Katedra Informatiky
Fakulta Matematiky, Fyziky a Informatiky
Univerzita Komenského, Bratislava

Biologicky motivované výpočtové modely

(Dizertačná práca)

Michal Kováč

Vedúci: doc. RNDr. Damas Gruska, PhD.

Bratislava, 2011

Čestne prehlasujem, že som túto dizertačnú prácu vypracoval samostatne s použitím citovaných zdrojov.

. . . . . .

Poďakovanie

Osobitná vďaka patrí vedúcemu diplomovej práce doc. RNDr. Damasovi Gruskovi PhD. za cenné rady, námety, podnetné pripomienky a všestrannú pomoc, ktorú si hlboko vážim. Len vďaka mnohým prínosným konzultáciam a intenzívnej spolupráci som bol schopný napísať toto dielo. Nesmiem zabudnúť ani na RNDr. Branislava Rovana, CSc. a spolužiakov za to, že si na dizertačnom seminári našli čas, aby si vypočuli moju prezentáciu dizertačnej práce. Ďalšie poďakovania venujem rodičom a známym, ktorí to so mnou dokázali vydržať posledné týždne pred odovzdaním.

Abstrakt

Autor: Michal Kováč
Názov diplomovej práce: Biologicky motivované výpočtové modely
Škola: Univerzita Komenského v Bratislave
Fakulta: Fakulta matematiky, fyziky a informatiky
Katedra: Katedra informatiky
Vedúci bakalárskej práce: doc. RNDr. Damas Gruska, PhD.
Bratislava, 2011

Práca začína definovaním základných pojmov a končí interaktívnou prílohou.

Kľúčové slová: Mariáš, Teória hier, Minimax, Neúplna informácia.

Obsah

Zoznam obrázkov

Zoznam tabuliek

Introduction

About computational models inspired by biology. Neural networks, evolution algorithms, membrane systems.

V teoretickej informatike je veľa oblastí, ktoré sú motivované inými vednými disciplínami. Veľkú skupinu tvoria modely motivované biológiou. Patria sem napríklad neurónové siete, výpočtové modely založené na DNA, evolučné algoritmy, ktoré si už našli svoje významné uplatnenie v informatike a dokázali, že sa oplatí inšpirovať biológiou. L-systémy sú špecializované na popisovanie rastu rastlín, ale našli si uplatnenie aj v počítačovej grafike, konkrétne vo fraktálnej geometrii. Ďalšie rozvíjajúce sa oblasti ešte čakajú na svoje významnejšie uplatnenie.

Jednou z nich sú membránové systémy. Je pomerne mladá oblasť - prvý článok bol publikovaný v roku 1998 (see [1])

Kapitola 1

Preliminaries

1.1 Formal languages

Our study is based on the classical theory of formal languages. We will recall some definitions:

Definition 1.1.1 *An* **alphabet** *is a finite nonempty set of symbols. Usually it is denoted by* Σ *or* V*.*

Definition 1.1.2 *A* **string** *over an alphabet is a finite sequence of symbols from alphabet.*

We denote by V^* the set of all strings over an alphabet V. By $V^+ = V^* - \{\varepsilon\}$ we denote the set of all nonempty strings over V.

Definition 1.1.3 *A* **language** *over the alphabet* V *is any subset of* V^**.*

Definition 1.1.4 *A* **family languages** *is a set of languages.*

1.2 Formal grammars

Definition 1.2.1 *A* **formal grammar** *is a tuple* $G = (N, T, P, \sigma)$*, where*

- N, T *are disjoint alphabets of non-terminal and terminal symbols,*
- $\sigma \in N$ *is the initial non-terminal,*

- *P is a finite set of rewriting rules of the form $u \to v$, with $u \in (N \cup T)^* N (N \cup T)^*$ and $v \in (N \cup T)^*$.*

Definition 1.2.2 *A* **rewriting step** *in the grammar G is a binary relation $\Rightarrow$ on $(N \cup T)^*$, where $x \Rightarrow y$ only if $\exists w_1, w_2 \in (N \cup T)^+$ and a rule $u \to v \in P$ such that $x = w_1 u w_2$ and $y = w_1 v w_2$.*

Definition 1.2.3 *Language defined by a grammar G is a set $L(G) = \{w \in T^* | \sigma \Rightarrow w\}$.*

Languages that can be generated by a formal grammar are the recursively enumerable languages RE.

1.3 Chomsky hierarchy

In this section we introduce several well-known families of languages.

Definition 1.3.1 Regular grammar *is formal grammar, where rewriting rules are of the form $u \to v$, where $u \in N$ and $v \in T^*(N \cup \{\varepsilon\})$.*

Definition 1.3.2 Regular languages *are languages generated by regular grammars. They are denoted R.*

Definition 1.3.3 Context-free grammar *is formal grammar, where rewriting rules are of the form $u \to v$, where $u \in N$ and $v \in (N \cup T)^*$.*

Definition 1.3.4 Context-free languages *are languages generated by context-free grammars. They are denoted CF.*

Definition 1.3.5 Context-sensitive grammar *is formal grammar, where rewriting rules are of the form $u \to v$, where $u \in N$ and $v \in (N \cup T)^*$.*

Definition 1.3.6 Context-sensitive languages *are languages generated by context-sensitive grammars. They are denoted CS.*

These families of languages forms the Chomsky hierarchy by means of inclusions: $R \subset CF \subset CS \subset RE$.

1.4 Matrix grammars

Definition 1.4.1 *A* **matrix grammar** *is a tuple $G = (N, T, M, \sigma)$, where:*

- *N, T are disjoint alphabets of non-terminal and terminal symbols,*
- *$\sigma \in N$ is the initial non-terminal,*
- *M is a finite set of matrices, which are sequences of context-free rules of the form rewriting rules of the form $u \to v$, where $u \in N$ and $v \in (N \cup T)^*$.*

Definition 1.4.2 *A* **rewriting step** *$x \Rightarrow y$ holds only if there is a matrix $(u_1 \to v_1, u_2 \to v_2, \ldots, u_n \to v_n) \in M$ such that for each $1 \leq i \leq n$ the following holds: $x_i = x'_i u_i x''_i$ and $x_{i+1} = x'_i v_i x''_i$, where $x_i, x'_i, x''_i \in (N \cup T)^*$ and $x_1 = x$ and $x_{n+1} = y$.*

Example 1.4.1 Consider the matrix grammar $G = (\{\sigma, X, Y\}, \{a, b, c\}, M, \sigma)$, where M contains three matrices: $[S \to XY], [X \to aXb, Y \to cY], [X \to ab, Y \to c]$. There are only context-free rules, yet the grammar generate the context-sensitive language $\{a^n b^n c^n | n \geq 1\}$.

The family of matrix grammars is denoted MAT.

It is known that $CF \subset MAT \subset RE$. Interestingly, $MAT \cap a^* \subset R$ (see [2]).

1.5 Register machines

Definition 1.5.1 *A* **n-register machine** *is a tuple $M = (n, P, i, h)$, where:*

- *n is the number of registers,*
- *P is a set of labeled instructions of the form $j : (op(r), k, l)$, where $op(r)$ is an operation on register r of M, and j, k, l are labels from the set $Lab(M)$ (which numbers the instructions in a one-to-one manner),*
- *i is the initial label, and*
- *h is the final label.*

The machine is capable of the following instructions:

- $(add(r), k, l)$: Add one to the contents of register r and proceed to instruction k or to instruction l; in the deterministic variants usually considered in the literature we demand $k = l$.
- $(sub(r), k, l)$: If register r is not empty, then subtract one from its contents and go to instruction k, otherwise proceed to instruction l.
- *halt* : This instruction stops the machine. This additional instruction can only be assigned to the final label h.

A deterministic m-register machine can analyze an input $(n_1, \ldots, n_m) \in N_0^m$ in registers 1 to m, which is recognized if the register machine finally stops by the halt instruction with all its registers being empty (this last requirement is not necessary). If the machine does not halt, the analysis was not successful.

1.6 Lindenmeyer systems

In 1968, a Hungarian botanist and theoretical biologist Aristid Lindenmeyer introduced [3] a new string rewriting algorithm named Lindenmeyer systems (or L-systems for short). They are used by biologists and theoretical computer scientists to mathematically model growth processes of living organisms, especially plants. The difference with Chomsky grammars is that rewriting is parallel, not sequential.

The simplest version of L-systems assumes that the development of a cell is free of influence of other cells. This type of L-systems is called $0L$ systems, where "0" stands for zero-sided communication between cells.

Definition 1.6.1 *$0L$ system is a triple (Σ, P, ω), where Σ is an alphabet, ω is a word over Σ and P is a finite set of rewriting rules of the form $a \to x$, where $a \in \Sigma, x \in \Sigma^*$.*

It is assumed there is at least one rewriting rule for each letter of Σ. $0L$ system works in parallel way, so all the symbols are rewritten in each step.

Example 1.6.1 Consider the $0L$ system with alphabet $\Sigma = \{a, b\}$, initial word $\omega = a$ and rewriting rules $P = \{a \to b, b \to ab\}$. Since in this system there is exactly one rule for every letter of the alphabet, the rewriting is deterministic and the generated words will be $\{a, b, ab, bab, abbab, \ldots\}$.

$1L$ systems allows the rewriting rules to include context of size 1, so it allows for rules of type $yaz \to x$.

L-systems with tables (T) have several sets of rewriting rules instead of just one set. At one step of the rewriting process, rules belonging to the same set have to be applied. The biological motivation for introducing tables is that one may want different rules to take care of different environmental conditions (heat, light, etc.) or of different stages of development.

Definition 1.6.2 *An extended ($E0L$) system is a pair $G_1 = (G, \Sigma_T)$, where $G = (\Sigma, P, \omega)$ is an $0L$ system, where $\Sigma_T \subseteq \Sigma$, referred to as the terminal alphabet. The language generated by G_1 is defined by $L(G_1) = L(G) \cap \Sigma_T^*$.*

Such languages are called $E0L$ languages. $E0L$ languages with tables are called $ET0L$ languages.

It is known that $CF \subset E0L \subset ET0L \subset CS$ (see section 1.3 for definitions of CF and CS).

1.7 Multisets

Definition 1.7.1 *A multiset over a set X is a mapping $M : X \to \mathbb{N}$.*

We denote by $M(x), x \in X$ the multiplicity of x in the multiset M.

Definition 1.7.2 *The* **support** *of a multiset M is the set $supp(M) = \{x \in X | M(x) \geq 1\}$.*

It is the set of items with at least one occurence.

Definition 1.7.3 *A multiset is* **empty** *when it's support is empty.*

A multiset M with finite support $X = \{x_1, x_2, \ldots, x_n\}$ can be represented by the string $x_1^{M(x_1)} x_2^{M(x_2)} \ldots x_n^{M(x_n)}$.

Definition 1.7.4 *Multiset inclusion. We say that mutiset M_1 is included in multiset M_2 if $M_1(x) \leq M_2(x) \forall x \in X$. We denote it by $M_1 \subseteq M_2$.*

Definition 1.7.5 *The* **union** *of two multisets $M_1 \cup M_2 : X \to \mathbb{N}$ is defined as $(M_1 \cup M_2)(x) = M_1(x) + M_2(x)$.*

Definition 1.7.6 *The* **difference** *of two multisets $M_1 - M_2 : X \to \mathbb{N}$ is defined as $(M_1 - M_2)(x) = M_1(x) - M_2(x)$.*

Definition 1.7.7 *Product of multiset M with natural number $n \in \mathbb{N}$ is $(n \cdot M)(x) = n \cdot M(x)$.*

1.8 Multiset languages

The number of occurences of a given symbol $a \in V$ in the string $w \in V^*$ is denoted by $|w|_a$.

Definition 1.8.1 *$\Psi_V(w) = (|w|_{a_1}, |w|_{a_2}, \ldots, |w|_{a_n})$ is called a Parikh vector associated with the string $w \in V^*$, where $V = \{a_1, a_2, \ldots a_n\}$.*

Definition 1.8.2 *For a language $L \subseteq V^*$, $\Psi_V(L) = \{\Psi_V(w) | w \in L\}$ is the Parikh mapping associated with V.*

Example 1.8.1 Consider an alphabet $V = \{a, b\}$ and a language $L = \{a, ab, ba\}$. $\Psi_V(L) = \{(1, 0), (1, 1)\}$. Notice that Parikh image of L has only 2 element while L has 3 elements.

Definition 1.8.3 *If FL is a family of languages, by $PsFL$ is denoted the family of Parikh images of languages in FL.*

Kapitola 2

Membrane computing

Recently, an interdisciplinary research between the fields of Computer Science and Biology has been rapidly growing.

Bioinformatic has undergone a fast evolving process, especially the areas of genomics and proteomics. Bioinformatics can be seen as the application of computing tools and techniques for the management of biological data. Just to mention a few, the design of efficient algorithms for sequence alignment, the investigation of methods for prediction of the 3D structure of molecules and proteins and the development of data structures to effectively store huge amount of structured data.

On the other hand, the birth of biologically inspired frameworks started the investigation of mathematical models and their properties and technological requirements for their implementation by biological hardware. Those frameworks are inspired by the nature in the way it "computes", and has gone through the evolution for billions of years.

Neural networks, genetic algorithms and DNA computing are already well established research fields.

However, nature computes not only at the neural or genetic level, but also at the cellular level. In general, any non-trivial biological system has a hierarchical structure where objects and information flows between regions, what can be interpreted as a computation process.

The regions are typically delimited by various types of membranes at different levels from cell membranes, through skin membrane to virtual membranes which delimits different parts of an ecosystem. This hierarchical system can be seen in other field such as distributed computing, where again well delimited computing units coexist and are hierarchically arranged in complex

systems from single processors to the internet.

Membranes keep together certain chemicals or information and selectively determines which of them may pass through.

From these observations, Paun [1] introduces the notion of a membrane structure as a mathematical representation of hierarchical architectures composed of membranes. It is usually represented as a Venn diagram with all the considered sets being subsets of a unique set and not allowed to be intersected. Every two sets are either one the subset of the other, or disjoint. Outermost membrane (also called skin membrane) delimits the finite "inside" and the infinite "outside".

Recently, several computational models based on the membrane structure have been created.

P systems were introduced in 1998 as a system with membrane structure that contains multisets of objects and rewrite rules that are executed in a maximaly parallel manner. Since then, a huge amount of variants have been created with various computation powers. We have investigated the power of several such variants. Additionaly, we propose a new variant with a vacuum, where the rules can describe what will happen to an object that arrives to an empty membrane.

P systems with existing and newly proposed variants will be discussed in the chapter 3.

Aside from P systems, other models based on the membrane structure have been created such as the Calculus of Looping Sequences (CLS), which was inspired by P systems.

2.1 The Calculus of Looping Sequences

Barbuti in [4] concluded that there is a need for a formalism having a simple notation, having the ability of describing biological systems at different levels of abstractions, having some notions of compositionality and being flexible enough to allow describing new kinds of phenomena as they are discovered, without being specialized to the description of a particular class of systems. The Calculus of Looping Sequences (CLS) was introduced in [4].

The membrane structure in CLS is defined recursively, consisting of terms T and sequences S:

$$T ::= S \mid (S)^L \rfloor T \mid T|T$$

$$S ::= \varepsilon \mid a \mid S \cdot S$$

, where a is a symbol from the alphabet, ε represents the empty sequence, $\cdot$ is a sequencing operator, $(S)^L$ is a looping operator, $|$ is a parallel composition operator and $\rfloor$ is a containment operator.

Membranes are represented by a sequence that is looped around. Several extensions of the CLS have been proposed. In CLS+ the looping operator can be applied to a parallel composition of sequences, which represents a notion of a fluid membrane. CLS+ can be translated to CLS (see [4]). Milazzo in his PhD thesis [5] includes also a simulation of a P system using CLS. The major difficulty is simulating the maximal parallelism of rule application.

Kapitola 3

P systems

In previous chapter we introduced the notions of membrane and membrane structure.

The next step is to place certain objects in the regions delimited by the membranes. The objects are identified by their names, mathematically symbols from a given alphabet.

Several copies of the same object can appear in a region, so we will work with multisets of objects.

In order to obtain a computing device, we will allow the objects to evolve according to evolution rules. Any object, alone or together with another objects, can be transformed in other objects, can pass through a membrane, and can dissolve the membrane in which it is placed.

All objects evolve at the same time, in parallel manner across all membranes.

The evolution rules are hierarchizes by a priority relation, which is a partial order.

These aspects all together forms a P system as introduced in [1].

In section 3.1 we will provide formal definition of a P system.

3.1 Definitions

P system is a tuple $(V, \mu, w_1, w_2, \ldots, w_m, R_1, R_2, \ldots, R_m)$, where:

- V is the alphabet of symbols,

- μ is a membrane structure consisting of m membranes labeled with numbers $1, 2, \ldots, m$,
- $w_1, w_2, \ldots w_m$ are multisets of symbols present in the regions $1, 2, \ldots, m$ of the membrane structure,
- $R_1, R_2, \ldots R_m$ are finite sets of the rewriting rules asssociated with the regions $1, 2, \ldots, m$ of the membrane structure.

Each rewriting rule may specify for each symbol on the right side, whether it stays in the current region, moves through the membrane to the parent region or through membrane to one of the child regions. An example of such rule is the following: $abb \rightarrow (a, here)(b, in)(c, out)(c, here)$.

A **configuration** of a P system is represented by it's membrane structure and the multisets of objects in the regions.

A **computation step** of P system is a relation $\Rightarrow$ on the set of configurations such that $C_1 \Rightarrow C_2$ iff:

For every region in C_1 (suppose it contains a multiset of objects w) the corresponding multiset in C_2 is the result of applying a multiset of maximal simultaneously applicable multiset rewriting rules in R_w^{msap} to w.

In other words, a maximal multiset of rules is applied in each region.

For example, let's have two regions with multisets aa and b. In the first region there is a rule $a \rightarrow b$ and in the second membrane there is a rule $b \rightarrow aa$. The only possible result of a computation step is bb, aa. The first rule was applied twice and the second rule once. No more object could be consumed by rewriting rules.

Computation of a P system consists of a sequence of steps. The step S_i is appied to result of previous step S_{i-1}. So when $S_i = (C_j, C_{j+1})$, $S_{i-1} = (C_{j-1}, C_j)$.

There are two possible ways of assigning a result of a computation:

1. By considering the multiplicity of objects present in a designated membrane in a halting configuration. In this case we obtain a vector of natural numbers. We can also represent this vector as a multiset of objects or as Parikh image of a language.
2. By concatenating the symbols which leave the system, in the order they are sent out of the skin membrane (if several symbols are expelled at the same time, then any ordering of them is accepted). In this case we generate a language.

The result of a computation is clearly only one multiset or a string, but for one initial configuration there can be multiple possible computations. It follows from the fact that there exist more than one maximal multiset of rules that can be applied in each step.

3.2 P system variants

Besozzi in his PhD thesis (see [2]) formulates three criteria that a good P system variant should satisfy:

1. It should be as much realistic as possible from the biological point of view, in order not to widen the distance between the inspiring cellular reality and the idealized theory.

2. It should result in computational completeness and efficiency, which would mean to obtain universal (and hence, programmable) computing devices, with a powerful and useful intrinsic parallelism;

3. It should present mathematical minimality and elegance, to the aim of proposing an alternative framework for the analysis of computational models.

In membrane computing, many models are equal in power with Turing machines. We should say they are Turing complete (or computationally complete), but because the proofs are always constructive, starting the constructions from these proofs from universal Turing machines or from equivalent devices, we obtain universal P systems (able to simulate any other P system of the given type). That is why we speak about universality results, and not about computational completeness.

3.2.1 Accepting vs generating

In the Chomsky hierarchy, there are language acceptors (finite automata, Turing machines) and language generators (formal grammars).

Bordhin in [6] extends grammars to allow for accepting languages by interchanging the left side with the right side of a rule. The mode will apply rewriting rules to an input word and accept it when it reaches the starting nonterminal. However, the input word consists of terminal symbols, which

could not be rewritten when using original definition, hence they consider the pure version of various grammar types where they give up the distinction between terminal and nonterminal symbols.

The regular, context-free, context-sensitive and recursively enumerable languages were shown to have equal power in accepting and generating mode. Some other grammars (programmed grammars with appearence checking) are shown to be more powerful in accepting mode than in generating mode. For deterministic Lindenmeyer systems, the generating and accepting mode are incomparable.

It can be interesting to investigate accepting and generating mode also in P system variants. Barbuti in [7] shown that in the nondeterministic case, when either promoters or cooperative rules are allowed, acceptor P systems have shown to be universal. The same in known to hold for the corresponding classes of nondeterministic generator P systems. In the deterministic case, acceptor P systems have been shown to be universal only if cooperative rules are allowed. Universality has been shown not to hold for the corresponding classes of generator P systems.

3.2.2 Active vs passive membranes

Most of the studied P system variants assumes that the number of membranes can only decrease during a computation, by dissolving membranes as a result of applying evolution rules to the objects present in the system. A natural possibility is to let the number of membranes also to increase during a computation, for instance, by division, as it is well-known in biology. Actually, the membranes from biochemistry are not at all passive, like those in the models briefly described above. For example, the passing of a chemical compound through a membrane is often done by a direct interaction with the membrane itself (with the so-called protein channels or protein gates present in the membrane); during this interaction, the chemical compound which passes through membrane can be modified, while the membrane itself can in this way be modified (at least locally).

In [8] Paun considers P systems with active membranes where the central role in the computation is played by the membranes: evolution rules are associated both with objects and membranes, while the communication through membranes is performed with the direct participation of the membranes; moreover, the membranes can not only be dissolved, but they also can multiply by division. An elementary membrane can be divided by means

of an interaction with an object from that membrane.

Each membrane is supposed to have an electrical polarization (we will say charge), one of the three possible: positive, negative, or neutral. If in a membrane we have two immediately lower membranes of opposite polarizations, one positive and one negative, then that membrane can also divide in such a way that the two membranes of opposite charge are separated; all membranes of neutral charge and all objects are duplicated and a copy of each of them is introduced in each of the two new membranes. The skin is never divided. If at the same time a membrane is divided and there are objects in this membrane which are being rewritten in the same step, then in the new copies of the membrane the result of the evolution is included.

In this way, the number of membranes can grow, even exponentially. As expected, by making use of this increased parallelism we can compute faster. For example, the SAT problem, which is NP complete, can be solved in linear time, when we consider the steps of computation as the time units. Moreover, the model is shown to be computationally universal.

3.2.3 Context in rules

Rewriting rules in P systems can be cooperative and non-cooperative, like in Chomsky's context-free and context-sensitive grammars. Non-cooperative rules are restricted to use only one object on the left side and cooperative rules do not have this restriction. P systems with cooperative rules are universal [1], while P systems with non-cooperative rules only characterize parikh image of context-free languages ($PsCF$) [9].

Paun [1] also defines P systems with catalysts where catalysts are a specified subset of the alphabet. Rewriting rules can contain catalysts, which are not modified by applying the rule. Suprisingly, P systems with catalytic rules are universal, actually two membranes in the P system are sufficient to achieve universality.

In systems with only catalytic rules (purely catalytic systems), three catalysts are enough. ¡citation needed¿

Freund in [10] shows that two catalysts are enough and raised an open problem whether one catalyst is sufficient. He conjectured that for computationally universal P systems the results obtained in this paper are optimal not only with respect to the number of membranes (2), but also with respect to the number of catalysts.

From some point of view, catalysts are way too powerful in restricting

the parallelism - they directly participate in the rules, hence the number of catalytic rules that can be applied in one step, is bounded by number of catalysts.

A variant with promoters and inhibotors have been proposed (see [11]).

In the case of promoters, the rules are possible only in the presence of certain symbols. An object p is a promoter for a rule $u \to v$ and we denote this by $u \to v|_p$, if the rule is active only in the presence of object p. Note that unlike in the case with catalysts, promoters allow the associated rules to be applied as many times as possible.

An object i is inhibitor for a rule $u \to v$ and we denote this by $u \to v|_{\neg i}$, if the rule is active only if inhibitor i is not present in the region.

One of our results uses inhibitors as a tool to achieve universality for sequential P systems.

Ionescu in [11] shows that P systems with non-cooperative catalytic rules with only one catalyst and with promoters/inhibitors are universal.

Non-cooperative rules with no catalysts and with inhibitors were studied in [12], the equivalence with Lindenmeyer systems ($ET0L$ as defined in section 1.6) was proved.

Dang [13] proposes a simple cooperative system (SCO) as a P system where the only rules allowed are of the form $a \to v$ or of the form $aa \to v$, where a is a symbol and v is a (possibly null) string of symbols not containing a. This variant is investigated with various modes of parallelism, so their results will be mentioned in the subsection 3.2.7

3.2.4 Rules with priorities

In the original definition of a P system [1], a partial order relation over set of rewriting rules have been specified. The rule can be used only if no rule of a higher priority in the region can be applied at the same time.

Sosík in [14] showed that the priorities may be omitted from the model without loss of computational power.

3.2.5 Energy in P systems

Various notions of energy has been proposed for use in P systems. Paun in [15] considers a P system where each evolution rule "produces" or "consumes" some quantity of energy, in amounts which are expressed as integer numbers. In each moment and in each membrane the total energy involved

in an evolution step should be positive, but if "Soo much" energy is present in a membrane, then the membrane will be destroyed (dissolved). This variant was investigated in two cases, both were shown to be universal:

1. when using only two membranes and unbounded amount of energy,
2. when using arbitrarily many membranes and a bounded energy associated with rules

Freund in [16] introduced a new variant where the rules are assigned directly to membranes (every rule consume objects on one side of the membrane and produce objects on the other side) and every membrane carries an energy value that can be changed during a computation by objects passing through the membrane.

This variant is universal even in sequential mode if we allow priorities on the objects. When omitting the priority relation, only the family of Parikh sets generated by context-free matrix grammars ($PsMAT$ as defined in section 1.4) is obtained.

3.2.6 Symport/antiport rules

Paun in [17] proposes a new way of communicating between membranes.

Symports allow two chemicals to pass together through a membrane in the same direction using symport rules of type (ab, in) or (ab, out). Antiports allow two chemicals to pass simultaneously through a membrane in opposite directions using antiport rules of type $(a, in; b, out)$.

Suprisingly, a P system variant, where only the symport/antiport rules are used are computationally complete. Five membranes are enough for this result. If more than two chemicals may collaborate when passing through membranes, two membranes are sufficient for universality. These results are proven in [17].

3.2.7 Parallelism options

Original definition of P system (see [1]) uses maximal parallelism when doing a step of computation. There is an obvious biological motivation relying on the assumption that "if we wait long enough, then all reaction which may take place will take place". This condition is rather powerful, because it decreases

the non-determinism of the sysem's evolution. For various reasons ranging from looking for more realistic models to just the mathematical challenge, the maximal parallelism was questioned.

Dang in [18] investigates the sequential mode. In each step, from the set of applicable rules across all membrane one is nondeterministically chosen and applied. For purely catalytic systems with 1 membrane, the sequential mode generates only the semilinear sets and thus is strictly weaker than the maximally parallel version. Sequential version of symport/antiport systems are equivalent to vector addition systems making it strictly weaker than the original maximally parallel version.

Investigation of the sequential mode continues in [19]. Sequential P system without priorities with cooperative rules with rules for membrane dissolution are not universal by showing they can be simulated by vector addition systems with states (VASS). This holds even when the membrane creation is allowed for bounded number of created membranes. However, if any number of membranes are allowed to be created, the system becomes universal. This result was shown by simulation of the register machine (see section 1.5).

We have further investigated this variant (sequential P system without priorities with cooperative rules) by allowing rules with inhibitors, which resulted in universality.

Dang in [13] proposes several restricted versions of parallelism. n**-Max-Parallel** version nondeterministically selects a maximal subset of at most n rules to apply. It is proved that 9**-Max-Parallel** SCO (defined in the subsection 3.2.3) is universal. $\leq n$**-Parallel** version is similar, but does not require the condition of a maximal subset of rules. It is shown to be weaker than n**-Max-Parallel** version. n**-Parallel** version requires the size of the subset of rules to apply to be exactly n. All three versions are equal to the sequential mode when $n = 1$. For non-universality results, Dang used the proof technique by simulation by vector addition systems. Our future research may be inspired by this techniue.

Ciobanu in [20] proposes a minimal parallelism: for each region if at least one rule can be applied, then at least one rule will be applied. The symport/antiport rules variant and variant with active membranes were both shown to be universal.

Freund in [21] studied the asynchronous mode of P systems, where in each step, arbitrary many rules can be applied. The application of rules is hence done in parallel way, but are not synchronized or somewhat controlled. In many cases the sequential and asynchronous modes were shown to be

equivalent.

3.2.8 Sequential P systems without priorities with cooperative rules and inhibitors

Original definition of P systems with inhibitors (see [11]) allow to use only one inhibitor pre rule, e.g. $u \to v|_{\neg i}$. Alternative definition (see [22]) allow to use whole inhibitor set in the rule like $u \to v|_{\neg B}$, where B is a set of objects. Such a rule can be applied only if no element of B is present in the region.

For sequential P systems, the following lemma will show the equivalence of these definitions. However, special condition must be fulfilled, the regions with such rules cannot be empty.

Lemma 3.2.1 *If there is at least one object present in each region of a P system, rewriting step in P system with inhibitor set can be simulated by multiple consecutive steps of P system with single inhibitor.*

Proof 3.2.1 Consider P system with alphabet V. For each rule $u \to v|_{\neg B}$, where $B = \{b_1, b_2, \ldots, b_n\}$ we will have rules:

- $c \to c|GONE_b|_{\neg b}$ for all $c \in V, b \in B$
- $u|GONE_{b_1}|GONE_{b_2}|\ldots|GONE_{b_n} \to v|GONE_{b_1}|GONE_{b_2}|\ldots|GONE_{b_n}$

Theorem 3.2.2 *The sequential P system with inhibitors defines the same parikh image of language as P system with maximal parallelism.*

Proof 3.2.2 We show that we can simulate maximal parallel step of P system with several steps of sequential P system with inhibitors. The proof is quite technical with some wokrarounds.

It is important to note that in the maximal parallel step the rewriting occurs in all membranes, so we need to synchronize this process. Every membrane will have a state, represented as an object.

The RUN state represents that the rewriting still occurs. When there is no more rules to apply, the region has done it's maximal parallel step and proceeds to the state SYNCHRONIZE. Other states are just technical - we need to implement sending objects between membranes and preparing for the next maximal parallel step by unmarking newly created objects in the current maximal parallel step, which have been marked to prevent double rewriting in one step.

- RUN: Rewriting occurs. Objects that are to be sent to parent membrane are directly sent because the parent membrane is already in RUN or $SYNCHRONIZE$ phase, so the a' symbols that are sent don't break anything. But objects that are to be sent down, can't be sent immediately because child membranes can be in previous phase waiting to restore symbols from previous step. Current symbols could interfere with them and be rewrited twice in this step. Such objects are only marked as "to be sent down": $a^{\downarrow\prime}$

- $SYNCHRONIZE$: Rewriting has ended and membrane is waiting to get signal $SYNCED$ from parent membrane to continue to next step.

- $SENDDOWN$: Signal was caught and now all descendant membranes are in $SYNCHRONIZE$ phase so $a^{\downarrow\prime}$ can be sent down.

- $RESTORE$: All a' symbols are restored to a, so the next step of rewriting can take place.

- For every rule $r_i \in R$ such that $r_i = a_1^{M(a_1)} a_2^{M(a_2)} \dots a_n^{M(a_n)} \rightarrow a_1^{N(a_1)} a_2^{N(a_2)} \dots a_n^{N(a_n)}$ we will have the following rules:
  $a_1^{M(a_1)-m_1} \dot{a}_1^{m_1} a_2^{M(a_2)} \dot{a}_2^{m_2} \dots a_n^{M(a_n)} \dot{a}_n^{m_n} | RUN \rightarrow a_1'^{N(a_1)} a_2'^{N(a_2)} \dots a_n'^{N(a_n)} | RUN$

  There will be such rule for each $0 \leq m_i \leq M(a_i)$. It represents the idea that $\dot{a}$ can be used in rewriting in the same way as a. Right side of the rules contains symbols a', that prevents the symbols to be rewritten again.

- For every symbol $a \in V$ we will have the following rules:

  $a|RUN \rightarrow \dot{a}|RUN|_{\neg \dot{a}}$

  There will be max one occurence of $\dot{a}$.

- For every rule $r_i \in R$ there will be a rule that detects if the rule r_i is not usable. According to left side of the rule r_i, symbol $UNUSABLE_i$ will be created when there is not enough objects to fire the rule r_i. It means that left side of rule r_i requires more instances of some object than are present in membrane.

  If the left side is of type:

  - a: It is a context free rule. The rule can't be used if there is no occurence of a nor $\dot{a}$.
    $RUN \rightarrow UNUSABLE_i|RUN|_{\neg\{UNUSABLE_i,a,\dot{a}\}}$
  - ab: It is a cooperative rule with two distinct objects on the left side. The rule can't be used if there is one of them missing.
    $RUN \rightarrow UNUSABLE_i|RUN|_{\neg\{UNUSABLE_i,a,\dot{a}\}}$
    $RUN \rightarrow UNUSABLE_i|RUN|_{\neg\{UNUSABLE_i,b,\dot{b}\}}$
  - a^2: It is a cooperative rule with two same objects. The rule can't be used if there is at most one occurence of the symbol. That happens if there is no occurence of a. There can still be $\dot{a}$, but at most one occurence.
    $RUN \rightarrow UNUSABLE_i|RUN|_{\neg\{UNUSABLE_i,a\}}$

- For every membrane with label i there will be rule:
  $UNUSABLE_1|UNUSABLE_2|\ldots|UNUSABLE_m|RUN \rightarrow SYNCHRONIZE|SYNCTO$
  If no rule can be used, maximal parallel step in the region is completed so it goes to synchronization phase and sends a synchronization token to the parent membrane.

- For every membrane there will be a rule:
  $SYNCHRONIZE|SYNCTOKEN_j \rightarrow SYNCHRONIZE|SYNCTOKEN_j \uparrow$
  Membrane resends all sync token to parent membrane.

- In skin membrane there is a rule which collects all the synchronization tokens from all membranes $1 \ldots k$ and then sends down signal that synchronization is complete. But before that, there can be some symbols that should be sent down, but they weren't, because the region below could have not started the rewriting phase that time. The result was just marked with $a^{\downarrow}$.
  $SYNCTOKEN_1|SYNCTOKEN_2|\ldots|SYNCTOKEN_k|SYNCHRONIZE \rightarrow SENDDOWN$

- Every membrane other than skin membrane have to recieve the signal to go to senddown phase:
  $SYNCHRONIZE|SYNCED \rightarrow SENDDOWN$

- Every membrane will have rules for every symbol $a \in V$ to send down all unsent object that should have been sent down:

  $SENDDOWN|a^{\downarrow\prime} \rightarrow SENDDOWN|a' \downarrow$

- Every membrane will have rule for detecting when all such objects have been sent and it goes to restore phase:

  $SENDDOWN \rightarrow RESTORE|_{\neg\{a_1^{\downarrow\prime}, a_n^{\downarrow\prime}, \ldots, a_n^{\downarrow\prime}\}}$

- In restore phase all symbols a' will be rewritten to a in order to be able to be rewritten in next maximal parallel step:

  $RESTORE|a' \rightarrow RESTORE|a$

- When restore phase ends, it sends down signal that all membranes have been synchronized and next phase of rewriting has began in upper membranes:

  $RESTORE \rightarrow RUN|SYNCED \downarrow |_{\neg\{a'_1, a'_2, \ldots, a'_n\}}$

Phase of membrane is represented by object so the region is never empty and by the Lemma 3.2.1 the rules with set of inhibitors can be simulated by single inhibitors.

We have also reached this result in the accepting case by simulation of register machines.

Theorem 3.2.3 *Sequential P system with inhibitors can simulate register machine and thus equals PsRE.*

Proof 3.2.3 Suppose we have a n-register machine $M = (n, P, i, h)$. In our simulation we will have a membrane structure consisting of one membrane and the contents of register j will be represented by the multiplicity of the object a_j.

We will have P system (V, μ, w, R), where:

- V is an alphabet consisting of symbols that represent registers $a_1, \ldots a_n$ and instruction labels in $Lab(M)$,

- μ is a membrane structure consiting of only one membrane,

- w is initial contents of the membrane. It contains symbols for the input for the machine $a_i^{n_i}$ where n_i is initial state of register with label i and initial instruction $e \in Lab(M)$.
- R is the set of rules in the skin membrane:

  For all instructions of type $(e : add(j), f)$ we will have rule:

  - $e \to a_j|f$.

  For all instructions of type $(e : sub(j), f, z)$ we will have rules:

  - $e|a_j \to f$ and
  - $e \to z|_{\neg a_j}$.

  And finally halting rules:

  - $h|a_j \to h|\#$ for all $a \leq j \leq n$,
  - $\# \to \#$,

When the halting instruction is reached, if there is an object present in the membrane, the hash symbol # is created and it will cycle forever. If there is no object present, there is no rule to apply and computation will halt. It corresponds to the condition that all registers should be empty when halting.

3.2.9 Vacuum

We propose a new variant of P system with vacuum. In the common sense, vacuum represents a state of space with no or a little matter in it. Using vacuum in modelling frameworks can help express certain phenomena more easily. We define a new P system variant, which creates a special vacuum object in a region as soon as the region becomes empty. The vacuum is removed whenever some object interacts with it. After the interaction, there is vacuum no longer. This removal process is realized by allowing the vacuum object to be used only on the left side of rules. If we made the vacuum to be removed automatically when an object enters the region, there would be no difference with the variant without vacuum objects because of no interactions with it. We are interested in how the variant with the vacuum improves the computation power of a sequential P system in comparison to the variant without using the vacuum.

Theorem 3.2.4 *The sequential P system with Vacuum is universal.*

Proof 3.2.4 We can simulate the variant of P system where the only cooperative rule is of type $a|a \to b$. According to [13] the variant, where the only cooperative rule is when both objects are the same, is universal. If there is no rule $a \to b$, we can rewrite a to a' so we can mark all present symbols. a' symbols are kept in special membrane so the Vacuum can be created in main membrane and we can synchronize.

But we won't do this madness again.

Instead, we will try to proove universality by simulating the register machine. We need to detect when the current register is empty. If there was a symbol for every register as in the proof 3.2.3, the Vacuum would be created only if all registers are empty. But the $sub()$ instruction need to detect when one concrete register is empty.

We will have a membrane for each register. That membrane will be contained in the skin membrane. The number of objects in membrane i will correspond to the value of register i.

The alphabet will consist of instruction labels and register counter a. The skin membrane will only have an instruction label. It is sent to corresponding membrane where the instruction is executed. Then, the following instruction is sent back to the skin membrane.

We will have following rules in the skin membrane:

- $e \to e^{\downarrow_j}$ for an instruction of type $e : add(j), f$ or $e : sub(j), f, z$ and
- $h \to h^{\downarrow}$ for a halting instruction h.

And in non-skin membranes:

- $e \to a|f^{\uparrow}$ instructions of type $e : add(j), f$,
- $e|a \to f^{\uparrow}$ for instructions of type $e : sub(j), f, z$,
- $e|VACUUM \to z^{\uparrow}$ for instructions of type $e : sub(j), f, z$, and
- $h|a \to h|a$

When halting, if there is an nonempty register, it will cycle forever with the last rule. However, if all registers are empty, the halting instruction label will stay in all mmebranes and the computation will halt.

3.3 Case studies

Vultures in Pyrenees, Scavangers of Pyrenees.

Conclusions

We have studied several variants of sequential P systems in order to obtain universality without using maximal parallelism. A variant with rewriting rules that can use inhibitors was shown to be universal in both generating and accepting case. The generating model is able to simulate maximal parallel P system and the accepting model can simulate a register machine.

In addition, we have defined a new notion of vacuum, which is immediately created in the region that becomes empty. A new P system variant was introduced, which allowed the vacuum to be used on the left side of P system rewriting rules. The accepting case of this variant was shown to be universal by direct implementation of a register machine.

A lot of other variants deserve to be combined with the vacuum, so we suggest to research them more thoroughly(non-cooperative rules, rules with priorities, decaying objects, deterministic steps, . . .).

Literatúra

[1] Gheorghe Păun. Computing with membranes. Technical Report 208, Turku Center for Computer Science-TUCS, 1998. (www.tucs.fi).

[2] Daniela Besozzi. *Computational and modelling power of P systems*. PhD thesis, Universita' degli Studi di Milano, Milano, Italy, 2004.

[3] Aristid Lindenmeyer. Mathematical models for cellular interactions in development, ii. simple and branching filaments with two-sided inputs. *J. Theoretical Biology*, pages 280–315, 1968.

[4] Roberto Barbuti, Paolo Milazzo, and Angelo Troina. The calculus of looping sequences for modeling biological membranes. In *8th Workshop on Membrane Computing (WMC8), LNCS 4860*, pages 54–76. Springer, 2007.

[5] Paolo Milazzo. *Qualitative and Quantitative Formal Modeling of Biological Systems*. PhD thesis, Universita di Pisa.

[6] Henning Bordihn, Henning Fernau, and Markus Holzer. Accepting pure grammars and systems. In *Otto-von-Guericke-Universitat Magdeburg, Fakultat fur Informatik, Preprint Nr*, 1999.

[7] Roberto Barbuti, Andrea Maggiolo-Schettini, Paolo Milazzo, and Simone Tini. Membrane systems working in generating and accepting modes: expressiveness and encodings. In *Proceedings of the 11th international conference on Membrane computing*, CMC'10, pages 103–118, Berlin, Heidelberg, 2010. Springer-Verlag.

[8] Gheorghe Păun. P systems with active membranes: Attacking np complete problems. *JOURNAL OF AUTOMATA, LANGUAGES AND COMBINATORICS*, 6:75–90, 1999.

[9] Dragos Sburlan. *Promoting and Inhibiting Contexts in Membrane Computing*. PhD thesis, University of Seville.

[10] Rudolf Freund, Lila Kari, Marion Oswald, and Petr Sosík. Computationally universal p systems without priorities: two catalysts are sufficient. *Theoretical Computer Science*, 330(2):251 – 266, 2005. ¡ce:title¿Descriptional Complexity of Formal Systems¡/ce:title¿.

[11] Mihai Ionescu and Dragos Sburlan. On p systems with promoters/inhibitors. *Journal of Universal Computer Science*, 10(5):581–599, may 2004.

[12] Dragoş Sburlan. Further results on p systems with promoters/inhibitors. *International Journal of Foundations of Computer Science*, 17(01):205–221, 2006.

[13] Oscar H. Ibarra, Hsu chun Yen, and Zhe Dang. Dang: The power of maximal parallelism in p systems. In *Proceedings of the Eight Conference on Developments in Language Theory*, pages 212–224. Springer, 2004.

[14] Petr Sosík and Rudolf Freund. P systems without priorities are computationally universal. In *Revised Papers from the International Workshop on Membrane Computing*, WMC-CdeA '02, pages 400–409, London, UK, UK, 2003. Springer-Verlag.

[15] Gheorghe Pă, Yasuhiro Suzuki, and Hiroshi Tanaka. P systems with energy accounting. *International Journal of Computer Mathematics*, 78(3):343–364, 2001.

[16] Rudolf Freund, Alberto Leporati, Marion Oswald, and Claudio Zandron. Sequential p systems with unit rules and energy assigned to membranes. In *Proceedings of the 4th international conference on Machines, Computations, and Universality*, MCU'04, pages 200–210, Berlin, Heidelberg, 2005. Springer-Verlag.

[17] Andrei Paun and Gheorghe Paun. The power of communication: P systems with symport/antiport. *New Gen. Comput.*, 20(3):295–305, July 2002.

[18] Zhe Dang and Oscar H. Ibarra. On p systems operating in sequential mode. In *International Journal of Foundations of Computer Science*, pages 164–177, 2004.

[19] Oscar H. Ibarra, Sara Woodworth, Hsu-Chun Yen, and Zhe Dang. On sequential and 1-deterministic p systems. In *Proceedings of the 11th annual international conference on Computing and Combinatorics*, COCOON'05, pages 905–914, Berlin, Heidelberg, 2005. Springer-Verlag.

[20] Gabriel Ciobanu, Linqiang Pan, Gheorghe Pun, and Mario J. Pérez-Jiménez. P systems with minimal parallelism. *Theor. Comput. Sci.*, 378(1):117–130, June 2007.

[21] Rudolf Freund. Asynchronous p systems and p systems working in the sequential mode. In *Proceedings of the 5th international conference on Membrane Computing*, WMC'04, pages 36–62, Berlin, Heidelberg, 2005. Springer-Verlag.

[22] Oana Agrigoroaiei and Gabriel Ciobanu. Flattening the transition p systems with dissolution. In *Proceedings of the 11th international conference on Membrane computing*, CMC'10, pages 53–64, Berlin, Heidelberg, 2010. Springer-Verlag.

Štatistiky

Tu budú štatistiky.