Scenario Based P Systems

Gabriel Ciobanu[1] and Dragoş Sburlan[2]

[1] Romanian Academy, Institute of Computer Science
Blvd. Carol I no. 8, 700505 Iasi, Romania
E-mail: gabriel@info.uaic.ro

[2] Faculty of Mathematics and Informatics
Ovidius University of Constanta, Romania
E-mail: dsburlan@univ-ovidius.ro

Summary. In this paper we define and study *Scenario Based P Systems*, a model of computation inspired by the metabolic pathways and networks. Starting from the classical definition of P systems with symbol objects and multiset rewriting rules, we define regular expressions able to capture the causal dependencies among different executions of the rules. The results show the computational power of this model.

1 Motivation

Metabolic pathways are sequences of biochemical reactions occurring inside the living cell which are involved in cell's energy management and in the synthesis of structural components. Because in such sequences participate many biochemicals (the metabolites), metabolic pathways are usually very complex. Moreover, many distinct pathways co-exist inside the cell and they form what is called the metabolic network. A metabolic pathway illustrate all the changes in time by which an initial molecule is transformed into another product. Usually, the products of one biochemical reaction constitute the substrate for the next biochemical reaction. The resulting product can be used by the cell to start another metabolic pathway, or it can be stored for a later use. Depending on the needs of the cell and on the availability of the substrate, these metabolic pathways are started.

In a broader perspective, the principle of causality plays the main role in finding/expressing metabolic pathways which connect parts of a metabolic network (our understanding of phenomena happening inside the cells is based on the causal relations existing among cell's "observable" events). In this context, one can consider the biochemical reactions as causal consequences where the input metabolites can cause the output metabolites. Moreover, there might be a certain temporal order by which any later event is determined by the earlier one, and which is not necessarily related with the involved metabolites.

This paper explores the concept of causality in the P system framework having as inspiration the biochemical dynamics expressed by the metabolic pathways. Its goal is to capture the causal dependencies existing among the executions of rules, while abstracting away other aspects. In the membrane computing literature there were several attempts to formalize causal semantics [3], [4], [2], and [8], most of them proposing a notion of causality based on the temporal order of single rule application. Our new approach introduces regular expressions to define the causal relation between the executions of rules; the time between the moments when these rules compete for objects can be also specified in the definition of regular expressions. Therefore, we define scenarios as a method to model different possible evolutions in the metabolic networks, and their causal relationships.

2 Preliminaries

We recall some notions and results from the classical theory of formal languages [5].

An *ET0L system* is a construct $H = (V, T, \omega, \Delta)$, where V is an alphabet, $T = \{T_1, \ldots T_m\}$, $m \geq 1$, such that T_i, $1 \leq i \leq m$, are finite complete sets of rules (tables) of non-cooperative rules over V, $\omega \in V^*$ is the axiom, and Δ is the terminal alphabet. In a derivation step, all the symbols present in the current sentential form are rewritten using one (nondeterministically chosen) table. The language generated by H consists of all the strings over Δ which can be generated in this way by starting from ω.

Lemma 1. *For each $L \in ET0L$ there is an extended tabled Lindemayer system $H = (V, T, \omega, \Delta)$ with two tables ($T = \{T_1, T_2\}$) generating L, such that for each $a \in \Delta$ if $a \to \alpha \in T_1 \cup T_2$ then $\alpha = a$.*

A *register machine* is a formal construct $M = (n, \mathcal{P}, l_0, l_h)$ where $n \geq 1$ is the number of registers, $\mathcal{P}$ is a finite set ($card(\mathcal{P} = k)$ of instructions bijectively labeled by elements from the set $B = \{l_0, \ldots, l_{k-1}\}$, $l_0 \in B$ is the initial label, and $l_h \in B$ is the final label. The instructions of M are of the following types:

- $l_1 : (add(r), l_2, l_3$ where $l_1 \in B \setminus \{l_h\}$, $l_2 \in B$, $1 \leq r \leq n$, increments the value stored by the register r and non-deterministically proceeds to the instruction labeled by l_2 or l_3;
- $l_1 : (sub(r), l_2, l_3)$ where $l_1 \in B \setminus \{l_h\}$, $l_2, l_3 \in B$, $1 \leq r \leq n$, if the value stored by register r is 0 then proceeds to the instruction labeled by l_3, otherwise decrements the value stored by register r and proceeds to the instruction labeled by l_2;
- $l_h : halt$ stops the machine.

A register machine is *deterministic* if $l_2 = l_3$ in all its add instructions.

A non-deterministic register machine M starts with all registers being empty and runs the program $\mathcal{P}$, starting from the instruction with the label l_0. Considering the content of register 1 for all possible computations of M which are ended by the execution of the instruction labeled l_h, one gets the set $N(M) \subseteq \mathbb{N}$ – the set generated by M.

A deterministic register machine M accepts a natural number by starting with the number as input in register 1, with all other registers being empty. M runs the program $\mathcal{P}$, starting from the instruction with the label l_0, and if it reaches the instruction l_h then it halts, accepting the number.

It is known the following result ([6]).

Theorem 1. *For any recursively enumerable set $Q \subseteq \mathbb{N}$ there exists a non-deterministic register machine with 3-registers generating Q such that when starting with all registers being empty, M non-deterministically computes and halts with n in register 1, and registers 2 and 3 being empty iff $n \in Q$.*

If FL is a family of languages, then by NFL we denote the family of length sets of languages in FL. We denote by REG, CF, $ET0L$, and RE the family of regular, context-free, extended tabled interactionless Lindemayer, and recursive enumerable languages, respectively. It is know that

$$NREG = NCF \subset NET0L \subset NRE.$$

The non-semilinear set $\{2^n \mid n \geq 0\} \in NETOL \setminus NCF$.

3 Scenario Based P Systems

The principle of causality implies a certain temporal order between some events and by which any later event is determined by the earlier one. However, the actual time elapsed between the occurrence of consecutive events that are in a given causality relation is not important. Based on these considerations we introduce a new model of P systems that use regular expressions to express a certain causal dependence relation between the execution of the rules.

The reader is assumed to be familiar with the basic notions, notations, and functioning of P Systems.

A *Scenario Based P System* (a SBP system, for short) of degree $m \geq 1$ is a construct $\Pi = (O, C, \mu, w_1, \ldots, w_m, R_1, \ldots, R_m, E_1, \ldots E_m, i_0)$, where

- O is an alphabet of *objects*;
- $C \subseteq O$ is the set of *catalysts*;
- μ is a tree structure of $m \geq 1$ uniquely labelled *membranes* (which delimit the regions of Π); usually, the set of labels is $\{1, \ldots, m\}$;
- $w_i \in O^*$, for $1 \leq i \leq m$, are multisets of objects which are initially present in the regions of μ (as indicated by the index);
- R_i, $1 \leq i \leq m$, is a finite set of labelled multiset rewriting rules. The set of labels is denoted by $\mathcal{L}_i$ and each label in $\mathcal{L}_i$ uniquely identifies a rule from R_i; in addition, $\mathcal{L}_i \cap \mathcal{L}_j = \emptyset$ for all $i \neq j$, $1 \leq i, j \leq m$. A rule from R_i is written as $l : \alpha \to \beta$ where $l \in \mathcal{L}_i$ and $\alpha, \beta \in O^*$. In particular, a rule can be *non-cooperative* $l : a \to v$ or *catalytic* $l : ca \to cv$, where $l \in \mathcal{L}_i$, $a \in O \setminus C$, $v \in ((O \setminus C) \times \{here, out, in\})^*$, and $c \in C$;

- E_i, $1 \leq i \leq m$, is a finite set of regular expressions over $\mathcal{L}_i \cup \{d\}$, where d is a special symbol (the "delay" symbol), $d \notin \bigcup_{i=0}^{m} \mathcal{L}_i$; moreover, if $e \in E_i$ then $L(e) \subseteq (\mathcal{L}_i \cup \{d\})^* \mathcal{L}_i (\mathcal{L}_i \cup \{d\})^*$ (that is, any word in $L(e)$ contains at least one symbol from $\mathcal{L}_i$);
- $i_0 \in \{1, \ldots, m\}$ is the label of the *output region* of Π.

A *configuration* of Π is a vector $(\alpha_1, \ldots, \alpha_m)$, where $\alpha_i \in O^*$, $1 \leq i \leq m$, is the multiset of objects present in the region i of Π. The *initial configuration* of Π is the vector $C_0 = (w_1, \ldots, w_m)$.

Let $E_i = \{e_{(i,1)}, \ldots, e_{(i,s_i)}\}$, where $1 \leq i \leq m$ and such that $s_i \geq 1$; in addition, let $L_{(i,1)}, \ldots, L_{(i,s_i)}$ be the corresponding regular languages. A word $l_0 \ldots l_t \in L_{(i,j)}$, $1 \leq i \leq m$, $1 \leq j \leq s_i$ (a finite sequence of symbols from $\mathcal{L}_i \cup \{d\}$) is called a *scenario* and illustrates the fact that the corresponding rules (if there exists such corresponding rules; recall that d is not associated with any rule) will be applied (if possible) in the implicit order of symbols. Given a multiset of objects w, a scenario $l_0 \ldots l_t$ is *applicable* to w if the rule having the label l_0 is applicable to w or $l_0 = d$; similarly, a scenario is *started* if the rule labeled with l_0 is applied to w or $l_0 = d$.

As usually in the P system framework, a computation of Π is a sequence of configurations (possibly infinite) $C_0, C_1, \ldots, C_k, C_{k+1}, \ldots$. Given a configuration $C_k = (w_{(k,1)}, \ldots, w_{(k,m)})$, then one gets the next configuration $C_{k+1} = (w_{(k+1,1)}, \ldots, w_{(k+1,m)})$ by applying on each multiset $w_{(k,i)}$, $1 \leq i \leq m$, some rules from R_i in a nondeterministic, maximal parallel manner and with competition on objects; these rules are selected according with the conditions described below.

For a scenario $l_0 \ldots l_t$ that is started in configuration C_k, the rule labeled l_i, $1 \leq i \leq t$, $l_i \neq d$, compete for objects in configuration C_{k+i} iff the rules labeled l_{i-j}, $1 \leq j \leq i$, $l_{i-j} \neq d$ were applied (won the competitions) in the corresponding configurations C_{k+i-j}. A started scenario is said to be *entirely applied* if the rules corresponding to all labels were applied in the given order, in consecutive configurations; in case there exists a rule labeled l_i, $1 \leq i \leq t$, $l_i \neq d$, that lost the competition on objects or if the rule cannot be applied then the started scenario is said to be *interrupted*; the executions of the remaining rules (in case they exist, that is, not all the remaining symbols in the scenario are d) in the subsequent configurations are dropped.

In any configuration, new scenarios from each $L_{(i,j)}$, $1 \leq i \leq m$, $1 \leq j \leq s_i$, are nondeterministically selected for applications.Given such a scenario and a configuration C_k, if the first label of rule appears in the scenario on position $l \geq 0$ (the first symbols being all d) then the corresponding rule will compete for objects with other rules (from the scenarios in progress) after l computational steps. For each multiset $w_{(k,i)}$ from C_k, $1 \leq i \leq m$, there might exist new scenarios, scenarios in progress, and interrupted scenarios, which determine the rules to be applied in order to obtain the next configuration C_{k+1}.

A computation of Π is a halting one if no rule can be applied (all the started scenarios are interrupted and no matter how a new scenario is selected for application it becomes interrupted at the first symbol corresponding to a rule) in the last configuration (the *halting configuration*). The result of a halting computation is the number of objects from O contained in the output region i_0, in the halting configuration. A non-halting computation yields no result. By collecting the results of all possible halting computations of a given P system Π, one gets $N(\Pi)$ – the set of all natural numbers generated by Π.

The family of all sets of numbers computed by SBP systems with at most m membranes and with a list of features f is denoted by $NOSBP_m(f)$. The features considered in this paper are *ncoo* (P systems using only non-cooperative rules) and cat_k (P systems using non-cooperative rules and catalytic rules with at most k catalysts).

The above definition can be relaxed such that in a halting configuration one counts only the symbols from a given alphabet $\Sigma \subseteq O$.

Given a scenario based P system Π with $m > 1$ membranes and using the features f, it is easy to construct an equivalent scenario based P system with the same features but having only one region. This can be accomplished by a simple encoding of the region labels into the objects and expressing the rules accordingly [1].

Theorem 2. $NOSBP_m(cat_1) = NRE$, $k \geq 1$.

Proof. The inclusion $NOSBP_m(cat) \subseteq NRE$ is supposed to be true by invoking the Church-Turing thesis. The opposite inclusion can be shown to be true by simulating the computation of an arbitrary register machine $M = (n, \mathcal{P}, l_0, l_h)$ with a scenario based P system $\Pi = (O, C, \mu, w_1, R_1, E_1)$ where

$$\begin{aligned} O &= \{a_i \mid 1 \leq i \leq n\} \\ &\quad \cup \{l_1, l_2 \mid l_1 : (add(r), l_2) \in \mathcal{P}\} \\ &\quad \cup \{l_1, l_2, l_3, \overline{l_1}, \overline{l_2}, S, \overline{S}, \overline{\overline{S}}, X \mid l_1 : (sub(r), l_2, l_3)\} \\ C &= \{c\}, \quad \mu = [\,]_1, \quad w_1 = l_0. \end{aligned}$$

The set of rules R_1 and the set of regular expressions E_1 are defined as follows:

- for each register machine instruction $l_1 : (add(r), l_2)$, the rule $r_{l_1} : l_1 \to a_r l_2$ is added to R_1 and the regular expression r_{l_1} is added to E_1.
- for each register machine instruction $l_1 : (sub(r), l_2, l_3)$, the next rules are added to R_1:

$$\begin{aligned} r_{(l_1,1)} : l_1 \to \overline{l_1} S \;&,\; r_{(l_1,2)} : ca_r \to cX \\ r_{(l_1,3)} : X \to \lambda \;&,\; r_{(l_1,4)} : \overline{l_1} \to \overline{l_2} \\ r_{(l_1,5)} : S \to \overline{S} \;&,\; r_{(l_1,6)} : \overline{S} \to \overline{\overline{S}} \\ r_{(l_1,7)} : \overline{\overline{S}} \to \lambda \;&,\; r_{(l_1,8)} : \overline{l_1} \to l_3 \\ r_{(l_1,9)} : \overline{l_2} \to l_2 \;&. \end{aligned}$$

The regular expressions $r_{(l_1,1)}r_{(l_1,2)}$, $r_{(l_1,3)}r_{(l_1,4)}$, $r_{(l_1,5)}r_{(l_1,6)}$, $r_{(l_1,7)}r_{(l_1,8)}$, $r_{(l_1,9)}$ are added to E_1.

- for the register machine instruction $l_1 : halt$, the rule $r_{l_1} : l_1 \rightarrow \lambda$ is added to R_1 and the regular expression r_{l_1} is added to E_1.

The simulation of the register machine M by the scenario based P system Π proceeds as follows. At a certain moment during the computation of M the values stored by the registers are $t_1, \ldots, t_r, \ldots t_n$, and the label of the instruction that has to be executed is l_1. Correspondingly, the multiset contained in the region of Π is $a_1^{t_1} \ldots a_r^{t_r} \ldots a_n^{t_n} l_1 c$ (that is, the value t_r stored by the register r of M is modeled in this simulation as the multiplicity of the object a_r in a configuration of Π).

If l_1 is the label of an addition instruction $l_1 : (add(r), l_2)$, then Π is executing the scenario described by r_{l_1}, that is the rule $l_1 \rightarrow a_r l_2$ is applied. As a consequence the next configuration of Π will be $a_1^{t_1} \ldots a_r^{t_r+1} \ldots a_n^{t_n} l_2 c$ (which indicates that the addition instruction was simulated correctly).

If l_1 is the label of a subtraction instruction $l_1 : (sub(r), l_2, l_3)$, then Π is executing the scenario described by $r_{(l_1,1)}r_{(l_1,2)}$. Consequently, because in this scenario the rule $r_{(l_1,1)} : l_1 \rightarrow \overline{l_1}S$ is executed firstly, the next configuration of Π is described by the multiset $a_1^{t_1} \ldots a_r^{t_r} \ldots a_n^{t_n} \overline{l_1} S c$. Because the object S appeared in the multiset, then the scenario $r_{(l_1,5)}r_{(l_1,6)}$ will be started. Next, two cases might happen:

- if $t_r > 0$ then the rule $r_{(l_1,2)} : ca_r \rightarrow cX$ is executed (the second rule from the already started scenario $r_{(l_1,1)}r_{(l_1,2)}$) in the same moment with the rule $r_{(l_1,5)} : S \rightarrow \overline{S}$ (from scenario $r_{(l_1,5)}r_{(l_1,6)}$). The configuration of Π becomes $a_1^{t_1} \ldots a_r^{t_r-1} \ldots a_n^{t_n} \overline{l_1} X \overline{S} c$. Next, the scenario $r_{(l_1,3)}r_{(l_1,4)}$ is started. It follows that the rules $r_{(l_1,6)} : \overline{S} \rightarrow \overline{\overline{S}}$ (from scenario $r_{(l_1,5)}r_{(l_1,6)}$) and $r_{(l_1,3)} : X \rightarrow \lambda$ (from scenario $r_{(l_1,3)}r_{(l_1,4)}$) are simultaneously executed; the configuration of Π becomes $a_1^{t_1} \ldots a_r^{t_r-1} \ldots a_n^{t_n} \overline{l_1}\, \overline{\overline{S}} c$. Finally, the scenario $r_{(l_1,7)}r_{(l_1,8)}$ is started. Accordingly, the rules $r_{(l_1,4)} : \overline{l_1} \rightarrow \overline{l_2}$ (from scenario $r_{(l_1,3)}r_{(l_1,4)}$) and $r_{(l_1,7)} : \overline{\overline{S}} \rightarrow \lambda$ (from scenario $r_{(l_1,7)}r_{(l_1,8)}$) are executed in the same time; the configuration of Π becomes $a_1^{t_1} \ldots a_r^{t_r-1} \ldots a_n^{t_n} \overline{l_2} c$. Next, the scenario $r_{(l_1,7)}r_{(l_1,8)}$ interrupts its execution (the object $\overline{l_1}$ is not anymore present in the current configuration of Π, hence the rule $r_{(l_1,8)}$ cannot be executed); the scenario $r_{(l_1,9)}$ starts its execution and this yields to the configuration $a_1^{t_1} \ldots a_r^{t_r-1} \ldots a_n^{t_n} l_2 c$ (which indicates a correct simulation of the register machine subtraction instruction in the case when register r is not empty);
- if $t_r = 0$ then the rule $r_{(l_1,2)} : ca_r \rightarrow cX$ cannot be executed and consequently the object X (which triggered the execution of the scenario $r_{(l_1,3)}r_{(l_1,4)}$) is not produced anymore. However, in this case the scenario $r_{(l_1,5)}r_{(l_1,6)}$ is started and the rules $r_{(l_1,5)} : S \rightarrow \overline{S}$ and $r_{(l_1,6)} : \overline{S} \rightarrow \overline{\overline{S}}$ are executed in consecutive configurations of Π. The resulting configuration becomes $a_1^{t_1} \ldots a_r^{t_r} \ldots a_n^{t_n} \overline{l_1}\, \overline{\overline{S}} c$. Next, the scenario $r_{(l_1,7)}r_{(l_1,8)}$ starts its execution and after two computational

steps the resulting multiset becomes $a_1^{t_1} \dots a_r^{t_r} \dots a_n^{t_n} l_3 c$ (which indicates a correct simulation of the register machine subtraction instruction in the case when register r is empty).

In case the configuration of Π is $a_1^{t_1} \dots a_n^{t_n} l_1 c$ where the object l_1 corresponds to the label of the register machine halting instruction, then the scenario r_{l_1} is started (the rule $r_{l_1} : l_1 \to \lambda$ is executed). The next configuration of Π becomes $a_1^{t_1} \dots a_n^{t_n} c$ and the computation stops.

Consequently, since the computation of M was correctly simulated by Π and the register machines are computational universal, we have $NOSBP_m(cat) \supseteq NRE$.

4 A More Realistic Scenario

A particular case, interesting from a biological point of view, is when all possible scenarios used by a SBP system Π in any region i are of type $d^{l_1} w_1 d^{l_2} w_2 d^{l_3} \dots d^{l_k} w_k d^{l_{k+1}}$, where $w_i \in \mathcal{L}_i^+$, $l_i \in I\!N$, $1 \leq i \leq k+1$. We will consider that the regular expressions from each E_i, $1 \leq i \leq m$, are of type $d^* \alpha_1 d^* \alpha_2 d^* \dots d^* \alpha_k d^*$ where each α_j, $1 \leq j \leq k$, are regular expressions over $\mathcal{L}_i$ which use only the grouping and the Boolean OR operations in their definitions (consequently, each α_i indicates a finite language). Such regular expressions and their corresponding scenarios suggest that one knows the application order of the rules but does not know when their executions will actually happen.

Let $E_i = \{e_{(i,1)}, \dots, e_{(i,s_i)}\}$, where $1 \leq i \leq m$, $s_i \geq 1$, and consider the corresponding regular languages $L_{(i,j)}$, $1 \leq i \leq m$, $1 \leq j \leq s_i$. In the above conditions, for a scenario $x = d^{l_1} w_1 d^{l_2} w_2 d^{l_3} \dots d^{l_k} w_k d^{l_{k+1}} \in L_{(i,j)}$ we define $deg(x) = \max\limits_{1 \leq i \leq k} \{|w_i|\}$.

For a given SBP system Π we define the *degree of synchronization*

$$deg(\Pi) = max\{deg(s) \mid (\exists) 1 \leq i \leq m, 1 \leq j \leq s_i \text{ such that } s \in L_{(i,j)}\}.$$

In this case, the family of all sets of numbers computed by such SBP systems with the feature $f \in \{ncoo, cat\}$ and of synchronization degree at most n will be denoted by $NOSBP_m^{d_n}(f)$.

The following example shows how to generate a non-semilinear set of numbers with a SBP systems with non-cooperative rules and with a synchronization degree 1.

Example 1. Let $\Pi_1 = (O, C, \mu, w_1, R_1, E_1, i_0)$ such that

$$O = \{a, b\}; \quad C = \emptyset; \quad \mu = [\,]_1; \quad w_1 = ab;$$
$$R_1 = \{r_1 : b \to b,\ r_2 : a \to aa,\ r_3 : b \to \lambda\};$$
$$E_1 = \{d^* r_1 d^* r_2 d^* r_3 d^*\};\ i_0 = 1.$$

The system Π_1 computes the set $\{a^{2^n} \mid n \geq 1\}$, the well know non–semilinear set from $NET0L \setminus NCF$. The regular expression used in the definition of Π can be simplified such that $E_1 = \{r_1 d^* r_2 d^* r_3\}$.Using this simplification, the system performs the computation as follows. In the first configuration $C_0 = (ab)$, a scenario $r_1 d^{n_1} r_2 d^{n_2} r_3$, $n_1, n_2 \geq 0$, is selected for application. This means that the rule labeled r_1 is applied in the configuration C_0 because there exists an object b; the rule labeled r_2 will compete for objects after n_1 computational steps (where n_1 can be any natural number) and if it is applied, it will double the objects a. Finally, after the next n_2 computational steps the rule r_3 compete for the object b, and if it is applied then it will delete the objects b (and consequently the selection of a scenario for an application is blocked). I all these computational steps (between the starting of the first scenario and the application of its last rule labeled r_3) new scenarios are selected for applications. Each of them will double the number of symbols a. Consequently, the system computes the set $\{a^{2^n} \mid n \geq 1\}$.

Theorem 3. *For any* $n \geq 2$*,*

$$NOSBP_m(f) \supseteq NOSBP_m^{d_n}(f) \supseteq NOSBP_m^{d_{n-1}}(f), f \in \{ncoo, cat_k\}, k \geq 1.$$

The following result shows the relation between the family of all sets of numbers computed by SBP systems with at most m membranes and using only non-cooperative rules and the family of length sets of context-free languages.

Proposition 1. $NOSBP_m^{d_1}(ncoo) \supset NCF = NREG$.

Proof. From the above observation one knows that $NOSBP_m(ncoo) = NOSBP_1(ncoo)$, hence in our proof we will use a scenario based P system with one region. Let $G = (N, T, P, S)$ be a context-free grammar and let $P = \{r_1, \ldots, r_k\}$ be the set of labeled productions. Then one can construct an equivalent scenario based P system $\Pi = (O, C, [\,]_1, R_1, E_1, i_0 = 1)$ defined by:

$$\begin{aligned} O &= N \cup T, \quad C = \emptyset, \\ R_1 &= P \cup \{r_A : A \to A \mid A \in N\}, \\ E_1 &= \{d^* r d^* \mid r \in P\} \cup \{d^* r_X d^* \mid X \in N\}. \end{aligned}$$

At any moment during the computation of Π scenarios from the languages indicated by the regular expressions from E_1 can be started. A scenario of type $d^k r d^p$, $r = A \to \alpha \in R_1$, $k, p \geq 0$, simulates the application of the context-free production $A \to \alpha \in P$. In order to prevent the maximal parallel rewriting of the object A in a given configuration of Π, scenario of type $d^k r_A d^p$, $r_A = A \to A \in R_1$ are employed. It follows that there exist a computation of Π where for any configuration exactly one object $A \in N$ is rewritten.

Thus, Π correctly simulates G, and so we conclude that $NOSBP_m^{d_1}(ncoo) \supseteq NCF = NREG$. The strict inclusion follows easily from Example 1.

The length set of any language generated by an ET0L system can be generated by a SBP systems with non-cooperative rules and synchronization degree 2.

Theorem 4. $NOSBP_m^{d_2}(ncoo) \supseteq NET0L$.

Proof. To prove this result, we simulate the computation of a arbitrary ET0L system using a SBP system with non-cooperative rules and having the synchronization degree 2. Without loss of generality, let $H = (V, T, \omega, \Delta)$ be an ET0L system, such that $V = \{a_1, \ldots, a_k\}$, $\Delta = \{a_1, \ldots, a_p\}$, $p \leq k$, and $T = \{T_1, T_2\}$, where

$$T_1 = \{a_i \to \alpha_{1,j} \mid 1 \leq i \leq k, 1 \leq j \leq l_{1,i}\},$$
$$T_2 = \{a_i \to \alpha_{2,j} \mid 1 \leq i \leq k, 1 \leq j \leq l_{2,i}\}.$$

Then we construct the SBP system $\Pi = (O, C, \mu, w_1, R_1, E_1, i_0 = 1)$ that simulates the computation of H as follows:

$$\begin{aligned} O = {} & V \cup \{\overline{a} \mid a \in V\} \\ & \cup \{t_{i,j} \mid i \in \{1,2\}, 1 \leq j \leq 2k+1\} \\ & \cup \{t, f, \#\}; \\ C = {} & \emptyset; \quad \mu = [\,]_1; \quad w_1 = t\omega. \end{aligned}$$

In order to simplify the notation and construction, we will present the regular expressions from E_1 by using directly the rules in their descriptions (and not the labels of the rules). The set of rules R_1 is composed by all the rules appearing in these regular expressions. In addition, the regular expressions will be grouped according to their usage in the simulation.

1. regular expressions/rules used to select a table to be simulated:
   $d^* \;\; t \to t_{1,1}^{max\{l_{1,i} \mid 1 \leq i \leq k\}} X \;\; d^*$
   $d^* \;\; t \to t_{2,1}^{max\{l_{2,i} \mid 1 \leq i \leq k\}} X \;\; d^*$
2. regular expressions/rules used to simulate an application of the table T_1:
   $d^* \;\; t_{1,1} \to t_{1,1} \;\; a_1 \to \overline{\alpha_{1,j}} \;\; d^*$ where $1 \leq j \leq l_{1,1}$
   $d^* \;\; t_{1,1} \to t_{1,2} \;\; a_1 \to \# \;\; d^*$
   $d^* \;\; t_{1,2} \to t_{1,2} \;\; a_2 \to \overline{\alpha_{1,j}} \;\; d^*$ where $1 \leq j \leq l_{1,2}$
   $d^* \;\; t_{1,2} \to t_{1,3} \;\; a_2 \to \# \;\; d^*$
   ...
   $d^* \;\; t_{1,k} \to t_{1,k} \;\; a_k \to \overline{\alpha_{1,j}} \;\; d^*$ where $1 \leq j \leq l_{1,k}$
   $d^* \;\; t_{1,k} \to t_{1,k+1} \;\; a_k \to \# \;\; d^*$

   $d^* \;\; t_{1,k+1} \to t_{1,k+2} \;\; \overline{a_1} \to a_1 \;\; d^*$
   $d^* \;\; t_{1,k+2} \to t_{1,k+3} \;\; \overline{a_2} \to a_2 \;\; d^*$
   ...
   $d^* \;\; t_{1,2k} \to t_{1,2k+1} \;\; \overline{a_k} \to a_k \;\; d^*$
3. regular expressions/rules used to simulate an application of the table T_2:
   $d^* \;\; t_{2,1} \to t_{2,1} \;\; a_1 \to \overline{\alpha_{2,j}} \;\; d^*$ where $1 \leq j \leq l_{2,1}$
   $d^* \;\; t_{2,1} \to t_{2,2} \;\; a_1 \to \# \;\; d^*$
   $d^* \;\; t_{2,2} \to t_{2,2} \;\; a_2 \to \overline{\alpha_{2,j}} \;\; d^*$ where $1 \leq j \leq l_{2,2}$
   $d^* \;\; t_{2,2} \to t_{2,3} \;\; a_2 \to \# \;\; d^*$
   ...
   $d^* \;\; t_{2,k} \to t_{2,k} \;\; a_k \to \overline{\alpha_{2,j}} \;\; d^*$ where $1 \leq j \leq l_{2,k}$

$d^* \quad t_{2,k} \rightarrow t_{2,k+1} \quad a_k \rightarrow \# \quad d^*$

$d^* \quad t_{2,k+1} \rightarrow t_{2,k+2} \quad \overline{a_1} \rightarrow a_1 \quad d^*$
$d^* \quad t_{2,k+2} \rightarrow t_{2,k+3} \quad \overline{a_2} \rightarrow a_2 \quad d^*$
...
$d^* \quad t_{2,2k} \rightarrow t_{2,2k+1} \quad \overline{a_k} \rightarrow a_k \quad d^*$

4. starting over the simulation or ending the simulation:
$d^* \quad t_{1,2k+1} \rightarrow \lambda \quad X \rightarrow t \quad d^*$
$d^* \quad t_{1,2k+1} \rightarrow \lambda \quad X \rightarrow f \quad d^*$
5. checking if there are "nonterminals" in the last configuration:
$d^* \quad f \rightarrow f_1 \quad a_1 \rightarrow \# \quad d^*$
$d^* \quad f_1 \rightarrow f_2 \quad a_2 \rightarrow \# \quad d^*$
...
$d^* \quad f_p \rightarrow \lambda \quad a_p \rightarrow \# \quad d^*$
$d^* \quad \# \rightarrow \# \quad d^*$.

The SBP system constructed above simulates the computation of an ET0L system as follows. At the beginning of simulation, scenarios from all the languages indicated by the regular expressions from E_1 are started. However, because there is an object t in the initial configuration, only the rules that appear in scenarios from the group 1 can be applied (that is it will be applied either $t \rightarrow t_{1,1}^{max\{l_{1,i}|1\leq i\leq k\}}X$ or $t \rightarrow t_{2,1}^{max\{l_{2,i}|1\leq i\leq k\}}X$). Let us assume that the rule $t \rightarrow t_{1,1}^{max\{l_{1,i}|1\leq i\leq k\}}X$ was executed, hence the table to be simulated is T_1. The number $max\{l_{1,i} \mid 1 \leq i \leq k\}$ of objects $t_{1,1}$ guarantees that any combination of the rules from T_1 which have the same symbol on the left and which are executed at certain moment by H, can be simulated by Π. Consequently, scenarios indicated by the regular expressions

$$d^* \quad t_{1,1} \rightarrow t_{1,1} \quad a_1 \rightarrow \overline{\alpha_{1,j}} \quad d^* \text{ where } 1 \leq j \leq l_{1,1}$$

are started. In these scenarios the rules of type $a_1 \rightarrow \overline{\alpha_{1,j}}$ (which correspond to the rules $a_1 \rightarrow \alpha_{1,j} \in T_1$) are applied at a certain moment. However, also the scenarios indicated by the regular expression

$$d^* \quad t_{1,1} \rightarrow t_{1,2} \quad a_1 \rightarrow \# \quad d^*$$

start their execution; in case the rule $t_{1,1} \rightarrow t_{1,2}$ is executed and there are objects a_1 in the region, then the symbol # will be produced and the system Π will never stop (no output). This scenario is used to check if all objects a_1 were rewritten.

The computation continues in the same manner for all the objects from V. After all objects from V were rewritten (i.e., in the current configuration there are only objects from the set $\{\overline{a} \mid a \in V\}$ and object $t_{1,k+1}$), the system Π rewrites back all the objects from the set $\{\overline{a} \mid a \in V\}$ into their corresponding version from V. Scenarios indicated by the following regular expressions are used to complete the task:

$$d^* \quad t_{1,k+1} \rightarrow t_{1,k+2} \quad \overline{a_1} \rightarrow a_1 \quad d^*$$
$$d^* \quad t_{1,k+2} \rightarrow t_{1,k+3} \quad \overline{a_2} \rightarrow a_2 \quad d^*$$
$$\dots$$
$$d^* \quad t_{1,2k} \rightarrow t_{1,2k+1} \quad \overline{a_k} \rightarrow a_k \quad d^*$$

Finally, scenarios from group 4 start, and object X is rewritten either into object t (and Π restarts the computation by simulating the application of another table of H) or into object f (which will be used to check whether the current configuration of Π corresponds to a string computed by H which is formed only by symbols from Δ). In case object f is generated, then scenarios indicated by regular expressions

$$\begin{array}{c} d^* \quad f \to f_1 \quad a_{p+1} \to \# \quad d^* \\ d^* \quad f_1 \to f_2 \quad a_{p+2} \to \# \quad d^* \\ \dots \\ d^* \quad f_p \to \lambda \quad a_k \to \# \quad d^* \\ d^* \quad \# \to \# \quad d^*. \end{array}$$

are applied and if in the current configuration there exists a symbol from $V \setminus \Delta$, then the symbol $\#$ is generated and the computation never stop. Otherwise, the system stops generating a multiset that correspond to a string from $L(H)$. Consequently, it was proved that $NOSBP_m^{d_2}(ncoo) \supseteq NET0L$.

The following result shows the computation power of SBP systems using non-cooperative and catalytic rules, and the degree of synchronization 3.

Theorem 5. $NOSBP_m^{d_3}(cat_1) = NRE$.

Proof. For the inclusion $NOSBP_m^{d_3}(cat_1) \supseteq NRE$ we will simulate with a SBP system with one region $\Pi = (O, C, \mu, w_1, R_1, E_1, i_0 = 1)$ a register machine $M = (n, \mathcal{P}, l_0, l_h)$. The system Π is defined as follows:

$$\begin{aligned} O &= \{a_i \mid 1 \le i \le n\} \cup \{l_1, l_2 \mid l_1 : (add(r), l_2) \in \mathcal{P}\} \\ &\quad \cup \{l_1, l_2, l_3, \overline{l_1}, \overline{l_2}, X, Y \mid l_1 : (sub(r), l_2, l_3)\}; \\ C &= \{c\}, \quad \mu = [\,]_1, \quad w_1 = l_0. \end{aligned}$$

The sets R_1 of rules and E_1 of regular expressions are constructed as follows:

- for any instruction $l_1 : (add(r), l_2)$ we add the rule $r_{(l_1,1)} : l_1 \to a_r l_2$ to R_1 and the regular expression $d^* r_{(l_1,1)} d^*$ to E_1;
- for any instruction $l_1 : (sub(r), l_2, l_3)$ we add the following rules to R_1 and the regular expressions to E_1

The rules added to R_1	The regular expressions added to E_1
$r_{(l_1,1)} : l_1 \to \overline{l_1} X$ $r_{(l_1,2)} : ca_r \to c$ $r_{(l_1,3)} : \overline{l_1} \to \overline{l_2}$	$d^* r_{(l_1,1)} r_{(l_1,2)} r_{(l_1,3)} d^*$
$r_{(l_1,4)} : X \to Y$ $r_{(l_1,5)} : Y \to \lambda$ $r_{(l_1,6)} : \overline{l_1} \to l_3$	$d^* r_{(l_1,4)} r_{(l_1,5)} r_{(l_1,6)} d^*$
$r_{(l_1,7)} : \overline{l_2} \to l_2$	$d^* r_{(l_1,7)} d^*$

- for the instruction $l_1 : halt$, the rule $r_{l_1} : l_1 \to \lambda$ is added to R_1 and the regular expression $d^* r_{l_1} d^*$ is added to E_1.

Similarly as in the proof of Theorem 2, we model the value stored in the register r of M as the multiplicity of the object a_r in a configuration of Π.

Since the scenarios are nondeterministically selected from the languages indicated by the regular expressions, and these scenarios may contain as a prefix a string of an arbitrary length and which is composed only by symbols d, then we don't know when the first rules of the scenarios will be executed.

Let Π be in a configuration $C_1 = a_1^{t_1} \dots a_r^{t_r} \dots a_n^{t_n} l_1 c$ and let us assume that in the configuration C_1 a rule from R_1 will be executed. In this configuration there might exist scenarios already in execution and/or scenarios that can be started. No matter which is the case, the single rule that can be applied in configuration C_1 is $l_1 \to \overline{l_1} X$ which belongs to a scenario s_1 from $L(d^* \;\; l_1 \to \overline{l_1} X \;\; ca_r \to c \;\; \overline{l_1} \to \overline{\overline{l_2}} \;\; d^*)$ (a scenario started in a previous configuration). This rule will be applied once and the resulting configuration will be $C_2 = a_1^{t_1} \dots a_r^{t_r} \dots a_n^{t_n} \overline{l_1} X c$. Next, we distinguish two cases:

- if $t_r > 0$ (that is, in C_2 there exists objects a_r) then the rule $ca_r \to c$ from scenario s will be executed. Moreover in this configuration will start new scenarios (apart from those already in execution). In particular, a scenario s_2 from $L(d^* \;\; X \to Y \;\; Y \to \lambda \;\; \overline{l_1} \to l_3 \;\; d^*)$ will be executed (which means that the rule $X \to Y$ will compete for objects, at a certain moment, in one subsequent configuration). However, there might be the case that a scenario of the same kind, started in a previous step, attempts to execute the rule $X \to Y$ in configuration C_2. Consequently we have two possible cases: in configuration C_2 will be only executed the rule $ca_r \to c$ (hence the next configuration will become $C_{(3,1)} = a_1^{t_1} \dots a_r^{t_r - 1} \dots a_n^{t_n} \overline{l_1} X c$) or the pair of rules $ca_r \to c$ and $X \to Y$ (hence the next configuration will become $C_{(3,2)} = a_1^{t_1} \dots a_r^{t_r - 1} \dots a_n^{t_n} \overline{l_1} Y c$). In the first case (i.e., in configuration $C_{(3,1)}$) we have again a branch in the computation: either will be executed the rule $\overline{l_1} \to \overline{l_2}$ (which means that the next configuration will be $C_{(3,1,1)} = a_1^{t_1} \dots a_r^{t_r - 1} \dots a_n^{t_n} \overline{l_2} X c$) or the pair of rules $\overline{l_1} \to \overline{l_2}$ and $X \to Y$ (which means that the next configuration will be $C_{(3,1,2)} = a_1^{t_1} \dots a_r^{t_r - 1} \dots a_n^{t_n} \overline{l_2} Y c$).
  It follows that for the configuration $C_{(3,1,1)}$ will be executed a scenario that, at a certain moment, will rewrite firstly the object X into Y (by an application of the rule $X \to Y$) and then will delete the object Y (by an application of the rule $Y \to \lambda$). In the same time, a scenario that applies the rule $\overline{\overline{l_2}} \to l_2$ will be executed and the configuration reached will be $a_1^{t_1} \dots a_r^{t_r - 1} \dots a_n^{t_n} l_2 c$ which corresponds to a correct simulation of the register machine subtraction instruction.
- if $t_r = 0$ then the rule $ca_r \to c$ from scenario s cannot be executed anymore and so, the execution of the scenario s will be interrupted (hence the rule $r_{(l_1,3)} : \overline{l_1} \to \overline{l_2}$ is not executed anymore in this simulation of the subtraction instruction). However the rule $r_{(l_1,4)} \; : \; X \to Y$ in a scenario from the set $L(d^* r_{(l_1,4)} r_{(l_1,5)} r_{(l_1,6)} d^*)$ will be executed at a certain moment. Next, the object Y will trigger the execution of the rule $r_{(l_1,5)} : Y \to \lambda$. Finally the rule $r_{(l_1,6)}$:

$\overline{l_1} \to l_3$ is applied and the resulting multiset will become $a_1^{t_1} \dots a_n^{t_n} l_3 c$ which again corresponds to a correct simulation of the register machine subtraction instruction.

It follows that Π correctly simulates the computation of M, and so, taking into account the Turing-Church thesis, $NOSBP_m^{d_3}(cat_1) = NRE$.

5 Conclusion

Metabolic pathways are usually composed of chains of enzymatically catalyzed chemical reactions. They are interconnected in a complex way in the framework of a metabolic network. Inspired by this biological phenomenon, we have defined and studied the scenario based P systems. In this computational model, regular expressions are used to express the causal dependence relations existing between various executions of the rules. In this way we intend to identify certain causalities in the chains of reactions connecting different parts of the metabolic network.

References

1. Agrigoroaiei, O., Ciobanu, G., Flattening the Transition P Systems with Dissolution, *Lecture Notes in Computer Science* vol.6501, pp.53–64, 2011.
2. Agrigoroaiei, O., Ciobanu, G., Quantitative Causality in Membrane Systems, *Lecture Notes in Computer Science* vol.7184, pp.62–72, 2012.
3. Busi, N., Causality in Membrane Systems, *Lecture Notes in Computer Science* vol.4860, pp.160–171, 2007.
4. Ciobanu, G., Lucanu, D., Events, Causality and Concurency in Membrane Systems, *Lecture Notes in Computer Science* vol.4860, pp.209–227, 2007.
5. Rozenberg, G., Salomaa, A. (Eds.): *Handbook of Formal Languages*, Springer Verlag, Berlin, 2004.
6. Minsky, M., *Computation: Finite and Infinite Machines*, Prentice-Hall, Englewood Cliffs, 1967.
7. Păun, G.: *Membrane Computing. An Introduction*, Springer, 2002.
8. Sburlan, D., P Systems with Chained Rules, *Lecture Notes in Computer Science* vol.7184, pp.359–370, 2012.