Available online at www.sciencedirect.com
SCIENCE DIRECT®

Theoretical Computer Science 321 (2004) 233–248

Theoretical Computer Science

www.elsevier.com/locate/tcs

Evolution and observation—a non-standard way to generate formal languages[☆]

Matteo Cavaliere, Peter Leupold*

Research Group in Mathematical Linguistics, Rovira i Virgili University, Pça. Imperial Tàrraco 1, Tarragona 43005, Spain

Received 9 September 2003; received in revised form 1 March 2004; accepted 11 March 2004

Abstract

In biology and chemistry a standard proceeding is to conduct an experiment, observe its progress, and then take the result of this observation as the final output. Inspired by this, we have introduced P/O systems (A. Alhazov, C. Martín-Vide, Gh. Păun, Pre-Proc. of the Workshop on Membrane Computing 2003, Tarrragona, Spain; http://pizarro.fll.urv.es/continguts/linguistica/proyecto/reports/wmc03.html), where languages are generated by multiset automata that observe the evolution of membrane systems.

Now we apply this approach also to more classical devices of formal language theory. Namely, we use finite automata observing the derivations of grammars or of Lindenmayer systems. We define several modes of operation for grammar/observer systems. In two of these modes a context-free grammar (or even a locally commutative context-free grammar) with a finite automaton as observer suffices to generate any recursively enumerable language. In a third case, we obtain a class of languages between the context-free and context-sensitive ones.
© 2004 Elsevier B.V. All rights reserved.

Keywords: Formal languages; Evolution; Observation

1. Introduction: when evolution is computation

We introduce and study a new way of looking at the concept of language generation different from the one in classical formal language theory. The so-called

[☆] This work was done, while both authors were funded by the Spanish Ministry of Culture, Education and Sport under the Programa Nacional de Formación de Profesorado Universitario (FPU).

* Corresponding author.

E-mail addresses: mc1.doc@estudiants.urv.es (M. Cavaliere), klauspeter.leupold@estudiants.urv.es (P. Leupold).

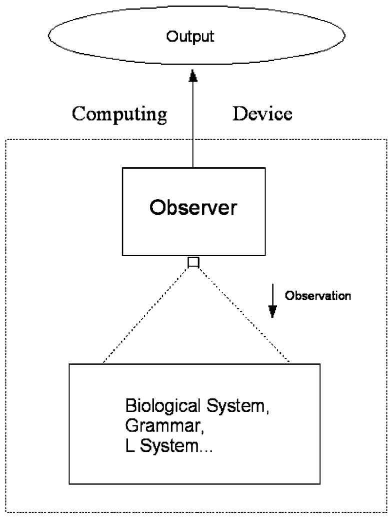

Fig. 1. The system/observer architecture.

grammar/observer (G/O) *system* that we consider in more detail is a particular case of a more general computing framework, the *system/observer* architecture. In earlier work [1] the authors have already joined together the concept of "behaviour" of a biological system (in the form of a membrane system) and the concept of "observer" to obtain a model of computation.

There it was shown how a computing device can be constructed using two less potent systems: the first one, which is a mathematical model of a biological system, simply "lives" (evolves), passing from one configuration to the next, producing in this way a "behaviour"; the second system, called "observer", is placed outside and watches the biological system. Following a set of rules the observer translates the behaviour of the underlying system into a "more readable" output: it associates a label to each configuration of the bio system and writes these labels according to their temporal order onto an output tape.

In this way the pair composed by the biological system and the observer can be considered a computing device, as described in Fig. 1, which also outlines the framework in the more general case. In this model the computation is divided between the biological system and the observer that is able to watch and to "interpret" the behaviour of the biological system. In this combination already rather simple membrane systems with only non-cooperating rules observed by multiset finite automata have been shown to suffice to obtain computational universality [1].

The general idea outlined above recalls what was already discussed by Rozenberg and Salomaa [6], who remarked that the result of a computation is already present in nature—we only need to look (in an appropriate way) at it. While in their case the observation is made applying a *gsm* machine to the language obtained using the (biologically inspired) *twin-shuffle* operation, in our framework the observer is not applied to the final result, but rather to the entire evolution of the system. This means as many applications as there are evolution steps.

Here, instead of considering biological systems, we investigate in the same way a more general system/observer architecture where the bio system is substituted by any system that, according to some specific rules, is able to move from one configuration to the next, producing in this way a behaviour; the observer is any recognizing machine that translates such a behaviour into a readable output.

We apply this general architecture in the field of Formal Language Theory, introducing and studying the class of G/O systems, where a grammar plays the role of the system and an automaton—be it a finite automaton or even a Turing machine—plays the role of the observer. A useful feature is to let the observer output a special label, if an undesired configuration/sentential form is observed. Any string containing it is then not considered as a valid result. This quite simple approach turns out to be very powerful.

We define several modes of generating languages for G/O systems and investigate more closely three of them. In the first mode G/O systems constituted by a context-free grammar and by a finite state automaton generate a family of languages between the context-free and context-sensitive ones. In the other two cases the same combination is even equivalent to a Turing machine. In one case this result can be further strengthened to using only a commutative context-free grammar instead of a conventional one. A similar result for E0L systems then follows directly from this.

2. Preliminaries

We now proceed to first recall and define the components that will make up the G/O systems used. Then we provide the formal definition of these systems and the languages they generate.

2.1. Formal language theory

In general, for all notions from general formal language theory we refer to the respective chapters of the Handbook of Formal Languages [7]. Only in a telegraphic manner we fix the notations we will use here. A generative grammar is a quadruple $G=(N,T,S,P)$, where N is the alphabet of non-terminals, T is the terminal alphabet disjoint from N, $S \in N$ is the start symbol and P is the set of productions. A derivation step of G leading from a sentential form w_1 to another one w_2 is denoted by $w_1 \Rightarrow_G w_2$ in the usual manner.

By *REG*, *LIN*, *CF*, *CS*, and *RE* we denote the classes of languages generated by regular, linear, context-free, context-sensitive, and unrestricted grammars, respectively.

2.2. Locally commutative context-free languages

We now define a variant of context-free grammars, for which the order of the symbols on the rules' right sides does not matter. Therefore we call them locally commutative context-free grammars. They are defined exactly as conventional context-free grammars, only with the additional condition that for any rule $A \to u$ in the rule set, also all rules $A \to \pi(u)$ are in the rule set, where $\pi(u)$ is a permutation of u.

Instead of writing a list of all these rules, we write each rule in the form $A \to [B_1, \dots, B_n]$, where the B_i can be terminals or non-terminals. The rule is applied by replacing A in a sentential form by a string of all the B_i in an arbitrary order. Right sides of length one can also be written without brackets. As one might expect, this variant is significantly weaker than general context-free grammars.

Theorem 1. *The family of locally commutative context-free languages is a proper subfamily of the family of context-free languages, i.e. $LCCF \subset CF$.*

Proof. The inclusion holds by definition. To show its properness, we consider the regular language $L = a^+b^+$ and suppose it is generated by a locally commutative context-free grammar. Instead of looking at the generation of certain symbols, we look at the generation of the borders between them. Every rule with a right side of length greater than one generates new such borders. Because all rules are context-free, the parts of the sentential form on the left and right of any border stay separate for the rest of the derivation.

Suppose now that L is generated by a locally commutative context-free grammar. Looking at a word $a^m b^n$ with $m, n \geqslant 1$, in some step the border $|_{ab}$ between the last a and the first b must be generated in a derivation by applying a rule that produces a string $U_1|_{ab}U_2$ with U_1, U_2 non-empty strings. Obviously, the U_1 derives to a string from a^+ and U_2 derives to a string from b^+. However, due to the commutative nature of the rules also the string U_2U_1 could have been derived and using the same rules for the rest of the derivation, a string containing a border $|_{ba}$ (hence a substring ba) could be derived. But the resulting word is not in L. Thus, there can be no locally commutative context-free grammar generating L. □

Because the language used in the proof is regular, we also see that the class $LCCF$ contains neither REG nor LIN. At the same time the language of all words with an equal number of occurrences of a and b over the corresponding two-letter alphabet can clearly be generated by a locally commutative context-free grammar. Because it is not linear, the $LCCF$ is not comparable with either of the classes REG and LIN.

2.3. Automata with singular output

There exist already many types of transducers mapping strings into other strings. For the observers as described in the introduction, however, we need a device mapping arbitrarily long strings into just one singular symbol. Therefore we define a special variant of finite automata: the set of states is labelled with the symbols of an output alphabet Σ or with λ. Any computation of the automaton produces as output the label of the state it halts in (we are not interested in accepting/not accepting computations and therefore also not interested in the presence of final states); because the observation of a certain string should always lead to a fixed result, we consider here only deterministic and complete automata.

This way an automaton with singular output is a tuple $\mathscr{A}=(Z,V,\Sigma,z_0,\delta,\sigma)$ with state set Z, input alphabet V, initial state $z_0 \in Z$, and a complete transition function δ as known from conventional finite automata; further there is the output alphabet Σ and a labelling function $\sigma : Z \mapsto \Sigma \cup \{\lambda\}$. The output of the automaton is the label of the state it stops in. For a string $w \in V^*$ and an automaton $\mathscr{A}$ we then write $\mathscr{A}(w)$ for this output; for a sequence $w_1,\dots,w_n$ of $n \geqslant 1$ strings over V^* we write $\mathscr{A}(w_1,\dots,w_n)$ for the string $\mathscr{A}(w_1)\cdots\mathscr{A}(w_n)$.

Moreover, we will often also want the observer to be able to reject some words. To realize this we simply choose a special symbol $\perp \notin \Sigma$ and an extended output alphabet $\Sigma_\perp = \Sigma \cup \{\perp\}$; σ then is a mapping from the set of states Z to $\Sigma_\perp \cup \{\lambda\}$. $\perp$ is produced, when a bad sentential form is observed and thus the entire sequence is to be rejected. Then, using the intersection with the set Σ^*, it is possible to filter out the strings produced containing the special symbol $\perp$.

The class of all (deterministic) automata with singular output will be denoted by FA_{O}. In the same way observers can be obtained from other well-known classes of automata such as pushdown automata, linear bounded automata or Turing machines.

2.4. *G/O systems*

Now we define the central notion of this paper, the *G/O system*. This is a special case of the general architecture proposed in the introduction.

A *G/O system* is a pair $\Omega=(G,\mathscr{A})$ constituted by a generative grammar $G=(N,T,S,P)$ and an observing automaton with singular output $\mathscr{A}=(Z,V,\Sigma,z_0,\delta,\sigma)$ with output alphabet Σ, which then is also the output alphabet of the entire system. The automaton's input alphabet must be the union of N and T from the grammar to make the desired interaction possible, i.e. $V=N\cup T$.

We distinguish three different modes of generation that define three different models of *G/O systems*:

(i) writing a non-empty output in every step (*always writing G/O system*),
(ii) writing a non-empty output in every step after an intialization phase of writing only λ (*initial G/O system*), and
(iii) changing between empty and non-empty output in an arbitrary manner (*free G/O system*).

In the case of an *always writing G/O system* Ω the language generated is

$$\begin{aligned} L_{\mathrm{a}}(\Omega) = \{\mathscr{A}(w_1,w_2,\dots,w_n) | S = w_0 \Rightarrow_G w_1 \Rightarrow_G \cdots \Rightarrow_G w_n \\ \wedge w_n \in T^* \wedge \forall i \in \{1,\dots,n\}[\mathscr{A}(w_i) \neq \lambda]\}. \end{aligned}$$

Note that the very first sentential form, which is always the starting symbol, is excluded from the observation. Otherwise all words in $L(\Omega)$ would start with the same letter, if the observer was deterministic. The best way to ensure the last condition, i.e. that λ is never written as output, is of course to define the observer in such a way that it can never produce empty output.

For an *initial G/O system* the output is defined as

$$L_{\mathrm{i}}(\Omega) = \{\mathscr{A}(w_0, w_1, \ldots, w_n) | S = w_0 \Rightarrow_G w_1 \Rightarrow_G \cdots \Rightarrow_G w_n \wedge w_n \in T^* \\ \wedge \forall i \in \{1, \ldots, n\}[\mathscr{A}(w_0, w_1, \ldots, w_{i-1}) \neq \lambda \text{ implies } \mathscr{A}(w_i) \neq \lambda]\}.$$

Finally, a *free G/O system* generates a language in the following non-restricted manner:

$$L_{\mathrm{f}}(\Omega) = \{\mathscr{A}(w_0, w_1, \ldots, w_n) | S = w_0 \Rightarrow_G w_1 \Rightarrow_G \cdots \Rightarrow_G w_n \wedge w_n \in T^*\}.$$

Thus in all three cases the language contains all those words which the observer can produce during the possible terminating derivations of the underlying grammar. Derivations which do not terminate do not produce a valid output; this means that we only take into account finite words. Of course, by considering the other case of non-terminating derivations the G/O systems could also be used to generate languages of infinite words.

We have already mentioned and will actually mainly investigate the variant, where the observer can also produce a special symbol $\perp \notin \Sigma$; whenever it appears in a word, this word should not be considered for the output language, thus we take

$$L_{\perp,\mathrm{a}}(\Omega) = L_{\mathrm{a}}(\Omega) \cap \Sigma^*.$$

Analogously the languages $L_{\perp,\mathrm{i}}(\Omega)$ and $L_{\perp,\mathrm{f}}(\Omega)$ are defined.

For a class $\mathscr{G}$ of grammars and a class $\mathscr{O}$ of observers, $\mathscr{L}_{\mathrm{a}}(\mathscr{G}, \mathscr{O})$, $\mathscr{L}_{\mathrm{i}}(\mathscr{G}, \mathscr{O})$, $\mathscr{L}_{\mathrm{f}}(\mathscr{G}, \mathscr{O})$, $\mathscr{L}_{\perp,\mathrm{a}}(\mathscr{G}, \mathscr{O})$, $\mathscr{L}_{\perp,\mathrm{i}}(\mathscr{G}, \mathscr{O})$, and $\mathscr{L}_{\perp,\mathrm{f}}(\mathscr{G}, \mathscr{O})$ denote the classes of all languages generated by G/O systems with grammars from $\mathscr{G}$ and observers from $\mathscr{O}$ in the respective modes. Quite obviously we obtain for fixed classes of grammars $\mathscr{G}$ and observers $\mathscr{O}$ the inclusions

$$\mathscr{L}_{\mathrm{a}}(\mathscr{G}, \mathscr{O}) \subseteq \mathscr{L}_{\mathrm{i}}(\mathscr{G}, \mathscr{O}) \subseteq \mathscr{L}_{\mathrm{f}}(\mathscr{G}, \mathscr{O})$$

and the same for the variants where the special symbol $\perp$ can be written.

In a completely analogous way, *L/O systems* can be defined, where the grammar is replaced by an L system. Here we refrain from doing so more formally, because in the sequel this notion will only be used on a rather informal level.

For simplicity, in what follows, we present only the mappings that the observers define, without giving a real implementation (in terms of finite automata) for them.

3. Always writing G/O systems

The first mode of generation we will investigate is the one of writing an output in every step, i.e. we consider the model of always writing G/O systems. This is maybe the most natural one, since in most cases the observation of an experiment should be complete, at least if about the outcome nothing is known beforehand. We will consider here the variant where also $\perp$ can be written as output.

We can see that every context-free language L is in $\mathscr{L}_{\perp,\mathrm{a}}(CF, FA_{\mathrm{O}})$. For this consider the Greibach normal form. There all right sides of rules are elements of TN^*; this means that in every step exactly one terminal is produced. Since the grammar is still context-free, there is a leftmost derivation for every word in L. In this derivation all sentential forms except the initial S are strings over T^+N^*. An observer can check that a derivation produces only sentential forms of this structure. Then it can output the rightmost terminal for each one of these sentential forms, and the result equals the string derived by the original grammar.

However, $\mathscr{L}_{\perp,\mathrm{a}}(CF, FA_{\mathrm{O}})$ is bigger than only CF. As an example for a non-context-free language from this class we present $\{a^nb^nc^n \mid n>0\}$. The grammar for this is

$$G = (\{S, A, B, C\}, \{t\}, S, \{S \to A, A \to AB, A \to C, B \to C, C \to t\}).$$

The derivations whose observations will result in the output of words $a^nb^nc^n$ are the ones of the form

$$S \Rightarrow A \overset{n-1}{\Rightarrow} AB^{n-1} \Rightarrow CB^{n-1} \overset{n-1}{\Rightarrow} C^n \Rightarrow tC^{n-1} \overset{n-1}{\Rightarrow} t^n.$$

To produce the output and rule out all other derivations, the observing automaton $\mathscr{A}$ will realize the following mapping from the set of sentential forms of G into $\{a, b, c, \perp\}$:

$$\mathscr{A}(w) = \begin{cases} a & \text{if } w \in AB^*, \\ b & \text{if } w \in C^+B^*, \\ c & \text{if } w \in t^+C^*, \\ \perp & \text{else.} \end{cases}$$

While $\{a^nb^nc^n \mid n>0\}$ is still semi-linear, also the language $\{a^{2^n} \mid n>0\}$, which is not semi-linear, lies in the class $\mathscr{L}_{\perp,\mathrm{a}}(CF, FA_{\mathrm{O}})$. To show this, we first recall that a binary tree of depth k has exactly $2^n - 1$ nodes, if the root is considered to already have depth one. Therefore a context-free grammar, which can have full binary trees as derivation trees, and an observer, which can check that only such derivations are made, can generate $\{a^{2^n} \mid n>0\}$ when interacting in a G/O system. For this example our grammar is $G = (\{S, A, B, C, T_1, T_2, T_3\}, \{t\}, S, P)$, where the set P of productions is $\{S \to A, A \to BB, B \to CC, C \to AA, A \to BT_1, B \to CT_2, C \to AT_3, A \to t, B \to t, C \to t,$
$T_1 \to t, T_2 \to t, T_3 \to t\}$.

Now, for example, derivations resulting in the outputs a^2 and a^8 are

$$S \Rightarrow A \Rightarrow t$$

and

$$S \Rightarrow A \Rightarrow BB \Rightarrow CCB \Rightarrow CCCT_2 \Rightarrow tCCT_2 \Rightarrow ttCT_2 \Rightarrow tttT_2 \Rightarrow tttt,$$

respectively. The conditions that the observer has to check (for putting out a in every step) are rather straightforward to see after the first example.

A sentential form containing any A must be completely changed to one from B^*, the same from B to C, and finally from C to A. This is done by use of the rules $A \to BB, B \to CC$ and $C \to AA$, respectively. To ensure that the entire sentential form is completely changed, the observer maps to a only the sentential forms of the form $A^+B^* \cup B^+C^* \cup C^+A^*$— others result in the output $\perp$. We notice that there are never more than two different non-terminals present at the same time.

To stop the derivation the rightmost non-terminal of the sentential form must produce the corresponding T_i, with $i \in \{1,2,3\}$, by using one of the rules $A \to BT_1$, $B \to CT_2$, or $C \to AT_3$, and then the only possible further steps are to derive all non-terminals to t. In these cases, the sentential forms mapped to a must be of the form $t^*A^*T_3 \cup t^*B^*T_1 \cup t^*C^*T_2 \cup t^+$.

Therefore the mapping of the observer is

$$\mathscr{A}(w) = \begin{cases} a & \text{if } w \in A^+B^* \cup B^+C^* \cup C^+A^* \cup \\ & \quad t^*A^*T_3 \cup t^*B^*T_1 \cup t^*C^*T_2 \cup t^+, \\ \perp & \text{else.} \end{cases}$$

We note here one difference between the two examples: while in the first case for a word of length n the workspace used by the grammar is $n/3$, in the second case the space is logarithmic in the length of the output.

Trying to characterize the class $\mathscr{L}_{\perp,\mathrm{a}}(CF, FA_{\mathrm{O}})$ more closely, we can easily see that it is contained in the class of context-sensitive languages. In fact, the G/O system must write a symbol of output at each step and then the total space used by such a system for the generation of a word w is bounded by a constant depending only on the context-free grammar. Therefore, every language generated by an always writing G/O system is context-sensitive by the workspace theorem [7].

Here we leave open the question, whether the classes of context-sensitive languages and $\mathscr{L}_{\perp,\mathrm{a}}(CF, FA_{\mathrm{O}})$ are identical, or whether the latter is properly included in the former. We do, however, suspect the following.

Conjecture 2. $CS \setminus \mathscr{L}_{\perp,\mathrm{a}}(CF, FA_{\mathrm{O}}) \neq \emptyset$.

To prove this one might search for a language whose generation by an LBA inevitably requires the entire (linearly bounded) space and at the same time heavily reuses all this space. Actually, even a lower bound of $n+1$ steps for the generation of a word of length n by an always writing G/O system would suffice. For example, the language over the alphabet $\{a\}$ consisting of all words of prime length appears as a good candidate for this.

4. Initial G/O systems

The second variant of G/O systems is called *initial G/O system* and in this model the sentential forms of an initial phase are mapped exclusively to λ. After the first non-empty output only non-empty outputs can be produced—looking back to the biochemical motivation of the concept of evolution and observer, this would correspond to a phase of initializing an experiment and then a phase of actual observation.

Such an initialization phase—which in our case is not restricted, but can be much longer than the actually observed phase—greatly enhances the power of our G/O systems. Indeed, with the same classes of grammars and observers as in Section 3 we obtain computational universality in this case. For the proof of this we first recall the Kuroda normal form of generative grammars: For every recursively enumerable language there exists a grammar generating it, which has only productions of the forms $A \to BC, A \to a, A \to \lambda, AB \to CD$, where the capital letters are non-terminals, and the lower-case letter stands for terminals.

Theorem 3. $\mathscr{L}_{\perp,\mathrm{i}}(CF, FA_{\mathrm{O}}) = RE$.

Proof. The inclusion from left to right is obvious, because every G/O system of the considered type can be simulated by a Turing machine. To show the opposite direction, we start from a grammar $G_1 = (N, T, S, P_1)$ in Kuroda normal form and construct an equivalent G/O system with a context-free grammar $G_2 = (N_2, T, S, P_2)$ and an FA_O observer $\mathscr{A}$ such that $L_{\perp,\mathrm{i}}(G_2, \mathscr{A}) = L(G_1)$.

For every derivation of G_1 resulting in a word w, grammar G_2 will derive a sentential form of nonterminals $\overline{a}$ corresponding to the letters a of w. Up to that point no output is written. Then, starting from the front, the non-terminals are rewritten to the corresponding terminals, and always the last one already converted is written to the output by the observer. Except for the last phase this follows very closely the lines of the universality proof for conditional grammars as given by Dassow and Păun [3].

Our starting point will be G_1. The rules of the forms $A \to BC, A \to a$, and $A \to \lambda$ from G_1 we can adopt for G_2 without any change. Only the rules $AB \to CD$ have to be simulated by the context-free grammar in several steps. For this we assume a one-to-one labelling from the set P_1' of all these rules in G_1 into $r_1, \dots, r_l$; without loss of generality we can assume that the four non-terminals of each such rule are distinct. Further we will use a set $\overline{T} = \{\overline{a} | a \in T\}$ of non-terminals and for every rule $r : AB \to CD$ a set $R_i = \{A_r, B_r, C_r, D_r\}$. Then we have

$$N_2 = N \cup \overline{T} \cup \bigcup_{r_i \in P_1'} R_i$$

as the set of non-terminals of G_2. The set of rules is

$$\begin{aligned} P_2 = {} & \{A \to BC, A \to \lambda | \text{productions of } P_1\} \\ & \cup \{A \to \overline{a} | A \to a \in P_1\} \\ & \cup \{A \to A_r, A_r \to C_r, C_r \to C, \\ & \quad B \to B_r, B_r \to D_r, D_r \to D | r : AB \to CD \in P_1'\} \\ & \cup \{\overline{a} \to a | a \in \Sigma\}. \end{aligned}$$

With this grammar any derivation of G_1 can be simulated, where all applications of rules of the forms $A \to BC, A \to a$, and $A \to \lambda$ can be done in one step, just as in the original grammar. Only when rules of the form $AB \to CD$ have to be simulated, the execution of the necessary steps in the correct order must be guaranteed by the observer such that only derivations possible already in G_1 can be realized. How all this is done can be seen again by looking at the mapping the observer $\mathscr{A}$ realizes:

$$\mathscr{A}(w) = \begin{cases} \lambda & \text{if } W \in N^*, \\ \lambda & \text{if } w \in N^*A_rN^* \text{ and } r: AB \to CD \in P_1', \\ \lambda & \text{if } w \in N^*A_rB_rN^* \text{ and } r: AB \to CD \in P_1', \\ \lambda & \text{if } w \in N^*C_rB_rN^* \text{ and } r: AB \to CD \in P_1', \\ \lambda & \text{if } w \in N^*C_rD_rN^* \text{ and } r: AB \to CD \in P_1', \\ \lambda & \text{if } w \in N^*D_rN^* \text{ and } r: AB \to CD \in P_1', \\ \lambda & \text{if } w \in \overline{T}^*N^*, \\ a_i & \text{if } w \in T^*a_i\overline{T}^*, \\ \perp & \text{else.} \end{cases}$$

As all the cases (except N^* and $\overline{T}^*N^*$, which have equal output λ, though) are disjoint and described in a regular way, clearly a finite automaton with singular output can realize this mapping. The structure of the case $T^*a_i\overline{T}^*$ guarantees that the resulting word is read from left to right and only after all non-terminals remaining in the case $\overline{T}^*N^*$ have been converted to non-terminals from $\overline{T}$. $\square$

Using this last theorem we can also give a characterization of the recursively enumerable languages in terms of always writing G/O systems. For this we need to recall the definition of the *left quotient* of one language with another. Given two languages L_1 and L_2 over a common alphabet Σ, the *left quotient* of L_1 with respect to L_2 is

$$L_2 \backslash L_1 = \{w \in \Sigma^* \mid \text{there is } x \in L_2 \text{ such that } xw \in L_1\}.$$

Using now the same construction as in the proof of Theorem 3, only writing a special letter c instead of λ, we arrive at the following result.

Corollary 4. *Every recursively enumerable language over an alphabet Σ is the left quotient of some $L_{\perp,\mathrm{a}}(G,\mathscr{A})$ with c^*, where $c \notin \Sigma$, G is a context-free grammar and $\mathscr{A}$ is a finite automaton with singular output with output alphabet $\Sigma \cup \{c\}$.*

5. Free G/O systems

An immediate corollary of Theorem 3 is the fact that $\mathscr{L}_{\lambda,\perp}(CF, FA_O) = RE$. In this section, however, we show how a G/O system composed of even less powerful components, namely a locally commutative context-free grammar and a finite state automaton is universal, if the output λ can be used without any restriction and if some derivations can be ignored by using the symbol $\perp$.

We give here only the sketch of the proof, because the idea is very similar to the proof given for the universality of a evolution/observation system composed by a membrane system, with non-cooperative rules, observed by a multiset finite automaton [1]. Essentially the membrane system's evolution must be sequentialized, therefore we concentrate here on the grammar.

The proof is based on the fact that every recursively enumerable language is accepted by a two-counter automaton [4]. These are automata with an input tape, which they can read from left to right only; they have two counters, the operations for which are increment by one, decrement by one, or remain unchanged. At each step, the automaton is in a certain state, a symbol from the input alphabet is read from the input tape, and it checks, if the values of two counters are zero. Then the automaton assumes a new state, moves the input head and the values of the two counters are changed, if required. An input word is accepted, if it is completely read, and the automaton stops in a final state with both counters being empty. So the transition function is

$$\delta : Q \times \mathbb{A} \times \{z, nz\} \times \{z, nz\} \mapsto Q \times \{\text{stay}, \text{right}\} \times \{+1, 0, -1\} \times \{+1, 0, -1\}.$$

This is the format we will refer to, when simulating a two-counter machine. The meanings of the components are the following: Q is the set of states, and $\mathbb{A}$ is the input alphabet; z and nz mean, respectively, that the counter is zero or not zero, *stay* and *right* are the movement orders for the head of the input tape, $+1, 0, -1$ are the orders for the changing of the value of the counters.

Two-counter automata serve very well for our purposes, because in contrast to a stack or a working tape a counter can be simulated without paying attention to the order of its elements; they are all the same and can therefore be distributed freely throughout the entire sentential form.

Theorem 5. $\mathscr{L}_{\perp,\mathrm{f}}(LCCF, FA_O) = RE$.

Proof. From an arbitrary two-counter automaton $\mathscr{C}$ accepting a language L, and thus for any recursively enumerable language L, we construct a G/O system $\Omega = (G, \mathscr{A})$ composed by a locally commutative context-free grammar G and a finite automaton with singular output $\mathscr{A}$, such that $L_{\perp,\mathrm{f}}(\Omega) = L$. The grammar will first generate the letters of the output word, then simulate a computation of $\mathscr{C}$; all this is done with the sentential form consisting only of non-terminals, i.e. everything remains rewriteable. In a final phase, G rewrites everything to terminals to end the derivation and thereby the computation of the entire G/O system. The observer's task is to guarantee that the grammar's derivation steps occur in the desired order.

Since $\mathscr{C}$ has only a finite number of transitions, we can label them starting from T_0; we will in the sequel speak of "transition T_t" or just "T_t" instead of "the transition with label T_t".

Now the locally commutative context-free grammar G for our system is the quadruple $(N, \{c, f\}, S, P)$ where

$$\begin{aligned} N = {} & \{S, \Delta_0, \Delta_1, \Delta_2, \Delta_3, M, M', M'', M_{\mathrm{pop}}, N, N', N'', N_{\mathrm{pop}}, \diamond\} \\ & \cup \{A_i, A_i', \underline{A}_i \mid a_i \in \mathbb{A}\} \\ & \cup \{T_t \mid t_j \text{ is a transition}\} \\ & \cup \{S_k \mid s_k \in Q\}. \end{aligned}$$

The set of productions of G is

$$
\begin{aligned}
P = \{&S \to [A_i, S], S \to [S_0, \Delta_0]\} \\
&\cup \{A_i \to A_i' | a_i \in \mathbb{A}\} \\
&\cup \{\Delta_0 \to \Delta_1, \Delta_1 \to \Delta_2, \Delta_2 \to \Delta_3, \Delta_3 \to \Delta_0\} \\
&\cup \{M \to M_{\text{pop}}, M \to M', M_{\text{pop}} \to \diamond, M' \to M, M'' \to M\} \\
&\cup \{N \to N_{\text{pop}}, N \to N', N_{\text{pop}} \to \diamond, N' \to N, N'' \to N\} \\
&\cup \{S_k \to S_k', S_k' \to \diamond | s_k \in Q\} \\
&\cup \{S_k \to f | s_k \text{ is a final state of } \mathscr{A}\} \\
&\cup \{X \to c | X \in N\} \\
&\cup \{T_t \to \diamond | T_t \text{ is a transition}\} \\
&\cup \{A_i' \to [S_k, M'', N'', T_t], \\
&\quad \underline{A}_i \to [S_k, M'', N'', T_t] | T_t \text{ is } \delta(s, a_i, \alpha, \beta) = [s_k, \text{ right}, +1, +1]\} \\
&\cup \{A_i' \to [S_k, M'', T_t], \\
&\quad \underline{A}_i \to [S_k, M'', T_t] | T_t \text{ is } \delta(s, a_i, \alpha, \beta) = [s_k, \text{ right}, +1, \eta]\} \\
&\cup \{A_i' \to [S_k, N'', T_t], \\
&\quad \underline{A}_i \to [S_k, N'', T_t] | T_t \text{ is } \delta(s, a_i, \alpha, \beta) = [s_k, \text{ right}, \eta, +1]\} \\
&\cup \{A_i' \to [S_k, T_t], \\
&\quad \underline{A}_i \to [S_k, T_t] | T_t \text{ is } \delta(s, a_i, \alpha, \beta) = [s_k, \text{ right}, \eta, \eta]\} \\
&\cup \{A_i' \to [S_k, M'', N'', T_t, \underline{A}_i], \\
&\quad \underline{A}_i \to [S_k, M'', N'', T_t, \underline{A}_i] | T_t \text{ is } \delta(s, a_i, \alpha, \beta) = [s_k, \text{ stay}, +1, +1]\} \\
&\cup \{A_i' \to [S_k, M'', T_t, \underline{A}_i], \\
&\quad \underline{A}_i \to [S_k, M'', T_t, \underline{A}_i] T_t \text{ is } \delta(s, a_i, \alpha, \beta) = [s_k, \text{ stay}, +1, \eta]\} \\
&\cup \{A_i' \to [S_k, N'', T_t, \underline{A}_i], \\
&\quad \underline{A}_i \to [S_k, N'', T_t, \underline{A}_i] | T_t \text{ is } \delta(s, a_i, \alpha, \beta) = [s_k, \text{ stay}, \eta, +1]\} \\
&\cup \{A_i' \to [S_k, T_t, \underline{A}_i], \\
&\quad \underline{A}_i \to [S_k, T_t, \underline{A}_i] | T_t \text{ is } \delta(s, a_i, \alpha, \beta) = [s_k, \text{ stay}, \eta, \eta]\} \\
&\quad \text{for all } a_i \in \mathbb{A}, s, s_k \in Q, \alpha, \beta \in \{z, nz\}, \eta \in \{0, -1\}.
\end{aligned}
$$

We describe a derivation in G that results in a valid output by $\mathscr{A}$ and along the way we hint to the conditions that this automaton has to check. We shall see that it will essentially suffice to check whether certain non-terminals are present zero, one, or more times (and these conditions can be represented using a regular expression and then checked by the observer).

To simulate a computation of the two-counter automaton accepting a word w, the grammar first produces in a regular derivation the word wS from S; here w stands for the word of nonterminals A_i corresponding to the word of letters a_i in the obvious way. Once W is generated, the rule $S \to S_0 \Delta_0$ introduces the initial state of $\mathscr{C}$ and Δ_0 to synchronize the further steps. The actual simulation of $\mathscr{C}$ can begin.

For a transition reading a new symbol, one A_j from the sentential form is selected to be read and marked by converting it to A'_j. Depending on whether the transition will pop from a counter or not, one M and one N are changed to M_{pop}, N_{pop} or M', N'. Ending this preparation phase, also the state symbol is primed.

After the application of the rule $\Delta_0 \to \Delta_1$, i.e. when all the mentioned symbols are present, and an A'_j has been "read", the observer produces the corresponding a_j as an output. This is the unique situation where $\mathscr{A}$ produces a non-empty output other than $\perp$.

Now the transition to be applied is selected, by applying the unique rule with the corresponding T_t in its right side. This is a very crucial step, because the compatibility of the transition and the symbols present must be checked by the observer: had the change of the stack been guessed right? Was the original state—still being present as S'_l—one in which T_t can be applied? We shall look at the conditions to be checked in some more detail and first recall that we can view a transition as an element of the set

$$Q \times \mathbb{A} \times \{z, nz\} \times \{z, nz\} \times Q \times \{\text{stay}, \text{right}\} \times \{+1, 0, -1\}, \times \{+1, 0, -1\}.$$

The S'_l and S_k present in the sentential form are the two elements from Q, the states before and after execution of the transition, respectively. Since they are unique for each transition T_j, their correctness can be checked. The letter from the input alphabet does not have to be checked, because any T_t is produced directly from the correct A_j. The emptiness of the counters is equivalent to the non-occurrence of M and M' and M_{pop} or the respective Ns. The correct move of the input head is already implicit in the rule producing T_t.

Moreover the observer must also check whether the update of the values of the counters is done correctly. Here the presence of only M_{pop} stands for decrement, the presence of only M' will result in an unchanged counter, and the presence of M' and M'' increments the counter, M'' alone increments an empty counter, and non-occurrence of all three means that there was an empty counter (no M) and it is not incremented. All other combinations of the three symbols are illegal, for the other counter the N's are checked analogously. Since every T_t effects a unique stack behaviour, this check does not require any additional states of $\mathscr{A}$. If not everything is alright, $\perp$ is written as output.

The next step in $\mathscr{C}$'s simulation is the production of Δ_2, which signals the beginning of the final phase, where all the symbols not necessary any more are cleaned up. In the remaining steps the stack symbols are either restored to M and N or converted to $\diamond$. Also T_t and S'_k become $\diamond$; this symbol is ignored by the observer in the sequel. Finally, when Δ_3 is present, $\mathscr{A}$ checks that this clean-up has completely been done. Now again we have arrived at a sentential form ready for the application of another transition, and Δ_0 can be produced.

In a very similar manner transitions not reading anything new from the input tape are simulated. Only in the first step no symbol A'_j is produced, but rather $\underline{A}_j$ is present from the start in the sentential form. It was already produced in the simulation of the last transition, which did not move the input head and therefore left $\underline{A}_j$ as a copy of the symbol it had read.

In contrast to the corresponding previous case no letter a_j must be produced as output, because the symbol read from the input tape has already been read in a previous step. In this case the $\underline{A}_j$ present selects the transition T_t to simulate (analogously to the A'_j in the previous case); then this transitions simulation continues with sentential forms of the same content as already explained for the latter case.

The two-counter automaton accepts a word, if it has read the entire input, the counters are empty and it is in a final state. Therefore for all final states a derivation to the terminal f exists. Whenever f and only non-terminals are present, the observer can check whether these conditions are satisfied (no M, N, A_i), and otherwise output $\perp$. Once f is present, no more transitions of $\mathscr{C}$ can be simulated and the observer maps every sentential form to λ. With a termination of the derivation taking all remaining non-terminals to c, the grammar stops and w has been produced as output.

Because the special symbol $\perp$ is introduced every time that the conditions checked by the observed are not respected and because of the synchronization provided by the Δ's, then every output word is also accepted by $\mathscr{C}$ and, on the other hand, every computation of $\mathscr{C}$ can be simulated in Ω. □

We remark the fact that in this proof a locally commutative context-free grammar works well, because the order of the symbols in the sentential forms observed by the automaton is not relevant. This is even true to the extent that the order given by the string could be completely abandoned. Thus the same result could be obtained for context-free (in that case equivalent to regular) multiset grammars as introduced by Crespi-Reghizzi and Mandrioli [2] under the name *commutative context-free grammars* and later developed further by Kudlek et al. [5]. As observers appropriate multiset finite automata like the ones used for membrane systems [1] should be employed. For the above result we used locally commutative context-free grammars, because we wanted to stay within classical formal language theory and did not want do abandon the concept of a string.

Taking into account the close relation between context-free grammars and 0L systems, it is natural to expect a similar result for L systems. And indeed we immediately obtain a corresponding result for a system consisting of an E0L sytem and again a finite automaton. For details about L systems we refer to the chapter on this topic in [7].

Corollary 6. *Every recursively enumerable language can be generated by a free L/O system constituted by an E0L system and a finite automaton with singular output.*

Proof. From the grammar of Theorem 5, we can construct an E0L system with the same set of rules plus a rule $X \to X$ for each non-terminal of the grammar except the synchronizing Δ's. Thus the enforced changing of Δ in every step ensures that all the other changes are made, especially that a configuration producing an output does not remain unchanged and produces this output twice. Using the same observer, this L/O system generates the same language as the G/O system from Theorem 5.

Another interesting fact is that intuitively the observer's ability to produce $\perp$, i.e. to eliminate certain computations, seems a powerful and essential feature in all three models investigated in this and the previous sections. However, we obtain all recursively enumerable languages over Σ simply by intersection of a language over $\Sigma_\perp$ with the regular language Σ^*. Since recursive languages are closed under intersection with regular sets, there must exist some grammar/observer systems Ω generating a non-recursive $\mathscr{L}_{\mathrm{f}}(\Omega)$. Because in the proof of Theorem 5 we make very heavy use of the option to trash strings, it is by no means obvious, how this result can be obtained in a direct way.

6. Final remarks

The most common computationally universal device is the Turing Machine. We will now compare this model to the G/O systems introduced here. This will exhibit some parallels and differences maybe not obvious at first sight. The Turing Machines we will speak about have a combined input/working tape, a finite state control and an output tape, on which they can only write.

This general setup resembles very much that of the G/O systems. In the sentential form of the context-free grammar one symbol can be rewritten in any step—if during almost the entire computation the sentential form consists of non-terminals as in Theorem 3, then at any time basically every symbol can be rewritten. The finite automaton generates an output depending on the content of the sentential form, just like the finite state control does depending on the symbol read from the input/working tape. So what are the differences?

For one thing the interaction between control and workspace is greatly decreased: the automaton only writes the output, but has no direct influence on the changes in the sentential form. Further it always starts in the same initial state and therefore cannot remember anything from the earlier computation steps. All this is compensated by the ability to read the entire workspace in every step. Thus the automaton can on the one hand have some memory of earlier steps there and on the other hand, based on this memory, it can check, whether the grammar has made exactly the changes which in a Turing machine the control would effect directly.

Using only regular grammars would mean that basically nothing that is written on the workspace can be rewritten. Therefore we conjecture that G/O systems with both the grammar and the observer regular should themselves only generate regular languages. If this proves to be true, then the same might hold for linear grammars, whose sentential forms also contain at most one non-terminal at a time.

There are many other interesting questions that remain to be answered about the G/O system architecture. The most interesting open problem left in this paper is the exact characterization of the languages generated by *always writing G/O systems*: are they the context sensitive languages? If—as we have conjectured—this is not the case, which are the properties of this language family? Perhaps it equals some family known from regulated rewriting.

Further, we have only marginally treated all the variants, where all observed behaviours must be considered and $\perp$ is not used. While we have mentioned that also these generate some non-recursive languages in the cases discussed in Sections 4 and 5, it seems doubtful that they are universal.

Of course, there remains the possibility to apply the architecture introduced to even other types of generating devices like contextual grammars, or to formal models of biological systems other than membrane systems, where some kind of evolution can be observed.

Acknowledgements

The authors are very thankful to Gheorghe Păun for very interesting discussions as well as many comments and suggestions on drafts of this work. Thanks are also due to Victor Mitrana and Max Garzón for their helpful and interesting comments.

References

[1] M. Cavaliere, P. Leupold, Evolution and observation—a new way to look at membrane systems, in: A. Alhazov, C. Martín-Vide, Gh. Păun (Eds.), Tech. Rep. No. 28/03 Research Group on Mathematical Linguistics, Tarragona; Pre-proc. of the Workshop on Membrane Computing 2003, Tarragona, Spain; http://pizarro.fll.urv.es/continguts/linguistica/proyecto/reports/wmc03.html.

[2] S. Crespi-Reghizzi, D. Mandrioli, Petri nets and commutative grammars, Tech. Rep. 74-5, Istituto di Elettrotecnica ed Elettronica—Politecnico di Milano, 1974.

[3] J. Dassow, Gh. Păun, Regulated rewriting in formal language theory, Springer, Berlin, 1989.

[4] J.E. Hopcroft, J.D. Ullmann, Introduction to Automata Theory, Languages and Computation, Addison-Wesley, Reading, MA, 1979.

[5] M. Kudlek, C. Martín Vide, Gh. Păun, Towards a formal macroset theory, in: C. Calude, Gh. Paun, G. Rozenberg, A. Salomaa (Eds.), Multiset Processing, Lecture Notes in Computer Science, vol. 2235, Springer, Berlin, 2001, pp. 123–133.

[6] G. Rozenberg, A. Salomaa, Watson–Crick complementarity, universal computations and genetic engineering, Tech. Rep. 96-28, Department of Computer Science, Leiden University, 1996.

[7] G. Rozenberg, A. Salomaa (Eds.), Handbook of Formal Languages, Springer, Berlin, 1997.