Available at
www.ComputerScienceWeb.com
POWERED BY SCIENCE DIRECT®

Theoretical Computer Science 306 (2003) 101–112

Theoretical
Computer Science

www.elsevier.com/locate/tcs

Watson–Crick D0L systems: generative power and undecidable problems

Petr Sosík

Institute of Computer Science, Silesian University, 74601 Opava, Czech Republic

Received 1 November 2001; received in revised form 11 July 2002; accepted 6 March 2003
Communicated by M. Margenstern

Abstract

The properties of Watson–Crick D0L system, a language–theoretical formalism inspired by natural DNA processing, are studied. The model incorporates the iterated D0L-like morphism and the DNA complementarity principle represented by a letter-to-letter morphism. These two morphisms are connected by a natural condition called the trigger.

We show first that this very simple model has rather unexpected power; it can closely and simply simulate any Minsky register machine. As a consequence, any recursively enumerable language can be obtained as a projection of the language of some standard Watson–Crick D0L system. Finally, we show that the graph reachability problem, equivalence problems and some other problems of standard Watson–Crick D0L systems are undecidable.

MSC: 68Q50; 68Q22

Keywords: Lindenmayer system; Watson–Crick complementarity; DNA computing

1. Introduction

The *Watson–Crick D0L system* is a string rewriting system inspired by both Lindenmayer rewriting [10] and DNA computing principles [8], whose attractivity for computer science increased mainly since the time of the Adleman's experiment [1]. The Watson–Crick D0L system was developed to study the properties of the DNA complementarity principle in language–theoretical framework. It turns out that this principle in the operational sense represents simple yet powerful operation over letters and strings, allowing to reach universal computational power (in Turing sense) using a very simple

E-mail address: petr.sosik@fpf.slu.cz (P. Sosík).

0304-3975/03/$ - see front matter
doi:10.1016/S0304-3975(03)00214-7

formalism. The model was suggested in 1997 by Mihalache and Salomaa [5]. For motivation underlying the concept of Watson–Crick D0L systems and some recent results see [4,6,11,12,14].

The basic concept of the model, the Watson–Crick complementation principle, can be expressed as a relation between complementary pairs of nucleotides of the "natural" DNA alphabet A, C, G, T. In the operational sense, we can describe the principle by the *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 *standard Watson–Crick D0L system* consists of a D0L morphism and a generalized Watson–Crick morphism defined over generalized DNA alphabet of an even cardinality, whose elements are grouped into complementary pairs. These two morphisms are triggered by a deterministic context-free condition.

As already shown (in a rather complicated proof) in [14], any partial recursive function can be computed by a standard Watson–Crick D0L system. In this paper following [14] we start with a simple proof of the above universality result using a different technique. We show that any Minsky register machine (or any similar model of programmable computer with integer variables as *while* program, etc.) can be closely simulated by a standard Watson–Crick D0L system, obtaining a very simple deterministic DNA-motivated abstract machine.

As a consequence of the above statement, we then prove that any recursively enumerable (RE) language is a projection of the language generated by some standard Watson–Crick D0L system. This result is related to the family of simple morphic representations of RE languages given, e.g. in [13, Section 8.2.], and also to the results in [2], where EDT0L and E0L Watson–Crick systems are studied. Finally, several open decidability problems reported in [6,12] are proven to be undecidable.

2. Watson–Crick D0L schemes

For elements of formal language theory we refer to [3,10,13]. Here we fix only some notation. For a finite alphabet Σ, denote by $(\Sigma, \cdot)$ 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 by w^n the catenation of n copies of w for $n \geqslant 1$.

The alphabet we use in most of the constructions in this paper is a straightforward generalization of the above notion of the "natural" DNA alphabet.

A *DNA-like alphabet* Σ is an alphabet with an even cardinality $2n$, $n \geqslant 1$, 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*. The coding h_W over Σ^* mapping each letter to the complementary letter is called the *Watson–Crick morphism*. Hence

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

In analogy with the DNA alphabet we call the non-barred letters *purines* and the barred letters *pyrimidines*. We 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 $h_W(\Delta) = \{h_W(a) |\ a \in \Delta\}$.

Now consider that a sequence of words over such a DNA-like alphabet Σ, expressing an organism growth, is generated by a deterministic Lindenmayer system. Under some circumstances which can be interpreted as a non-suitable for further growth, we want to replace the current string by its complement $h_W(w)$. The set of all the words w representing these "non-suitable" circumstances we call the *trigger*. Various kinds of triggers (regular, context-free, context-sensitive) are studied in [4,6,11,12]. We deal in this paper with the *standard trigger* PYR, the set of all words with the number of occurrences of pyrimidines strictly greater than that of purines. The complement of PYR is denoted by PUR. Notice that both PYR and PUR are deterministic context-free non-regular languages.

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

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

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

The Watson–Crick D0L system defined in [6,11] and other citations differs from the Watson–Crick D0L scheme only by adding an axiom $w_0 \in \mathrm{PUR}$. In [6], the Watson–Crick D0L system is compared also with a DT0L system with two morphisms p and h_W and with a regulation mechanism selecting the applied morphism. Contrary to DT0L system, the regulation mechanism here gives rise to determinism and the system generates a unique sequence of words, and it also allows to regulate the length of the derivation.

The sequence of words generated by a Watson–Crick D0L system G defines the language $L(G) = \{w_i \mid w_0 \Rightarrow^* w_i\}$, as well as the *length sequence* $|w_i|$, $i \geqslant 0$, and the *growth function* $f(i) = |w_i|$. It was shown in [6] that Watson–Crick D0L systems can define growth functions which are not $\mathbb{Z}$-rational. Nevertheless, the capacity of defining function can be further extended with the concept of Watson–Crick D0L scheme. The notion of a *function computed* by a Watson–Crick D0L scheme was introduced in [14].

Definition 2.2. Consider a Watson–Crick D0L scheme $G = (\Sigma, p)$. A partial recursive function $f: \mathbb{N} \to \mathbb{N}$ is computed by G if the alphabet Σ contains the letters B, b, E, e with the productions $p(E) = E$ and $p(e) = e$ and satisfying the following condition. For all $i \geqslant 0$, the equation $f(i) = j$ holds exactly in case there is a derivation according

to G

$$Bb^i \Rightarrow^* Ee^j \tag{1}$$

and, moreover, the letters E and e appear in this derivation at the last step only.

It follows from the above definition that if the value of $f(i)$ is undefined for some $i \in \mathbb{N}$, then the string of the form Ee^j, $j \geqslant 0$ never appears in the sequence $S(G, Bb^i)$. For examples of Watson–Crick D0L schemes computing functions we refer to [14].

3. Register machine

In this section, we briefly recall the concept of the Minsky register machine. Minsky showed, e.g. in [7] that universal computational power can be reached by the abstract machine using a finite number of registers for storing arbitrarily large non-negative integers. The machine runs a program consisting of numbered instructions of several simple types. Several variants of the machine with different number of registers and different instruction sets were shown to be computationally universal. The basic instruction types we use here are:

> a' add 1 to the content of the register a and continue with the next instruction
> $a^-(k)$ if the content of the register a is nonzero, then subtract 1 from it and continue with the next instruction, else continue with the kth instruction H halt the machine.

We can assume that H is used only once as the last instruction in each program. Minsky proved in [7, Section 11.2], that any Turing machine can be simulated by a register machine with the above instruction types and five registers. Moreover, one of the registers called **w** contains zero all the time.

It follows from the construction given by Minsky that each partial recursive function f can be computed by the register machine mentioned above, starting with the argument value n and ending with the value $f(n)$ in some register. Let us further assume that if the function $f(n)$ is undefined for some $n \geqslant 0$, then the corresponding register machine never halts with input n.

4. Computation with Watson–Crick D0L schemes

First, we describe informally the representation of integers suitable for simulation of register machines by Watson–Crick D0L schemes. We do not use D0L growth functions which are frequently used in the cited literature to express the connection of D0L systems to integer functions. Our approach here is rather similar to the representation used with Turing machine, but we must use different symbols to distinguish between different registers and different instructions of the register machine program.

Consider a program P consisting of the instructions $I_1, I_2, \ldots, I_n$ which runs on a register machine with the registers $\mathbf{r}_1, \mathbf{r}_2, \ldots, \mathbf{r}_m$, and $\mathbf{w}$, $m \geqslant 1$, and with the instruction set as in Section 3. To represent the program by a Watson–Crick D0L scheme $G = (\Sigma, p)$, the set Σ contains symbols $\#_{j,i}$, $r_{j,i}$, $1 \leqslant j \leqslant n$, $1 \leqslant i \leqslant m$. When we simulate the execution of the instruction I_j, the string derived by G adopts the form $\#_{j,1} r_{j,1}^{x_1} \#_{j,2} r_{j,2}^{x_2} \cdots \#_{j,m} r_{j,m}^{x_m}$, where $x_1, \ldots, x_m$ are the contents of the registers $\mathbf{r}_1, \ldots, \mathbf{r}_m$, respectively.

We need no symbols to represent the content of the register $\mathbf{w}$ as it contains always zero and hence its content is always represented by the empty string according to the above convention. This register is never incremented or decremented and can be used only in the unconditional transfer instruction $\mathbf{w}^-(k)$. We now show that any register machine M with a program P can be simulated by a standard Watson–Crick D0L scheme G such that the values stored in registers at each step of computation of P are represented by numbers of the corresponding symbols in the string derived by G.

Theorem 4.1. *Consider a register machine M with the registers $\mathbf{r}_1, \ldots, \mathbf{r}_m$ and $\mathbf{w}$, $m \geqslant 1$. Then for each program P with n instructions there exists a standard Watson–Crick D0L scheme $G = (\Sigma, p)$, where Σ^{PUR} contains the symbols $\#_{1,i}, r_{1,i}$ and $\#_{n,i}, r_{n,i}$, such that $p(\#_{n,i}) = \#_{n,i}$, $p(r_{n,i}) = r_{n,i}$, $1 \leqslant i \leqslant m$, and the following holds:*

The program P starts with the initial values $x_1, \ldots, x_m$ in $\mathbf{r}_1, \ldots, \mathbf{r}_m$, and halts with the corresponding final values $y_1, \ldots, y_m$, if and only if

$$\#_{1,1} r_{1,1}^{x_1} \cdots \#_{1,m} r_{1,m}^{x_m} \Rightarrow_G^* \#_{n,1} r_{n,1}^{y_1} \cdots \#_{n,m} r_{n,m}^{y_m} \tag{2}$$

and the symbols $\#_{n,i}, r_{n,i}$, $1 \leqslant i \leqslant m$, appear in this derivation at the last step only.

Proof. Let

$$\Sigma = \Sigma_1 \cup \Sigma_2 \cup \cdots \cup \Sigma_n, \quad \Sigma_k \cap \Sigma_j = \emptyset \quad \text{for } 1 \leqslant k \neq j \leqslant n,$$
$$p = p_1 \cup p_2 \cup \cdots \cup p_n, \quad p_k \cap p_j = \emptyset \quad \text{for } 1 \leqslant k \neq j \leqslant n.$$

Let $\mathrm{DOM}(p_j) = \Sigma_j$, $1 \leqslant j \leqslant n$. Each sub-morphism p_j performs the action of the instruction I_j. We adopt the convention that the elements of Σ_j are denoted by the first subscript j. It remains to show the construction of p_j's such that when $x_1, \ldots, x_m$ and $x_1', \ldots, x_m'$ are the contents of the registers $\mathbf{r}_1, \ldots, \mathbf{r}_m$, respectively, before and after the execution of I_j, and the next instruction to be executed is I_k, then

$$\#_{j,1} r_{j,1}^{x_1} \cdots \#_{j,m} r_{j,m}^{x_m} \Rightarrow_G^* \#_{k,1} r_{k,1}^{x_1'} \cdots \#_{k,m} r_{k,m}^{x_m'} \tag{3}$$

and, moreover, only the elements of p_j are used during this derivation.

- Assume that $I_j = r_i'$, $1 \leqslant i \leqslant m$. Then let $\Sigma_j^{\mathrm{PUR}} = \{\#_{j,i}, r_{j,i} \mid 1 \leqslant i \leqslant m\}$ and let

$$p_j(\#_{j,i}) = \#_{j+1,i} r_{j+1,i},$$
$$p_j(\#_{j,\ell}) = \#_{j+1,\ell}, \quad 1 \leqslant \ell \leqslant m, \ \ell \neq i,$$
$$p_j(r_{j,\ell}) = r_{j+1,\ell}, \quad 1 \leqslant \ell \leqslant m$$

and $p_j(a) = a$ for all other $a \in \Sigma_j$.

- Assume that $I_j = r_i^-(k)$, $1 \leqslant i \leqslant m$. Then let $\Sigma_j^{\mathrm{PUR}} = \{\#_{j,i}, a_{j,i}, b_{j,i}, c_{j,i}, d_{j,i}, r_{j,i} \mid 1 \leqslant i \leqslant m\} \cup \{e_j, f_j\}$ and let

$$
\begin{aligned}
&p_j(\#_{j,i}) = \bar{a}_{j,i}, && p_j(a_{j,\ell}) = \#_{k,\ell}, \quad 1 \leqslant \ell \leqslant m,\\
&p_j(\#_{j,\ell}) = \bar{a}_{j,\ell} f_j, \quad 1 \leqslant \ell \leqslant m,\ \ell \neq i, && p_j(\bar{f}_j) = \lambda,\\
&p_j(r_{j,i}) = r_{j,i}, && p_j(\bar{r}_{j,i}) = \lambda,\\
&p_j(r_{j,\ell}) = \bar{c}_{j,\ell} f_j, \quad 1 \leqslant \ell \leqslant m,\ \ell \neq i, && p_j(c_{j,\ell}) = r_{k,\ell}, \quad 1 \leqslant \ell \leqslant m,\ \ell \neq i.
\end{aligned}
$$

  Recall that the derivation starts with the string $\#_{j,1} r_{j,1}^{x_1} \cdots \#_{j,m} r_{j,m}^{x_m}$. If $x_i = 0$, then the first step is the complementation step and the string $\#_{k,1} r_{k,1}^{x_1} \cdots \#_{k,m} r_{k,m}^{x_m}$ is produced. Assume now $x_i > 0$ and let further

$$
\begin{aligned}
&p_j(\bar{a}_{j,i}) = \bar{b}_{j,i}\bar{e}_j, && p_j(\bar{b}_{j,i}) = \bar{b}_{j,i}\bar{d}_{j,i},\\
&p_j(\bar{a}_{j,\ell}) = \bar{b}_{j,\ell}, \quad 1 \leqslant \ell \leqslant m,\ \ell \neq i, && p_j(b_{j,\ell}) = \#_{j+1,\ell}, \quad 1 \leqslant \ell \leqslant m,\\
&p_j(\bar{c}_{j,\ell}) = \bar{d}_{j,\ell}, \quad 1 \leqslant \ell \leqslant m,\ \ell \neq i, && p_j(d_{j,\ell}) = r_{j+1,\ell}, \quad 1 \leqslant \ell \leqslant m
\end{aligned}
$$

  and, moreover, $p_j(e_j) = \lambda$ and $p_j(a) = a$ for all other $a \in \Sigma_j$. Then since the third step the symbols $\bar{d}_{j,i}$ are produced until their number equals $x_i - 1$. At this moment complementation step occurs, rewriting the string to the form $\#_{j+1,1} r_{j+1,1}^{x'_1} \cdots \#_{j+1,m} r_{j+1,m}^{x'_m}$, where $x'_i = x_i - 1$ and $x'_\ell = x_\ell$ for $\ell \neq i$.
- Assume that $I_j = H$. Then let $\Sigma_j^{\mathrm{PUR}} = \{\#_{j,i}, r_{j,i} \mid 1 \leqslant i \leqslant m\}$ and let $p_j(a) = a$ for each $a \in \Sigma_j$.

We showed that relation (3) holds for any considered type of the register machine instruction. Due to the transitivity of $\Rightarrow_G^*$ relation (3) can be extended to an arbitrary sequence of executed instructions of P. Due to the fact that H occurs in P only as its n-th instruction, we can conclude that relation (2) holds for any program P which halts with the input $(x_1, \ldots, x_n)$, on the one hand.

If, on the other hand, the program P never halts with the input $(x_1, \ldots, x_n)$, then the string of the form $\#_{n,1} r_{n,1}^{y_1} \ldots \#_{n,m} r_{n,m}^{y_m}$ never appears in the sequence $S(G, \#_{n,1} r_{n,1}^{x_1} \cdots \#_{n,m} r_{n,m}^{x_m})$. □

Corollary 4.2. *For each partial recursive function $f : \mathbb{N} \to \mathbb{N}$ there exists (effectively) a Watson–Crick D0L scheme G computing f.*

Proof. We can assume without loss of generality that the function f is computed by a program P with n instructions for a register machine with the registers $\mathbf{r}_1, \ldots, \mathbf{r}_m$ and $\mathbf{w}$ such that the input and the output value of P is stored at register $\mathbf{r}_m$, and P never halts with the input i if the value of $f(i)$ is undefined. The initial values of all other registers are zeros.

Let $G = (\Sigma, p)$ be the Watson–Crick D0L scheme simulating P in the sense of Theorem 4.1 and hence satisfying relation (2). Assume that Σ does not contain any

of the symbols B, b, E, e, and consider the scheme $G' = (\Sigma', p')$ such that $\Sigma' = \Sigma \cup \{B, b, E, e\}$,

$$p'(B) = \#_{1,1} \cdots \#_{1,m}, \quad p'(b) = r_{1,m}, \quad p'(\#_{n,m}) = E, \quad p'(r_{n,m}) = e,$$
$$p'(E) = E, \quad p'(e) = e$$

and further $p'(\#_{n,k}) = p'(r_{n,k}) = \lambda$, $1 \leqslant k < m$, and $p'(x) = p(x)$ for all other $x \in \Sigma$. Then clearly the function f is computed by the scheme G. □

Notice that adding an axiom to a Watson–Crick D0L scheme corresponds to fixing the initial register values. Hence, in the sense of Theorem 4.1, each register machine with fixed initial register values can be simulated by some standard Watson–Crick D0L system.

5. Languages of Watson–Crick D0L systems

We show that any recursively enumerable language L (over an alphabet V_L) containing the empty word λ can be obtained as a weak coding of the language of a standard Watson–Crick D0L system. Consider an effective (in Gödel sense) encoding $\psi : V_L^* \to \mathbb{N}$ and define $\psi(L) = \{\psi(w) \mid w \in L\}$. If L is recursively enumerable, then so is $\psi(L)$. Then either $\psi(L)$ is empty, or there exists a total recursive function $f_L : \mathbb{N} \to \mathbb{N}$ such that

$$\psi(L) = \text{range}(f_L). \tag{4}$$

Intuitively, the function f_L corresponds to an effective procedure for listing the members of the set $\psi(L)$ (with repetitions allowed), see e.g. [9] for details. The function f_L can be computed by a standard Watson–Crick D0L scheme G_L due to Theorem 4.1. Hence L can be generated by enumerating the sequence $f_L(0)$, $f_L(1)$, $f_L(2), \ldots$ and by rewriting $f_L(i)$ to $\psi^{-1}(f_L(i))$ for each $i \geqslant 0$.

Theorem 5.1. *For any recursively enumerable language L there is a projection h and a standard Watson–Crick D0L system G such that $h(L(G)) = L \cup \{\lambda\}$.*

Proof. Let L be an RE language over an alphabet $V_L = \{a_1, \ldots, a_k\}$, $k \geqslant 1$. For an arbitrary word $w \in V_L^*$ of the form

$$w = a_{j_1} a_{j_2} \cdots a_{j_{|w|}}, \quad 1 \leqslant j_1, j_2, \ldots, j_{|w|} \leqslant k, \tag{5}$$

we define

$$\psi(w) = j_1(k+1)^0 + j_2(k+1)^1 + \cdots + j_{|w|}(k+1)^{|w|-1}. \tag{6}$$

Assume first that $L \neq \emptyset$ and consider a function f_L satisfying (4). Then there exists a register machine with the registers $\mathbf{w}$ and $\mathbf{r}_1, \ldots \mathbf{r}_m$, $m \leqslant 4$, starting with the value i in $\mathbf{r}_m$ and stopping with the value $f_L(i)$ in $\mathbf{r}_1$, keeping the value i in $\mathbf{r}_m$ for an arbitrary $i \in \mathbb{N}$.

Due to Theorem 4.1 we can find a standard Watson–Crick D0L scheme $G_L=(\Sigma_L, p_L)$ with the symbols $\#_{1,1},\ldots,\#_{1,m}$, $\#_{n,1},\ldots,\#_{n,m}$, s_1, r_n, s_n in Σ_L such that for each $i\geqslant 0$,

$$\#_{1,1}\cdots\#_{1,m}s_1^i \Rightarrow_{G_L}^* \#_{n,1}r_n^{f_L(i)}\#_{n,2}\cdots\#_{n,m}s_n^i. \tag{7}$$

Let $\Sigma_L\cap V_L=\emptyset$. Consider the standard Watson–Crick D0L system $G=(\Sigma, p, \#_{1,1}\cdots\#_{1,m})$, where $(\Sigma_L\cup V_L)\subset\Sigma$ and

$$p(a)=p_L(a) \quad \text{for each } a\in\Sigma_L-\{\#_{n,1},\ldots,\#_{n,m},\bar{r}_n,s_n\}. \tag{8}$$

Hence G performs derivation (7). Due to (4) we have $f_L(i)=\psi(w)$ for some $w\in L$. We show the construction of G such that for each $i\in\mathbb{N}$ and $w\in L$,

$$\#_{n,1}r_n^{\psi(w)}\#_{n,2}\cdots\#_{n,m}s_n^i \Rightarrow_G^* w\#_{1,1}\cdots\#_{1,m}s_1^{i+1} \tag{9}$$

and moreover $p(w)=\lambda$. Let

$$\Sigma^{\mathrm{PUR}}=\Sigma_L^{\mathrm{PUR}}\cup V_L\cup\{A_1,\ldots,A_k\}\cup\{b_1,\ldots,b_k\}\cup\{B_1,\ldots,B_k\}\cup\{\#,C,R,s\}.$$

(i) First, assume $w\neq\lambda$ and hence $\psi(w)>0$. Let $p(\#_{n,\ell})=\lambda$, $1\leqslant\ell\leqslant m-1$, $p(\#_{n,m})=\bar{\#}$ and $p(s_n)=s\bar{C}$, which implies the derivation

$$\#_{n,1}r_n^{\psi(w)}\#_{n,2}\cdots\#_{n,m}s_n^i \Rightarrow_G r_n^{\psi(w)}\bar{\#}(s\bar{C})^i. \tag{10}$$

Let further

$$p(s)=S,\quad p(\bar{\#})=\bar{A}_1\,\bar{C},\quad p(\bar{A}_2)=\bar{A}_3\bar{C},\quad p(\bar{A}_{k-1})=\bar{A}_k\bar{C},\quad p(S)=S,$$
$$p(\bar{C})=\bar{C},\quad p(\bar{A}_1)=\bar{A}_2\bar{C},\quad \vdots\quad p(\bar{A}_k)=\bar{R}\,\bar{\#},\quad p(\bar{R})=\bar{R}.$$

In the following derivation, the symbols $\bar{A}_1,\bar{A}_2,\ldots,\bar{A}_k$ appear subsequently and cyclically in the derived string, and simultaneously the number of pyrimidines is incremented by 1 at each step. It can be easily checked that $\psi(w)$th step is the complementation step and hence

$$r_n^{\psi(w)}\bar{\#}(s\bar{C})^i \Rightarrow_G^* \bar{r}_n^{\psi(w)}R^{\psi(w')}A_{j_1}C^{\psi(w)-\psi(w')}(\bar{S}C)^i, \tag{11}$$

where $\psi(w')=\psi(w)\bmod(k+1)$, and thus due to (5) and (6), $w'=a_{j_2}\cdots a_{j_{|w|}}$. Let us define further

$$p(\bar{r}_n)=\lambda,\quad p(R)=r_n,\quad p(C)=\lambda,\quad p(\bar{S})=s\bar{C},$$

and

$$p(A_j)=b_j\overline{C\#},\quad p(b_j)=B_j,\quad p(B_j)=B_j,\quad p(\bar{B}_j)=b_j\bar{C},\quad 1\leqslant j\leqslant k.$$

If $\psi(w')>0$, then

$$\bar{r}_n^{\psi(w)}R^{\psi(w')}A_{j_1}C^{\psi(w)-\psi(w')}(\bar{S}C)^i \Rightarrow_G r_n^{\psi(w')}b_{j_1}\overline{C\#}(s\bar{C})^i, \tag{12}$$

where the string on the right-hand side is analogous to that on the left-hand side of (11). The two strings differ in the presence of $b_{j_1}\bar{C}$ and in replacement of w with w'.

Hence, a derivation analogous to (11) and (12) starts over. Iterating this derivation, we obtain

$$r_n^{\psi(w)}\bar{\#}(s\bar{C})^i \Rightarrow_G^* r_n^{\psi(w')}b_{j_1}\overline{C\#}(s\bar{C})^i \Rightarrow_G^* r_n^{\psi(w'')}b_{j_1}\bar{C}b_{j_2}\overline{C\#}(s\bar{C})^i \Rightarrow_G^* \cdots \Rightarrow_G^* r_n^{j_{|w|}}b_{j_1}\bar{C}b_{j_2}\bar{C}\cdots b_{j_{|w|-1}}\overline{C\#}(s\bar{C})^i \Rightarrow_G \bar{b}_{j_1}C\bar{b}_{j_2}C\cdots\bar{b}_{j_{|w|}}C\#(\bar{s}C)^i. \tag{13}$$

The last step was the complementation step due to the fact that unlike in (12) no symbols r_n remained in the string.

Finally, let $p(\bar{b}_j)=a_j$, $p(a_j)=\lambda$ for each j, $1\leqslant j\leqslant k$, and let $p(\#)=\#_{1,1}\cdots\#_{1,m}s_1$, $p(\bar{s})=s_1$. These rules, together with (5), (10) and (13), imply exactly the derivation according to (9).

(ii) Second, assume that $\psi(w)=0$ in the left-hand side of (10), then

$$\#_{n,1}r_n^{\psi(w)}\#_{n,2}\cdots\#_{n,m}s_n^i \Rightarrow_G \#(\bar{s}C)^i \Rightarrow_G \#_{1,1}\cdots\#_{1,m}s_1^{i+1},$$

corresponding again to (9) since $w=\lambda$. Notice that no symbol from V_L' had appeared during derivation (9) before its last step. Moreover $p(w)=\lambda$ and hence according to (8),

$$p(w\#_{1,1}\cdots\#_{1,m}s_1^{i+1}) = p_L(\#_{1,1}\cdots\#_{1,m}s_1^{i+1}).$$

We can conclude that iterating derivations (7) and (9), we obtain

$$\#_{1,1}\cdots\#_{1,m}s_1^0 \Rightarrow_G^* w_0\#_{1,1}\cdots\#_{1,m}s_1^1 \Rightarrow_G^* w_1\#_{1,1}\cdots\#_{1,m}s_1^2 \Rightarrow_G^* \cdots,$$

where $w_i=\psi^{-1}(f_L(i))$, $i\geqslant 0$. Considering the projection $h\colon \Sigma\to\Sigma$ such that $h(a)=a$ for $a\in V_L'$ and $h(a)=\lambda$ otherwise, we have $h(L(G))=\{\lambda,w_0,w_1,w_2,\ldots\}=L\cup\{\lambda\}$ due to (4).

Finally, consider the case $L=\emptyset$. Then for an arbitrary Watson–Crick D0L system $G=(\Sigma,p,w_0)$ we define $h(a)=\lambda$, $a\in\Sigma$, and hence again $h(L(G))=\{\lambda\}=L\cup\{\lambda\}$ which concludes the proof. □

If we wanted to obtain also any λ-free RE language as a morphic image of $L(G)$, we would use a partial projection h' instead of h and modify the system G slightly.

Theorem 5.1 completes the results in [2], where both EDT0L and E0L Watson–Crick systems with the standard trigger are shown to generate all recursively enumerable languages. In our case a more powerful filtering mechanism of the projection has to be used instead of the terminal filter. It could be easily seen that no standard Watson–Crick ED0L systems can generate, e.g. the language $\{aa,ab,ba\}$ due to its determinism.

This result is also related to the family of simple morphic representations of RE languages. Among the similar results there is the representation of any RE language as a morphic image of the intersection of a *twin-shuffle* language (which is a context-sensitive language but not ET0L) and a regular language [13]. Comparing both results, Theorem 5.1 can be interpreted as an iterated intersection of a D0L language with the standard trigger (which is the deterministic context-free language).

6. Undecidable problems of Watson–Crick D0L systems

As a direct consequence of Theorems 4.1 and 5.1, several open problems reported in [6,12] are solved.

Letter-appearance problem: Given a Watson–Crick D0L system $G=(\Sigma,g,w_0)$ and a letter $a\in\Sigma$, decide whether or not a appears in some word in the sequence $S(G)$.

Reachability problem for graphs H_G: Given a Watson–Crick D0L system G and a node x in the graph H_G, decide whether or not the Watson–Crick walk $W(H_G)$ passes through x.

For definitions of the graph H_G and the Watson–Crick walk we refer to [6,12].

Theorem 6.1. *The letter-appearance problem and the reachability problem for a given standard Watson–Crick D0L system G are both undecidable.*

Proof. (i) Consider a recursively enumerable language L and let G be the Watson–Crick D0L system constructed according to the proof of Theorem 5.1 so that $L\cup\{\lambda\}=h(L(G))$, h being a projection. The letter # appears in some word in the sequence $S(G)$ *iff* L is not empty language which is undecidable by Rice's theorem.

(ii) Due to Theorem 4 in [6] an algorithm for solving the reachability problem for graphs H_G can be converted to an algorithm for solving the letter appearance problem. □

Stability problem. Given a Watson–Crick D0L system G, decide whether or not the complementarity transition ever takes place in the sequence $S(G)$.

Ultimate stability problem. Given a Watson–Crick D0L system $G=(\Sigma,p,w_0)$, decide whether there is a word w in the sequence $S(G)$ such that the system (Σ,p,w) is stable.

The ultimate stability problem is equivalent to the problem whether or not only finitely many complementarity transitions occur in the sequence $S(G)$.

Theorem 6.2. *The ultimate stability problem for a given standard Watson–Crick D0L system G is undecidable.*

Proof. Consider a program P for the Minsky register machine from Section 3 with fixed initial values of registers, and the standard Watson–Crick D0L system G simulating P with these initial values. It follows from the proof of Theorem 4.1 that a complementarity transition occurs every time when the instruction $a^-(k)$ is executed. Hence, G is ultimately stable *iff* P halts after a finite number of steps which is undecidable. □

We do not know whether or not the stability problem for a given standard Watson–Crick D0L system is decidable. It is known [6] that this problem is algorithmically equivalent to the $\mathbb{Z}_{\text{pos}}$ problem, the long-standing open problem of the theory of integer matrices: given a $\mathbb{Z}$-rational sequence $z(i)$ (by some effective means such as a matrix or two D0L systems), decide whether or not $z(i)\geqslant 0$ holds for all $i\geqslant 0$.

Growth, sequence and language equivalence problem. Given Watson–Crick D0L systems G_1 and G_2, decide whether or not they generate the same growth function, sequence and language, respectively.

Theorem 6.3. *The growth, sequence and language equivalence problem for given standard Watson–Crick D0L systems G_1 and G_2 are all undecidable.*

Proof. Consider a program P for a Minsky register machine with fixed initial register values and with m registers, $m \geqslant 1$, and the corresponding standard Watson–Crick D0L system $G = (\Sigma, p, w_0)$ simulating P due to Theorem 4.1. Then there is the symbol $\#_{n,1} \in \Sigma$ which appears in some string w_j, $j \geqslant 1$, in the sequence $S(G)$ if and only if P reaches the instruction H. Moreover, in such a case $w_j \in PUR$, $p(a) = a$ for each symbol a in w_j, and none of these symbols appeared in w_i for $i < j$. Let us define $G' = (\Sigma, p', w_0)$, where

$$p'(\#_{n,1}) = \lambda, \quad p'(a) = p(a) \quad \text{for all other } a \in \Sigma.$$

Then the growth functions, sequences and languages of G and G' are equivalent *iff* P does not halt which is undecidable. (Notice that the statement holds also in the case $w_j = \#_{n,1}$ due to the fact that empty string never appears in $S(G)$.) □

Acknowledgements

I am grateful to Arto Salomaa for inspiration, to Jiří Wiedermann for useful remarks, and to Jozef Kelemen for continuous support. Research was supported by the Grant Agency of Czech Republic, Grant No. 201/02/P079.

References

[1] L. Adleman, Molecular computation of solutions to combinatorial problems, Science 266 (1994) 1021–1024.
[2] J. Csima, E. Csuhaj Varjú, A. Salomaaa, Power and size of extended Watson–Crick L systems, TUCS Report 424, Turku Centre for Computer Science, Turku, 2001. Theoret. Comput. Sci. 270 (2003) 1665–1678.
[3] J. Dassow, G. Păun, Regulated Rewriting in Formal Language Theory, Springer, Berlin, 1989.
[4] J. Honkala, A. Salomaa, Watson–Crick D0L systems with regular triggers, Theoret. Comput. Sci. 259 (2001) 689–698.
[5] V. Mihalache, A. Salomaa, Lindenmayer and DNA: Watson–Crick D0L systems, EATCS Bull. 62 (1997) 160–175.
[6] V. Mihalache, A. Salomaa, Language–theoretic aspects of DNA complementarity, Theoret. Comput. Sci. 250 (2001) 163–178.
[7] M.L. Minsky, Finite and Infinite Machines, Prentice-Hall, Englewood Cliffs, NJ, 1967.
[8] G. Păun, G. Rozenberg, A. Salomaa, DNA Computing, New Computing Paradigms, Springer, Berlin, 1998.
[9] H. Rogers Jr., Theory of Recursive Functions and Effective Computability, McGraw-Hill, New York, 1967.
[10] G. Rozenberg, A. Salomaa, The Mathematical Theory of L systems, Academic Press, New York, 1980.

[11] 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, Berlin, 1998, pp. 94–107.
[12] A. Salomaa, Watson–Crick walks and roads in D0L graphs, Acta Cybernet. 14 (1999) 179–192.
[13] A. Salomaa, G. Rozenberg, (Eds.), Handbook of Formal Languages, Springer, Berlin, 1997.
[14] P. Sosík, D0L systems + Watson–Crick complement = universal computation, in: M. Margenstern, J. Rogozhin (Eds.), Machines, Computations and Universality, Lecture Notes in Computer Science, Vol. 2055, Springer, Berlin, 2001, pp. 308–320.