<table>
<thead>
<tr>
<th>Title</th>
<th>Computing by networks of standard Watson-Crick $D0L$ systems (Algebraic Systems, Formal Languages and Computations)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Author(s)</td>
<td>Csuhaj-Varju, Erzsebet</td>
</tr>
<tr>
<td>Citation</td>
<td>数理解析研究所講究録 (2000), 1166: 43-51</td>
</tr>
<tr>
<td>Issue Date</td>
<td>2000-08</td>
</tr>
<tr>
<td>URL</td>
<td><a href="http://hdl.handle.net/2433/64357">http://hdl.handle.net/2433/64357</a></td>
</tr>
<tr>
<td>Type</td>
<td>Departmental Bulletin Paper</td>
</tr>
<tr>
<td>Textversion</td>
<td>publisher</td>
</tr>
</tbody>
</table>

Kyoto University
Computing by networks of standard Watson-Crick 
$D0L$ systems

Erzsébet Csuhaý-Varjú
Computer and Automation Research Institute
Hungarian Academy of Sciences
Kende utca 13-17, 1111 Budapest, Hungary
E-mail: csuhay@sztaki.hu

Abstract

Standard Watson-Crick $D0L$ systems ($SWD0L$ systems) are variants of $D0L$ systems with controlled derivations, motivated by the paradigm of Watson-Crick complementarity in the operational sense. In these systems each letter has a complementary letter and depending on a special condition (a trigger), a derivation step is applied either to the string or to its complementary. A network of standard Watson-Crick $D0L$ systems (an $NSWD0L$ system) is a finite set of $SWD0L$ systems over a common DNA-like alphabet which act on their own strings in parallel and after each derivation step communicate some of the obtained words to each other. The condition for communication is determined by the trigger for turning to the complementary. In this paper we show that $NSWD0L$ systems form a class of computationally complete devices, that is, any recursively enumerable language can be identified by a network of standard Watson-Crick $D0L$ systems.

1 Introduction

Watson-Crick complementarity is a fundamental concept in DNA computing. A notion, called Watson-Crick $D0L$ system ($WDOl$ system), where the paradigm of complementarity is considered in the operational sense, was introduced and proposed for further investigations in [6] and [7]. In this paper we deal with networks of so-called standard Watson-Crick $D0L$ systems, important particular variants of Watson-Crick $D0L$ systems.

A standard Watson-Crick $D0L$ system is a $D0L$ system having a so-called DNA-like alphabet and functioning with controlled derivations. A DNA-like alphabet consists of $2n$ letters, $n \geq 1$, where $n$ symbols are called purines and the $n$ other symbols are called pyrimidines. Each purine has a complementary letter which is a pyrimidine and each pyrimidine has a complementary symbol which is a purine, and this relation is symmetric. The controlled derivation in a standard Watson-Crick $D0L$ system is as follows: after rewriting the string by applying rules of the $D0L$ system in parallel, the number of occurrences of purines and that of pyrimidines in the obtained string are checked. If in the new string there are more occurrences of pyrimidines than that of purines, then each
letter in the string is replaced by its complementary letter and the derivation continues from this string, otherwise the derivation continues in the usual manner.

The idea behind the concept is the following: in the course of the computation or development things can go wrong to such extent that it is of worth to continue with the complementary string, which is always available. Watson-Crick complementarity is viewed as an operation: together with or instead of a word we consider its complementary word.

Watson-Crick $D0L$ systems raise a lot of interesting questions to study: stability problems, characterization of length sequences, etc. The interested reader can find further information on properties of these systems in [8], [9].

A step further was made in [1] where networks of Watson-Crick $D0L$ systems ($NW D0L$ systems) were introduced and their behaviour was studied. In the general case, the trigger, the condition for turning to the complementary, is a logical valued mapping on the set of words over the alphabet of the system with the property that whenever for a string the value of this mapping is false (the string is not correct) then the value for the complementary of the string must be true (the complementary string must be correct). Moreover, every axiom in the system is a correct string. A network of Watson-Crick $D0L$ systems (an $NW D0L$ system) is a finite set of $WD0L$ systems over a common DNA-like alphabet which act on their own strings in parallel and after each derivation step communicate some of the obtained words to each other. The condition for communication is determined by the trigger for turning to the complementary. In [1] $NW D0L$ systems with two main variants of protocols were studied: in the first case, after a parallel rewriting step the nodes keep the correct strings and the corrected strings (complements of the not correct strings) and communicate a copy of every correct string to each other node. In the second case, the nodes, again, keep both the correct and the corrected strings but communicate copies of the corrected strings. The two protocols realize different philosophies: in the first case the nodes inform each other about their correct activities, in the second case they give information on the correction of their failures.

Network architectures are in the focus of interest in present computer science. One of the main areas of investigations is the study how powerful computational tools can be obtained by using networks of simple computing devices functioning with simple communication protocols.

In this paper we deal with this question. We prove that networks of standard Watson-Crick $D0L$ systems, working with the first type of communication protocol mentioned above, form a class of computationally complete devices, that is, any recursively enumerable language can be obtained as the language of an extended $NSWD0L$ system. The language of an extended $NSWD0L$ system is the set of words which are over a special subalphabet of the system (the terminal alphabet) and which appear at a dedicated node, the master, at a derivation step during the computation.

2 Preliminaries and basic notions

Throughout the paper we assume that the reader is familiar with the basic notions of formal language theory. For further details and unexplained notions consult [3], [4], and [5].
The set of nonempty words over an alphabet $\Sigma$ is denoted by $\Sigma^+$; if the empty string, $\lambda$, is included, then we use notation $\Sigma^*$. A set of strings $L \subseteq \Sigma^*$ is said to be a language over alphabet $\Sigma$. For a string $w \in L$ and for a set $U \subseteq \Sigma$, we denote by $|w|_U$ the number of occurrences of letters of $U$ in $w$.

Now we recall the basic notions concerning standard Watson-Crick $DOL$ systems, introduced in [6] and [7]. The interested reader can find further information on the topic in [8] and [9].

By a DNA-like alphabet we mean an alphabet $\Sigma$ with $2n$ letters, $n \geq 1$, where $\Sigma$ is of the form $\Sigma = \{a_1, \ldots, a_n, \overline{a}_1, \ldots, \overline{a}_n\}$. Letters $a_i$ and $\overline{a}_i$, $1 \leq i \leq n$, are said to be complementary letters. $\Sigma_1 = \{a_1, \ldots, a_n\}$ is said to be the subalphabet of purines of $\Sigma$ and $\Sigma_2 = \{\overline{a}_1, \ldots, \overline{a}_n\}$ is called the subalphabet of pyrimidines.

A string $w \in \Sigma^*$ is said to be correct if $|w|_{\Sigma_1} \geq |w|_{\Sigma_2}$ holds, otherwise the string is called not correct.

We denote by $h_w$ the letter to letter endomorphism of a DNA-like alphabet $\Sigma$ mapping each letter to its complementary.

A standard Watson-Crick $DOL$ system (an $SWDOL$ system, for short) is a triple $H = (\Sigma, P, w_0)$, where $\Sigma$ is a DNA-like alphabet, the alphabet of the system, $P$ is a set of pure context-free rules over $\Sigma$, the set of rewriting rules of the system, and $w_0$ is a nonempty correct word over $\Sigma$, the axiom of $H$. Furthermore, $P$ is complete and deterministic, that is, $P$ has for each letter $b$ in $\Sigma$ exactly one rule of the form $b \rightarrow u$, $u \in \Sigma^*$.

The direct derivation step in $H$ is defined as follows: for two strings $x, y \in \Sigma^*$ we say that $x$ directly derives $y$ in $H$, denoted by $x \Rightarrow_H y$, if $x = x_1 \ldots x_m$, $y = y_1 \ldots y_m$, $m \geq 1$, and $z_i = y_i$ if $y_i \ldots y_m$ is a correct word and $z_i = h_w(y_i)$ otherwise, where $x_i \rightarrow y_i \in P$, $1 \leq i \leq m$. The empty word directly derives itself. The parallel rewriting of each $x_i$ to $y_i$, $1 \leq i \leq m$, is denoted by $x_1 \ldots x_m \Rightarrow_P y_1 \ldots y_m$.

Thus, if after applying a parallel rewriting to the string the obtained new string has less occurrences of purines than that of pyrimidines, then the new string must turn to the complementary and the derivation continues from this complementary word, otherwise the derivation continues in the usual manner.

Now we define the basic notions concerning networks of standard Watson-Crick $DOL$ systems, introduced in [1]. A network of standard Watson-Crick $DOL$ systems is a finite set of standard Watson-Crick $DOL$ systems over a common DNA-like alphabet which are located at nodes of a virtual graph (a network). These $SWDOL$ systems generate strings in parallel and communicate them to the other $SWDOL$ systems in the network. Communication is defined via correctness/incorrectness of the rewritten string: whenever the obtained new string is correct, the node sends a copy of the string to each other node and if it proves to be not correct, then the string turns to its complementary and this complementary string is not communicated. The node keeps both the correct and the corrected strings.

By a network of standard Watson-Crick $DOL$ systems (an $NSWDOL$ system, for short) with $m$ components, where $m \geq 1$, we mean an $m+1$-tuple

$$\Gamma = (\Sigma, (P_1, w_1), \ldots, (P_m, w_m)),$$

where
• $\Sigma$ is a DNA-like alphabet, the alphabet of the system,

• $P_i$ is a complete deterministic set of pure context-free rules over $\Sigma$, the set of rules of the $i$-th component (or the $i$-th node) of $\Gamma$, $1 \leq i \leq m$, and

• $w_i$ is a correct nonempty word over $\Sigma$, the axiom of the $i$-th component, $1 \leq i \leq m$.

The first component, $(P_1, w_1)$, is said to be the master node.

By a state of an NSWD0L system $\Gamma = (\Sigma, (P_1, w_1), \ldots, (P_m, w_m))$, $m \geq 1$, we mean an $m$-tuple $(L_1, \ldots, L_m)$, where $L_i$ is a set of correct words over $\Sigma$, $1 \leq i \leq m$.

The initial state of the system is $\{(w_1), \ldots, \{w_m\}\}$.

NSWD0L systems change their state as follows:

Let $\Gamma = (\Sigma, (P_1, w_1), \ldots, (P_m, w_m))$, $m \geq 1$, be an NSWD0L system and let $s_1 = (L_1, \ldots, L_m)$ and $s_2 = (L'_1, \ldots, L'_m)$ be two states of $\Gamma$.

We say that $s_1$ directly derives $s_2$, written as $s_1 \Rightarrow_{\Gamma} s_2$, if the following condition holds: for each $i$, $1 \leq i \leq m$, $L'_i = A'_i \cup B'_i$, where

$A'_i = \{z \mid z = h_w(y), x \Rightarrow P_i y, x \in L_i, y \text{ is a not correct string}\}$ and

$B'_i = \{y \mid x \Rightarrow P_j y, x \in L_j, 1 \leq j \leq m, y \text{ is a correct string}\}$.

The transitive and reflexive closure of $\Rightarrow_{\Gamma}$ is denoted by $\Rightarrow_{\Gamma}^*$.

Thus, according to this protocol, after applying a parallel rewriting step, the node sends a copy of every obtained correct string to each other node and changes each obtained not correct string to the complementary. At any derivation step, there are only correct strings in the network.

The language of an NSWD0L system $\Gamma = (\Sigma, (P_1, w_1), \ldots, (P_m, w_m))$, $m \geq 1$, is

$L(\Gamma) = \{u_1 \in L_1 \mid \{(w_1), \ldots, \{w_m\}\} \Rightarrow_{\Gamma}^* (L_1, \ldots, L_m)\}$.

That is, the language of $\Gamma$ is the set of strings which appear at the master node at the derivation steps of the computation, included the initial step with the axioms.

By an extended NSWD0L system (an ENSWD0L system, for short) we mean an $m + 2$-tuple $\Gamma = (\Sigma, T, (P_1, w_1), \ldots, (P_m, w_m))$, $m \geq 1$, where $T \subseteq \Sigma$ and all other components of $\Gamma$ are defined in the same way as in the case of NSWD0L systems. The language of $\Gamma$ is defined by $L(\Gamma) = \{u_1 \in (T^* \cap L_1) \mid \{(w_1), \ldots, \{w_m\}\} \Rightarrow_{\Gamma}^* (L_1, \ldots, L_m)\}$.

### 3 Power of ENSWD0L systems

In the following we show that any recursively enumerable language can be obtained as the language of an extended NSWD0L system. The idea of the proof is based on simulation of generation of words of the recursively enumerable language according to the Extended Post Correspondence (EPC).

Let $T = \{a_1, \ldots, a_n\}$ be an alphabet, where $n \geq 1$. An Extended Post Correspondence (an EPC, for short) is a pair $P = \{(u_1, v_1), \ldots, (u_r, v_r)\}, (z_{a_1}, \ldots, z_{a_n})$, where $u_j, v_j, z_{a_i} \in \{0, 1\}^*$, $1 \leq j \leq r, 1 \leq i \leq n$.

The language represented by $P$ in $T$, written as $L(P)$, is

$L(P) = \{x_1 \ldots x_m \in T^* \mid \text{there are indices } s_1, \ldots, s_t \in \{1, \ldots, r\}, t \geq 1,$

such that $u_{s_1} \ldots u_{s_t} = v_{s_1} \ldots v_{s_t} z_{x_1} \ldots z_{x_m}\}$. 

It is known that for each recursively enumerable language $L$ there exists an Extended Post Correspondence $P$ such that $L = L(P)$ [2].

Thus, according to the above theorem, a word $w = x_1 \ldots x_m$, $x_i \in T$, $1 \leq i \leq m$, is in $L$ if and only if there exist indices $s_1, \ldots, s_t \in \{1, \ldots, r\}$ such that the two words $u_{s_1} \ldots u_{s_t}$ and $u_{s_1} \ldots u_{s_t}z_{x_1} \ldots z_{x_m}$ consist of the same number of digits and they have the same value as binary numbers.

It is easy to see that we can determine the words of $L$ as follows: We start the generation with a string of the form $u_{s_1}v_{s_1}$, $s_1 \in \{1, \ldots, r\}$. Then we add $u$- and $v$-strings to the string in the correct manner while obtaining a string of the form $u_{s_1} \ldots u_{s_t}v_{s_1} \ldots v_{s_t}$, $t \geq 1$. Then, in the second phase of the generation we add $x$- and $z$-strings to the string in a correct manner while obtaining $x_1 \ldots x_m u_{s_1} \ldots u_{s_t}v_{s_1} \ldots v_{s_t} z_{x_1} \ldots z_{x_m}$. In the final phase we check whether $\alpha = u_{s_1} \ldots u_{s_t}$ and $\beta = v_{s_1} \ldots v_{s_t} z_{x_1} \ldots z_{x_m}$ are equal or not, and if they are equal, we eliminate both substrings from the string. If the empty word is in $L$, then after the first phase of the above process, we continue with the final generation phase.

We shall use the following notation in the sequel: for a word $u \in \{0,1\}^*$ we denote by $\text{val}(u)$ the value of $u$ as a binary number and by $\text{dig}(u)$ the number of digits in $u$.

**Theorem 3.1** For each recursively enumerable language $L$ there exists an ENSWD0L system $\Gamma$ such that $L = L(\Gamma)$.

**Proof.** Let $L \subseteq T^*$, where $T = \{a_1, \ldots, a_n\}$, $n \geq 1$, and let $L$ be represented by an EPC $P = \{(u_1, v_1), \ldots, (u_r, v_r)\}$, $(z_{a_1}, \ldots, z_{a_n})$, where $u_j, v_j, z_{a_i} \in \{0,1\}^*$, $1 \leq j \leq r$, $1 \leq i \leq n$. To prove the statement, we construct an ENSWD0L system $\Gamma$ such that $L(\Gamma) = L(P)$ and $\Gamma$ simulates the generation of words of $L$ according to $P$. For each pair $(u_j, v_j)$, $1 \leq j \leq r$, and for each pair $(a_i, z_{a_i})$, $1 \leq i \leq n$, $\Gamma$ will have a dedicated node which simulates the effect of adding the pair to the string in generation in a correct manner. (For legibility, we use the short term "the node for the pair $(u, v)$ or $(a, z)$" in the sequel instead of the long version "the node dedicated for simulating the effect of adding the pair $(u, v)$ or $(a, z)$ to the string in generation."")

Furthermore, $\Gamma$ will have a node dedicated for deciding whether or not the two auxiliary substrings $\alpha$ and $\beta$ over $\{0,1\}$ (see the short explanation before the theorem) are equal.

The nodes for the pairs $(a_i, z_{a_i})$, $1 \leq i \leq n$, will have the strings $x_1 \ldots x_m u_{s_1} \ldots u_{s_t} v_{s_1} \ldots v_{s_t} z_{x_1} \ldots z_{x_m}$, $x_i \in T$, $1 \leq i \leq m$, $s_1, \ldots, s_t \in \{1, \ldots, r\}$, $t \geq 1$, in the coded form $x_1 \ldots x_m s_0 A_1^k s_2 A_2^l s_1 B_1^j s_2 B_2^k s_3 C_1^h s_3 C_2^l s_4 D_1^g s_4 D_2^l s_5$, where $k$ is the binary value of $\alpha = u_{s_1} \ldots u_{s_t}$, $l$ is the binary value of $\beta = v_{s_1} \ldots v_{s_t}$, $h$ is the number of digits of $\alpha$, and $g$ is the number of digits of $\beta$.

Similarly, the nodes for the pairs $(u_j, v_j)$, $1 \leq j \leq r$, will have the strings $u_{s_1} \ldots u_{s_t} v_{s_1} \ldots v_{s_t}$, $s_1, \ldots, s_t \in \{1, \ldots, r\}$, $t \geq 1$, in the coded form $s_0 A_1^k s_1 B_1^j s_1 B_2^k s_3 C_1^h s_3 C_2^l s_4 D_1^g s_4 D_2^l s_5$, where $k$ is the binary value of $\alpha = u_{s_1} \ldots u_{s_t}$, $l$ is the binary value of $\beta = v_{s_1} \ldots v_{s_t}$, $h$ is the number of digits of $\alpha$, and $g$ is the number of digits of $\beta$.

When we simulate the effect of adding the pair $(u_j, v_j)$, $1 \leq j \leq r$, or the pair $(a_i, z_{a_i})$, $1 \leq i \leq n$, to the string, then the number of occurrences of $A$-s, $B$-s, $C$-s, and $D$-s in the string will change according to the simulated effect.

Finally, the node dedicated for deciding whether or not $\alpha = \beta$ holds will decide whether or not $k = l$ and $h = g$ hold for the string $x_1 \ldots x_m s_0 A_1^k s_2 A_2^l s_1 B_1^j s_2 B_2^k s_3 C_1^h s_3 C_2^l s_4 D_1^g s_4 D_2^l s_5$ or for the string $s_0 A_1^k s_1 B_1^j s_1 B_2^k s_3 C_1^h s_3 C_2^l s_4 D_1^g s_4 D_2^l s_5$. (The latter case is for determining whether or not the empty word is in $L$.)
After the above short explanation, we define \( \Gamma \). To help the legibility, we provide the reader only with the necessary details.

Let

\[
\Gamma = (\Sigma, T, (P_e, w_e), (P_{(u_1,v_1)}, w_{(u_1,v_1)}), \ldots, (P_{(u_r,v_r)}, w_{(u_r,v_r)}), (P_{(a_1,z_{a_1})}, w_{(a_1,z_{a_1})}), \ldots, (P_{(a_n,z_{a_n})}, w_{(a_n,z_{a_n})})),
\]

where \( n \) and \( r \) are given by \( \text{EPC} \ P \).

Let

\[
\Sigma = \{ a_i, a_{i,j}, \overline{a}_i, \overline{a}_{i,j} \mid 1 \leq i \leq n, 3 \leq j \leq 7 \} \cup \{ \bar{0}, F, \bar{0}, \bar{F}, \bar{E} \}
\]

\[
\cup \{ X_j, \bar{X}_j \mid 1 \leq j \leq 7, X \in \{ A, B, C, D \} \}
\]

\[
\cup \{ \bar{s}_j, \bar{X}_j \mid 1 \leq j \leq 2, X \in \{ A, B, C, D \} \}.
\]

We note that \( F \) is a special symbol, the so-called trap symbol.

The axioms are defined as follows: \( w_{(u_j,v_j)} = E \), for \( 1 \leq j \leq r \), \( w_{(a_i,z_{a_i})} = F \), for \( 1 \leq i \leq n \), and \( w_e = F \).

Notice that the master node is \((P_e, w_e)\).

In the following we define the rule sets of the nodes, with some short explanations concerning their functioning.

The rule set \( P_{(u_j,v_j)} \) of the node dedicated for simulating the effect of adding the pair \((u_j, v_j), 1 \leq j \leq r\), to the string consists of the following rules:

- \( A_1 \rightarrow A_1^{\text{dig}(u_j)}, \quad B_1 \rightarrow B_1^{\text{dig}(v_j)}, \quad C_1 \rightarrow C_1, \quad D_1 \rightarrow D_1, \)
- \( \bar{s}_1 \rightarrow A_1^{\text{val}(u_j)} \bar{s}_1 A, \quad \bar{s}_1 B \rightarrow B_1^{\text{val}(v_j)} \bar{s}_1 B, \quad \bar{s}_1 C \rightarrow C_1^{\text{dig}(u_j)} \bar{s}_1 C, \quad \bar{s}_1 D \rightarrow D_1^{\text{dig}(v_j)} \bar{s}_1 D, \)
- \( \bar{s}_0 \rightarrow \bar{s}_0, \) and for any other letter \( Y \) from \( \Sigma \) but \( E \) the node contains rule \( Y \rightarrow F. \)

Moreover, the node has rule \( E \rightarrow \bar{s}_0 A_1^{h_j} \bar{s}_1 A \bar{s}_1 B_1^{l_j} \bar{s}_1 B_1 C_1^{k_j} \bar{s}_1 C_1 \bar{s}_1 D_1^{g_j} \bar{s}_1 D_1, \) where \( A_1^{h_j} \) and \( C_1^{h_j} \) represent the value and the number of digits of \( u_j \), and \( B_1^{l_j} \) and \( D_1^{g_j} \) represent the value and the number of digits of \( v_j \), respectively.

The reader can easily check that starting from a string of the form

\[
\bar{s}_0 A_1^{h_1} \bar{s}_1 A B_1^{l_1} \bar{s}_1 B C_1^{h_1} \bar{s}_1 C D_1^{g_1} \bar{s}_1 D,
\]

where \( A_1^{h_1} \) and \( C_1^{h_1} \) represent the value and the number of digits of \( \alpha = u_{s_1} \ldots u_{s_t} \), and \( B_1^{l_1} \) and \( D_1^{g_1} \) represent the value and the number of digits of \( \beta = v_{s_1} \ldots v_{s_t} \), \( s_1, \ldots, s_t \in \{1, \ldots, r\}, t \geq 1 \), by applying the above rules we obtain a string

\[
\bar{s}_0 A_1^{k_{2}} \bar{s}_1 A B_1^{l_{2}} \bar{s}_1 B C_1^{h_2} \bar{s}_1 C D_1^{g_2} \bar{s}_1 D,
\]

where \( k_2 = k_1 \cdot 2^{\text{dig}(u_j)} + \text{val}(u_j), \quad l_2 = l_1 \cdot 2^{\text{dig}(v_j)} + \text{val}(v_j), \quad h_2 = h_1 + \text{dig}(u_j) \) and \( g_2 = g_1 + \text{dig}(v_j) \). Thus, the rewriting simulates the effect of adding pair \((u_j, v_j)\) to the string \( u_{s_1} \ldots u_{s_t} v_{s_1} \ldots v_{s_t} \) in the correct manner, obtaining \( u_{s_1} \ldots u_{s_t} u_j v_{s_1} \ldots v_{s_t} v_j \).

The rule set \( P_{(a_i,z_{a_i})} \) of the node dedicated for simulating the effect of adding the pair \((a_i, z_{a_i}), 1 \leq i \leq n\), to the string contains the following rules:

- \( A_1 \rightarrow A_2, \quad B_1 \rightarrow B_2^{\text{dig}(z_{a_i})}, \quad C_1 \rightarrow C_2, \quad D_1 \rightarrow D_2, \)
- \( \bar{s}_1 A \rightarrow \bar{s}_2 A, \quad \bar{s}_1 B \rightarrow B_2^{\text{val}(z_{a_i})} \bar{s}_2 B, \quad \bar{s}_1 C \rightarrow \bar{s}_2 C, \quad \bar{s}_1 D \rightarrow D_2^{\text{dig}(z_{a_i})} \bar{s}_2 D, \)
and
\[ A_2 \rightarrow A_2, \quad B_2 \rightarrow B_2^{\text{dig}(z_{a_i})}, \quad C_2 \rightarrow C_2, \quad D_2 \rightarrow D_2, \]
\[ S_{2,A} \rightarrow S_{2,A}, \quad S_{2,B} \rightarrow B_2^{\text{val}(z_{a_i})}S_{2,B}, \quad S_{2,C} \rightarrow S_{2,C}, \quad S_{2,D} \rightarrow D_2^{\text{dig}(z_{a_i})}S_{2,D}, \]

moreover, it has \( S_0 \rightarrow a_iS_0 \), and \( a_j \rightarrow a_j \), for \( 1 \leq j \leq n \), and for the other letters \( Y \) in \( \Sigma \) not listed above the node has rule \( Y \rightarrow F \).

Analogously to the previous considerations, we can see that starting from a string
\[ uS_0A_p^{k_1}S_{p,A}B_p^{l_1}S_{p,B}C_p^{h_1}S_{p,C}D_p^{g_1}S_{p,D}, \]
where \( k_2 = k_1, \ l_2 = l_1 \cdot 2^{\text{dig}(z_{a_i})} + \text{val}(z_{a_i}), \ h_2 = h_1 \) and \( g_2 = g_1 + \text{dig}(z_{a_i}) \). Thus, the rewriting simulates the effect of adding pair \( (a_i, z_{a_i}) \) to the string \( uu_1 \ldots u_{s_i}v_{s_i} \ldots v_{s_i}z_u \) in the correct manner, obtaining \( ua_iu_1 \ldots u_{s_i}v_1 \ldots v_{s_i}z_u z_{a_i} \).

Finally, we list the rules in the rule set \( P_e \) of the node dedicated for deciding whether the generated string satisfies EPC \( P \) or not, that is, whether the corresponding two strings, \( \alpha \) and \( \beta \), mentioned above, are equal or not. This is done by using the possibility of turning to the complementary. To help the reader in understanding how the decision is done, we list the rules together with a derivation.

We first mention that in \( P_e \) the rule for the trap symbol, \( F \), is \( F \rightarrow F \). Let
\[ x_1 \ldots x_m S_0 A_2^{k_2} S_{2,A} B_2^{l_2} S_{2,B} B_2^{h_2} S_{2,C} D_2^{g_2} S_{2,D}, \]
be a string at node \( (P_e, w_e) \), \( x_i \in T \), \( 1 \leq i \leq m \).

Then, by the first step, having rules \( S_0 \rightarrow \lambda, \ S_{i,X} \rightarrow \lambda, \ X_i \rightarrow X_3 \), for \( i = 1, 2, X \in \{ A, B, C, D \} \), and \( a_j \rightarrow a_j, 1 \leq j \leq n \), the string changes for \( x_1 \ldots x_m A_2^{k_2} B_2^{l_2} C_2^{h_2} D_2^{g_2} S_{2,D} \).

Then, in the next derivation step having productions \( a_{i,3} \rightarrow a_{i,4} \bar{a}_{i,4}, 1 \leq i \leq n \), and \( A_3 \rightarrow \bar{A}_4, B_3 \rightarrow B_4, C_3 \rightarrow C_4 \bar{C}_4, D_3 \rightarrow D_4 \bar{D}_4 \), and \( A_4 \rightarrow F, B_4 \rightarrow \bar{F}_4 \), we obtain a string of the form
\[ x_{1,4} \bar{x}_{1,4} \ldots x_{m,4} \bar{x}_{m,4} A_4^{k_4} B_4(C_4 \bar{C}_4)^h(D_4 \bar{D}_4)^g. \]
The derivation will lead to a string over \( T \) only if \( k \leq l \), otherwise the string turns to its complementary and at the next step occurrences of the trap symbol \( F \) will be introduced which never will disappear from the string. Suppose that the derivation leads to a terminal string. Then, having productions \( a_{i,4} \rightarrow a_{i,5}, \bar{a}_{i,4} \rightarrow \bar{a}_{i,5}, 1 \leq i \leq n, \bar{A}_4 \rightarrow A_5, B_4 \rightarrow \bar{B}_5, \bar{A}_5 \rightarrow F, B_5 \rightarrow F, C_4 \rightarrow C_5, \bar{C}_4 \rightarrow \bar{C}_5, D_4 \rightarrow D_5, \bar{D}_4 \rightarrow \bar{D}_5 \), we obtain a string of the form
\[ x_{1,5} \bar{x}_{1,5} \ldots x_{m,5} \bar{x}_{m,5} A_5^{k_5} B_5^l(C_5 \bar{C}_5)^h(D_5 \bar{D}_5)^g. \]
Again, the derivation can lead to a terminal string only if \( k \geq l \), otherwise, at the next step occurrences of the trap symbol will be introduced. Suppose that this is not the case. Then we continue the rewriting. Having rules \( a_{i,5} \rightarrow a_{i,6}, \bar{a}_{i,5} \rightarrow \bar{a}_{i,6}, 1 \leq i \leq n, \) and
$A_5 \rightarrow \lambda$, $\bar{B}_5 \rightarrow \lambda$, $\bar{C}_5 \rightarrow \lambda$, $\bar{D}_5 \rightarrow \lambda$, $C_5 \rightarrow \bar{C}_6$, $D_5 \rightarrow D_6$, $C_6 \rightarrow F$, $\bar{D}_6 \rightarrow F$, the obtained string is

$$x_{1,6}\bar{x}_{1,6} \ldots x_{m,6}\bar{x}_{m,6}\bar{C}^h_6 D^g_6.$$  

As in the previous cases, the derivation leads to a terminal string only if $h \leq g$. Let us suppose that this condition holds. Then we continue the derivation. Having rules $a_{i,6} \rightarrow a_{i,7}$, $\bar{a}_{i,6} \rightarrow \bar{a}_{i,7}$, $1 \leq i \leq n$, and $\bar{C}_6 \rightarrow C_7$, $D_6 \rightarrow \bar{D}_7$, $\bar{C}_7 \rightarrow F$, $D_7 \rightarrow \lambda$ the next step leads to string

$$x_{1,7}\bar{x}_{1,7} \ldots x_{m,7}\bar{x}_{m,7}C^h_7 \bar{D}^g_7.$$  

Similarly to the cases above, a terminal word can be obtained only if $h \geq g$. Then, having rules $a_{i,7} \rightarrow a_i$, $\bar{a}_{i,7} \rightarrow \lambda$, $1 \leq i \leq n$, $C_7 \rightarrow \lambda$, $\bar{D}_7 \rightarrow \lambda$, we derive

$$x_1 \ldots x_m.$$

For any other letter $Y$ in $\Sigma$ we have not listed above, the node has rule $Y \rightarrow F$.

We note that the above procedure also works if we start from a string

$$\$0_A^k\$s_{1,A}B_1^l\$s_{1,B}C_1^h\$s_{1,C}D_1^g.$$

The derivation results in the empty word only if $\lambda \in L(P)$ holds.

Now we should prove that $\Gamma$ derives all words of $L$ but not more.

Suppose that $x_1 \ldots x_m \in L$, $x_i \in T$, $1 \leq i \leq m$, that is, there are indices $s_1, \ldots, s_t \in \{1, \ldots, r\}$ such that $u_{s_1} \ldots u_{s_t} = v_{s_1} \ldots v_{s_t}z_{x_1} \ldots z_{x_m}$ holds. Then $x_1 \ldots x_m$ can be obtained in $\Gamma$ as follows: First $E$, the axiom of the node for simulating the effect of adding the pair $(u_{s_1}, v_{s_1})$, the axiom of node for $(u_{s_1}, v_{s_1})$, for short, is rewritten to the string representing $u_{s_1}v_{s_1}$ in the coded form, and then, by communication the string is forwarded to the node for $(u_{s_2}, v_{s_2})$. Then, the communicated string is rewritten at this node and it is forwarded to the next node for $(u, v)$ in the order. We continue this procedure while the string representing $u_{s_1} \ldots u_{s_t}v_{s_1} \ldots v_{s_t}$ is generated at node for $(u_{s_2}, v_{s_2})$. Then, the string is communicated to the node for $(x_1, z_{x_1})$, where it is rewritten and then it is communicated to the next node in the order, a node for some pair $(x, z_x)$. Continuing this procedure, we finish this part of the generation at node for $(x_m, z_{x_m})$ with a string representing $x_1 \ldots x_m u_{s_1} \ldots u_{s_t}v_{s_1} \ldots v_{s_t}z_{x_1} \ldots z_{x_m}$. Then the string is forwarded to node $(P_e, w_e)$, where in some steps its substring $u_{s_1} \ldots u_{s_t}v_{s_1} \ldots v_{s_t}z_{x_1} \ldots z_{x_m}$ is eliminated. Thus, $x_1 \ldots x_m$ is an element of $L(\Gamma)$. The procedure for computing $\lambda \in L(P)$, if $\lambda \in L(P)$, is analogous.

We should prove that $\Gamma$ does not generate a word not in $L$. By the definition of the rule sets of the nodes, we can see that for each string at the node or communicated to the node, the node for the pair $(u_j, v_j)$, $1 \leq j \leq r$, either produces a new string representing a word of one of the forms $u_jv_j$ or $u_{s_1} \ldots u_{s_t}u_jv_{s_1} \ldots v_{s_t}v_j$, $s_1, \ldots, s_t \in \{1, \ldots, r\}$, $t \geq 1$, or it produces a new string which contains the trap symbol $F$, which symbol never disappears from the string, so the obtained string is irrelevant from the point of view of generation of terminal words of $\Gamma$. Notice that after each derivation step all the strings that can be found at the node are communicated to the other nodes. Analogously, for each string at the node or communicated to the node, it holds that the node for $(a_i, z_{a_i})$, $1 \leq i \leq n$, either produces a string representing a string of the form $u_{a_i}u_{s_1} \ldots u_{s_t}v_{s_1} \ldots v_{s_t}z_{a_s}$, $u \in T^*$, $z_a$ is the sequence of $z$-s which corresponds to $u$, or it generates a string with an occurrence of the trap symbol, $F$. Similarly to the previous case, after each derivation step the node
communicates all the strings it has. But only those strings have no occurrence of the trap symbol at the above two types of nodes which represent strings which correspond to generation phases of words of \( L \) according to EPC \( P \). Similarly to the above cases, the master node, \((P_e, w_e)\), either produces a terminal string (or the empty word) from a string it has or it received by communication, or the node generates a string with an occurrence of the trap symbol. After each derivation step the master node receives all strings of the previous two types of nodes and communicates the correct strings it produces to them. At the next step these communicated strings at the new location either will introduce the trap symbol or if the string is a terminal string or the empty word at an appropriate node it will return without any change to the master node. Thus, any terminal word (included the empty word) which can be generated by \( \Gamma \) can be generated according to \( P \) but not more. Hence the result.

Acknowledgement
Research supported in part by the Hungarian Scientific Research Fund "OTKA" Grant no. T 029615.

References


