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

Úvod

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 $x = w_1uw_2$ and $y = w_1vw_2$.*

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.

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 $1, 2, \dots, m$,
- $w_1, w_2, \dots w_m$ are multisets of symbols present in the regions $1, 2, \dots, m$ of the membrane structure,
- $R_1, R_2, \dots R_m$ are finite sets of the rewriting rules asssociated with the regions $1, 2, \dots, m$ 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: $abb \to (a, here)(b, in)(c, out)(c, here)$.

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 $a \to b$ and in the second membrane there is a rule $b \to aa$. 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.

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 [4] 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 [5] 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 [6] 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 ($PsCF$) [7].

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 [8] 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.

Ionescu in [9] 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 [10], the equivalence with Lindenmeyer systems ($ET0L$ as defined in section 1.6) was proved.

Dang [11] 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 [12] 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 [13] 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 [14] 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 is obtained.

3.2.6 Calculi of Looping Sequences

Toto neviem presne kam dat, lebo tu budem pisat aj simulacii P systemov, takze v kapitole "Membrane systems" asi nie, ale zas to nie je ani variant P systemu... najlepsie by bolo dat mozno tu niekde ku koncu sekciu “Related models”?

3.2.7 Parallelism options

Maximal parallelism, minimal parallelism, n-parallelism, sequential models.

3.3 Case studies

Vultures in Pyrenees, Scavangers of Pyrenees.

Záver

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

[5] 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.

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

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

[8] 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¿.

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

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

[11] 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.

[12] 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.

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

[14] 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.

Štatistiky

Tu budú štatistiky.