uparrow \langle\updownarrow \alpha_{17}\rangle\, \alpha_{18}\rangle\rangle\ \ \langle\uparrow \langle\updownarrow \alpha_{19}\rangle \langle\updownarrow \alpha_{20}\rangle\rangle\ \big\rangle\,. \end{aligned}$$

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 7.25, 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

$$\begin{aligned} E = \big\langle\downarrow \big\langle\downarrow\ \big\langle\uparrow\ &\langle\updownarrow \alpha_1\alpha_2\alpha_3\alpha_4 c(\alpha_5)\alpha_6 c(\alpha_7)\rangle \\ &\langle\uparrow \langle\updownarrow \alpha_8\alpha_9\rangle\, \alpha_{10}\, \langle\downarrow \langle\updownarrow \alpha_{11}\rangle\, \alpha_{12}\, \langle\updownarrow \alpha_{13}\rangle\rangle\rangle\ \ \langle\updownarrow \alpha_{14}\rangle \\ &\langle\downarrow \langle\updownarrow \alpha_{15}\rangle\, \alpha_{16}\, \langle\uparrow \langle\updownarrow \alpha_{17}\rangle\, \alpha_{18}\rangle\rangle\ \ \langle\uparrow \langle\updownarrow \alpha_{19}\rangle \langle\updownarrow \alpha_{20}\rangle\rangle\ \big\rangle\ \big\rangle \\ \langle\uparrow \langle\downarrow \alpha_{21}\rangle\rangle\ \ &\langle\uparrow \langle\updownarrow \alpha_{22}\rangle\, \alpha_{23}\rangle\ \big\rangle\,. \end{aligned}$$

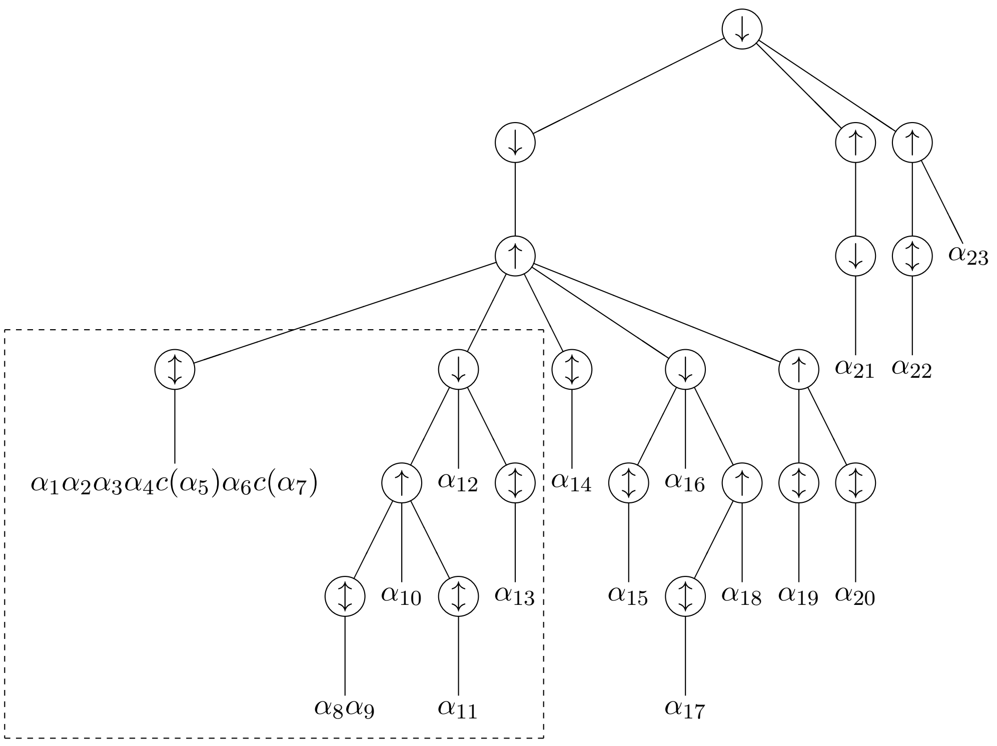


Figure 7.9: Structure tree of the example DNA expression for the algorithm for minimality, after five substitutions, (7.18). The dashed box encloses the part of the tree that has changed as compared to Figure 7.8.

We also apply procedure `RotateToMinimal` to the fourth argument

$$E_4 = \langle\downarrow \langle\updownarrow \alpha_{15}\rangle\, \alpha_{16}\, \langle\uparrow \langle\updownarrow \alpha_{17}\rangle\, \alpha_{18}\rangle\rangle\,.$$

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 7.6). This yields

$$\begin{aligned} E = \langle\downarrow \langle\downarrow\ &\langle\uparrow \langle\updownarrow \alpha_1\alpha_2\alpha_3\alpha_4 c(\alpha_5)\alpha_6 c(\alpha_7)\rangle \\ &\qquad \langle\uparrow \langle\updownarrow \alpha_8\alpha_9\rangle\, \alpha_{10}\, \langle\downarrow \langle\updownarrow \alpha_{11}\rangle\, \alpha_{12}\, \langle\updownarrow \alpha_{13}\rangle\rangle\rangle\ \langle\updownarrow \alpha_{14}\rangle \\ &\qquad \langle\uparrow \langle\downarrow \langle\updownarrow \alpha_{15}\rangle\, \alpha_{16}\, \langle\updownarrow \alpha_{17}\rangle\rangle\, \alpha_{18}\rangle\ \langle\uparrow \langle\updownarrow \alpha_{19}\rangle \langle\updownarrow \alpha_{20}\rangle\rangle\ \rangle\ \rangle \\ &\langle\uparrow \langle\downarrow \alpha_{21}\rangle\rangle\ \langle\uparrow \langle\updownarrow \alpha_{22}\rangle\, \alpha_{23}\rangle\ \rangle\,, \end{aligned} \tag{7.19}$$

as depicted in Figure 7.10. In this overall DNA expression, the DNA subexpression E^s that we consider has become

$$\begin{aligned} E^s = \langle\uparrow\ &\langle\updownarrow \alpha_1\alpha_2\alpha_3\alpha_4 c(\alpha_5)\alpha_6 c(\alpha_7)\rangle \\ &\langle\uparrow \langle\updownarrow \alpha_8\alpha_9\rangle\, \alpha_{10}\, \langle\downarrow \langle\updownarrow \alpha_{11}\rangle\, \alpha_{12}\, \langle\updownarrow \alpha_{13}\rangle\rangle\rangle\ \langle\updownarrow \alpha_{14}\rangle \\ &\langle\uparrow \langle\downarrow \langle\updownarrow \alpha_{15}\rangle\, \alpha_{16}\, \langle\updownarrow \alpha_{17}\rangle\rangle\, \alpha_{18}\rangle\ \langle\uparrow \langle\updownarrow \alpha_{19}\rangle \langle\updownarrow \alpha_{20}\rangle\rangle\ \rangle\,. \end{aligned}$$

We proceed with the fourth for-loop of `MakeMinimal`. As we have seen in Example 7.8, in this loop, we substitute the three $\uparrow$-arguments of E^s by their respective arguments.

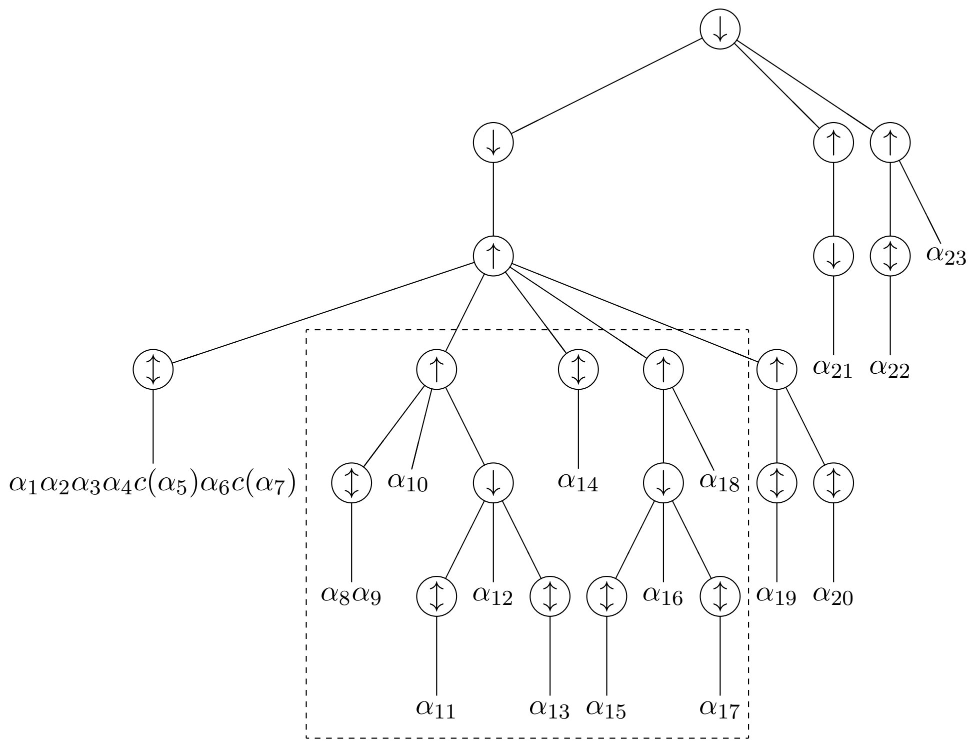


Figure 7.10: Structure tree of the example DNA expression for the algorithm for minimality, after seven substitutions, (7.19). The dashed box encloses the part of the tree that has changed as compared to Figure 7.9.

The overall DNA expression resulting from these three substitutions is

$$\begin{aligned} E = \langle\downarrow \langle\downarrow \langle\uparrow &\langle\updownarrow \alpha_1\alpha_2\alpha_3\alpha_4c(\alpha_5)\alpha_6c(\alpha_7)\rangle \\ &\langle\updownarrow \alpha_8\alpha_9\rangle\ \alpha_{10}\ \langle\downarrow \langle\updownarrow \alpha_{11}\rangle\, \alpha_{12}\, \langle\updownarrow \alpha_{13}\rangle\rangle\ \langle\updownarrow \alpha_{14}\rangle \\ &\langle\downarrow \langle\updownarrow \alpha_{15}\rangle\, \alpha_{16}\, \langle\updownarrow \alpha_{17}\rangle\rangle\ \alpha_{18}\ \langle\updownarrow \alpha_{19}\rangle\ \langle\updownarrow \alpha_{20}\rangle\ \rangle\ \rangle \\ \langle\uparrow \langle\downarrow \alpha_{21}\rangle\rangle\ &\langle\uparrow \langle\updownarrow \alpha_{22}\rangle\, \alpha_{23}\rangle\ \rangle, \end{aligned} \tag{7.20}$$

as depicted in Figure 7.11. As we have seen in Example 7.9, 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

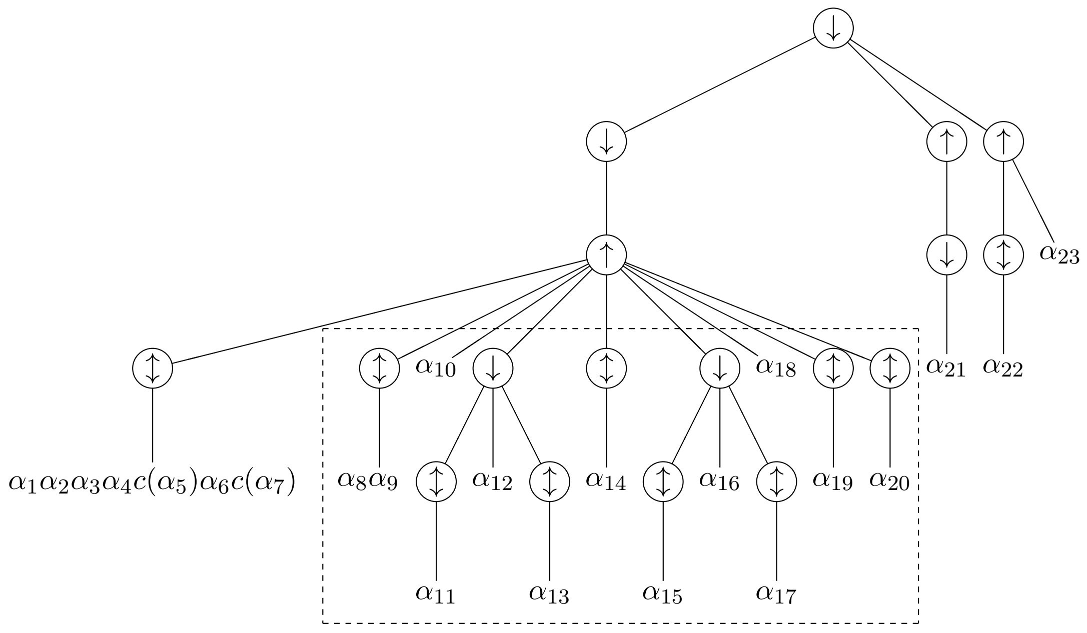
$$\begin{aligned} E^s = \langle\downarrow \langle\uparrow &\langle\updownarrow \alpha_1\alpha_2\alpha_3\alpha_4c(\alpha_5)\alpha_6c(\alpha_7)\rangle \\ &\langle\updownarrow \alpha_8\alpha_9\rangle\ \alpha_{10}\ \langle\downarrow \langle\updownarrow \alpha_{11}\rangle\, \alpha_{12}\, \langle\updownarrow \alpha_{13}\rangle\rangle\ \langle\updownarrow \alpha_{14}\rangle \\ &\langle\downarrow \langle\updownarrow \alpha_{15}\rangle\, \alpha_{16}\, \langle\updownarrow \alpha_{17}\rangle\rangle\ \alpha_{18}\ \langle\updownarrow \alpha_{19}\rangle\ \langle\updownarrow \alpha_{20}\rangle\ \rangle\ \rangle, \end{aligned}$$

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 7.23, by procedure `Denickify`, E_1 is substituted in E by

$$\begin{aligned} &\langle\uparrow \langle\updownarrow \alpha_1\alpha_2\alpha_3\alpha_4c(\alpha_5)\alpha_6c(\alpha_7)\alpha_8\alpha_9\rangle\ \alpha_{10} \\ &\quad \langle\downarrow \langle\updownarrow \alpha_{11}\rangle\, \alpha_{12}\, \langle\updownarrow \alpha_{13}\alpha_{14}\alpha_{15}\rangle\, \alpha_{16}\, \langle\updownarrow \alpha_{17}\rangle\rangle\ \alpha_{18}\ \langle\updownarrow \alpha_{19}\alpha_{20}\rangle\ \rangle. \end{aligned}$$

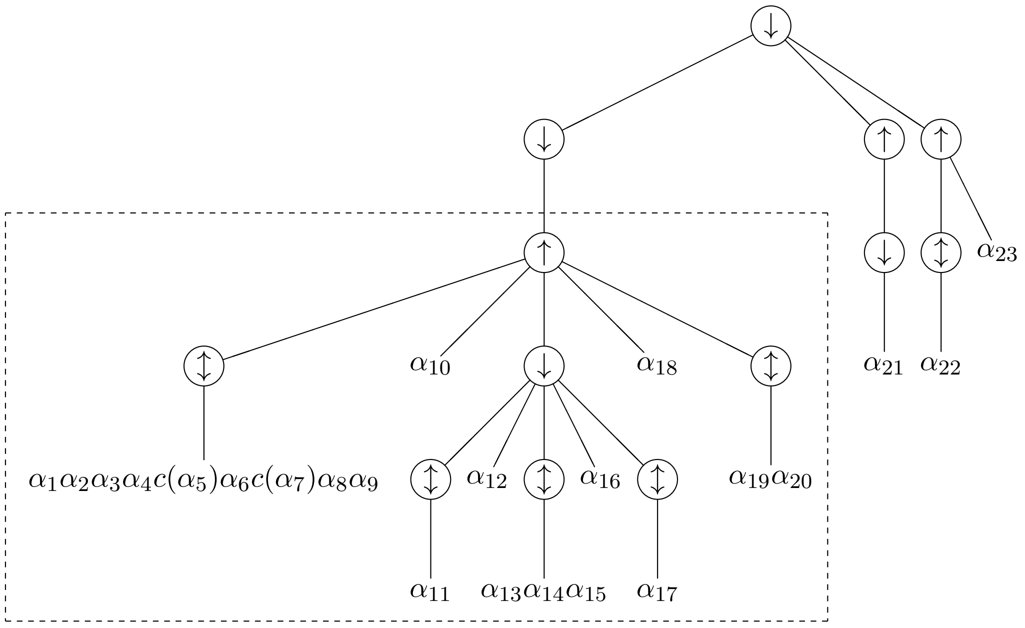
This yields

$$\begin{aligned} E = \langle\downarrow \langle\downarrow \langle\uparrow &\langle\updownarrow \alpha_1\alpha_2\alpha_3\alpha_4c(\alpha_5)\alpha_6c(\alpha_7)\alpha_8\alpha_9\rangle\ \alpha_{10} \\ &\langle\downarrow \langle\updownarrow \alpha_{11}\rangle\, \alpha_{12}\, \langle\updownarrow \alpha_{13}\alpha_{14}\alpha_{15}\rangle\, \alpha_{16}\, \langle\updownarrow \alpha_{17}\rangle\rangle\ \alpha_{18}\ \langle\updownarrow \alpha_{19}\alpha_{20}\rangle\ \rangle\ \rangle \\ \langle\uparrow \langle\downarrow \alpha_{21}\rangle\rangle\ &\langle\uparrow \langle\updownarrow \alpha_{22}\rangle\, \alpha_{23}\rangle\ \rangle, \end{aligned} \tag{7.21}$$



α_{23}
α_{10}
α_{18}
α_{21} α_{22}
$\alpha_1\alpha_2\alpha_3\alpha_4c(\alpha_5)\alpha_6c(\alpha_7)$
$\alpha_8\alpha_9$
α_{12}
α_{14}
α_{16}
α_{19} α_{20}
α_{11} α_{13} α_{15} α_{17}

Figure 7.11: Structure tree of the example DNA expression for the algorithm for minimality, after ten substitutions, (7.20). The dashed box encloses the part of the tree that has changed as compared to Figure 7.10.



α_{23}
α_{10}
α_{18}
α_{21} α_{22}
$\alpha_1\alpha_2\alpha_3\alpha_4c(\alpha_5)\alpha_6c(\alpha_7)\alpha_8\alpha_9$
α_{12}
α_{16}
$\alpha_{19}\alpha_{20}$
α_{11} $\alpha_{13}\alpha_{14}\alpha_{15}$ α_{17}

Figure 7.12: Structure tree of the example DNA expression for the algorithm for minimality, after eleven substitutions, (7.21). The dashed box encloses the part of the tree that has changed as compared to Figure 7.11.

as depicted in Figure 7.12. In this overall DNA expression, the DNA expression E^s that we consider has become

$$\begin{aligned} E^s = \big\langle{\downarrow}\ \big\langle{\uparrow}\ \langle{\updownarrow}\ \alpha_1\alpha_2\alpha_3\alpha_4 c(\alpha_5)\alpha_6 c(\alpha_7)\alpha_8\alpha_9\rangle\ \alpha_{10} \\ \langle{\downarrow}\ \langle{\updownarrow}\ \alpha_{11}\rangle\ \alpha_{12}\ \langle{\updownarrow}\ \alpha_{13}\alpha_{14}\alpha_{15}\rangle\ \alpha_{16}\ \langle{\updownarrow}\ \alpha_{17}\rangle\rangle\ \ \alpha_{18}\ \ \langle{\updownarrow}\ \alpha_{19}\alpha_{20}\rangle\ \big\rangle\ \big\rangle\,. \end{aligned}$$

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

$$\begin{aligned} E = \big\langle{\downarrow}\ \big\langle{\uparrow}\ \langle{\updownarrow}\ \alpha_1\alpha_2\alpha_3\alpha_4 c(\alpha_5)\alpha_6 c(\alpha_7)\alpha_8\alpha_9\rangle\ \alpha_{10} \\ \langle{\downarrow}\ \langle{\updownarrow}\ \alpha_{11}\rangle\ \alpha_{12}\ \langle{\updownarrow}\ \alpha_{13}\alpha_{14}\alpha_{15}\rangle\ \alpha_{16}\ \langle{\updownarrow}\ \alpha_{17}\rangle\rangle\ \ \alpha_{18}\ \ \langle{\updownarrow}\ \alpha_{19}\alpha_{20}\rangle\ \big\rangle \\ \langle{\uparrow}\ \langle{\downarrow}\ \alpha_{21}\rangle\rangle\ \ \langle{\uparrow}\ \langle{\updownarrow}\ \alpha_{22}\rangle\ \alpha_{23}\rangle\ \big\rangle\,. \end{aligned}$$

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 7.10, 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

$$\begin{aligned} E = \big\langle{\downarrow}\ \big\langle{\uparrow}\ \langle{\updownarrow}\ \alpha_1\alpha_2\alpha_3\alpha_4 c(\alpha_5)\alpha_6 c(\alpha_7)\alpha_8\alpha_9\rangle\ \alpha_{10} \\ \langle{\downarrow}\ \langle{\updownarrow}\ \alpha_{11}\rangle\ \alpha_{12}\ \langle{\updownarrow}\ \alpha_{13}\alpha_{14}\alpha_{15}\rangle\ \alpha_{16}\ \langle{\updownarrow}\ \alpha_{17}\rangle\rangle\ \ \alpha_{18}\ \ \langle{\updownarrow}\ \alpha_{19}\alpha_{20}\rangle\ \big\rangle \\ \langle{\downarrow}\ \alpha_{21}\rangle\ \ \langle{\uparrow}\ \langle{\updownarrow}\ \alpha_{22}\rangle\ \alpha_{23}\rangle\ \big\rangle\,. \end{aligned}$$

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

$$\begin{aligned} E = \big\langle{\downarrow}\ \big\langle{\uparrow}\ \langle{\updownarrow}\ \alpha_1\alpha_2\alpha_3\alpha_4 c(\alpha_5)\alpha_6 c(\alpha_7)\alpha_8\alpha_9\rangle\ \alpha_{10} \\ \langle{\downarrow}\ \langle{\updownarrow}\ \alpha_{11}\rangle\ \alpha_{12}\ \langle{\updownarrow}\ \alpha_{13}\alpha_{14}\alpha_{15}\rangle\ \alpha_{16}\ \langle{\updownarrow}\ \alpha_{17}\rangle\rangle\ \ \alpha_{18}\ \ \langle{\updownarrow}\ \alpha_{19}\alpha_{20}\rangle\ \big\rangle \\ \alpha_{21}\ \ \langle{\uparrow}\ \langle{\updownarrow}\ \alpha_{22}\rangle\ \alpha_{23}\rangle\ \big\rangle\,, \end{aligned} \tag{7.22}$$

as depicted in Figure 7.13. 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 its first argument and its last argument are $\uparrow$-arguments. Hence, according to line 35, we apply procedure `RotateToMinimal` to E. As we have seen in Example 7.26, E is substituted by

$$\begin{aligned} E' = \big\langle{\uparrow}\ \langle{\updownarrow}\ \alpha_1\alpha_2\alpha_3\alpha_4 c(\alpha_5)\alpha_6 c(\alpha_7)\alpha_8\alpha_9\rangle\ \alpha_{10} \\ \langle{\downarrow}\ \langle{\updownarrow}\ \alpha_{11}\rangle\ \alpha_{12}\ \langle{\updownarrow}\ \alpha_{13}\alpha_{14}\alpha_{15}\rangle\ \alpha_{16}\ \langle{\updownarrow}\ \alpha_{17}\rangle\rangle\ \ \alpha_{18} \\ \langle{\downarrow}\ \langle{\updownarrow}\ \alpha_{19}\alpha_{20}\rangle\ \alpha_{21}\ \langle{\updownarrow}\ \alpha_{22}\rangle\rangle\ \ \alpha_{23}\ \big\rangle\,. \end{aligned} \tag{7.23}$$

This is the final result E_2^* of the algorithm, and has been depicted in Figure 7.14.

We encountered E_2^* already in Example 7.11. There, it appeared to be one of the two minimal DNA expressions that are equivalent to the original DNA expression E_1^* from (7.1) and (7.15). Note, however, that in Example 7.11, we obtained the two minimal DNA expressions from the semantics of E_1^*, using the recursive construction from Theorem 5.12. Here, we have simply performed string manipulations on the DNA expression itself, as prescribed by procedure `RotateToMinimal`. ■

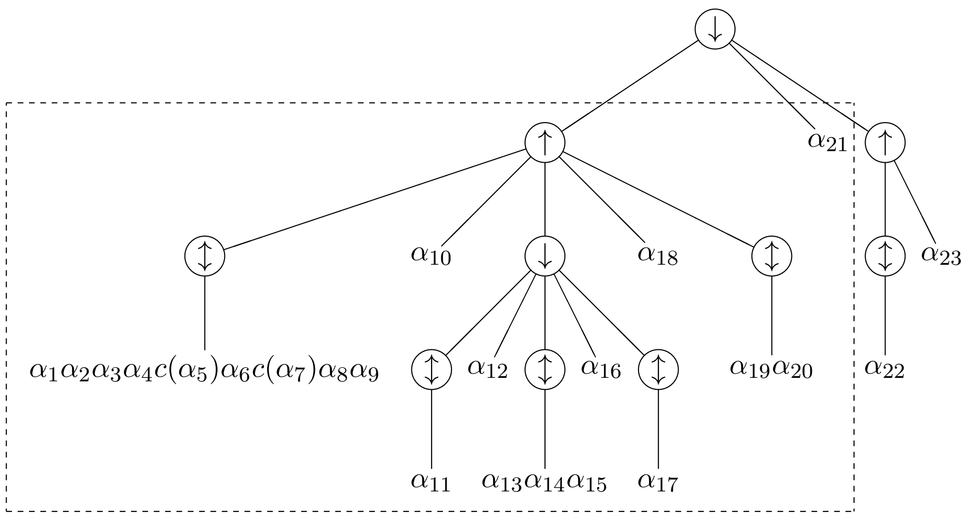


Figure 7.13: Structure tree of the example DNA expression for the algorithm for minimality, after fourteen substitutions, (7.22). The dashed box encloses the part of the tree that has changed as compared to Figure 7.12.

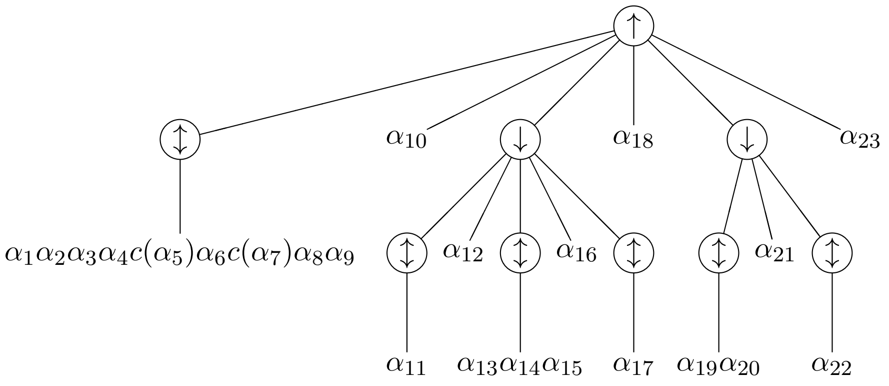


Figure 7.14: Structure tree of the example DNA expression E_2^* for the algorithm for minimality, after all fifteen substitutions, (7.23).

In the above example, when we compare the original DNA expression E_1^* and its structure tree (in Figure 7.6) to the final DNA expression E_2^* and its structure tree (in Figure 7.14), 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$\updownarrow$M.5 of procedure `Make`$\updownarrow$`ExprMinimal`. There, we substitute a $\downarrow$-expression $E_{1,i}$ by the $\updownarrow$-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 $\updownarrow$, 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$\updownarrow$M.5 of `Make`$\updownarrow$`ExprMinimal` once, in Example 7.18. There, we substituted the (small) $\downarrow$-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 7.6 and Figure 7.7 can be traced back to this substitution.

7.3 Detailed implementation and complexity of the algorithm

In § 7.1, 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 datastructure 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 7.28. 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`$\updownarrow$`ExprMinimal`, `Denickify` and `RotateToMinimal` more directly, we obtain the function in Figure 7.15.

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 7.17.

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 § 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

```
1'.   MakeMinimal ($E$)
        // recursively rewrites an arbitrary DNA expression $E$
        // into an equivalent, minimal DNA expression
2'.   {
3'.     if ($E$ is an $\updownarrow$-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 $\updownarrow$-expression)
7'.               then substitute $E$ by $E_1$;                      ($\mathcal{D}_{\mathtt{Min}}$.1)
8'.               else   // $E_1$ is an $\uparrow$-expression or a $\downarrow$-expression
9'.                    substitute $E$ by the result
                         of procedure Make$\updownarrow$ExprMinimal;         ($\mathcal{D}_{\mathtt{Min}}$.1)
10'.              fi
11'.         fi

12'.    else    // $E$ is an $\uparrow$-expression or a $\downarrow$-expression;
                // without loss of generality, assume it is
                // an $\uparrow$-expression $\langle\uparrow \varepsilon_1 \dots \varepsilon_n\rangle$ for some $n \geq 1$
                // and $\mathcal{N}$-words and DNA expressions $\varepsilon_1, \dots, \varepsilon_n$
13'.         for ($i = 1$ to $n$)
14'.         do   if ($\varepsilon_i$ 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 $\downarrow$-expression which is not alternating)
17'.                   then substitute $E_i$ in $E$ by the result
                              of procedure Denickify;                 ($\mathcal{D}_{\mathtt{Min}}$.4)
18'.                   fi

19'.                   if ($E_i$ is a $\downarrow$-expression for which the first argument
                            or the last argument is an $\uparrow$-argument)
20'.                   then substitute $E_i$ in $E$ by the result
                              of procedure RotateToMinimal;           ($\mathcal{D}_{\mathtt{Min}}$.5)
21'.                   fi

22'.                   if ($E_i$ is an $\uparrow$-expression)
23'.                   then substitute $E_i$ in $E$ by its arguments;  ($\mathcal{D}_{\mathtt{Min}}$.2)
24'.                   fi
25'.              fi
26'.         od

27'.         if ($E$ has only one argument $\varepsilon_1$)
28'.         then if ($\varepsilon_1$ is a DNA expression $E_1$)
29'.              then substitute $E$ by $E_1$;                          ($\mathcal{D}_{\mathtt{Min}}$.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 $\downarrow$-arguments)
33'.              then substitute $E$ by the result
                         of procedure RotateToMinimal;               ($\mathcal{D}_{\mathtt{Min}}$.6)
34'.              fi
35'.         fi
36'.    fi
37'.  }
```

Figure 7.15: More detailed pseudo-code of the recursive function `MakeMinimal` (cf. Figure 7.1).

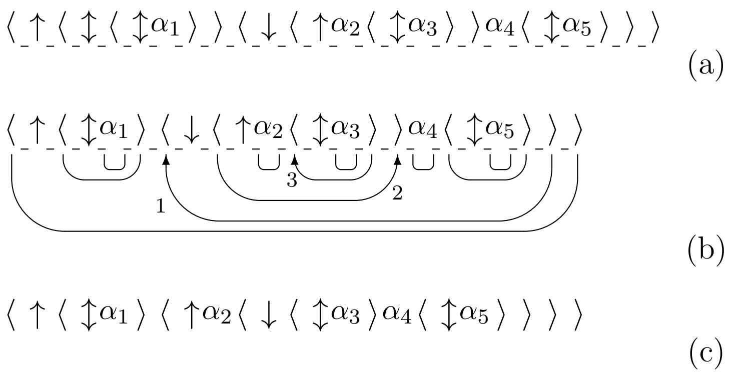


Figure 7.16: First two features of the datastructure used in the implementation of the algorithm for minimality. (a) An example DNA expression, where the letters are stored in a doubly-linked list (indicated by the dashes). (b) The result after substituting the $\updownarrow$-subexpression $\langle\updownarrow\langle\updownarrow\alpha_1\rangle\rangle$ 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.

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 datastructure to store E in. We now discuss two features of a possible, proper datastructure. 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, 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 7.16(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

$$E_i = \langle\downarrow\langle\uparrow\varepsilon_{1,1}\ldots\varepsilon_{1,m_1-1}\varepsilon_{1,m_1}\rangle\,\varepsilon_2\ldots\varepsilon_n\rangle \tag{7.24}$$

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

$$E'_i = \langle\uparrow\varepsilon_{1,1}\ldots\varepsilon_{1,m_1-1}\langle\downarrow\varepsilon_{1,m_1}\varepsilon_2\ldots\varepsilon_n\rangle\rangle \tag{7.25}$$

(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 datastructure: 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 (7.24) into the $\uparrow$-expression E'_i from (7.25). Let us use ε_1 to denote the first argument $\langle\uparrow \varepsilon_{1,1}\dots\varepsilon_{1,m_1-1}\varepsilon_{1,m_1}\rangle$ 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 7.16(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. 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`$\updownarrow$`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 datastructure 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.

In procedure `Make`$\updownarrow$`ExprMinimal`, we first substitute all $\downarrow$-arguments of the 'working DNA expression' $\widehat{E}_1$, and then substitute all $\mathcal{N}$-word-arguments. 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 7.29 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,

$$E_1 \;=\; \langle\updownarrow \alpha\alpha\rangle$$

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

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 6.12(5), Theorem 5.28 and Theorem 5.26, 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 6.14(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↕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. 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 7.30 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,

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

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,
- $$\begin{array}{lcl} \mathcal{S}(E_{2p}) & = & \underbrace{\binom{\alpha\alpha}{c(\alpha\alpha)}\binom{\alpha}{-} \dots \binom{\alpha\alpha}{c(\alpha\alpha)}\binom{\alpha}{-}}_{p \text{ times}} \binom{\alpha}{c(\alpha)} \triangle \binom{\alpha}{c(\alpha)} \\ \mathcal{S}(E_{2p+1}) & = & \underbrace{\binom{\alpha\alpha}{c(\alpha\alpha)}\binom{\alpha}{-} \dots \binom{\alpha\alpha}{c(\alpha\alpha)}\binom{\alpha}{-}}_{p \text{ times}} \binom{\alpha\alpha}{c(\alpha\alpha)} \end{array}$$
- $|E_{2p}| = 3 \cdot 4p + (3p+2) \cdot |\alpha|$ 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 6.12(5), Theorem 5.28 and Theorem 5.26, 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

$$E'_{2p} = \left\langle \uparrow \underbrace{\langle \updownarrow \alpha\alpha \rangle\, \alpha \dots \langle \updownarrow \alpha\alpha \rangle\, \alpha}_{p \text{ times}} \langle \updownarrow \alpha \rangle \langle \updownarrow \alpha \rangle \right\rangle .$$

By Lemma 6.14(1), the only minimal DNA expression denoting $\mathcal{S}(E_{2p+1})$ is

$$E'_{2p+1} = \left\langle \uparrow \underbrace{\langle \updownarrow \alpha\alpha \rangle\, \alpha \dots \langle \updownarrow \alpha\alpha \rangle\, \alpha}_{p \text{ times}} \langle \updownarrow \alpha\alpha \rangle \right\rangle .$$

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 $2p, 2p-2, 2p-4, \dots, 4$ arguments of $E'_{2(p-1)}, E'_{2(p-2)}, E'_{2(p-3)}, \dots, E'_2$, respectively. Altogether, this takes time that is quadratic in p, and thus in the length of E_{2p+1}. ■

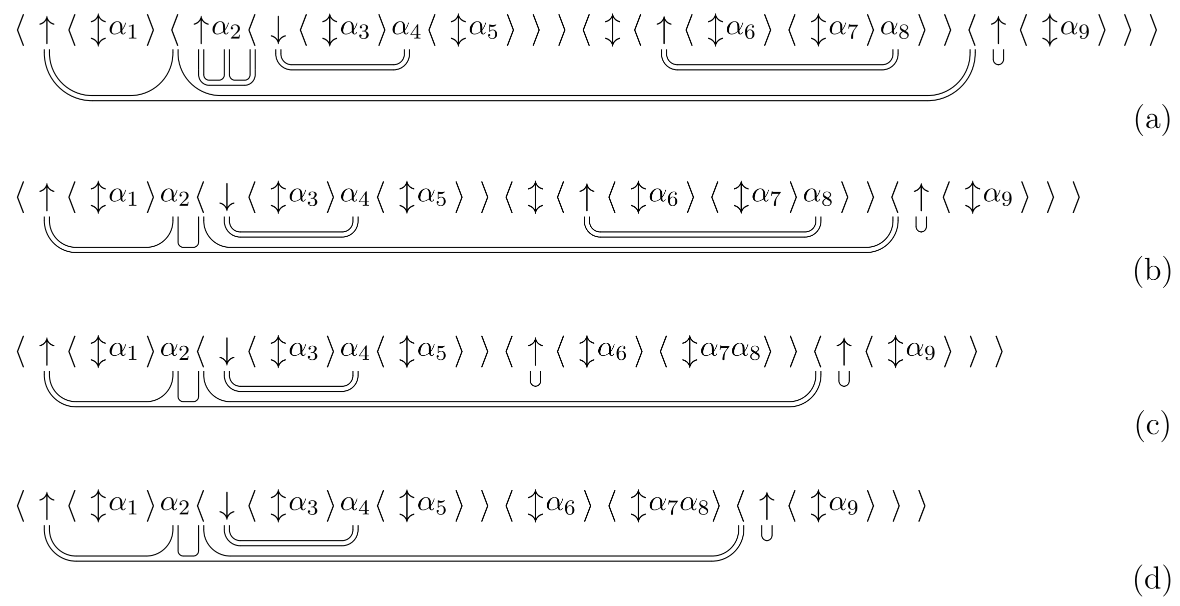


Figure 7.17: Third feature of the datastructure used in the implementation of the algorithm for minimality. Each occurrence of $\uparrow$ or $\downarrow$ has a circular, doubly-linked list of its non-$\updownarrow$-arguments. (a) The lists for an example DNA expression. Note that the list is empty for 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 datastructure, 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 datastructure. In Figure 7.17(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 7.17(b)).[4] As both lists are doubly-linked lists, we can do this in constant time.

[4]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 7.17(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.

The only exception is line $9'$ of the recursive function. There, we substitute an $\updownarrow$-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 7.17(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 7.17(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 7.31 In Example 7.29, 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↕Expr-Minimal`. 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 datastructure are indeed sufficient to execute procedure `Make↕ExprMinimal` efficiently. For this proof and a later proof, we need some additional notation:

Definition 7.32 *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 4.5, we introduced the notation $n_\updownarrow(X)$, for the number of double components of a formal DNA moleculte 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 7.33 Let E be the DNA expression E_1^* from (7.1) and let $X = \mathcal{S}(E)$ (see Figure 7.2). 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 7.34 *Let E_1^* be an arbitrary DNA expression. The total time that the function* `MakeMinimal` *applied to E_1^* spends in procedure* `Make↕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↕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↕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↕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 $n_{\mathcal{N}\uparrow\downarrow}(E^s)$. [5]

By Corollary 6.2, 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 6.3, 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

$$n_{\mathcal{N}\uparrow\downarrow}(E^s) = \sum_{\mathcal{N}\text{-words } \alpha_{1,i}} |\alpha_{1,i}| + \sum_{\downarrow\text{-expr. } E_{1,i}} n_{\mathcal{N}\uparrow\downarrow}(E_{1,i}). \tag{7.26}$$

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 6.17(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 6.17(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 maximal $\mathcal{N}$-word occurrences with parent operator $\uparrow$ or $\downarrow$). Consider an arbitrary minimal, nick free $\uparrow$-expression or $\downarrow$-expression E,

[5] 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 § 7.4.

and let $X = \mathcal{S}(E)$. By Summary 6.12, X contains at least one single-stranded component and E is constructed according to Theorem 5.12. 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 2.9, 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 $\downarrow$. 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 (7.26), this is in $\mathcal{O}(n_{\mathcal{N}\uparrow\downarrow}(E^s))$. Hence, the total time spent in procedure `Make`$\updownarrow$`Expr-Minimal` 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 $\langle\uparrow \langle\updownarrow \alpha_{1,1}\rangle \dots \langle\updownarrow \alpha_{1,m}\rangle\rangle$ for some $m \geq 2$ and $\mathcal{N}$-words $\alpha_{1,1}, \dots, \alpha_{1,m}$. 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`$\updownarrow$`ExprMinimal`. Let $E_1^s, \dots, E_r^s$ for some $r \geq 0$ be all these $\updownarrow$-subexpressions. For $h = 1, \dots, r$, we spend $\mathcal{O}(1 + n_{\mathcal{N}\uparrow\downarrow}(E_h^s))$ time in procedure `Make`$\updownarrow$`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`$\updownarrow$`ExprMinimal` to $E_1^s, \dots, E_r^s$ 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`$\updownarrow$`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`$\updownarrow$`ExprMinimal` while executing function `MakeMinimal` for E_1^* is in $\mathcal{O}(|E_1^*|)$.

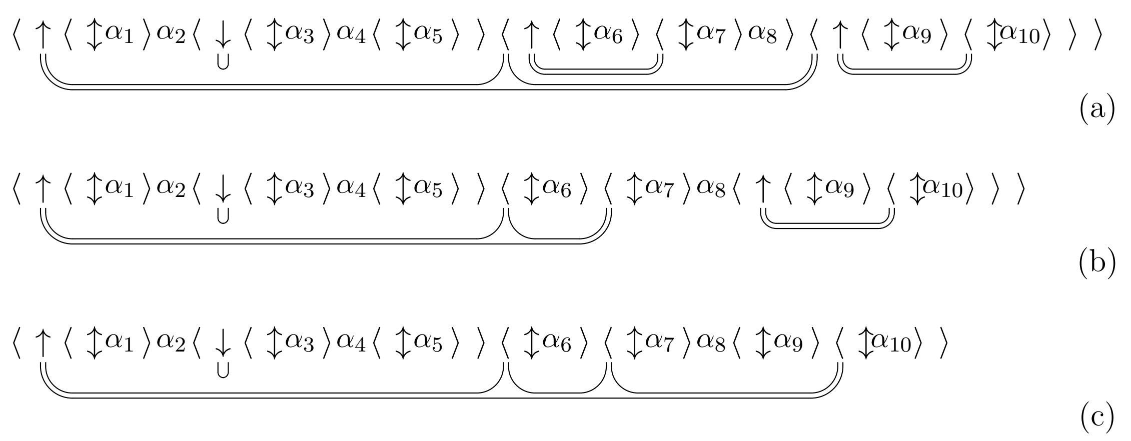

⟨ ↑ ⟨ ↕α1 ⟩α2 ⟨ ↓ ⟨ ↕α3 ⟩α4 ⟨ ↕α5 ⟩ ⟩ ⟨ ↑ ⟨ ↕α6 ⟩ ⟨ ↕α7 ⟩α8 ⟩ ⟨ ↑ ⟨ ↕α9 ⟩ ⟨ ↕α10⟩ ⟩ ⟩
U
(a)
⟨ ↑ ⟨ ↕α1 ⟩α2 ⟨ ↓ ⟨ ↕α3 ⟩α4 ⟨ ↕α5 ⟩ ⟩ ⟨ ↕α6 ⟩ ⟨ ↕α7 ⟩α8 ⟨ ↑ ⟨ ↕α9 ⟩ ⟨ ↕α10⟩ ⟩ ⟩
U
(b)
⟨ ↑ ⟨ ↕α1 ⟩α2 ⟨ ↓ ⟨ ↕α3 ⟩α4 ⟨ ↕α5 ⟩ ⟩ ⟨ ↕α6 ⟩ ⟨ ↕α7 ⟩α8 ⟨ ↕α9 ⟩ ⟨ ↕α10⟩ ⟩
U
(c)

Figure 7.18: Fourth feature of the datastructure 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.

This completes the proof of Lemma 7.34. □

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 datastructure 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 position of the first letter of $\widehat{\varepsilon}_j$ (which is an opening bracket). In Figure 7.18(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 $E_i = \langle\uparrow_1 \varepsilon_{i,1} \dots \varepsilon_{i,n_i}\rangle$ for some $n_i \geq 1$ and $\mathcal{N}$-words and DNA expressions $\varepsilon_{i,1}, \dots \varepsilon_{i,n_i}$. 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 7.18(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 7.24(3), the result of the procedure is completely independent of this order. In Figure 7.18(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 7.27, 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 7.35 In Example 7.30, 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 7.36 *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 substition 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.[6] 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 $\mathcal{O}(|E_1^*|)$. $\square$

By now, we know the time requirements of procedures `Make`$\updownarrow$`ExprMinimal` and `Denickify`. There is one more point we like to make before we determine the total time complexity of the function `MakeMinimal`.

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 pass 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`$\updownarrow$`ExprMinimal` and `Denickify`. The fact that we

[6]If in procedure `Make`$\updownarrow$`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 $\langle\updownarrow \alpha_{1,1}\alpha_{1,2}\alpha_{1,3}\rangle$.

ignored it there, does not mean that Lemma 7.34 and Lemma 7.36 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`.

Theorem 7.37 *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 $T_{\mathrm{CM}}(E)$ to denote the time required by `MakeMinimal` for E, except the time spent in procedures `Make`$\updownarrow$`ExprMinimal` and `Denickify`. We prove that $T_{\mathrm{CM}}(E)$ is in $\mathcal{O}(|E|)$. Then the claim follows from Lemma 7.34 and Lemma 7.36.

To analyse $T_{\mathrm{CM}}(E)$, 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 datastructure (on pages 117 and 127, respectively) and from the observation about passing the parameter for a (recursive) call of `MakeMinimal` (on page 128), 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 $T_{\mathrm{CM}}(E) \leq c^* \cdot |E| - c_3$. Here, we subtract c_3, to be prepared for the additional constant time required for every argument of an $\uparrow$-expression E. [7] We will come back to this later. Although 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 $T_{\mathrm{CM}}(E) \leq c_1$. Clearly, $|E| \geq 4$. We now distinguish two (overlapping) subcases. If $c_1 \geq 3c_3$, then

  $$T_{\mathrm{CM}}(E) \leq c_1 = c_1 + c_3 - c_3 \leq \frac{4}{3}c_1 - c_3 \leq 4c^* - c_3 \leq c^* \cdot |E| - c_3,$$

  where the third inequality follows from $c^* \geq \frac{c_1}{3}$. If, on the other hand, $c_1 \leq 3c_3$, then

  $$T_{\mathrm{CM}}(E) \leq c_1 \leq 3c_3 = 4c_3 - c_3 \leq 4c^* - c_3 \leq c^* \cdot |E| - c_3,$$

  where the third inequality follows from $c^* \geq c_3$.

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

  $$T_{\mathrm{CM}}(E) \quad \leq \quad c_2 + n \cdot c_3 \leq 3c^* - c_3 + n \cdot c^* \leq c^* \cdot |E| - c_3,$$

  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 $T_{\mathrm{CM}}(E) \leq c^* \cdot |E| - c_3$ 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, $T_{\mathrm{CM}}(E) \leq c_1 + T_{\mathrm{CM}}(E_1)$. Because E_1 contains p operators, we can apply the induction hypothesis to it:

  $$\begin{aligned} T_{\mathrm{CM}}(E) &\leq c_1 + T_{\mathrm{CM}}(E_1) \leq c_1 + c^* \cdot |E_1| - c_3 \\ &\leq c^* \cdot (|E_1| + 3) - c_3 = c^* \cdot |E| - c_3, \end{aligned}$$

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

  If E is an $\uparrow$-expression, then $E = \langle \uparrow \varepsilon_1 \dots \varepsilon_n \rangle$ for some $n \geq 1$ and $\mathcal{N}$-words and DNA expressions $\varepsilon_1, \dots, \varepsilon_n$. We examine the time spent in `MakeMinimal` on an

[7] 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.

argument ε_i with $1 \le i \le n$. If ε_i is an $\mathcal{N}$-word α_i, then the time spent on this argument is bounded by

$$c_3 \le c^* \le c^* \cdot |\alpha_i|,$$

because obviously $|\alpha_i| \ge 1$. 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 $c_3 + T_{\mathrm{CM}}(E_i)$. 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

$$c_3 + T_{\mathrm{CM}}(E_i) \le c_3 + c^* \cdot |E_i| - c_3 = c^* \cdot |E_i|.$$

Note that here we benefit from the term $-c_3$ in the upper bound for $T_{\mathrm{CM}}(E)$.

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

$$\begin{array}{rcl} T_{\mathrm{CM}}(E) & \le & c_2 + c^* \cdot (|\varepsilon_1| + \cdots |\varepsilon_n|) \\ & \le & 3c^* - c_3 + c^* \cdot (|\varepsilon_1| + \cdots |\varepsilon_n|) = c^* \cdot |E| - c_3, \end{array}$$

where the second inequality follows from $c^* \ge \frac{c_2+c_3}{3}$.

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 7.37, we obtain the following result:

Corollary 7.38 *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 datastructure 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 datastructure, 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 7.39 Let α be an arbitrary $\mathcal{N}$-word, and let E_p be defined by

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

It is easy to see that for any $p \ge 1$, 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 $p \ge 1$,

- E_p contains $p + 1$ pairs of matching brackets. Hence, the second feature of the datastructure 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 datastructure 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 datastructure 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 datastructure 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 datastructure 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 7.37 (on page 128), 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 7.40 *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.

7.4 Decrease of length by the algorithm

In the previous section, we proved that the total time required by the function `Make-Minimal` 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 datastructure, which can be initialized in linear time. We proved that with this datastructure, 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 datastructure. 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 2.13, 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 7.34, we observed that in procedure `Make`$\updownarrow$`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 "$c(\alpha)$". [8] 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`$\updownarrow$`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 the observation in the proof of Theorem 7.20(2), 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 7.41 *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`$\updownarrow$`ExprMinimal`.

[8] In this report, 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 13).

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 4.1, 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 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'

  $$\left\langle \updownarrow \widehat{E}_1 \right\rangle = \langle \updownarrow \langle \uparrow \langle \updownarrow \alpha_{1,1} \rangle \ldots \langle \updownarrow \alpha_{1,m} \rangle \rangle \rangle$$

  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 $\langle \updownarrow \alpha_{E_{1,i}} \rangle$. In the proof of Lemma 7.34, 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 $\langle \updownarrow \alpha_{E_{1,i}} \rangle$. 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 7.34, $p \leq n_\alpha(E_{1,i})$. On the other hand, let $X_{1,i} = \mathcal{S}(E_{1,i})$. By Corollary 6.13(2),

  $$p = 1 + B_\uparrow(X_{1,i}) + n_\updownarrow(X_{1,i}) \geq 1 + n_\updownarrow(X_{1,i}). \tag{7.27}$$

  We also observed in the proof of Lemma 7.34 that

  $$\begin{aligned} n_{\alpha\uparrow\downarrow}(E_{1,i}) &= n_{\uparrow\downarrow}(X_{1,i}), &(7.28)\\ n_{\alpha\updownarrow}(E_{1,i}) &= n_\updownarrow(X_{1,i}) &(7.29)\end{aligned}$$

  Now by Corollary 2.9, double components and single-stranded components alternate in the nick free formal DNA molecule $X_{1,i}$. By Lemma 6.17(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 (7.27)–(7.29), 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 $\langle \updownarrow \alpha_{E_{1,i}} \rangle$, 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$\updownarrow$M.8–M$\updownarrow$M.18 and the iteration of the for-loops in lines M$\updownarrow$M.4 (given that we do not have a list of $\downarrow$-arguments of E_1, but only a list of non-$\updownarrow$-arguments) and M$\updownarrow$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$\updownarrow$M.8–M$\updownarrow$M.17. At that point in the procedure, the only other arguments of the 'working $\uparrow$-subexpression' $\widehat{E}_1$ are $\updownarrow$-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 $\updownarrow$-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 $\updownarrow$-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 $\updownarrow$-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 $\updownarrow$-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

$$\begin{aligned} &c_1 + c_2 \cdot \sum_{\downarrow\text{-arg } E_{1,i}} n_\alpha(E_{1,i}) + c_3 \cdot k \\ &\qquad\qquad \leq T_{\mathrm{M}\updownarrow\mathrm{M}}(E) \leq d_1 + d_2 \cdot \sum_{\downarrow\text{-arg } E_{1,i}} n_\alpha(E_{1,i}) + d_3 \cdot k, \quad (7.30) \end{aligned}$$

where k is (again) the number of $\mathcal{N}$-word-arguments of the $\uparrow$-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 $\uparrow$-expression E_1 has at least one expression-argument and $k \leq 2$ $\mathcal{N}$-word-arguments, then

$$1 + \frac{1}{2} \sum_{\downarrow\text{-arg } E_{1,i}} n_\alpha(E_{1,i}) + 0 \leq \delta(E) \leq 2 + \sum_{\downarrow\text{-arg } E_{1,i}} (n_\alpha(E_{1,i}) - 1) + k. \quad (7.31)$$

Finally, if the $\uparrow$-expression E_1 has at least one expression-argument and $k \geq 3$ $\mathcal{N}$-word-arguments,[9] 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 $\downarrow$-arguments $E_{1,i}$ at all. Hence, the value $\delta(E) = 1$ also satisfies (7.31).

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 (7.30). If, on the other hand, $k \geq 3$, then

$$\begin{aligned}
\delta(E) &\geq 1 + \frac{1}{2} \sum_{\downarrow\text{-arg } E_{1,i}} n_\alpha(E_{1,i}) + k - 2 \\
&\geq d^* \cdot \left(d_1 + 2d_3 + d_2 \cdot \sum_{\downarrow\text{-arg } E_{1,i}} n_\alpha(E_{1,i}) + d_3 \cdot (k-2) \right) \geq d^* \cdot T_{\mathrm{M}\updownarrow\mathrm{M}}(E),
\end{aligned}$$

[9]Because the $\mathcal{N}$-word-arguments are maximal $\mathcal{N}$-word occurrences, E_1 actually has at least $k-1 \geq 2$ expression-arguments.

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 (7.30).

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 (7.30).

We can conclude that $T_{\mathrm{M}\updownarrow\mathrm{M}}(E)$ (the time spent in procedure `Make`$\updownarrow$`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`$\updownarrow$`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 7.36, $T_{\mathrm{Dni}}(E_i)$ 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, 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 4.1, $T_{\text{Dni}}(E_i)$ 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 7.41. □

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 7.41 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 7.41 yields an alternative proof for Theorem 7.12.

This conclusion does not depend on the assumptions in Theorem 7.41, 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 7.41, 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.

Chapter 8

A Minimal Normal Form for DNA Expressions

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. It can be shown that this approach takes time that is linear in the length of E_1 and E_2. This is certainly efficient.

There is also another approach, which is based on a *normal form*. A normal form is a set of properties, such that for each DNA expression there is exactly one equivalent DNA expression with these properties. If we can find the normal form versions of E_1 and E_2, then it is easy to decide if E_1 and E_2 are equivalent: this is the case, if and only if their normal form versions are the same. This alternative approach is more elegant, because it operates at the level of DNA expressions only. It does not refer to the semantics, at all.

To implement this approach, we first have to decide what our 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 6.12). We also have a useful characterization of minimal DNA expressions in general (see Lemma 6.15 and Theorem 6.16). 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. 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.

8.1 Definition of the minimal normal form

As we have seen in Chapter 5 and Chapter 6 (and especially in § 6.4), 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 6.12. 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 $E_{\text{MinNF}}(X)$ denotes the normal form DNA expression for a formal DNA molecule X (cf. Summary 6.12):

Definition 8.1 *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 5.12(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 5.12(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 5.12(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 5.12(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 5.26. $E_{MinNF}(X)$ is the minimal $\uparrow$-expression denoting X based on $E_1, \ldots, E_m$, as described in Theorem 5.28.*

6. *If X contains at least one upper nick letter, then let $Z_1{}^{\triangledown}Z_2{}^{\triangledown}\dots{}^{\triangledown}Z_m$ for some $m \geq 2$ be the nick free decomposition of X. For $h = 1,\dots,m$, 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 5.26. $E_{MinNF}(X)$ is the minimal $\downarrow$-expression denoting X based on $E_1,\dots,E_m$, analogous to the description in Theorem 5.28.*

Example 8.2 Consider the nick free formal DNA molecule X from Figure 5.3, for which $B_{\uparrow}(X) = 4$ and $B_{\downarrow}(X) = 3$. In Example 5.13, we have used Theorem 5.12 to construct two minimal DNA expressions denoting X. They were based on the lower block partitionings of X shown in Figure 5.2(a3) and (a4). By Case 3 of Definition 8.1, $E_{\mathrm{MinNF}}(X)$ 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 5.2(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 5.12(2). By Lemma 5.10, 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,

$$\begin{aligned} E_{\mathrm{MinNF}}(X) &= \langle\uparrow\ \alpha_1\ E_1\ \alpha_7\ E_2\ \alpha_{11}\, \langle\updownarrow \alpha_{12}\rangle\, \alpha_{13}\ E_3\ \alpha_{17}\, \langle\updownarrow \alpha_{18}\rangle\ \rangle \\ &= \langle\uparrow\ \alpha_1\ \ \langle\downarrow \langle\updownarrow \alpha_2\rangle\, \alpha_3\, \langle\updownarrow \alpha_4\rangle\, \alpha_5\, \langle\updownarrow \alpha_6\rangle\rangle\ \ \alpha_7\ \ \langle\downarrow \langle\updownarrow \alpha_8\rangle\, \alpha_9\, \langle\updownarrow \alpha_{10}\rangle\rangle \\ &\qquad \alpha_{11}\, \langle\updownarrow \alpha_{12}\rangle\, \alpha_{13}\ \ \langle\downarrow \langle\updownarrow \alpha_{14}\rangle\, \alpha_{15}\, \langle\updownarrow \alpha_{16}\rangle\rangle\ \ \alpha_{17}\, \langle\updownarrow \alpha_{18}\rangle\ \rangle\,. \end{aligned} \tag{8.1}$$

■

Example 8.3 Consider the nick free formal DNA molecule X from Figure 5.4, for which $B_{\uparrow}(X) = B_{\downarrow}(X) = 2$ and whose first single-stranded component is an upper component. In Example 5.14 we have used Theorem 5.12 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 8.1, $E_{\mathrm{MinNF}}(X)$ is the $\uparrow$-expression based on the primitive lower block partitioning of X. This partitioning is depicted in Figure 5.4(a). Hence,

$$\begin{aligned} E_{\mathrm{MinNF}}(X) &= E_a \\ &= \ \langle\uparrow \alpha_1\ \ \langle\downarrow \langle\updownarrow \alpha_2\rangle\, \alpha_3\, \langle\updownarrow \alpha_4\rangle\rangle\ \ \alpha_5\, \langle\updownarrow \alpha_6\rangle\, \alpha_7\ \ \langle\downarrow \langle\updownarrow \alpha_8\rangle\, \alpha_9\, \langle\updownarrow \alpha_{10}\rangle\rangle\rangle\,. \end{aligned} \tag{8.2}$$

(see (5.8)). ■

Example 8.4 Consider the formal DNA molecule X from Figure 5.5, which contains four lower nick letters. The nick free decomposition of X is $Z_1{}_{\vartriangle}Z_2{}_{\vartriangle}Z_3{}_{\vartriangle}Z_4{}_{\vartriangle}Z_5$ for the submolecules $Z_1,\dots,Z_5$ from (5.12).

In Example 5.29, 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 (5.19):

$$\begin{aligned} E_{\mathrm{MinNF}}(X) = \langle\uparrow\ & \alpha_1 \langle\downarrow \langle\updownarrow \alpha_2\rangle\, \alpha_3 \langle\updownarrow \alpha_4\rangle\rangle \\ & \langle\downarrow \langle\updownarrow \alpha_5\rangle\, \alpha_6 \langle\updownarrow \alpha_7\rangle\rangle\, \alpha_8 \langle\downarrow \langle\updownarrow \alpha_9\rangle\, \alpha_{10} \langle\updownarrow \alpha_{11}\rangle\rangle \\ & \langle\updownarrow \alpha_{12}\rangle\, \alpha_{13} \langle\updownarrow \alpha_{14}\rangle\, \alpha_{15} \langle\updownarrow \alpha_{16}\rangle \quad \langle\updownarrow \alpha_{17}\rangle \\ & \langle\downarrow \langle\updownarrow \alpha_{18}\rangle\, \alpha_{19} \langle\updownarrow \alpha_{20}\rangle\rangle\, \alpha_{21} \langle\updownarrow \alpha_{22}\rangle \quad \rangle\,. \end{aligned} \tag{8.3}$$

■

At several places, Definition 8.1 refers to the construction of (operator-)minimal DNA expressions, as described in Theorem 5.12 and Theorem 5.26. These constructions involve minimal DNA expressions E_j denoting lower blocks or upper blocks $\overline{X}_j$. In Definition 8.1, 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 5.12 and Theorem 5.26, 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 6.14(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 5.3, 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 8.5 *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.*

8.2 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 8.1, and finally comparing E and E'. In this section we will describe a more elegant way to achieve the same goal.

In § 6.2, we have given a characterization of minimal DNA expressions by six simple properties. This characterization makes it easy to decide whether or not a given DNA expression is minimal. We now do something similar for DNA expressions is 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 8.6 *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 ($\mathcal{D}_{\mathrm{MinNF}}$.1), ($\mathcal{D}_{\mathrm{MinNF}}$.2) and ($\mathcal{D}_{\mathrm{MinNF}}$.3) are equal to Properties ($\mathcal{D}_{\mathrm{Min}}$.1), ($\mathcal{D}_{\mathrm{Min}}$.2) and ($\mathcal{D}_{\mathrm{Min}}$.3) of minimal DNA expressions in general.

Property ($\mathcal{D}_{\mathrm{MinNF}}$.4) includes Properties ($\mathcal{D}_{\mathrm{Min}}$.4) and ($\mathcal{D}_{\mathrm{Min}}$.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 ($\mathcal{D}_{\mathrm{MinNF}}$.5) is a stronger version of Property ($\mathcal{D}_{\mathrm{Min}}$.6). We will see in the proof that the difference between the two properties is caused by the second choice 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 (8.1), (8.2) and (8.3) have all five properties. In Table 8.1, 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 8.6: Let $X = \mathcal{S}(E)$, i.e., X is the formal DNA molecule for which $E = E_{\mathrm{MinNF}}(X)$.

By Lemma 8.5(2), E is minimal. Hence, Properties ($\mathcal{D}_{\mathrm{MinNF}}$.1), ($\mathcal{D}_{\mathrm{MinNF}}$.2) and

Properties	E $\quad X = \mathcal{S}(E) \quad E_{\text{MinNF}}(X)$
$(\mathcal{D}_{\text{MinNF}}.4)$	$\langle\downarrow \langle\uparrow \alpha_1 \langle\downarrow \langle\updownarrow \alpha_2\rangle \alpha_3 \langle\updownarrow \alpha_4\rangle\rangle \alpha_5 \langle\updownarrow \alpha_6\rangle\rangle \langle\updownarrow \alpha_7\rangle\rangle$ $\binom{\alpha_1}{-}\binom{\alpha_2}{c(\alpha_2)}\binom{-}{\alpha_3}\binom{\alpha_4}{c(\alpha_4)}\binom{\alpha_5}{-}\binom{\alpha_6}{c(\alpha_6)}^{\triangledown}\binom{\alpha_7}{c(\alpha_7)}$ $\langle\downarrow \langle\uparrow \alpha_1 \langle\updownarrow \alpha_2\rangle\rangle \alpha_3 \langle\uparrow \langle\updownarrow \alpha_4\rangle \alpha_5 \langle\updownarrow \alpha_6\rangle\rangle \langle\updownarrow \alpha_7\rangle\rangle$
$(\mathcal{D}_{\text{MinNF}}.4)$	$\langle\uparrow \alpha_1 \langle\downarrow \langle\updownarrow \alpha_2\rangle \alpha_3 \langle\uparrow \langle\updownarrow \alpha_4\rangle \alpha_5 \langle\updownarrow \alpha_6\rangle\rangle \alpha_7\rangle\rangle$ $\binom{\alpha_1}{-}\binom{\alpha_2}{c(\alpha_2)}\binom{-}{\alpha_3}\binom{\alpha_4}{c(\alpha_4)}\binom{\alpha_5}{-}\binom{\alpha_6}{c(\alpha_6)}\binom{-}{\alpha_7}$ $\langle\uparrow \alpha_1 \langle\downarrow \langle\updownarrow \alpha_2\rangle \alpha_3 \langle\updownarrow \alpha_4\rangle\rangle \alpha_5 \langle\downarrow \langle\updownarrow \alpha_6\rangle \alpha_7\rangle\rangle$
$(\mathcal{D}_{\text{MinNF}}.5)$	$\langle\downarrow \langle\uparrow \alpha_1 \langle\updownarrow \alpha_2\rangle\rangle \alpha_3\rangle$ $\binom{\alpha_1}{-}\binom{\alpha_2}{c(\alpha_2)}\binom{-}{\alpha_3}$ $\langle\uparrow \alpha_1 \langle\downarrow \langle\updownarrow \alpha_2\rangle \alpha_3\rangle\rangle$
$(\mathcal{D}_{\text{MinNF}}.5)$	$\langle\downarrow \langle\uparrow \alpha_1 \langle\updownarrow \alpha_2\rangle\rangle \alpha_3 \langle\uparrow \langle\updownarrow \alpha_4\rangle \alpha_5 \langle\updownarrow \alpha_6\rangle\rangle \alpha_7\rangle$ $\binom{\alpha_1}{-}\binom{\alpha_2}{c(\alpha_2)}\binom{-}{\alpha_3}\binom{\alpha_4}{c(\alpha_4)}\binom{\alpha_5}{-}\binom{\alpha_6}{c(\alpha_6)}\binom{-}{\alpha_7}$ $\langle\uparrow \alpha_1 \langle\downarrow \langle\updownarrow \alpha_2\rangle \alpha_3 \langle\updownarrow \alpha_4\rangle\rangle \alpha_5 \langle\downarrow \langle\updownarrow \alpha_6\rangle \alpha_7\rangle\rangle$
$(\mathcal{D}_{\text{MinNF}}.4)$, $(\mathcal{D}_{\text{MinNF}}.5)$	$\langle\downarrow \langle\uparrow \alpha_1 \langle\downarrow \langle\updownarrow \alpha_2\rangle \alpha_3 \langle\updownarrow \alpha_4\rangle\rangle \alpha_5 \langle\updownarrow \alpha_6\rangle\rangle \alpha_7\rangle$ $\binom{\alpha_1}{-}\binom{\alpha_2}{c(\alpha_2)}\binom{-}{\alpha_3}\binom{\alpha_4}{c(\alpha_4)}\binom{\alpha_5}{-}\binom{\alpha_6}{c(\alpha_6)}\binom{-}{\alpha_7}$ $\langle\uparrow \alpha_1 \langle\downarrow \langle\updownarrow \alpha_2\rangle \alpha_3 \langle\updownarrow \alpha_4\rangle\rangle \alpha_5 \langle\downarrow \langle\updownarrow \alpha_6\rangle \alpha_7\rangle\rangle$
$(\mathcal{D}_{\text{MinNF}}.4)$, $(\mathcal{D}_{\text{MinNF}}.5)$	$\langle\downarrow \langle\uparrow \alpha_1 \langle\downarrow \langle\updownarrow \alpha_2\rangle \alpha_3 \langle\uparrow \langle\updownarrow \alpha_4\rangle \alpha_5 \langle\updownarrow \alpha_6\rangle\rangle \alpha_7 \langle\updownarrow \alpha_8\rangle\rangle \alpha_9 \langle\updownarrow \alpha_{10}\rangle\rangle \alpha_{11}\rangle$ $\binom{\alpha_1}{-}\binom{\alpha_2}{c(\alpha_2)}\binom{-}{\alpha_3}\binom{\alpha_4}{c(\alpha_4)}\binom{\alpha_5}{-}\binom{\alpha_6}{c(\alpha_6)}\binom{-}{\alpha_7}\binom{\alpha_8}{c(\alpha_8)}\binom{\alpha_9}{-}\binom{\alpha_{10}}{c(\alpha_{10})}\binom{-}{\alpha_{11}}$ $\langle\uparrow \alpha_1 \langle\downarrow \langle\updownarrow \alpha_2\rangle \alpha_3 \langle\updownarrow \alpha_4\rangle\rangle \alpha_5 \langle\downarrow \langle\updownarrow \alpha_6\rangle \alpha_7 \langle\updownarrow \alpha_8\rangle\rangle \alpha_9 \langle\downarrow \langle\updownarrow \alpha_{10}\rangle \alpha_{11}\rangle\rangle$

Table 8.1: Examples of minimal DNA expressions which do not have all properties from Lemma 8.6. 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 $E_{\text{MinNF}}(X)$. As usual, the α_i's occurring represent (arbitrary) $\mathcal{N}$-words.
The DNA expressions in the third case are the ones from Example 5.2. The second, the fourth and the fifth case deal with the same formal DNA molecule, which is similar to the molecule from Example 5.14 (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.

$(\mathcal{D}_{\text{MinNF}}.3)$ are valid for E, simply because, by Lemma 6.15, they are valid for any minimal DNA expression. The other two properties require some specific considerations.

$(\mathcal{D}_{\text{MinNF}}.4)$ If E is an $\updownarrow$-expression, then by Property $(\mathcal{D}_{\text{MinNF}}.1)$, $E = \langle\updownarrow \alpha\rangle$ 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 $(\mathcal{D}_{\text{MinNF}}.4)$.

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 8.1 is applicable to $E = E_{\text{MinNF}}(X)$. In both cases, E is based on the primitive lower block partitioning $\mathcal{P}$ of X, as described in Theorem 5.12(1).

By the construction from Theorem 5.12(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 8.5, the lower blocks $\overline{X}_j$ occurring in $\mathcal{P}$ are primitive lower blocks of X. Hence, by Lemma 6.14(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 8.1 is applicable to $E = E_{\text{MinNF}}(X)$. 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 5.28. 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 5.26. Because the constructions from Theorem 5.12(1) and Theorem 5.26 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 $E = E_{\text{MinNF}}(X)$ has Property ($\mathcal{D}_{\text{MinNF}}$.4).

The proof for the case that E is a $\downarrow$-expression is analogous.

($\mathcal{D}_{\text{MinNF}}$.5) Assume that the outermost operator of E is $\uparrow$. Then in particular, by Theorem 5.3, X is not double-complete.

Assume further that E does not have two consecutive expression-arguments. By Property ($\mathcal{D}_{\text{MinNF}}$.4) and Lemma 3.5, X is nick free. Hence, either Case 2a or Case 3 of Definition 8.1 is applicable to $E = E_{\text{MinNF}}(X)$. In both cases, E is based on the primitive lower block partitioning of X, as described in Theorem 5.12(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 4.6(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 5.6(3), the maximal upper prefix Y_0 of X is not empty, Hence, in the construction from Theorem 5.12(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 6.15).

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 8.7 *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 6.15 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 6.16, 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 5.3, $E = \langle \updownarrow \alpha_1 \rangle$. Indeed, $E = E_{\text{MinNF}}(X)$ (see Case 1 of Definition 8.1).

2. If X is nick free, contains at least one single-stranded component and $B_{\uparrow}(X) = B_{\downarrow}(X)$, then by Summary 6.12(2), E is either an $\uparrow$-expression based on a lower block partitioning of X as described in Theorem 5.12(1), or a $\downarrow$-expression based on an upper block partitioning of X as described in Theorem 5.12(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 4.6(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 5.12.

   • By Lemma 5.15, 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 5.12(1). Let $\mathcal{P} = Y_0\overline{X}_1Y_1 \dots \overline{X}_rY_r$ for some $r \geq 0$ be the lower block partitioning that E is based on. By construction and by Corollary 5.17(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 5.7(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 $E = E_{\mathrm{MinNF}}(X)$ (see Case 2a of Definition 8.1).

(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 5.12(2). Hence, also in this case, $E = E_{\mathrm{MinNF}}(X)$ (see Case 2b of Definition 8.1).

3. If X is nick free and $B_{\uparrow}(X) > B_{\downarrow}(X)$, then by Summary 6.12(3), E is an $\uparrow$-expression which is based on a lower block partitioning of X, as described in Theorem 5.12(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 $E = E_{\mathrm{MinNF}}(X)$ (see Case 3 of Definition 8.1).

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, $E = E_{\mathrm{MinNF}}(X)$ (see Case 4 of Definition 8.1).

5. If X contains at least one lower nick letter, then let $Z_1{}_\triangle Z_2{}_\triangle \dots {}_\triangle Z_m$ for some $m \geq 2$ be the nick free decomposition of X. By Summary 6.12(5), E is an $\uparrow$-expression which is based on operator-minimal $\uparrow$-expressions $E_1, \dots, E_m$ denoting $Z_1, \dots, Z_m$, respectively, as described in Theorem 5.28. For $h = 1, \dots, m$, E_h is in turn based on a lower block partitioning $\mathcal{P}_h$ of Z_h, as described in Theorem 5.26.

   Because the arguments of E are precisely the arguments of $E_1, \dots, E_m$, and the constructions from Theorem 5.12(1) and Theorem 5.26 are in fact identical, we can proceed in the same way as in (the second part of) the proof for Case 2a: for $h = 1, \dots, m$, 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 $h = 1, \dots, m$, 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 8.1).

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 8.1).

□

When we combine Lemma 8.6 and Lemma 8.7, we obtain

Theorem 8.8 *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 8.6 to prove other properties of normal form DNA expressions.

Lemma 8.9 *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 ($\mathcal{D}_{\text{MinNF}}$.1) and ($\mathcal{D}_{\text{MinNF}}$.2), the parent operator of E_1 is not $\updownarrow$ or $\uparrow$. Hence, it must be $\downarrow$. This, however, contradicts Property ($\mathcal{D}_{\text{MinNF}}$.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 ($\mathcal{D}_{\text{MinNF}}$.1) and ($\mathcal{D}_{\text{MinNF}}$.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.

$\square$

In general, the nesting level of a DNA expression can get arbitrarily high. 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. (cf. the proof of Lemma 2.16). As we have seen in the proof of Lemma 5.20, even the nesting level of a minimal DNA expression can get arbitrarily high. For DNA expressions in minimal normal form, however, the nesting level is limited:

Lemma 8.10 *Let E be a DNA expression in minimal normal form. The maximal nesting level of E is at most 3.*

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 2.14(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. $\square$

8.3 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 § 2.8 (see § 6.3). 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 8.8 (and Lemma 8.6) t is in minimal normal form if and only if

($\mathcal{D}_{\mathbf{MinNF}}$.1) each node labelled by $\updownarrow$ in t has a (single) child labelled by an $\mathcal{N}$-word α, and

($\mathcal{D}_{\mathbf{MinNF}}$.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

($\mathcal{D}_{\mathbf{MinNF}}$.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

($\mathcal{D}_{\mathbf{MinNF}}$.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

($\mathcal{D}_{\mathbf{MinNF}}$.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 8.9 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 8.10 and Lemma 2.22 If t is in minimal normal form, then the height of t is at most 4.

As we observed in Example 8.4, the minimal DNA expression from (5.19) is in minimal normal form. Hence, the corresponding structure tree, which we have shown in Figure 6.1(c), is also in minimal normal form. Indeed, it has all properties listed above.

On the other hand, although the DNA expression E from (5.6) is minimal, it is not in minimal normal form. Consequently, the corresponding structure tree in Figure 6.1(a) is not in minimal normal form, either. It does not satisfy (the tree-version of) Property ($\mathcal{D}_{\mathrm{MinNF}}$.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 (5.11) and the corresponding structure tree in Figure 6.1(b) are not in minimal normal form. They violate both Property ($\mathcal{D}_{\mathrm{MinNF}}$.4) and Property ($\mathcal{D}_{\mathrm{MinNF}}$.5). Again, the height of the tree is greater than 4.

In Example 8.2 and Example 8.3, we have given the DNA expressions in minimal normal form for the last two cases. The corresponding structure trees are depicted in Figure 8.1.

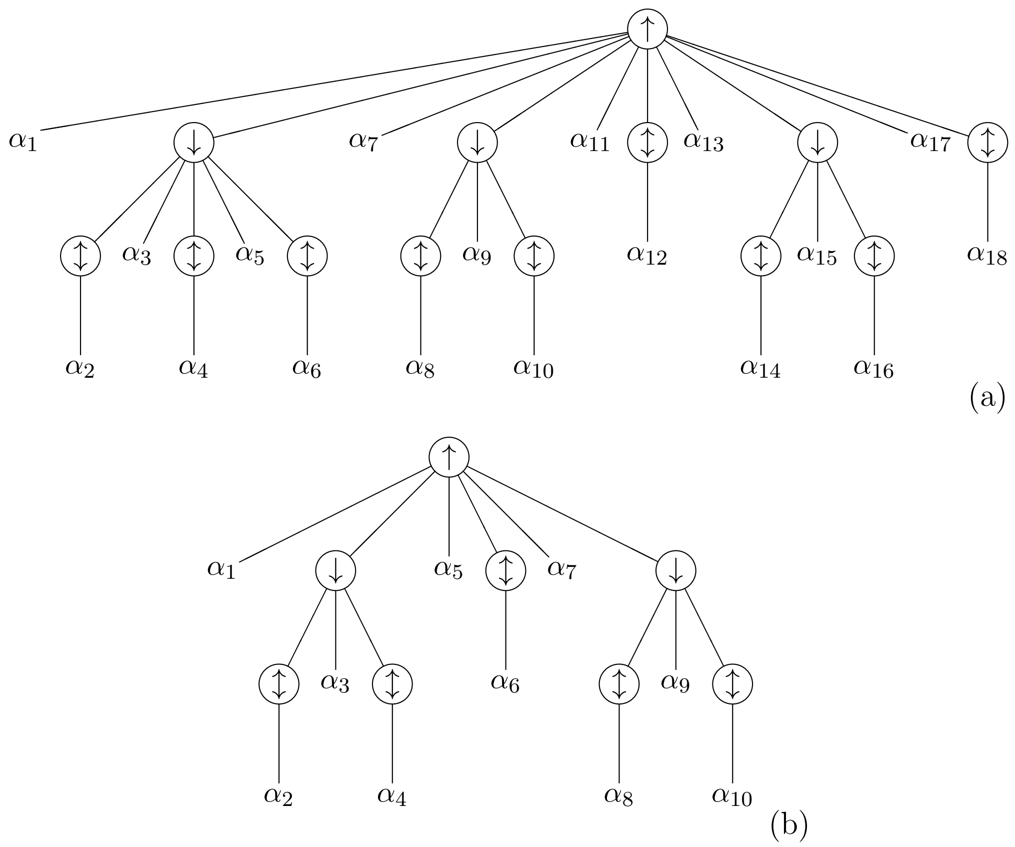

$\uparrow$
α_1 $\downarrow$ α_7 $\downarrow$ α_{11} $\updownarrow$ α_{13} $\downarrow$ α_{17} $\updownarrow$
$\updownarrow$ α_3 $\updownarrow$ α_5 $\updownarrow$ $\updownarrow$ α_9 $\updownarrow$ α_{12} $\updownarrow$ α_{15} $\updownarrow$ α_{18}
α_2 α_4 α_6 α_8 α_{10} α_{14} α_{16}
(a)
$\uparrow$
α_1 $\downarrow$ α_5 $\updownarrow$ α_7 $\downarrow$
$\updownarrow$ α_3 $\updownarrow$ α_6 $\updownarrow$ α_9 $\updownarrow$
α_2 α_4 α_8 α_{10}
(b)

Figure 8.1: Two structure trees that are in minimal normal form. (a) The structure tree of the DNA expression $E_{\text{MinNF}}(X)$ from (8.1), denoting the nick free formal DNA molecule from (a.o.) Figure 5.3. (b) The structure tree of the DNA expression $E_{\text{MinNF}}(X) = E_a$ from (8.2), denoting the nick free formal DNA molecule from Figure 5.4.

8.4 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 2.16 and Lemma 5.20). 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, whereas 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 8.10, 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 appears 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 report, 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 § 2.7). 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 first has to be a DNA expression. 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 profer 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 ($\mathcal{D}_{\text{MinNF}}.3$), 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 6.17(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 6.17(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 $(\Sigma_2, \Delta_2, P_2, S_2)$, where $\Sigma_2 = \{E_\star, U_\star, U_\alpha, U_E, \widehat{U}_E, E_{+,+}, E_{+,\star}, L_\star, L_\alpha, L_E, \widehat{L}_E, E_{-,-}, E_{-,\star}, A_{\alpha,\updownarrow}, A_{\updownarrow,\updownarrow}, A_{\alpha,\star}, A_{\updownarrow,\star}, \alpha, \mathrm{A}, \mathrm{C}, \mathrm{G}, \mathrm{T}, \uparrow, \downarrow, \updownarrow, \langle\ ,\ \rangle\}$ is the total alphabet, $\Delta_2 = \{\mathrm{A}, \mathrm{C}, \mathrm{G}, \mathrm{T}, \uparrow, \downarrow, \updownarrow, \langle\ ,\ \rangle\}$ is the alphabet of terminal symbols, P_2 is the set of productions (which is given below), and $S_2 = E_\star$ is the axiom of G_2.

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. $U_\star \longrightarrow \langle\downarrow \alpha A_{\updownarrow,\updownarrow}\rangle \alpha\widehat{U}_E \mid \langle\downarrow \alpha A_{\updownarrow,\updownarrow}\rangle U_E \mid \langle\downarrow \langle\updownarrow \alpha\rangle A_{\alpha,\updownarrow}\rangle \alpha\widehat{U}_E \mid \langle\downarrow \langle\updownarrow \alpha\rangle A_{\alpha,\updownarrow}\rangle U_E$
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. $E_{+,+} \longrightarrow \langle\updownarrow \alpha\rangle \mid \langle\downarrow \langle\updownarrow \alpha\rangle A_{\alpha,\updownarrow}\rangle$
8. $E_{+,\star} \longrightarrow \langle\updownarrow \alpha\rangle \mid \langle\downarrow \langle\updownarrow \alpha\rangle A_{\alpha,\star}\rangle$

[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 $\widehat{U}_E$.

9. $L_\star \longrightarrow \alpha \mid \alpha L_E \mid \langle \updownarrow \alpha \rangle L_\alpha \mid \langle \updownarrow \alpha \rangle L_E$
10. $L_\star \longrightarrow \langle \uparrow \alpha A_{\updownarrow,\updownarrow} \rangle \alpha \widehat{L}_E \mid \langle \uparrow \alpha A_{\updownarrow,\updownarrow} \rangle L_E \mid \langle \uparrow \langle \updownarrow \alpha \rangle A_{\alpha,\updownarrow} \rangle \alpha \widehat{L}_E \mid \langle \uparrow \langle \updownarrow \alpha \rangle A_{\alpha,\updownarrow} \rangle L_E$
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. $E_{-,-} \longrightarrow \langle \updownarrow \alpha \rangle \mid \langle \uparrow \langle \updownarrow \alpha \rangle A_{\alpha,\updownarrow} \rangle$
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. $\alpha \longrightarrow \text{A} \mid \text{C} \mid \text{G} \mid \text{T} \mid \text{A}\alpha \mid \text{C}\alpha \mid \text{G}\alpha \mid \text{T}\alpha$

As is the case with most of the DNA expressions in this report, 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 right-linear productions in line 20.

Example 8.11 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 8.1,

$$E_{\text{MinNF}}(X) = \langle \uparrow \alpha_1 \langle \downarrow \langle \updownarrow \alpha_2 \rangle \alpha_3 \rangle \rangle, \tag{8.4}$$

which is DNA expression E from Example 5.2 (see also Table 8.1). The following, leftmost derivation in G_2 yields $E_{\text{MinNF}}(X)$:

$$\begin{aligned}
E_\star &\overset{1,2}{\Longrightarrow} \langle \uparrow U_\star \rangle \\
&\overset{2,2}{\Longrightarrow} \langle \uparrow \alpha_1 U_E \rangle \\
&\overset{5,1}{\Longrightarrow} \langle \uparrow \alpha_1 E_{+,\star} \rangle \\
&\overset{8,2}{\Longrightarrow} \langle \uparrow \alpha_1 \langle \downarrow \langle \updownarrow \alpha_2 \rangle A_{\alpha,\star} \rangle \rangle \\
&\overset{18,1}{\Longrightarrow} \langle \uparrow \alpha_1 \langle \downarrow \langle \updownarrow \alpha_2 \rangle \alpha_3 \rangle \rangle \\
&= E_{\text{MinNF}}(X).
\end{aligned}$$

As in earlier derivations, numbers i, j above an arrow indicate that we have used production (i, j). ■

Example 8.12 Consider the minimal normal form DNA expression $E_{\text{MinNF}}(X)$ from (8.1), which denotes the nick free formal DNA molecule X from (a.o.) Figure 5.2 and

Figure 5.3. 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 \\
&\qquad \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 \\
&\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} 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 \\
&\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} E_{+,\star}\rangle \\
&\overset{8,1}{\Longrightarrow} E_{\mathrm{MinNF}}(X).
\end{aligned}$$

■

Example 8.13 Consider the minimal normal form DNA expression $E_{\mathrm{MinNF}}(X)$ from (8.3). This DNA expression denotes the formal DNA molecule X from Figure 5.5, 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 \\
&\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 \overset{7,2}{\Longrightarrow} \ldots \overset{16,1}{\Longrightarrow} \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 \\
&\qquad E_{+,+} U_\alpha\rangle
\end{aligned}$$

$$\begin{aligned}
&\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 \\
&\qquad \langle\updownarrow \alpha_{12}\rangle U_\alpha \rangle \\
&\overset{4,2}{\Longrightarrow} \dots \overset{5,2}{\Longrightarrow} \dots \overset{7,1}{\Longrightarrow} \dots \overset{4,2}{\Longrightarrow} \dots \\
&\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 \\
&\qquad \langle\updownarrow \alpha_{12}\rangle \alpha_{13} \langle\updownarrow \alpha_{14}\rangle \alpha_{15} E_{+,+} U_E \rangle \\
&\overset{7,1}{\Longrightarrow} \dots \overset{5,3}{\Longrightarrow} \dots \overset{7,1}{\Longrightarrow} \dots \\
&\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 \\
&\qquad \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} \dots \overset{16,1}{\Longrightarrow} \dots \\
&\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 \\
&\qquad \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 \\
&\qquad \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_{\text{MinNF}}(X).
\end{aligned}$$

■

Example 8.14 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 5.26, 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 8.1, $E_{\text{MinNF}}(X)$ is the $\downarrow$-expression based on E_1 and E_2 analogous to the description in Theorem 5.28:

$$E_{\text{MinNF}}(X) = \langle\downarrow \langle\uparrow \alpha_1 \langle\updownarrow \alpha_2\rangle\rangle \alpha_3 \langle\updownarrow \alpha_4\rangle \langle\updownarrow \alpha_5\rangle\rangle .$$

The following, leftmost derivation in G_2 yields $E_{\text{MinNF}}(X)$:

$$\begin{aligned}
E_\star &\overset{1,3}{\Longrightarrow} \langle\downarrow L_\star\rangle \\
&\overset{10,1}{\Longrightarrow} \left\langle\downarrow \langle\uparrow \alpha_1 A_{\updownarrow,\updownarrow}\rangle \alpha_3 \widehat{L}_E\right\rangle \\
&\overset{17,1}{\Longrightarrow} \left\langle\downarrow \langle\uparrow \alpha_1 \langle\updownarrow \alpha_2\rangle\rangle \alpha_3 \widehat{L}_E\right\rangle \\
&\overset{13,2}{\Longrightarrow} \langle\downarrow \langle\uparrow \alpha_1 \langle\updownarrow \alpha_2\rangle\rangle \alpha_3 E_{-,-} L_E\rangle
\end{aligned}$$

$$\begin{aligned}
&\overset{14,1}{\Longrightarrow} && \langle\downarrow \langle\uparrow \alpha_1 \langle\updownarrow \alpha_2\rangle\rangle\, \alpha_3 \langle\updownarrow \alpha_4\rangle\, L_E\rangle \\
&\overset{12,1}{\Longrightarrow} && \langle\downarrow \langle\uparrow \alpha_1 \langle\updownarrow \alpha_2\rangle\rangle\, \alpha_3 \langle\updownarrow \alpha_4\rangle\, E_{-,\star}\rangle \\
&\overset{15,1}{\Longrightarrow} && \langle\downarrow \langle\uparrow \alpha_1 \langle\updownarrow \alpha_2\rangle\rangle\, \alpha_3 \langle\updownarrow \alpha_4\rangle \langle\updownarrow \alpha_5\rangle\rangle \\
&= && E_{\mathrm{MinNF}}(X).
\end{aligned}$$

■

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 axiom $E_\star$ of the grammar.

Some of the languages consist of sequences $\varepsilon_1 \dots \varepsilon_n$ 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 $i = 1, \dots, n$, ε_i is either a maximal $\mathcal{N}$-word occurrence α (in $\varepsilon_1 \dots \varepsilon_n$), 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) $\varepsilon_1, \dots, \varepsilon_n$ fit together by upper strands.

(LU.3) $L(\mathcal{S}^+(\varepsilon_1)) \in \mathcal{A}_\pm \cup \mathcal{A}_+$.

Lemma 8.15 *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 $\langle\downarrow \varepsilon_{1,1} \dots \varepsilon_{1,m}\rangle$ with $m \geq 2$, such that $\varepsilon_{1,1}, \dots, \varepsilon_{1,m}$ 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 $\langle\downarrow \varepsilon_{1,1} \dots \varepsilon_{1,m}\rangle$ with $m \geq 2$, such that $\varepsilon_{1,1}, \dots, \varepsilon_{1,m}$ 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 $\langle\downarrow \varepsilon_{1,1} \dots \varepsilon_{1,m}\rangle$ with $m \geq 2$, such that $\varepsilon_{1,1}, \dots, \varepsilon_{1,m}$ 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 $\langle\downarrow \varepsilon_{1,1} \dots \varepsilon_{1,m}\rangle$ with $m \geq 2$, such that $\varepsilon_{1,1}, \dots, \varepsilon_{1,m}$ 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 $\langle\downarrow \varepsilon_{1,1} \dots \varepsilon_{1,m}\rangle$ with $m \geq 2$, such that $\varepsilon_{1,1}, \dots, \varepsilon_{1,m}$ 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 $\varepsilon_1 \dots \varepsilon_n$ 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 $\varepsilon_1 \dots \varepsilon_n$).*

13. *$\mathcal{L}(U_E)$ is the set of all sequences of arguments $\varepsilon_1 \dots \varepsilon_n$ 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 $\varepsilon_1 \dots \varepsilon_n$ 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 $\varepsilon_1 \dots \varepsilon_n$ 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 8.1).

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,

$$\begin{aligned} &L(\mathcal{S}^+(\alpha)) = L(\tbinom{\alpha}{-}) \in \mathcal{A}_+, \ R(\mathcal{S}^+(\alpha)) = R(\tbinom{\alpha}{-}) \in \mathcal{A}_+, \\ &L(\mathcal{S}(\langle \updownarrow \alpha \rangle)) = L(\tbinom{\alpha}{c(\alpha)}) \in \mathcal{A}_\pm \text{ and } R(\mathcal{S}(\langle \updownarrow \alpha \rangle)) = R(\tbinom{\alpha}{c(\alpha)}) \in \mathcal{A}_\pm. \end{aligned}$$

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 2.15(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 2.15(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

$$\underbrace{\underbrace{E_{+,+}\alpha} \ldots \underbrace{E_{+,+}\alpha}}_{\frac{i_0-1}{2}\text{ times}} E_{+,+}U_E.$$

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 2.15(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 $\varepsilon_3 \ldots \varepsilon_n$ 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 $\varepsilon_3 \ldots \varepsilon_n$ is an element of $\mathcal{L}(\widehat{U}_E)$. Hence, either $X = \varepsilon_1 \alpha \varepsilon_3 \ldots \varepsilon_n \in \mathcal{L}(\langle\downarrow \alpha A_{\updownarrow,\updownarrow}\rangle\, \alpha \widehat{U}_E)$, or $X = \varepsilon_1 \alpha \varepsilon_3 \ldots \varepsilon_n \in \mathcal{L}(\langle\downarrow \langle\updownarrow \alpha\rangle\, A_{\alpha,\updownarrow}\rangle\, \alpha \widehat{U}_E)$.

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 $\varepsilon_2 \ldots \varepsilon_n$ has Properties (LU.1)–(LU.3) and ε_2 is a DNA expression. By Claim 13, $\varepsilon_2 \ldots \varepsilon_n$ is an element of $\mathcal{L}(U_E)$. Hence, either $X = \varepsilon_1 \varepsilon_2 \ldots \varepsilon_n \in \mathcal{L}(\langle\downarrow \alpha A_{\updownarrow,\updownarrow}\rangle\, U_E)$, or $X = \varepsilon_1 \varepsilon_2 \ldots \varepsilon_n \in \mathcal{L}(\langle\downarrow \langle\updownarrow \alpha\rangle\, A_{\alpha,\updownarrow}\rangle\, U_E)$.

16 Let X be an arbitrary element of $\mathcal{L}(\langle\uparrow U_\star\rangle)$. By the previous claim, $X = \langle\uparrow \varepsilon_1 \ldots \varepsilon_n\rangle$ for $n \geq 1$ $\mathcal{N}$-words and DNA expressions $\varepsilon_1, \ldots, \varepsilon_n$ with some special properties. By Property (LU.2), the arguments $\varepsilon_1, \ldots, \varepsilon_n$ 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 8.8, X is in minimal normal form.

On the other hand, let X be an arbitrary $\uparrow$-expression in minimal normal form. Then $X = \langle\uparrow \varepsilon_1 \ldots \varepsilon_n\rangle$ for $n \geq 1$ maximal $\mathcal{N}$-word occurrences and DNA expressions $\varepsilon_1, \ldots, \varepsilon_n$ 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 $\varepsilon_1 \ldots \varepsilon_n$ have the properties listed in the previous claim. Hence, $\varepsilon_1 \ldots \varepsilon_n$ is an element of $\mathcal{L}(U_\star)$ and $X = \langle\uparrow \varepsilon_1 \ldots \varepsilon_n\rangle \in \mathcal{L}(\langle\uparrow U_\star\rangle)$.

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: $\varepsilon_1, \ldots, \varepsilon_n$ 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 $\varepsilon_1 \ldots \varepsilon_n$ 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 8.15(2), (16) and (17) is

Theorem 8.16 *$\mathcal{L}(G_2) = \mathcal{L}_{G_2}(E_\star)$ is the language $\mathcal{D}_{\text{MinNF}}$ 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 2.4 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 8.2: 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 2.4 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 8.2. 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 8.2, 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 8.2, 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 8.17 *Let A be an arbitrary non-terminal symbol in $\Sigma_2 \setminus \Delta_2$ and let X be a string over Σ_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 8.17 that none of the non-terminal symbols in G_2 is self-embedding, and thus that G_2 is not self-embedding. By Theorem 8.16, $\mathcal{L}(G_2)$ equals the language of all DNA expressions in minimal normal form. Hence, by Proposition 2.4,

Theorem 8.18 *The language* $\mathcal{D}_{MinNF}$ *of DNA expressions in minimal normal form is regular.*

Chapter 9

Algorithms for the Minimal Normal Form

At the beginning of Chapter 8, we introduced the (minimal) normal form 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 8.1 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 ways to rewrite an arbitrary DNA expression E_1^* into its normal form equivalent, without referring to $\mathcal{S}(E_1^*)$. First, we propose a direct, recursive function. This function turns out to use at least quadratic time in the worst case. We subsequently describe an alternative, two-step algorithm, and prove that it is correct and uses linear time and space.

Note that the recursive function `MakeMinimal`, which we have described in Chapter 7, is not sufficient to produce some kind of a normal form. By Corollary 7.13, `MakeMinimal` does not necessarily yield the same output for different equivalent inputs, which is required for a normal form.

9.1 Recursive algorithm for the minimal normal form

In Chapter 7, we have described a recursive function `MakeMinimal`, which rewrites a given DNA expression E_1^* into an equivalent, minimal DNA expression. We proved that, with a proper datastructure, this function requires time and space that are linear in $|E_1^*|$ (see Corollary 7.38 and Theorem 7.40).

```
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 $\updownarrow$-expression)
4.     **then** **if** (the argument of $E$ is a DNA expression $E_1$)
5.          **then** MakeMinimalNF ($E_1$);
6.               substitute $E$ by a DNA expression $E'$ in minimal normal form
                   satisfying $E' \equiv E$;
7.          **fi**

8.     **else**   // $E$ is an $\uparrow$-expression or a $\downarrow$-expression
9.          **for** all expression-arguments $E_i$ of $E$ (in some order)
10.         **do**  MakeMinimalNF ($E_i$);
11.         **od**
12.         substitute $E$ by a DNA expression $E'$ in minimal normal form
              satisfying $E' \equiv E$;
13.    **fi**
14.  }
```

Figure 9.1: Pseudo-code of the recursive function `MakeMinimalNF`.

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 9.1 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.

A possible implementation of the two lines would be to first determine $X = \mathcal{S}(E)$, and then to use Definition 8.1 to construct E'. Of course, since we prefer not to use the semantics of the DNA expression, this is not a type of implementation that we look for. Moreover, with such an implementation, we would not benefit at all from the fact that the expression-arguments of E are in minimal normal form already. Hence, the recursive calls we have made for these expression-arguments would be useless, after all. However, it does make clear that in principle, the two lines can be effectively executed in finite time.

Note that indeed, the structure of `MakeMinimalNF`is equal to that of `MakeMinimal`(see Figure 7.1). 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 possible to prove that the set-up of `MakeMinimalNF` is correct.

Theorem 9.1 *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 8.1, $E_2^* = E_1^* = \langle \updownarrow \alpha_1 \rangle = E_{\text{MinNF}}(X)$ 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 `Make-MinimalNF` yield DNA expressions that are equivalent to the expression-arguments E_i of $E = E_1^*$. Then Lemma 3.7 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.

   Note that we did not use the fact that the expression-arguments resulting from the recursive calls 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 of `MakeMinimalNF`, we can also draw another important conclusion: the recursive approach of the function is not as efficient as that of `MakeMinimal`. We demonstrate this by examining its complexity for DNA expressions of a specific type.

Example 9.2 Let α be an arbitrary $\mathcal{N}$-word, and let

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

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,
-

$$\begin{aligned}
\mathcal{S}(E_{2p}) &= \tbinom{\alpha}{c(\alpha)}\tbinom{\alpha}{-}\underbrace{\tbinom{\alpha}{c(\alpha)}\tbinom{-}{\alpha}\tbinom{\alpha}{c(\alpha)}\tbinom{\alpha}{-}\ldots\tbinom{\alpha}{c(\alpha)}\tbinom{-}{\alpha}\tbinom{\alpha}{c(\alpha)}\tbinom{\alpha}{-}}_{p-1\text{ times}}\cdot\\
&\quad\tbinom{\alpha}{c(\alpha)}\tbinom{-}{\alpha}\tbinom{\alpha}{c(\alpha)}\cdot\\
&\quad\underbrace{\tbinom{\alpha}{-}\tbinom{\alpha}{c(\alpha)}\tbinom{-}{\alpha}\tbinom{\alpha}{c(\alpha)}\ldots\tbinom{\alpha}{-}\tbinom{\alpha}{c(\alpha)}\tbinom{-}{\alpha}\tbinom{\alpha}{c(\alpha)}}_{p-1\text{ times}}\tbinom{\alpha}{-}\tbinom{\alpha}{c(\alpha)}\\
&= \tbinom{\alpha}{c(\alpha)}\tbinom{\alpha}{-}\underbrace{\tbinom{\alpha}{c(\alpha)}\tbinom{-}{\alpha}\tbinom{\alpha}{c(\alpha)}\tbinom{\alpha}{-}\ldots\tbinom{\alpha}{c(\alpha)}\tbinom{-}{\alpha}\tbinom{\alpha}{c(\alpha)}\tbinom{\alpha}{-}}_{2p-1\text{ times}}\tbinom{\alpha}{c(\alpha)},\\
\mathcal{S}(E_{2p+1}) &= \underbrace{\tbinom{\alpha}{c(\alpha)}\tbinom{-}{\alpha}\tbinom{\alpha}{c(\alpha)}\tbinom{\alpha}{-}\ldots\tbinom{\alpha}{c(\alpha)}\tbinom{-}{\alpha}\tbinom{\alpha}{c(\alpha)}\tbinom{\alpha}{-}}_{p\text{ times}}\cdot\\
&\quad\tbinom{\alpha}{c(\alpha)}\tbinom{-}{\alpha}\tbinom{\alpha}{c(\alpha)}\cdot\\
&\quad\underbrace{\tbinom{\alpha}{-}\tbinom{\alpha}{c(\alpha)}\tbinom{-}{\alpha}\tbinom{\alpha}{c(\alpha)}\ldots\tbinom{\alpha}{-}\tbinom{\alpha}{c(\alpha)}\tbinom{-}{\alpha}\tbinom{\alpha}{c(\alpha)}}_{p\text{ times}}\\
&= \tbinom{\alpha}{c(\alpha)}\tbinom{-}{\alpha}\underbrace{\tbinom{\alpha}{c(\alpha)}\tbinom{\alpha}{-}\tbinom{\alpha}{c(\alpha)}\tbinom{-}{\alpha}\ldots\tbinom{\alpha}{c(\alpha)}\tbinom{\alpha}{-}\tbinom{\alpha}{c(\alpha)}\tbinom{-}{\alpha}}_{2p\text{ times}}\tbinom{\alpha}{c(\alpha)},
\end{aligned}$$

-

$$\begin{aligned}
B_{\uparrow}(\mathcal{S}(E_{2p})) &= B_{\downarrow}(\mathcal{S}(E_{2p}))+1=2p,\\
B_{\downarrow}(\mathcal{S}(E_{2p+1})) &= B_{\uparrow}(\mathcal{S}(E_{2p+1}))+1=2p+1,
\end{aligned}$$

- $n_{\updownarrow}(\mathcal{S}(E_q))=2q$, 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 6.12(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 8.1(3) and (4) and the construction from Theorem 5.12, the corresponding DNA expressions in minimal normal form are

$$\begin{aligned}
E'_{2p} &= E_{\text{MinNF}}(\mathcal{S}(E_{2p}))\\
&= \left\langle\uparrow\langle\updownarrow\alpha\rangle\,\alpha\,\underbrace{\langle\downarrow\langle\updownarrow\alpha\rangle\,\alpha\,\langle\updownarrow\alpha\rangle\rangle\,\alpha\ldots\langle\downarrow\langle\updownarrow\alpha\rangle\,\alpha\,\langle\updownarrow\alpha\rangle\rangle\,\alpha}_{2p-1\text{ times}}\langle\updownarrow\alpha\rangle\right\rangle, \qquad (9.1)\\
E'_{2p+1} &= E_{\text{MinNF}}(\mathcal{S}(E_{2p+1}))\\
&= \left\langle\downarrow\langle\updownarrow\alpha\rangle\,\alpha\,\underbrace{\langle\uparrow\langle\updownarrow\alpha\rangle\,\alpha\,\langle\updownarrow\alpha\rangle\rangle\,\alpha\ldots\langle\uparrow\langle\updownarrow\alpha\rangle\,\alpha\,\langle\updownarrow\alpha\rangle\rangle\,\alpha}_{2p\text{ times}}\langle\updownarrow\alpha\rangle\right\rangle.
\end{aligned}$$

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

$$\langle \downarrow \langle \updownarrow \alpha \rangle \, \alpha \, E'_{2p} \, \alpha \, \langle \updownarrow \alpha \rangle \rangle$$

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 $2p-2, 2p-3, 2p-4, \dots, 1$ occurrences of operators in $E'_{2p-1}, E'_{2p-2}, E'_{2p-3}, \dots, E'_2$, 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 9.2 and Figure 9.3 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 9.3 *The worst case time complexity of the recursive function* `MakeMinimalNF` *is at least quadratic.*

9.2 Two-step algorithm for the minimal normal form

As we have seen in § 9.1, the direct, recursive function `MakeMinimalNF` does produce an equivalent DNA expression in minimal normal form for its argument E, but it is not really 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 9.4, 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 5.3, 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 procedure `RotateToMinimal`.

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 8.9(1), such inner occurrences correspond

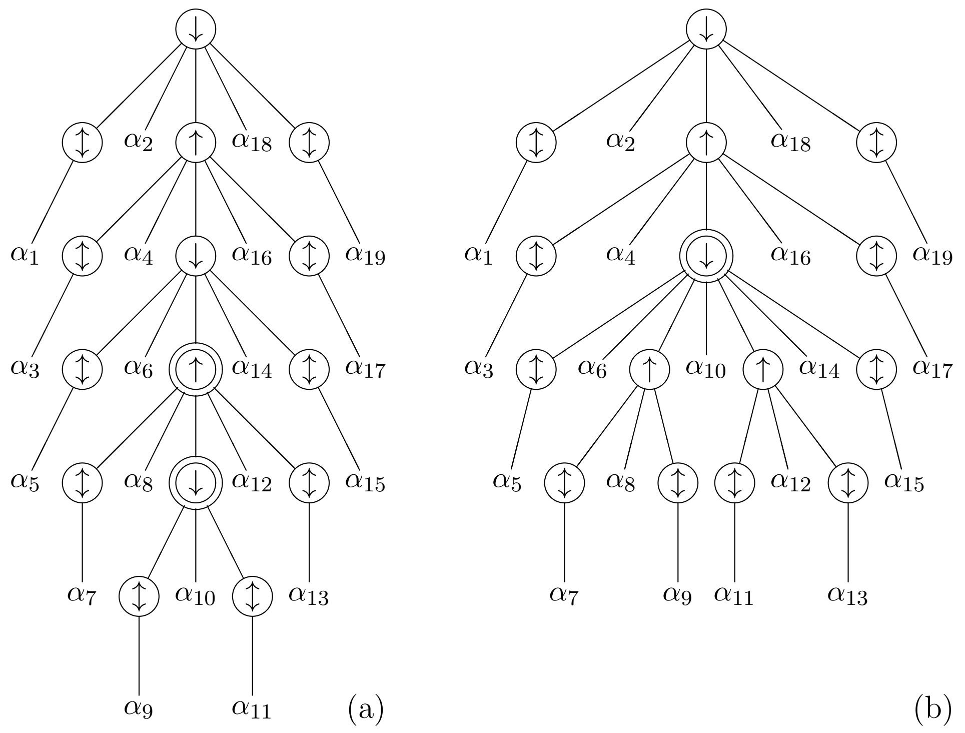


Figure 9.2: Structure trees of the DNA expressions that we successively obtain, when we apply the recursive function `MakeMinimalNF` to the $\downarrow$-expression E_5 from Example 9.2. To make the structure trees easier to compare, we have added subscripts to the occurring $\mathcal{N}$-words. (a) Structure 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 9.3)

to violations of Property ($\mathcal{D}_{\mathrm{MinNF}}$.4). When we perform the substitution in line 12, we get rid of one inner occurrence of $\uparrow$.

In Lemma 8.10, 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 8.1: the substitution in line 12 corresponds to breaking a large lower block into two smaller lower blocks.

Note that Properties ($\mathcal{D}_{\mathrm{MinNF}}$.1)–($\mathcal{D}_{\mathrm{MinNF}}$.3) are not mentioned in the pseudo-code. This is natural, as they equal Properties ($\mathcal{D}_{\mathrm{Min}}$.1)–($\mathcal{D}_{\mathrm{Min}}$.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 9.4 In Example 5.14, we have constructed four minimal DNA expressions for the formal DNA molecule X depicted in Figure 5.4. Let

$$E = E_c = \langle\downarrow \langle\uparrow \alpha_1 \langle\updownarrow \alpha_2\rangle\rangle\, \alpha_3 \langle\uparrow \langle\updownarrow \alpha_4\rangle\, \alpha_5 \langle\updownarrow \alpha_6\rangle\, \alpha_7 \langle\updownarrow \alpha_8\rangle\rangle\, \alpha_9 \langle\updownarrow \alpha_{10}\rangle\rangle \tag{9.2}$$

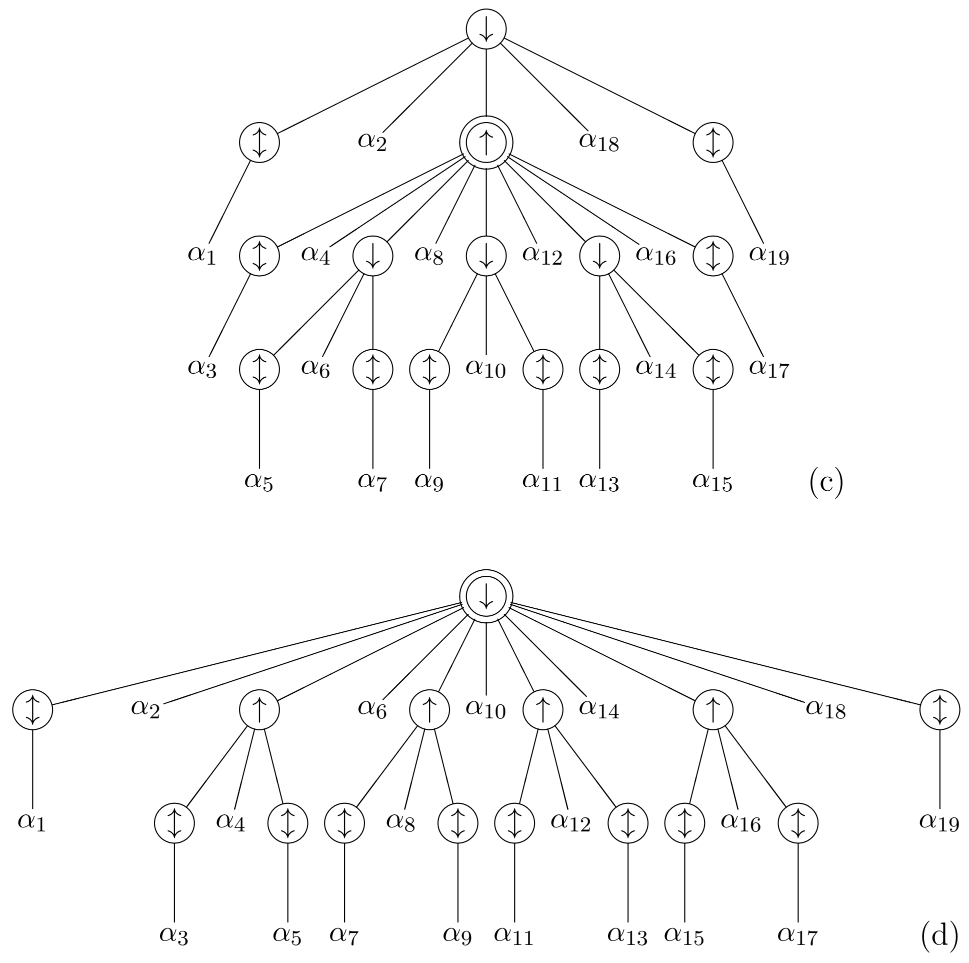


Figure 9.3: Structure trees of the DNA expressions that we successively obtain, when we apply the recursive function `MakeMinimalNF` to the $\downarrow$-expression E_5 from Example 9.2 (continuation of Figure 9.2). (c) Structure tree after rewriting the DNA subexpression E_4 into the minimal normal form 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.

(see (5.10)), which has been depicted in Figure 9.5(a). The fact that E is minimal implies (1) that, by Theorem 7.12, 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

$$E = \langle\uparrow \alpha_1 \langle\downarrow \langle\updownarrow \alpha_2\rangle\, \alpha_3 \langle\uparrow \langle\updownarrow \alpha_4\rangle\, \alpha_5 \langle\updownarrow \alpha_6\rangle\, \alpha_7 \langle\updownarrow \alpha_8\rangle\rangle\, \alpha_9 \langle\updownarrow \alpha_{10}\rangle\rangle\rangle\,. \tag{9.3}$$

This is the minimal DNA expression E_b from (5.9). It has been depicted in Figure 9.5(b). Because the $\uparrow$-expression E has an inner occurrence of $\uparrow$, we enter the while-loop. We select the $\downarrow$-subexpression

$$\widehat{E} = \langle\downarrow \langle\updownarrow \alpha_2\rangle\, \alpha_3 \langle\uparrow \langle\updownarrow \alpha_4\rangle\, \alpha_5 \langle\updownarrow \alpha_6\rangle\, \alpha_7 \langle\updownarrow \alpha_8\rangle\rangle\, \alpha_9 \langle\updownarrow \alpha_{10}\rangle\rangle\,,$$

```
NormalizeMinimal ($E_2^*$)
  // rewrites an arbitrary minimal DNA expression $E_2^*$
  // into a DNA expression $E_3^*$ in minimal normal form
  // satisfying $E_3^* \equiv E_2^*$;
  // uses local rearrangements of the DNA expression for this
{
  $E = E_2^*$;
  if ($E$ is an $\updownarrow$-expression)
  then $E_3^* = E$;
  else   // $E$ is an $\uparrow$-expression or a $\downarrow$-expression;
         // without loss of generality, assume it is an $\uparrow$-expression
      if ($E$ is alternating and its first argument is a $\downarrow$-argument)
      then substitute $E$ by the result of procedure RotateToMinimal;
                                                        ($\mathcal{D}_{\texttt{MinNF}}$.5)
      fi
         // $E$ is an $\uparrow$-expression or a $\downarrow$-expression;
         // without loss of generality, assume it is an $\uparrow$-expression
      while ($E$ has inner occurrences of $\uparrow$)
      do  select a $\downarrow$-subexpression $\widehat{E}$ of $E$
            which has at least one $\uparrow$-argument $E_i$;
              // $\widehat{E} = \langle\downarrow \varepsilon_1 \ldots \varepsilon_{i-1} E_i \varepsilon_{i+1} \ldots \varepsilon_n\rangle$
              // and $E_i = \langle\uparrow \varepsilon_{i,1} \varepsilon_{i,2} \ldots \varepsilon_{i,m-1} \varepsilon_{i,m}\rangle$
          substitute $\widehat{E}$ in $E$
            by $\langle\downarrow \varepsilon_1 \ldots \varepsilon_{i-1} \varepsilon_{i,1}\rangle\, \varepsilon_{i,2} \ldots \varepsilon_{i,m-1}\, \langle\downarrow \varepsilon_{i,m} \varepsilon_{i+1} \ldots \varepsilon_n\rangle$;  ($\mathcal{D}_{\texttt{MinNF}}$.4)
      od
      $E_3^* = E$;
  fi
}
```

Figure 9.4: Pseudo-code of the algorithm NormalizeMinimal.

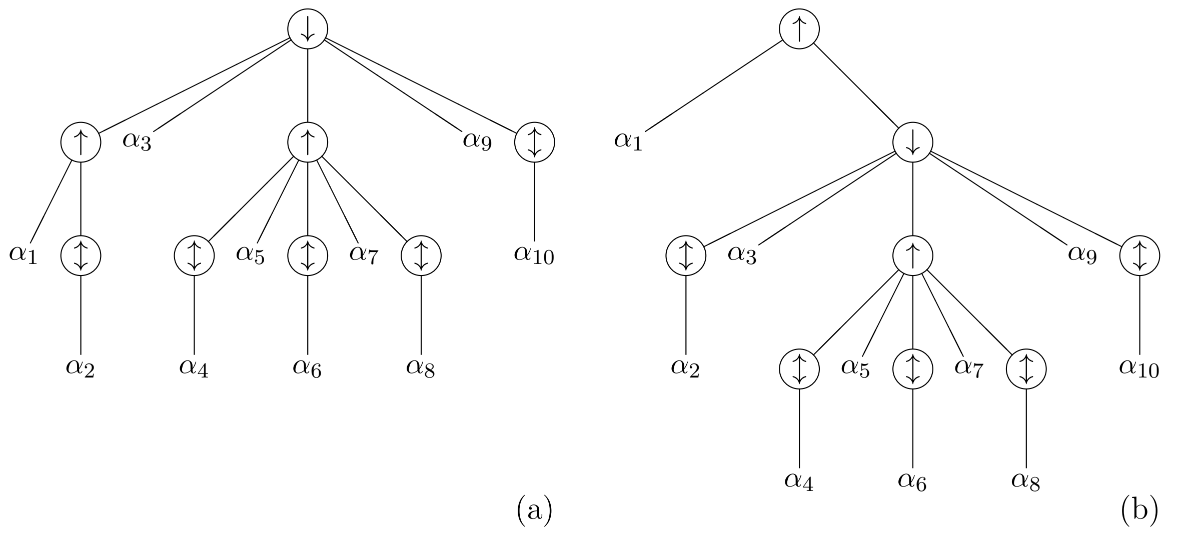


Figure 9.5: Structure trees of the first two minimal DNA expressions occurring in Example 9.4, denoting the formal DNA molecule from Figure 5.4. (a) The structure tree of E_c from (9.2). (b) The structure tree of E_b from (9.3).

(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 ($\mathcal{D}_{\mathrm{MinNF}}$.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

$$E = \langle\uparrow\alpha_1\,\langle\downarrow\langle\updownarrow\alpha_2\rangle\,\alpha_3\,\langle\updownarrow\alpha_4\rangle\rangle\,\alpha_5\,\langle\updownarrow\alpha_6\rangle\,\alpha_7\,\langle\downarrow\langle\updownarrow\alpha_8\rangle\,\alpha_9\,\langle\updownarrow\alpha_{10}\rangle\rangle\rangle\,. \tag{9.4}$$

After the substitution, E has no inner occurrences of $\uparrow$, anymore, and we exit the while-loop. We do not rewrite the DNA expression any further. Indeed, E has all five properties from Lemma 8.6, and thus is in minimal normal form. It equals $E_{\mathrm{MinNF}}(X) = E_a$ from (5.8) and (8.2), which has been depicted in Figure 8.1(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 § 9.3.

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 ($\mathcal{D}_{\mathrm{MinNF}}$.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 9.5 *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 6.2, $\widehat{\varepsilon}$ 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 6.2, 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 9.6 *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 5.3, $E_2^* = \langle\updownarrow \alpha_1\rangle$ for an $\mathcal{N}$-word α_1. By Case 1 of Definition 8.1, 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 6.15. E also has Properties $(\mathcal{D}_{\text{MinNF}}.1)$–$(\mathcal{D}_{\text{MinNF}}.3)$ from Lemma 8.6, 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 6.2, 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 7.27, 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 is a minimal $\uparrow$-expression with Property $(\mathcal{D}_{\text{MinNF}}.5)$, satisfying $E \equiv E_2^*$. (9.5)

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 (9.5) and Theorem 8.8 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 9.5 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 6.2, 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 (9.5) 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 (9.5) is indeed an invariant for the while-loop.

- Clearly, before the first iteration of the while-loop, Property (9.5) is valid.
- Suppose that Property (9.5) 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

  $$\widehat{E} = \langle\downarrow \varepsilon_1 \dots \varepsilon_{i-1} \langle\uparrow \varepsilon_{i,1}\varepsilon_{i,2} \dots \varepsilon_{i,m-1}\varepsilon_{i,m}\rangle \varepsilon_{i+1} \dots \varepsilon_n\rangle$$

  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 $\varepsilon_{i,1}, \varepsilon_{i,2}, \dots, \varepsilon_{i,m-1}, \varepsilon_{i,m}$.

  We zoom in on the $\uparrow$-argument $E_i = \langle\uparrow \varepsilon_{i,1}\varepsilon_{i,2} \dots \varepsilon_{i,m-1}\varepsilon_{i,m}\rangle$. 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 6.17(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), $\varepsilon_{i,1}, \varepsilon_{i,2}, \dots, \varepsilon_{i,m-1}, \varepsilon_{i,m}$ 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 3.11(1) and (2) to $\widehat{E}$ (with $r = 1$), we find that

  $$\widehat{E}' = \langle\uparrow \langle\downarrow \varepsilon_1 \dots \varepsilon_{i-1}\varepsilon_{i,1}\rangle \varepsilon_{i,2} \dots \varepsilon_{i,m-1} \langle\downarrow \varepsilon_{i,m}\varepsilon_{i+1} \dots \varepsilon_n\rangle\rangle$$

  is a DNA expression satisfying $\widehat{E}' \equiv \widehat{E}$.

[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.

By Lemma 6.17(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 $\uparrow_0$:

$$E_0 = \left\langle \uparrow_0 \widehat{\varepsilon}_1 \dots \widehat{\varepsilon}_{j-1} \widehat{E} \widehat{\varepsilon}_{j+1} \dots \widehat{\varepsilon}_l \right\rangle \tag{9.6}$$

for some $l \geq 1$ and $\mathcal{N}$-words and DNA expressions $\widehat{\varepsilon}_1, \dots, \widehat{\varepsilon}_{j-1}, \widehat{\varepsilon}_{j+1}, \dots, \widehat{\varepsilon}_l$. Note that E_0 may be equal to E, but that is not important for the moment. By Lemma 3.7 and Lemma 3.6,

$$\begin{aligned} E_0 &\equiv \left\langle \uparrow_0 \ \widehat{\varepsilon}_1 \dots \widehat{\varepsilon}_{j-1} \widehat{E}' \widehat{\varepsilon}_{j+1} \dots \widehat{\varepsilon}_l \right\rangle \\ &= \Big\langle \uparrow_0 \ \widehat{\varepsilon}_1 \dots \widehat{\varepsilon}_{j-1} \\ &\qquad \langle \uparrow \langle \downarrow \varepsilon_1 \dots \varepsilon_{i-1} \varepsilon_{i,1} \rangle \, \varepsilon_{i,2} \dots \varepsilon_{i,m-1} \langle \downarrow \varepsilon_{i,m} \varepsilon_{i+1} \dots \varepsilon_n \rangle \rangle \\ &\qquad \widehat{\varepsilon}_{j+1} \dots \widehat{\varepsilon}_l \Big\rangle \\ &\equiv \Big\langle \uparrow_0 \ \widehat{\varepsilon}_1 \dots \widehat{\varepsilon}_{j-1} \\ &\qquad \langle \downarrow \varepsilon_1 \dots \varepsilon_{i-1} \varepsilon_{i,1} \rangle \, \varepsilon_{i,2} \dots \varepsilon_{i,m-1} \langle \downarrow \varepsilon_{i,m} \varepsilon_{i+1} \dots \varepsilon_n \rangle \\ &\qquad \widehat{\varepsilon}_{j+1} \dots \widehat{\varepsilon}_l \Big\rangle . \end{aligned} \tag{9.7}$$

Hence, when we substitute $\widehat{E}$ in E_0 (and thus in E) by

$$\langle \downarrow \varepsilon_1 \dots \varepsilon_{i-1} \varepsilon_{i,1} \rangle \, \varepsilon_{i,2} \dots \varepsilon_{i,m-1} \langle \downarrow \varepsilon_{i,m} \varepsilon_{i+1} \dots \varepsilon_n \rangle , \tag{9.8}$$

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 Property ($\mathcal{D}_{\text{MinNF}}$.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 (9.8). 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 ($\mathcal{D}_{\text{MinNF}}$.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 (9.6) and (9.7) 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 ($\mathcal{D}_{\text{MinNF}}$.5) after the substitution.

Indeed, Property (9.5) is an invariant of the while-loop. After the last iteration of the loop, E has no inner occurrences of $\uparrow$, anymore, 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

$$\widehat{E} = \langle\downarrow \varepsilon_1 \dots \varepsilon_{i-1} \langle\uparrow \varepsilon_{i,1}\varepsilon_{i,2} \dots \varepsilon_{i,m-1}\varepsilon_{i,m}\rangle\, \varepsilon_{i+1} \dots \varepsilon_n\rangle$$

   of 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 way, we decrease the number of inner occurrences of $\uparrow$ in E by 1. Because this number cannot become negative, the number of iterations of the while-loop is limited, 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 9.6. $\square$

9.3 Implementation and complexity of the algorithm

In the description of algorithm `NormalizeMinimal` in Figure 9.4, 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. We now make the description more precise. In fact, we completely rewrite the while-loop. However, the purpose of the loop (to achieve Property ($\mathcal{D}_{\text{MinNF}}$.4)) and the types of substitutions performed in the loop remain the same.

We also describe three features of a datastructure to store the DNA expression in. We prove that with this datastructure, the algorithm can be carried out in linear time.

In the proof of Theorem 9.6(1) and (3), we have established that during the while-loop of `NormalizeMinimal`, the $\uparrow$-expression E is minimal. Hence, by Lemma 9.5, the condition

while (E has inner occurrences of $\uparrow$)

in line 10 of Figure 9.4 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 $\langle\downarrow \varepsilon_1 \dots \varepsilon_{i-1}\varepsilon_{i,1}\rangle\,, \varepsilon_{i,2}, \dots, \varepsilon_{i,m-1}, \langle\downarrow \varepsilon_{i,m}\varepsilon_{i+1} \dots \varepsilon_n\rangle$ 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 9.6. 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 9.2, for which the recursive function `MakeMinimalNF` appeared to use quadractic time.

```
1.   NormalizeMinimal2 ($E_2^*$)
       // rewrites an arbitrary minimal DNA expression $E_2^*$
       // into a DNA expression $E_3^*$ in minimal normal form
       // satisfying $E_3^* \equiv E_2^*$;
       // uses local rearrangements of the DNA expression for this
2.   {
3.     $E = E_2^*$;
4.     if ($E$ is an $\updownarrow$-expression)
5.     then $E_3^* = E$;
6.     else   // $E$ is an $\uparrow$-expression or a $\downarrow$-expression;
              // without loss of generality, assume it is an $\uparrow$-expression
7.          if ($E$ is alternating and its first argument is a $\downarrow$-argument)
8.          then substitute $E$ by the result of procedure RotateToMinimal;
                                                                  ($\mathcal{D}_{\texttt{MinNF}}$.5)
9.          fi
              // $E$ is an $\uparrow$-expression or a $\downarrow$-expression;
              // without loss of generality, assume it is an $\uparrow$-expression
10.         $\widehat{\varepsilon}$ = first argument of $E$;
11.         stop = false;
12.         while (not stop)
13.         do  if ($\widehat{\varepsilon}$ is a $\downarrow$-expression with at least one $\uparrow$-argument)
                  // let $\widehat{\varepsilon} = \langle\downarrow \varepsilon_1 \dots \varepsilon_{i-1} E_i \varepsilon_{i+1} \dots \varepsilon_n\rangle$,
                  // 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 $\widehat{\varepsilon}$
14.             then substitute $\widehat{\varepsilon}$ in $E$
                       by $\langle\downarrow \varepsilon_1 \dots \varepsilon_{i-1}\varepsilon_{i,1}\rangle\, \varepsilon_{i,2} \dots \varepsilon_{i,m-1}\, \langle\downarrow \varepsilon_{i,m}\varepsilon_{i+1} \dots \varepsilon_n\rangle$;
                                                                  ($\mathcal{D}_{\texttt{MinNF}}$.4)
15.                  $\widehat{\varepsilon} = \varepsilon_{i,2}$;
16.             else if ($\widehat{\varepsilon}$ is not the last argument of $E$)
17.                  then $\widehat{\varepsilon}$ = next argument of $E$;
18.                  else stop = true;
19.                  fi
20.             fi
21.         od
22.         $E_3^* = E$;
23.    fi
24.  }
```

Figure 9.6: Pseudo-code of the algorithm `NormalizeMinimal2`, which is a more detailed version of the algorithm `NormalizeMinimal` from Figure 9.4.

Example 9.7 Let α be an arbitrary $\mathcal{N}$-word, and let

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

As we observed in Example 9.2, 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 7.12, 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 $p \geq 2$:

$$E = E_{2p} \;=\; \langle\uparrow \langle\updownarrow \alpha\rangle\, \alpha\; E_{2p-1}\; \alpha\, \langle\updownarrow \alpha\rangle\rangle$$

$$= \langle\uparrow \langle\updownarrow \alpha\rangle\, \alpha\, \langle\downarrow \langle\updownarrow \alpha\rangle\, \alpha\, \langle\uparrow \langle\updownarrow \alpha\rangle\, \alpha\ E_{2(p-1)-1}\ \alpha\, \langle\updownarrow \alpha\rangle\rangle\, \alpha\, \langle\updownarrow \alpha\rangle\rangle\, \alpha\, \langle\updownarrow \alpha\rangle\rangle\,.$$

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 ($\mathcal{D}_{\mathrm{MinNF}}$.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

$$E \;=\; \langle\uparrow \langle\updownarrow \alpha\rangle\, \alpha\, \langle\downarrow \langle\updownarrow \alpha\rangle\, \alpha\, \langle\updownarrow \alpha\rangle\rangle\, \alpha\ E_{2(p-1)-1}\ \alpha\, \langle\downarrow \langle\updownarrow \alpha\rangle\, \alpha\, \langle\updownarrow \alpha\rangle\rangle\, \alpha\, \langle\updownarrow \alpha\rangle\rangle\,.$$

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 this $\downarrow$-expression has as an argument the $\uparrow$-expression $E_{2(p-2)}$. The outermost operator $\downarrow$ of $E_{2(p-1)-1}$ violates Property ($\mathcal{D}_{\mathrm{MinNF}}$.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 (9.1), which is in minimal normal form. For each substitution, we perform a constant amount of work: remove one occurrence of $\uparrow$, add one occurrence of $\downarrow$ 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 $|E_{2p}|$.

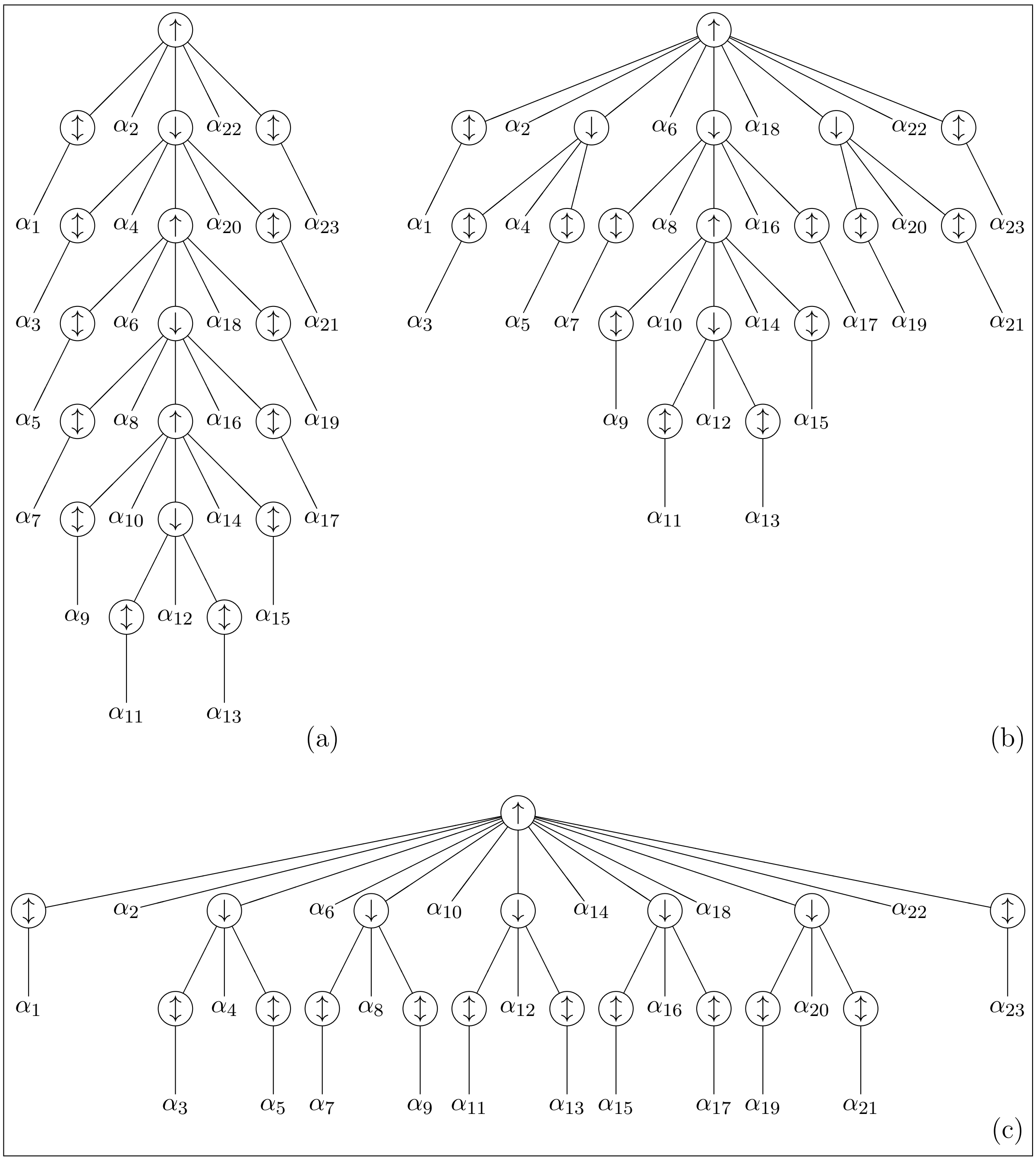
The effect of the while-loop on the $\downarrow$-expressions E_{2p+1} is analogous. ■

Indeed, for the $\uparrow$-expressions E_{2p} with $p \geq 3$ in the example, the substitution of a $\downarrow$-argument in line 14 of `NormalizeMinimal2` introduces a new $\downarrow$-argument with an $\uparrow$-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 9.7, we have depicted the effect of algorithhm `NormalizeMinimal2` for DNA expression E_6 from Example 9.7. In two steps (iterations of the while-loop), we transform the original, high tree in Figure 9.7(a) into the relatively flat tree in Figure 9.7(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 9.2 and 9.3).

The difference in complexity between `MakeMinimalNF` and `NormalizeMinimal2` is not just this factor of 2. 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 9.2 and Example 9.7. In Theorem 9.10, 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



(a)
(b)
(c)
$\uparrow$
$\downarrow$
$\updownarrow$
α_1 α_2 α_3 α_4 α_5 α_6 α_7 α_8 α_9 α_{10} α_{11} α_{12}
α_{13} α_{14} α_{15} α_{16} α_{17} α_{18} α_{19} α_{20} α_{21} α_{22} α_{23}

Figure 9.7: Structure trees of the three DNA expressions we successively obtain, when we apply algorithm `NormalizeMinimal2` to the $\uparrow$-expression E_6 from Example 9.7. 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 $\downarrow$-expression E_5, which has in turn the $\uparrow$-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 $\downarrow$-expression E_3, which has in turn the $\uparrow$-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.

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 9.6, where we established the correctness of `NormalizeMinimal`, because the while-loop in the algorithm has significantly changed, We can, however, reuse some elements of the argumentation.

Theorem 9.8 *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 `Nor-malizeMinimal2` 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 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 `Normalize-Minimal`. By Property (9.5) from the proof of Theorem 9.6, 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`:

E is a minimal $\uparrow$-expression with Property ($\mathcal{D}_{\text{MinNF}}$.5), satisfying $E \equiv E_2^*$, $\widehat{\varepsilon}$ is an argument of E and the arguments of E to the left of $\widehat{\varepsilon}$ do not contain any occurrence of $\uparrow$. (9.9)

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 (9.9) valid.

- Suppose that Property (9.9) 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 $\varepsilon_1, \ldots, \varepsilon_{i-1}, \varepsilon_{i+1}, \ldots, \varepsilon_n$, and $\varepsilon_{i,1}, \varepsilon_{i,2}, \ldots, \varepsilon_{i,m-1}, \varepsilon_{i,m}$, where $E_i = \langle\uparrow \varepsilon_{i,1}\varepsilon_{i,2} \ldots \varepsilon_{i,m-1}\varepsilon_{i,m}\rangle$ 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 6.17(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 6.2, 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 $\varepsilon_1, \ldots, \varepsilon_{i-1}$ 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

$$\langle\downarrow \varepsilon_1 \ldots \varepsilon_{i-1}\varepsilon_{i,1}\rangle\, \varepsilon_{i,2} \ldots \varepsilon_{i,m-1}\, \langle\downarrow \varepsilon_{i,m}\varepsilon_{i+1} \ldots \varepsilon_n\rangle\,.$$

This 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 9.6(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 $\langle\downarrow \varepsilon_1 \ldots \varepsilon_{i-1}\varepsilon_{i,1}\rangle$ of E, which precedes $\widehat{\varepsilon} = \varepsilon_{i,2}$, does not contain any occurrence of $\uparrow$. Hence, Property (9.9) 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 ($\mathcal{D}_{\text{MinNF}}$.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 6.2, 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 6.2 (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 (9.9) 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 (9.9) 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 (9.9) is also valid at the end of the iteration.

Indeed, Property (9.9) 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 $\uparrow$, anymore. In other words, the $\uparrow$-expression E does not contain any inner occurrence of $\uparrow$, anymore.

We can again reuse part of the proof of Theorem 9.6(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 limited. By Property (9.9), 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 limited by the length of E. Because this length is constant, the number of arguments to the left of $\widehat{\varepsilon}$ can only increase a limited number of times.

   Consequently, the number of iterations of the while-loop is limited.

This completes the proof of Theorem 9.8. □

Recall that in § 9.2, 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 § 7.1 and § 7.3, 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 9.9 *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 § 7.3, we used a datastructure 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.[2]

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 6.15. 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 datastructure and their usage are given in § 7.3. In particular, Figure 7.16 and Figure 7.18 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 `Normal-izeMinimal2` is not recursive. When we apply it to a minimal DNA expression E_2^*, we do not have multiple calls of the algorithm, for different arguments. This implies that the time (and space) required for passing the parameter of `NormalizeMinimal2` is not an issue in the analysis of its complexity.

Of course, if `NormalizeMinimal2` *had* been recursive, we could have established that the time (and space) required for passing its parameter for a single call is constant, just like we have done for `MakeMinimal` in § 7.3. As it is, however, we can simply ignore this aspect in the proofs below.

We now have

Theorem 9.10 *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 datastruture 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 $2p$

[2]In § 7.3, 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.

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^*.

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 excecuting 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.

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 datastructure, 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 5.3, $E_2^* = \langle\updownarrow \alpha_1\rangle$ 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 9.8(2), the number of iterations of the loop is finite, say it is N. As we have established in the proof of Theorem 9.8(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 9.8(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: $\langle\uparrow \widehat{\varepsilon} \ldots\rangle$, $\langle\uparrow \widehat{\varepsilon}_1\widehat{\varepsilon} \ldots\rangle$, $\langle\uparrow \widehat{\varepsilon}_1\widehat{\varepsilon}_2\widehat{\varepsilon} \ldots\rangle$, ..., $\langle\uparrow \widehat{\varepsilon}_1\widehat{\varepsilon}_2 \ldots \widehat{\varepsilon}_{N-1}\widehat{\varepsilon}\rangle$. 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 $\langle\downarrow \varepsilon_1 \ldots \varepsilon_n\rangle$ for some $n \geq 1$ and $\mathcal{N}$-words and DNA expressions $\varepsilon_1, \ldots, \varepsilon_n$, 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

  $$T_j \leq c_3 + c_4 \cdot n. \tag{9.10}$$

  In this case, $\widehat{\varepsilon}_j = \widehat{\varepsilon} = \langle\downarrow \varepsilon_1 \ldots \varepsilon_n\rangle$. By Lemma 6.17(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 (9.10), 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 $\varepsilon_i = E_i = \langle\uparrow \varepsilon_{i,1}\varepsilon_{i,2} \ldots \varepsilon_{i,m-1}\varepsilon_{i,m}\rangle$ for some $m \geq 1$ and $\mathcal{N}$-words and DNA expressions $\varepsilon_{i,1}, \varepsilon_{i,2}, \ldots, \varepsilon_{i,m-1}$, $\varepsilon_{i,m}$ 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

  $$T_j \leq c_3 + c_4 \cdot i. \tag{9.11}$$

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 $i \le |\widehat{\varepsilon}_j| - 6$. When we combine this with (9.11), we obtain

$$T_j \le c_3 + c_4 \cdot i \le c_3 + c_4 \cdot (|\widehat{\varepsilon}_j| - 6) = c_4 \cdot |\widehat{\varepsilon}_j| + c_3 - 6c_4.$$

In both subcases ($\widehat{\varepsilon}$ without or with an $\uparrow$-argument), we find that $T_j \le c_4 \cdot |\widehat{\varepsilon}_j| + c_3 - 6c_4$.

Now, if $c_3 \le 6c_4$, then

$$T_j \le c_4 \cdot |\widehat{\varepsilon}_j| \le c^* \cdot |\widehat{\varepsilon}_j|,$$

where the last inequality follows from $c^* \ge c_4$. If, on the other hand $c_3 > 6c_4$, which is equivalent to $c_4 < \frac{c_3}{6}$, then

$$T_j \le c_3 + c_4 \cdot (|\widehat{\varepsilon}_j| - 6) < c_3 + \frac{c_3}{6} \cdot (|\widehat{\varepsilon}_j| - 6) = \frac{c_3}{6} \cdot |\widehat{\varepsilon}_j| < c_3 \cdot |\widehat{\varepsilon}_j| \le c^* \cdot |\widehat{\varepsilon}_j|,$$

where the last inequality follows from $c^* \ge c_3$.

In each case, we have obtained that $T_j \le c^* \cdot |\widehat{\varepsilon}_j|$. Now, it is not difficult to derive an upper bound on $T_{\mathrm{NM2}}(E_2^*)$:

$$\begin{aligned} T_{\mathrm{NM2}}(E_2^*) &\le c_2 + T_1 + \dots + T_N \\ &\le c^* \cdot 3 + c^* \cdot |\widehat{\varepsilon}_1| + \dots + c^* \cdot |\widehat{\varepsilon}_N| \\ &= c^* \cdot |\, \langle \uparrow \widehat{\varepsilon}_1 \dots \widehat{\varepsilon}_N \rangle \,| = c^* \cdot |E_3^*| = c^* \cdot |E_2^*|, \end{aligned}$$

where the second inequality follows from $c^* \ge \frac{c_2}{3}$, 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 9.10. □

As part of Theorem 7.40, we established that the datastructure we propose to carry out the recursive function `MakeMinimal` efficiently, has linear size. For `NormalizeMinimal2`, we only use part of this datastructure: 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 datastructure, the space requirements depend on the DNA expression. We cannot reuse Example 7.39 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 9.11 Let α be an arbitrary $\mathcal{N}$-word, and let E_p be defined by

$$E_p = \left\langle \uparrow \underbrace{\langle \updownarrow \alpha \rangle \langle \updownarrow \alpha \rangle \dots \langle \updownarrow \alpha \rangle}_{p \text{ times}} \right\rangle \qquad (p \ge 2).$$

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 6.14(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 datastructure 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 datastructure 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 datastructure 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.[3] ■

We conclude

Theorem 9.12 *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. We can combine the complexities of the recursive function `MakeMinimal` and algorithm `NormalizeMinimal2` to find the complexity of the total two-step algorithm:

Theorem 9.13 *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 the naive, single-pass recursive function `MakeMinimalNF`. That function also yields the normal form version of its input, but, by Theorem 9.3, requires at least quadratic time in the worst case.

[3]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 datastructure 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 datastructure require linear space. We leave it to the reader to verify that the 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.

Proof: Let us apply the two-step algorithm to an arbitrary DNA expression E_1^*, and let us denote the result of `MakeMinimal` (the first step of the algorithm) by E_2^*.

By Corollary 7.38 and Theorem 7.40, `MakeMinimal` requires time and space which both are linear in the length $|E_1^*|$ of E_1^*. By Theorem 9.10 and Theorem 9.12, algorithm `NormalizeMinimal2` requires time and space which both are linear in the length $|E_2^*|$ of E_2^*. Because, by Theorem 7.17(2), E_2^* is a minimal DNA expression which is equivalent to E_1^*, $|E_2^*| \leq |E_1^*|$. This implies that the time and the space required by algorithm `NormalizeMinimal2` is at most linear in $|E_1^*|$. We conclude that the total time and the total space required by the composition of `MakeMinimal` and algorithm `NormalizeMinimal2` (i.e., by the two-step algorithm) are linear in $|E_1^*|$. $\square$

Chapter 10

Conclusions and directions for future research

We have introduced a (minimal) normal form for DNA expressions. This normal form is characterized by five syntactic properties, which are easy to check. We have described a two-step algorithm, which computes the normal form version of a given DNA expression. This is useful, e.g., to decide if two DNA expressions are equivalent. The algorithm first determines a minimal DNA expression that is equivalent to its input, and then rewrites this minimal DNA expression into the normal form. The algorithm is elegant, because it does not refer to the semantics of the DNA expression involved. It consists of string manipulations on the DNA expression itself. The algorithm requires linear time and space.

An important research line for the future could be to define and analyse new types of DNA expressions. These should be based on operators that directly model operations that are performed on real-world DNA. With new operators, one might also be able to represent DNA molecules with other 'imperfections' than nicks and gaps, e.g., DNA molecules with hairpin loops. It would certainly be a challenge to define DNA expressions that not only *denote* DNA molecules, but also implicitly describe how to synthesize them from the basic elements A, C, G and T.

Bibliography

L.M. Adleman: Molecular computation of solutions to combinatorial problems, *Science* **266** (1994), 1021-1024.

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* (R.J. Lipton, E.B. Baum, eds), American Mathematical Society, Providence, RI (1996), 37-66.

J. Chen, N. Jonoska, G. Rozenberg (eds): *Nanotechnology: Science and Computation*, Natural Computing Series, Springer, Berlin (2006).

N. Chomsky: A note on phrase structure grammars, *Information and Control*, **2**(4) (1959), 393-395.

R. Deaton, A. Suyama (eds): *DNA Computing and Molecular Programming – 15th International Conference, DNA 15, Fayetteville, AR, USA, June 8–11, 2009 – Revised Selected Papers*, Lecture Notes in Computer Science **5877**, Springer, Berlin (2009).

A. Ehrenfeucht, T. Harju, I. Petre, D.M. Prescott, G. Rozenberg: *Computation in Living Cells – Gene Assembly in Ciliates*, Springer, Berlin (2004).

C. Ferretti, G. Mauri, C. Zandron (eds): *DNA Computing – 10th International Workshop on DNA Computing, DNA10, Milan, Italy, June 7-10, 2004 – Revised Selected Papers*, Lecture Notes in Computer Science **3384**, Springer, Berlin (2005).

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.

L. Kari, S. Konstantinidis, P. Sosík: On properties of bond-free DNA languages, *Theoretical Computer Science* **334** (2005), 131-159.

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.

Gh. Păun, G. Rozenberg, A. Salomaa: *DNA Computing – New Computing Paradigms*, Springer, Berlin (1998).

J.H. Reif: The design of autonomous DNA nano-mechanical devices: Walking and rolling DNA, *Natural Computing* **2**(4) (2003), 439-461.

P.W.K. Rothemund: Folding DNA to create nanoscale shapes and patterns, *Nature* **440**, (2006) 297-302.

Y. Sakakibara, Y. Mi (eds): *DNA Computing and Molecular Programming – 16th International Conference, DNA 16, Hong Kong, China, June 14–17, 2010 – Revised Selected Papers*, Lecture Notes in Computer Science **6518**, Springer, Berlin (2011).

R. van Vliet: *Combinatorial Aspects of Minimal DNA Expressions (ext.)*, Technical Report 2004-03, Leiden Institute of Advanced Computer Science, Leiden University (2004), see `http://www.liacs.nl/home/rvvliet/dnaexpressions/`.

R. van Vliet, H.J. Hoogeboom, G. Rozenberg: Combinatorial aspects of minimal DNA expressions, [Ferretti et al., 2005] (2005), 375-388.

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

E. Winfree: DNA computing by self-assembly, *The Bridge* **33**(4) (2003), 31-38.

List of Symbols

symbol	introduced on page
$\longrightarrow$	7
$\Longrightarrow$	7
$\overset{i,j}{\Longrightarrow}$	23
$\underline{\sqsubseteq}$	12
$\overline{\sqsubset}$	12
$\sqsubset$	12
$\equiv$	25
$\underset{\triangledown}{\equiv}$	25
${}_{\triangledown}{\equiv}$	26
$\equiv_{\triangledown}$	26
$\uparrow$	13, 14
$\downarrow$	13, 14
$\updownarrow$	13, 14
$\triangledown$	10
$\vartriangle$	10
$\#_a(X)$	4
$\#_{a,b}(X)$	4
1_K	4
α, α_i	3
α_E	35, 86
a, a_i	3
$\mathcal{A}$	10
$\|X\|_{\mathcal{A}}$	37
$\mathcal{A}_{\pm}$	10
$\mathcal{A}_{+}$	10
$\mathcal{A}_{-}$	10
$\mathcal{A}_{\triangledown\vartriangle}$	10
$B_{\uparrow}(X)$	39
$B_{\downarrow}(X)$	39
$c(a)$	5
$\delta(E)$	135
Δ	7
$\mathcal{D}$	13
$\mathcal{D}_{\mathrm{Min}}$	62
$(\mathcal{D}_{\mathrm{Min}}.1)$–$(\mathcal{D}_{\mathrm{Min}}.6)$	61
$\mathcal{D}_{\mathrm{MinNF}}$	147
$(\mathcal{D}_{\mathrm{MinNF}}.1)$–$(\mathcal{D}_{\mathrm{MinNF}}.5)$	144
$\varepsilon, \varepsilon_i$	13
E, E_i	13
$E(\alpha_1, \ldots, \alpha_k)$	17
E_i^*	72
$E_{\mathrm{MinNF}}(X)$	141
$\mathcal{F}$	10
G_1	20
G_2	151, 154
$\kappa(X)$	13
$\mathcal{K}$	3
λ	3
$L(X)$	3
$\mathcal{L}(G)$	7
$\mathcal{L}(X)$	7
$\mathcal{L}_G(X)$	7
$\nu(X)$	12
$\nu^+(X)$	12
$\nu^-(X)$	12
$n_{\uparrow\downarrow}(X)$	122
$n_{\updownarrow}(X)$	39
$n_{\alpha}(E)$	122
$n_{\alpha\uparrow\downarrow}(E)$	122
$n_{\alpha\updownarrow}(E)$	122
$n_{\mathrm{imus}}(X)$	42
$n_{\mathrm{iter}}(E)$	128
$n_{\mathrm{min}}(X)$	65
$n_{\mathrm{min}\uparrow}(X)$	65
$n_{\mathrm{min}\downarrow}(X)$	65
$n_{\mathrm{min}\updownarrow}(X)$	65
$n_{\mathrm{mus}}(X)$	42
$n_{\mathcal{N}\uparrow\downarrow}(E)$	122
$n_{\mathrm{opermin}\uparrow}(X)$	65
$n_{\mathrm{opermin}\downarrow}(X)$	65
$n_{\mathrm{opermin}\updownarrow}(X)$	65
$\mathcal{N}$	3
$\mathcal{O}(\|E\|)$	9
$\mathrm{Op}(E)$	135
P	7
$\mathcal{P}$	45
$R(X)$	3
xRy	25
Σ	7
$\Sigma_{\mathcal{D}}$	13
Σ^*	3
Σ^+	3
S	7
$\mathcal{S}(E)$	13
$\mathcal{S}^+(\varepsilon)$	15

$\mathcal{S}^-(\varepsilon)$ 15
$T_{\mathrm{CM}}(E)$ 129
$T_{\mathrm{Dni}}(E)$ 137
$T_{\mathrm{M}\updownarrow\mathrm{M}}(E)$ 135
$T_{\mathrm{NM2}}(E)$ 187
$|X|$ 3
$\overline{X}_j$ 44
$(X^s)^k$ 4

Index

term — introduced on page

↑-argument ... 17
↑-block
  primitive ∼ ... 37
↑-component ... 37
↑-expression ... 16
  alternating ∼ ... 17
↑-subexpression ... 17
↓-argument ... 17
↓-block
  primitive ∼ ... 38
↓-component ... 37
↓-expression ... 16
  alternating ∼ ... 17
↓-subexpression ... 17
↕-argument ... 17
  non-∼ ... 17, 120
↕-expression ... 16
↕-subexpression ... 17
$\mathcal{A}$-letter ... 10
  double ∼ ... 10
  lower ∼ ... 10
  upper ∼ ... 10
algorithm ... 9
  ∼ for minimality ... 68
  recursive ∼
    for minimal normal form ... 168
  two-step ∼
    for minimal normal form ... 171
alphabet ... 3
alternating
  ∼ ↑- or ↓-expression ... 17
  ∼ occurrence of ↑ or ↓ ... 17
ancestor
  ∼ of node ... 5
  ∼ operator ... 17
apply
  ∼ operator ... 13
  ∼ production ... 7
argument
  ↑-∼ ... 17
  ↓-∼ ... 17
  ↕-∼ ... 17
  ∼ of DNA expression ... 16
  ∼ of operator ... 13
  expression-∼ ... 17
  $\mathcal{N}$-word-∼ ... 17
$\mathcal{A}$-word ... 10
  double ∼ ... 10
  lower ∼ ... 10
  upper ∼ ... 10
axiom ... 7
base pair ... 10
  complete ∼ ... 10
big O notation ... 9
binary
  ∼ relation ... 25
    reflexive ∼ ... 25
    symmetric ∼ ... 25
    transitive ∼ ... 25
block
  lower ∼ ... 44
    ∼ partitioning ... 44
    primitive ∼ ... 41
      ∼ partitioning ... 41
  primitive ↑-∼ ... 37
  primitive ↓-∼ ... 38
  primitive ∼ ... 38
  upper ∼ ... 44
    ∼ partitioning ... 45
    primitive ∼ ... 41
      ∼ partitioning ... 42
characterization of minimal
  ∼ DNA expressions ... 62
  ∼ normal form ... 144, 149
child of node ... 5
closing bracket of operator ... 13
complement function ... 5
complete base pair ... 10
complexity
  linear ∼ ... 9
  quadratic ∼ ... 9
  space ∼ ... 9
    ∼ of `MakeMinimal` ... 132
    ∼ of `NormalizeMinimal2` ... 190
    ∼ of two-step algorithm

for minimal normal form . 190
time ∼ . . . 9
∼ of `MakeMinimal` . . . 114, 129
∼ of `MakeMinimalNF` . . . 171
∼ of `NormalizeMinimal2` . . . 186
∼ of two-step algorithm
for minimal normal form . 190
component
∼ of formal DNA molecule . . . 11
↑-∼ . . . 37
↓-∼ . . . 37
double ∼ . . . 11
lower ∼ . . . 11
non-double ∼ . . . 11
single-stranded ∼ . . . 11
upper ∼ . . . 11
concatenation
∼ of strings . . . 3
∼ of words . . . 3
contain
substring ∼s other substring . . . 4
context-free
∼ grammar . . . 7
axiom of ∼ . . . 7
∼ for $\mathcal{D}$ . . . 20
∼ for $\mathcal{D}_{\text{MinNF}}$ . . . 151
derivation in ∼ . . . 7
leftmost ∼ . . . 8
language generated by ∼ . . . 7
language of ∼ . . . 7
non-terminal symbol of ∼ . . . 7
productions of ∼ . . . 7
self-embedding ∼ . . . 9, 165
sentential forms of ∼ . . . 7
terminal symbol of ∼ . . . 7
∼ language . . . 7
$\mathcal{D}$ is ∼ . . . 20, 23
counting function . . . 39
cover to the left/right . . . 12
strictly ∼ . . . 12
datastructure
for `MakeMinimal` . . . 116
for `NormalizeMinimal2` . . . 185
decomposition of formal
DNA molecule . . . 11
nick free ∼ . . . 52
`Denickify` . . . 94
derivation . . . 7
leftmost ∼ . . . 8
descendant of node . . . 5
directed
∼ tree . . . 5
heigth of ∼ . . . 6
level of ∼ . . . 6
ordered, ∼ . . . 6
ordered, ∼,
node-labelled tree . . . 6
disjoint substrings . . . 4
distance between nodes in tree . . . 5
DNA computing . . . 1
DNA expression . . . 1, 13, 15
argument of ∼ . . . 16
equivalent ∼s . . . 25
length of ∼ . . . 37
level of ∼ . . . 18
minimal ∼ . . . 41, 41, 46, 55
algorithm for ∼ . . . 68
characterization of ∼ . . . 62
number of ∼s . . . 65
operator-minimal ∼ . . . 52, 54
number of ∼s . . . 65
semantics of ∼ . . . 13
structure tree of ∼ . . . 23
DNA molecule . . . 1
formal ∼ . . . 10
DNA subexpression . . . 16
proper ∼ . . . 16
DNA submolecule
formal ∼ . . . 11
double
∼ $\mathcal{A}$-letter . . . 10
∼ $\mathcal{A}$-word . . . 10
∼ complete formal
DNA molecule . . . 38
∼ component of formal
DNA molecule . . . 11
empty string . . . 3
endomorphism . . . 5
equivalence relation . . . 25
equivalent DNA expressions . . . 25
∼ modulo nicks . . . 25
∼ post-modulo nicks . . . 26
∼ pre-modulo nicks . . . 26
strictly ∼ . . . 25
expressible formal
DNA molecule . . . 27
expression
↑-∼ . . . 16
↓-∼ . . . 16
↕-∼ . . . 16
∼-argument . . . 17
fit together . . . 12
formal
∼ DNA molecule . . . 10

- component of $\sim$ 11
- decomposition of $\sim$ 11
- double-complete $\sim$ 38
- expressible $\sim$ 27
- lower strand of $\sim$ 11
- upper strand of $\sim$ 11
- $\sim$ DNA submolecule 11

gap 1
govern 16
grammar 6
- context-free $\sim$ 7
  - axiom of $\sim$ 7
  - $\sim$ for $\mathcal{D}$ 20
  - $\sim$ for $\mathcal{D}_{\mathrm{MinNF}}$ 151
  - derivation in $\sim$ 7
    - leftmost $\sim$ 8
  - language generated by $\sim$ 7
  - language of $\sim$ 7
  - non-terminal symbol of $\sim$ 7
  - productions of $\sim$ 7
  - self-embedding $\sim$ 9, 165
  - sentential forms of $\sim$ 7
  - terminal symbol of $\sim$ 7
- right-linear $\sim$ 8
  - $\sim$ for $\mathcal{F}$ 11

height
- $\sim$ of directed tree 6
- $\sim$ of structure tree 24
  - $\sim$ in minimal normal form 150

homomorphism 4
identity of set with operation 4
include
- substring $\sim$s other substring 4

inner occurrence of operator 16
internal
- $\sim$ maximal lower sequence 42
- $\sim$ maximal upper sequence 42
- $\sim$ node 5

intersecting substrings 4
labelled
- ordered, directed,
  - node-$\sim$ tree 6

language 3
- context-free $\sim$ 7
  - $\mathcal{D}$ is $\sim$ 20, 23
- $\sim$ generated by
  - context-free grammar 7
- $\sim$ of context-free grammar 7
- regular $\sim$ 8
  - $\mathcal{D}$ is no $\sim$ 20
  - $\mathcal{D}_{\mathrm{Min}}$ is no $\sim$ 51
  - $\mathcal{D}_{\mathrm{MinNF}}$ is $\sim$ 151, 166
  - $\mathcal{F}$ is $\sim$ 11
  - pumping lemma for $\sim$s 20, 52

leaf in tree 5
leftmost derivation 8
length
- of DNA expression 37
  - lower bound on $\sim$ 39
- of string 3

letter 3
level
- $\sim$ of DNA expression 18
- $\sim$ of directed tree 6
- nesting $\sim$ 18
  - maximal $\sim$ 18

linear complexity 9
linked list 116, 185
lower
- $\sim$ $\mathcal{A}$-letter 10
- $\sim$ $\mathcal{A}$-word 10
- $\sim$ block 44
  - $\sim$ partitioning 44
  - primitive $\sim$ 41
    - $\sim$ partitioning 41
- $\sim$ bound on length of
  - DNA expression 39
- $\sim$ component of formal
  - DNA molecule 11
- $\sim$ nick letter 10
- $\sim$ postfit 12
- $\sim$ prefit 12
- $\sim$ strand 11
- maximal $\sim$
  - $\sim$ prefix 42
  - $\sim$ sequence 42
    - internal $\sim$ 42
  - $\sim$ suffix 42

`Make`$\updownarrow$`ExprMinimal` 86
`MakeMinimal` 70, 115
- datastructure for $\sim$ 116
- space complexity of $\sim$ 132
- time complexity of $\sim$ 114, 129

`MakeMinimalNF` 168
- time complexity of $\sim$ 171

maximal
- $\sim$ lower
  - $\sim$ prefix 42
  - $\sim$ sequence 42
    - internal $\sim$ 42
  - $\sim$ suffix 42
- $\sim$ $\mathcal{N}$-word occurrence 16
- $\sim$ nesting level
  - of DNA expression 18

~ in minimal normal form . 149
~ upper
~ prefix 42
~ sequence 42
internal ~ 42
~ suffix 42
minimal
~ DNA expression 41, 41, 46, 55
algorithm for ~ 68
characterization of ~ 62
number of ~s 65
operator-~ 52, 54
number of ~s 65
~ normal form 140, 141
characterization of ~ . . 144, 149
recursive algorithm for ~ . . . 168
structure tree in ~ 150
two-step algorithm for ~ . . . 171
~ structure tree 63
modulo
equivalent ~ nicks 25
nesting level of DNA expression . . . 18
maximal ~ 18
~ in minimal normal form . 149
nick 1
equivalent modulo ~s 25
equivalent post-modulo ~s 26
equivalent pre-modulo ~s 26
~ free 11
~ decomposition 52
~ letter 10
lower ~ 10
upper ~ 10
$\mathcal{N}$-letter 3
node
ancestor of ~ 5
child of ~ 5
descendant of ~ 5
distance between ~s 5
internal ~ 5
ordered, directed, ~-labelled tree 6
parent of ~ 5
non-$\updownarrow$-argument 17, 120
non-double component 11
non-root 5
non-terminal symbol 7
self-embedding ~ 9, 165
normal form 140
minimal ~ 140, 141
characterization of ~ . . 144, 149
recursive algorithm for ~ . . . 168
structure tree in ~ 150
two-step algorithm for ~ . . . 171
`NormalizeMinimal` 174
`NormalizeMinimal2` 180
datastructure for ~ 185
space complexity of ~ 190
time complexity of ~ 186
notation
big O ~ 9
nucleotide 1
number
~ of occurrences of operators in (operator-)minimal DNA expressions 61
~ of minimal DNA expressions 65
~ of operator-minimal $\uparrow$-/$\downarrow$-expressions 65
$\mathcal{N}$-word 3
concatenation of ~s 3
maximal ~ occurrence 16
~-argument 17
occurrence
alternating ~ of $\uparrow$ or $\downarrow$ 17
inner ~ of operator 16
maximal $\mathcal{N}$-word ~ 16
number of ~s of operators in (operator-)minimal DNA expressions 61
~ of substring 3
preceding ~ 3
opening bracket of operator 13
operator 13
ancestor ~ 17
apply ~ 13
argument of ~ 13
closing bracket of ~ 13
inner occurrence of ~ 16
number of occurrences of ~s in (operator-)minimal DNA expressions 61
opening bracket of ~ 13
~-minimal DNA expression 52, 54
number of ~s 65
outermost ~ 16
parent ~ 17
scope of ~ 13
ordered, directed
~ tree 6
~, node-labelled tree 6

outermost operator . . . 16
overlapping substrings . . . 4
parent
  ~ of node . . . 5
  ~ operator . . . 17
partitioning
  lower block ~ . . . 44
    primitive ~ . . . 41
  upper block ~ . . . 45
    primitive ~ . . . 42
phosphodiester bond . . . 1
postfit
  lower ~ . . . 12
  ~ by lower strands . . . 12
  ~ by upper strands . . . 12
  upper ~ . . . 12
post-modulo
  equivalent ~ nicks . . . 26
preceding occurrence of substring . . 3
prefit . . . 12
  lower ~ . . . 12
  ~ by lower strands . . . 12
  ~ by upper strands . . . 12
  upper ~ . . . 12
prefix
  maximal lower ~ . . . 42
  maximal upper ~ . . . 42
  ~ of string . . . 3
    proper ~ . . . 3
pre-modulo
  equivalent ~ nicks . . . 26
primitive
  ~ $\uparrow$-block . . . 37
  ~ $\downarrow$-block . . . 38
  ~ block . . . 38
  ~ lower block . . . 41
    ~ partitioning . . . 41
  ~ upper block . . . 41
    ~ partitioning . . . 42
production . . . 7
  apply ~ . . . 7
proper
  ~ DNA subexpression . . . 16
  ~ prefix . . . 3
  ~ substring . . . 3
  ~ suffix . . . 3
pumping lemma
  for regular languages . . . 20, 52
quadratic complexity . . . 9
recursive function
  ~ `MakeMinimal` . . . 70, 115
  ~ `MakeMinimalNF` . . . 168
reflexive binary relation . . . 25
regular language . . . 8
  $\mathcal{D}$ is no ~ . . . 20
  $\mathcal{D}_{\text{Min}}$ is no ~ . . . 51
  $\mathcal{D}_{\text{MinNF}}$ is ~ . . . 151, 166
  $\mathcal{F}$ is ~ . . . 11
  pumping lemma for ~s . . . 20, 52
relation
  binary ~ . . . 25
    reflexive ~ . . . 25
    symmetric ~ . . . 25
    transitive ~ . . . 25
  equivalence ~ . . . 25
rewriting rule . . . 7
right-linear grammar . . . 8
  ~ for $\mathcal{F}$ . . . 11
root of tree . . . 5
`RotateToMinimal` . . . 102
rotation in tree . . . 29
scope of operator . . . 13
self-embedding
  ~ context-free grammar . . . 9, 165
  ~ non-terminal symbol . . . 9, 165
semantics of DNA expression . . . 13
sentential form . . . 7
single-stranded component of
  formal DNA molecule . . . 11
space complexity . . . 9
  ~ of two-step algorithm
    for minimal normal form . . . 190
  ~ of `MakeMinimal` . . . 132
  ~ of `NormalizeMinimal2` . . . 190
strand
  lower ~ . . . 11
  upper ~ . . . 11
strictly
  ~ cover to the left/right . . . 12
  ~ equivalent
    DNA expressions . . . 25
string . . . 3
  empty ~ . . . 3
  length of ~ . . . 3
structure tree . . . 23
  height of ~ . . . 24
    ~ in minimal normal form . 150
  minimal ~ . . . 63
  ~ in minimal normal form . . . 150
subexpression
  $\uparrow$-~ . . . 17
  $\downarrow$-~ . . . 17
  $\updownarrow$-~ . . . 17
substring . . . 3

disjoint ~s 4
intersecting ~s 4
occurrence of ~ 3
overlapping ~s 4
proper ~ 3
~ contains other ~ 4
~ includes other ~ 4
subtree rooted in node 5
subword
maximal ~ 16
suffix
maximal lower ~ 42
maximal upper ~ 42
~ of string 3
proper ~ 3
symbol 3
symmetric binary relation 25
terminal symbol 7
non-~ 7
self-embedding ~ 9, 165
time complexity 9
~ of two-step algorithm for minimal normal form . . . 190
~ of `MakeMinimal` 114, 129
~ of `MakeMinimalNF` 171
~ of `NormalizeMinimal2` 186
transitive binary relation 25
tree 5
directed ~ 5
height of ~ 6
level of ~ 6
ordered, ~ 6
leaf in ~ 5
ordered, directed, node-labelled ~ 6
root of ~ 5
rotation in ~ 29
structure ~ 23
minimal ~ 63
~ in minimal normal form 150
two-step algorithm for
minimal normal form 171
space complexity of ~ 190
time complexity of ~ 190
upper
maximal ~
~ prefix 42
~ sequence 42
internal ~ 42
~ suffix 42
~ $\mathcal{A}$-letter 10
~ $\mathcal{A}$-word 10
~ block 44
~ partitioning 45
primitive ~ 41
~ partitioning 42
~ component of formal DNA molecule 11
~ nick letter 10
~ postfit 12
~ prefit 12
~ strand 11