J. Dassow, B. Truthe (Eds.): Colloquium on the Occasion of the 50th Birthday of Victor Mitrana.
Otto von Guericke Universität Magdeburg, Germany, June 27, 2008. Proceedings, pages 37 – 52.

On Small Accepting Networks of Evolutionary Processors with Regular Filters

BIANCA TRUTHE

Otto-von-Guericke-Universität Magdeburg, Fakultät für Informatik
PSF 4120, D-39016 Magdeburg, Germany

truthe@iws.cs.uni-magdeburg.de

Abstract: We show that every context-sensitive language can be accepted by an accepting network of non-increasing evolutionary processors with one substitution processor and one output node whose communication is controlled by regular languages. Every recursively enumerable language can be accepted by a network with three evolutionary processors: one substitution processor, one insertion processor and one output node. Also with insertion and deletion processors only (without substitution nodes), all recursively enumerable languages can be accepted. Then one insertion node, one deletion node and one output node are sufficient.

Keywords: Accepting networks of evolutionary processors, regular filters, size complexity.

1. Introduction

Motivated by some models of massively parallel computer architectures (see [10, 9]) networks of language processors have been introduced in [6] by ERZSÉBET CSUHAJ-VARJÚ and ARTO SALOMAA. Such a network can be considered as a graph where the nodes are sets of productions and at any moment of time a language is associated with a node.

Inspired by biological processes, JUAN CASTELLANOS, CARLOS MARTÍN-VIDE, VICTOR MITRANA and JOSÉ M. SEMPERE introduced in [4] a special type of networks of language processors which are called networks with evolutionary processors because the allowed productions model the point mutation known from biology. The sets of productions have to be substitutions of one letter by another letter or insertions of letters or deletion of letters; the nodes are then called substitution node or insertion node or deletion node, respectively. Results on networks of evolutionary processors can be found e. g. in [4, 5, 3, 2].

Accepting networks of evolutionary processors with regular filters were first investigated by JÜRGEN DASSOW and VICTOR MITRANA in [7]. Especially, they have shown that every network of non-increasing processors accepts a context-sensitive language.

In [8] and [1], we investigated the generative capacity of networks with evolutionary processors where only two types of nodes are allowed. Especially, we proved two results:

- Networks with substitution nodes and insertion nodes (but without deletion nodes) generate all context-sensitive languages and one substitution node and one insertion node are sufficient.
- Networks with insertion nodes and deletion nodes (but without substitution nodes) generate all recursively enumerable languages and one insertion node and one deletion node are sufficient.

In the present paper, we show the dual case:

- Every context-sensitive language can be accepted by an accepting network of evolutionary processors with regular filters and with one substitution node, one deletion node and one output node.
- Every recursively enumerable language can be accepted by an accepting network of evolutionary processors with regular filters and with one insertion node, one deletion node and one output node.

Further, we show that every recursively enumerable language can be accepted by a network with one substitution processor, one insertion processor and one output node.

Whereas networks consisting of substitution processors only cannot generate other languages than finite ones, accepting pure substitution networks can accept infinite languages. The reason is that generating networks start with a finite set (and substitution nodes cannot increase the length of the words) while accepting networks can get infinitely many input words. We show that all context-sensitive languages can be accepted by networks of substitution processors and that one substitution processor is sufficient (apart from an output node).

2. Definitions

We assume that the reader is familiar with the basic concepts of formal language theory (see e. g. [13]). We here only recall some notations used in the paper.

By V^* we denote the set of all words (strings) over V (including the empty word λ). The length of a word w is denoted by $|w|$.

In the proofs we shall often add new letters of an alphabet U to a given alphabet V. In all these situations, we assume that $V \cap U = \emptyset$.

A phrase structure grammar is specified as a quadruple $G = (N, T, P, S)$ where N is a set of non-terminals, T is a set of terminals, P is a finite set of productions which are written as $\alpha \to \beta$ with $\alpha \in (N \cup T)^* \setminus T^*$ and $\beta \in (N \cup T)^*$, and $S \in N$ is the axiom.

The grammar G is called monotone, if $|\alpha| \leq |\beta|$ holds for every rule $\alpha \to \beta$ of P where the exception $S \to \lambda$ is permitted if S does not occur on a right hand side of a rule. A monotone grammar is in Kuroda normal form if all its productions have one of the following forms:

$$AB \to CD,\ A \to CD,\ A \to x,\ \text{where } A,B,C,D \in N,\ x \in N \cup T;$$

again, $S \to \lambda$ is permitted if S does not occur on a right hand side of a rule.

We call a production $\alpha \to \beta$ a

- substitution if $|\alpha| = |\beta| = 1$,
- deletion if $|\alpha| = 1$ and $\beta = \lambda$.

We regard insertion as a counterpart of deletion. We write $\lambda \to a$, where a is a letter. The application of an insertion $\lambda \to a$ derives from a word w any word $w_1 a w_2$ with $w = w_1 w_2$ for some (possibly empty) words w_1 and w_2.

We now introduce the basic concept of this paper, the accepting networks of evolutionary processors.

Definition 1.

(i) *An accepting network of evolutionary processors of size n is a tuple*

$$\mathcal{N}^{(n)} = (U, V, N_1, N_2, \ldots, N_n, E, j, O)$$

where

- *U and V are finite alphabets (the input and network alphabet, resp.), $U \subseteq V$,*
- *for $1 \leq i \leq n$, $N_i = (M_i, I_i, O_i)$ where*
  - *M_i is a set of evolution rules of a certain type, $M_i \subseteq \{a \to b \mid a,b \in V\}$ or $M_i \subseteq \{a \to \lambda \mid a \in V\}$ or $M_i \subseteq \{\lambda \to b \mid b \in V\}$,*
  - *I_i and O_i are regular sets over V,*
- *E is a subset of $\{1,2,\ldots,n\} \times \{1,2,\ldots,n\}$,*
- *j is a natural number such that $1 \leq j \leq n$, and*
- *O is a subset of $\{1,2,\ldots,n\}$.*

(ii) *A configuration C of $\mathcal{N}^{(n)}$ is an n-tuple $C = (C(1), C(2), \ldots, C(n))$ if $C(i)$ is a subset of V^* for $1 \leq i \leq n$.*

(iii) *Let $C = (C(1), C(2), \ldots, C(n))$ and $C' = (C'(1), C'(2), \ldots, C'(n))$ be two configurations of $\mathcal{N}^{(n)}$. We say that C derives C' in one*

- *evolution step (written as $C \Longrightarrow C'$) if, for $1 \leq i \leq n$, $C'(i)$ consists of all words $w \in C(i)$ to which no rule of M_i is applicable and of all words w for which there are a word $v \in C(i)$ and a rule $p \in M_i$ such that $v \Longrightarrow_p w$ holds,*

– *communication step (written as $C \vdash C'$) if, for $1 \leq i \leq n$,*

$$C'(i) = (C(i) \setminus O_i) \cup \bigcup_{(k,i) \in E} C(k) \cap O_k \cap I_i.$$

The computation of a network $\mathcal{N}^{(n)}$ on an input word $w \in U^*$ is a sequence of configurations $C_t^w = (C_t^w(1), C_t^w(2), \ldots, C_t^w(n))$, $t \geq 0$, such that

– *$C_0^w = (C_0^w(1), C_0^w(2), \ldots, C_0^w(n))$ where $C_0^w(j) = \{w\}$ and $C_0^w(i) = \emptyset$ for $1 \leq i \neq j \leq n$,*

– *for any $t \geq 0$, C_{2t}^w derives C_{2t+1}^w in one evolution step: $C_{2t}^w \Longrightarrow C_{2t+1}^w$,*

– *for any $t \geq 0$, C_{2t+1}^w derives C_{2t+2}^w in one communication step: $C_{2t+1}^w \vdash C_{2t+2}^w$.*

(iv) *The languages $L_{\mathrm{w}}(\mathcal{N})$ weakly accepted by $\mathcal{N}$ and $L_{\mathrm{s}}(\mathcal{N})$ strongly accepted by $\mathcal{N}$ are defined as*

$$L_{\mathrm{w}}(\mathcal{N}) = \{\, w \in U^* \mid \exists t \geq 0 \, \exists o \in O : C_t^w(o) \neq \emptyset \,\},$$
$$L_{\mathrm{s}}(\mathcal{N}) = \{\, w \in U^* \mid \exists t \geq 0 \, \forall o \in O : C_t^w(o) \neq \emptyset \,\},$$

where $C_t^w = (C_t^w(1), C_t^w(2), \ldots, C_t^w(n))$, $t \geq 0$ is the computation of $\mathcal{N}$ on w.

Intuitively a network with evolutionary processors is a graph consisting of some, say n, nodes $N_1, N_2, \ldots, N_n$ (called processors) and the set of edges given by E such that there is a directed edge from N_k to N_i if and only if $(k,i) \in E$. The node N_j is called the input node; every node N_o with $o \in O$ is called an output node. Any processor N_i consists of a set of evolution rules M_i, an input filter I_i and an output filter O_i. We say that N_i is a substitution node or a deletion node or an insertion node if $M_i \subseteq \{a \to b \mid a, b \in V\}$ or $M_i \subseteq \{a \to \lambda \mid a \in V\}$ or $M_i \subseteq \{\lambda \to b \mid b \in V\}$, respectively. The input filter I_i and the output filter O_i control the words which are allowed to enter and to leave the node, respectively. With any node N_i and any time moment $t \geq 0$ we associate a set $C_t(i)$ of words (the words contained in the node at time t). Initially, the input node N_j contains an input word w; all other nodes do not contain words. In an evolution step, we derive from $C_t(i)$ all words applying rules from the set M_i. In a communication step, any processor N_i sends out all words $C_t(i) \cap O_i$ (which pass the output filter) to all processors to which a directed edge exists (only the words from $C_t(i) \setminus O_i$ remain in the set associated with N_i) and, moreover, it receives from any processor N_k such that there is an edge from N_k to N_i all words sent by N_k and passing the input filter I_i of N_i, i. e., the processor N_i gets in addition all words of $(C_t(k) \cap O_k) \cap I_i$. We start with an evolution step and then communication steps and evolution steps are alternately performed. The language accepted consists of all words w such that if w is given as an input word in the node N_j then, at some moment t, $t \geq 0$, one output node contains a word (weak acceptance) or all output nodes contain a word (strong acceptance).

3. Networks of Non-Increasing Nodes

Deletion and substitution nodes do not increase the length of the words. Such nodes are also called non-increasing. In a network with only non-increasing nodes, the length of every word in every node at any step in the computation is bounded by the length of the input word.

In this section, we show that every context-sensitive language can be accepted by an accepting network of evolutionary processors with deletion and substitution nodes but no insertion nodes. Especially, one substitution node (which is the input node), one deletion node and one output node are sufficient.

This is the inverse situation to that one considered in [8], where we have shown that networks with insertion nodes and substitution nodes (but without deletion nodes) generate all context-sensitive languages and one insertion node and one substitution node are sufficient.

Theorem 2. *For any context-sensitive language L, there is an accepting network $\mathcal{N}$ of evolutionary processors with exactly one substitution node, one deletion node and one output node without rules that weakly and strongly accepts the language L:*

$$L = L_{\mathrm{w}}(\mathcal{N}) = L_{\mathrm{s}}(\mathcal{N}).$$

Proof. Let L be a context-sensitive language and $G = (N, T, P, S)$ be a grammar in Kuroda normal form with $L(G) = L$. Let $R_1, R_2, \ldots, R_7$ be the following sets:

$$\begin{aligned}
R_1 &= \{\, x \to x_{p,0},\ x_{p,0} \to A \mid A \to x \in P,\ A \in N,\ x \in N \cup T \,\},\\
R_2 &= \{\, C \to C_{p,1} \mid p = A \to CD \in P \text{ or } p = AB \to CD \in P,\ A,B,C,D \in N \,\},\\
R_3 &= \{\, D \to D_{p,2} \mid p = A \to CD \in P \text{ or } p = AB \to CD \in P,\ A,B,C,D \in N \,\},\\
R_4 &= \{\, C_{p,1} \to C_{p,3} \mid p = A \to CD \in P \text{ or } p = AB \to CD \in P,\ A,B,C,D \in N \,\},\\
R_5 &= \{\, D_{p,2} \to D_{p,4} \mid p = A \to CD \in P \text{ or } p = AB \to CD \in P,\ A,B,C,D \in N \,\},\\
R_6 &= \{\, C_{p,3} \to A \mid p = A \to CD \in P \text{ or } p = AB \to CD \in P,\ A,B,C,D \in N \,\},\\
R_7 &= \{\, D_{p,4} \to B \mid p = AB \to CD \in P,\ A,B,C,D \in N \,\}.
\end{aligned}$$

We construct a network of evolutionary processors

$$\mathcal{N} = (T, V, (M_1, V^*, O_1), (M_2, I_2, V^*), (\emptyset, I_3, \emptyset), \{\,(1,2),(2,1),(1,3),(2,3)\,\}, 1, \{3\})$$

with

$$\begin{aligned}
V &= N \cup T \cup \bigcup_{p=A\to x} \{x_{p,0}\} \cup \bigcup_{\substack{p=A\to CD\\ p=AB\to CD}} \{C_{p,1}, D_{p,2}, C_{p,3}, D_{p,4}\},\\
M_1 &= R_1 \cup R_2 \cup R_3 \cup R_4 \cup R_5 \cup R_6 \cup R_7,\\
O_1 &= \{S, \lambda\} \cup V^* \setminus ((N \cup T)^* \bar{O} (N \cup T)^*),
\end{aligned}$$

where

$$\begin{aligned}\bar{O} = &\{\, x_{p,0} \mid p = A \to x \in P,\ A \in N,\ x \in N \cup T \,\} \\ &\cup \{\, C_{p,1} \mid p = A \to CD \in P \text{ or } p = AB \to CD \in P,\ A,B,C,D \in N \,\} \\ &\cup \{\, C_{p,1}D_{p,2} \mid p = A \to CD \in P \text{ or } p = AB \to CD \in P,\ A,B,C,D \in N \,\} \\ &\cup \{\, C_{p,3}D_{p,2} \mid p = A \to CD \in P \text{ or } p = AB \to CD \in P,\ A,B,C,D \in N \,\} \\ &\cup \{\, C_{p,3}D_{p,4} \mid p = A \to CD \in P \text{ or } p = AB \to CD \in P,\ A,B,C,D \in N \,\} \\ &\cup \{\, AD_{p,4} \mid p = AB \to CD \in P,\ A,B,C,D \in N \,\} \\ &\cup \{\, \lambda \,\},\end{aligned}$$

and

$$\begin{aligned}M_2 &= \{\, D_{p,4} \to \lambda \mid p = A \to CD \in P,\ A,B,C,D \in N \,\}, \\ I_2 &= (N \cup T)^* \{\, AD_{p,4} \mid p = A \to CD \in P,\ A,B,C,D \in N \,\} (N \cup T)^*, \\ I_3 &= \begin{cases} \{S, \lambda\} & \text{if } \lambda \in L, \\ \{S\} & \text{otherwise.} \end{cases}\end{aligned}$$

The network $\mathcal{N}$ has only one output node. Therefore, there is no difference between weak and strong acceptance, and we write $L(\mathcal{N})$ for the language accepted by the network $\mathcal{N}$.

First, we prove that every word $w \in L(G)$ is accepted by the network $\mathcal{N}$.

If $\lambda \in L(G)$ then $(\{\lambda\}, \emptyset, \emptyset) \Longrightarrow (\{\lambda\}, \emptyset, \emptyset) \vdash (\emptyset, \emptyset, \{\lambda\})$ and λ is accepted by the network $\mathcal{N}$.

Any derivation $w \Longrightarrow^* v$ with $v \neq \lambda$ of the grammar G can be simulated by the network $\mathcal{N}$ in reverse direction (by a reduction $v \Longrightarrow^* w$). We show that the application of a rule of the grammar G can be simulated by the network $\mathcal{N}$ in reverse direction. A direct reduction $v \Longrightarrow w$ always starts in the first node, the first step is an evolution step, and ends in the first node after a communication step (so the next step would be an evolution step again). Then $v \in C_{2t}(1)$ and $w \in C_{2(t+k)}(1)$ for two numbers $t \geq 0$ and $k > 0$. In the sequel, A, B, C, D are non-terminals, x is a non-terminal or terminal symbol and $w_1 w_2 \in (N \cup T)^*$.

Case 1. Application of a rule $p = A \to x \in P$ to a word $w_1 A w_2$.

This application leads in the grammar G to the word $w_1 x w_2$. We assume that the word $w_1 x w_2$ is in the first node at some moment before an evolution step ($w_1 x w_2 \in C_{2t}(1)$). We apply the rule $x \to x_{p,0} \in R_1$ and obtain the word $w_1 x_{p,0} w_2$ which cannot pass the output filter, so it remains in the first node. Then we apply the rule $x_{p,0} \to A \in R_1$ and obtain $w_1 A w_2 \in (N \cup T)^*$. This word also does not pass the output filter, so it remains in the first component: $w_1 A w_2 \in C_{2(t+2)}(1)$. Hence, the application of a rule $p = A \to x \in P$ to a word $w_1 A w_2$ can be simulated reversely in two evolution steps (the two corresponding communication steps have no effect).

Case 2. Application of a rule $p = AB \to CD \in P$ to a word $w_1 AB w_2$.

We assume $w_1 CD w_2 \in C_{2t}(1)$. This word is changed to $w_1 C_{p,1} D w_2$ (by an appropriate rule of R_2) which cannot pass the output filter, so it remains in the first node. It

is then changed to $w_1C_{p,1}D_{p,2}w_2$ (by R_3), and further, without leaving the first node, changed to $w_1C_{p,3}D_{p,2}w_2$ (by R_4), to $w_1C_{p,3}D_{p,4}w_2$ (by R_5), to $w_1AD_{p,4}w_2$ (by R_6) and finally to w_1ABw_2 (by R_7). This word is not communicated in the next step, since it cannot pass the output filter and we have $w_1ABw_2 \in C_{2(t+6)}(1)$. Hence, the application of a rule $p = AB \to CD \in P$ to a word w_1ABw_2 can be simulated reversely in six evolution steps (the six corresponding communication steps have no effect).

Case 3. Application of a rule $p = A \to CD \in P$ to a word w_1Aw_2.

We assume $w_1CDw_2 \in C_{2t}(1)$. As in the case before, this word is changed to the word $w_1C_{p,1}Dw_2$ and further, without leaving the first node, changed to $w_1C_{p,1}D_{p,2}w_2$ (by a rule of R_3), to $w_1C_{p,3}D_{p,2}w_2$ (by R_4), to $w_1C_{p,3}D_{p,4}w_2$ (by R_5), and to $w_1AD_{p,4}w_2$ (by R_6). This word passes the output filter of the node and the input put filter of the second node. So we have $w_1AD_{p,4}w_2 \in C_{2(t+5)}(2)$. The second node changes the word to w_1Aw_2. In the next communication step, this word moves to the first node and we obtain $w_1Aw_2 \in C_{2(t+6)}(1)$. Hence the application of a rule $p = A \to CD \in P$ to a word w_1Aw_2 can be simulated reversely in six evolution steps and two effective communication steps (the other four have no effect).

For any derivation $S \Longrightarrow^* w$ in the grammar G to a terminal word $w \in T^+$, there is a computation $C_0, C_1, \ldots, C_{2t+1}$ with $C_0 = (\{w\}, \emptyset, \emptyset)$ and $C_{2t+1} = (C_{2t+1}(1), C_{2t+1}(2), \emptyset)$ with $S \in C_{2t+1}(1) \cup C_{2t+1}(2)$ (the final reduction to S is achieved by some rule $x \to S$ in the first node or – if the first direct derivation is $S \Longrightarrow_p AB$ – by the rule $B_{p,4} \to \lambda$ in the second node). From both nodes, S reaches the third node in the next communication step. Hence, $S \in C_{2(t+1)}(3)$.

Thus, we have the inclusion $L(G) \subseteq L(\mathcal{N})$. We now show $L(\mathcal{N}) \subseteq L(G)$.

Let us first consider the case that the computation starts with λ in the first node. No rule can be applied. The word leaves the node (because it passes the output filter O_1). It enters the third node if λ belongs to $L(G)$, otherwise the word is lost and there is no word in the network any more. Hence, if λ is not in $L(G)$ the third node never obtains a word. Thus, $\lambda \in L(\mathcal{N})$ if and only if $\lambda \in L(G)$.

We now show that if a word $w \in T^+$ is accepted by the network then, at some time, the symbol S enters the third node and then a derivation $S \Longrightarrow^* w$ in G has been simulated reversely by reducing w to S in the network.

The computation starts in the first node with a word $w \in T^+$. The word can only be accepted if it can be reduced to S or λ (only these can enter the third node). No word can be reduced to λ, because if a word enters the deletion node, then it contains at least two letters A and $D_{p,4}$ for a rule $p = A \to CD$ and only $D_{p,4}$ can be deleted before the word moves back to the first node. Hence, if a word is accepted then it can be reduced to S (after the last reduction step, it moves from the first or second node to the output node).

In the sequel, A, B, C, D denote non-terminals, x denotes a non-terminal or terminal symbol and $w_1w_2 \in (N \cup T)^*$.

We now consider a word $w = w_1xw_2$ in the first node after an even number of computation steps (the next step is an evolution step). If we can apply a rule $x \to x_{p,0} \in R_1$

to w then we obtain the word $w_1 x_{p,0} w_2$ which does not leave the node (because it does not pass the output filter). If now another rule than $x_{p,0} \to A$ is applied, then the word passes the output filter and leaves the network. If $x_{p,0} \to A$ is applied, then we obtain the word $w_1 A w_2$ which does not leave the node and the derivation $w_1 A w_2 \Longrightarrow w_1 x w_2$ is possible in G (due to the construction of the set R_1).

Let us now consider a word $w = w_1 C D w_2$ in the first node after an even number of computation steps (the next step is an evolution step). If a rule of R_1 is applied then we have the situation described for the word $w = w_1 x w_2$. If we can apply another rule, then we have one of the following cases:

Case 1. Application of a rule $D \to D_{p,2} \in R_3$.
This leads to the word $w_1 C D_{p,2} w_2$ in the first node, which is then sent out. Since the other nodes do not accept it, the word is lost.

Case 2. Application of a rule $C \to C_{p,1} \in R_2$.
This leads to the word $w_1 C_{p,1} D w_2$ in the first node which is kept in the node. If the next evolution step does not yield the word $w_1 C_{p,1} D_{p,2} w_2$, then the word disappears in the next communication step. (Here is the reason why the rules in R_1 make the 'detour' via intermediate symbols. If there would be direct rules $x \to A$, we could apply them here and had more cases to discuss.) Let us assume, we obtain $w_1 C_{p,1} D_{p,2} w_2$, then this word is kept in the first node. The next evolution step yields $w_1 C_{p,3} D_{p,2} w_2$ or the word is lost. Also this word remains in the node. The fourth evolution step leads to $w_1 C_{p,3} D_{p,4} w_2$ or the word is lost. This word remains in the node, too. The fifth evolution step leads to $w_1 A D_{p,4} w_2$ or the word is lost. There are two possibilities for the rule p that belongs to $D_{p,4}$.

Case 2.1. $p = A \to CD$. In this case, the word $w_1 A D_{p,4} w_2$ is sent out and caught by the second node. The second node deletes the symbol $D_{p,4}$. In the next communication step, the word $w_1 A w_2$ is sent back to the first node. The described six evolution steps (together with the corresponding communication steps) represent the inverse of the derivation $w_1 A w_2 \Longrightarrow w_1 C D w_2$ in G.

Case 2.2. $p = AB \to CD$. In this case, the word $w_1 A D_{p,4} w_2$ remains in the first node. The next evolution step yields $w_1 A B w_2$ or a word that is lost, because applying any rule of $R_1 \cup R_2 \cup R_3$ (rules of R_5, R_6, R_7 and other rules of R_4 are not applicable) leads to a word which passes the output filter but no input filter. The word $w_1 A B w_2$ remains in the first node. The described six evolution steps (the communication steps have no effect) represent the inverse of the derivation $w_1 A B w_2 \Longrightarrow w_1 C D w_2$ in G.

Hence, in this case, the derivation $w_1 A w_2 \Longrightarrow w_1 C D w_2$ or $w_1 A B w_2 \Longrightarrow w_1 C D w_2$ (which in G is obtained by the initially chosen rule p) is simulated reversely.

Other rules are not applicable to the word w.

By the case distinction above, we have shown that, for every reduction $w \Longrightarrow v$ in the network $\mathcal{N}$, the derivation $v \Longrightarrow w$ is possible in the grammar G. Hence, if a word w can be reduced to S, then the derivation $S \Longrightarrow^* w$ exists in G. Together, we obtain that if a

word w is accepted by the network $\mathcal{N}$ then it is generated by the grammar G.

With the first part of the proof, we obtain $L(G) = L_{\mathrm{w}}(\mathcal{N}) = L_{\mathrm{s}}(\mathcal{N}) = L$. □

In generating networks, only substitution nodes yield not more than finite languages. But accepting networks can accept infinitely many input words. Surprisingly, even every context-sensitive language can be accepted by a network with only one substitution node (and one output node without rules). The trick is that the substitution processor can simulate the deletion by marking symbols as deleted. The filters then 'ignore' the deletion markers.

Theorem 3. *For any context-sensitive language L, there is an accepting network $\mathcal{S}$ of evolutionary processors with exactly one substitution node and one output node without rules that weakly and strongly accepts the language L:*

$$L = L_{\mathrm{w}}(\mathcal{S}) = L_{\mathrm{s}}(\mathcal{S}).$$

Proof. Let L be a context-sensitive language and $G = (N, T, P, S)$ be a grammar in Kuroda normal form with $L(G) = L$. The network $\mathcal{S}$ is constructed similarly to the network $\mathcal{N}$ in the proof of Theorem 2.

Let $R_1, R_2, \ldots, R_7$ be the sets used for $\mathcal{N}$ and let R_8 be the additional set

$$R_8 = \{\, D_{p,4} \to \textvisiblespace \mid p = A \to CD \in P,\ A, B, C, D \in N \,\}.$$

We construct a network of evolutionary processors

$$\mathcal{S} = (T, V^{\mathcal{S}}, (M_1^{\mathcal{S}}, \emptyset, O_1^{\mathcal{S}}), (\emptyset, I_2^{\mathcal{S}}, \emptyset), \{\,(1,2)\,\}, 1, \{2\})$$

with

$$V^{\mathcal{S}} = N \cup T \cup \{\textvisiblespace\} \cup \bigcup_{p=A\to x} \{x_{p,0}\} \cup \bigcup_{\substack{p=A\to CD \\ p=AB\to CD}} \{C_{p,1}, D_{p,2}, C_{p,3}, D_{p,4}\},$$

$$M_1^{\mathcal{S}} = R_1 \cup R_2 \cup R_3 \cup R_4 \cup R_5 \cup R_6 \cup R_7 \cup R_8,$$

$$O_1^{\mathcal{S}} = \{\textvisiblespace\}^*\{S\}\{\textvisiblespace\}^* \cup \{\lambda\} \cup V^* \setminus ((N \cup T \cup \{\textvisiblespace\})^* \bar{O}' (N \cup T \cup \{\textvisiblespace\})^*),$$

where

$$\begin{aligned}
\bar{O}^{\mathcal{S}} = {} & \{\, x_{p,0} \mid p = A \to x \in P,\ A \in N,\ x \in N \cup T \,\} \\
& \cup \{\, C_{p,1} \mid p = A \to CD \in P \text{ or } p = AB \to CD \in P \,\} \\
& \cup \{\, C_{p,1}\varepsilon D_{p,2} \mid p = A \to CD \in P \text{ or } p = AB \to CD \in P,\ \varepsilon \in \{\textvisiblespace\}^* \,\} \\
& \cup \{\, C_{p,3}\varepsilon D_{p,2} \mid p = A \to CD \in P \text{ or } p = AB \to CD \in P,\ \varepsilon \in \{\textvisiblespace\}^* \,\} \\
& \cup \{\, C_{p,3}\varepsilon D_{p,4} \mid p = A \to CD \in P \text{ or } p = AB \to CD \in P,\ \varepsilon \in \{\textvisiblespace\}^* \,\} \\
& \cup \{\, A\varepsilon D_{p,4} \mid p = A \to CD \in P \text{ or } p = AB \to CD \in P,\ \varepsilon \in \{\textvisiblespace\}^* \,\} \\
& \cup \{\, \lambda \,\},
\end{aligned}$$

and

$$I_2^{\mathcal{S}} = \begin{cases} \{\sqcup\}^*\{S\}\{\sqcup\}^* \cup \{\lambda\} & \text{if } \lambda \in L, \\ \{\sqcup\}^*\{S\}\{\sqcup\}^* & \text{otherwise.} \end{cases}$$

The network $\mathcal{S}$ has only one output node. Therefore, there is no difference between weak and strong acceptance, and we write $L(\mathcal{S})$ for the language accepted by the network $\mathcal{S}$.

The network $\mathcal{S}$ behaves almost in the same way as the network $\mathcal{N}$ in the proof of Theorem 2 does. The main difference is that symbols are not deleted but marked by the special symbol $\sqcup$. This can be done by the single substitution processor as well (using rules of R_8). When simulating a derivation $w \Longrightarrow v$ in G, we have to take into account that the corresponding word in the substitution node may contain gaps in form of several occurrences of the special symbol $\sqcup$. This is realized by the new formulation of the set $\bar{O}^{\mathcal{S}}$ of such subwords that are forbidden to leave the node.

An input word $w \in T^+$ can be reduced to a word $s \in \{\sqcup\}^*\{S\}\{\sqcup\}^*$ if and only if w is generated by the grammar G. Then and only then, s moves to the output node. If the input word is λ, then it is not modified in the first node but sent out. The second node receives the word if $\lambda \in L$. If $\lambda \notin L$ then the output node does not receive anything.

Hence, we have proved $L(G) = L_{\mathrm{w}}(\mathcal{S}) = L_{\mathrm{s}}(\mathcal{S}) = L$. □

This number of processors is optimal since the input node and output node have to be different (otherwise every input word would be accepted).

4. Networks of Non-Deleting Nodes

The main difference between context-sensitive and non-context-sensitive grammars is that, in arbitrary phrase structure grammars, erasing rules (λ-rules) are allowed. In order to simulate a λ-rule in reverse direction, we introduce an insertion node.

Theorem 4. *For any recursively enumerable language L, there is an accepting network $\mathcal{N}$ of evolutionary processors with exactly one substitution node, one insertion node and one output node without rules that weakly and strongly accepts the language L:*

$$L = L_{\mathrm{w}}(\mathcal{N}) = L_{\mathrm{s}}(\mathcal{N}).$$

Proof. Let L be a recursively enumerable language and $G = (N, T, P, S)$ be a grammar in Kuroda normal form with $L(G) = L$.

The idea of the proof is to extend the network $\mathcal{S}$ constructed in the proof of Theorem 3 by an inserting processor who is responsible for the reverse simulation of λ-rules, see Figure 1.

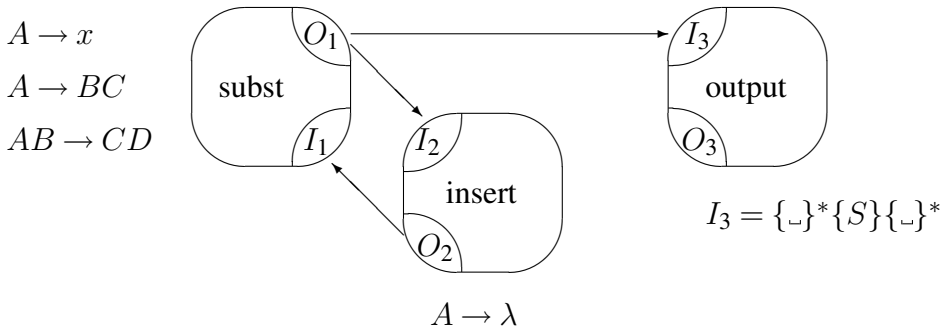

Figure 1: A network for simulating a grammar in Kuroda normal form

For a formal definition, we have to implement that the insertion node gets the opportunity to do something (the substitution processor must indicate that a word can pass to the insertion node).

We construct a network of evolutionary processors

$$\mathcal{N} = (T, V^{\mathcal{N}}, (M_1^{\mathcal{N}}, I_1^{\mathcal{N}}, O_1^{\mathcal{N}}), (\emptyset, I_2^{\mathcal{N}}, \emptyset), (M_3^{\mathcal{N}}, I_3^{\mathcal{N}}, O_3^{\mathcal{N}}), \{(1,2),(1,3),(3,1)\}, 1, \{2\})$$

with

$$\begin{aligned}
V^{\mathcal{N}} &= V^{\mathcal{S}} \cup \{ x' \mid x \in N \cup T \}, \\
M_1^{\mathcal{N}} &= M_1^{\mathcal{S}} \cup \{ x \to x' \mid x \in N \cup T \} \cup \{ x' \to x \mid x \in N \cup T \}, \\
I_1^{\mathcal{N}} &= I_3^{\mathcal{N}} = O_3^{\mathcal{N}} = (N \cup T \cup \{␣\})^* \{ x' \mid x \in N \cup T \} (N \cup T \cup \{␣\})^*, \\
O_1^{\mathcal{N}} &= O_1^{\mathcal{S}}, \\
I_2^{\mathcal{N}} &= I_2^{\mathcal{S}}, \\
M_3^{\mathcal{N}} &= \{ \lambda \to A \mid A \to \lambda \in P \},
\end{aligned}$$

where $V^{\mathcal{S}}$, $M_1^{\mathcal{S}}$, $O_1^{\mathcal{S}}$, and $I_2^{\mathcal{S}}$ are defined as in the proof of Theorem 3.

Between two simulation phases, the substitution node can mark a symbol such that the word can leave the node and enter the insertion node. This processor inserts a nonterminal that belongs to a λ-rule of the grammar G and returns the word to the substitution node. This processor then has to unmark the primed symbol. If marking or unmarking is not performed in the correct moment, the word will be lost. Due to the definition of the filters, we can connect all nodes with each other (to obtain a complete graph) without changing the behaviour of the network. □

5. Networks without Substitution Processors

In [1], we have shown that every recursively enumerable language can be generated by a network of one inserting processor and one deleting processor. Similar to the proof of this statement, we can prove the following result.

Theorem 5. *For any recursively enumerable language L, there is an accepting network $\mathcal{N}$ of evolutionary processors with exactly one insertion node, one deletion node and one output node without rules that weakly and strongly accepts the language L:*

$$L = L_{\mathrm{w}}(\mathcal{N}) = L_{\mathrm{s}}(\mathcal{N}).$$

Proof. Let L be a recursively enumerable language and $G = (N, T, P, S)$ be a grammar in Kuroda normal form with $L(G) = L$.

We define the sets of partial prefixes and partial suffixes of a word u by

$$\begin{aligned} \mathit{PPref}(u) &= \{x \mid u = xy,\ |y| \geq 1\}, \\ \mathit{PSuf}(u) &= \{y \mid u = xy,\ |x| \geq 1\}, \end{aligned}$$

respectively.

Let $V = N \cup T$ and $V_{\text{␣}} = V \cup \{\text{␣}\}$. We define a homomorphism $h : V^* \to V_{\text{␣}}^*$ by $h(a) = a$ for $a \in T$ and $h(A) = A\text{␣}$ for $A \in N$ and set

$$W = \{\, h(w) \mid w \in V^* \,\}.$$

We construct the following network

$$\mathcal{N} = (T, X, (M_1, I_1, O_1), (M_2, I_2, O_2), (\emptyset, \{S\text{␣}\}, \emptyset), E, 1, \{3\})$$

of evolutionary processors with

$$\begin{aligned} X &= V_{\text{␣}} \cup \bigcup_{p \in P} \{p_1, p_2, p_3, p_4\}, \\ M_1 &= \{\, \lambda \to \text{␣} \,\} \cup \{\, \lambda \to p_i \mid p \in P, 1 \leq i \leq 4 \,\} \cup \{\, \lambda \to A \mid A \in N \,\}, \\ I_1 &= W \setminus \{\, S\text{␣} \,\}, \\ O_1 &= X^* \setminus (W R_{1,1} W), \\ M_2 &= \{\, p_i \to \lambda \mid p \in P, 1 \leq i \leq 4 \,\} \cup \{\, x \to \lambda \mid x \in V_{\text{␣}} \,\}, \\ I_2 &= W R_{1,2} W, \\ O_2 &= X^* \setminus (W R_{2,2} W), \\ E &= \{(1,2), (2,1), (2,3)\} \end{aligned}$$

where

$$\begin{aligned} R_{1,1} = \bigcup_{p = u \to v \in P} & (\{p_1 h(v), p_1 h(v) p_2, p_1 p_3 h(v) p_2, p_1 p_3 h(v) p_2 p_4\} \\ & \cup \{p_1 p_3\} \mathit{PSuf}(h(u)) \{h(v) p_2 p_4\}) \\ R_{1,2} = \{\, & p_1 p_3 h(uv) p_2 p_4 \mid p = u \to v \in P \,\}, \\ R_{2,2} = \bigcup_{p = u \to v \in P} & (\{p_1 p_3 h(u)\} \mathit{PPref}(h(v)) \{p_2 p_4\} \cup \{p_3 h(u) p_2 p_4, p_3 h(u) p_4, h(u) p_4\}). \end{aligned}$$

The reverse simulation of the application of a rule $p = a_1 \dots a_s \to b_1 \dots b_t$ to a sentential form $\alpha a_1 \cdots a_s \beta$ with $x = h(\alpha)$ and $y = h(\beta)$ has the following form.

In the insertion node, we have

$$\begin{aligned} xh(b_1)\dots h(b_t)y &\Longrightarrow xp_1h(b_1)\dots h(b_t)y \\ &\Longrightarrow xp_1h(b_1)\dots h(b_t)p_2y \\ &\Longrightarrow xp_1p_3h(b_1)\dots h(b_t)p_2y \\ &\Longrightarrow xp_1p_3h(b_1)\dots h(b_t)p_2p_4y \\ &\Longrightarrow xp_1p_3\sqcup h(b_1)\dots h(b_t)p_2p_4y \\ &\Longrightarrow xp_1p_3a_s\sqcup h(b_1)\dots h(b_t)p_2p_4y \\ &\Longrightarrow^* xp_1p_3a_2\sqcup\dots a_s\sqcup(b_1)\dots h(b_t)p_2p_4y \\ &\Longrightarrow xp_1p_3\sqcup a_2\sqcup\dots a_s\sqcup h(b_1)\dots h(b_t)p_2p_4y \\ &\Longrightarrow xp_1p_3a_1\sqcup\dots a_s\sqcup h(b_1)\dots h(b_t)p_2p_4y. \end{aligned}$$

This word leaves the insertion node and enters the deletion node. There, the evolution continues to

$$\begin{aligned} xp_1p_3a_1\sqcup\dots a_s\sqcup h(b_1)\dots h(b_t)p_2p_4y &\Longrightarrow^{|h(b_t)|} xp_1p_3a_1\sqcup\dots a_s\sqcup h(b_1)\dots h(b_{t-1})p_2p_4y \\ &\Longrightarrow^* xp_1p_3a_1\sqcup\dots a_s\sqcup h(b_1)p_2p_4y \\ &\Longrightarrow^{|h(b_1)|} xp_1p_3a_1\sqcup\dots a_s\sqcup p_2p_4y \\ &\Longrightarrow xp_3a_1\sqcup\dots a_s\sqcup p_2p_4y \\ &\Longrightarrow xp_3a_1\sqcup\dots a_s\sqcup p_4y \\ &\Longrightarrow xa_1\sqcup\dots a_s\sqcup p_4y \\ &\Longrightarrow xa_1\sqcup\dots a_s\sqcup y. \end{aligned}$$

This word leaves the node and enters the output node if it is $S\sqcup$ (corresponding to the axiom of the grammar G) or it enters the insertion node for the next simulation phase. Only those words remain in the network that are obtained in the sequence described above; all other words get lost. □

Every regular language R is accepted by a network of two nodes where none of the nodes contains any rules. The input node keeps all words not belonging to R; all words belonging to R move to the output node which accepts all arriving words. Hence, the network accepts exactly the language R. Such networks are optimal with respect to the number of processors, because input node and output node have to be different. Otherwise the network would accept every input word.

Corollary 6. *Every regular language can be accepted by a pure deleting or pure inserting network with two nodes. This number of processors is optimal.*

Pure deleting networks can accept non-context-free languages.

Lemma 7. *There is a network of deletion nodes only that accepts the language*

$$L = \{\, a^n b^n c^n \mid n \geq 1 \,\}.$$

Proof. Let $T = \{\, a, b, c \,\}$ and $O = \{a\}^+\{b\}^+\{c\}^+$. We construct the following network

$$\mathcal{N} = (T, T, N_{\text{in}}, N_a, N_b, N_c, N_{\text{out}}, E, 1, \{5\})$$

of deletion processors with the nodes $N_{\text{in}} = (\emptyset, \emptyset, O)$, $N_x = (\{\, x \to \lambda \,\}, T^*, O)$ for $x \in T$ and $N_{\text{out}} = (\emptyset, \{\, abc \,\}, \emptyset)$, and the set $E\,\{\,(1,5),(1,2),(2,3),(3,4),(4,2),(4,5)\,\}$ of edges. The network is illustrated in Figure 2.

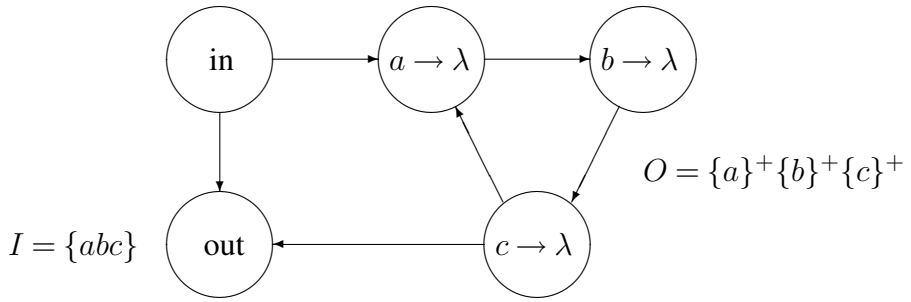

Figure 2: A deleting network for accepting the language $\{\, a^n b^n c^n \mid n \geq 1 \,\}$

Every word which has not the form $a^p b^q c^r$ with $p \geq 1$, $q \geq 1$, $r \geq 1$, remains in the first node for ever. The word abc will be sent directly to the output node and is accepted. A word $a^p b^q c^r$ with $p \geq 1$, $q \geq 1$, $r \geq 1$, and $pqr > 1$ is sent into the 'cycle' where one letter of a, b and c is deleted. The word moves on to the next node only if there is one letter of each kind left. If the word is stuck in a node, then it was abc when it entered the node which deletes a or p, q and r were not equal. If the word leaves the node which deletes c, then it has the form $a^{p-1} b^{q-1} c^{r-1}$. If this word is abc it goes to the output node and the input word is accepted. Otherwise it starts same cycle. Hence, a word is accepted if and only if it belongs to the language L. □

In this manner, networks can be constructed for similarly structured languages. They are accepted after a cyclic deletion process.

Also pure inserting networks can accept non-context-free languages.

Lemma 8. *There is a network of insertion nodes only that accepts the language*

$$L = \{\, a^n b^n c^n \mid n \geq 1 \,\}.$$

Proof. Let $T = \{\, a, b, c \,\}$, $V = T \cup \{\, a', b', c' \,\}$, and

$$O = \{aa'\}^*\{a\}^+\{bb'\}^*\{b\}^+\{cc'\}^*\{c\}^+.$$

We construct the following network

$$\mathcal{N} = (V, T, N_{\text{in}}, N_a, N_b, N_c, N_{\text{out}}, E, 1, \{5\})$$

of insertion processors with the nodes $N_{\text{in}} = (\emptyset, \emptyset, O)$, $N_x = (\{\lambda \to x'\}, V^*, O)$ for $x \in T$ and $N_{\text{out}} = (\emptyset, \{aa'\}^+\{bb'\}^+\{cc'\}^+ \cup \{abc\}, \emptyset)$, and the set

$$E\{(1,5),(1,2),(2,3),(3,4),(4,2),(4,5)\}$$

of edges. The network is illustrated in Figure 3.

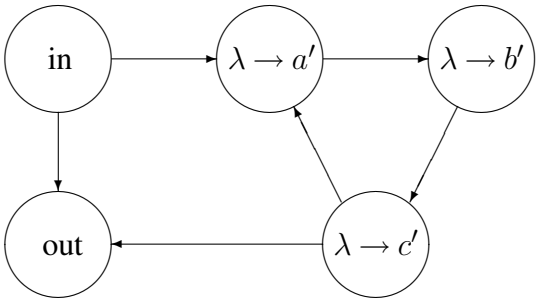

Figure 3: An inserting network for accepting the language $\{a^n b^n c^n \mid n \geq 1\}$

Every word which has not the form $a^p b^q c^r$ with $p \geq 1$, $q \geq 1$, $r \geq 1$, remains in the first node for ever. The word abc will be sent directly to the output node and is accepted. A word $a^p b^q c^r$ with $p \geq 1$, $q \geq 1$, $r \geq 1$, and $pqr > 1$ is sent into the 'cycle' where one letter of a, b and c is marked (by inserting a primed version after the first non-marked letter). The word moves on to the next node only if there is one unmarked letter of each kind left. If the word is stuck in a node, then a primed letter was inserted at a wrong position. If the word leaves the node which marks c, then it has the form $aa'a^{p-1}bb'b^{q-1}cc'c^{r-1}$. This word runs in the cycle until it stucks or it is transformed into the word $(aa')^p(bb')^q(cc')^r$. In the latter case, we have $p = q = r$, the word moves to the output node and, hence, the input word is accepted. Hence, a word is accepted if and only if it belongs to the language L. □

It remains as a task to characterize the family of languages that are accepted by pure deleting networks (which have only deleting processors) and pure inserting networks.

References

[1] A. ALHAZOV, J. DASSOW, C. MARTÍN-VIDE, YU. ROGOZHIN, and B. TRUTHE, On Networks of Evolutionary Processors with Nodes of Two Types. Submitted.

[2] A. ALHAZOV, C. MARTÍN-VIDE, and YU. ROGOZHIN, On the number of nodes in universal networks of evolutionary processors. *Acta Inf.* **43** (2006), 331–339.

[3] J. CASTELLANOS, P. LEUPOLD, and V. MITRANA, On the size complexity of hybrid networks of evolutionary processors. *Theor. Comput. Sci.* **330** (2005), 205–220.

[4] J. CASTELLANOS, C. MARTÍN-VIDE, V. MITRANA, and J. M. SEMPERE, Solving NP-complete problems with networks of evolutionary processors. In: *Proc. IWANN*, Lecture Notes in Computer Science **2084**, Springer-Verlag, Berlin, 2001, 621–628.

[5] J. CASTELLANOS, C. MARTÍN-VIDE, V. MITRANA, and J. M. SEMPERE, Networks of evolutionary processors. *Acta Informatica* **38** (2003), 517–529.

[6] E. CSUHAJ-VARJÚ and A. SALOMAA, Networks of parallel language processors. In: GH. PĂUN and A. SALOMAA (eds.), *New Trends in Formal Language Theory*. Lecture Notes in Computer Science **1218**, Springer-Verlag, Berlin, 1997, 299–318.

[7] J. DASSOW and V. MITRANA, Accepting Networks of Non-Increasing Evolutionary Processors. In: I. PETRE and G. ROZENBERG (eds.), *Proceedings of NCGT 2008. Workshop on Natural Computing and Graph Transformations. September 8, 2008, Leicester, UK*. University of Leicester, 2008, 29–41.

[8] J. DASSOW and B. TRUTHE, On the Power of Networks of Evolutionary Processors. In: J. DURAND-LOSE and M. MARGENSTERN (eds.), *Machines, Computations and Universality, MCU 2007, Orléans, France, September 10–13, 2007, Proceedings*. Lecture Notes in Computer Science **4664**, Springer, 2007, 158–169.

[9] S. E. FAHLMANN, G. E. HINTON, and T. J. SEIJNOWSKI, Massively parallel architectures for AI: NETL, THISTLE and Boltzmann machines. In: *Proc. AAAI National Conf. on AI*, William Kaufman, Los Altos, 1983, 109–113.

[10] W. D. HILLIS, *The Connection Machine*. MIT Press, Cambridge, 1985.

[11] F. MANEA and V. MITRANA, All NP-problems can be solved in polynomial time by accepting hybrid networks of evolutionary processors of constant size. *Information Processing Letters* **103** (2007) 3, 112–118.

[12] M. MARGENSTERN, V. MITRANA, and M. J. PÉREZ-JIMÉNEZ, Accepting Hybrid Networks of Evolutionary Processors. In: C. FERRETTI, G. MAURI, and C. ZANDRON (eds.), *10th International Workshop on DNA Computing*. Lecture Notes in Computer Science **3384**, Springer, 2005, 235–246.

[13] G. ROZENBERG and A. SALOMAA, *Handbook of Formal Languages*. Springer-Verlag, Berlin, 1997.