Fundamenta Informaticae 123 (2013) 227–243
DOI 10.3233/FI-2012-808
IOS Press

A Minimal Normal Form for DNA Expressions

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 define a (minimal) normal form for the language of DNA expressions, and describe an algorithm to rewrite a given DNA expression into the normal form.

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

1. Introduction

Most research in the field of DNA computing concerns questions like what kind of DNA molecules (or other types of molecules) can be constructed, and what these molecules may be used for. In fact, DNA turns out to have unexpected applications, see, e.g., [2] and [1]. Less attention is paid in literature on formal ways to denote the molecules involved. Formal notations may, however, be useful, e.g., to describe precisely what computations are carried out with the molecules and what the results of these computations are.

In [5] and [6], we have introduced *DNA expressions*, as a formal notation for DNA molecules that may contain nicks (missing phosphodiester bonds between adjacent nucleotides in the same strand) and gaps (missing nucleotides in one of the strands). Different DNA expressions may denote the same DNA molecule. Such molecules are called *equivalent*. In [6], it is explained how to construct *minimal*

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

DNA expressions, i.e., the shortest possible DNA expressions denoting a given DNA molecule. This construction is based on certain characteristics of the molecule involved.

When one wants to decide whether or not two DNA expressions E_1 and E_2 are equivalent, one may determine the DNA molecules that they denote and check if these are the same. In this paper, we present a different approach. We define a *normal form*: a set of properties, such that for each DNA expression there is exactly one equivalent DNA expression with these properties. As the DNA expressions that satisfy the normal form are minimal, it is called a *minimal normal form*. We also describe an algorithm to rewrite an arbitrary DNA expression into the normal form. Now to decide whether or not E_1 and E_2 are equivalent, one determines their normal form versions and then checks if these are the same. This approach is elegant, because it operates at the level of DNA expressions only, rather than to refer to the DNA molecules denoted.

This paper is the second part of a diptych. Part 1, [4], describes an efficient, recursive algorithm to rewrite a given DNA expression into an equivalent, minimal DNA expression. For many DNA molecules, however, there exist several (equivalent) minimal DNA expressions. Depending on the input, the algorithm may yield each of these. Hence, by itself, this algorithm is not sufficient to produce a normal form. However, as we see in the present paper, the algorithm can function as a first step towards a true normal form.

The paper is organized as follows. Section 2 summarizes a few important definitions and results from earlier publications, illustrated by an example. The minimal normal form is introduced in Section 3. In Section 4, we describe a direct, recursive algorithm to rewrite a given DNA expression into the equivalent DNA expression in minimal normal form. The set-up of this algorithm is the same as that of the algorithm for minimality in [4]. Unfortunately, its time complexity turns out to be worse: at least quadratic in the worst case, whereas the algorithm for minimality is linear. Therefore, in Section 5, we describe an alternative, two-step algorithm. It first uses the algorithm from [4] to obtain a minimal DNA expression, and then rewrites the result into the minimal normal form. This two-step algorithm requires only linear time and space. Finally, in Section 6, we draw conclusions and suggest directions for future research.

We assume the reader to be familiar with the terminology and notation used in part 1 of the diptych, [4]. We only repeat those definitions and results that we want to refer to frequently, or that are otherwise important for understanding the present paper. More details about the minimal normal form and the algorithm we describe in this paper can be found in the technical report [3].

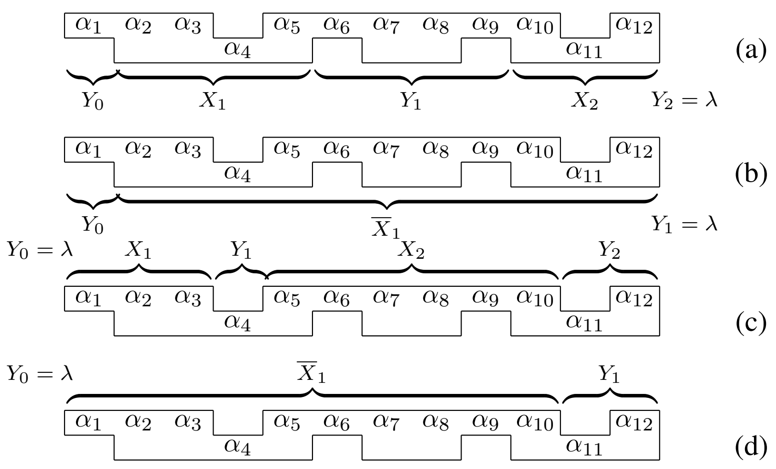
2. Minimal DNA expressions

In [6], it is described how to construct minimal DNA expressions denoting a given formal DNA molecule. For perfect double-stranded molecules, the construction is trivial:

Theorem 1. 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 $\mathcal{S}(E) = \binom{\alpha_1}{c(\alpha_1)}$.

For nick free molecules with at least one single-stranded component, the construction is more involved. We need the following terminology to describe the construction:

Definition 2. Let X be a nick free formal DNA molecule.



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 $\overline{X}_1$ $Y_1 = \lambda$
$Y_0 = \lambda$ X_1 Y_1 X_2 Y_2
(c)
$Y_0 = \lambda$ $\overline{X}_1$ Y_1
(d)

Figure 1. Pictorial representation of the formal DNA molecule X from Example 3. (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.

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 partitioning $Y_0 X_1 Y_1 \ldots X_{r_0} Y_{r_0}$ of X, such that $X_1, \ldots, X_{r_0}$ are all primitive lower blocks of X.

A lower block $\overline{X}_1$ of X is a substring of X that starts and ends with a primitive lower block.

A lower block partitioning of X is a partitioning $Y_0 \overline{X}_1 Y_1 \ldots \overline{X}_r Y_r$ of X, such that each $\overline{X}_j$ is a lower block of X, and each primitive lower block of X is contained in an $\overline{X}_j$.

A primitive upper block, the primitive upper block partitioning, an upper block and an upper block partitioning are defined completely analogously. We use $B_\downarrow(X)$ and $B_\uparrow(X)$ to denote the number of primitive lower blocks and the number of primitive upper blocks of X, respectively.

Example 3. In Figure 1, we have depicted the two possible lower block partitionings and the two possible upper block partitionings of the nick free formal DNA molecule

$$X = \binom{\alpha_1}{-}\binom{\alpha_2\alpha_3}{c(\alpha_2\alpha_3)}\binom{-}{\alpha_4}\binom{\alpha_5}{c(\alpha_5)}\binom{\alpha_6}{-}\binom{\alpha_7\alpha_8}{c(\alpha_7\alpha_8)}\binom{\alpha_9}{-}\binom{\alpha_{10}}{c(\alpha_{10})}\binom{-}{\alpha_{11}}\binom{\alpha_{12}}{c(\alpha_{12})}. \tag{1}$$

The submolecules X_1 and X_2 in Figure 1(a) are the two primitive lower blocks of X. The submolecules X_1 and X_2 in Figure 1(c) are the two primitive upper blocks of X. Hence, $B_\downarrow(X) = B_\uparrow(X) = 2$. Note that the two lower block partitionings end with an empty substring Y_2 or Y_1, respectively. The two upper block partitionings start with an empty substring Y_0. ■

We then have

Theorem 4. Let X be a nick free formal DNA molecule which contains at least one single-stranded component, and let $x'_1 \ldots x'_k$ for some $k \geq 1$ be the decomposition of X.

1. If $B_{\uparrow}(X) \geq B_{\downarrow}(X)$, then
   - let $\mathcal{P} = Y_0\overline{X}_1Y_1 \dots \overline{X}_rY_r$ for some $r \geq 0$ be an arbitrary lower block partitioning of X;
   - for $j = 1, \dots, r$, let E_j be an arbitrary minimal DNA expression denoting $\overline{X}_j$;
   - for $j = 0, 1, \dots, r$, let $Y_j = x'_{a_j} \dots x'_{b_j}$ for some $a_j \geq 1$ and $b_j \leq k$;
   - for $j = 0, 1, \dots, r$ and for $i = a_j, \dots, b_j$, let

$$\varepsilon_i = \begin{cases} \alpha_i & \text{if } x'_i = \binom{\alpha_i}{-} \text{ for an } \mathcal{N}\text{-word } \alpha_i \\ \langle \updownarrow \alpha_i \rangle & \text{if } x'_i = \binom{\alpha_i}{c(\alpha_i)} \text{ for an } \mathcal{N}\text{-word } \alpha_i; \end{cases} \quad \text{and} \tag{2}$$

   - let

$$E = \langle \uparrow \varepsilon_{a_0} \dots \varepsilon_{b_0} E_1 \varepsilon_{a_1} \dots \varepsilon_{b_1} \ \dots \ E_r \varepsilon_{a_r} \dots \varepsilon_{b_r} \rangle . \tag{3}$$

   Then

   (a) all ingredients needed to construct E (i.e., the lower block partitioning $\mathcal{P}$, the minimal DNA expressions E_j, the indices a_j and b_j, and the arguments ε_i) are well defined, and

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

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

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

Note that if $B_{\uparrow}(X) = B_{\downarrow}(X) \geq 1$, then both claims are applicable, and we obtain both minimal $\uparrow$-expressions and minimal $\downarrow$-expressions denoting X. Different lower (or upper) block partitionings result in different, but of course equivalent, minimal DNA expressions. All minimal DNA expressions denoting a nick free formal DNA molecule with at least one single-stranded component satisfy the construction from Theorem 4.

Example 5. Let X be the formal DNA molecule from Example 3. As $B_{\uparrow}(X) = B_{\downarrow}(X) = 2$, we can use both claims in Theorem 4 to construct a minimal DNA expression denoting X. The minimal DNA expressions corresponding to the four partitionings depicted in Figure 1 are

$$E_a = \langle \uparrow \alpha_1 \langle \downarrow \langle \updownarrow \alpha_2\alpha_3 \rangle \alpha_4 \langle \updownarrow \alpha_5 \rangle\rangle \alpha_6 \langle \updownarrow \alpha_7\alpha_8 \rangle \alpha_9 \langle \downarrow \langle \updownarrow \alpha_{10} \rangle \alpha_{11} \langle \updownarrow \alpha_{12} \rangle\rangle\rangle , \tag{5}$$

$$E_b = \langle \uparrow \alpha_1 \langle \downarrow \langle \updownarrow \alpha_2\alpha_3 \rangle \alpha_4 \langle \uparrow \langle \updownarrow \alpha_5 \rangle \alpha_6 \langle \updownarrow \alpha_7\alpha_8 \rangle \alpha_9 \langle \updownarrow \alpha_{10} \rangle\rangle \alpha_{11} \langle \updownarrow \alpha_{12} \rangle\rangle\rangle , \tag{6}$$

$$E_c = \langle \downarrow \langle \uparrow \alpha_1 \langle \updownarrow \alpha_2\alpha_3 \rangle\rangle \alpha_4 \langle \uparrow \langle \updownarrow \alpha_5 \rangle \alpha_6 \langle \updownarrow \alpha_7\alpha_8 \rangle \alpha_9 \langle \updownarrow \alpha_{10} \rangle\rangle \alpha_{11} \langle \updownarrow \alpha_{12} \rangle\rangle , \tag{7}$$

$$E_d = \langle \downarrow \langle \uparrow \alpha_1 \langle \downarrow \langle \updownarrow \alpha_2\alpha_3 \rangle \alpha_4 \langle \updownarrow \alpha_5 \rangle\rangle \alpha_6 \langle \updownarrow \alpha_7\alpha_8 \rangle \alpha_9 \langle \updownarrow \alpha_{10} \rangle\rangle \alpha_{11} \langle \updownarrow \alpha_{12} \rangle\rangle . \tag{8}$$

■

In the above example, there was exactly one minimal DNA expression for each lower block partitioning and each upper block partitioning. In general, however, there may be more than one minimal DNA expression corresponding to the same partitioning.

For a formal DNA molecule X with nick letters, minimal DNA expressions are constructed in a similar way. The construction is based on lower block partitionings (if X has lower nick letters) or upper block partitionings (if X has upper nick letters) of the nick free pieces of X. Note that formal DNA molecules with both upper nick letters and lower nick letters are not expressible.

In order to decide whether or not a given DNA expression is minimal, we can use the following characterization from [5]. It describes syntactic requirements on the arguments of the occurring operators. It is valid both for nick free DNA expressions and for DNA expressions with nicks:

Theorem 6. 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,
- 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.

The adjective 'alternating' in Property ($\mathcal{D}_{\mathrm{Min}}$.4) means that the arguments of the operator involved are $\mathcal{N}$-words and DNA expressions, alternately. In this definition, we consider consecutive $\mathcal{N}$-word-arguments as a single argument. If the outermost operator of an $\uparrow$-expression or $\downarrow$-expression E is alternating, then E itself is also called alternating.

It is easily verified that indeed the four DNA expressions from (5)–(8) have all the properties from Theorem 6. On the other hand, the DNA expression

$$\begin{aligned} E = {} & \langle\downarrow \langle\uparrow \alpha_1 \langle\updownarrow \langle\uparrow \alpha_2 \langle\updownarrow \langle\updownarrow \alpha_3\rangle\rangle\rangle\rangle\rangle \\ & \quad \langle\uparrow \langle\downarrow \alpha_4 \langle\uparrow \langle\updownarrow \alpha_5\rangle \alpha_6 \langle\downarrow \langle\updownarrow \alpha_7\rangle \langle\updownarrow \alpha_8\rangle\rangle\rangle\rangle \alpha_9 \langle\uparrow \langle\downarrow \langle\updownarrow \alpha_{10}\rangle \alpha_{11} \langle\updownarrow \alpha_{12}\rangle\rangle\rangle\rangle \, \rangle \,, \end{aligned} \tag{9}$$

which denotes the same molecule but is much longer, has only Property ($\mathcal{D}_{\mathrm{Min}}$.6).

For certain DNA expressions it is easy to decide whether or not they are nick free:

Lemma 7. Let E be an $\uparrow$-expression with Properties ($\mathcal{D}_{\mathrm{Min}}$.2), ($\mathcal{D}_{\mathrm{Min}}$.4) and ($\mathcal{D}_{\mathrm{Min}}$.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.

3. A minimal normal form

Minimal DNA expressions can be considered as the 'best' DNA expressions to denote a given formal DNA molecule. Therefore, it is desirable that the normal form DNA expressions are, in particular, minimal. In this section, we present a normal form that achieves this goal. For that reason, it is called a *minimal normal form*.

We define the minimal normal form in a constructive way. That is, for each expressible formal DNA molecule, we describe how to construct the corresponding DNA expression in minimal normal form. Later, we list five properties that characterize the normal form DNA expressions in terms of the arguments of the operators occurring in them.

As we have seen, for many formal DNA molecules, there exists more than one minimal DNA expression. For example, there are four minimal DNA expressions denoting the molecule from (1), which are given in (5)–(8). From among all different minimal DNA expressions denoting the same formal DNA molecule, we have to choose one to be the normal form DNA expression. We do this by explicitly fixing the choices that are made in the construction of a minimal DNA expression (see Theorem 4).

First, this construction is based on lower block partitionings and upper block partitionings of nick free (sub)molecules. If these partitionings are not unique, then the resulting DNA expression depends on the partitionings that we choose. Here, we make a very natural choice: we always use the *primitive* lower block partitioning or upper block partitioning.

In addition, if a formal DNA molecule X is nick free, contains at least one single-stranded component and $B_{\uparrow}(X) = B_{\downarrow}(X)$, then by Theorem 4, there exist both minimal $\uparrow$-expressions and minimal $\downarrow$-expressions for X. Here, our choice for an $\uparrow$-expression or a $\downarrow$-expression is determined by the first single-stranded component of X. An upper component results in an $\uparrow$-expression; a lower component results in a $\downarrow$-expression.

We thus have the following definition of the minimal normal form for the nick free case, where $E_{\text{MinNF}}(X)$ denotes the normal form DNA expression denoting X:

Definition 8. Let X be a nick free formal DNA molecule.

1. If $X = \binom{\alpha_1}{c(\alpha_1)}$ for an $\mathcal{N}$-word α_1, then $E_{\text{MinNF}}(X) = \langle\updownarrow \alpha_1\rangle$.
2. If X contains at least one single-stranded component and $B_{\uparrow}(X) = B_{\downarrow}(X)$, then
   (a) if the first single-stranded component of X is an upper component, then $E_{\text{MinNF}}(X)$ is the minimal $\uparrow$-expression denoting X based on the primitive lower block partitioning of X, as described in Theorem 4(1);
   (b) if the first single-stranded component of X is a lower component, then ... (symmetric to Case 2a).
3. If $B_{\uparrow}(X) > B_{\downarrow}(X)$, then $E_{\text{MinNF}}(X)$ is the minimal $\uparrow$-expression denoting X based on the primitive lower block partitioning of X, as described in Theorem 4(1).
4. If $B_{\downarrow}(X) > B_{\uparrow}(X)$, then ... (symmetric to Case 3).

For an expressible formal DNA molecule X with nick letters, $E_{\text{MinNF}}(X)$ is based on the primitive lower block partitionings (if X has lower nick letters) or primitive upper block partitionings (if X has upper nick letters) of the nick free pieces of X. We describe the details in [3].

Example 9. Let X be the nick free formal DNA molecule from (1). We have $B_\uparrow(X) = B_\downarrow(X) = 2$, and the first single-stranded component of X is an upper component. According to Case 2a of Definition 8, $E_{\text{MinNF}}(X)$ is the minimal $\uparrow$-expression based on the primitive lower block partitioning of X. This is the lower block partitioning from Figure 1(a). Hence, $E_{\text{MinNF}}(X)$ is the $\uparrow$-expression E_a from (5). ■

Theorem 6 characterized the minimal DNA expressions in general, in terms of operators and arguments. We have a similar result for the DNA expressions in minimal normal form. It applies both to nick free molecules and to molecules with nicks.

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

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

($\mathcal{D}_{\text{MinNF}}$.2) no occurrence of the operator $\uparrow$ in E has an $\uparrow$-argument, and no occurrence of the operator $\downarrow$ in E has a $\downarrow$-argument; and

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

($\mathcal{D}_{\text{MinNF}}$.4) for each inner occurrence of an operator $\uparrow$ or $\downarrow$ in E, the arguments are $\mathcal{N}$-words α and $\updownarrow$-expressions $\langle\updownarrow \alpha\rangle$ for $\mathcal{N}$-words α, alternately; and

($\mathcal{D}_{\text{MinNF}}$.5) if the outermost operator of E is $\uparrow$ or $\downarrow$, then

- either its first argument is an $\mathcal{N}$-word α or an $\updownarrow$-expression $\langle\updownarrow \alpha\rangle$ for an $\mathcal{N}$-word α,
- or it has two consecutive expression-arguments.

Note that Properties ($\mathcal{D}_{\text{MinNF}}$.1), ($\mathcal{D}_{\text{MinNF}}$.2) and ($\mathcal{D}_{\text{MinNF}}$.3) are equal to Properties ($\mathcal{D}_{\text{Min}}$.1), ($\mathcal{D}_{\text{Min}}$.2) and ($\mathcal{D}_{\text{Min}}$.3) of (general) minimal DNA expressions.

Property ($\mathcal{D}_{\text{MinNF}}$.4) includes Properties ($\mathcal{D}_{\text{Min}}$.4) and ($\mathcal{D}_{\text{Min}}$.5). It is stronger, however, than these two properties together. The property ensures that, unlike in (general) minimal DNA expressions, the nesting level of the brackets is limited to 3. This is due to the choice for *primitive* lower block partitionings and *primitive* upper block partitionings in the definition of the minimal normal form.

Finally, Property ($\mathcal{D}_{\text{MinNF}}$.5) is a stronger version of Property ($\mathcal{D}_{\text{Min}}$.6). The difference between the two properties is caused by the second choice we make in the definition of the minimal normal form: if $B_\uparrow(X) = B_\downarrow(X) \geq 1$ for a nick free formal DNA molecule X, then the first single-stranded component of X determines whether $E_{\text{MinNF}}(X)$ is an $\uparrow$-expression or a $\downarrow$-expression.

The $\uparrow$-expression E_a from (5), which is in minimal normal form, may serve as an illustration. It is easily verified that it has all five properties, and in particular Properties ($\mathcal{D}_{\text{MinNF}}$.4) and ($\mathcal{D}_{\text{MinNF}}$.5).

Proof of Theorem 10: We only give the proof for the nick free case, because we have not precisely defined the minimal normal form for molecules with nicks. However, by Lemma 7, a minimal $\uparrow$-expression or $\downarrow$-expression with nicks has two consecutive expression-arguments. Then it certainly has Property ($\mathcal{D}_{\text{MinNF}}$.5).

"$\Longrightarrow$" Let E be a DNA expression in minimal normal form, and let $X = \mathcal{S}(E)$. If E is an $\updownarrow$-expression, then $E = \langle\updownarrow \alpha_1\rangle$ for an $\mathcal{N}$-word α_1, and it obviously has all the properties from the claim.

Now assume, that E is a nick free $\uparrow$-expression. Because E is minimal, it has Properties ($\mathcal{D}_{\text{Min}}$.1)–($\mathcal{D}_{\text{Min}}$.6) from Theorem 6. Then it certainly has Properties ($\mathcal{D}_{\text{MinNF}}$.1)–($\mathcal{D}_{\text{MinNF}}$.3). By Definition 8, E is based on the primitive lower block partitioning $Y_0\overline{X}_1Y_1\ldots\overline{X}_rY_r$ of X, as described in Theorem 4(1).

Each primitive lower block $\overline{X}_j$ in the partitioning consists of only lower components (at least one) and double components. Hence, $B_\downarrow(\overline{X}_j) = 1$ and $B_\uparrow(\overline{X}_j) = 0$. It is easily verified that, according to the construction from Theorem 4, the only minimal DNA expression E_j denoting $\overline{X}_j$ is a $\downarrow$-expression, with as arguments an alternating sequence of $\mathcal{N}$-words α and $\updownarrow$-expressions $\langle\updownarrow\alpha\rangle$. This implies that E has Property ($\mathcal{D}_{\text{MinNF}}$.4).

By Lemma 7, E (which is nick free) does not have consecutive expression-arguments. Hence, to establish Property ($\mathcal{D}_{\text{MinNF}}$.5), we must examine its first argument. Both if $B_\uparrow(X) = B_\downarrow(X)$ and if $B_\uparrow(X) > B_\downarrow(X)$, the first single-stranded component of X is an upper component. This implies that the substring Y_0 in the primitive lower block partitioning is non-empty, and thus yields at least one argument for E. Then it follows from (2) and (3) that the first argument of E is either an $\mathcal{N}$-word α or an $\updownarrow$-expression $\langle\updownarrow\alpha\rangle$. Indeed, E also has Property ($\mathcal{D}_{\text{MinNF}}$.5).

"$\Longleftarrow$" If a DNA expression E has Properties ($\mathcal{D}_{\text{MinNF}}$.1)–($\mathcal{D}_{\text{MinNF}}$.5), then it certainly has Properties ($\mathcal{D}_{\text{Min}}$.1)–($\mathcal{D}_{\text{Min}}$.6), and thus is minimal. Let $X = \mathcal{S}(E)$. If E is an $\updownarrow$-expression, then by Property ($\mathcal{D}_{\text{MinNF}}$.1), it must be $\langle\updownarrow\alpha_1\rangle$ for an $\mathcal{N}$-word α_1, which is in minimal normal form.

We now assume that E is a nick free $\uparrow$-expression. Because E is minimal, it satisfies the construction from Theorem 4 and we must have $B_\uparrow(X) \geq B_\downarrow(X)$. By Lemma 7, E is alternating. We can use Property ($\mathcal{D}_{\text{MinNF}}$.4) to prove that each $\downarrow$-argument E_j of E denotes a primitive lower block of X. This implies that E is based on the *primitive* lower block partitioning of X. Moreover, by Property ($\mathcal{D}_{\text{MinNF}}$.5), the first argument of E is either an $\mathcal{N}$-word α or an $\updownarrow$-expression $\langle\updownarrow\alpha\rangle$. In the latter case, because E is alternating, the second argument is an $\mathcal{N}$-word. This implies that the first single-stranded component of X is an upper component. Indeed, both if $B_\uparrow(X) = B_\downarrow(X)$ and if $B_\uparrow(X) > B_\downarrow(X)$, E is in minimal normal form. □

Note that both the language of all DNA expressions and the language of all minimal DNA expressions are context-free but not regular. Due to the limited nesting level of the brackets, the language of all DNA expressions in minimal normal form turns out to be regular. In [3], this is proven by (1) defining a context-free grammar generating the language, and (2) proving that this grammar is not self-embedding.

4. Recursive algorithm for the minimal normal form

In Section 4 of [4], we have described a recursive function `MakeMinimal` for rewriting an arbitrary DNA expression E into an equivalent, minimal DNA expression. For an expression-argument E_i of E, the function first performs a recursive call. If necessary, the result is subject to some local rearrangements, to make E minimal itself. We have established that this approach requires both linear time and space. Therefore, when we wish to rewrite an arbitrary DNA expression E into an equivalent DNA expression in minimal normal form, our first attempt may be a recursive function `MakeMinimalNF`: we first rewrite the arguments of E into the minimal normal form, and then deal with the DNA expression as a whole. The global set-up of this function is given in Figure 2.

Note that lines 6 and 12 should not be implemented by a call `MakeMinimalNF`(E), because that would trigger an infinite sequence of recursive calls, with the same argument E. Instead, we should try

```
1.   MakeMinimalNF ($E$)
       // recursively rewrites an arbitrary DNA expression $E$
       // into an equivalent DNA expression in minimal normal form
2.   {
3.     if ($E$ is an $\updownarrow$-expression)
4.     then if (the argument of $E$ is a DNA expression $E_1$)
5.          then MakeMinimalNF ($E_1$);
6.               substitute $E$ by a DNA expression $E'$ in minimal normal form
                   satisfying $E' \equiv E$;
7.          fi

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

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

to devise procedures consisting of local rearrangements at the string level, which make sure that a DNA expression with normal form arguments becomes in normal form itself.

Note also that the structure of MakeMinimalNF is equal to that of MakeMinimal in [4]. The main difference between the description of MakeMinimal and that of MakeMinimalNF is that the former has more detail. Both lines 6–10 and lines 16–35 of MakeMinimal are an implementation of the general statement '*substitute E by a minimal DNA expression E' satisfying $E' \equiv E$*' (cf. lines 6 and 12 of MakeMinimalNF).

Regardless of the actual implementations of lines 6 and 12 of MakeMinimalNF, we can already draw an important conclusion: the recursive approach of the function is not as efficient as that of MakeMinimal. We demonstrate this by examining its complexity for DNA expressions of a specific type.

Example 11. Let α be an arbitrary $\mathcal{N}$-word, and let

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

Hence,

$$\begin{aligned}
E_1 &= \langle\downarrow \langle\updownarrow \alpha\rangle\, \alpha\, \langle\updownarrow \alpha\rangle\rangle\,, \\
E_2 &= \langle\uparrow \langle\updownarrow \alpha\rangle\, \alpha\, \langle\downarrow \langle\updownarrow \alpha\rangle\, \alpha\, \langle\updownarrow \alpha\rangle\rangle\, \alpha\, \langle\updownarrow \alpha\rangle\rangle\,, \\
E_3 &= \langle\downarrow \langle\updownarrow \alpha\rangle\, \alpha\, \langle\uparrow \langle\updownarrow \alpha\rangle\, \alpha\, \langle\downarrow \langle\updownarrow \alpha\rangle\, \alpha\, \langle\updownarrow \alpha\rangle\rangle\, \alpha\, \langle\updownarrow \alpha\rangle\rangle\, \alpha\, \langle\updownarrow \alpha\rangle\rangle\,, \\
E_4 &= \langle\uparrow \langle\updownarrow \alpha\rangle\, \alpha\, \langle\downarrow \langle\updownarrow \alpha\rangle\, \alpha\, \langle\uparrow \langle\updownarrow \alpha\rangle\, \alpha\, \langle\downarrow \langle\updownarrow \alpha\rangle\, \alpha\, \langle\updownarrow \alpha\rangle\rangle\, \alpha\, \langle\updownarrow \alpha\rangle\rangle\, \alpha\, \langle\updownarrow \alpha\rangle\rangle\, \alpha\, \langle\updownarrow \alpha\rangle\rangle\,, \\
&\quad \text{etc.}
\end{aligned}$$

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

- both E_{2p} and E_{2p+1} are DNA expressions,
- $$\begin{aligned}
\mathcal{S}(E_{2p}) &= \binom{\alpha}{c(\alpha)}\binom{\alpha}{-}\underbrace{\binom{\alpha}{c(\alpha)}\binom{-}{\alpha}\binom{\alpha}{c(\alpha)}\binom{\alpha}{-}\cdots\binom{\alpha}{c(\alpha)}\binom{-}{\alpha}\binom{\alpha}{c(\alpha)}\binom{\alpha}{-}}_{p-1\text{ times}}\cdot\\
&\quad\ \binom{\alpha}{c(\alpha)}\binom{-}{\alpha}\binom{\alpha}{c(\alpha)}\cdot\\
&\quad\ \underbrace{\binom{\alpha}{-}\binom{\alpha}{c(\alpha)}\binom{-}{\alpha}\binom{\alpha}{c(\alpha)}\cdots\binom{\alpha}{-}\binom{\alpha}{c(\alpha)}\binom{-}{\alpha}\binom{\alpha}{c(\alpha)}}_{p-1\text{ times}}\binom{\alpha}{-}\binom{\alpha}{c(\alpha)}\\
&= \binom{\alpha}{c(\alpha)}\binom{\alpha}{-}\underbrace{\binom{\alpha}{c(\alpha)}\binom{-}{\alpha}\binom{\alpha}{c(\alpha)}\binom{\alpha}{-}\cdots\binom{\alpha}{c(\alpha)}\binom{-}{\alpha}\binom{\alpha}{c(\alpha)}\binom{\alpha}{-}}_{2p-1\text{ times}}\binom{\alpha}{c(\alpha)},\\
\mathcal{S}(E_{2p+1}) &= \underbrace{\binom{\alpha}{c(\alpha)}\binom{-}{\alpha}\binom{\alpha}{c(\alpha)}\binom{\alpha}{-}\cdots\binom{\alpha}{c(\alpha)}\binom{-}{\alpha}\binom{\alpha}{c(\alpha)}\binom{\alpha}{-}}_{p\text{ times}}\cdot\\
&\quad\ \binom{\alpha}{c(\alpha)}\binom{-}{\alpha}\binom{\alpha}{c(\alpha)}\cdot\\
&\quad\ \underbrace{\binom{\alpha}{-}\binom{\alpha}{c(\alpha)}\binom{-}{\alpha}\binom{\alpha}{c(\alpha)}\cdots\binom{\alpha}{-}\binom{\alpha}{c(\alpha)}\binom{-}{\alpha}\binom{\alpha}{c(\alpha)}}_{p\text{ times}}\\
&= \binom{\alpha}{c(\alpha)}\binom{-}{\alpha}\underbrace{\binom{\alpha}{c(\alpha)}\binom{\alpha}{-}\binom{\alpha}{c(\alpha)}\binom{-}{\alpha}\cdots\binom{\alpha}{c(\alpha)}\binom{\alpha}{-}\binom{\alpha}{c(\alpha)}\binom{-}{\alpha}}_{2p\text{ times}}\binom{\alpha}{c(\alpha)},
\end{aligned}$$
- $$\begin{aligned}
B_{\uparrow}(\mathcal{S}(E_{2p})) &= B_{\downarrow}(\mathcal{S}(E_{2p}))+1 = 2p,\\
B_{\downarrow}(\mathcal{S}(E_{2p+1})) &= B_{\uparrow}(\mathcal{S}(E_{2p+1}))+1 = 2p+1,
\end{aligned}$$
- $n_{\updownarrow}(\mathcal{S}(E_q)) = 2q$, both if $q = 2p$ and if $q = 2p+1$,
- $|E_q| = 3\cdot 3q + (4q-1)\cdot|\alpha|$, both if $q = 2p$ and if $q = 2p+1$.

In particular, E_{2p} and E_{2p+1} are nick free, and their lengths are linear in p. Moreover, both E_{2p} and E_{2p+1} are minimal, because they achieve the minimal lengths mentioned in Theorem 4. However, for $q \geq 3$, E_q is not in minimal normal form, because it violates Property ($\mathcal{D}_{\text{MinNF}}$.4).

By Definition 8(3) and (4) and the construction from Theorem 4, the corresponding DNA expressions in minimal normal form are

$$\begin{aligned}
E'_{2p} &= E_{\text{MinNF}}(\mathcal{S}(E_{2p}))\\
&= \left\langle \uparrow\langle\updownarrow\alpha\rangle\,\alpha\,\underbrace{\langle\downarrow\langle\updownarrow\alpha\rangle\,\alpha\,\langle\updownarrow\alpha\rangle\rangle\,\alpha\ldots\langle\downarrow\langle\updownarrow\alpha\rangle\,\alpha\,\langle\updownarrow\alpha\rangle\rangle\,\alpha}_{2p-1\text{ times}}\,\langle\updownarrow\alpha\rangle\right\rangle, \qquad (10)\\
E'_{2p+1} &= E_{\text{MinNF}}(\mathcal{S}(E_{2p+1}))\\
&= \left\langle \downarrow\langle\updownarrow\alpha\rangle\,\alpha\,\underbrace{\langle\uparrow\langle\updownarrow\alpha\rangle\,\alpha\,\langle\updownarrow\alpha\rangle\rangle\,\alpha\ldots\langle\uparrow\langle\updownarrow\alpha\rangle\,\alpha\,\langle\updownarrow\alpha\rangle\rangle\,\alpha}_{2p\text{ times}}\,\langle\updownarrow\alpha\rangle\right\rangle.
\end{aligned}$$

Now, let $p \geq 1$ and let us apply the function `MakeMinimalNF` to the $\downarrow$-expression E_{2p+1}, with the $\uparrow$-expression E_{2p} as one of its arguments. When we call the function recursively for E_{2p}, this argument is rewritten into the $\uparrow$-expression E'_{2p}. The other two expression-arguments $\langle \updownarrow \alpha \rangle$ of E_{2p+1} are already in minimal normal form. In order to rewrite the result

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

into the corresponding DNA expression in minimal normal form E'_{2p+1}, we must remove the $2p-1$ occurrences of $\downarrow$ in E'_{2p}, add $2p-1$ occurrences of $\uparrow$ at other positions in the DNA expression, and also rearrange the brackets. This requires time that is linear in p.

Likewise, at a higher level of the recursion, we have had to rearrange $2p-2, 2p-3, 2p-4, \ldots, 1$ occurrences of operators in $E'_{2p-1}, E'_{2p-2}, E'_{2p-3}, \ldots, E'_2$, respectively. Altogether, this takes time that is quadratic in p, and thus in the length of E_{2p+1}.

The analysis for the $\uparrow$-expression E_{2p} is completely analogous. ■

5. Two-step algorithm for the minimal normal form

Instead of the direct, recursive approach by the function `MakeMinimalNF`, we propose a two-step approach to construct the normal form version of a given DNA expression E_1^*. We first use the function `MakeMinimal` from [4] to produce an equivalent, minimal DNA expression E_2^*. We subsequently rewrite E_2^* into the equivalent DNA expression in minimal normal form E_3^*.

In Figure 3, we give pseudo-code for the algorithm `NormalizeMinimal`, which performs this second step. It can be used both for nick free DNA expressions and for DNA expressions with nicks.

The two substitutions in the algorithm are justified by violations of a property from Theorem 10. At each of them, we have indicated the property involved. Note that Properties ($\mathcal{D}_{\text{MinNF}}$.1)–($\mathcal{D}_{\text{MinNF}}$.3) are not mentioned. This is natural, as they equal Properties ($\mathcal{D}_{\text{Min}}$.1)–($\mathcal{D}_{\text{Min}}$.3) of minimal DNA expressions, and the input of `NormalizeMinimal` is supposed to be minimal. The first substitution, in line 8, involves the procedure `RotateToMinimal` that was also used by the function `MakeMinimal`. In Figure 4, we have copied the pseudo-code of this procedure from [4]. To illustrate the algorithm, we continue with an example from [4].

Example 12. We again consider the formal DNA molecule X depicted in Figure 1. In (9), we have given a DNA expression denoting X, which is not minimal. In Example 12 in [4], we have used `MakeMinimal` to rewrite this DNA expression into an equivalent, minimal DNA expression. The result was

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

which equals E_c from (7). We apply `NormalizeMinimal` to E.

The $\downarrow$-expression E is alternating. Because its first argument is the $\uparrow$-expression $E_1 = \langle \uparrow \alpha_1 \langle \updownarrow \alpha_2\alpha_3 \rangle\rangle$, E violates Property ($\mathcal{D}_{\text{MinNF}}$.5). According to (the analogue for $\downarrow$-expressions of) line 8 of algorithm `NormalizeMinimal` and line RtM.6 of procedure `RotateToMinimal`, E is substituted by

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

This is the minimal DNA expression E_b from (6). The second argument of E is the $\downarrow$-expression

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

```
1.   NormalizeMinimal ($E_2^*$)
       // locally rewrites an arbitrary minimal DNA expression $E_2^*$
       // into a DNA expression $E_3^*$ in minimal normal form satisfying $E_3^* \equiv E_2^*$
2.   {
3.     $E = E_2^*$;
4.     if ($E$ is an $\updownarrow$-expression)
5.     then $E_3^* = E$;
6.     else   // $E$ is an $\uparrow$-expression or a $\downarrow$-expression;
              // without loss of generality, assume it is an $\uparrow$-expression
7.          if ($E$ is alternating and its first argument is a $\downarrow$-argument)
8.          then substitute $E$ by the result of procedure RotateToMinimal;
                                                                   ($\mathcal{D}_{\mathrm{MinNF}}$.5)
9.          fi
              // $E$ is an $\uparrow$-expression or a $\downarrow$-expression;
              // without loss of generality, assume it is an $\uparrow$-expression
10.         $\widehat{\varepsilon}$ = first argument of $E$;
11.         stop = false;
12.         while (not stop)
13.         do   if ($\widehat{\varepsilon}$ is a $\downarrow$-expression with at least one $\uparrow$-argument)
                   // let $\widehat{\varepsilon} = \langle\downarrow \varepsilon_1 \dots \varepsilon_{i-1} \langle\uparrow \varepsilon_{i,1} \dots \varepsilon_{i,m}\rangle \varepsilon_{i+1} \dots \varepsilon_n\rangle$,
                   // where $\langle\uparrow \varepsilon_{i,1} \dots \varepsilon_{i,m}\rangle$ is the first $\uparrow$-argument of $\widehat{\varepsilon}$
14.              then substitute $\widehat{\varepsilon}$ in $E$
                        by $\langle\downarrow \varepsilon_1 \dots \varepsilon_{i-1}\varepsilon_{i,1}\rangle \varepsilon_{i,2} \dots \varepsilon_{i,m-1} \langle\downarrow \varepsilon_{i,m}\varepsilon_{i+1} \dots \varepsilon_n\rangle$;    ($\mathcal{D}_{\mathrm{MinNF}}$.4)
15.                   $\widehat{\varepsilon} = \varepsilon_{i,2}$;
16.              else if ($\widehat{\varepsilon}$ is not the last argument of $E$)
17.                   then $\widehat{\varepsilon}$ = next argument of $E$;
18.                   else stop = true;
19.                   fi
20.              fi
21.         od
22.         $E_3^* = E$;
23.    fi
24.  }
```

Figure 3. Pseudo-code of the algorithm NormalizeMinimal.

whose third argument is the $\uparrow$-expression $E_3 = \langle\uparrow \langle\updownarrow \alpha_5\rangle \alpha_6 \langle\updownarrow \alpha_7\alpha_8\rangle \alpha_9 \langle\updownarrow \alpha_{10}\rangle\rangle$. Because the occurrence of the operator $\downarrow$ in $\widehat{\varepsilon}$ is an inner occurrence in E, it violates Property ($\mathcal{D}_{\mathrm{MinNF}}$.4). According to line 14 of algorithm NormalizeMinimal, $\widehat{\varepsilon}$ is substituted in E by the sequence of arguments

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

yielding

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

After the substitution, we proceed with $\widehat{\varepsilon} = \alpha_6$. In this case, the only other $\downarrow$-argument that we consider in the while-loop, is $\langle\downarrow \langle\updownarrow \alpha_{10}\rangle \alpha_{11} \langle\updownarrow \alpha_{12}\rangle\rangle$, which does satisfy Property ($\mathcal{D}_{\mathrm{MinNF}}$.4). Indeed, the DNA expression in (11) has all five properties from Theorem 10, and thus is in minimal normal form. It equals the minimal DNA expression E_a from (5). ■

```
RtM.1.  RotateToMinimal ($E$)
          // locally rewrites an alternating $\downarrow$-expression $E = \langle \downarrow \varepsilon_1 \dots \varepsilon_n \rangle$
          // with Properties ($\mathcal{D}_{\texttt{Min}}$.1)-($\mathcal{D}_{\texttt{Min}}$.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 4. Pseudo-code of the procedure `RotateToMinimal`.

In general, in the while-loop in algorithm `NormalizeMinimal`, we may encounter several $\downarrow$-arguments $\widehat{\varepsilon}$ that do not satisfy Property ($\mathcal{D}_{\text{MinNF}}$.4) and must be substituted. In particular, such $\downarrow$-arguments may be among the arguments $\varepsilon_{i,2}, \dots, \varepsilon_{i,m-1}$, which, in an earlier iteration, have become arguments of the outermost operator $\uparrow$. Successive substitutions bring down the nesting level of the brackets to at most 3. As an illustration, we revisit the DNA expressions from Example 11, for which the recursive function `MakeMinimalNF` appeared to use quadratic time.

Example 13. Let α be an arbitrary $\mathcal{N}$-word, and let

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

As we observed in Example 11, for $p \geq 1$, both E_{2p} and E_{2p+1} are minimal. The starting DNA expression E_1 is also minimal. We can thus apply algorithm `NormalizeMinimal` to these DNA expressions. For $q \geq 1$, E_q is alternating but its first argument is $\langle \updownarrow \alpha \rangle$. Hence, lines 7–9 of the algorithm are not applicable. We examine the effect of the while-loop on an $\uparrow$-expression E_{2p} for $p \geq 2$:

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

The third argument of E_{2p} is the $\downarrow$-expression E_{2p-1}, which has in turn as an argument the $\uparrow$-expression $E_{2(p-1)}$. According to line 14, E_{2p-1} is substituted in E by

$$\langle \downarrow \langle \updownarrow \alpha \rangle \, \alpha \, \langle \updownarrow \alpha \rangle \rangle \, \alpha E_{2(p-1)-1} \alpha \, \langle \downarrow \langle \updownarrow \alpha \rangle \, \alpha \, \langle \updownarrow \alpha \rangle \rangle \,,$$

yielding

$$E \;=\; \langle \uparrow \langle \updownarrow \alpha \rangle \, \alpha \, \langle \downarrow \langle \updownarrow \alpha \rangle \, \alpha \, \langle \updownarrow \alpha \rangle \rangle \, \alpha E_{2(p-1)-1} \alpha \, \langle \downarrow \langle \updownarrow \alpha \rangle \, \alpha \, \langle \updownarrow \alpha \rangle \rangle \, \alpha \, \langle \updownarrow \alpha \rangle \rangle \,.$$

Now, the fifth argument of E is the $\downarrow$-expression $E_{2(p-1)-1}$. If $p \geq 3$, then this $\downarrow$-expression has as an argument the $\uparrow$-expression $E_{2(p-2)}$. According to line 14, $E_{2(p-1)-1}$ is substituted in E by

$$\langle\downarrow \langle\updownarrow \alpha\rangle\, \alpha\, \langle\updownarrow \alpha\rangle\rangle\, \alpha E_{2(p-2)-1}\alpha\, \langle\downarrow \langle\updownarrow \alpha\rangle\, \alpha\, \langle\updownarrow \alpha\rangle\rangle\, .$$

In $p-1$ substitutions, we obtain the DNA expression E'_{2p} from (10), which is in minimal normal form. For each substitution, we perform a constant amount of work: remove one occurrence of $\uparrow$, add one occurrence of $\downarrow$ and rearrange two brackets. Hence, the total amount of work (and time) to get from E_{2p} to E'_{2p} is linear in p, and thus linear in $|E_{2p}|$.

The effect of the while-loop on the $\downarrow$-expressions E_{2p+1} is analogous. ■

It is instructive to examine two differences between the operation of MakeMinimalNF and that of NormalizeMinimal, for the DNA expressions from Examples 11 and 13. First, due to its recursive set-up, MakeMinimalNF rewrites the DNA expressions from the inside outwards: first E_3, then E_4, then E_5, etc. NormalizeMinimal, on the other hand, rewrites the DNA expression from the outside inwards.

The second difference concerns the nesting level of the brackets of the DNA expression. In every step of MakeMinimalNF, we rewrite a DNA subexpression with nesting level 4 (which is not in normal form) into a DNA subexpression with nesting level 3 (which is in normal form). Hence, the nesting level of the overall DNA expression decreases by 1. On the other hand, every time we execute line 14 of NormalizeMinimal, the nesting level of the DNA expression decreases by 2. [1] It is not the factor 2 itself which makes NormalizeMinimal more efficient. However, because we make two steps at a time, we no longer introduce operators and brackets in one step, which we have to remove in the next step, as we do with MakeMinimalNF.

For the DNA expressions in Examples 12 and 13, algorithm NormalizeMinimal correctly produces the equivalent DNA expressions in minimal normal form. This is the case for *arbitrary* minimal DNA expressions:

Theorem 14. For each minimal DNA expression E_2^*, the algorithm NormalizeMinimal produces an equivalent DNA expression E_3^* in minimal normal form.

Proof: In lines 4 and 5 of NormalizeMinimal, we have a (minimal) $\updownarrow$-expression E_2^*. By Theorem 1, E_2^* must be of the form $\langle\updownarrow \alpha_1\rangle$ for an $\mathcal{N}$-word α_1. In this case, by Definition 8(1), E_2^* is in minimal normal form already, and there is no need to rewrite it.

In line 6, we have a (minimal) $\uparrow$-expression $E = E_2^*$. By Properties ($\mathcal{D}_{\text{Min}}$.1) and ($\mathcal{D}_{\text{Min}}$.2) from Theorem 6, each argument of E is either an $\mathcal{N}$-word α, or an $\updownarrow$-expression $\langle\updownarrow \alpha\rangle$, or a $\downarrow$-argument.

If E does not have Property ($\mathcal{D}_{\text{MinNF}}$.5), then E must be alternating and its first argument must be a $\downarrow$-argument $\varepsilon_1 = \langle\downarrow \varepsilon_{1,1} \dots \varepsilon_{1,m}\rangle$. In lines 7–9, we deal with this case. By Property ($\mathcal{D}_{\text{Min}}$.5), the first argument $\varepsilon_{1,1}$ of ε_1 is either an $\mathcal{N}$-word α or an $\updownarrow$-expression $\langle\updownarrow \alpha\rangle$.

When we apply (the analogue for $\uparrow$-expressions of) procedure RotateToMinimal to E, line RtM.6 is applicable, and the result is an equivalent, minimal $\downarrow$-expression. The first argument of this $\downarrow$-expression is $\varepsilon_{1,1}$, i.e., an $\mathcal{N}$-word α or an $\updownarrow$-expression $\langle\updownarrow \alpha\rangle$. Hence, E has acquired Property ($\mathcal{D}_{\text{MinNF}}$.5).

Let us assume that after lines 7–9, E is an $\uparrow$-expression. We argue that the following property is an invariant for the while-loop:

[1]There is one exception. If the original DNA expression was a $\downarrow$-expression E_{2p+1} for some $p \geq 1$, then the final rewriting step yields a decrease of the nesting level by only 1.

E is a minimal $\uparrow$-expression with Property ($\mathcal{D}_{\text{MinNF}}.5$), satisfying $E \equiv E_2^*$, $\widehat{\varepsilon}$ is an argument of E, and the arguments to the left of $\widehat{\varepsilon}$ do not contain any occurrence of $\uparrow$.

The property is obviously true before the while-loop, when $\widehat{\varepsilon}$ is the first argument of E. Assume the invariant holds before a certain iteration of the loop.

We first consider the case that $\widehat{\varepsilon}$ is a $\downarrow$-expression with at least one $\uparrow$-argument. Then let $\widehat{\varepsilon} = \langle\downarrow \varepsilon_1 \dots \varepsilon_n\rangle$ for some $n \geq 1$ and arguments $\varepsilon_1, \dots, \varepsilon_n$, and let $\varepsilon_i = \langle\uparrow \varepsilon_{i,1} \dots \varepsilon_{i,m}\rangle$ for some $m \geq 1$ and arguments $\varepsilon_{i,1}, \dots, \varepsilon_{i,m}$ be the first $\uparrow$-argument of $\widehat{\varepsilon}$. By Property ($\mathcal{D}_{\text{Min}}.4$), both $\widehat{\varepsilon}$ and ε_i are alternating and nick free. By Property ($\mathcal{D}_{\text{Min}}.3$), $m \geq 2$ and the substution in line 14 is well defined. We substitute $\widehat{\varepsilon}$ by the sequence of arguments $\langle\downarrow \varepsilon_1 \dots \varepsilon_{i-1}\varepsilon_{i,1}\rangle\, \varepsilon_{i,2} \dots \varepsilon_{i,m-1}\, \langle\downarrow \varepsilon_{i,m}\varepsilon_{i+1} \dots \varepsilon_n\rangle$.

We can use the definition of (the semantics of) a DNA expression to prove that the overall string E is still a DNA expression, and in particular an $\uparrow$-expression, after the substitution. Moreover, $\mathcal{S}(E)$ has not changed. Even the nick letters in $\mathcal{S}(E)$ (if any) are the same before and after the substitution, because both the original argument $\widehat{\varepsilon}$ and the new sequence of arguments are nick free.[2] The length of E has not changed, either, so E is still minimal.

We can also prove that E has consecutive expression-arguments before the substitution, if and only if it has such arguments afterwards. This implies that E still has Property ($\mathcal{D}_{\text{MinNF}}.5$), after the substitution.

The choice of ε_i as the *first* $\uparrow$-argument of $\widehat{\varepsilon}$ ensures that the new $\downarrow$-argument $\langle\downarrow \varepsilon_1 \dots \varepsilon_{i-1}\varepsilon_{i,1}\rangle$ of E does not contain any occurrence of $\uparrow$. Hence, if we set $\widehat{\varepsilon}$ to $\varepsilon_{i,2}$ after the substitution, as we do in line 15, the arguments to the left of $\widehat{\varepsilon}$ still do not contain $\uparrow$.

We conclude that the invariant remains valid in the case that $\widehat{\varepsilon}$ is a $\downarrow$-expression with at least one $\uparrow$-argument. If $\widehat{\varepsilon}$ is not such an argument, then Properties ($\mathcal{D}_{\text{Min}}.1$) and ($\mathcal{D}_{\text{Min}}.2$) imply that $\widehat{\varepsilon}$ does not contain any occurrence of $\uparrow$. When we execute line 17, the invariant remains valid.

In every iteration of the while-loop (except the last one, when we set the variable `stop` to true), $\widehat{\varepsilon}$ moves to the right. This guarantees that the loop terminates.

After the loop, the only remaining occurrence of $\uparrow$ in E is the outermost operator. E also has Property ($\mathcal{D}_{\text{MinNF}}.4$) now, and thus is in minimal normal form. The semantics were not changed during the algorithm. □

To carry out `NormalizeMinimal` efficiently, we use a data structure with the first, the second and the fourth feature we described in Section 5 of [4]: (1) we store (the letters of) a DNA expression E in a doubly-linked list, (2) 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, and (4) for each occurrence of $\uparrow$ or $\downarrow$ in E, we maintain a circular, doubly-linked list of its consecutive expression-arguments. This data structure can be initialized in linear time, and for each substitution performed by the algorithm, it can be updated in constant time. We then have:

Theorem 15. For each minimal DNA expression E_2^*, both the time and the space required by the algorithm `NormalizeMinimal` are linear in $|E_2^*|$.

As we have seen in Example 13, there exist minimal DNA expressions for which `NormalizeMinimal` performs a linear number of rewriting steps. Then obviously, the time complexity cannot be better than linear.

[2]The substitution in line 14 of `NormalizeMinimal` is almost the reverse of line RtM.5 of `RotateToMinimal`. This explains why we use the same type of arguments to prove that the operations do not affect the semantics, see the proof of Lemma 15 in [4].

Proof: Let E_2^* be an arbitrary minimal DNA expression. It is not hard to see that the data structure we propose requires space that is linear in $|E_2^*|$. Intuitively, the linear time complexity is not surprising, either, because in the while-loop in the algorithm, we simply traverse the DNA expression from left to right.

More formally, we first note that the instructions outside the while-loop require constant time. In particular, for line 7, we need to know whether or not E is alternating. This is the case, if and only if the list of consecutive expression-arguments of the outermost operator of E is empty. As the fourth feature of our data structure provides this list, we can check this in constant time.

To analyse the time requirements of the while-loop, we observe, just like we did in the proof of Theorem 14, that in every iteration except the last one, $\widehat{\varepsilon}$ moves to the right. The list of arguments to the left of $\widehat{\varepsilon}$ grows with one argument. Let us use $\widehat{\varepsilon}_j$ to denote the argument that is added in iteration j. We examine the time spent in this iteration. This time depends on (the type of) the argument $\widehat{\varepsilon}$ that we consider in the iteration.

We first assume that $\widehat{\varepsilon}$ is a $\downarrow$-expression. In line 13, we check if $\widehat{\varepsilon}$ has at least one $\uparrow$-argument, and if so, then we need its first $\uparrow$-argument for the substitution in line 14. A natural implementation of this consists of checking the arguments of $\widehat{\varepsilon}$ from left to right, until we find an $\uparrow$-argument or reach the end of $\widehat{\varepsilon}$. Thanks to the first two features of the data structure, this costs only constant time per argument. When we check i arguments of $\widehat{\varepsilon}$ for this, we spend time that is linear in i.

If we find an $\uparrow$-argument ε_i in $\widehat{\varepsilon}$, then we perform lines 14 and 15. We can do this in constant time, which does not affect the complexity. In this case, $\widehat{\varepsilon}_j = \langle\downarrow \varepsilon_1 \ldots \varepsilon_{i-1}\varepsilon_{i,1}\rangle$. If, on the other hand, we do not find an $\uparrow$-argument, then $i = n$ and $\widehat{\varepsilon}_j = \widehat{\varepsilon} = \langle\downarrow \varepsilon_1 \ldots \varepsilon_i\rangle$. In both cases, $|\widehat{\varepsilon}_j| \geq i + 3$. Then obviously, the time spent in the iteration (which is linear in i) is at most linear in $|\widehat{\varepsilon}_j|$.

If $\widehat{\varepsilon}$ is not a $\downarrow$-expression, then $\widehat{\varepsilon}_j = \widehat{\varepsilon}$. In this case, the iteration costs only constant time, which is obviously at most linear in $|\widehat{\varepsilon}_j|$.

We conclude that the time required by the while-loop as a whole is at most linear in the length of the list of arguments of the resulting DNA expression E_3^*, and thus in $|E_3^*| = |E_2^*|$ itself. Then the same goes for the time required by the entire algorithm `NormalizeMinimal`. □

Recall that `NormalizeMinimal` is the second step of our algorithm to rewrite an arbitrary DNA expression E_1^* into the minimal normal form, where the first step consists of applying the recursive function `MakeMinimal` to E_1^*. As the DNA expression E_2^* resulting from this first step is an equivalent, minimal DNA expression, it certainly satisfies $|E_2^*| \leq |E_1^*|$. Then we can combine Theorem 20 from [4] and Theorem 15, to determine the complexity of the total two-step algorithm:

Theorem 16. For each DNA expression E_1^*, both the time and the space required by the two-step algorithm to rewrite E_1^* into the minimal normal form are linear in $|E_1^*|$.

This implies that the two-step algorithm is better than the naive, single-pass recursive function `Make-MinimalNF`. That function, if worked out in detail, would also yield the normal form version of its input. However, as we have seen in Example 11, it requires at least quadratic time in the worst case.

6. Conclusions and directions for future research

We have introduced a (minimal) normal form for DNA expressions. This normal form is characterized by five syntactic properties, which are easy to check. We have described a two-step algorithm, which computes the normal form version of a given DNA expression. This is useful, e.g., to decide if two DNA expressions are equivalent. The algorithm first determines a minimal DNA expression that is equivalent to its input, and then rewrites this minimal DNA expression into the normal form. The algorithm is elegant, because it does not refer to the semantics of the DNA expression involved. It consists of string manipulations on the DNA expression itself. The algorithm requires linear time and space.

An important research line for the future could be to define and analyse new types of DNA expressions. These should be based on operators that directly model operations that are performed on real-world DNA. With new operators, one might also be able to represent DNA molecules with other 'imperfections' than nicks and gaps, e.g., DNA molecules with hairpin loops. It would certainly be a challenge to define DNA expressions that not only *denote* DNA molecules, but also implicitly describe how to synthesize them from the basic elements A, C, G and T.

Acknowledgement

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

References

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

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

[3] 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/`

[4] R. van Vliet, H.J. Hoogeboom: Making DNA expressions minimal, *Fundamenta Informaticae*, **123**, 2013, 199-226.

[5] 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.

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