Fundamenta Informaticae 123 (2013) 199–226
DOI 10.3233/FI-2012-807
IOS Press

Making DNA Expressions Minimal

Rudy van Vliet[*], **Hendrik Jan Hoogeboom**
Leiden Institute of Advanced Computer Science, Leiden University
Niels Bohrweg 1, 2333 CA Leiden, The Netherlands
rvvliet@liacs.nl

Abstract. DNA expressions constitute a formal notation for DNA molecules that may contain nicks and gaps. Different DNA expressions may denote the same DNA molecule. We describe an algorithm to rewrite a given DNA expression into a DNA expression of minimal length denoting the same molecule.

Keywords: DNA molecules, formal notation, minimal DNA expressions, algorithm, complexity

1. Introduction

In the past twenty-five years, DNA computing has become a flourishing research area. Since [8] and [1], 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., [13] and [3]. Current topics of interest include, a.o., gene assembly in ciliates, DNA sequence design, self-assembly and nanotechnology, see, e.g., [6], [9], [19], [12], [7] and [4]. The basic concepts of DNA computing are described in [11].

Despite the growing interest in DNA computing, not much attention is paid in literature to formal ways to denote the DNA molecules – exceptions are [2] and [10]. Formal notations can, however, be useful, e.g., to precisely denote molecules and to compactly describe the computations carried out using them.

In [17] and [18], 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

[*]Address for correspondence: Leiden Institute of Advanced Computer Science, Leiden University, Niels Bohrweg 1, 2333 CA Leiden, The Netherlands

gaps (missing nucleotides in one of the strands). Different DNA expressions may denote the same DNA molecule. Such DNA expressions are called *equivalent*. In [18], it is explained how to construct *minimal* DNA expressions: the shortest possible DNA expressions denoting a given molecule. Different types of molecules are dealt with by different constructions. Minimal DNA expressions can be characterized by six syntactic properties, which are described in [17]. This characterization makes it easy to decide whether or not a given DNA expression is minimal.

For a given DNA expression E, one may want to find an equivalent, minimal DNA expression, e.g., in order to save space for storing the description of a DNA molecule. A natural way to achieve this, consists of two steps: (1) to determine the molecule denoted by E, and (2) to use the constructions from [18] to obtain a minimal DNA expression for that molecule. In this paper, we present a different approach. We describe an algorithm, which *directly* rewrites E into an equivalent, minimal DNA expression. In fact, the algorithm modifies the DNA expression in such a way that it achieves the six properties mentioned above (and thus becomes minimal), but still denotes the same molecule. This approach is elegant, because it operates at the level of DNA expressions only, rather than to refer to the DNA molecules they denote.

The paper is organized as follows. In Sections 2 and 3, we recall some definitions and results we have published before. In particular, in Section 2, we discuss formal DNA molecules and DNA expressions in general, and in Section 3, we consider minimal DNA expressions. In Section 4, we present the algorithm to rewrite an arbitrary DNA expression into an equivalent, minimal DNA expression. The correctness of the algorithm and its complexity are considered in Section 5. Finally, in Section 6, we draw the conclusions from this paper.

This paper is the first part of a diptych. In part 2, [16], we describe a normal form for the DNA expressions. As we see then, the algorithm described in the present paper may also serve as the first step of an algorithm to efficiently rewrite arbitrary DNA expressions into this normal form. More details about the results in both papers can be found in the technical report [15].

2. Formal DNA molecules and DNA expressions

We use the alphabet $\mathcal{N} = \{\text{A, C, G, T}\}$ to denote the four possible nucleotides in a DNA molecule. The nucleotides A and T are each other's Watson-Crick complements: if A is at a certain position in one strand of a double-stranded DNA molecule, then the nucleotide at the corresponding position in the other strand must be T, and vice versa. Likewise, C and G are each other's complements.

The elements of $\mathcal{N}$ are called $\mathcal{N}$*-letters*. A non-empty string α over $\mathcal{N}$ is an $\mathcal{N}$*-word*. The function c over $\mathcal{N}^*$ produces the element-wise (non-reversed) Watson-Crick complement of its argument. Technically speaking, c is a letter-to-letter morphism. For example, $c(\text{ACATG}) = \text{TGTAC}$.

Formal DNA molecules constitute the semantical basis of our language. They are strings over the alphabet $\mathcal{A}_{\triangledown\vartriangle} = \mathcal{A}_\pm \cup \mathcal{A}_+ \cup \mathcal{A}_- \cup \{^\triangledown, {}_\vartriangle\}$, where $\mathcal{A}_\pm = \{\binom{\text{A}}{\text{T}}, \binom{\text{C}}{\text{G}}, \binom{\text{G}}{\text{C}}, \binom{\text{T}}{\text{A}}\}$, $\mathcal{A}_+ = \{\binom{\text{A}}{-}, \binom{\text{C}}{-}, \binom{\text{G}}{-}, \binom{\text{T}}{-}\}$, and $\mathcal{A}_- = \{\binom{-}{\text{A}}, \binom{-}{\text{C}}, \binom{-}{\text{G}}, \binom{-}{\text{T}}\}$. The elements of $\mathcal{A} = \mathcal{A}_\pm \cup \mathcal{A}_+ \cup \mathcal{A}_-$ are called $\mathcal{A}$*-letters*. The elements of $\mathcal{A}_+$ and $\mathcal{A}_-$ correspond to gaps in the lower strand and upper strand of the DNA molecule, respectively. The symbols $^\triangledown$ and ${}_\vartriangle$ are called *nick letters*; $^\triangledown$ is the upper nick letter and ${}_\vartriangle$ is the lower nick letter. They represent nicks (missing bonds) in the upper strand and lower strand, respectively.

Not all strings over $\mathcal{A}_{\triangledown\vartriangle}$ are formal DNA molecules. The strings must satisfy certain local constraints that, a.o., prevent the molecule from 'falling apart'.

Definition 1. A formal DNA molecule is a string $X = x_1x_2 \dots x_r$ with $r \geq 1$ and for $i = 1, \dots, r$, $x_i \in \mathcal{A}_{\triangledown\triangle}$, satisfying

1. if $x_i \in \mathcal{A}_+$, then $x_{i+1} \notin \mathcal{A}_-$ $\quad (i = 1, 2, \dots, r-1)$,
   if $x_i \in \mathcal{A}_-$, then $x_{i+1} \notin \mathcal{A}_+$ $\quad (i = 1, 2, \dots, r-1)$,
2. $x_1, x_r \in \mathcal{A}$,
3. if $x_i \in \{^\triangledown, {}_\triangle\}$, then $x_{i-1}, x_{i+1} \in \mathcal{A}_\pm$ $\quad (i = 2, 3, \dots, r-1)$.

Examples of formal DNA molecules are

$$X_1 = \binom{\mathrm{A}}{\mathrm{T}}\binom{\mathrm{C}}{\mathrm{G}}\binom{\mathrm{A}}{\mathrm{T}}\binom{\mathrm{T}}{\mathrm{A}}\binom{\mathrm{G}}{\mathrm{C}}, \tag{1}$$

$$X_2 = \binom{\mathrm{A}}{\mathrm{T}}{}^\triangledown\binom{\mathrm{C}}{\mathrm{G}}\binom{\mathrm{A}}{\mathrm{T}}{}_\triangle\binom{\mathrm{T}}{\mathrm{A}}\binom{\mathrm{G}}{-}, \tag{2}$$

$$X_3 = \binom{-}{\mathrm{T}}\binom{\mathrm{C}}{\mathrm{G}}\binom{\mathrm{A}}{-}\binom{\mathrm{T}}{-}\binom{\mathrm{G}}{\mathrm{C}}, \tag{3}$$

$$X_4 = \binom{\mathrm{A}}{\mathrm{T}}\binom{\mathrm{C}}{\mathrm{G}}{}_\triangle\binom{\mathrm{A}}{\mathrm{T}}\binom{\mathrm{T}}{-}\binom{\mathrm{G}}{\mathrm{C}}. \tag{4}$$

Because X_1 and X_3 do not contain nick letters, they are called *nick free*.

We define three functions on formal DNA molecules X: $L(X)$ and $R(X)$ are the leftmost letter and the rightmost letter of X, respectively, and $|X|_\mathcal{A}$ counts the $\mathcal{A}$-letters occurring in X. For example, $L(X_2) = \binom{\mathrm{A}}{\mathrm{T}}$, $R(X_2) = \binom{\mathrm{G}}{-}$ and $|X_2|_\mathcal{A} = 5$.

Usually, we abbreviate the notation of a formal DNA molecule X. If X has a substring $\binom{a_i}{c(a_i)} \dots$ $\dots \binom{a_j}{c(a_j)}$, then we write $\binom{a_i \dots a_j}{c(a_i \dots a_j)}$ instead. Likewise, we may write $\binom{a_i \dots a_j}{-}$ and $\binom{-}{a_i \dots a_j}$. A *double component* of a formal DNA molecule X is a maximal (non-empty) substring of X of the form $\binom{a_i \dots a_j}{c(a_i \dots a_j)}$. An *upper component* (of the form $\binom{a_i \dots a_j}{-}$) and a *lower component* (of the form $\binom{-}{a_i \dots a_j}$) are defined analogously. Upper components and lower components are also called *single-stranded components*. The nick letters occurring in a formal DNA molecule are components by themselves.

The *decomposition* of a formal DNA molecule X is a sequence $x'_1, \dots, x'_k$ for some $k \geq 1$, such that $X = x'_1 \dots x'_k$ and each x'_i is a component of X. For the ease of notation, we usually write $x'_1 \dots x'_k$ instead of $x'_1, \dots, x'_k$ to denote the decomposition. The decompositions of the four example formal DNA molecules are

$$\begin{aligned} X_1 &= \binom{\mathrm{ACATG}}{\mathrm{TGTAC}} \text{ with } k = 1, \\ X_2 &= \binom{\mathrm{A}}{\mathrm{T}}{}^\triangledown\binom{\mathrm{CA}}{\mathrm{GT}}{}_\triangle\binom{\mathrm{T}}{\mathrm{A}}\binom{\mathrm{G}}{-} \text{ with } k = 6, \\ X_3 &= \binom{-}{\mathrm{T}}\binom{\mathrm{C}}{\mathrm{G}}\binom{\mathrm{AT}}{-}\binom{\mathrm{G}}{\mathrm{C}} \text{ with } k = 4, \\ X_4 &= \binom{\mathrm{AC}}{\mathrm{TG}}{}_\triangle\binom{\mathrm{A}}{\mathrm{T}}\binom{\mathrm{T}}{-}\binom{\mathrm{G}}{\mathrm{C}} \text{ with } k = 5. \end{aligned}$$

DNA expressions represent formal DNA molecules. We recall the formalism introduced in [17]. DNA expressions are the result of applying the *operators* $\uparrow$, $\downarrow$ and $\updownarrow$ to $\mathcal{N}$-words. The scope of each operator occurring in a DNA expression is delimited by brackets $\langle$ and $\rangle$. Hence, DNA expressions are strings over the alphabet $\Sigma_\mathcal{D} = \{\mathrm{A}, \mathrm{C}, \mathrm{G}, \mathrm{T}, \uparrow, \downarrow, \updownarrow, \langle\, , \rangle\}$. However, not all strings over $\Sigma_\mathcal{D}$ are DNA expressions; DNA expressions have a certain syntax.

For example, an $\uparrow$-*expression* is of the form $\langle\uparrow \varepsilon_1 \dots \varepsilon_n\rangle$ for some $n \geq 1$, where $\varepsilon_1, \dots, \varepsilon_n$ are the *arguments* of $\uparrow$. The arguments may be either $\mathcal{N}$-words or other DNA expressions. Analogously, we have $\downarrow$-*expressions* $\langle\downarrow \varepsilon_1 \dots \varepsilon_n\rangle$. The operator $\updownarrow$ has only one argument ε_1, which may again be either an $\mathcal{N}$-word or a DNA expression, yielding $\langle\updownarrow \varepsilon_1\rangle$.

The effect of the operator $\uparrow$ is three-fold: (1) it interprets an argument that is an $\mathcal{N}$-word α as the upper strand $\binom{\alpha}{-}$, (2) it removes the nicks occurring in the upper strands of its arguments, and (3) it connects the upper strands of consecutive arguments. For the last effect to be possible, the arguments must *fit together by upper strands*, i.e., if ε_i and ε_{i+1} are consecutive arguments, and X_i and X_{i+1} are the formal DNA molecules represented by these arguments, then both $R(X_i)$ and $L(X_{i+1})$ must be elements of $\mathcal{A}_\pm \cup \mathcal{A}_+$. Otherwise, there would be a gap in the upper strand 'between' the two arguments. This is a semantical restriction in the syntactical description. Where $\uparrow$ connects the upper strands of consecutive arguments, it does not connect the lower strands. Hence, if both $R(X_i)$ and $L(X_{i+1})$ are elements of $\mathcal{A}_\pm$, then a nick is introduced into the lower strand.

The effect of the operator $\downarrow$ is analogous to that of $\uparrow$. We should simply read 'lower' for 'upper' and vice versa. The operator $\updownarrow$ does something completely different: it fills the gaps occurring in both strands of its (single) argument ε_1. It provides the missing nucleotides and connects them to their neighbours. The operator $\updownarrow$ neither introduces nor removes nicks in its argument. If ε_1 is an $\mathcal{N}$-word α_1, then $\updownarrow$ interprets it as the upper strand $\binom{\alpha_1}{-}$, before it performs its action.

The concatenation of two or more DNA expressions is *not* a DNA expression. DNA expressions can be concatenated only as arguments of $\uparrow$-expressions or $\downarrow$-expressions.

The *semantics* of a DNA expression E is the formal DNA molecule represented by E and is denoted by $\mathcal{S}(E)$. The simplest DNA expressions are $\langle\uparrow \alpha_1\rangle$, $\langle\downarrow \alpha_1\rangle$ and $\langle\updownarrow \alpha_1\rangle$ for an $\mathcal{N}$-word α_1, with $\mathcal{S}(\langle\uparrow \alpha_1\rangle) = \binom{\alpha_1}{-}$, $\mathcal{S}(\langle\downarrow \alpha_1\rangle) = \binom{-}{\alpha_1}$ and $\mathcal{S}(\langle\updownarrow \alpha_1\rangle) = \binom{\alpha_1}{c(\alpha_1)}$. Less trivial examples are

$$E_1 = \langle\updownarrow \langle\uparrow \langle\updownarrow \alpha_1\rangle\, \alpha_2\rangle\rangle\,, \text{ with } \mathcal{S}(E_1) = \tbinom{\alpha_1\alpha_2}{c(\alpha_1\alpha_2)}, \tag{5}$$

$$E_2 = \langle\uparrow \langle\downarrow \alpha_1\, \langle\updownarrow \alpha_2\rangle\rangle\, \alpha_3\, \langle\updownarrow \alpha_4\rangle\rangle\,, \text{ with } \mathcal{S}(E_2) = \tbinom{-}{\alpha_1}\tbinom{\alpha_2}{c(\alpha_2)}\tbinom{\alpha_3}{-}\tbinom{\alpha_4}{c(\alpha_4)}, \tag{6}$$

$$E_3 = \langle\downarrow \langle\updownarrow \alpha_1\rangle\, \langle\updownarrow \alpha_2\rangle\rangle\,, \text{ with } \mathcal{S}(E_3) = \tbinom{\alpha_1}{c(\alpha_1)}^{\triangledown}\tbinom{\alpha_2}{c(\alpha_2)}, \tag{7}$$

$$E_4 = \langle\uparrow \langle\downarrow \langle\updownarrow \alpha_1\rangle\, \langle\updownarrow \alpha_2\rangle\rangle\, \langle\updownarrow \alpha_3\rangle\, \alpha_4\, \langle\updownarrow \alpha_5\rangle\rangle\,,$$
$$\text{with } \mathcal{S}(E_4) = \tbinom{\alpha_1\alpha_2}{c(\alpha_1\alpha_2)}_{\vartriangle}\tbinom{\alpha_3}{c(\alpha_3)}\tbinom{\alpha_4}{-}\tbinom{\alpha_5}{c(\alpha_5)}. \tag{8}$$

On the other hand, the string $\langle\uparrow \langle\downarrow \langle\updownarrow \alpha_1\rangle\, \alpha_2\rangle\, \alpha_3\, \langle\updownarrow \alpha_4\rangle\rangle$, which is not too different from E_2, is not a DNA expression, because the first argument $\langle\downarrow \langle\updownarrow \alpha_1\rangle\, \alpha_2\rangle$ (which denotes $\binom{\alpha_1}{c(\alpha_1)}\binom{-}{\alpha_2}$) and the second argument α_3 (which corresponds to $\binom{\alpha_3}{-}$) of the operator $\uparrow$ do not fit together by upper strands.

Let ε_i be an argument of a DNA expression E. If ε_i is an $\mathcal{N}$-word, then it is called an *$\mathcal{N}$-word-argument*. Otherwise, it is an *expression-argument*. If ε_i is an $\uparrow$-expression (or $\downarrow$-expression or $\updownarrow$-expression), then it is also called an *$\uparrow$-argument* (*$\downarrow$-argument* or *$\updownarrow$-argument*, respectively). At some point in this paper, 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*.

A substring E^s of a DNA expression E that is itself a DNA expression, is called a *DNA subexpression* of E. If $E^s \neq E$, then it is a *proper DNA subexpression*. If E^s is an $\uparrow$-expression, then it is an $\uparrow$-subexpression of E. A $\downarrow$-subexpression and an $\updownarrow$-subexpression are defined analogously.

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 in (8), the parent operator of the $\mathcal{N}$-word α_4 is the operator $\uparrow$.

The *outermost operator* of a DNA expression E is (the occurrence of) the operator that governs the entire DNA expression. For example, if E is an $\uparrow$-expression $\langle\uparrow \varepsilon_1 \dots \varepsilon_n\rangle$, then the outermost operator of E is (the first occurrence of) $\uparrow$. All other occurrences of operators in E are called *inner occurrences*.

An occurrence of the operator $\uparrow$ or $\downarrow$ is called *alternating*, if its arguments are $\mathcal{N}$-words and DNA expressions, alternately. In this definition, we consider consecutive $\mathcal{N}$-word-arguments as a single argument. Hence, the operator is alternating, if and only if it does not have two (or more) consecutive expression-arguments. If the outermost operator of an $\uparrow$-expression or a $\downarrow$-expression E is alternating, then E itself is also called alternating.

A formal DNA molecule is called *expressible*, if there exists a DNA expression denoting it. As mentioned in, e.g., [17], not all formal DNA molecules are expressible:

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.

Hence, X is expressible, if and only if X is nick free, or X only contains nicks in the upper strand, or X only contains nicks in the lower strand. The formal DNA molecules X_1, X_3 and X_4 from (1), (3) and (4) are expressible. For example, proper choices for the α_i's in (5), (6) and (8) yield DNA expressions denoting these molecules. On the other hand, the molecule X_2 from (2) is not expressible.

3. Minimal DNA expressions

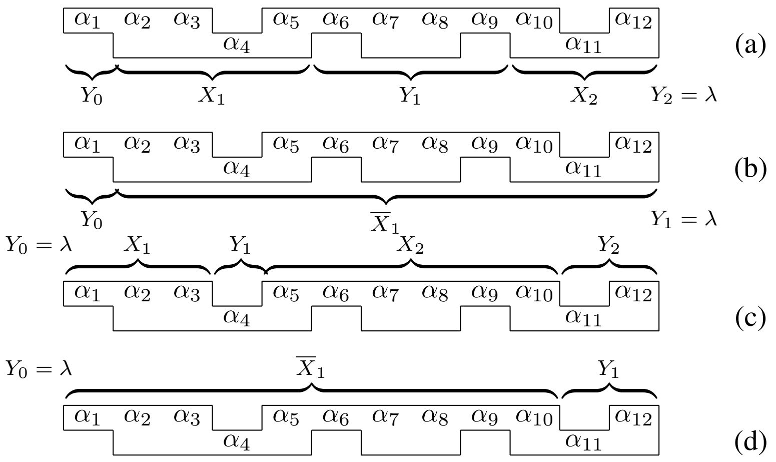
For each expressible formal DNA molecule, there exist infinitely many DNA expressions denoting it. For example, if E is an $\uparrow$-expression $\langle\uparrow \varepsilon_1 \dots \varepsilon_n\rangle$ denoting a formal DNA molecule X, then so is $\langle\uparrow E\rangle = \langle\uparrow \langle\uparrow \varepsilon_1 \dots \varepsilon_n\rangle\rangle$. DNA expressions E and E' with the same semantics are called *equivalent*, and we write $E \equiv E'$. The *length* of a DNA expression E, denoted by $|E|$, is the number of letters from $\Sigma_\mathcal{D}$ occurring in E. Clearly, the DNA expression E is three letters shorter than the equivalent DNA expression $\langle\uparrow E\rangle$. A DNA expression E is *minimal*, if for each equivalent DNA expression E', we have $|E'| \geq |E|$.

In [18], we have described how to construct minimal DNA expressions for all types of expressible formal DNA molecules. Here, we only give a complete description for *nick free* molecules. First, we consider perfect double-stranded molecules:

Theorem 3. 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 $\mathcal{S}(E) = \binom{\alpha_1}{c(\alpha_1)}$.

We need some additional terminology to describe the minimal DNA expressions denoting nick free formal DNA molecules with at least one single-stranded component.

Definition 4. Let X be a nick free formal DNA molecule.
A primitive lower block X_1 of X is a maximal substring of X consisting of only lower components and double components, and containing at least one lower component.
The primitive lower block partitioning of X is the sequence of strings $Y_0, X_1, Y_1, \dots, X_{r_0}, Y_{r_0}$ with $r_0 \geq 0$, such that $X = Y_0 X_1 Y_1 \dots X_{r_0} Y_{r_0}$ and $X_1, \dots, X_{r_0}$ are all primitive lower blocks of X.



α_1 α_2 α_3 α_4 α_5 α_6 α_7 α_8 α_9 α_{10} α_{11} α_{12}
(a) Y_0 X_1 Y_1 X_2 $Y_2 = \lambda$
(b) Y_0 $\overline{X}_1$ $Y_1 = \lambda$
(c) $Y_0 = \lambda$ X_1 Y_1 X_2 Y_2
(d) $Y_0 = \lambda$ $\overline{X}_1$ Y_1

Figure 1. Pictorial representation of the formal DNA molecule X from (9). (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.

The definitions of a primitive upper block and the primitive upper block partitioning of X are completely analogous: we only have to substitute all occurrences of "lower" by "upper". Usually, we 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 denote a primitive lower (or upper) block partitioning.

In Figure 1(a) and (c), we have depicted the primitive lower block partitioning and the primitive upper block partitioning of the formal DNA molecule [1]

$$X = \binom{\alpha_1}{-}\binom{\alpha_2\alpha_3}{c(\alpha_2\alpha_3)}\binom{-}{\alpha_4}\binom{\alpha_5}{c(\alpha_5)}\binom{\alpha_6}{-}\binom{\alpha_7\alpha_8}{c(\alpha_7\alpha_8)}\binom{\alpha_9}{-}\binom{\alpha_{10}}{c(\alpha_{10})}\binom{-}{\alpha_{11}}\binom{\alpha_{12}}{c(\alpha_{12})}. \tag{9}$$

Note that the string Y_2 occurring in the primitive lower block partitioning and the string Y_0 occurring in the primitive upper block partitioning equal the empty string λ.

We define three counting functions on nick free formal DNA molecules:

Definition 5. Let X be a nick free formal DNA molecule.

- $B_{\downarrow}(X)$ is the number of primitive lower blocks of X;
- $B_{\uparrow}(X)$ is the number of primitive upper blocks of X;
- $n_{\updownarrow}(X)$ is the number of double components of X.

For the formal DNA molecule X from (9), we have $B_{\downarrow}(X) = B_{\uparrow}(X) = 2$ and $n_{\updownarrow}(X) = 5$.

Note that a perfect double-stranded molecule $X = \binom{\alpha_1}{c(\alpha_1)}$ does not have any primitive lower block or primitive upper block. Hence, both the primitive lower block partitioning and the primitive upper block partitioning of X reduce to Y_0, with $Y_0 = X$. This is the only nick free formal DNA molecule X for which $B_{\downarrow}(X) = B_{\uparrow}(X) = 0$.

[1]We could have used a single symbol α to denote the concatenation of the two $\mathcal{N}$-words α_2 and α_3 determining the first double component of X. However, in Example 12, it will appear to be useful to have α_2 and α_3 separated. The same goes for α_7 and α_8.

Definition 6. Let X be a nick free formal DNA molecule and let $Y_0X_1Y_1 \dots X_{r_0}Y_{r_0}$ with $r_0 = B_\downarrow(X) \geq 0$ be the primitive lower block partitioning of X.

A lower block $\overline{X}_1$ of X is a substring of X that starts and ends with a primitive lower block. Hence, $\overline{X}_1 = X_{j_1}Y_{j_1} \dots X_{j_2}$ for some j_1 and j_2 with $1 \leq j_1 \leq j_2 \leq r_0$.

A lower block partitioning of X is a sequence of strings $Y_0, \overline{X}_1, Y_1, \dots, \overline{X}_r, Y_r$ for some $r \geq 0$, such that $X = Y_0\overline{X}_1Y_1 \dots \overline{X}_rY_r$, and each $\overline{X}_j$ is a lower block of X, and each primitive lower block of X is contained in an $\overline{X}_j$.

Hence, a lower block partitioning of X is a partitioning of X based on (disjoint) lower blocks, which together contain all primitive lower blocks. As each lower component is part of a primitive lower block, the substrings Y_j occurring in a lower block partitioning consist of only upper components and double components.

An upper block and an upper block partitioning of X are defined completely analogously. For notational convenience, we usually write $Y_0\overline{X}_1Y_1 \dots \overline{X}_rY_r$ instead of $Y_0, \overline{X}_1, Y_1, \dots, \overline{X}_r, Y_r$ to denote a lower (or upper) block partitioning.

If each lower block in a lower block partitioning contains exactly one primitive lower block, then we have the primitive lower block partitioning. The formal DNA molecule from (9) has one more lower block partitioning, which we have depicted in Figure 1(b). It also has another upper block partitioning, which we have depicted in Figure 1(d).

We now have the tools to construct minimal DNA expressions for nick free formal DNA molecules with at least one single-stranded component.

Theorem 7. 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 $b_j \leq k$; [2]
   - for $j = 0, 1, \dots, r$ and for $i = a_j, \dots, b_j$, let

   $$\varepsilon_i = \begin{cases} \alpha_i & \text{if } x'_i = \binom{\alpha_i}{-} \text{ for an } \mathcal{N}\text{-word } \alpha_i \\ \langle \updownarrow \alpha_i \rangle & \text{if } x'_i = \binom{\alpha_i}{c(\alpha_i)} \text{ for an } \mathcal{N}\text{-word } \alpha_i; \end{cases} \quad \text{and} \tag{10}$$

   - let

   $$E = \langle \uparrow \varepsilon_{a_0} \dots \varepsilon_{b_0} E_1 \varepsilon_{a_1} \dots \varepsilon_{b_1} \ \dots \ E_r \varepsilon_{a_r} \dots \varepsilon_{b_r} \rangle \, . \tag{11}$$

   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

[2] In principle, the indices a_j and b_j satisfy $1 \leq a_j \leq b_j \leq k$. However, if $Y_j = \lambda$ (which is possible for $j = 0$ and $j = r$), then by definition $a_j = b_j + 1$.

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

$$|E| = 3 + 3 \cdot B_{\downarrow}(X) + 3 \cdot n_{\updownarrow}(X) + |X|_{\mathcal{A}}. \tag{12}$$

2. If $B_{\downarrow}(X) \geq B_{\uparrow}(X)$, then ... (symmetric to Claim 1).

All minimal DNA expressions denoting a nick free formal DNA molecule with at least one single-stranded component satisfy the above construction. (The molecules without single-stranded components were covered by Theorem 3.)

Note that if $B_{\uparrow}(X) = B_{\downarrow}(X) \geq 1$, then there exist both minimal $\uparrow$-expressions and minimal $\downarrow$-expressions denoting X, obviously with the same length. Moreover, if $B_{\uparrow}(X) \geq B_{\downarrow}(X) \geq 2$, then the lower block partitioning of X is not unique: different lower block partitionings yield different minimal $\uparrow$-expressions. Of course, we have an analogous property if $B_{\downarrow}(X) \geq B_{\uparrow}(X) \geq 2$.

This holds in particular for the formal DNA molecule X from Figure 1, for which $B_{\downarrow}(X) = B_{\uparrow}(X) = 2$. As we see in the example below, there are two minimal $\uparrow$-expressions denoting X, corresponding to the two lower block partitionings of X, and there are two minimal $\downarrow$-expressions denoting X, corresponding to the two upper block partitionings of X.

Example 8. Let X be the nick free formal DNA molecule from (9) and consider the lower block partitioning $Y_0\overline{X}_1Y_1$ from Figure 1(b). Here $Y_0 = \binom{\alpha_1}{-}$,

$$\overline{X}_1 = \binom{\alpha_2\alpha_3}{c(\alpha_2\alpha_3)}\binom{-}{\alpha_4}\binom{\alpha_5}{c(\alpha_5)}\binom{\alpha_6}{-}\binom{\alpha_7\alpha_8}{c(\alpha_7\alpha_8)}\binom{\alpha_9}{-}\binom{\alpha_{10}}{c(\alpha_{10})}\binom{-}{\alpha_{11}}\binom{\alpha_{12}}{c(\alpha_{12})}$$

and $Y_1 = \lambda$. By Theorem 7(1), the resulting minimal $\uparrow$-expression is $E_b = \langle\uparrow \alpha_1 E_1\rangle$, where E_1 is a minimal DNA expression denoting $\overline{X}_1$.

As $B_{\downarrow}(\overline{X}_1) = 2 > B_{\uparrow}(\overline{X}_1) = 1$, E_1 must be a $\downarrow$-expression, which is recursively constructed using Theorem 7(2). The result is

$$E_1 = \langle\downarrow \langle\updownarrow \alpha_2\alpha_3\rangle\, \alpha_4 \langle\uparrow \langle\updownarrow \alpha_5\rangle\, \alpha_6 \langle\updownarrow \alpha_7\alpha_8\rangle\, \alpha_9 \langle\updownarrow \alpha_{10}\rangle\rangle\, \alpha_{11} \langle\updownarrow \alpha_{12}\rangle\rangle\,.$$

This way, we can construct the four minimal DNA expressions denoting X:

$$\begin{aligned}
E_a &= \langle\uparrow \alpha_1 \langle\downarrow \langle\updownarrow \alpha_2\alpha_3\rangle\, \alpha_4 \langle\updownarrow \alpha_5\rangle\rangle\, \alpha_6 \langle\updownarrow \alpha_7\alpha_8\rangle\, \alpha_9 \langle\downarrow \langle\updownarrow \alpha_{10}\rangle\, \alpha_{11} \langle\updownarrow \alpha_{12}\rangle\rangle\rangle\,, &(13)\\
E_b &= \langle\uparrow \alpha_1 \langle\downarrow \langle\updownarrow \alpha_2\alpha_3\rangle\, \alpha_4 \langle\uparrow \langle\updownarrow \alpha_5\rangle\, \alpha_6 \langle\updownarrow \alpha_7\alpha_8\rangle\, \alpha_9 \langle\updownarrow \alpha_{10}\rangle\rangle\, \alpha_{11} \langle\updownarrow \alpha_{12}\rangle\rangle\rangle\,, &(14)\\
E_c &= \langle\downarrow \langle\uparrow \alpha_1 \langle\updownarrow \alpha_2\alpha_3\rangle\rangle\, \alpha_4 \langle\uparrow \langle\updownarrow \alpha_5\rangle\, \alpha_6 \langle\updownarrow \alpha_7\alpha_8\rangle\, \alpha_9 \langle\updownarrow \alpha_{10}\rangle\rangle\, \alpha_{11} \langle\updownarrow \alpha_{12}\rangle\rangle\,, &(15)\\
E_d &= \langle\downarrow \langle\uparrow \alpha_1 \langle\downarrow \langle\updownarrow \alpha_2\alpha_3\rangle\, \alpha_4 \langle\updownarrow \alpha_5\rangle\rangle\, \alpha_6 \langle\updownarrow \alpha_7\alpha_8\rangle\, \alpha_9 \langle\updownarrow \alpha_{10}\rangle\rangle\, \alpha_{11} \langle\updownarrow \alpha_{12}\rangle\rangle\,. &(16)
\end{aligned}$$

■

In the above example, each lower block partitioning and each upper block partitioning of X corresponds to a single minimal DNA expression. In general, however, there may be more than one minimal DNA expression E_i denoting a lower block $\overline{X}_i$ occurring in a lower block partitioning. Then the construction from Theorem 7(1) yields more than one minimal $\uparrow$-expression corresponding to the same lower block partitioning. This is the case if $B_{\downarrow}(\overline{X}_i) \geq 3$. Of course, an analogous property holds for minimal $\downarrow$-expressions corresponding to an upper block partitioning. In [17], we address the *number* of minimal DNA expressions denoting a given formal DNA molecule.

For expressible formal DNA molecules with nick letters, the construction of minimal DNA expressions is somewhat similar to that for nick free molecules. If the molecule has lower nick letters, then each minimal DNA expression is an $\uparrow$-expression. It is obtained from lower block partitionings of the nick free pieces of the molecule. We illustrate this by a special example, which is useful later in the paper. In this example, the molecule does not have single-stranded components. A general description and a more general example of the construction can be found in [18].

Example 9. Let

$$X = \binom{\alpha_1}{c(\alpha_1)} \triangle \binom{\alpha_2}{c(\alpha_2)} \triangle \cdots \triangle \binom{\alpha_m}{c(\alpha_m)}$$

for some $m \geq 2$ and $\mathcal{N}$-words $\alpha_1, \ldots, \alpha_m$. The nick free pieces of X are the double components $\binom{\alpha_1}{c(\alpha_1)}, \ldots, \binom{\alpha_m}{c(\alpha_m)}$. For $h = 1, \ldots, m$, $\binom{\alpha_h}{c(\alpha_h)}$ does not contain any primitive lower block. Hence, its only lower block partitioning is $\mathcal{P} = Y_0$, where $Y_0 = \binom{\alpha_h}{c(\alpha_h)}$. We apply the construction from Theorem 7 to this simple partitioning, obtaining the $\uparrow$-expression $\langle \uparrow \langle \updownarrow \alpha_h \rangle \rangle$. We now combine the m $\uparrow$-expressions for the m nick free pieces into a single $\uparrow$-expression, leaving out redundant occurrences of $\uparrow$:

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

This appears to be the unique minimal DNA expression denoting X. ■

When we want to know whether or not a given DNA expression E is minimal, we can compute its semantics $X = \mathcal{S}(E)$ and check if $|E|$ equals the minimal length of DNA expressions denoting X (which is, e.g., given in (12) for nick free molecules X with at least one single-stranded component and $B_\uparrow(X) \geq B_\downarrow(X)$). There is, however, a more elegant method, which does not refer to the semantics at all. It is based on a characterization of minimal DNA expressions by six properties of (the arguments of) the operators occurring in them. We only have to check these properties to decide if E is minimal.

Theorem 10. A DNA expression E is minimal, if and only if

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

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

($\mathcal{D}_{\mathbf{Min}}$.3) unless $E = \langle \uparrow \alpha \rangle$ or $E = \langle \downarrow \alpha \rangle$ for an $\mathcal{N}$-word α, each occurrence of an operator $\uparrow$ or $\downarrow$ in E has at least two arguments; and

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

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

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

- either its first argument is an $\mathcal{N}$-word α or an $\updownarrow$-expression $\langle \updownarrow \alpha \rangle$ for an $\mathcal{N}$-word α,

Table 1. Examples of DNA expressions with all six properties from Theorem 10 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.

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

- 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.

This characterization also applies to minimal DNA expressions with nicks. The formal proof of the result can be found in [14]. Here, we give a few examples and provide some intuition for each of the properties.

In Table 1, for each property, we give one or two DNA expressions that lack only that property, and thus are not minimal. These examples demonstrate that none of the properties follows from the remaining five properties, and could therefore be omitted.

Now, let E be an arbitrary minimal DNA expression. First, we observe that each DNA subexpression E^s of E must also be minimal. Otherwise, we could substitute E^s in E by a shorter, equivalent DNA subexpression. The resulting overall DNA expression would be equivalent to E, but shorter.

In particular, each $\updownarrow$-subexpression E^s of E must be minimal. By Theorem 3, E^s must be of the form $\langle \updownarrow \alpha \rangle$ for an $\mathcal{N}$-word α. This explains Property $(\mathcal{D}_{\text{Min}}.1)$.

Next, suppose that E has an $\uparrow$-subexpression $E^s = \langle \uparrow \varepsilon_1 \dots \varepsilon_n \rangle$ where the ith argument ε_i is an $\uparrow$-argument $\langle \uparrow \varepsilon_{i,1} \dots \varepsilon_{i,m} \rangle$. Because the effect of the outermost operator $\uparrow$ of ε_i on $\varepsilon_{i,1}, \dots, \varepsilon_{i,m}$ can as well be achieved by the outermost operator $\uparrow$ of E^s, we may substitute ε_i by its arguments. The result,

$$\langle \uparrow \varepsilon_1 \dots \varepsilon_{i-1} \varepsilon_{i,1} \dots \varepsilon_{i,m} \varepsilon_{i+1} \dots \varepsilon_n \rangle ,$$

is equivalent to E^s, but three letters shorter. Hence, E^s is not minimal, and neither is E. This explains Property $(\mathcal{D}_{\text{Min}}.2)$.

Intuitively, Property $(\mathcal{D}_{\text{Min}}.3)$ means that the effect of an operator $\uparrow$ or $\downarrow$ with a single argument is often too small to justify the presence of the operator. In particular such an operator cannot express its ability to connect consecutive arguments. Property $(\mathcal{D}_{\text{Min}}.4)$ ensures that the arguments of a minimal DNA expression are nick free. It is not efficient to first introduce nick letters and to later remove them. All nick letters in the formal DNA molecule denoted can be produced by the outermost operator.

Let E^s be a proper $\uparrow$-subexpression of E and let $X^s = \mathcal{S}(E^s)$. By Property $(\mathcal{D}_{\text{Min}}.5)$ (and Property $(\mathcal{D}_{\text{Min}}.4)$), both the first and the last single-stranded component of X^s are upper components. In fact, the submolecule is an upper block and $B_{\uparrow}(X^s) = B_{\downarrow}(X^s) + 1$. This justifies the use of the operator $\uparrow$ here. If either the first or the last single-stranded component were a lower component, then it would be more efficient to have this lower component produced by the parent operator of E^s, which is an occurrence of $\downarrow$.

The *outermost* operator has a weaker property. Assume that the outermost operator is $\uparrow$. By Property ($\mathcal{D}_{\text{Min}}$.6), it is possible that this operator has only one of the two subproperties from Property ($\mathcal{D}_{\text{Min}}$.5) (which is, e.g., the case if the formal DNA molecule X denoted is nick free and $B_{\uparrow}(X) = B_{\downarrow}(X) \geq 1$), or that the operator has two consecutive expression-arguments (as we will see in Lemma 11, this is the case if and only if X contains lower nick letters).

4. An algorithm for minimality

For a given DNA expression E, we may use Theorem 10 to decide whether or not E is minimal. If it is not, then we may wish to obtain an equivalent, minimal DNA expression E'.

An indirect way to achieve this, is by determining the semantics $\mathcal{S}(E)$ of E and then using Theorem 3 or Theorem 7 to construct a minimal DNA expression denoting $\mathcal{S}(E)$.

Here, we work out a different approach. We use a recursive function `MakeMinimal` which *directly* rewrites E into an equivalent, minimal DNA expression, i.e., which does not refer to the semantics $\mathcal{S}(E)$. In `MakeMinimal`, we first substitute the expression-arguments of E by equivalent, minimal DNA expressions. Subsequently, step by step, we rewrite the overall DNA expression in such a way, that it becomes minimal itself.

One of the steps involves a weaker version of the notion of equivalence. As we observed earlier, it is not efficient to first introduce nick letters and to later remove them. Therefore, it may be useful to substitute a proper DNA subexpression of E which does introduce nick letters by a nick free version. To formalize this, we say that a DNA expression E_1 is equivalent to a DNA expression E_2 *post-modulo nicks*, denoted $E_1 \equiv_{\triangledown} E_2$, if $\mathcal{S}(E_1)$ and $\mathcal{S}(E_2)$ are the same, except that $\mathcal{S}(E_2)$ may contain additional nick letters. Note that E_1 is not necessarily nick free.

For example, if

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

then $E_1 \equiv_{\triangledown} E_2$, $E_2 \equiv_{\triangledown} E_3$ and $E_1 \equiv_{\triangledown} E_3$, because

$$\begin{aligned}
\mathcal{S}(E_1) &= \binom{\alpha_1\alpha_2}{c(\alpha_1\alpha_2)}\binom{\alpha_3}{-}\binom{\alpha_4\alpha_5}{c(\alpha_4\alpha_5)}, \\
\mathcal{S}(E_2) &= \binom{\alpha_1\alpha_2}{c(\alpha_1\alpha_2)}\binom{\alpha_3}{-}\binom{\alpha_4}{c(\alpha_4)}\triangle\binom{\alpha_5}{c(\alpha_5)}, \text{ and} \\
\mathcal{S}(E_3) &= \binom{\alpha_1}{c(\alpha_1)}\triangle\binom{\alpha_2}{c(\alpha_2)}\binom{\alpha_3}{-}\binom{\alpha_4}{c(\alpha_4)}\triangle\binom{\alpha_5}{c(\alpha_5)}.
\end{aligned}$$

Note that if $E_1 \equiv E_2$, then certainly $E_1 \equiv_{\triangledown} E_2$.

In Figure 2, we give the pseudo-code of the function `MakeMinimal`. It can be applied both to nick free DNA expressions and to DNA expressions with nicks. The description of the function contains four instructions in a style like

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

```
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 ($E_1$);
6.               if ($E_1$ is an $\updownarrow$-expression)
7.               then substitute $E$ by $E_1$;                              ($\mathcal{D}_{\texttt{Min}}$.1)
8.               else    // $E_1$ is an $\uparrow$-expression or a $\downarrow$-expression
9.                    substitute $E$ by a minimal DNA expression $E'$
                        satisfying $E' \equiv E$; (proc. Make$\updownarrow$ExprMinimal)     ($\mathcal{D}_{\texttt{Min}}$.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 $\langle\uparrow \varepsilon_1 \ldots \varepsilon_n\rangle$ for some $n \geq 1$
               // and $\mathcal{N}$-words and DNA expressions $\varepsilon_1, \ldots, \varepsilon_n$
13.         for ($i$ = 1 to $n$)
14.         do   if ($\varepsilon_i$ is a DNA expression $E_i$)
15.              then MakeMinimal ($E_i$);
16.                   if ($E_i$ is a $\downarrow$-expression which is not alternating)
17.                   then substitute $E_i$ in $E$ by a minimal, nick free
                             DNA expression $E'_i$ satisfying $E'_i \equiv_\triangledown E_i$;
                             (proc. Denickify)                              ($\mathcal{D}_{\texttt{Min}}$.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 a minimal $\uparrow$-expression $E'_i$
                             satisfying $E'_i \equiv E_i$; (proc. RotateToMinimal) ($\mathcal{D}_{\texttt{Min}}$.5)
21.                   fi
22.                   if ($E_i$ is an $\uparrow$-expression)
23.                   then substitute $E_i$ in $E$ by its arguments;        ($\mathcal{D}_{\texttt{Min}}$.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 $E_1$;                              ($\mathcal{D}_{\texttt{Min}}$.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 a minimal $\downarrow$-expression $E'$
                        satisfying $E' \equiv E$; (proc. RotateToMinimal)     ($\mathcal{D}_{\texttt{Min}}$.6)
34.              fi
35.         fi
36.    fi
37.  }
```

Figure 2. Pseudo-code of the recursive function MakeMinimal.

These instructions will be worked out in detail later, by the procedures mentioned between the brackets.

Each substitution in the function is justified by the violation of a particular property from Theorem 10. Such a violation implies that the DNA expression is not (yet) minimal. In the pseudo-code, we indicate the properties involved.

As a result of subsequent substitutions, the DNA expression gets more and more of the properties. In the end, it has all six properties and thus is minimal. This will become clearer, when we apply the function to an example DNA expression. If the original DNA expression is minimal already (and thus has the six properties), then it is not modified by the function `MakeMinimal` at all.

In order to understand one of the steps in `MakeMinimal`, it is useful to first establish that under certain conditions, a DNA expression is nick free, if and only it is alternating.

Lemma 11. Let E be an $\uparrow$-expression with Properties ($\mathcal{D}_{\text{Min}}$.2), ($\mathcal{D}_{\text{Min}}$.4) and ($\mathcal{D}_{\text{Min}}$.5). Then E is nick free, if and only if E is alternating.

There is of course an analogous result for $\downarrow$-expressions. In particular, for a minimal $\uparrow$-expression (or $\downarrow$-expression), the adjectives 'nick free' and 'alternating' are equivalent.

Proof: 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.

By definition, the operators $\uparrow$ and $\downarrow$ may only introduce nick letters between consecutive expression-arguments. By Property ($\mathcal{D}_{\text{Min}}$.4), the arguments of E are nick free. Any nick in E must be introduced by the outermost operator $\uparrow$. If E is alternating, then E is certainly nick free.

Now assume that E is not alternating, i.e., that it has two consecutive expression-arguments ε_i and ε_{i+1}. By Property ($\mathcal{D}_{\text{Min}}$.2), these are $\updownarrow$-expressions and/or $\downarrow$-expressions. If ε_i is an $\updownarrow$-expression, then by definition $R(\mathcal{S}(\varepsilon_i)) \in \mathcal{A}_\pm$. If, on the other hand, ε_i is a $\downarrow$-expression, then by Property ($\mathcal{D}_{\text{Min}}$.5), its last argument is an $\mathcal{N}$-word α or an $\updownarrow$-expression $\langle\updownarrow \alpha\rangle$. It cannot be an $\mathcal{N}$-word α, because in that case, the arguments ε_i and ε_{i+1} of E would not fit together by upper strands, which is required for an $\uparrow$-expression. Hence, the last argument of ε_i is an $\updownarrow$-expression $\langle\updownarrow \alpha\rangle$, and again $R(\mathcal{S}(\varepsilon_i)) \in \mathcal{A}_\pm$.

Analogously, we find that $L(\mathcal{S}(\varepsilon_{i+1})) \in \mathcal{A}_\pm$, both if ε_{i+1} is an $\updownarrow$-expression, and if it is a $\downarrow$-expression. This implies that the outermost operator $\uparrow$ of E introduces a lower nick letter between ε_i and ε_{i+1}. □

The instructions in lines 9, 17, 20 and 33 are not precisely specified. Each of them requires finding a minimal DNA expression satisfying certain requirements. In the context of these instructions in `MakeMinimal`, this can be implemented by local, non-recursive rearrangements of the DNA expression at hand. Before giving more details of how this is achieved, we explain why (additional) recursive calls of `MakeMinimal` would not be appropriate.

For 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 17 is not equivalent to E_i. As follows from Lemma 11, E_i contains nicks, whereas E'_i must be nick free. Because the function `MakeMinimal` yields an *equivalent*, minimal DNA expression, it is not applicable. Apart from that, it would not make sense to call the function here, because we have just done so in line 15.

In line 20, 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 15–18, the $\downarrow$-expression E_i is minimal already. Hence, a call MakeMinimal(E_i) would simply yield E_i. It would never produce the desired $\uparrow$-expression.

Although the situation in line 33 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.

We now work out the instructions in lines 9, 17, 20 and 33. In line 9, we have an $\updownarrow$-expression $E = \langle \updownarrow E_1 \rangle$, where E_1 is a minimal $\uparrow$-expression or $\downarrow$-expression. We need a minimal DNA expression E' satisfying $E' \equiv E$. We use a procedure Make$\updownarrow$ExprMinimal for this.

In order to describe this procedure, we need one more definition. Let E be an arbitrary DNA expression, and let $\alpha_1, \ldots, \alpha_k$ for some $k \geq 1$ be the $\mathcal{N}$-words occurring in E, in the order of their occurrence. Then $\alpha_E = \alpha'_1 \ldots \alpha'_k$, where

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

Hence, we concatenate the $\mathcal{N}$-words occurring in E, possibly after complementation. Intuitively, α_E corresponds to the nucleotides in the upper strand of the complemented version of $\mathcal{S}(E)$. For example, for the DNA expression E_2 from (6), $\alpha_{E_2} = c(\alpha_1)\alpha_2\alpha_3\alpha_4$.

Now, Make$\updownarrow$ExprMinimal is given in Figure 3, for the case that E_1 is an $\uparrow$-expression. In the original $\updownarrow$-expression $E = \langle \updownarrow E_1 \rangle$, we first apply $\uparrow$ (the outermost operator of E_1) and then $\updownarrow$ (the outermost operator of E). That is, we first combine the arguments of E_1 into one molecule, and then complement the result. The idea behind the result E' of Make$\updownarrow$ExprMinimal is just the reverse: we first complement the arguments of E_1, and then combine the results into one molecule. Of course, in the first step, we do not have to complement $\updownarrow$-arguments of E_1, as these are complemented already. By Property ($\mathcal{D}_{\text{Min}}$.2), the minimal $\uparrow$-expression E_1 does not have $\uparrow$-arguments. Hence, in lines M$\updownarrow$M.4–M$\updownarrow$M.6 and lines M$\updownarrow$M.7–M$\updownarrow$M.18, we only complement the $\downarrow$-arguments and the $\mathcal{N}$-word-arguments.

In line 17 of MakeMinimal, we have a minimal $\downarrow$-expression E_i which is not alternating. As mentioned, E_i contains nicks, and we need a minimal, nick free DNA expression E'_i satisfying $E'_i \equiv_{\triangledown} E_i$. For this, we use a procedure Denickify, which we describe in Figure 4.

First, in lines Dni.4–Dni.19, we make the DNA expression alternating and thus, by Lemma 11, nick free. We do this by combining pairs of consecutive expression-arguments into single expression-arguments. Once the DNA expression is alternating, we may have to perform a simple post-processing step to make it minimal again. Line Dni.24 will be implemented by the same procedure RotateToMinimal that we use for lines 20 and 33 of MakeMinimal. Each substitution in procedure Denickify can be achieved by a few insertions and removals of brackets and operators in the DNA expression.

The only instructions left to be worked out are the ones in lines 20 and 33 of MakeMinimal, and the one in line Dni.24 of Denickify. Because in general, rewriting $\uparrow$-expressions is analogous to rewriting $\downarrow$-expressions, these instructions are similar. A single procedure, called RotateToMinimal,[3] suffices for their implementation. In Figure 5, we give this procedure for $\downarrow$-expressions.

We illustrate the different steps in MakeMinimal by means of an example DNA expression.

[3]The structure of a DNA expression can be visualized by a tree. The name RotateToMinimal refers to the effect on the tree that corresponds to the effect of the procedure on the DNA expression: one or two tree rotations (see, e.g., Section 14.2 in [5]).

```
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 some $m \geq 1$
                          // 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 $\langle \updownarrow \alpha_{1,1} \rangle$;                    ($\mathcal{D}_{\mathtt{Min}}$.3)
M$\updownarrow$M.21.    fi
M$\updownarrow$M.22.    $E' = \widehat{E}_1$;
M$\updownarrow$M.23.  }
```

Figure 3. Pseudo-code of the procedure `Make`$\updownarrow$`ExprMinimal`.

Example 12. Let

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

which denotes the formal DNA molecule from (9). The $\updownarrow$-subexpression $E^s = \langle \updownarrow \langle \updownarrow \alpha_3 \rangle \rangle$ has as its argument the (minimal) $\updownarrow$-expression $E_1 = \langle \updownarrow \alpha_3 \rangle$, and thus violates Property ($\mathcal{D}_{\text{Min}}$.1). Indeed, applying the same operator $\updownarrow$ to the same argument for a second time, does not change the result. According to line 7, E^s is substituted by E_1, yielding

$$\begin{aligned} E = {} & \langle \downarrow \langle \uparrow \alpha_1 \langle \updownarrow \langle \uparrow \alpha_2 \langle \updownarrow \alpha_3 \rangle \rangle \rangle \rangle \\ & \quad \langle \uparrow \langle \downarrow \alpha_4 \langle \uparrow \langle \updownarrow \alpha_5 \rangle \, \alpha_6 \langle \downarrow \langle \updownarrow \alpha_7 \rangle \langle \updownarrow \alpha_8 \rangle \rangle \rangle \rangle \, \alpha_9 \langle \uparrow \langle \downarrow \langle \updownarrow \alpha_{10} \rangle \, \alpha_{11} \langle \updownarrow \alpha_{12} \rangle \rangle \rangle \rangle \; \rangle . \end{aligned}$$

```
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 $E'_i \equiv_\triangledown E_i$;
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 $\widehat{\varepsilon}_j$ and $\widehat{\varepsilon}_{j+1}$ of $\widehat{E}_i$;
Dni.6.          if ($\widehat{\varepsilon}_j$ is an $\uparrow$-expression $\langle\uparrow \ldots \langle\updownarrow \alpha_{j,m_j}\rangle\rangle$)
Dni.7.          then if ($\widehat{\varepsilon}_{j+1}$ is an $\uparrow$-expression $\langle\uparrow \langle\updownarrow \alpha_{j+1,1}\rangle \ldots\rangle$)
Dni.8.               then substitute $\widehat{\varepsilon}_j\widehat{\varepsilon}_{j+1}$ in $\widehat{E}_i$
                         by $\langle\uparrow \ldots \langle\updownarrow \alpha_{j,m_j}\alpha_{j+1,1}\rangle \ldots\rangle$;
Dni.9.               else    // $\widehat{\varepsilon}_{j+1}$ is an $\updownarrow$-expression $\langle\updownarrow \alpha_{j+1,1}\rangle$
Dni.10.                   substitute $\widehat{\varepsilon}_j\widehat{\varepsilon}_{j+1}$ in $\widehat{E}_i$
                         by $\langle\uparrow \ldots \langle\updownarrow \alpha_{j,m_j}\alpha_{j+1,1}\rangle\rangle$;
Dni.11.              fi
Dni.12.         else    // $\widehat{\varepsilon}_j$ is an $\updownarrow$-expression $\langle\updownarrow \alpha_{j,1}\rangle$
Dni.13.              if ($\widehat{\varepsilon}_{j+1}$ is an $\uparrow$-expression $\langle\uparrow \langle\updownarrow \alpha_{j+1,1}\rangle \ldots\rangle$)
Dni.14.              then substitute $\widehat{\varepsilon}_j\widehat{\varepsilon}_{j+1}$ in $\widehat{E}_i$
                         by $\langle\uparrow \langle\updownarrow \alpha_{j,1}\alpha_{j+1,1}\rangle \ldots\rangle$;
Dni.15.              else    // $\widehat{\varepsilon}_{j+1}$ is an $\updownarrow$-expression $\langle\updownarrow \alpha_{j+1,1}\rangle$
Dni.16.                   substitute $\widehat{\varepsilon}_j\widehat{\varepsilon}_{j+1}$ in $\widehat{E}_i$
                         by $\langle\updownarrow \alpha_{j,1}\alpha_{j+1,1}\rangle$;
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 $\widehat{E}'_i \equiv \widehat{E}_i$; (proc. RotateToMinimal)      ($\mathcal{D}_{\mathtt{Min}}$.6)
Dni.25.         fi
Dni.26.    fi
Dni.27.    $E'_i = \widehat{E}_i$;
Dni.28.  }
```

Figure 4. Pseudo-code of the procedure `Denickify`.

The $\updownarrow$-subexpression $E^s = \langle\updownarrow \langle\uparrow \alpha_2 \langle\updownarrow \alpha_3\rangle\rangle\rangle$ has as its argument the (minimal) $\uparrow$-expression $E_1 = \langle\uparrow \alpha_2 \langle\updownarrow \alpha_3\rangle\rangle$, and thus also violates Property ($\mathcal{D}_{\text{Min}}$.1). According to line 9, E^s is substituted by the result of procedure `Make`$\updownarrow$`ExprMinimal`.

$\widehat{E}_1 = E_1$ does not have $\downarrow$-arguments, but it has a single $\mathcal{N}$-word-argument α_2. According to line M$\updownarrow$M.14, the arguments $\alpha_2 \langle\updownarrow \alpha_3\rangle$ are substituted in $\widehat{E}_1$ by $\langle\updownarrow \alpha_2\alpha_3\rangle$. As a result, the $\uparrow$-expresion $\widehat{E}_1$

```
RtM.1.  RotateToMinimal ($E$)
          // locally rewrites an alternating $\downarrow$-expression $E = \langle\downarrow \varepsilon_1 \dots \varepsilon_n\rangle$
          // with Properties ($\mathcal{D}_{\text{Min}}$.1)-($\mathcal{D}_{\text{Min}}$.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 $E' \equiv E$
RtM.2.  {
RtM.3.    if ($\varepsilon_1$ is an $\uparrow$-expression $\langle\uparrow \varepsilon_{1,1} \dots \varepsilon_{1,m_1-1}\varepsilon_{1,m_1}\rangle$)
RtM.4.    then if ($\varepsilon_n$ is an $\uparrow$-expression $\langle\uparrow \varepsilon_{n,1}\varepsilon_{n,2} \dots \varepsilon_{n,m_n}\rangle$)
RtM.5.         then $E' = \langle\uparrow \varepsilon_{1,1} \dots \varepsilon_{1,m_1-1} \langle\downarrow \varepsilon_{1,m_1}\varepsilon_2 \dots \varepsilon_{n-1}\varepsilon_{n,1}\rangle \varepsilon_{n,2} \dots \varepsilon_{n,m_n}\rangle$;
RtM.6.         else $E' = \langle\uparrow \varepsilon_{1,1} \dots \varepsilon_{1,m_1-1} \langle\downarrow \varepsilon_{1,m_1}\varepsilon_2 \dots \varepsilon_{n-1}\varepsilon_n\rangle\rangle$;
RtM.7.         fi
RtM.8.    else  // $\varepsilon_n$ must be an $\uparrow$-expression $\langle\uparrow \varepsilon_{n,1}\varepsilon_{n,2} \dots \varepsilon_{n,m_n}\rangle$
RtM.9.         $E' = \langle\uparrow \langle\downarrow \varepsilon_1\varepsilon_2 \dots \varepsilon_{n-1}\varepsilon_{n,1}\rangle \varepsilon_{n,2} \dots \varepsilon_{n,m_n}\rangle$;
RtM.10.   fi
RtM.11. }
```

Figure 5. Pseudo-code of the procedure RotateToMinimal.

has $m = 1$ argument left, which is an $\updownarrow$-argument. It thus violates Property ($\mathcal{D}_{\text{Min}}$.3). According to line M$\updownarrow$M.20, $\widehat{E}_1$ is substituted by this argument, yielding $E' = \widehat{E}_1 = \langle\updownarrow \alpha_2\alpha_3\rangle$. Indeed, E' is a minimal DNA expression satisfying $E' \equiv E^s$. This yields

$$E = \langle\downarrow \langle\uparrow \alpha_1 \langle\updownarrow \alpha_2\alpha_3\rangle\rangle \\ \langle\uparrow \langle\downarrow \alpha_4 \langle\uparrow \langle\updownarrow \alpha_5\rangle \alpha_6 \langle\downarrow \langle\updownarrow \alpha_7\rangle \langle\updownarrow \alpha_8\rangle\rangle\rangle\rangle \alpha_9 \langle\uparrow \langle\downarrow \langle\updownarrow \alpha_{10}\rangle \alpha_{11} \langle\updownarrow \alpha_{12}\rangle\rangle\rangle\rangle\ \rangle .$$

The third argument of the $\uparrow$-subexpression $E^s = \langle\uparrow \langle\updownarrow \alpha_5\rangle \alpha_6 \langle\downarrow \langle\updownarrow \alpha_7\rangle \langle\updownarrow \alpha_8\rangle\rangle\rangle$ is the (minimal) $\downarrow$-expression $E_3 = \langle\downarrow \langle\updownarrow \alpha_7\rangle \langle\updownarrow \alpha_8\rangle\rangle$. The occurrence of the operator $\downarrow$ in E_3 is an inner occurrence in E^s. Because the operator is not alternating, it violates Property ($\mathcal{D}_{\text{Min}}$.4).

E_3 denotes $\binom{\alpha_7}{c(\alpha_7)}^{\triangledown}\binom{\alpha_8}{c(\alpha_8)}$, and thus is not nick free. The occurring upper nick letter is removed by the outermost operator $\uparrow$ of E^s. According to line 17 of MakeMinimal, E_3 is substituted in E^s by the result of procedure Denickify.

$\widehat{E}_3 = E_3$ has one pair of consecutive expression-arguments $\langle\updownarrow \alpha_7\rangle$ and $\langle\updownarrow \alpha_8\rangle$. According to line Dni.16, they are substituted in $\widehat{E}_3$ by $\langle\updownarrow \alpha_7\alpha_8\rangle$. As a result, the $\downarrow$-expression $\widehat{E}_3$ has one argument left, which is an $\updownarrow$-argument. It is (trivially) alternating, but violates Property ($\mathcal{D}_{\text{Min}}$.3). According to line Dni.21, $\widehat{E}_3$ is substituted by its argument, yielding $E_3' = \widehat{E}_3 = \langle\updownarrow \alpha_7\alpha_8\rangle$. Indeed, this is a minimal, nick free DNA expression satisfying $E_3' \equiv_\triangledown E_3$. This yields

$$E = \langle\downarrow \langle\uparrow \alpha_1 \langle\updownarrow \alpha_2\alpha_3\rangle\rangle \langle\uparrow \langle\downarrow \alpha_4 \langle\uparrow \langle\updownarrow \alpha_5\rangle \alpha_6 \langle\updownarrow \alpha_7\alpha_8\rangle\rangle\rangle \alpha_9 \langle\uparrow \langle\downarrow \langle\updownarrow \alpha_{10}\rangle \alpha_{11} \langle\updownarrow \alpha_{12}\rangle\rangle\rangle\rangle\rangle .$$

The first argument of the $\uparrow$-subexpression

$$E^s = \langle\uparrow \langle\downarrow \alpha_4 \langle\uparrow \langle\updownarrow \alpha_5\rangle \alpha_6 \langle\updownarrow \alpha_7\alpha_8\rangle\rangle\rangle \alpha_9 \langle\uparrow \langle\downarrow \langle\updownarrow \alpha_{10}\rangle \alpha_{11} \langle\updownarrow \alpha_{12}\rangle\rangle\rangle\rangle$$

is the (minimal) alternating $\downarrow$-expression $E_1 = \langle\downarrow \alpha_4 \langle\uparrow \langle\updownarrow \alpha_5\rangle \alpha_6 \langle\updownarrow \alpha_7\alpha_8\rangle\rangle\rangle$, whose last argument is an $\uparrow$-argument. The occurrence of the operator $\downarrow$ in E_1 is an inner occurrence in E^s, and thus violates Property ($\mathcal{D}_{\text{Min}}$.5). According to line 20 of function MakeMinimal and line RtM.9 of procedure

RotateToMinimal, E_1 is substituted in E^s by the (minimal) $\uparrow$-expression $E_1' = \langle\uparrow \langle\downarrow \alpha_4 \langle\updownarrow \alpha_5\rangle\rangle \alpha_6$ $\langle\updownarrow \alpha_7\alpha_8\rangle\rangle$, yielding

$$E = \langle\downarrow \langle\uparrow \alpha_1 \langle\updownarrow \alpha_2\alpha_3\rangle\rangle \langle\uparrow \langle\uparrow \langle\downarrow \alpha_4 \langle\updownarrow \alpha_5\rangle\rangle \alpha_6 \langle\updownarrow \alpha_7\alpha_8\rangle\rangle \alpha_9 \langle\uparrow \langle\downarrow \langle\updownarrow \alpha_{10}\rangle \alpha_{11} \langle\updownarrow \alpha_{12}\rangle\rangle\rangle\rangle\rangle\,.$$

Note that the first argument $\langle\downarrow \alpha_4 \langle\updownarrow \alpha_5\rangle\rangle$ of the $\uparrow$-expression E_1' is a $\downarrow$-expression. Hence, the occurrence of $\uparrow$ in E_1', which is an inner occurrence in the resulting DNA expression E, also violates Property ($\mathcal{D}_{\text{Min}}$.5). It is not hard to prove that this is the case in general, after application of instruction 20 of MakeMinimal. This is, however, resolved in the next step.

The first argument of the $\uparrow$-subexpression

$$E^s = \langle\uparrow \langle\uparrow \langle\downarrow \alpha_4 \langle\updownarrow \alpha_5\rangle\rangle \alpha_6 \langle\updownarrow \alpha_7\alpha_8\rangle\rangle \alpha_9 \langle\uparrow \langle\downarrow \langle\updownarrow \alpha_{10}\rangle \alpha_{11} \langle\updownarrow \alpha_{12}\rangle\rangle\rangle\rangle$$

is the (minimal) $\uparrow$-expression $E_1 = \langle\uparrow \langle\downarrow \alpha_4 \langle\updownarrow \alpha_5\rangle\rangle \alpha_6 \langle\updownarrow \alpha_7\alpha_8\rangle\rangle$. Hence, E^s violates Property ($\mathcal{D}_{\text{Min}}$.2). According to line 23, E_1 is substituted in E^s by its arguments, yielding

$$E = \langle\downarrow \langle\uparrow \alpha_1 \langle\updownarrow \alpha_2\alpha_3\rangle\rangle \langle\uparrow \langle\downarrow \alpha_4 \langle\updownarrow \alpha_5\rangle\rangle \alpha_6 \langle\updownarrow \alpha_7\alpha_8\rangle \alpha_9 \langle\uparrow \langle\downarrow \langle\updownarrow \alpha_{10}\rangle \alpha_{11} \langle\updownarrow \alpha_{12}\rangle\rangle\rangle\rangle\rangle\,.$$

The $\uparrow$-subexpression

$$E^s = \langle\uparrow \langle\downarrow \langle\updownarrow \alpha_{10}\rangle \alpha_{11} \langle\updownarrow \alpha_{12}\rangle\rangle\rangle$$

has exactly one argument, the (minimal) $\downarrow$-expression $E_1 = \langle\downarrow \langle\updownarrow \alpha_{10}\rangle \alpha_{11} \langle\updownarrow \alpha_{12}\rangle\rangle$. Hence, E^s violates Property ($\mathcal{D}_{\text{Min}}$.3). Because E_1 is alternating and thus nick free, the outermost operator $\uparrow$ of E^s does not have any effect on the semantics, and $E^s \equiv E_1$. According to line 29, E^s is substituted by E_1, yielding

$$E = \langle\downarrow \langle\uparrow \alpha_1 \langle\updownarrow \alpha_2\alpha_3\rangle\rangle \langle\uparrow \langle\downarrow \alpha_4 \langle\updownarrow \alpha_5\rangle\rangle \alpha_6 \langle\updownarrow \alpha_7\alpha_8\rangle \alpha_9 \langle\downarrow \langle\updownarrow \alpha_{10}\rangle \alpha_{11} \langle\updownarrow \alpha_{12}\rangle\rangle\rangle\rangle\,.$$

The arguments of the $\uparrow$-subexpression

$$E^s = \langle\uparrow \langle\downarrow \alpha_4 \langle\updownarrow \alpha_5\rangle\rangle \alpha_6 \langle\updownarrow \alpha_7\alpha_8\rangle \alpha_9 \langle\downarrow \langle\updownarrow \alpha_{10}\rangle \alpha_{11} \langle\updownarrow \alpha_{12}\rangle\rangle\rangle$$

are $\mathcal{N}$-word-arguments and (minimal) expression-arguments, alternately, and both the first argument and the last argument are $\downarrow$-arguments. Consequently, E^s is not a minimal DNA subexpression, because its outermost operator $\uparrow$ violates Property ($\mathcal{D}_{\text{Min}}$.6). Moreover, this occurrence of $\uparrow$ is an inner occurrence in E. Therefore, in the context of E, it violates Property ($\mathcal{D}_{\text{Min}}$.5). According to line 33 of function MakeMinimal and line RtM.5 of (the analogue for $\uparrow$-expressions of) procedure RotateToMinimal, E^s is substituted by the (minimal) $\downarrow$-expression

$$E^{s\prime} = \langle\downarrow \alpha_4 \langle\uparrow \langle\updownarrow \alpha_5\rangle \alpha_6 \langle\updownarrow \alpha_7\alpha_8\rangle \alpha_9 \langle\updownarrow \alpha_{10}\rangle\rangle \alpha_{11} \langle\updownarrow \alpha_{12}\rangle\rangle\,,$$

yielding

$$E = \langle\downarrow \langle\uparrow \alpha_1 \langle\updownarrow \alpha_2\alpha_3\rangle\rangle \langle\downarrow \alpha_4 \langle\uparrow \langle\updownarrow \alpha_5\rangle \alpha_6 \langle\updownarrow \alpha_7\alpha_8\rangle \alpha_9 \langle\updownarrow \alpha_{10}\rangle\rangle \alpha_{11} \langle\updownarrow \alpha_{12}\rangle\rangle\rangle\,.$$

The second argument of the $\downarrow$-expression E is the (minimal) $\downarrow$-expression

$$E_2 = \langle\downarrow \alpha_4 \langle\uparrow \langle\updownarrow \alpha_5\rangle \alpha_6 \langle\updownarrow \alpha_7\alpha_8\rangle \alpha_9 \langle\updownarrow \alpha_{10}\rangle\rangle \alpha_{11} \langle\updownarrow \alpha_{12}\rangle\rangle\,.$$

Hence, E violates Property ($\mathcal{D}_{\text{Min}}$.2), and according to (the analogue for $\downarrow$-expressions of) line 23, E_2 is substituted by its arguments, yielding

$$E = \langle\downarrow \langle\uparrow \alpha_1 \langle\updownarrow \alpha_2\alpha_3\rangle\rangle \alpha_4 \langle\uparrow \langle\updownarrow \alpha_5\rangle \alpha_6 \langle\updownarrow \alpha_7\alpha_8\rangle \alpha_9 \langle\updownarrow \alpha_{10}\rangle\rangle \alpha_{11} \langle\updownarrow \alpha_{12}\rangle\rangle .$$

This DNA expression has all six properties from Theorem 10, and thus is minimal. It is the final result of the recursive function `MakeMinimal`, when applied to the original DNA expression in (17) at the start of this example. It equals DNA expression E_c from (15), which we obtained from the *semantics* of the DNA expression, using Theorem 7. ■

Note that for many of the substitutions we performed in the above example, the original DNA subexpression resembled one of the DNA expressions in the second column of Table 1, in a row for the property involved. In those cases, the result indeed resembled the corresponding minimal DNA expression in the fourth column.

5. Correctness and complexity of the algorithm

For the DNA expression from (17), the recursive function `MakeMinimal` correctly produces an equivalent, minimal DNA expression. In this section, we establish the correctness of `MakeMinimal` for *arbitrary* DNA expressions. We subsequently analyse its complexity. We first consider the correct performance of the three procedures used by the function.

Lemma 13. Let $E = \langle\updownarrow E_1\rangle$ be an $\updownarrow$-expression whose argument E_1 is a minimal $\uparrow$-expression.

Then the string E' resulting from procedure `Make`$\updownarrow$`ExprMinimal` is a minimal DNA expression satisfying $E' \equiv E$. Moreover, E' is independent of the orders in which $\downarrow$-arguments and $\mathcal{N}$-word-arguments are considered in lines M$\updownarrow$M.4 and M$\updownarrow$M.7, respectively.

Proof: We establish that throughout the first for-loop in `Make`$\updownarrow$`ExprMinimal`, $\widehat{E}_1$ remains a minimal $\uparrow$-expression satisfying $\left\langle\updownarrow \widehat{E}_1\right\rangle \equiv E$. First, we observe that $\widehat{E}_1$ remains a DNA expression (and thus, an $\uparrow$-expression), when we substitute a $\downarrow$-argument $E_{1,i}$ by $\langle\updownarrow \alpha_{E_{1,i}}\rangle$: an $\updownarrow$-argument fits together by upper strands with whatever argument preceding or succeeding it. Second, we note that before the substitution of $E_{1,i}$, $\widehat{E}_1$ has all properties from Theorem 10. It is easy to verify that it still has these properties after the substitution, and thus is minimal.

Finally, we check that the semantics of $\left\langle\updownarrow \widehat{E}_1\right\rangle$ is not changed by the substitution of $E_{1,i}$. This check consists of two parts: (1) We check that the (lower) nick letters in $\mathcal{S}(\left\langle\updownarrow \widehat{E}_1\right\rangle)$ (which are introduced by the outermost operator of $\widehat{E}_1$) are the same before and after the substitution. (2) As for the $\mathcal{A}$-letters in $\mathcal{S}(\left\langle\updownarrow \widehat{E}_1\right\rangle)$, we observe, that it does not matter for the semantics, when exactly we complement $\mathcal{N}$-words occurring in $E_{1,i}$. That is, whether we do this with the outermost operator $\updownarrow$ of $\left\langle\updownarrow \widehat{E}_1\right\rangle$ (as we do before the substitution), or that we do this with the operator $\updownarrow$ in $\langle\updownarrow \alpha_{E_{1,i}}\rangle$ (as we do after the substitution).

We subsequently consider the second for-loop in `Make`$\updownarrow$`ExprMinimal`. We establish that throughout this loop $\widehat{E}_1$ remains an $\uparrow$-expression whose only arguments are $\mathcal{N}$-words $\alpha_{1,i}$ and $\updownarrow$-expressions $\langle\updownarrow \alpha_{1,i}\rangle$ for $\mathcal{N}$-words $\alpha_{1,i}$, and which still satisfies $\left\langle\updownarrow \widehat{E}_1\right\rangle \equiv E$. The validity of the final equivalence follows from the observation that the operator $\uparrow$ never introduces nick letters before or after an $\mathcal{N}$-word-argument. This holds in particular for the outermost operator $\uparrow$ of $\widehat{E}_1$ before the substitution of an $\mathcal{N}$-word-argument.

After the second for-loop, $\widehat{E}_1 = \langle\uparrow \langle\updownarrow \alpha_{1,1}\rangle \dots \langle\updownarrow \alpha_{1,m}\rangle\rangle$ for some $m \geq 1$ and $\mathcal{N}$-words $\alpha_{1,1}, \dots, \alpha_{1,m}$. In $\mathcal{S}(\widehat{E}_1)$, there are no single strands left to be complemented. Hence, $\widehat{E}_1 \equiv \left\langle \updownarrow \widehat{E}_1 \right\rangle \equiv E$. It is easily verified that if $m \geq 2$, then $\widehat{E}_1$ has all properties from Theorem 10 and thus is minimal. If, on the other hand, $m = 1$, then the outermost operator $\uparrow$ of $\widehat{E}_1$ has no effect. Hence $\widehat{E}_1 \equiv \langle\updownarrow \alpha_{1,1}\rangle$, which, by Theorem 3, is minimal.

It follows from Example 9 and Theorem 3, that in both cases, the result E' is the unique minimal DNA expression denoting $\mathcal{S}(E)$. In particular, it is independent of the orders in which $\downarrow$-arguments and $\mathcal{N}$-word-arguments are considered in the two for-loops. □

Lemma 14. Let E_i be a minimal $\downarrow$-expression which is not alternating.

Then the string E_i' resulting from procedure `Denickify` is a minimal, nick free DNA expression satisfying $E_i' \equiv_\triangledown E_i$. Moreover, E_i' is independent of the order in which pairs of consecutive expression-arguments $\widehat{\varepsilon}_j$ and $\widehat{\varepsilon}_{j+1}$ are selected in line Dni.5.

Proof: We establish that throughout the while-loop, $\widehat{E}_i$ is a $\downarrow$-expression satisfying $\widehat{E}_i \equiv_\triangledown E_i$, $\widehat{E}_i$ has at least one expression-argument, has Properties ($\mathcal{D}_{\text{Min}}$.1), ($\mathcal{D}_{\text{Min}}$.2), ($\mathcal{D}_{\text{Min}}$.4) and ($\mathcal{D}_{\text{Min}}$.5), and each inner occurrence of $\uparrow$ or $\downarrow$ has at least two arguments. By Lemma 11, $\widehat{E}_i$ is nick free, if and only if it is alternating. Before every iteration of the loop, $\widehat{E}_i$ has at least two consecutive expression-arguments. This implies that $\widehat{E}_i$ is not nick free. Moreover, at that moment it also has Properties ($\mathcal{D}_{\text{Min}}$.3) and ($\mathcal{D}_{\text{Min}}$.6), and thus is minimal.

We verify that the expression-arguments $\widehat{\varepsilon}_j$ and $\widehat{\varepsilon}_{j+1}$ selected in line Dni.5 are $\updownarrow$-expressions or $\uparrow$-expressions of the forms considered in the if-then-else construction in lines Dni.6–Dni.18 (cf. the proof of Lemma 11). The left (or right) end of the expression-argument that we substitute for $\widehat{\varepsilon}_j\widehat{\varepsilon}_{j+1}$ is equal to the left end of $\widehat{\varepsilon}_j$ (the right end of $\widehat{\varepsilon}_{j+1}$). Hence, after the substitution, the arguments of $\widehat{E}_i$ still fit together by lower strands, and hence, $\widehat{E}_i$ is still a DNA expression. It follows from the definition of a $\downarrow$-expression that the only difference in the semantics of $\widehat{E}_i$, is that before the substitution, there was an upper nick letter between $\widehat{\varepsilon}_j$ and $\widehat{\varepsilon}_{j+1}$, which is no longer present after the substitution. In particular, $\widehat{E}_i \equiv_\triangledown E_i$ remains valid.

After the while loop, $\widehat{E}_i$ is alternating, but does not necessarily have Properties ($\mathcal{D}_{\text{Min}}$.3) and ($\mathcal{D}_{\text{Min}}$.6), anymore. This is made up for in the if-then-else construction following the loop. If $\widehat{E}_i$ has only one argument left, then this must be a (minimal, nick free) DNA expression $E_{i,1}$ and the outermost operator $\downarrow$ has no effect. Hence, $\widehat{E}_i \equiv E_{i,1}$. Otherwise, $\widehat{E}_i$ has at least Properties ($\mathcal{D}_{\text{Min}}$.1)–($\mathcal{D}_{\text{Min}}$.5). If necessary, we use procedure `RotateToMinimal` to also acquire Property ($\mathcal{D}_{\text{Min}}$.6).

In order to establish that the resulting DNA expression E_i' is independent of the order in which we select consecutive expression-arguments, we first observe that the expression-arguments in the original $\downarrow$-expression E_i are split up over one or more subsequences of consecutive expression-arguments, which are separated by $\mathcal{N}$-word-arguments. In $\widehat{E}_i$ after the while-loop, each of this subsequences has been substituted by a single expression-argument, and the result for one subsequence is independent of the result for another. We therefore examine what happens to a single subsequence of expression-arguments

$\varepsilon_{j_0} \dots \varepsilon_{j_1}$ with $j_0 < j_1$. Let us use ε'_{j_1} to denote the corresponding single expression-argument of $\widehat{E}_i$ after the while-loop.

If each ε_j in the subsequence is an $\updownarrow$-expression, then it is not hard to see that ε'_{j_1} is also an $\updownarrow$-expression, which is independent of the order in which pairs of consecutive expression-arguments from $\varepsilon_{j_0} \dots \varepsilon_{j_1}$ have been selected.

If, on the other hand, at least one of the ε_j's is an $\uparrow$-expression, then we can prove by induction that ε'_{j_1} is a minimal, nick free $\uparrow$-expression, which satisfies $\varepsilon'_{j_1} \equiv_\triangledown \langle\downarrow \varepsilon_{j_0} \dots \varepsilon_{j_1}\rangle$, and whose $\downarrow$-arguments are exactly the $\downarrow$-arguments of $\varepsilon_{j_0}, \dots, \varepsilon_{j_1}$. In fact, these $\downarrow$-arguments determine a lower block partitioning of $\mathcal{S}(\varepsilon'_{j_1})$. Now, ε'_{j_1} is completely determined by the construction from Theorem 7. In particular, it does not depend on the order in which pairs of arguments from $\varepsilon_{j_0} \dots \varepsilon_{j_1}$ have been selected. Then the same goes for the complete $\downarrow$-expression $\widehat{E}_i$ after the while-loop, and for the final result E'_i. □

Lemma 15. Let E be an alternating $\downarrow$-expression with Properties ($\mathcal{D}_{\text{Min}}$.1)–($\mathcal{D}_{\text{Min}}$.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: We first use the definition of (the semantics of) a DNA expression to prove that the string E' is indeed a DNA expression, and that $\mathcal{S}(E')$ and $\mathcal{S}(E)$ are the same, possibly apart from nick letters. That is, $\mathcal{S}(E')$ may contain nick letters not occurring in $\mathcal{S}(E)$ and vice versa.

We subsequently establish that E' is alternating and has Properties ($\mathcal{D}_{\text{Min}}$.1)–($\mathcal{D}_{\text{Min}}$.5), just like E. In particular, because E and E' are alternating and have Property ($\mathcal{D}_{\text{Min}}$.4), both of them are nick free. This implies that $\mathcal{S}(E')$ really equals $\mathcal{S}(E)$.

We finally observe that E' also has Property ($\mathcal{D}_{\text{Min}}$.6). Hence, E' has all properties from Theorem 10, and thus is minimal. □

We now get to the recursive function `MakeMinimal` itself.

Theorem 16. For each DNA expression E_1^*, the function `MakeMinimal` produces an equivalent, minimal DNA expression E_2^*. [4]

Proof: We use induction by the number of operators occurring in E_1^*. If E_1^* contains only one operator, then either $E_1^* = \langle\updownarrow \alpha_1\rangle$, or $E_1^* = \langle\uparrow \alpha_1\rangle$, or $E_1^* = \langle\downarrow \alpha_1\rangle$ for an $\mathcal{N}$-word α_1. It is easily verified that in this case, E_1^* is minimal itself, and that $E_2^* = E_1^*$. Given that `MakeMinimal` produces an equivalent, minimal DNA expression for DNA expressions containing at most p operators, we must prove that it also does so if E_1^* contains $p + 1$ operators. For this, we examine the effects of the various substitutions that are carried out in `MakeMinimal`.

In lines 5–11, we have an $\updownarrow$-expression E with an expression-argument E_1. By the induction hypothesis, the recursive call in line 5 makes E_1 minimal without changing the semantics.

In line 7, E_1 is a (minimal) $\updownarrow$-expression. Substituting E by E_1, i.e., skipping the outermost operator $\updownarrow$ of E, does not effect the semantics, and leaves a minimal DNA expression.

[4]The reason for using the notation E_1^* and E_2^* to denote the input and the output of `MakeMinimal`, is that we want to clearly distinguish these DNA expressions from the 'running DNA expression' E and the expression-arguments E_i occurring in the function.

In line 9, E_1 is a (minimal) $\uparrow$-expression or $\downarrow$-expression. Indeed, procedure `Make`$\updownarrow$`ExprMinimal` (or its analogue for a $\downarrow$-expression E_1) is applicable to E, and yields an equivalent, minimal DNA expression E'.

In lines 13–35, we have an $\uparrow$-expression E. In the for-loop, we consider the expression-arguments E_i and perform at most four actions on them. We now establish that throughout the loop, E remains an $\uparrow$-expression that is equivalent to the original DNA expression E_1^*.

By the induction hypothesis, the recursive call in line 15 makes E_i minimal, without changing its semantics.

Now, if E_i is a (minimal) $\downarrow$-expression which is not alternating, then by (the analogue for $\downarrow$-expressions of) Lemma 11, E_i is not nick free. In fact, $\mathcal{S}(E_i)$ contains upper nick letters. In the context of E, these upper nick letters are removed by the outermost operator $\uparrow$ of E. Hence, $\mathcal{S}(E)$ is not affected by line 17, where we substitute E_i by a minimal, nick free DNA expression E_i' satisfying $E_i' \equiv_{\triangledown} E_i$. Indeed, procedure `Denickify` yields such a DNA expression. The new argument E_i' may be any type of expression-argument. However, if it is an $\uparrow$-argument or a $\downarrow$-expression, then it is alternating. As a result, after lines 16–18, E_i is still minimal and if E_i is a $\downarrow$-expression, then it is certainly alternating.

If, in line 19, E_i is a (minimal, alternating) $\downarrow$-expression for which the first argument or the last argument is an $\uparrow$-argument, then procedure `RotateToMinimal` is applicable and indeed yields a minimal $\uparrow$-expression E_i' satisfying $E_i' \equiv E_i$. Hence, after lines 19–21, E_i is still minimal, and if E_i is a $\downarrow$-expression, then it is alternating and neither its first argument nor its last argument is an $\uparrow$-argument. In this case, by Properties ($\mathcal{D}_{\text{Min}}$.1) and ($\mathcal{D}_{\text{Min}}$.2) from Theorem 10,

- the first argument of E_i is an $\mathcal{N}$-word α or an $\updownarrow$-expression $\langle\updownarrow \alpha\rangle$, and
- the last argument of E_i is an $\mathcal{N}$-word α or an $\updownarrow$-expression $\langle\updownarrow \alpha\rangle$.

If (by now) E_i is an $\uparrow$-expression, then in line 23 we substitute E_i by its arguments. That is, we skip the outermost operator of E_i. This does not affect $\mathcal{S}(E)$, because the effect of the skipped operator is now achieved by the outermost operator $\uparrow$ of E. After this, however, we can no longer speak of an expression-argument E_i, but we can speak of *the argument(s) corresponding to* E_i. Because E_i was a minimal $\uparrow$-expression, by Theorem 10, the arguments corresponding to E_i are minimal, there is no $\uparrow$-expression among these arguments, and if any of these arguments is a $\downarrow$-expression, then it is alternating and

- its first argument is an $\mathcal{N}$-word α or an $\updownarrow$-expression $\langle\updownarrow \alpha\rangle$, and
- its last argument is an $\mathcal{N}$-word α or an $\updownarrow$-expression $\langle\updownarrow \alpha\rangle$.

Obviously, all of this also holds after lines 22–24, if E_i was not an $\uparrow$-expression. In that case the only argument corresponding to E_i is E_i itself.

As a result of all this, it is easily verified that after the for-loop, E is an $\uparrow$-expression satisfying $E \equiv E_1^*$, whose expression-arguments are minimal and nick free, and which has Properties ($\mathcal{D}_{\text{Min}}$.1), ($\mathcal{D}_{\text{Min}}$.2), ($\mathcal{D}_{\text{Min}}$.4) and ($\mathcal{D}_{\text{Min}}$.5).

In lines 28–30, E in addition has only one argument ε_1. If ε_1 is a (minimal, nick free) expression-argument E_1, then the outermost operator $\uparrow$ of E does not have any effect. Hence, substituting E by E_1, i.e., skipping this operator $\uparrow$, as we do in line 29, leaves an equivalent, minimal DNA expression. If, on the other hand, ε_1 is an $\mathcal{N}$-word α_1, then $E = \langle\uparrow \alpha_1\rangle$ is minimal already.

In lines 32–34, E has at least two arguments. We observe that none of the arguments of E can be of the form $\langle\downarrow \alpha\rangle$, because in that case, the arguments of E would not fit together by upper strands. As a result, E also has Property ($\mathcal{D}_{\text{Min}}$.3).

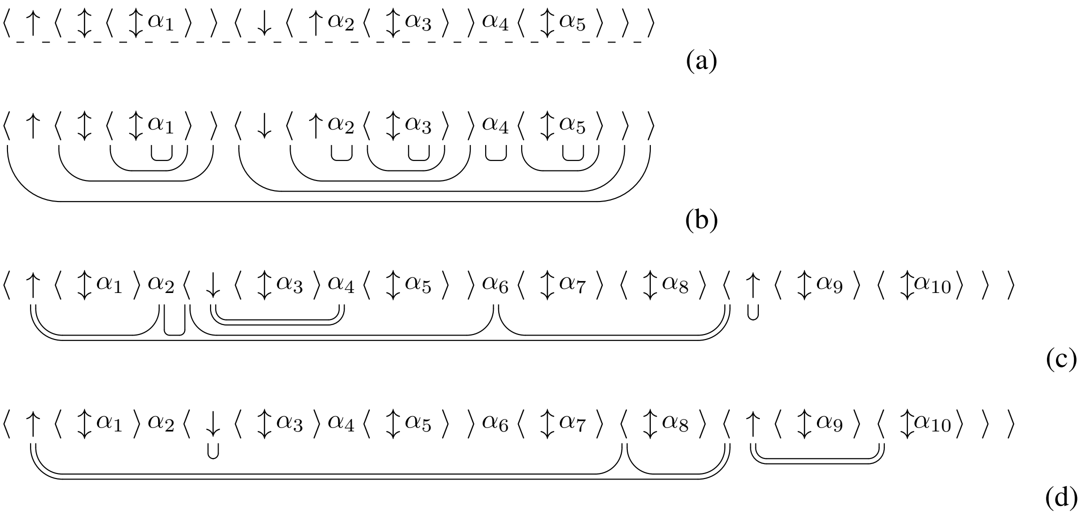


Figure 6. Data structure used in the implementation of the algorithm for minimality. (a) First feature for an example DNA expression: the letters are stored in a doubly-linked list (indicated by the dashes). (b) Second feature for the same DNA expression: corresponding brackets are connected, and the first letter and the last letter of each $\mathcal{N}$-word are connected. (c) Third feature for an example DNA expression: each occurrence of $\uparrow$ or $\downarrow$ has a circular, doubly-linked list of its non-$\updownarrow$-arguments. Note that the list is empty for the last occurrence of $\uparrow$. (d) Fourth feature for the same DNA expression: each occurrence of $\uparrow$ or $\downarrow$ has a circular, doubly-linked list of its consecutive expression-arguments. Note that the list is empty for the only occurrence of $\downarrow$.

If E is alternating and both its first argument and its last argument are $\downarrow$-arguments, then (the analogue for $\uparrow$-expressions of) procedure `RotateToMinimal` is applicable and the substitution in line 33 yields an equivalent, minimal DNA expression.

Otherwise, there are two possibilities: (1) E is not alternating, i.e., it has two consecutive expression-arguments. (2) E is alternating and either its first argument or its last argument (or both) is not a $\downarrow$-argument. In this case, the argument(s) concerned must be an $\mathcal{N}$-word α or an $\updownarrow$-expression $\langle\updownarrow\alpha\rangle$. We conclude that E also has Property ($\mathcal{D}_{\text{Min}}$.6). Hence, it has all properties from Theorem 10 and thus is minimal already. $\square$

With a proper data structure, the function `MakeMinimal` can be carried out efficiently. We propose a data structure with four 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. This feature is depicted for an example DNA expression in Figure 6(a).

It is also important that we can easily *approach* the positions in the DNA expression where operations like insertions and removals must be performed. In particular, it is useful if we can step directly from, for example, the first letter to the last letter of an argument, and vice versa. This is the second feature of the data structure: we connect each opening bracket to the corresponding closing bracket, and for each $\mathcal{N}$-word-argument of an operator, we connect the first letter to the last letter. We establish such connections both from left to right and from right to left. In Figure 6(b), we show these connections for an example DNA expression.

The third feature is useful to carry out procedure `Make`$\updownarrow$`ExprMinimal` efficiently. In this procedure, we wish to complement the non-$\updownarrow$-arguments of an $\uparrow$-expression (or a $\downarrow$-expression) E_1, i.e., the arguments that are not yet complemented. It is important that we can quickly find these arguments. Therefore, 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 these arguments. In Figure 6(c), we have depicted the lists for all occurrences of $\uparrow$ and $\downarrow$ in an example DNA expression.

The fourth feature of the data structure is meant to carry out procedure `Denickify` efficiently. In this procedure, we wish to combine pairs of consecutive expression-arguments into single expression-arguments. For this, we must be able to quickly find these pairs. Therefore, 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 succeeded by another expression-argument $\widehat{\varepsilon}_{j+1}$, the list contains the position of the first letter of $\widehat{\varepsilon}_{j+1}$ (which is an opening bracket). In Figure 6(d), we show the lists for all occurrences of $\uparrow$ and $\downarrow$ in an example DNA expression.

All connections can be initialized in linear time, e.g., in a recursive pass through the DNA expression. For any (basic) operation applied to E, the connections can be updated in constant time.[5] We conclude that the overhead for maintaining the four types of connections is linear in the time required for the function `MakeMinimal` itself. As an illustration, we examine the effects on the data structure for one of the substitutions in `MakeMinimal`:

Example 17. Let E be an $\uparrow$-expression which has a minimal $\uparrow$-argument E_i. In line 23 of `MakeMinimal`, we substitute E_i in E by its arguments. This implies that the brackets enclosing E_i and its operator $\uparrow$ are removed. We use the second feature of the data structure to find the closing bracket corresponding to the opening bracket of E_i. We use the first feature of the data structure to do the actual removals.

Because of the substitution, the non-$\updownarrow$-argument E_i must be removed from the list of non-$\updownarrow$-arguments of E (the third feature). On the other hand, the list of non-$\updownarrow$-arguments of E_i must be inserted into the list for E, as these non-$\updownarrow$-arguments become arguments of E. If E_i was preceded in E by an $\mathcal{N}$-word-argument α_{i-1} and the first argument of E_i itself was an $\mathcal{N}$-word-argument $\alpha_{i,1}$, then we may choose to combine the two $\mathcal{N}$-words into a single $\mathcal{N}$-word-argument $\alpha_{i-1}\alpha_{i,1}$ of E. This again affects the list of non-$\updownarrow$-arguments of E, but it also affects the connections between the first letter and the last letter of the (old and new) $\mathcal{N}$-word-arguments (the second feature).

If E_i was preceded in E by an expression-argument, then (the opening bracket of) E_i used to be in the list of consecutive expression-arguments of E (the fourth feature), and it must be removed from this list. On the other hand, the list of consecutive expression-arguments of E_i must be inserted into the list for E. Moreover, if E_i was preceded in E by an expression-argument E_{i-1} and the first argument of E_i itself was an expression-argument $E_{i,1}$, then $E_{i,1}$ must also be inserted into the list of consecutive expression-arguments of E. It does not matter where in this list the insertions are carried out.

In the discussion of the last two features, we zoomed in on the effects for the argument preceding E_i (if any) and the first argument of E_i. Of course, there are similar effects for the last argument of E_i and the argument succeeding E_i.

We conclude that the simple substitution in line 23 of `MakeMinimal` has many effects on our data structure. Nevertheless, these effects (together) can be accomplished in constant time. ■

The importance of the first two features of the data structure may be more obvious than that of the last two features. In [15], we give examples of DNA expressions for which `MakeMinimal` requires quadratic

[5]The substitution in line M$\updownarrow$M.5 of procedure `Make`$\updownarrow$`ExprMinimal` must be considered a composite operation. In the proof of Lemma 18, we explain how it can be implemented.

time, if it does not have lists of non-$\updownarrow$-arguments and lists of consecutive expression-arguments. In particular, in that case, the algorithm may spend quadratic time in the procedures `Make↕ExprMinimal` and `Denickify`. We now establish that with the complete data structure, we do better. Before we reach this conclusion for `MakeMinimal` itself, we consider these procedures.

Lemma 18. 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 at most linear in $|E_1^*|$.

Proof: We first analyse the time spent in a single call of procedure `Make↕ExprMinimal`, for $E = \langle\updownarrow E_1\rangle$, where E_1 is a minimal $\uparrow$-expression. If E_1 does not have non-$\updownarrow$-arguments, then this call requires constant time.

From now on, we assume that E_1 has at least one non-$\updownarrow$-argument. In this case, we spend most time in the two for-loops, where we complement the $\downarrow$-arguments and the $\mathcal{N}$-word-arguments of $\widehat{E}_1 = E_1$. We use the list of non-$\updownarrow$-arguments of (the outermost operator $\uparrow$ of) $\widehat{E}_1$ to iterate along these arguments.

For each $\downarrow$-argument $E_{1,i}$, we have to determine $\alpha_{E_{1,i}}$. We do this by traversing $E_{1,i}$ from left to right, and concatenating the $\mathcal{N}$-words we encounter. If the parent operator of an $\mathcal{N}$-word α is $\downarrow$, then, of course, we determine $c(\alpha)$ (which takes time that is linear in $|\alpha|$) and concatenate that. We can prove that at least one third of the $\mathcal{N}$-words we encounter have parent operator $\uparrow$ or $\downarrow$ in $E_{1,i}$. In other words, the total number of $\mathcal{N}$-words we encounter is at most three times the number of $\mathcal{N}$-words with parent operator $\uparrow$ or $\downarrow$. This implies that the time required to determine $\alpha_{E_{1,i}}$ (and thus $\langle\updownarrow \alpha_{E_{1,i}}\rangle$) is at most linear in the number of $\mathcal{N}$-letters with such parent operator in $E_{1,i}$.

For each $\mathcal{N}$-word-argument $\alpha_{1,i}$ of $\widehat{E}_1$, the time to combine this argument with preceding and succeeding $\updownarrow$-arguments is constant. In case $\widehat{E}_1$ is not an $\uparrow$-expression but a $\downarrow$-expression, we also have to determine $c(\alpha_{1,i})$ (which takes time that is linear in $|\alpha_{1,i}|$). In total, the time spent on $\alpha_{1,i}$ is at most linear in the number of $\mathcal{N}$-letters in $\alpha_{1,i}$. Obviously, the parent operator of these $\mathcal{N}$-letters is the outermost operator of $\widehat{E}_1$ (either $\uparrow$ or $\downarrow$).

We can conclude that the total time spent in a single call of `Make↕ExprMinimal` is at most linear in the number of $\mathcal{N}$-letters in E_1 that have parent operator $\uparrow$ or $\downarrow$. As a result of this call, these $\mathcal{N}$-letters get parent operator $\updownarrow$, and this does not change anymore. Hence, the total time spent in `Make↕ExprMinimal` (over all recursive calls of `MakeMinimal`) is at most linear in the number of $\mathcal{N}$-letters in E_1^* with parent operator $\uparrow$ or $\downarrow$, and thus at most linear in $|E_1^*|$. The final conclusion remains valid when we drop the assumption that the $\uparrow$-expression or $\downarrow$-expression E_1 has at least one non-$\updownarrow$-argument. $\square$

Lemma 19. Let E_1^* be an arbitrary DNA expression. The total time that the function `MakeMinimal` applied to E_1^* spends in procedure `Denickify` is at most linear in the number of $\mathcal{N}$-words occurring in E_1^*, and thus in $|E_1^*|$.

Proof: We first analyse the time spent in a single call of procedure `Denickify`, for a minimal $\downarrow$-expression E_i which is not alternating.

We spend most time in the while-loop. We use the list of the consecutive expression-arguments of $\widehat{E}_i = E_i$, to select $\widehat{\varepsilon}_j$ and $\widehat{\varepsilon}_{j+1}$. Consequently, a complete iteration of the loop requires constant time. As a result of the substitution in an iteration, two $\mathcal{N}$-words which were separate, form one $\mathcal{N}$-word. Hence, the number of (maximal) $\mathcal{N}$-words in $\widehat{E}_i$ decreases by 1.

Then the time spent in `Denickify` for E_i is linear in the number of iterations of the while-loop for E_i, which is in turn equal to the decrease of the number of (maximal) $\mathcal{N}$-words in E_i. This carries over to the time spent in `Denickify` over all recursive calls of `MakeMinimal` for E_1^*: this time is linear in the decrease of the number of (maximal) $\mathcal{N}$-words due to `Denickify`. As we never split $\mathcal{N}$-words in the course of `MakeMinimal`, this is at most the number of $\mathcal{N}$-words occurring in E_1^* minus 1. $\square$

Theorem 20. For each DNA expression E_1^*, both the time and the space required by the function `MakeMinimal` are linear in $|E_1^*|$.

Proof: Let E_1^* be an arbitrary DNA expression.

We first observe that when we call `MakeMinimal` recursively for an expression-argument E_i of the current DNA expression E, we do not have to explicitly copy E_i, in order to pass it as a parameter to the recursive call. It is sufficient to make a 'call by reference.' For example, we may simply pass the position of the opening bracket of E_i, because it completely determines the expression-argument in E. The same holds for calls of the three procedures used by `MakeMinimal`. Hence, there is only a constant overhead (both in space and in time) due to a recursive call or a call of a procedure. In terms of complexity, we can ignore such overhead.

It is not hard to see that the data structure we propose requires space that is linear in $|E_1^*|$.

As for the time complexity, we know by Lemma 18 and Lemma 19 that we spend at most time that is linear in $|E_1^*|$ in procedures `Make`$\updownarrow$`ExprMinimal` and `Denickify`. We now analyse the total time spent (over all recursive calls of the function) in the other parts of `MakeMinimal`. Let us use $T_{\text{MM}}(E)$ to denote this time for a DNA expression E.

It is not hard to verify that every instruction in the function, other than the recursive calls and the calls of procedures `Make`$\updownarrow$`ExprMinimal` and `Denickify`, can be done in constant time. We can therefore define three constants for the time spent in specific parts of `MakeMinimal`:

c_1 is the maximum time required for an $\updownarrow$-expression E, except the time spent in recursive calls and procedure `Make`$\updownarrow$`ExprMinimal`;

c_2 is the maximum time required for an $\uparrow$-expression or $\downarrow$-expression E, except the time spent for each of its n arguments $\varepsilon_1, \ldots, \varepsilon_n$;

c_3 is the maximum time spent on an argument ε_i of an $\uparrow$-expression or $\downarrow$-expression E, except the time spent in recursive calls and procedure `Denickify`.

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 can prove by induction on the number of operators occurring in E, that $T_{\text{MM}}(E) \leq c^* \cdot |E| - c_3$. Here, we subtract c_3, to be prepared for the additional constant time required for every argument of an $\uparrow$-expression or $\downarrow$-expression E. $\square$

As mentioned before, if the original DNA expression E_1^* is minimal already, then the function `MakeMinimal` leaves it unchanged. 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 necessarily yield the same output for different equivalent inputs. In other words, the output of `MakeMinimal` cannot be considered as (some kind of) a normal form.

6. Conclusions

We have described a recursive algorithm, which rewrites a given DNA expression into an equivalent, minimal DNA expression. This is useful, e.g., to save space for storing a description of the DNA molecule denoted. In the algorithm, step by step, the DNA expression acquires the six properties that characterize minimal DNA expressions. 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. In [16], we use the algorithm as the first step of a larger algorithm, for rewriting arbitrary DNA expressions into some normal form.

Acknowledgement

The authors thank Grzegorz Rozenberg for useful comments on the manuscript.

References

[1] L.M. Adleman: Molecular computation of solutions to combinatorial problems, *Science* **266**, 1994, 1021–1024.

[2] 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.

[3] L. Cardelli, W. Shih (Eds.): *DNA Computing and Molecular Programming – 17th International Conference, DNA 17, Pasadena, CA, USA, September 19–23, 2011 – Proceedings*, LNCS 6937, Springer, Berlin, 2011.

[4] J. Chen, N. Jonoska, G. Rozenberg (Eds.): *Nanotechnology: Science and Computation*, Natural Computing Series, Springer, Berlin, 2006.

[5] T.H. Cormen, C.E. Leiserson, R.L. Rivest: *Introduction to Algorithms*, The MIT Press, Cambridge and McGraw-Hill, New York, 1990.

[6] A. Ehrenfeucht, T. Harju, I. Petre, D.M. Prescott, G. Rozenberg: *Computation in Living Cells – Gene Assembly in Ciliates*, Natural Computing Series, Springer, Berlin, 2004.

[7] H. Gu, J. Chao, S.-J. Xiao, N.C. Seeman: A proximity-based programmable DNA nanoscale assembly line, *Nature* **465**, 2010, 202–205.

[8] 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.

[9] L. Kari, S. Konstantinidis, P. Sosík: On properties of bond-free DNA languages, *Theoretical Computer Science* **334**, 2005, 131–159.

[10] 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.

[11] Gh. Păun, G. Rozenberg, A. Salomaa: *DNA Computing – New Computing Paradigms*, Springer, Berlin, 1998.

[12] P.W.K. Rothemund: Folding DNA to create nanoscale shapes and patterns, *Nature* **440**, 2006, 297–302.

[13] 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*, LNCS 6518, Springer, Berlin, 2011.

[14] R. van Vliet: *Combinatorial Aspects of Minimal DNA Expressions (ext.)*, Technical Report 2004-03, Leiden Institute of Advanced Computer Science, Leiden University, March 2004, see `www.liacs.nl/home/rvvliet/dnaexpressions/`

[15] R. van Vliet: *All about a Minimal Normal Form for DNA Expressions*, Technical Report 2011-03, Leiden Institute of Advanced Computer Science, Leiden University, July 2011, see `www.liacs.nl/home/rvvliet/dnaexpressions/`

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

[17] R. van Vliet, H.J. Hoogeboom, G. Rozenberg: Combinatorial aspects of minimal DNA expressions, *DNA Computing – 10th International Workshop on DNA Computing, DNA10, Milan, Italy, June 7–10, 2004 – Revised Selected Papers* (C. Ferretti, G. Mauri, C. Zandron, Eds.), LNCS 3384, Springer, Berlin, 2005, 375–388.

[18] R. van Vliet, H.J. Hoogeboom, G. Rozenberg: The construction of minimal DNA expressions, *Natural Computing* **5**, 2006, 127–149.

[19] E. Winfree: DNA computing by self-assembly, *The Bridge* **33**(4), 2003, 31–38.