Universal Computation with Watson-Crick D0L Systems

Petr Sosík

Institute of Computer Science, Silesian University, Bezručovo náměstí 13, 746 01 Opava, Czech Republic, e-mail:petr.sosik@fpf.slu.cz

Technical report No. 2000/1

Abstract

Watson-Crick D0L systems, introduced in 1997 by V. Mihalache and A. Salomaa, arise from two major principles: the Lindenmayer rewriting and the Watson-Crick complementarity principle. Complementarity can be viewed as a purely language-theoretic operation. Majority of a certain type of symbols in a string (purines vs. pyrimidines) triggers a transition to the complementary string. The paper deals with an expressive power of deterministic interactionless Watson-Crick Lindenmayer systems. A rather surprising result is obtained: these systems, consisting of iterated morphism and a basic DNA operation, are alone able to express any Turing computable function.

1 Introduction

Since the time of the Adleman's celebrated experiment [1], a number of theoretical studies concerning various ideas of universal DNA computing have been published. For an overview see [7] or http://www.wi.leidenuniv.nl/˜pier/dna.bib. Nevertheless, the idea of creating a practical synthetic computing system based on macromolecules still remains elusive due to some unrealistic properties of these theoretical models, outrunning the progress in biotechnology. On the other hand, large amount of work was done in study of mathematical foundations of DNA computing principles, with results hopefully applicable also in other areas.

In this paper we demonstrate the power of the DNA complementarity principle, one of two main sources of the DNA computing power. During the process of rewriting genetic information between DNA and RNA, each symbol of the "natural" DNA alphabet A, C, G, T is assigned to its complementary counterpart. In the operational sense, we can describe the complementation principle as a *Watson-Crick morphism* h_W :

$$h_W(A) = T, \quad h_W(T) = A \quad h_W(C) = G, \quad h_W(G) = C.$$

The Watson-Crick complementarity principle has among others a close connection to the powerful *twin-shuffle language TS* [7, 9]. Unfortunatelly, the properties of *TS* or reversed *TS* cannot be used in this paper since they are not *HD0L* languages.

The massive parallelism of DNA operations, the second source of the DNA computing strength, is also naturally present in our model since the morphisms operate all the symbols in strings in parallel.

We show that the universal computational power can be reached with a very simple machinery of a D0L morphism and a generalized Watson-Crick morphism, connected by a natural deterministic context-free condition.

2 Watson-Crick D0L schemes

For elements of formal language theory we refer to [8, 11]. Here we only briefly fix some notation. For a finite alphabet Σ, denote $(\Sigma, \cdot)$ a free monoid with the catenation operation and the empty word λ. For $a \in \Sigma$, $w \in \Sigma^*$, $|w|_a$ is the number of occurrences of a in w. For $\Gamma \subseteq \Sigma$, $|w|_\Gamma = \sum_{a \in \Gamma} |w|_a$. For $w \in \Sigma^*$ we denote w^n the catenation of n copies of w for $n \geq 1$.

A concept of Watson-Crick alphabet is a straightforward generalization of notions of "natural" DNA alphabet consisting of symbols $\{A, C, G, T\}$.

A *DNA-like alphabet* Σ is an alphabet with an even cardinality $2n$, $n \geq 1$, where the letters are enumerated as follows:

$$\Sigma = \{a_1, \ldots, a_n, \overline{a_1}, \ldots, \overline{a_n}\}.$$

We say that a_i and $\overline{a_i}$ are *complementary letters.* The letter to letter endomorphism h_W of Σ^* mapping each letter to the complementary letter is called the *Watson-Crick morphism.* Hence

$$h_W(a_i) = \overline{a_i},\ h_w(\overline{a_i}) = a_i, \quad 1 \leq i \leq n.$$

In analogy with the DNA alphabet we call the non-barred letters *purines* and the barred letters *pyrimidines.* The subset of Σ^* consisting of all words, where the number of occurrences of pyrimidines is strictly greater than that of purines is denoted by *PYR.* The complement of *PYR* is denoted by *PUR.* Clearly, both *PYR* and *PUR* are context-free nonregular languages. We further denote the sets $\Sigma^{\mathrm{PUR}} = \{a_1, \ldots, a_n\}$ and $\Sigma^{\mathrm{PYR}} = \{\overline{a_1}, \ldots, \overline{a_n}\}$. For a set $\Delta \subseteq \Sigma$, we denote $h_W(\Delta) = \{h_W(a) \mid a \in \Delta\}$.

Definition 2.1 *A* Watson-Crick D0L scheme *(or shortly* scheme*) is a construct $G = (\Sigma, p)$, where $\Sigma = \{a_1, \ldots, a_n, \overline{a_1}, \ldots, \overline{a_n}\}$, $p : \Sigma^* \longrightarrow \Sigma^*$ is a morphism. Given a word $w_0 \in \Sigma^*$, the* derivation sequence *$S(G, w_0)$ defined by G from w_0 consists of the words $w_0, w_1, w_2, \ldots$, where for $i \geq 0$,*

$$w_{i+1} = \begin{cases} p(w_i), & \text{if } p(w_i) \in PUR \\ h_W(p(w_i)), & \text{if } p(w_i) \in PYR. \end{cases}$$

The transition $w_i \Longrightarrow_G w_{i+1}$ is also called the derivation step of G. If $w_{i+1} = h_W(p(w_i))$, then we speak about complementation derivation step. We denote $\Longrightarrow_G^*$ the transitive and reflexive closure of $\Longrightarrow_G$ as usual.

For more details and motivation underlying the concept of Watson-Crick D0L systems see [5, 6, 9, 10] (where a Watson-Crick D0L system differs from scheme only by adding an axiom $w_0 \in PUR$). In [5] and others, a Watson-Crick D0L system is viewed also as a DT0L system with two morphisms p and h_W, together with a regulation mechanism guiding the selection of the morphism. Contrary to DT0L system, the regulation mechanism gives rise to determinism and the system generates a unique sequence of words. Another important difference is in fact that *the length of the derivation can be determined by the triggering mechanism*, as the further examples show. This is the main principle of the below proofs showing universality of Watson-Crick D0L schemes.

3 Partial recursive functions

In this section we briefly resume for further use a famous characterization of partial recursive functions in a machine-independent way. For more details we refer to [12, 3].

The symbol $\mathbb{N}$ is used for the set of all nonnegative integers. We denote (partial) recursive functions the class of functions $f : \mathbb{N}^t \longrightarrow \mathbb{N}$ for some $t \geq 0$, computable by Turing machines. The word "partial" can be omitted if the domain of f is $\mathbb{N}^t$. The below notation can be easily extended to functions $\mathbb{N}^t \longrightarrow \mathbb{N}^s$, $s > 0$.

Definition 3.1 *The family of primitive recursive functions is the smallest family of integer-to-integer functions with the following properties:*

(i) It contains the following base functions:

0	*(nullary constant),*
$S(x) = x + 1$	*(successor function),*
$U_i^n(x_1, \ldots, x_n) = x_i$	*(projection functions), for* $1 \leq i \leq n$.

(ii) It is closed under the following operations:

- composition: *if* $h : \mathbb{N}^m \longrightarrow \mathbb{N}$, $g_1 : \mathbb{N}^n \longrightarrow \mathbb{N}$, $\ldots$, $g_m : \mathbb{N}^n \longrightarrow \mathbb{N}$ *are primitive recursive functions, then so is the function* $f : \mathbb{N}^n \longrightarrow \mathbb{N}$ *defined as follows:*

$$f(x_1, \ldots, x_n) = h(g_1(x_1, \ldots, x_n), \ldots, g_m(x_1, \ldots, x_n)).$$

- primitive recursion: *if* $h : \mathbb{N}^n \longrightarrow \mathbb{N}$, $g : \mathbb{N}^{n+2} \longrightarrow \mathbb{N}$ *are primitive recursive functions, then so is the function* $f : \mathbb{N}^{n+1} \longrightarrow \mathbb{N}$ *defined as follows:*

$$\begin{aligned} f(0, x_1, \ldots, x_n) &= h(x_1, \ldots, x_n), \\ f(z+1, x_1, \ldots, x_n) &= g(z, f(z, x_1, \ldots, x_n), x_1, \ldots, x_n), \ z \geq 0. \end{aligned} \tag{1}$$

Theorem 3.2 *The family of partial recursive functions is the smallest family of integer-to-integer functions with the following properties:*

(i) It contains the nullary constant, the successor function and the projection functions.

(ii) It is closed under the operations composition, primitive recursion and minimalization, *defined as follows:*

If $h : \mathbb{N}^{n+1} \longrightarrow \mathbb{N}$ *is a partial recursive function, then so is the function* $f : \mathbb{N}^n \longrightarrow \mathbb{N}$, *where*

$$f(x_1, \ldots, x_n) = \min\{y \in \mathbb{N} \,|\, h(x_1, \ldots, x_n, y) = 0\}, \tag{2}$$

and $h(x_1, \ldots, x_n, z)$ *is defined for all integers* z, $0 \leq z \leq y$. *Otherwise,* $f(x_1, \ldots, x_n)$ *is undefined.*

Kleene's theorem claims existence of an universal function.

Theorem 3.3 (Kleene) *There exist primitive recursive functions* g *and* h *such that for each partial recursive function* f *of one variable there is an integer* i_f *such that*

$$f(x) = g(\min\{y \mid h(x, i_f, y) = 0\}).$$

4 Computation by Watson-Crick D0L schemes

In this section we define a representation of functions suitable for computing with Watson-Crick D0L schemes. Our approach is different from D0L growth functions which are frequently used in the cited literature. It is rather similar to the representation used with Turing machine. A substantial difference is the usage of different symbols for each function argument, since D0L schemes cannot distinguish order of symbols in a string as Turing machines do.

In the rest of the paper, we represent an n-tuple of nonnegative numbers by the Parikh vector of a string of symbols. Notice that both this representation and the derivation in a Watson-Crick D0L scheme is independent on the order of symbols in a string. The following definitions utilize this fact to simplify notation.

Definition 4.1 *Let* $G = (\Sigma, p)$ *be a Watson-Crick D0L scheme. For* $x, y \in \Sigma^*$ *we write*

$$x \stackrel{\cdot}{\Longrightarrow}_G y \quad \text{iff} \quad x \Longrightarrow_G y',$$

where y' *is an arbitrary symbol permutation of* y. *We denote* $\stackrel{\cdot}{\Longrightarrow}_G^*$ *the transitive and reflexive closure of* $\stackrel{\cdot}{\Longrightarrow}_G$ *as usual.*

Definition 4.2 *Let* $f : \mathbb{N}^n \longrightarrow \mathbb{N}^m$ *be a (partial) function and let* $G = (\Sigma, p)$ *be a Watson-Crick D0L scheme. We say that* G *computes* f *if* Σ *contains subsets* $\Gamma = \{\$, A_{[1]}, \ldots, A_{[n]}\}$ *and* $\Delta = \{\#, Z_{[1]}, \ldots, Z_{[m]}\}$ *such that the following holds:*

- $f(x_1, \ldots, x_n) = (y_1, \ldots, y_m)$ *iff* $\$A_{[1]}^{x_1} \ldots A_{[n]}^{x_n} \stackrel{\cdot}{\Longrightarrow}{}^*_G \# Z_{[1]}^{y_1} \ldots Z_{[m]}^{y_m}$,
- $p(X) = X$ *for* $X \in \Delta$,
- *either* $|w|_\Delta = 0$ *or* $w \in \Delta^*$ *for each string* w *in the derivation sequence* $S(G, \$A_{[1]}^{x_1} \ldots A_{[n]}^{x_n})$.

The sets Γ and Δ are called the *input set* and the *output set,* respectively. The last condition requires that no symbol in Δ appears in the derivation sequence until the computation is finished and the output string is derived.

Notice that we can assume without loss of generality that different schemes can have disjoint alphabets, simply by renaming the symbols of Γ and Δ.

It immediately follows from the above definition that a scheme computing a partial function $f : \mathbb{N}^n \longrightarrow \mathbb{N}^m$ never produces any output symbol if $f(x_1, \ldots, x_n)$ is undefined for a certain n-tuple $x_1, \ldots, x_n$. It also follows that the result of computation is independent on order of symbols in the input string.

Example 4.3 *The following Watson-Crick D0L scheme* $G = (\Sigma, p)$ *computes the function* $f(n) = n^2$.

$$\Sigma = \{\$, A_{[1]}, D, E, S, T, Z_{[1]}, \#, \overline{\$}, \overline{A_{[1]}}, \overline{D}, \overline{E}, \overline{S}, \overline{T}, \overline{Z_{[1]}}, \overline{\#}\},$$

$$\begin{array}{llll} p(\$) = \overline{ST} & p(\overline{S}) = \overline{ST} & p(T) = \lambda & p(A_{[1]}) = A_{[1]}D\overline{E} \\ p(S) = \# & p(\overline{D}) = \lambda & p(E) = Z_{[1]} & p(\overline{A_{[1]}}) = \lambda \end{array}$$

and $p(X) = X$ *for all other* $X \in \Sigma$. *The following derivations show how complementation can be used to regulate the computation: starting with* n *input symbols* $A_{[1]}$, *at each step* n *pairs* $D\overline{E}$ *are produced. Simultaneously the number of* $\overline{T}$*'s is incremented. After* n *such a steps the number of* $\overline{T}$*'s reaches the number of* $A_{[1]}$*'s and the complementation occurs, resulting in erasing all the auxiliary symbols and rewriting* E*'s to the output symbols* $Z_{[1]}$.

$$\$A_{[1]}^3 \stackrel{\cdot}{\Longrightarrow} \overline{ST}A_{[1]}^3 D^3 \overline{E}^3 \stackrel{\cdot}{\Longrightarrow} \overline{ST}^2 A_{[1]}^3 D^6 \overline{E}^6 \stackrel{\cdot}{\Longrightarrow} ST^3 \overline{A_{[1]}}^3 \overline{D}^9 E^9 \stackrel{\cdot}{\Longrightarrow} \# Z_{[1]}^9$$

Example 4.4 *The following Watson-Crick D0L scheme computes the function* $f(n) = \lceil \log_3 n \rceil$, *where* $\lceil x \rceil$ *is the ceiling of* x. *Denote* $G = (\Sigma, p)$, *where*

$$\Sigma = \{\$, A, B, C, D, S, Z, \#, \overline{\$}, \overline{A}, \overline{B}, \overline{C}, \overline{D}, \overline{S}, \overline{Z}, \overline{\#}\},$$

$$\begin{array}{llll} p(\$) = \overline{SB} & p(S) = \# & p(B) = \lambda & p(\overline{S}) = \overline{S}C\overline{D} \\ p(\overline{C}) = \lambda & p(\overline{A_{[1]}}) = \lambda & p(\overline{B}) = \overline{BBB} & p(D) = Z_{[1]} \end{array}$$

and $p(X) = X$ *for all other* $X \in \Sigma$. *At the first step, the symbols* $\overline{S}$ *and* $\overline{B}$ *are generated, while* $A_{[1]}$*'s representing the value of* n *remain unchanged. Then at each step the number of* $\overline{B}$*'s is triplicated, and a pair* $C\overline{D}$ *is added. When the number of* $\overline{B}$*'s reaches the number of* $A_{[1]}$*'s, the complementary transition occurs, resulting in rewritting* D *to the output symbol* $Z_{[1]}$. *For example,*

$$\begin{aligned} \$A_{[1]}^9 &\Longrightarrow \overline{SB}A_{[1]}^9 \Longrightarrow \overline{S}C\overline{DB}^3 A_{[1]}^9 \Longrightarrow \overline{S}(C\overline{D})^2 \overline{B}^3 A_{[1]}^9 \Longrightarrow \\ &\Longrightarrow S(\overline{C}D)^3 B^9 \overline{A_{[1]}}^9 \Longrightarrow \# Z_{[1]}^3 \end{aligned}$$

Notice again that contrary to DT0L system, the number of output symbols is nonlinear with respect to the number of input symbols, which is allowed by the complementarity triggering mechanism.

5 Main result

We show that any partial recursive function can be computed by a Watson-Crick D0L scheme. The proof is based on the fact that the base functions and operations in Theorem 3.2 can be realized by Watson-Crick D0L schemes. To simplify the proofs, we decompose these operations into even simpler parts, which are first showed to be realizable by Watson-Crick D0L schemes.

Definition 5.1 *Let $G = (\Sigma, p)$ be a Watson-Crick D0L scheme with an input set Γ and an output set Δ. G is said to be* k-lazy *if for each axiom $w_0 \in \Gamma^*$ and $q \geq 0$, $p(w_q) \in PYR$ implies $p(w_{q+i}) \in PUR$ and $w_{q+i} \notin \Delta^*$, $1 \leq i \leq k$.*

Informally, each complementation step is followed by at least k non-complementation steps during which G does not produce an output. This property will turn out important for merging several schemes into one, as following lemmata do. It follows from the above definition that if G never performs a complementation step during a derivation of $S(G, w_0)$, then G is k-lazy for any $k \geq 0$.

Definition 5.2 *A (partial) function f is said to be* W_k-computable, *if there is a k-lazy Watson-Crick D0L scheme computing f. f is called to be* W-computable, *if it is W_k computable for some $k \geq 0$.*

Lemma 5.3 *Let $f : \mathbb{N}^n \longrightarrow \mathbb{N}^m$ be a W_i-computable function, $i \geq 0$. Let $\ell \geq n$, $k \leq m$, let P and Q be arbitrary permutations over the sets $\{1, \ldots, \ell\}$ and $\{1, \ldots, m\}$, respectively. Then the following function $g : \mathbb{N}^\ell \longrightarrow \mathbb{N}^k$ is W_i-computable:*

$$g(x_{P(1)}, \ldots, x_{P(\ell)}) = (y_{Q(1)}, \ldots, y_{Q(k)}) \text{ iff } f(x_1, \ldots, x_n) = (y_1, \ldots, y_m).$$

Proof. Denote $G = (\Sigma, p)$ a scheme computing f, with the input set $\Gamma = \{\$, A_{[1]}, \ldots, A_{[n]}\}$ and the output set $\Delta = \{\#, Z_{[1]}, \ldots, Z_{[m]}\}$. Consider the scheme $G' = (\Sigma', p')$ with the input set $\Gamma' = \{\$', A'_{[1]}, \ldots, A'_{[\ell]}\}$ and the output set $\Delta' = \{\#', Z'_{[1]}, \ldots, Z'_{[k]}\}$, where $\Sigma' = \Sigma \cup \Gamma' \cup \Delta' \cup h_W(\Gamma' \cup \Delta')$,

$$p'(\$') = \$ \qquad p'(A'_{[P(j)]}) = \begin{cases} A_{[j]} & 1 \leq j \leq n, \\ \lambda & n < j \leq \ell, \end{cases}$$
$$p'(\#) = \#' \qquad p'(Z_{[j]}) = \begin{cases} Z'_{[Q(j)]} & 1 \leq j \leq k, \\ \lambda & k < j \leq m, \end{cases}$$

$p'(X) = p(X)$ for $X \in (\Sigma - \Delta)$ and $p'(X) = X$ for all other $X \in \Sigma'$. Then obviously G' computes g since the only difference between G and G' is changing the order of arguments and function results, and omitting some of them. $\square$

Lemma 5.4 *Let functions* $f : \mathbb{N}^n \longrightarrow \mathbb{N}^m$ *and* $g : \mathbb{N}^m \longrightarrow \mathbb{N}^\ell$ *be* W_i *and* W_j*-computable, respectively. Then the function* $h : \mathbb{N}^n \longrightarrow \mathbb{N}^\ell$, $h(x_1, \dots, x_n) = g \circ f(x_1, \dots, x_n)$, *is* W_k*-computable, where* $k = \min(i, j)$,

Proof. It is enough to join the alphabets and morphisms of the schemes computing f and g, respectively, and to change the resulting morphism so that output symbols of the scheme for f are rewriten to input symbols of the scheme for g.

Formally, denote $G_f = (\Sigma_f, p_f)$ and $G_g = (\Sigma_g, p_g)$. Assume that $\Sigma_f \cap \Sigma_g = \emptyset$. Denote

$$\begin{array}{rcl} \Gamma_f & = & \{\$_f, A_{f,[1]}, \dots, A_{f,[n]}\}, \\ \Delta_f & = & \{\#_f, Z_{f,[1]}, \dots, Z_{f,[m]}\}, \\ \Gamma_g & = & \{\$_g, A_{g,[1]}, \dots, A_{g,[m]}\}, \\ \Delta_g & = & \{\#_g, Z_{g,[1]}, \dots, Z_{g,[\ell]}\}. \end{array}$$

Let p be a morphism over $\Sigma_f \cup \Sigma_g$ defined as follows:

$$\begin{array}{rcll} p(\#_f) & = & \$_g, & \\ p(Z_{f,[i]}) & = & A_{g,[i]}, & 1 \leq i \leq m, \\ p(X) & = & p_f(X), & X \in \Sigma_f - \Delta_f, \\ p(X) & = & p_g(X), & X \in \Sigma_g. \end{array}$$

Then obviously the scheme $(\Sigma_f \cup \Sigma_g, p)$ with the input set Γ_f and the output set Δ_g computes the function h. □

The following two lemmata show that it is possible to "call" one function from another via Watson-Crick D0L schemes, having saved an actual state of computation of the calling function.

Lemma 5.5 *Let* $f : \mathbb{N}^n \longrightarrow \mathbb{N}$ *be a function computed by a* k*-lazy Watson-Crick D0L scheme* F, $k \geq 1$, *with an input set* $\Gamma = \{\$, A_{[1]}, \dots, A_{[n]}\}$ *and an output set* $\Delta = \{\#, Z\}$. *Then there is a* k*-lazy Watson-Crick D0L scheme* $G_1 = (\Sigma_1, p_1)$ *such that* $y = f(x_1, \dots, x_n)$ *iff*

$$\$' A'^{x_1}_{[1]} \dots A'^{x_n}_{[n]} \stackrel{\cdot}{\Longrightarrow}^{+}_{G_1} \# Z^y (B_{[1]}\overline{C_{[1]}})^{x_1} \dots (B_{[n]}\overline{C_{[n]}})^{x_n},$$

where $\$', A'_{[1]}, \dots, A'_{[n]}, B_{[1]}, \dots, B_{[n]}, \overline{C_{[1]}}, \dots, \overline{C_{[n]}} \in \Sigma_1$.

Proof. Denote $F = (\Sigma, p)$, and let

$$\Sigma_1^{\mathrm{PUR}} = \Sigma^{\mathrm{PUR}} \cup \{\$', A'_{[j]}, B_{[j]}, C_{[j]}, C_{[j]i} \mid 1 \leq j \leq n,\ 1 \leq i \leq k\},$$

the sets on the the right hand side being mutually disjoint. Let $\Sigma_1 = \Sigma_1^{\mathrm{PUR}} \cup h_W(\Sigma_1^{\mathrm{PUR}})$.

Let $p_1(X) = p(X)$ for each $X \in \Sigma$. Consider an arbitrary input string $w_0 = \$A^{x_1}_{[1]} \dots A^{x_n}_{[n]}$ and denote $w_0, w_1, w_2, \dots$ members of the sequence $S(G, w_0)$. Below we define elements of p_1 and simultaneously show their effect on the derived string.

Permutation of derived strings	Elements of p_1
$\$' A'^{x_1}_{[1]} \dots A'^{x_n}_{[n]}$	
$\Downarrow \cdot$	$p_1(\$') = \$,\ p_1(A'_{[j]}) = A_{[j]}B_{[j]}\overline{C_{[j]}}$, $1 \leq j \leq n$
$w_0(B_{[1]}\overline{C_{[1]}})^{x_1} \dots (B_{[n]}\overline{C_{[n]}})^{x_n}$	
$\Downarrow \cdot$	$p_1(B_{[j]}) = B_{[j]},\ p_1(\overline{C_{[j]}}) = \overline{C_{[j]}}$, $1 \leq j \leq n$
$w_1(B_{[1]}\overline{C_{[1]}})^{x_1} \dots (B_{[n]}\overline{C_{[n]}})^{x_n}$	
$\vdots$	

Denote for brevity

$$\alpha = (B_{[1]}\overline{C_{[1]}})^{x_1} \dots (B_{[n]}\overline{C_{[n]}})^{x_n}. \tag{3}$$

Consider now that $p(w_r) \in \mathrm{PYR}$ for some $r \geq 0$ and notice that $p_1(w_r\alpha) \in PYR$ *iff* $p(w_r) \in PYR$. Then G_1 performs a complementation step and its behavior will be the following:

Permutation of derived strings	Elements of p_1
$\vdots$	
$w_r\alpha$	
$\Downarrow \cdot$	$p_1(\alpha) = \alpha$, see above
$w_{r+1}h(\alpha)$	
$\Downarrow \cdot$	$p_1(\overline{B_{[j]}}) = \lambda,\ p_1(C_{[j]}) = B_{(j)}\overline{C_{[j]1}}$, $1 \leq j \leq n$
$w_{r+2}(B_{[1]}\overline{C_{[1]1}})^{x_1} \dots (B_{[n]}\overline{C_{[n]1}})^{x_n}$	
$\Downarrow \cdot_*$	$p_1(\overline{C_{[j]i}}) = \overline{C_{[j]i+1}}$, $1 \leq j \leq n,\ 1 < i \leq k$
$w_{r+k+1}(B_{[1]}\overline{C_{[1]k}})^{x_1} \dots (B_{[n]}\overline{C_{[n]k}})^{x_n}$	
$\Downarrow \cdot$	$p_1(\overline{C_{[j]k}}) = \overline{C_{[j]}}$, $1 \leq j \leq n$
$w_{r+k+2}\,\alpha$	for $p(w_{r+k+1}) \in PUR$, or
$w_{r+k+2}\,h(\alpha)$	for $p(w_{r+k+1}) \in PYR$.

These strings contain the same symbols as the previously derived $w_r\alpha$ or $w_{r+1}h(\alpha)$, respectively, and the derivation will continue as above. The above derivation holds since F is k-lazy by assumption and hence $p(w_{r+i}) \in PUR$, $1 \leq i \leq k$. All yet undefined elements of p_1 adopt the form $p_1(X) = X$, $X \in \Sigma_1$.

Now consider that $w_q = \#Z^y \in \Delta^*$ for some $q \geq 1$. Then we know that a complementation did not occurr in the last k steps, since F is k-lazy. The corresponding string derived by G_1 adopts the form $w_q\beta$, where

$$\beta = \begin{cases} (B_{[1]}\overline{C_{[1]k}})^{x_1} \dots (B_{[n]}\overline{C_{[n]k}})^{x_n}, & \text{if a complementation occurred just } k \text{ steps ago,} \\ \alpha & \text{otherwise.} \end{cases} \tag{4}$$

In both cases $\#Z^y\beta \overset{\cdot}{\Longrightarrow}_{G_1} \#Z^y\alpha$ since $\#Z^y\beta \in PUR$, and hence a permutation of the string

$$\#Z^y(B_{[1]}\overline{C_{[1]}})^{x_1} \dots (B_{[n]}\overline{C_{[n]}})^{x_n}$$

is produced by G_1 as the lemma claimed. □

Lemma 5.6 *Let $f : \mathbb{N}^n \longrightarrow \mathbb{N}$ be a W_k computable function, $k \geq 1$. Then the function $g : \mathbb{N}^n \longrightarrow \mathbb{N}^{n+1}$, $g(x_1, \dots, x_n) = (x_1, \dots, x_n, f(x_1, \dots, x_n))$, is W_{k-1}-computable.*

Proof. Proof of Lemma 5.5 showed that there is a scheme G_1 such that $y = f(x_1, \dots, x_n)$ *iff* $\$'A'^{x_1}_{[1]} \dots A'^{x_n}_{[n]} \overset{\cdot}{\Longrightarrow}^+_{G_1} \#Z^y\beta$, where $\#, Z$ did not appear in previous members of the sequence, and moreover

$$p_1(\#Z^y\beta) = \#Z^y\alpha = \#Z^y(B_{[1]}\overline{C_{[1]}})^{x_1} \dots (B_{[n]}\overline{C_{[n]}})^{x_n},$$

with α, β as in (3), (4). Nevertheless, the symbols $B_{[j]}$, $\overline{C_{[j]}}$, $1 \leq j \leq n$, could appear in previous members of the sequence. Now it remains to construct a scheme G_2 which rewrites all the symbols of $\#Z^y\beta$ to its output symbols which did not appear in the sequence yet.

Let $G_2 = (\Sigma_2, p_2)$, denote Γ_2 and Δ_2 the input and the output set of G_2, respectively. Let $\Gamma_2 = \Gamma_1$, $\Delta_2 = \{\#', Z'_{[1]}, \dots, Z'_{[n+1]}\}$. Let

$$\Sigma_2^{\mathrm{PUR}} = \Sigma_1^{\mathrm{PUR}} \cup \Delta_2 \cup \{S_i, D_i \mid 0 \leq i \leq 2k-1\} \cup \{C_{[j]i} \mid k < i \leq 2k-1,\ 1 \leq j \leq n\},$$

the sets on the the right hand side being mutually disjoint. Let $\Sigma_2 = \Sigma_2^{\mathrm{PUR}} \cup h_W(\Sigma_2^{\mathrm{PUR}})$.

Let $p_2(X) = p_1(X)$ for each $X \in \Sigma_1 - \Delta - \{C_{[j]k} \mid 1 \leq j \leq n\}$. Hence the derivation of G_2 is identical to that of G_1 until the string $\#Z^y\beta$ is produced, since the symbols $C_{[j]k}$ never appeared during this derivation.

Having the string $\#Z^y\beta$, G_2 rewrites the symbols $\#$, Z to new symbols and performs a complementation step to rewrite β to $h(\alpha)$ (see the above proof for elements of p_1). Then $k-1$ non-complementation steps follow to keep the scheme $(k-1)$-lazy.

Permutation of derived strings	Elements of p_2
$\#Z^y\beta$	
$\Downarrow \cdot$	$\boxed{p_2(\#) = \overline{S_0},\ p_2(Z) = \overline{D_0}}$

$S_0 D_0^y h(\alpha)$	
$\Downarrow \cdot_*$	$\boxed{\begin{array}{c} p_2(S_i) = S_{i+1},\ p_2(D_i) = D_{i+1} \\ 0 \le i < k-1 \end{array}}$
$S_{k-1} D_{k-1}^y (B_{[1]} \overline{C_{[1]k-1}})^{x_1} \ldots (B_{[n]} \overline{C_{[n]k-1}})^{x_n}$	
$\Downarrow \cdot$	$\boxed{p_2(S_{k-1}) = \overline{S_k},\ p_2(D_{k-1}) = \overline{D_k}}$
$S_k D_k^y (\overline{B_{[1]}} C_{[1]k})^{x_1} \ldots (\overline{B_{[n]}} C_{[n]k})^{x_n}$	

In the last step another complementation occurred, rewriting the symbols $\overline{C_{[j]k}}$ in an other way than G_1 would do. Notice that this complementation occurs only when the symbols S_{k-1}, D_{k-1} appear in the sequence, which is possible only when a computation of $y = f(x_1, \ldots, x_n)$ is finished. Until then, the derivation was fully controlled by elements of p_1.

Permutation of derived strings	Elements of p_2
$\vdots$	
$\Downarrow \cdot_*$	$\boxed{\begin{array}{c} p_2(S_i) = S_{i+1},\ p_2(D_i) = D_{i+1} \\ p_2(C_{[j]i}) = C_{[j]i+1} \\ 1 \le j \le n,\ k \le i < 2k-1 \end{array}}$
$S_{2k-1} D_{2k-1}^y C_{[1]2k-1}^{x_1} \cdots C_{[n]2k-1}^{x_n}$	
$\Downarrow \cdot$	$\boxed{\begin{array}{c} p_2(S_{2k-1}) = \#', \\ p_2(D_{2k-1}) = Z'_{[n+1]}, \\ p_2(C_{[j]2k-1}) = Z'_{[j]},\ 1 \le j \le n \end{array}}$
$\#' Z'^{x_1}_{[1]} \ldots Z'^{x_n}_{[n]} Z'^{y}_{[n+1]}$	

The last $k-1$ non-complementation steps was again to keep the scheme $(k-1)$-lazy. All yet undefined elements of p_2 adopt the form $p_2(X) = X,\ X \in \Sigma_2$. It follows immediately from the above description that G_2 is $(k-1)$-lazy and computes g, since the symbols from Δ_2 did not appear in the derivation sequence until the output string was produced. $\square$

Note. Consider that in the above lemma the function f is W_k-computable for any $k \ge 0$, and hence the scheme computing f never performs a complementation step. On the contrary, the resulting scheme G_2 computing g *does* perform a complementation. Hence an arbitrary but fixed lazy value k must be chosen, according to which the construction of G_2 is done, and the function g is then W_{k-1} computable. The same holds also for Lemma 5.7.

For the next lemma we fix the following notation: let $f : \mathbb{N}^n \longrightarrow \mathbb{N}^n$ be a (partial) function. Denoting $\circ$ the composition operation, we define

$$\begin{aligned} f^0(x_1, \ldots, x_n) &= (x_1, \ldots, x_n), \\ f^k(x_1, \ldots, x_n) &= f \circ f^{k-1}(x_1, \ldots, x_n),\ k > 0. \end{aligned}$$

Lemma 5.7 *Let $f : \mathbb{N}^n \longrightarrow \mathbb{N}^n$, $n \geq 2$, be a W_k-computable function, $k \geq 1$. Then so is the function $g : \mathbb{N}^n \longrightarrow \mathbb{N}^n$, defined as*

$$g(x_1, \dots, x_n) = f^i(x_1, \dots, x_n), \quad i = \min\{\ell \mid f^\ell(x_1, \dots, x_n) = (y_1, \dots, y_n),\ y_1 \geq y_2\},$$

if such an i exists, otherwise $g(x_1, \dots, x_n)$ is undefined.

Proof. Denote $F = (\Sigma, p)$ a scheme computing f with an input set $\Gamma = \{\$, A_{[1]}, \dots, A_{[n]}\}$ and an output set $\Delta = \{\#, Z_{[1]}, \dots, Z_{[n]}\}$. Let $G = (\Sigma', p')$ be a scheme with an input set $\Gamma' = \{\$', A'_{[1]}, \dots, A'_{[n]}\}$ and an output set $\Delta' = \{\#', Z'_{[1]}, \dots, Z'_{[n]}\}$. Let

$$\Sigma'^{\mathrm{PUR}} = \Sigma^{\mathrm{PUR}} \cup \Gamma' \cup \Delta' \cup \{S, B_{[j]}, C_{[j]}, E_{[j]i} \mid 1 \leq i \leq k,\ 1 \leq j \leq n\},$$

the sets on the right hand side being mutually disjoint. Let $\Sigma' = \Sigma'^{\mathrm{PUR}} \cup h_W(\Sigma'^{\mathrm{PUR}})$. Let $p'(X) = p(X)$ for each $X \in \Sigma - \Delta$.

We start the derivation with a test whether $x_1 \geq x_2$. If not, we calculate $(y_1, \dots, y_n) := f(x_1, \dots, x_n)$ and repeat from the beginning until $y_1 \geq y_2$. Consider first that $x_1 < x_2$.

Permutation of derived strings	Elements of p'
$\$' A'^{x_1}_{[1]} \dots A'^{x_n}_{[n]}$	
$\Downarrow \cdot$	$\boxed{\begin{array}{c} p'(\$') = \overline{S},\ p'(A'_{[1]}) = \overline{B_{[1]}} \\ p'(A'_{[2]}) = C_{[2]},\ p'(A'_{[j]}) = \overline{B_{[j]}} C_{[j]} \\ 2 < j \leq n \end{array}}$
$\overline{S}\,\overline{B_{[1]}}^{x_1} C_{[2]}^{x_2} (\overline{B_{[3]}} C_{[3]})^{x_3} \dots (\overline{B_{[n]}} C_{[n]})^{x_n}$	
$\Downarrow \cdot$	$\boxed{\begin{array}{c} p'(\overline{S}) = \$,\ p'(\overline{B_{[1]}}) = A_{[1]} \\ p'(\overline{B_{[j]}}) = \lambda,\ p'(C_{[j]}) = A_{[j]} \\ 2 \leq j \leq n \end{array}}$
$\$ A^{x_1}_{[1]} \dots A^{x_n}_{[n]}$	
$\Downarrow \cdot_*$	Elements of p will apply during this derivation.
$\# Z^{y_1}_{[1]} \dots Z^{y_n}_{[n]}$	
$\Downarrow \cdot$	$\boxed{\begin{array}{c} p'(\#) = \$',\ p'(Z_{[j]}) = A'_{[j]} \\ 1 \leq j \leq n \end{array}}$
$\$' A'^{y_1}_{[1]} \dots A'^{y_n}_{[n]}$	

At this moment the derived string consists again of input symbols and the derivation is repeated analogously. Consider now that $y_1 \geq y_2$ holds. Then in the first step of the above derivation a complementation occurs and the current values of $(y_1, \dots, y_n)$ are transformed to an output string:

Permutation of derived strings	Elements of p'
$\vdots$	
$\Downarrow \cdot$	
$SB_{[1]}^{y_1}\overline{C_{[2]}}^{y_2}(B_{[3]}\overline{C_{[3]}})^{y_3}\dots(B_{[n]}\overline{C_{[n]}})^{y_n}$	
$\Downarrow \cdot$	$\boxed{\begin{array}{c} p'(S)=S_1,\ p'(\overline{C_{[2]}})=E_{[2]1} \\ p'(\overline{C_{[j]}})=\lambda,\ 2<j\leq n \\ p'(B_{[j]})=E_{[j]1},\ 1\leq j\leq n \end{array}}$
$S_1E_{[1]1}^{y_1}\dots E_{[n]1}^{y_n}$	
$\Downarrow \cdot_*$	$\boxed{\begin{array}{c} p'(S_i)=S_{i+1},\ p'(E_{[j]i})=E_{[j]i+1} \\ 1\leq j\leq n,\ 1\leq i<k \end{array}}$
$S_kE_{[1]k}^{y_1}\dots E_{[n]k}^{y_n}$	
$\Downarrow \cdot$	$\boxed{\begin{array}{c} p'(S_k)=\#',\ p'(E_{[j]k})=Z'_{[j]} \\ 1\leq j\leq n \end{array}}$
$\#'Z_{[1]}'^{y_1}\dots Z_{[n]}'^{y_n}$	

The last k steps are to keep the scheme k-lazy. It follows from the above description that the output of the scheme is $(y_1,\dots,y_n)=f^i(x_1,\dots,x_n)$, where $i\geq 0$ is minimal such that $y_1\geq y_2$.

Notice that if no such i exists for a certain n-tuple $x_1,\dots,x_n$, then the scheme G never produces an output symbol, which corresponds to Definition 4.2 since $g(x_1,\dots,x_n)$ is undefined. □

Lemma 5.8 *The following functions are W_k-computable for any $k\geq 0$: (i) the nullary constant 0, (ii) the successor function $S(x)$, (iii) the projection function $U_i^n(x_1,\dots,x_n)$.*

Proof.

(i) Consider the scheme $G=(\{\$,A,\#,Z,\overline{\$},\overline{A},\overline{\#},\overline{Z}\},p)$ such that $p(\$)=\#$, $p(A)=\lambda$ and $p(X)=X$ for all other $X\in\Sigma$.

(ii) Consider the scheme $G=(\{\$,\#\,A,Z,\overline{\$},\overline{\#},\overline{A},\overline{Z}\},p)$ such that $p(\$)=\#Z$, $p(A)=Z$ and $p(X)=X$ for all other $X\in\Sigma$.

(iii) Consider the scheme $G=(\{\$,\#\,A_{[1]},\dots,A_{[n]},Z,\overline{\$},\overline{\#},\overline{A_{[1]}},\dots,\overline{A_{[n]}},\overline{Z}\},p,\$)$ such that $p(\$)=\#$, $p(A_{[i]})=Z$, $p(A_{[j]})=\lambda$ for $j\neq i$ and $p(X)=X$ for all other $X\in\Sigma$.

□

Lemma 5.9 *Let $h:\mathbb{N}^m\longrightarrow\mathbb{N}$, $g_1:\mathbb{N}^n\longrightarrow\mathbb{N}$, $\dots$, $g_m:\mathbb{N}^n\longrightarrow\mathbb{N}$ be W_i-computable functions, $i\geq 1$. Then the function $f:\mathbb{N}^n\longrightarrow\mathbb{N}$, $f(x_1,\dots,x_n)=h(g_1(x_1,\dots,x_n),\dots,g_m(x_1,\dots,x_n))$, is W_{i-1}-computable.*

Proof. We can derive according to previous lemmata that the following functions are W_{i-1}-computable:

- $f_1 : \mathbb{N}^n \longrightarrow \mathbb{N}^{n+1}$, defined as $f_1(x_1, \ldots, x_n) = (x_1, \ldots, x_n, y_1)$, where $y_1 = g_1(x_1, \ldots, x_n)$, due to Lemma 5.6;

- $f_2(x_1, \ldots, x_n) = (x_1, \ldots, x_n, y_1, y_2)$, where $y_2 = g_2(x_1, \ldots, x_n)$, since

  - $g_2'(x_1, \ldots, x_n, y_1) = g_2(x_1, \ldots, x_n)$ is W_k-computable due to Lemma 5.3,
  - $g_2''(x_1, \ldots, x_n, y_1) = (x_1, \ldots, x_n, y_1, g_2'(x_1, \ldots, x_n, y_1))$ is W_{k-1}-computable due to Lemma 5.6, and
  - $f_2(x_1, \ldots, x_n) = g_2'' \circ f_1(x_1, \ldots, x_n)$ is W_{k-1}-computable due to Lemma 5.4;

- $f_3(x_1, \ldots, x_n) = (x_1, \ldots, x_n, y_1, y_2, y_3)$, where $y_3 = g_3(x_1, \ldots, x_n)$, due to Lemmata 5.3, 5.4 and 5.6 as in the construction of f_2 above;

$$\vdots$$

- $f_m(x_1, \ldots, x_n) = (x_1, \ldots, x_n, y_1, \ldots, y_m)$, where $y_m = g_m(x_1, \ldots, x_n)$, analogously as in the construction of f_2 above;

- $f_{m+1}(x_1, \ldots, x_n) = (x_1, \ldots, x_n, y_1, \ldots, y_m, h(y_1, \ldots, y_m))$ as in the construction of f_2 above;

- $f(x_1, \ldots, x_n) = U^{n+m+1}_{n+m+1} \circ f_{m+1}(x_1, \ldots, x_n)$ due to Lemmata 5.4 and 5.8 (iii).

We constructed the function f using the constructions of the Lemmata 5.3, 5.4, 5.6 and 5.8 in a straightforward manner, hence f is W_{i-1}-computable function. □

Lemma 5.10 *Let $h : \mathbb{N}^n \longrightarrow \mathbb{N}$, $g : \mathbb{N}^{n+2} \longrightarrow \mathbb{N}$ be W_i-computable functions, $i \geq 1$. Then the function $f : \mathbb{N}^{n+1} \longrightarrow \mathbb{N}$ defined as in (1) is W_{i-1}-computable.*

Proof. The following functions are W_{i-1}-computable:

- $S'(y, z, w, x_1, \ldots, x_n) = S(y)$ due to Lemmata 5.3 and 5.8 (ii);

- $f_1 : \mathbb{N}^{n+3} \longrightarrow \mathbb{N}^{n+4}$, defined as $f_1(y, z, w, x_1, \ldots, x_n) = (y, z, w, x_1, \ldots, x_n, y_1)$, where $y_1 = S'(y, z, w, x_1, \ldots, x_n)$, due to Lemma calling2;

- $f_2(y, z, w, x_1, \ldots, x_n) = (y, z, w, x_1, \ldots, x_n, y_1, y_2)$, where $y_2 = g(y, w, x_1, \ldots, x_n)$, analogously to the construction of f_2 in the above proof;

- $f_3(y, z, w, x_1, \ldots, x_n) = (y_1, z, y_2, x_1, \ldots, x_n)$ due to Lemma 5.3;

- $f_4(y, z, w, x_1, \ldots, x_n) = f_3^k(y, z, w, x_1, \ldots, x_n)$, where $k \geq 0$ is minimal such that $y \geq z$, due to Lemma 5.7.

Consider $y = 0$, then the calculation of $f_4(y, z, w, x_1, \ldots, x_n)$ will proceed as follows: $w := g(y, w, x_1, \ldots, x_n)$, $y := S(y)$, repeatedly while $y < z$, i.e. z-times. If we at the beginning assume $w = h(x_1, \ldots, x_n)$, then value of w is computed exactly according to (1) in Definition 3.1 of the primitive recursion.

Notice that if g and h are defined for all values of arguments, then so is f_4 since the computation according to Lemma 5.7 finishes after z cycles. It remains to show that the following functions are W_{i-1}-computable, establishing w as the value computed by the function f :

- $h_1(z, x_1, \ldots, x_n) = h(x_1, \ldots, x_n)$ due to Lemma 5.3;
- $h_2(z, x_1, \ldots, x_n) = (z, x_1, \ldots, x_n, h_1(z, x_1, \ldots, x_n))$ due to Lemma 5.6;
- $h_3(z, x_1, \ldots, x_n) = (z, x_1, \ldots, x_n, h_1(z, x_1, \ldots, x_n), 0)$ analogously to the construction of f_2 in the above proof, having 0 the nullary function;
- $h_4(z, x_1, \ldots, x_n) = (0, z, h(x_1, \ldots, x_n), x_1, \ldots, x_n)$ due to Lemma 5.3;
- $f_5(z, x_1, \ldots, x_n) = f_4 \circ h_4(z, x_1, \ldots, x_n)$ due to Lemma 5.4;
- $f(z, x_1, \ldots, x_n) = U_3^{n+3} \circ f_5(z, x_1, \ldots, x_n)$ due to Lemmata 5.4 and 5.8 (iii).

$\square$

Lemma 5.11 *Let $h : \mathbb{N}^{n+1} \longrightarrow \mathbb{N}$ be a W_i-computable function, $i \geq 1$. Then the function $f : \mathbb{N}^n \longrightarrow \mathbb{N}$, $f(x_1, \ldots, x_n) = \min\{y \in \mathbb{N} \,|\, h(x_1, \ldots, x_n, y) = 0\}$ is W_{i-1}-computable.*

Proof. The following functions are W_{i-1}-computable:

- $f_1(v, w, x_1, \ldots, x_n, y, z) = (v, h(x_1, \ldots, x_n, y), x_1, \ldots, x_n, S(y), y)$, analogously to the construction of f_1 in the proof of Lemma 5.10;
- $f_2(v, w, x_1, \ldots, x_n, y, z) = f_1^k(v, w, x_1, \ldots, x_n, y, z)$, where $k \geq 0$ is minimal such that $v \geq w$, due to Lemma 5.7.

Consider that we compute f_2 with $v = y = z = 0$, $w = 1$, then its values will be calculated as follows: $w := h(x_1, \ldots, x_n, y)$, $y := S(y)$, $z := y$, repeatedly until $h(x_1, \ldots, x_n, y) = 0$. Taking as function value the value of y in the previous step (which is stored in z at that moment), we obtain exactly the value in 2 where the minimalization is defined.

Notice that if no $y \in \mathbb{N}$ exists such that $h(x_1, \ldots, x_n, y) = 0$, or $h(x_1, \ldots, x_n, z)$ is undefined for some $z < y$, then $f(x_1, \ldots, x_n)$ is undefined and the computation of f according to Lemma 5.7 never produces an output symbol.

It remains to show that the following functions are W_{i-1}-computable:

- $f_3(x_1, \ldots, x_n) = f_2(0, 1, x_1, \ldots, x_n, 0, 0)$, due to Lemmata 5.3, 5.4, 5.6 and 5.8 (i),(ii), analogously to the construction of $h_1, \ldots, h_4$ and f_5 in the above proof;
- $f(x_1, \ldots, x_n) = U_{n+4}^{n+4} \circ f_3(x_1, \ldots, x_n)$ due to Lemmata 5.4 and 5.8 (iii).

□

Theorem 5.12 *Any (partial) recursive function is W-computable.*

Proof. Follows from Theorem 3.2 and from Lemmata 5.8 – 5.11. Notice that a necessary condition for Lemmata 5.9 – 5.11 is that all the functions that are subject to these operations must be W_i-computable, $i \geq 1$. Taking into account that the result of each elementary operation is then W_{i-1} computable, we can ask what happens when the value 0 is reached.

Fortunately, when constructing any (partial) recursive function, we always start with base functions which are W_i computable for each i, and then we apply only a fixed number of the operations. This number is of course independent on the function arguments. During the first application of these operation over the base functions, a construction according to Lemma 5.6 is applied and an arbitrary but fixed lazy value must be chosen (see also note following Lemma 5.6). Hence it is enough to choose this value larger than the number of the operations further applied, so that the lazy value 0 is never reached. □

Corollary 5.13 *The universal function is W-computable.*

6 Conclusions

We studied the properties of Watson-Crick complementation in the framework of deterministic Lindenmayer systems. We obtained an universal computational power only by enhancing the D0L iterated morphism by a complementation mechanism, which is one of the basic DNA operations. This unexpected result may be inspiring for both biological and computational research in molecular computing. There may be also further consequences of the result in the theory of formal languages and automata.

A computer experiments were also performed to study the properties of Watson-Crick D0L schemes computing various functions. It turned out, however, that for more complex functions the number of necessary symbols in the resulting Watson-Crick D0L schemes can be enormously large.

7 Acknowledgements

I am grateful to Arto Salomaa for inspiration, useful remarks and discussions about this paper, to Erzsebet Csuhaj-Varjú for her invitation to Sztaki, where I found the primary idea and to Alica Kelemenová and Jozef Kelemen for continuous support. Research was supported by the Grant Agency of Czech Republic, grants No. 201/98/P047 and 201/99/1086.

References

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

[2] J. Dassow, G. Păun, *Regulated Rewriting in Formal Language Theory* (Springer-Verlag, Berlin, 1989).

[3] J. Gruska, *Foundations of Computing* (International Thomson Computer Press, London, 1997).

[4] S. Kleene, General recursive functions on natural numbers, *Mathematische Annalen* **112** (1936) 727–742.

[5] V. Mihalache, A. Salomaa, Language-theoretic aspects of DNA complementarity, *Theoretical Computer Science* **250** (2001) 163-178.

[6] V. Mihalache, A. Salomaa, Lindenmayer and DNA: Watson-Crick D0L systems, *EATCS Bulletin* **62** (1997) 160–175.

[7] G. Păun, G. Rozenberg, A. Salomaa, *DNA Computing. New Computing Paradigms* (Springer-Verlag, Berlin, 1998).

[8] G. Rozenberg, A. Salomaa, *The Mathematical Theory of L systems* (Academic Press, New York, 1980).

[9] A. Salomaa, Turing, Watson-Crick and Lindenmayer. Aspects of DNA complementarity, in C.S. Calude, J. Casti, M.J. Dinneen, eds., *Unconventional Models of Computation* (Springer-Verlag, Berlin, 1998) 94–107.

[10] A. Salomaa, Watson-Crick walks and roads in D0L graphs, *Acta Cybernetica* **14** (1999) 179–192.

[11] A. Salomaa, G. Rozenberg, eds., *Handbook of Formal Languages* (Springer-Verlag, Berlin, 1997).

[12] C. H. Smith, *A Recursive Introduction to the Theory of Computation* (Springer-Verlag, Berlin, 1994).

[13] P. Sosík, *On properties of Watson-Crick D0L systems,* TR 1999/1, Institute of Computer Science, Silesian University at Opava, 1999.