Models of natural computation : gene assembly and membrane systems

Brijder, R.

Part I

Gene Assembly in Ciliates

Chapter 2

Reducibility of Gene Patterns in Ciliates using the Breakpoint Graph

Abstract
Gene assembly in ciliates is one of the most involved DNA processings going on in any organism. This process transforms one nucleus (the micronucleus) into another functionally different nucleus (the macronucleus). We continue the development of the theoretical models of gene assembly, and in particular we demonstrate the use of the concept of the breakpoint graph, known from another branch of DNA transformation research. More specifically: (1) we characterize the *intermediate* gene patterns that can occur during the transformation of a *given* micronuclear gene pattern to its macronuclear form; (2) we determine the number of applications of the loop recombination operation (the most basic of the three molecular operations that accomplish gene assembly) needed in this transformation; (3) we generalize previous results (and give elegant alternatives for some proofs) concerning characterizations of the micronuclear gene patterns that can be assembled using a specific subset of the three molecular operations.

2.1 Introduction

Ciliates are single cell organisms that have two functionally different nuclei, one called micronucleus and the other called macronucleus (both of which can occur in various multiplicities). At some stage in sexual reproduction a micronucleus is transformed into a macronucleus in a process called gene assembly. This is the most involved DNA processing in living organisms known today. The reason that gene assembly is so involved is that the genome of the micronucleus may be dramatically different from the genome of the macronucleus — this is particularly

true in the stichotrichs group of ciliates, which we consider in this chapter. The investigation of gene assembly turns out to be very exciting from both biological and computational points of view.

Another research area concerned with transformations of DNA is *sorting by reversal*, see, e.g., [23, 21, 1]. Two different species can have several contiguous segments in their genome that are very similar, although their relative order (and orientation) may differ in both genomes. In the theory of sorting by reversal one tries to determine the number of operations needed to reorder such a series of genomic 'blocks' from one species into that of another. An essential tool is the *breakpoint graph* (or reality and desire diagram) which is used to capture both the present situation, the genome of the first species, and the desired situation, the genome of the second species.

Motivated by the breakpoint graph, we introduce the notion of *reduction graph* into the theory of gene assembly. The intuition of 'reality and desire' remains in place, but the technical details are different. Instead of one operation, the reversal, we have three operations. Furthermore, these operations are irreversible and can only be applied on special positions in the string, called *pointers*. Also, instead of two different species, we deal with two different nuclei — the reality is a gene in its micronuclear form, and desire is the same gene but in its macronuclear form. Surprisingly, where the breakpoint graph in the theory of sorting by reversal is mostly useful to determine the number of needed operations, the reduction graph has different uses in the theory of gene assembly, providing valuable insights into the gene assembly process. Adapted from the theory of sorting by reversal, and applied to the theory of gene assembly in ciliates, we hope the reduction graph can serve as a 'missing link' to connect the two fields.

For example, the reduction graph allows for a direct characterization of the *intermediate* strings that may be constructed during the transformation of a given gene from its micronuclear form to its macronuclear form (Theorem 11). Also, it makes the number of loop recombination operations (see Figure 2.3 below) needed in this transformation quite explicit as the number of cyclic (connected) components in the reduction graph (Theorem 18).

Each micronuclear form of a gene defines a sequence of (oriented) segments, the boundaries of which define the pointers where splicing takes place. In abstract representation, the gene defines a so-called *realistic* string in which every pointer is denoted by a single symbol. Each pointer occurs twice (up to inversion) in that string. Not every string in which each symbol has two occurrences (up to inversion) can be obtained as the representation of a micronuclear gene. Our results are obtained in the larger context, i.e., they are not only valid for realistic strings, but for *legal* strings in general.

The chapter is organized as follows. In Section 2.2 we briefly discuss the basics of gene assembly in ciliates, and describe three molecular operations stipulated to accomplish gene assembly. The reader is referred to monograph [12] for more background information. In Section 2.3 we recall some basic notions and notation concerning strings and graphs, and then in Section 2.4 we recall the string

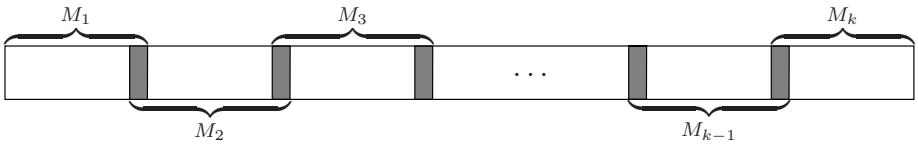


Figure 2.1: The MAC form of genes.

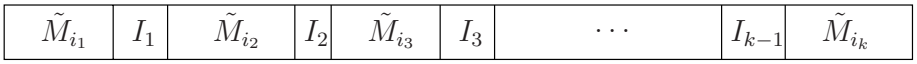


Figure 2.2: The MIC form of genes.

pointer reduction system, which is a formal model of gene assembly. This model is used throughout the rest of this chapter. In Section 2.5 we introduce the operation of pointer removal, which forms a useful formal tool in this chapter. Then in Sections 2.6 and 2.7 we introduce our main construct, the reduction graph, and discuss the transformations of it that correspond to the three molecular operations. In Section 2.8 we provide a characterization of intermediate forms of a gene resulting from its assembly to the macronuclear form — then, in Section 2.9 we determine the number of loop recombination operations required in this assembly. As an application of this last result, in Section 2.10 we generalize some well-known results from [13] (and Chapter 13 in [12]) as well as give elegant alternatives for these proofs. A conference edition of this chapter, containing selected results without proofs, was presented at CompLife [5].

2.2 Background: Gene Assembly in Ciliates

This section discusses the biological origin for the string pointer reduction system, the formal model we discuss in Section 2.4 and use throughout this chapter. Let us recall that the *inversion* of a double stranded DNA sequence M, denoted by $\bar{M}$, is the point rotation of M by 180 degrees. For example, if $M = \begin{matrix} GACGT \\ CTGCA \end{matrix}$, then $\bar{M} = \begin{matrix} ACGTC \\ TGCAG \end{matrix}$.

Ciliates are unicellular organisms (eukaryotes) that have two kinds of functionally different nuclei: the micronucleus (MIC) and the macronucleus (MAC). All the genes occur in both MIC and MAC, but in very different forms. For a given individual gene (in given species) the relationship between its MAC and MIC form can be described as follows.

The MAC form G of a given gene can be represented as the sequence M_1, M_2, ..., M_k of overlapping segments (called MDSs) which form G in the way shown in Figure 2.1 (where the overlaps are given by the shaded areas). The MIC form g of the same gene is formed by a specific permutation $M_{i_1}, \ldots, M_{i_k}$ of $M_1, \ldots, M_k$ in the way shown in Figure 2.2, where $I_1, I_2, \ldots, I_{k-1}$ are segments of DNA (called

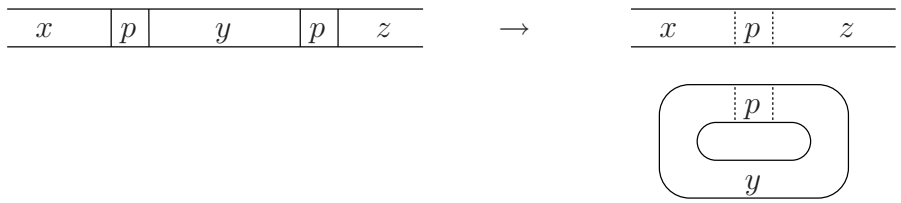


Figure 2.3: The loop recombination operation.

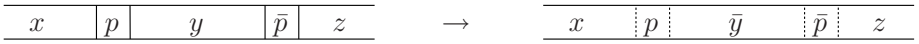


Figure 2.4: The hairpin recombination operation.

IESs) inserted in-between segments $\tilde{M}_{i_1}, \ldots, \tilde{M}_{i_k}$ with each $\tilde{M}_i$ equal to either M_i or $\bar{M}_i$ (the inversion of M_i). As clear from Figure 2.1, each MDS M_i except for M_1 and M_k (the first and the last one) begins with the overlap with M_{i-1} and ends with the overlap with M_{i+1} — these overlap areas are called pointers; the former is the incoming pointer of M_i denoted by p_i, and the latter is the outgoing pointer of M_i denoted by p_{i+1}. Then M_1 has only the outgoing pointer p_2, and M_k has only the incoming pointer p_k.

The MAC is the (standard eukaryotic) 'household' nucleus that provides RNA transcripts for the expression of proteins — hence MAC genes are functional expressible genes. On the other hand the MIC is a dormant nucleus where no production of RNA transcripts occurs. As a matter of fact MIC becomes active only during sexual reproduction. Within a part of sexual reproduction in a process called *gene assembly*, MIC genes are transformed into MAC genes (as MIC is transformed into MAC). In this transformation the IESs from the MIC gene g (see Figure 2.2) must be excised and the MDSs must be spliced (overlapping on pointers) in their order $M_1, \ldots, M_k$ to form the MAC gene G (see Figure 2.1).

The gene assembly process is accomplished through the following three molecular operations, which through iterative applications beginning with the MIC form g of a gene, and going through intermediate forms, lead to the formation of the MAC form G of the gene.

Loop recombination The effect of the loop recombination operation is illustrated in Figure 2.3. The operation is applicable to a gene pattern (i.e., MIC or an intermediate form of a gene) which has two identical pointers p, p separated by a single IES y. The application of this operation results in the excision from the DNA molecule of a circular molecule consisting of y (and a copy of the involved pointer) only.

Hairpin recombination The effect of the hairpin recombination operation is

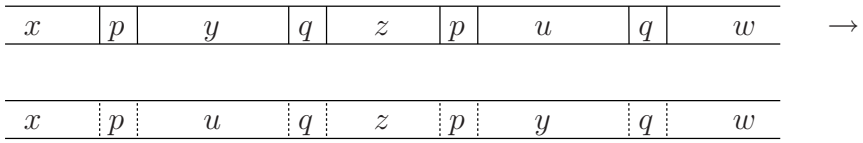


Figure 2.5: The double-loop recombination operation.

illustrated in Figure 2.4. The operation is applicable to a gene pattern containing a pair of pointers p, $\bar{p}$ in which one pointer is an inversion of the other. The application of this operation results in the inversion of the DNA molecule segment that is contained between the mentioned pair of pointers.

Double-loop recombination The effect of the double-loop recombination operation is illustrated in Figure 2.5. The operation is applicable to a gene pattern containing two identical pairs of pointers for which the segment of the molecule between the first pair of pointers overlaps with the segment of the molecule between the second pair of pointers. The application of this operation results in interchanging the segment of the molecule between the first two (of the four) pointers in the gene pattern and the segment of the molecule between the last two (of the four) pointers in the gene pattern.

For a given MIC gene g, a sequence of (applications of) these molecular operations is *successful* if it transforms g into its MAC form G. The gluing of MDS M_j with MDS M_{j+1} on the common pointer p_{j+1} results in a composite MDS. This means that after gluing, the outgoing pointer of M_j and the incoming pointer of M_{j+1} are not pointers anymore, because pointers are always positioned on the boundary of MDSs (hence they are adjacent to IESs). Therefore, the molecular operations can be seen as operations that remove pointers. This is an important property of gene assembly which is crucial in the formal models of the gene assembly process (see [12]).

2.3 Basic Notions and Notation

In this section we recall some basic notions concerning functions, strings, and graphs. We do this mainly to set up the basic notation and terminology for this chapter.

The empty set will be denoted by $\varnothing$. The composition of functions $f : X \to Y$ and $g : Y \to Z$ is the function $gf : X \to Z$ such that $(gf)(x) = g(f(x))$ for every $x \in X$. The restriction of f to a subset A of X is denoted by $f|A$.

We will use λ to denote the empty string. For strings u and v, we say that v is a *substring of* u if $u = w_1 v w_2$, for some strings w_1, w_2; we also say that v *occurs in* u. For a string $x = x_1 x_2 \dots x_n$ over Σ with $x_1, x_2, \dots, x_n \in \Sigma$, we say

that substrings $x_{i_1} \cdots x_{j_1}$ and $x_{i_2} \cdots x_{j_2}$ of x *overlap in* x if $i_1 < i_2 < j_1 < j_2$ or $i_2 < i_1 < j_2 < j_1$.

For alphabets Σ and Δ, a *homomorphism* is a function $\varphi : \Sigma^* \to \Delta^*$ such that $\varphi(xy) = \varphi(x)\varphi(y)$ and for all $x, y \in \Sigma^*$. Let $\varphi : \Sigma^* \to \Delta^*$ be a homomorphism. If there is a $\Gamma \subseteq \Sigma$ such that

$$\varphi(x) = \begin{cases} x & x \notin \Gamma \\ \lambda & x \in \Gamma \end{cases},$$

then φ is denoted by $erase_\Gamma$.

We move now to graphs. A *labelled graph* is a 4-tuple $G = (V, E, f, \Psi)$, where V is a finite set, Ψ is an alphabet, E is a finite subset of $V \times \Psi^* \times V$, and $f : D \to \Gamma$, for some $D \subseteq V$ and some alphabet Γ, is a partial function on V. The elements of V are called *vertices*, and the elements of E are called *edges*. Function f is the *vertex labelling function*, the elements of Γ are the *vertex labels*, and the elements of Ψ^* are the *edge labels*.

For $e = (x, u, y) \in V \times \Psi^* \times V$, x is called the *initial vertex* of e, denoted by $\iota(e)$, y is called the *terminal vertex* of e, denoted by $\tau(e)$, and u is called the *label* of e, denoted by $\ell(e)$. Labelled graph $G' = (V', E', f|V', \Psi)$ is an *induced subgraph of* G if $V' \subseteq V$ and $E' = E \cap (V' \times \Psi^* \times V')$. We also say that G' is the *subgraph of* G *induced by* V'.

A *walk in* G is a string $\pi = e_1 e_2 \cdots e_n$ over E with $n \geq 1$ such that $\tau(e_i) = \iota(e_{i+1})$ for $1 \leq i < n$. The *label of* π is the string $\ell(\pi) = \ell(e_1)\ell(e_2)\cdots\ell(e_n)$. Vertex $\iota(e_1)$ is called the *initial vertex of* π, denoted by $\iota(\pi)$, vertex $\tau(e_n)$ is called the *terminal vertex of* π, denoted by $\tau(\pi)$ and we say that π is a *walk between* $\iota(\pi)$ *and* $\tau(\pi)$ (or that π is a *walk from* $\iota(\pi)$ *to* $\tau(\pi)$). We say that G is *weakly connected* if for every two vertices v_1 and v_2 of G with $v_2 \neq v_1$, there is string $e_1 e_2 \cdots e_n$ over $E \cup \{(\tau(e), \ell(e), \iota(e)) \mid e \in E\}$ with $n \geq 1$, $\iota(e_1) = v_1$, $\tau(e_n) = v_2$, and $\tau(e_i) = \iota(e_{i+1})$ for $1 \leq i < n$. A subgraph H of G induced by $V_H \subseteq V$ is a *component of* G if H is weakly connected, and for every edge $e \in E$ either $\iota(e), \tau(e) \in V_H$ or $\iota(e), \tau(e) \in V \backslash V_H$.

The isomorphism between two labelled graphs is defined in the usual way. Two labelled graphs $G = (V, E, f, \Psi)$ and $G' = (V', E', f', \Psi)$ are *isomorphic*, denoted by $G \approx G'$, if there is a bijection $\alpha : V \to V'$ such that $f(v) = f'(\alpha(v))$ for all $v \in V$, and

$$(x, u, y) \in E \text{ iff } (\alpha(x), u, \alpha(y)) \in E',$$

for all $x, y \in V$ and $u \in \Psi^*$. The bijection α is then called an *isomorphism from* G *to* G'.

In this chapter we will consider walks in labelled graphs that often originate in a fixed source vertex and will end in a fixed target vertex. Therefore, we need the following notion.

A *two-ended graph* is a 6-tuple $G = (V, E, f, \Psi, s, t)$, where (V, E, f, Ψ) is a labelled graph, f is a function on $V \backslash \{s, t\}$ and $s, t \in V$ where $s \neq t$. Vertex s is called the *source vertex* of G and vertex t is called the *target vertex* of G. The

basic notions and notation for labelled graphs carry over to two-ended graphs. However, for the notion of isomorphism, care must be taken that the two ends are preserved. Thus, if G and G' are two-ended graphs, and α is a isomorphism from G to G', then $\alpha(s) = s'$ and $\alpha(t) = t'$, where s (s', resp.) is the source vertex of G (G', resp.) and t (t', resp.) is the target vertex of G (G', resp.).

2.4 The String Pointer Reduction System

In this chapter we consider the string pointer reduction system, which we will recall now (see also [11] and Chapter 9 in [12]).

We fix $\kappa \geq 2$, and define the alphabet $\Delta = \{2, 3, \ldots, \kappa\}$. For $D \subseteq \Delta$, we define $\bar{D} = \{\bar{a} \mid a \in D\}$ and $\Pi_D = D \cup \bar{D}$; also $\Pi = \Pi_\Delta$. We will use the alphabet Π to formally denote the pointers — the intuition is that the pointer p_i will be denoted by either i or $\bar{i}$. Accordingly, elements of Π will also be called *pointers*.

We use the 'bar operator' to move from Δ to $\bar{\Delta}$ and back from $\bar{\Delta}$ to Δ. Hence, for $p \in \Pi$, $\bar{\bar{p}} = p$. For a string $u = x_1 x_2 \cdots x_n$ with $x_i \in \Pi$, the *inverse* of u is the string $\bar{u} = \bar{x}_n \bar{x}_{n-1} \cdots \bar{x}_1$. For $p \in \Pi$, we define $\mathbf{p} = \begin{cases} p & \text{if } p \in \Delta \\ \bar{p} & \text{if } p \in \bar{\Delta} \end{cases}$, i.e., $\mathbf{p}$ is the 'unbarred' variant of p. The *domain* of a string $v \in \Pi^*$ is $dom(v) = \{\mathbf{p} \mid p \text{ occurs in } v\}$. A *legal string* is a string $u \in \Pi^*$ such that for each $p \in \Pi$ that occurs in u, u contains exactly two occurrences from $\{p, \bar{p}\}$.

We define the alphabet $\Theta_\kappa = \{M_i, \bar{M}_i \mid 1 \leq i \leq \kappa\}$ — these symbols denote the MDSs and their inversions. With each string over Θ_κ, we associate a unique string over Π through the homomorphism $\pi_\kappa : \Theta_\kappa^* \to \Pi^*$ defined by:

$$\pi_\kappa(M_1) = 2, \quad \pi_\kappa(M_\kappa) = \kappa, \quad \pi_\kappa(M_i) = i(i+1) \quad \text{for } 1 < i < \kappa,$$

and $\pi_\kappa(\bar{M}_j) = \overline{\pi_\kappa(M_j)}$ for $1 \leq j \leq \kappa$. A permutation of the string $M_1 M_2 \cdots M_\kappa$, with possibly some of its elements inverted, is called a *micronuclear pattern* since it can describe the MIC form of a gene. String u is *realistic* if there is a micronuclear pattern δ such that $u = \pi_\kappa(\delta)$.

Example 1
The MIC form of the gene that encodes the actin protein in the stichotrich *Sterkiella nova* is described by micronuclear pattern

$$\delta = M_3 M_4 M_6 M_5 M_7 M_9 \bar{M}_2 M_1 M_8$$

(see [22, 12]). The associated realistic string is $\pi_9(\delta) = 344567567893\bar{2}\bar{2}89$.

Note that every realistic string is legal, but a legal string need not be realistic. For example, a realistic string cannot have 'gaps' (missing pointers): thus 2244 is not realistic while it is legal. It is also easy to produce examples of legal strings which do not have gaps but still are not realistic — 3322 is such an example. For a pointer p and a legal string u, if both p and $\bar{p}$ occur in u then we say that both p

and $\bar{p}$ are *positive* in u; if on the other hand only p or only $\bar{p}$ occurs in u, then both p and $\bar{p}$ are *negative* in u. So, every pointer occurring in a legal string is either positive or negative in it. A nonempty legal string with no proper nonempty legal substrings is called *elementary*. For example, the legal string 234324 is elementary, while the legal string 234342 is not (because 3434 is a proper legal substring).

Definition 1
Let $u = x_1x_2\cdots x_n$ be a legal string with $x_i \in \Pi$ for $1 \leq i \leq n$. For a pointer $p \in \Pi$ such that $\{x_i, x_j\} \subseteq \{p, \bar{p}\}$ and $1 \leq i < j \leq n$, the *p-interval* of u is the substring $x_ix_{i+1}\cdots x_j$. Two distinct pointers $p, q \in \Pi$ *overlap* in u if the p-interval of u overlaps with the q-interval of u. ■

The string pointer reduction system consists of three types of reduction rules operating on legal strings. For all $p, q \in \Pi$ with $\mathbf{p} \neq \mathbf{q}$, we define:

- the *string negative rule* for p by $\mathbf{snr}_p(u_1ppu_2) = u_1u_2$,
- the *string positive rule* for p by $\mathbf{spr}_p(u_1pu_2\bar{p}u_3) = u_1\bar{u}_2u_3$,
- the *string double rule* for p, q by $\mathbf{sdr}_{p,q}(u_1pu_2qu_3pu_4qu_5) = u_1u_4u_3u_2u_5$,

where $u_1, u_2, \ldots, u_5$ are arbitrary strings over Π.

Note that each of these rules is defined only on legal strings that satisfy the given form. For example, $\mathbf{snr}_2$ is not defined on legal string 2323. It is important to realize that for every non-empty legal string there is at least one reduction rule applicable. Indeed, every legal string for which no string positive rule and no string double rule is applicable must have only nonoverlapping, negative pointers and thus a string negative rule is applicable.

We also define $Snr = \{\mathbf{snr}_p \mid p \in \Pi\}$, $Spr = \{\mathbf{spr}_p \mid p \in \Pi\}$ and $Sdr = \{\mathbf{sdr}_{p,q} \mid p, q \in \Pi, \mathbf{p} \neq \mathbf{q}\}$ to be the sets containing all the reduction rules of a specific type.

The string negative rule corresponds to the loop recombination operation, the string positive rule corresponds to the hairpin recombination operation, and the string double rule corresponds to the double-loop recombination operation. Note that the fact (pointed out at the end of Section 2.2) that the molecular operations remove pointers is explicit in the string pointer reduction system — indeed when a string rule for a pointer p (or pointers p and q) is applied, then all occurrences of p and $\bar{p}$ (or p, $\bar{p}$, q and $\bar{q}$) are removed.

Definition 2
The *domain* $dom(\rho)$ of a reduction rule ρ equals the set of unbarred variants of the pointers the rule is applied to, i.e., $dom(\mathbf{snr}_p) = dom(\mathbf{spr}_p) = \{\mathbf{p}\}$ and $dom(\mathbf{sdr}_{p,q}) = \{\mathbf{p}, \mathbf{q}\}$ for $p, q \in \Pi$. For a composition $\varphi = \varphi_1\ \varphi_2\ \cdots\ \varphi_n$ of reduction rules $\varphi_1, \varphi_2, \ldots, \varphi_n$, the *domain* $dom(\varphi)$ is the union of the domains of its constituents, i.e., $dom(\varphi) = dom(\varphi_1) \cup dom(\varphi_2) \cup \cdots \cup dom(\varphi_n)$. ■

Definition 3
Let u and v be legal strings and $S \subseteq \{Snr, Spr, Sdr\}$. Then a composition φ of reduction rules from S is called an *(S-)reduction of* u, if φ is applicable to (defined on) u. A *successful reduction* φ *of* u is a reduction of u such that $\varphi(u) = \lambda$. We then also say that φ is *successful for* u. We say that u is *reducible to* v *in* S if there is a S-reduction φ of u such that $\varphi(u) = v$. We simply say that u is *reducible to* v if u is reducible to v in $\{Snr, Spr, Sdr\}$. We say that u is *successful in* S if u is reducible to λ in S. ■

Note that if φ is a reduction of u, then $dom(\varphi) = dom(u) \backslash dom(\varphi(u))$. Because (as pointed out already) for every non-empty legal string there is at least one reduction rule applicable, we easily obtain Theorem 9.1 in [12] which states that every legal string is successful in $\{Snr, Spr, Sdr\}$.

Example 2
Let $S = \{Snr, Spr\}$, $u = 3245\bar{4}5\bar{3}\bar{2}$, and $v = \bar{5}4\bar{5}\bar{4}$. Then u is reducible to v in S, because $(\mathbf{snr}_3\ \mathbf{spr}_2)(u) = v$. Since applying $\varphi = \mathbf{spr}_{\bar{5}}\ \mathbf{spr}_4\ \mathbf{snr}_{\bar{2}}\ \mathbf{spr}_3$ to u yields λ, φ is successful for u. On the other hand, $u = 3232$ is not reducible to any v in S, because none of the rules in Snr and none of the rules in Spr is applicable for this u.

Referring to the Introduction, in Theorem 11 we present a characterization of the intermediate strings that may be constructed during the transformation of a given gene from its micronuclear form to its macronuclear form. Formally, this is a characterization of reducibility, which allows one to determine for any given legal strings u and v and $S \subseteq \{Snr, Spr, Sdr\}$, whether or not u is reducible to v in S. This result can be seen as a generalization of the results from Chapter 13 in [12], which provide a characterization of successfulness for realistic strings, that is, for the case where u is realistic and $v = \lambda$.

2.5 Pointer Removal Operation

Let φ be a reduction of a legal string u. If we let u' be the legal string obtained from u be deleting all pointers from $\Pi_{dom(\varphi(u))}$, then it turns out that φ is also a reduction of u'. In fact, φ is a successful reduction of u'. This is formalized in Theorem 6, and thus it states a necessary condition for reducibility. In the following sections we will strengthen Theorem 6 to obtain a characterization of reducibility.

Definition 4
For a subset $D \subseteq \Delta$, the D-removal operation, denoted by rem_D, is defined by $rem_D = erase_{D \cup \bar{D}}$. We also refer to rem_D operations, for all $D \subseteq \Delta$, as *pointer removal operations.* ■

Example 3
Let $u = 3245\bar{4}5\bar{3}\bar{2}$ and $D = \{4, 5\}$. Then $rem_D(u) = 32\bar{3}\bar{2}$. Note that $2, 3 \notin D$. Note also that $\varphi = \mathbf{snr}_3\ \mathbf{spr}_2$ is applicable to both u and $rem_D(u)$, but for $rem_D(u)$, φ is also successful.

The following easy to verify lemma formalizes the essence of the above example.

Lemma 5
Let u be a legal string and $D \subseteq dom(u)$. Let φ be a composition of reduction rules.

1. If φ is applicable to $rem_D(u)$ and φ does not contain string negative rules, then φ is applicable to u.

2. If φ is applicable to u and $dom(\varphi) \subseteq dom(u)\backslash D$, then φ is applicable to $rem_D(u)$.

3. If φ is applicable to both u and $rem_D(u)$, then $\varphi(rem_D(u)) = rem_D(\varphi(u))$.

Note that the first statement of Lemma 5 may not be true when φ is allowed to contain string negative rules. The obvious reason for this is that two identical occurrences of a pointer p may end up to be next to each other only if some pointers in between those occurrences are first removed by rem_D. This is illustrated in the following example.

Example 4
Let $u = 3245\bar{4}5\bar{3}66\bar{2}$, $v = \bar{5}45\bar{4}66$ and $D = dom(v)$. Then $rem_D(u) = 32\bar{3}\bar{2}$. Note that although $\varphi = \mathbf{snr}_3\ \mathbf{spr}_2$ is a successful reduction of $rem_D(u)$, φ is not applicable to u.

The following theorem is an immediate consequence of the previous lemma.

Theorem 6
Let $S \subseteq \{Snr, Spr, Sdr\}$. For legal strings u and v, if u is reducible to v in S and $D = dom(v)$, then $rem_D(u)$ is successful in S.

Proof
Let u be reducible to v in S. Then there is an S-reduction φ such that $\varphi(u) = v$. By Lemma 5, φ is an S-reduction of $rem_D(u)$ and $\varphi(rem_D(u)) = rem_D(\varphi(u)) = rem_D(v) = \lambda$. Hence, φ is a successful S-reduction of $rem_D(u)$. ∎

The proof of the above result observes that any reduction of u into v must be a successful reduction of $rem_D(u)$ where $D = dom(v)$. Referring to Example 4, we now note that u is not reducible to v, because $rem_D(u)$ has two successful reductions and neither is applicable to u. In fact, there is no v' with $D = dom(v')$ such that u is reducible to v'.

Figure 2.6: Part of a genome with three pointer pairs corresponding to the same gene.

2.6 Reduction Graphs

The main purpose of this section is to define the notion of reduction graph. A reduction graph represents some key aspects of reductions from a legal string u to a legal string v: it provides the additional requirements on u and v to make the reverse implication of Theorem 6 hold. In addition, it allows one to easily determine the number of string negative rules needed to successfully reduce u. We will first define the notion of a 2-edge coloured graph.

Definition 7
A *2-edge coloured graph* is a 7-tuple

$$G = (V, E_1, E_2, f, \Psi, s, t),$$

where both (V, E_1, f, Ψ, s, t) and (V, E_2, f, Ψ, s, t) are two-ended graphs. Note that E_1 and E_2 are not necessary disjoint. ■

The terminology and notation for the two-ended graph carries over to 2-edge coloured graphs. However, for the notion of isomorphism, care must be taken that the two sorts of edges are preserved. Thus, if $G = (V, E_1, E_2, f, \Psi, s, t)$ and $G' = (V', E_1', E_2', f', \Psi, s', t')$ are two-ended graphs, then it must hold that for any isomorphism α from G to G',

$$(x, u, y) \in E_i \text{ iff } (\alpha(x), u, \alpha(y)) \in E_i'$$

for all $x, y \in V$, $u \in \Psi$ and $i \in \{1, 2\}$.

We say that edges e_1 and e_2 have the *same colour* if either $e_1, e_2 \in E_1$ or $e_1, e_2 \in E_2$, otherwise they have *different colours*. An *alternating walk* in G is a walk $\pi = e_1 e_2 \cdots e_n$ in G such that e_i and e_{i+1} have different colours for $1 \leq i < n$. For each edge e with $\ell(e) \in \Pi^*$, we define $(\tau(e), \overline{\ell(e)}, \iota(e))$, denoted by $\bar{e}$, as the *reverse of* e.

We are ready now to define the notion of a reduction graph, the main technical notion of this chapter. The reduction graph is a 2-edge coloured graph and it is defined for a legal string u and a set of pointers $D \subseteq dom(u)$. The intuition behind it is as follows.

Figure 2.6 depicts a part of a genome with three pointer pairs corresponding to the same gene g. The reduction graph introduces two vertices for each pointer and two special vertices s and t representing the ends. It connects adjacent pointers through *reality edges* and connects pointers corresponding to the same pointer

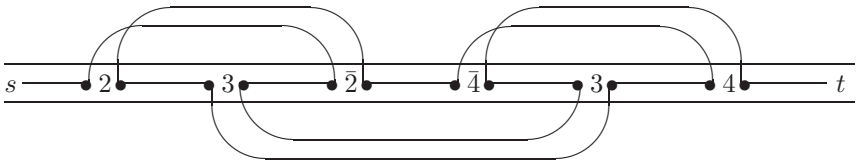


Figure 2.7: The reduction graph corresponding to the underlying genome.

pair through *desire edges* in a way that reflects how the parts will be glued after a molecular operation is applied on that pointer. The resulting reduction graph is depicted in Figure 2.7. Thus, every reality edge corresponds to a certain DNA segment. If such a DNA segment contains other pointers of g, then these pointers form the label of that reality edge.

By definition a realistic string has a physical interpretation. It shows the boundaries of the MDSs, and how these should be recombined (following their orientation). Considering a subset of these pointers, we still have the physical interpretation, although the other pointers are hidden in the segments. Technically, however, removing a subset of the pointers may change a realistic string into a legal one that is no longer realistic or even realizable (by renaming pointers we cannot obtain a realistic string). An example of such a case is given in the introduction of Section 2.10. In fact, each legal string has a physical interpretation with pointers indicating how parts of the string are to be reconnected, cf. Figure 2.7, where no use is made of any MDS-IES segmentation. Thus our definition of reduction graph works for legal strings in general, rather than only for realistic ones. The intuition of a reduction graph is similar to the intuition behind a reality and desire diagram (or breakpoint graph) from [16, 21].

Formally, the reduction graph of legal string u with respect to $D \subseteq dom(u)$ shows how u is reduced to a legal string v with $dom(v) = D$ by any possible reduction φ. The vertices of the graph correspond to (two copies of each of) the pointers that are removed during the reduction (those in $\Pi_{dom(u)\setminus D}$). As illustrated above, we have two types of edges. The desire edges are unlabelled and connect the pointer pairs in $\Pi_{dom(u)\setminus D}$, while reality edges connect the successive pointers in $\Pi_{dom(u)\setminus D}$ and are labelled by the strings over Π_D^* that are in between these pointers in u.

Definition 8
Let $D \subseteq \Delta$ and let u be a legal string, such that $u = \delta_0 p_1 \delta_1 p_2 \dots p_n \delta_n$ where $\delta_0, \dots, \delta_n \in \Pi_D^*$ and $p_1, \dots, p_n \in \Pi_{dom(u)\setminus D}$. The *reduction graph of u with respect to D*, denoted by $\mathcal{R}_{u,D}$, is a 2-edge coloured graph $(V, E_1, E_2, f, \Pi, s, t)$, where

$$V = \{I_1, I_2, \dots, I_n\} \cup \{I'_1, I'_2, \dots, I'_n\} \cup \{s, t\},$$

$$E_1 = E_{1,r} \cup E_{1,l}, \text{where}$$

$$E_{1,r} = \{e_0, e_1, \dots, e_n\} \text{ with } e_i = (I'_i, \delta_i, I_{i+1}) \text{ for } 1 \leq i \leq n-1,$$

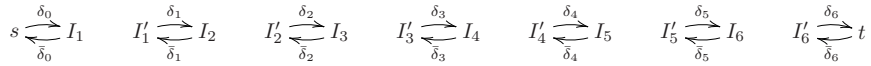


Figure 2.8: The part of the reduction graph of the legal string u with respect to D as defined in Example 5 which involves only reality edges (the vertex labels are omitted).

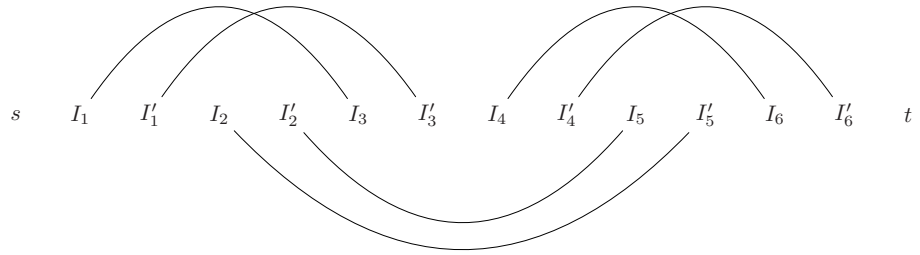


Figure 2.9: The part of the reduction graph of the legal string u with respect to D as defined in Example 5, where only desire edges are shown (the vertex labels are omitted). Crossing edges correspond to positive pointers.

$$e_0 = (s, I_1), e_n = (I'_n, t),$$
$$E_{1,l} = \{\bar{e} \mid e \in E_{1,r}\},$$

$$E_2 = \{(I'_i, \lambda, I_j), (I_i, \lambda, I'_j) \mid i, j \in \{1, 2, \ldots, n\} \text{ with } i \neq j \text{ and } p_i = p_j\} \cup$$
$$\{(I_i, \lambda, I_j), (I'_i, \lambda, I'_j) \mid i, j \in \{1, 2, \ldots, n\} \text{ and } p_i = \bar{p}_j\}, \text{and}$$
$$f(I_i) = f(I'_i) = \mathbf{p}_i \text{ for } 1 \leq i \leq n.$$

■

The edges of E_1 are called the *reality edges*, and the edges of E_2 are called the *desire edges*. Note that E_1 and E_2 are not necessary disjoint. The components of $\mathcal{R}_{u,D}$ that do not contain s and t are called *cyclic components*. When $D = \varnothing$, we simply refer to $\mathcal{R}_{u,D}$ as the *reduction graph of* u.

Thus the reduction graph is a 'superposition' of two graphs on the same set of vertices V: one graph with edges from E_1 (reality edges), and one graph with edges from E_2 (desire edges). The following example should make the notion of reduction graph more clear.

Example 5
Let $u = 526883\bar{2}5\bar{4}37746$ be a legal string and $D = \{5, 6, 7, 8\} \subseteq dom(u)$. Thus, $\{2, 3, 4\} = dom(u) \backslash D$, and

$$u = \delta_0\ 2\ \delta_1\ 3\ \delta_2\ \bar{2}\ \delta_3\ \bar{4}\ \delta_4\ 3\ \delta_5\ 4\ \delta_6$$

with $\delta_0 = 5$, $\delta_1 = 688$, $\delta_2 = \lambda$, $\delta_3 = 5$, $\delta_4 = \lambda$, $\delta_5 = 77$ and $\delta_6 = 6$. Notice that $\delta_1, \delta_2, \ldots, \delta_6 \in \Pi^*_D$. This example corresponds to the situation in Figure 2.6.

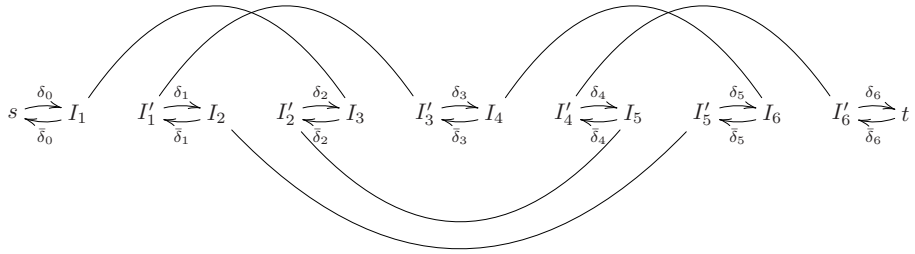


Figure 2.10: The reduction graph $\mathcal{R}_{u,D}$ as defined in Example 5 (the vertex labels are omitted).

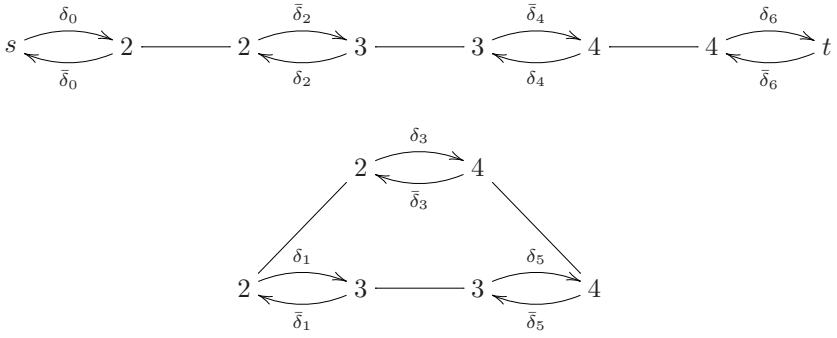


Figure 2.11: The reduction graph of Figure 2.10 where every vertex (except s and t) is represented by its label.

The reduction graph $\mathcal{R}_{u,D}$ of u with respect to D is given in Figure 2.10. It is the union of the graphs in Figure 2.8 and Figure 2.9. Note that for every desire edge e, we represent both e and $\bar{e}$ by a single unlabelled, undirected edge. The graphs are drawn in a form that closely relates to the linear ordering of u. The desire edges that cross correspond to positive pointers, and the desire edges that do not cross correspond to negative pointers.

Since the exact identity of the vertices in a reduction graph is not essential for the problems considered in this chapter (we need only to know, modulo 'bar', which pointer is represented by a given vertex), in order to simplify the pictorial notation of reduction graphs we will replace the vertices (except for s and t) by their labels. Figure 2.11 gives $\mathcal{R}_{u,D}$ in this way. In this figure we have reordered the vertices, making it transparent that $\mathcal{R}_{u,D}$ has a single cyclic component (the figure illustrates why the adjective 'cyclic' was added).

Note that a reduction graph is an undirected graph in the sense that if $e \in E_1$ ($e \in E_2$, resp.) then also $\bar{e} \in E_1$ ($\bar{e} \in E_2$, resp.). If we think of a reduction graph as an undirected graph by considering edges e and $\bar{e}$ as one undirected edge, then both s and t are connected to exactly one (undirected) edge, and every other vertex is connected to exactly two (undirected) edges. As as corollary to Euler's theorem, a reduction graph has exactly one component that has a linear structure with s and t as endpoints and possibly one or more components that have a cyclic structure (the cyclic components). Thus, there is a unique alternating walk from s to t in every reduction graph.

If a 2-edge coloured graph G has a unique alternating walk from s to t, then the label of this walk is called the *reduct of* G, denoted by $red(G)$. We know now that if $\mathcal{R}_{u,D}$ is a reduction graph of a legal string u with respect to $D \subseteq dom(u)$, then the reduct exists. It is then also called the *reduct of* u *to* D, and denoted by $red(u, D)$. Since $\mathcal{R}_{u,dom(u)}$ consists of the vertices s and t connected by a (reality) edge labelled by u (and by $\bar{u}$ in the reverse direction), we have $red(u, dom(u)) = u$. Also, it is clear that if 2-edge coloured graphs G_1 and G_2 are isomorphic, then $red(G_1) = red(G_2)$.

Example 6
If we take u and D from Example 5, then

$$red(u, D) = \delta_0\bar{\delta}_2\bar{\delta}_4\delta_6 = 56,$$

which is easy to see in Figure 2.11.

2.7 Reduction Function

Before we can prove (in the next section) our main theorem on reducibility, we need to define reduction functions. A reduction function operates on reduction graphs. As we will see, these functions simulate the effect (up to isomorphism) of each of the three string pointer reduction rules on a reduction graph. For a vertex

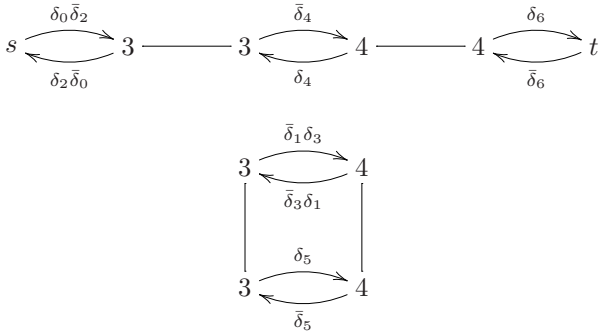


Figure 2.12: The reduction graph obtained when applying rf_2 to the reduction graph of Figure 2.11.

label p, the p-reduction function merges edges that form a walk 'over' vertices labelled by p and removes all vertices labelled by p.

Definition 9
For each vertex label p, we define the *p-reduction function* rf_p, which constructs for every 2-edge coloured graph $G = (V, E_1, E_2, f, \Psi, s, t)$, the 2-edge coloured graph

$$rf_p(G) = (V', (E_1 \backslash E_{rem}) \cup E_{add}, E_2 \backslash E_{rem}, f|V', \Psi, s, t),$$

with

$$\begin{aligned} V' &= \{s,t\} \cup \{v \in V \backslash \{s,t\} \mid f(v) \neq p\}, \\ E_{rem} &= \{e \in E_1 \cup E_2 \mid f(\iota(e)) = p \text{ or } f(\tau(e)) = p\}, \text{and} \\ E_{add} &= \{(\iota(\pi), \ell(\pi), \tau(\pi)) \mid \pi = e_1 e_2 \cdots e_n \text{ with } n > 2 \text{ is an alternating walk} \\ &\quad \text{in } G \text{ with } f(\iota(\pi)) \neq p,\ f(\tau(\pi)) \neq p, \text{ and } f(\tau(e_i)) = p \text{ for } 1 \leq i < n\}. \end{aligned}$$

■

Example 7
If we take the reduction graph $\mathcal{R}_{u,D}$ from Example 5, cf. Figure 2.11, then $rf_2(\mathcal{R}_{u,D})$ is given in Figure 2.12.

It is easy to see that the following property holds for each reduction graph $\mathcal{R}_{u,D}$ and all $p \in dom(u) \backslash D$:

$$red(\mathcal{R}_{u,D}) = red(rf_p(\mathcal{R}_{u,D})).$$

Also, reduction functions commute under composition. Thus, if moreover there is a $q \in dom(u) \backslash D$ such that $p \neq q$, then

$$(rf_q\ rf_p)(\mathcal{R}_{u,D}) = (rf_p\ rf_q)(\mathcal{R}_{u,D}).$$

The main property of reduction functions is that they simulate the effect (up to isomorphism) of each of the three string pointer reduction rules on a reduction graph.

Theorem 10
Let u be a legal string, let $D \subseteq dom(u)$, and let φ be a reduction of u such that $dom(\varphi) = \{p_1, p_2, \ldots, p_n\} \subseteq dom(u) \backslash D$. Then

$$(rf_{p_n} \ \cdots \ rf_{p_2} \ rf_{p_1})(\mathcal{R}_{u,D}) \approx \mathcal{R}_{\varphi(u),D},$$

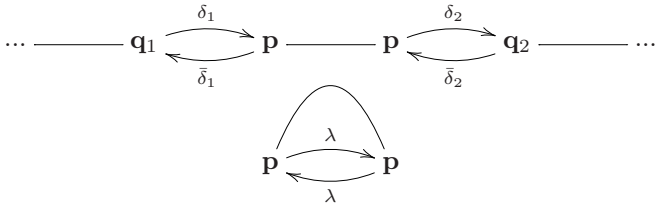
and $red(u, D) = red(\varphi(u), D)$.

Proof
To prove the first statement, it suffices to prove the cases where $\varphi = \mathbf{snr}_p$, $\varphi = \mathbf{spr}_p$ and $\varphi = \mathbf{sdr}_{p,q}$ for $p, q \in \Pi_{dom(u)\backslash D}$.

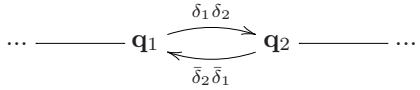
We first prove the **snr** case. Assume $\mathbf{snr}_p$ is applicable to u. We consider the general case

$$u = u_1 q_1 \delta_1 p p \delta_2 q_2 u_2$$

for some $\delta_1, \delta_2 \in \Pi_D^*$, $q_1, q_2 \in \Pi_{dom(u)\backslash D}$ and $u_1, u_2 \in \Pi^*$. In the special case where q_1 (q_2, resp.) does not exist, the vertex labelled by $\mathbf{q}_1$ ($\mathbf{q}_2$, resp.) in the graphs below equals the source vertex s (target vertex t, resp.). We will first prove that $rf_{\mathbf{p}}(\mathcal{R}_{u,D}) = \mathcal{R}_{\mathbf{snr}_p(u),D}$. Because $u = u_1 q_1 \delta_1 p p \delta_2 q_2 u_2$, the reduction graph $\mathcal{R}_{u,D}$ is



where we omitted the parts of the graph that remain the same after applying $rf_{\mathbf{p}}$. Now, the graph $rf_{\mathbf{p}}(\mathcal{R}_{u,D})$ is given below.



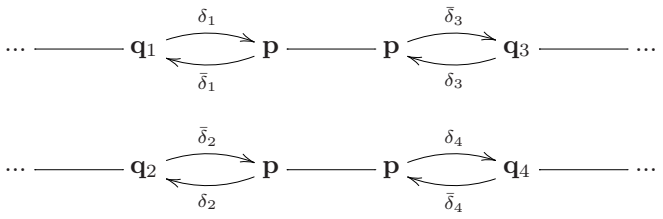
This is clearly the reduction graph of $\mathbf{snr}_p(u) = u_1 q_1 \delta_1 \delta_2 q_2 u_2$ with respect to D. Thus, indeed $rf_{\mathbf{p}}(\mathcal{R}_{u,D}) \approx \mathcal{R}_{\mathbf{snr}_p(u),D}$.

We now prove the **spr** case. Assume $\mathbf{spr}_p$ is applicable to u. We may distinguish three cases, which differ in the number of elements of $\Pi_{dom(u)\backslash D}$ in between p and $\bar{p}$ in u:

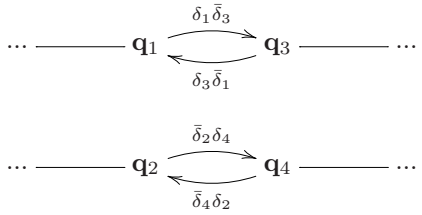
1. $u = u_1 q_1 \delta_1 p \delta_2 \bar{p} \delta_4 q_4 u_3$
2. $u = u_1 q_1 \delta_1 p \delta_2 q_2 \delta_3 \bar{p} \delta_4 q_4 u_3$

3. $u = u_1 q_1 \delta_1 p \delta_2 q_2 u_2 q_3 \delta_3 \bar{p} \delta_4 q_4 u_3$

for some $\delta_1, \ldots, \delta_4 \in \Pi_D^*$, $q_1, \ldots, q_4 \in \Pi_{dom(u)\setminus D}$, and $u_1, u_2, u_3 \in \Pi^*$. Note that we have assumed that p is preceded and that $\bar{p}$ is followed by an element from $\Pi_{dom(u)\setminus D}$. The special cases where q_1 or q_4 do not exist, can be handled in the same way as we did for the **snr** case (by setting them equal to s and t, resp.). In each of the three cases, one can prove that $rf_{\mathbf{p}}(\mathcal{R}_{u,D}) \approx \mathcal{R}_{\mathbf{spr}_p(u),D}$. We will discuss it in detail only for the third case. The reduction graph $\mathcal{R}_{u,D}$ is



where we again omitted the parts of the graph that remain the same after applying $rf_{\mathbf{p}}$. Now, the graph $rf_{\mathbf{p}}(\mathcal{R}_{u,D})$ is given below.



This graph is clearly isomorphic to the reduction graph of

$$\mathbf{spr}_p(u) = u_1 q_1 \delta_1 \bar{\delta}_3 \bar{q}_3 \bar{u}_2 \bar{q}_2 \bar{\delta}_2 \delta_4 q_4 u_3$$

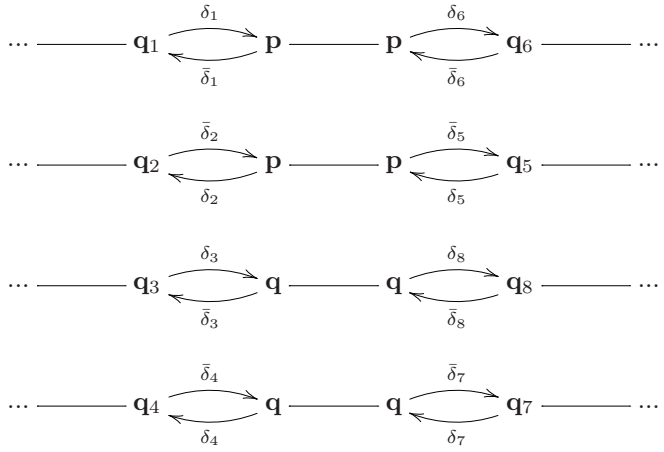
with respect to D. Thus, indeed $rf_{\mathbf{p}}(\mathcal{R}_{u,D}) \approx \mathcal{R}_{\mathbf{spr}_p(u),D}$.

Finally, we prove the **sdr** case. Assume $\mathbf{sdr}_{p,q}$ is applicable to u. We only consider the general case (the other cases are proved similarly):

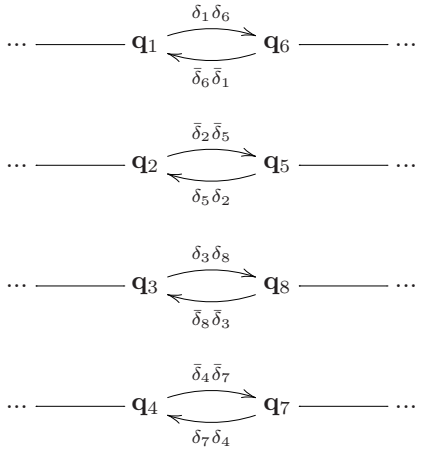
$$u = u_1\ q_1 \delta_1 p \delta_2 q_2\ u_2\ q_3 \delta_3 q \delta_4 q_4\ u_3\ q_5 \delta_5 p \delta_6 q_6\ u_4\ q_7 \delta_7 q \delta_8 q_8\ u_5$$

for some $\delta_1, \ldots, \delta_8 \in \Pi_D^*$, $q_1, \ldots, q_8 \in \Pi_{dom(u)\setminus D}$, and $u_1, \ldots, u_5 \in \Pi^*$. The

reduction graph $\mathcal{R}_{u,D}$ is



where we omitted the parts of the graph that remain the same after applying $(rf_{\mathbf{q}}\ rf_{\mathbf{p}})$. Now, the graph $rf_{\mathbf{q}}(rf_{\mathbf{p}}(\mathcal{R}_{u,D}))$ is given below.



This graph is clearly isomorphic to the reduction graph of

$$\mathbf{sdr}_{p,q}(u) = u_1 q_1 \delta_1 \delta_6 q_6 u_4 q_7 \delta_7 \delta_4 q_4 u_3 q_5 \delta_5 \delta_2 q_2 u_2 q_3 \delta_3 \delta_8 q_8 u_5$$

with respect to D. Thus, indeed $rf_{\mathbf{q}}(rf_{\mathbf{p}}(\mathcal{R}_{u,D})) \approx \mathcal{R}_{\mathbf{sdr}_{p,q}(u),D}$. This proves the first statement.

Now, by the fact that the reduction function does not change the reduct of the graph, and by the first statement, we have

$$red(\mathcal{R}_{u,D}) = red((rf_{p_1}\ rf_{p_2}\ \cdots\ rf_{p_n})(\mathcal{R}_{u,D})) = red(\mathcal{R}_{\varphi(u),D}).$$

Thus, $red(u, D) = red(\varphi(u), D)$ and this proves the second statement. ∎

2.8 Characterization of Reducibility

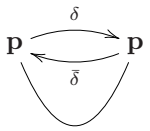
We are now ready to prove our main theorem on reducibility. In Theorem 6 we have shown that if u is reducible to v in S, then $rem_{dom(v)}(u)$ is successful in S. Here we strengthen this theorem into an iff statement by additionally requiring that v equals the reduct of u to $dom(v)$. The resulting characterization is independent of the chosen set of reduction rules $S \subseteq \{Snr, Spr, Sdr\}$.

Theorem 11
Let u and v be legal strings, $D = dom(v) \subseteq dom(u)$ and $S \subseteq \{Snr, Spr, Sdr\}$. Then u is reducible to v in S iff $rem_D(u)$ is successful in S and $red(u, D) = v$.

Proof
Let u be reducible to v in S. Therefore, there is an S-reduction φ of u such that $\varphi(u) = v$. Also, $rem_D(u)$ is successful in S by Theorem 6. By Theorem 10, we have $red(u, D) = red(\varphi(u), D)$. Now, $red(\varphi(u), D) = \varphi(u) = v$, because $D = dom(\varphi(u))$.

To prove the reverse implication, let $rem_D(u)$ be successful in S and $red(u, D) = v$. We have to prove that u is reducible to v in S. Clearly, there is a successful S-reduction φ of $rem_D(u)$.

Assume that φ is not applicable to u. Since φ is applicable to $rem_D(u)$, we know from Lemma 5 that $\varphi = \varphi_2\ \mathbf{snr}_p\ \varphi_1$ for some φ_1, φ_2 and p, where φ_1 is applicable to u and $\mathbf{snr}_p$ is not applicable to $\varphi_1(u)$. Thus, $p\delta p$ is a substring of $\varphi_1(u)$ with $\delta \in \Pi_D^* \backslash \{\lambda\}$. Therefore the following graph



must be isomorphic to a cyclic component of the reduction graph $\mathcal{R}_{\varphi_1(u),D}$ of $\varphi_1(u)$ with respect to D. Because $v = red(u, D) = red(\varphi_1(u), D)$ is a legal string and $dom(v) = D$, the labels of the reality edges of $\mathcal{R}_{\varphi_1(u),D}$ belonging to cyclic components are empty. This is a contradiction and therefore φ is applicable to u. Now, we have $\varphi(u) = red(\varphi(u), D) = red(u, D) = v$, because $D = dom(\varphi(u))$. Thus, u is reducible to v in S. ∎

Note that the proof of Theorem 11 even proves a stronger fact. The S-reduction φ of u with $\varphi(u) = v$ can be taken to be same as the (successful) S-reduction φ of $rem_D(u)$. The following corollary follows directly from the previous theorem and the fact that every legal string is successful in $\{Snr, Spr, Sdr\}$.

Corollary 12
Let u and v be legal strings and $D = dom(v) \subseteq dom(u)$. Then u is reducible to v iff $red(u, D) = v$.

The previous corollary shows that reducibility can be checked quite efficiently. Since the reduction graph of a legal string u has $2|u| + 2$ vertices and $8|u| + 4$ edges (counting an undirected desire edge as two (directed) edges), it takes only linear time $O(|u|)$ to generate $\mathcal{R}_{u,\varnothing}$ using the adjacency lists representation. Also, generating $\mathcal{R}_{u,D}$ for any $D \subseteq dom(u)$ is of at most the same complexity as $\mathcal{R}_{u,\varnothing}$. Now, since the walk from s to t does not contain vertices more than once, it takes only linear time to determine $red(u, D) = v$, and therefore, by the previous corollary, it takes linear time to determine whether or not u is reducible to v.

The next corollary illustrates that the function of the reduct is twofold: it does not only determine, given u and $D \subseteq dom(u)$, *which* legal string is obtained by applying a reduction φ of u with $dom(\varphi(u)) = D$, but also *whether or not* there is such a φ.

Corollary 13
Let u be a legal string and $D \subseteq dom(u)$. Then u there is a reduction φ of u with $dom(\varphi(u)) = D$ iff $red(u, D)$ is legal and $dom(red(u, D)) = D$.

Proof
We first prove the forward implication. If we let $v = \varphi(u)$, then v is a legal string, u is reducible to v, and $D = dom(v)$. By Corollary 12, $red(u, D) = v$ and therefore $red(u, D)$ is legal and $dom(red(u, D)) = D$.

We now prove the reverse implication. If we let $v = red(u, D)$, then v is legal and $dom(v) = D$. By Corollary 12, u is reducible to v. ∎

Example 8
Let u and D be as in Example 5. By Example 6, $red(u, D) = 56$. Therefore by Corollary 13, there is no reduction φ of u with $dom(\varphi(u)) = D$. Thus, there is no reduction φ of u with $dom(\varphi) = \{2, 3, 4\}$.

2.9 Cyclic Components

In this section we consider the cyclic components of the 'full' reduction graph $\mathcal{R}_{u,\varnothing}$ of a legal string u. We show that if $\mathbf{snr}_p$ is applicable to u for some pointer p, then the number of cyclic components of $\mathcal{R}_{\mathbf{snr}_p(u),\varnothing}$ is exactly one less than the number of cyclic components of $\mathcal{R}_{u,\varnothing}$. On the other hand, if either $\mathbf{spr}_p$ or $\mathbf{sdr}_{p,q}$ is applicable to u for some pointer p, q, then the number of cyclic components remains the same. Before we state this result (Theorem 17), we will prepare for its proof by studying some elementary connections between u and the structures in $\mathcal{R}_{u,\varnothing}$. Since all the edges of $\mathcal{R}_{u,\varnothing}$ are labelled λ, we will omit the labels of the edges in the figures.

Because desire edges in a reduction graph connect vertices that are of the same label, for every label $\mathbf{p}$, there are exactly 0, 2 or 4 vertices labelled by $\mathbf{p}$ in every cyclic component of a reduction graph. The following lemma establishes an additional property of the number of vertices of a single label in a cyclic component.

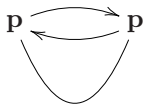
Lemma 14
Let u be a legal string, and let P be a cyclic component in $\mathcal{R}_{u,\varnothing}$. Let p (q, resp.) be the first (last, resp.) pointer (from left to right) in u such that there is a vertex in P with label **p** (**q**, resp.). Then there are exactly two vertices of P labelled by **p** and there are exactly two vertices of P labelled by **q**.

Proof
Assume that all four vertices labelled by **p** are in P. Then these vertices are I_i, I'_i, I_j and I'_j for some i and j with $i < j$. By the definition of reduction graph, there is a reality edge from vertex I_i to vertex I'_{i-1}. But by the definition of p, vertex I'_{i-1} cannot belong to P, which is a contradiction. Therefore, there are only two vertices labelled by **p** in P. The second claim is proved analogously. ∎

Note that in the previous lemma, **p** and **q** need not be distinct. Note also that if all the vertices of a cyclic component have the same label, than the cyclic component has exactly two vertices.

Lemma 15
Let u be a legal string, and let $p \in \Pi$. Then $\mathcal{R}_{u,\varnothing}$ has a cyclic component consisting of exactly two vertices, which are both labelled by **p** iff either pp or $\bar{p}\bar{p}$ is a substring of u.

Proof
Let either pp or $\bar{p}\bar{p}$ be a substring of u. Then

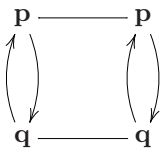


is a cyclic component of $\mathcal{R}_{u,\varnothing}$ consisting of exactly two vertices, both labelled by **p**.

To prove the forward implication, let $\mathcal{R}_{u,\varnothing}$ have a cyclic component P consisting of exactly two vertices, both labelled by **p**. Clearly, every vertex of a cyclic component has exactly one incoming and one outgoing edge in each colour. Because there is a reality edge between the two vertices of P, I'_i and I_{i+1} are the vertices of P for some i. Now, since there is a desire edge (I'_i, I_{i+1}) in P, either p or $\bar{p}$ occurs twice in u. As reality edges in $\mathcal{R}_{u,\varnothing}$ connect adjacent pointers in u, either pp or $\bar{p}\bar{p}$ is a substring of u. ∎

Lemma 16
Let u be a legal string, let p and q be negative pointers occurring in u. Then $\mathcal{R}_{u,\varnothing}$ has a cyclic component consisting of exactly two vertices labelled by **p** and two vertices labelled by **q** iff either $u = u_1pqu_2qpu_3$ or $u = u_1qpu_2pqu_3$ for some strings $u_1, u_2, u_3 \in \Pi^*$.

Proof
Let either $u = u_1pqu_2qpu_3$ or $u = u_1qpu_2pqu_3$ for some strings $u_1, u_2, u_3 \in \Pi^*$. Then



is a cyclic component of $\mathcal{R}_{u,\varnothing}$ consisting of exactly two vertices labelled by **p** and two vertices labelled by **q**.

To prove the forward implication, let $\mathcal{R}_{u,\varnothing}$ have a cyclic component P consisting of exactly two vertices labelled by **p** and two vertices labelled by **q**. Since each cyclic component 'is' a cycle of edges of alternating colour, and since desire edges connect only vertices with the same label, the component looks like the figure above. Since reality edges in $\mathcal{R}_{u,\varnothing}$ connect adjacent pointers in u and since p and q are negative, either $u = u_1pqu_2qpu_3$ or $u = u_1pqu_2pqu_3$ with $u_i \in \Pi^*$ (with possibly p and q interchanged). Assume that $u = u_1pqu_2pqu_3$ (with possibly p and q interchanged). Then there must be vertices I'_i and I'_j labelled by **p** with a desire edge (I'_i, I'_j) in P. But this is impossible since p is negative. Consequently, $u = u_1pqu_2qpu_3$ (with possibly p and q interchanged). ∎

The following theorem states that only the string negative rules can remove cyclic components. This is consistent with the fact that only loop recombination introduces a new (cyclic) molecule, cf. Figure 2.3. Clearly, by the definition of reduction function, a cyclic component is removed by simply removing its vertices and edges and not by merging with another component.

Theorem 17
Let u be a legal string, let N be the number of cyclic components of $\mathcal{R}_{u,\varnothing}$, and let $p \in \Pi$ with $\mathbf{p} \in dom(u)$.

- If $\mathbf{snr}_p$ is applicable to u, then the reduction graph of $\mathbf{snr}_p(u)$ has exactly $N - 1$ cyclic components.
- If $\mathbf{spr}_p$ is applicable to u, then the reduction graph of $\mathbf{spr}_p(u)$ has exactly N cyclic components.

Now let $q \in \Pi$ with $\mathbf{q} \in dom(u)$ and $\mathbf{p} \neq \mathbf{q}$.

- If $\mathbf{sdr}_{p,q}$ is applicable to u, then the reduction graph of $\mathbf{sdr}_{p,q}(u)$ has exactly N cyclic components.

Proof
First note that by the definition of reduction function and Theorem 10 the number of cyclic components cannot increase when applying reduction rules.

Let $\mathbf{snr}_p$ be applicable to u. By Lemma 15, $\mathcal{R}_{u,\varnothing}$ has a cyclic component consisting of exactly two vertices, which are both labelled by **p**. It follows then from

Theorem 10 that the reduction graph of $\mathbf{snr}_p(u)$ has at most $N-1$ cyclic components. The other two vertices labelled by $\mathbf{p}$ are connected by reality edges to vertices that are not labelled by $\mathbf{p}$, and therefore this component does not disappear. Hence, the reduction graph of $\mathbf{snr}_p(u)$ has exactly $N-1$ cyclic components.

Let $\mathbf{spr}_p$ be applicable to u. Assume that the reduction graph of $\mathbf{spr}_p(u)$ has less than N cyclic components. Then by Theorem 10, there exist a cyclic component P of $\mathcal{R}_{u,\varnothing}$ consisting of only vertices labelled by $\mathbf{p}$. By Lemma 14, P consists of only two vertices. By Lemma 15, either pp or $\bar{p}\bar{p}$ is a substring of u and thus $\mathbf{spr}_p$ is not applicable to u. This is a contradiction. Consequently, the reduction graph of $\mathbf{spr}_p(u)$ has exactly N cyclic components.

Let $\mathbf{sdr}_{p,q}$ be applicable to u. Assume that the reduction graph of $\mathbf{sdr}_{p,q}(u)$ has less than N cyclic components. Then there exist a cyclic component P in $\mathcal{R}_{u,\varnothing}$ consisting only of vertices labelled by $\mathbf{p}$ and $\mathbf{q}$. Assume that all vertices of P are labelled by $\mathbf{p}$. Then, analogously to the previous case, we deduce that either pp or $\bar{p}\bar{p}$ is a substring of u. Thus $\mathbf{sdr}_{p,q}$ is not applicable to u. This is a contradiction. Similarly, P cannot consist only of vertices labelled by $\mathbf{q}$. Assume then that P consists of vertices that are labelled by both $\mathbf{p}$ and $\mathbf{q}$. By Lemma 14 and the fact that pointers p and q overlap, there are only two vertices labelled by $\mathbf{p}$ in P and two vertices labelled by $\mathbf{q}$ in P. By Lemma 16, either $u = u_1pqu_2qpu_3$ or $u = u_1qpu_2pqu_3$ for some strings $u_1, u_2, u_3 \in \Pi^*$. Thus $\mathbf{sdr}_{p,q}$ is not applicable to u. This is a contradiction. Therefore, such a component P cannot exist and so the reduction graph of $\mathbf{sdr}_{p,q}(u)$ has exactly N cyclic components. ∎

The previous theorem can be reformulated as follows, yielding a key property of reduction graphs.

Theorem 18
Let N be the number of cyclic components of the reduction graph of legal string u. Then every successful reduction of u has exactly N string negative rules.

The Invariant Theorem [14] (and Chapter 12 in [12]) shows that all successful reductions of a *realistic* string u have the same number of string negative rules. Therefore, Theorem 18 can be seen as a generalization of this result, since it holds for *legal* strings in general. Indeed, the technical framework used in [14] is the MDS descriptor reduction system which is only suited to model realistic strings.

Moreover, Theorem 18 shows that this number N is an elegant graph theoretical property of the reduction graph. As a consequence, it can be efficiently obtained. Since it takes $O(|u|)$ to generate $\mathcal{R}_{u,\varnothing}$, and again $O(|u|)$ to determine the number of connected components of $\mathcal{R}_{u,\varnothing}$, the previous theorem implies that it takes only linear time to determine how many string negative rules are needed to successfully reduce legal string u. Theorem 18 will be used in the next section, when we characterize successfulness in $S \subseteq \{Spr, Sdr\}$.

Example 9
Let $u = 23\bar{2}\bar{4}34$ be a legal string. The reduction graph of u is depicted in Figure 2.11, where $\delta_i = \lambda$ for all $i \in \{0, 1, \ldots, 6\}$. By Theorem 18 every reduction of u

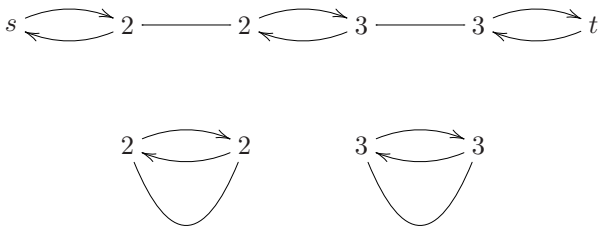


Figure 2.13: The reduction graph of $u = 2233$.

has exactly one string negative rule. There are exactly four successful reductions of u, these are $\mathbf{snr}_2\ \mathbf{spr}_3\ \mathbf{spr}_{\bar{4}}$, $\mathbf{snr}_{\bar{3}}\ \mathbf{spr}_2\ \mathbf{spr}_{\bar{4}}$, $\mathbf{snr}_{\bar{3}}\ \mathbf{spr}_{\bar{4}}\ \mathbf{spr}_2$ and $\mathbf{snr}_4\ \mathbf{spr}_{\bar{3}}\ \mathbf{spr}_2$. Notice that each of these reductions has exactly one string negative rule.

Remark
Results in [13] (and Chapter 13 in [12]) show that a successful reduction of a realistic string u has at least one string negative rule if the string has a disjoint cycle. Clearly, the notions of disjoint cycle and (cyclic) component are related. It is easy to verify that every disjoint cycle of a string can be found as a connected component of the reduction graph of the string, although that might be the linear component. As an example, consider the realistic string $u = \pi_3(M_1M_2M_3) = 2233$. This realistic string has three disjoint cycles $\{22\}$, $\{33\}$, and $\{23, 32\}$ corresponding to the connected components of the reduction graph of u, see Figure 2.13. This correspondence is not a bijection for all legal strings, not even for realistic ones. E.g., realistic string $u = \pi_3(M_3\bar{M}_1M_2) = 3\bar{2}23$ has only a single disjoint cycle $\{33\}$ whereas its reduction graph has two components, one linear and one cyclic. Hence, the number of disjoint cycles cannot be used to characterize the number of string negative rules present in every successful reduction of u.

It is easy to see that for legal string u and $D \subseteq dom(u)$, $\mathcal{R}_{rem_D(u),\varnothing}$ is isomorphic to $\mathcal{R}_{u,D}$ modulo the labels of the edges. Now, we have the following corollary to Theorems 18.

Corollary 19
Let u be a legal string, $D \subseteq dom(u)$, and let N be the number of cyclic components of $\mathcal{R}_{u,D}$. Then every reduction φ of u with $dom(\varphi(u)) = D$ has exactly N string negative rules.

Proof
Let φ be a reduction of u with $dom(\varphi(u)) = D$. Then by Theorem 6, φ is a successful reduction of $rem_D(u)$. Since $\mathcal{R}_{u,D}$ is isomorphic to $\mathcal{R}_{rem_D(u),\varnothing}$ modulo the labels of the edges, $\mathcal{R}_{rem_D(u),\varnothing}$ has N cyclic components. By Theorem 18, φ has exactly N string negative rules. ∎

2.10 Successfulness of Legal Strings

In [13] (and Chapter 13 in [12]) an elementary characterization of the *realistic* strings that are successful in any given $S \subseteq \{Snr, Spr, Sdr\}$ is presented. This is helpful in applying Theorem 11, where reducibility of legal string u into legal string v is translated into successfulness of $rem_D(u)$ with $D = dom(v)$. Unfortunately, even when u is a realistic string, $rem_D(u)$ for some $D \subseteq dom(u)$ is not necessary a realistic string. For example, $u = \pi_5(M_1M_2\bar{M}_3M_4M_5) = 223\bar{4}\bar{3}455$ is realistic, while $rem_{\{4\}}(u) = 223\bar{3}55$ is not. As a matter of fact, it can be shown that this legal string is not even *realizable*, that is, the legal string can not be transformed into a realistic string by renaming pointers. Formally, legal string v is *realizable* if there exists a homomorphism $h : \Pi \to \Pi$ with $h(\bar{p}) = \overline{h(p)}$ for all $p \in \Pi$ such that $h(v)$ is realistic. Thus, e.g., $223\bar{3}44$ and $\bar{2}\bar{2}\bar{3}344$ are also not realistic.

In this section we generalize the results from [13], and give a characterization of the *legal* strings that are successful in any given $S \subseteq \{Snr, Spr, Sdr\}$. Theorems 22, 23, and 25 are the 'legal counterparts' of Theorems 8, 9, and 6 in [13], respectively. These results are independent of the results in the previous sections of this chapter. On the other hand, Theorems 27, 28, and 30 (the 'legal counterparts' of Theorems 14, 11, and 13 in [13], respectively) rely heavily on Theorem 18.

2.10.1 Trivial Generalizations and Known Results

In the cases of $\{Snr, Spr\}$, $\{Snr, Sdr\}$, and $\{Snr, Spr, Sdr\}$, the characterizations from [13] (and Chapter 13 in [12]) and their proofs, although stated in terms of realistic strings, are valid for legal strings in general. The results are given below for completeness. First we restate Lemma 4 and Lemma 7 from [13] respectively, which will be used in our considerations below.

Lemma 20
Let $u = \alpha v \beta$ be a legal string such that v is also a legal string, and let $S \subseteq \{Snr, Spr, Sdr\}$. Then u is successful in S iff both v and $\alpha\beta$ are successful in S.

Lemma 21
Let u be an elementary legal string. Then u is successful in $\{Snr, Spr\}$ iff either u contains at least one positive pointer or $u = pp$ for some $p \in \Pi$.

The following result follows directly from Lemma 20 and Lemma 21. It is the 'legal version' of Theorem 8 in [13], which can be taken almost verbatim.

Theorem 22
Let u be a legal string. Then u is successful in $\{Snr, Spr\}$ iff for all legal substrings v of u, if $v = v_1u_1v_2 \cdots v_ju_jv_{j+1}$, where each u_i is a legal substring, then $v_1v_2 \cdots v_{j+1}$ either contains a positive pointer or is successful in $\{Snr\}$.

The previous theorem can be stated more elegantly in terms of connected components of the overlap graph of u, see [12, p.141]. Note that characterization

for case $\{Snr, Spr\}$ refers to the case of $\{Snr\}$. The latter case does differ from the realistic characterization in [13], and is treated later.

Theorem 23
Let u be a legal string. Then u is successful in $\{Snr, Sdr\}$ iff all the pointers in u are negative.

We give now the legal version of Theorem 9.1 in [12] — it is a direct consequence of Theorems 22 and 23. Without restrictions on the types of reduction rules used, every legal string is successful, cf. the remark below the definition of the reduction rules, in Section 2.4.

Theorem 24
Every legal string is successful in $\{Snr, Spr, Sdr\}$.

2.10.2 Non-Trivial Generalizations

The following theorem is the legal counterpart of Theorem 6 in [13]. It turns out to be much less restrictive than the original realistic version.

Theorem 25
Let u be a legal string. Then u is successful in $\{Snr\}$ iff u consists of negative pointers only and no two pointers overlap in u.

Proof
The condition from the statement of the lemma is obviously necessary, because **snr** cannot resolve overlapping or positive pointers. We will now prove that this condition is also sufficient. If no two pointers overlap in u, then there must be a substring pp or $p\bar{p}$ of u for some pointer p. If moreover u consists of negative pointers only, then pp is a substring of u. So $\mathbf{snr}_p$ is applicable to u. Now, again no two pointers overlap in legal string $\mathbf{snr}_p(u)$, and $\mathbf{snr}_p(u)$ consists of negative pointers only. By iteration of this argument we conclude that u is successful in $\{Snr\}$. ∎

Observe that the $\{Snr\}$ case is referred to in the characterization of $\{Snr, Spr\}$ in Theorem 22. With the above result we can rephrase the latter result as follows.

Corollary 26
Let u be a legal string. Then u is successful in $\{Snr, Spr\}$ iff for all legal substrings v of u, if $v = v_1u_1v_2 \cdots v_ju_jv_{j+1}$, where each u_i is a legal substring, then, if $v_1v_2 \cdots v_{j+1}$ consists of negative pointers only, they are nonoverlapping.

The following result follows directly from Theorem 18; a successful reduction without string negative rules means that the reduction graph has a single (linear) connected component.

Theorem 27
Let u be a legal string. Then u is successful in $\{Spr, Sdr\}$ iff the reduction graph of u has no cyclic component.

Theorem 14 in [13] is the realistic predecessor of this result, but instead of cyclic components it uses disjoint cycles, cf. Remark 1. The latter notion cannot be used in the general case, as, e.g., the legal string $23\bar{3}24\bar{4}$ has no disjoint cycle, but its reduction graph has one cyclic component. Obviously, the only way to reduce this string is to apply $\mathbf{spr}_3$ and $\mathbf{spr}_4$ (in either order) and then to apply $\mathbf{snr}_2$. In particular, the converse of Corollary 13.1 in [12] does not hold.

In the same way as Theorem 27 relates to Theorem 14 in [13], the following theorem and lemma relate to Theorem 11 and Lemma 12 from [13], respectively.

Theorem 28
Let u be a legal string. Then u is successful in $\{Sdr\}$ iff u consists of negative pointers only and $\mathcal{R}_{u,\varnothing}$ has no cyclic component.

Proof
The forward implication follows directly from Theorem 18 and the fact that **sdr** cannot resolve positive pointers. To prove the reverse implication, let u consist of negative pointers only, and let the corresponding reduction graph $\mathcal{R}_{u,\varnothing}$ have no cyclic component. By Theorem 27, there is a successful $\{Spr, Sdr\}$-reduction φ of u. Since u consists of negative pointers only, φ is a successful $\{Sdr\}$-reduction of u (as applications of string double rules do not introduce positive pointers). ■

Lemma 29
Let u be an elementary legal string. Then u is successful in $\{Spr\}$ iff u contains a positive pointer and $\mathcal{R}_{u,\varnothing}$ has no cyclic component.

Proof
The forward implication follows directly from Theorem 18. To prove the reverse implication, let u contain a positive pointer and let $\mathcal{R}_{u,\varnothing}$ have no cyclic component. By Lemma 21, there is a successful $\{Snr, Spr\}$-reduction φ of u. By Theorem 18, φ is a $\{Spr\}$-reduction of u. ■

The following result follows directly from Lemmas 20 and 29 — it relates to Theorem 13 in [13].

Theorem 30
Let u be a legal string. Then u is successful in $\{Spr\}$ iff for all legal substrings v of u, if $v = v_1u_1v_2 \cdots v_ju_jv_{j+1}$, where each u_i is a legal substring, then $v_1v_2 \cdots v_{j+1}$ either is λ or contains a positive pointer and its reduction graph has no cyclic component.

Similarly to Theorem 22, the previous theorem can be stated in terms of connected components of the overlap graph of u.

Recall that for legal string u and $D \subseteq dom(u)$, $\mathcal{R}_{rem_D(u),\varnothing}$ is isomorphic to $\mathcal{R}_{u,D}$ modulo the labels of the edges. Then, by Theorems 11 and 27, we have the following corollary. In this result it is especially apparent that both the linear component *and* the cyclic components of reduction graphs reveal crucial properties concerning reducibility.

Corollary 31
Let u and v be legal strings with $D = dom(v) \subseteq dom(u)$. Then u is reducible to v in $\{Spr, Sdr\}$ iff $\mathcal{R}_{u,D}$ has no cyclic component and $red(\mathcal{R}_{u,D}) = v$.

2.11 Discussion

This chapter introduces the concept of breakpoint graph (or reality and desire diagram) into gene assembly models, through the notion of reduction graph. The reduction graph provides surprisingly valuable insights into the gene assembly process. First, it allows one to characterize which gene patterns can occur during the transformation of a given gene from its MIC form to its MAC form. Formally, in the string pointer reduction system we characterize whether a legal string u is reducible to a legal string v for a given set of reduction rule types. The characterization is independent from the chosen subset of the three types of string pointer rules, and it allows us to determine whether a legal string u is reducible to a legal string v in linear time. This generalizes the characterization of successfulness in [13], since the reduced string need not be the empty string. Secondly, the reduction graph allows one to determine the number of loop recombination operations that are necessary in the transformation of a given gene from its MIC form to its MAC form. This result allows for a second generalization of the characterization of successfulness, since we consider legal strings instead of realistic strings.

Reduction graphs are defined for legal strings, the basic notion of the string pointer reduction system that represents the genes. Future research could focus on the possibility of defining a similar notion for overlap graphs, which are used in the the graph pointer reduction system — a model (almost) equivalent to the string pointer reduction system. This would allow results in this chapter to be carried over to the graph pointer reduction system.