J.LAP, Volume 79(6), pp 326-333. 2010

P systems with control nuclei: The concept

Camelia Chira[*], Traian Florin Șerbănuță[†], Gheorghe Ștefănescu[‡]

July 26, 2010

Abstract

We describe an extension of P systems where each membrane has an associated control nucleus responsible with the generation of the rules to be applied in that membrane. The nucleus exports a set of rules which are applied in the membrane region (only for one step, but in the usual maximal-parallel way), then the rules are removed and a new iteration of this process takes place. This way, powerful control mechanisms may be included in P systems themselves, as opposed to using the level of "strategies" previously exploited for simulating P systems. The nuclei may contain general programs for generating rules, ranging from those using information on the full system, to more restricted programs where only local information in the nuclei themselves and the associated membranes is used. The latter approach, mixed with a particular mechanism for the representation of the control programs, the rules, and the export procedure is powerful enough for modeling complex biological applications, e.g., to develop a detailed model for cell growth and division in normal and abnormal (tumoral) evolution of biological systems.

1 Introduction

A previous paper [13] exploiting the relation between P systems [8] and the K rewrite-based framework [10, 11] introduces an extension of P systems with structural data, accompanied by an implementation using K and Maude rewriting engine. In this paper, we describe a further extension of P systems, briefly mentioned in [13], obtained by integrating powerful mechanisms to control the activity in P systems, called "control nuclei." With these two extensions, we foresee the development of a high level modeling and programming language on top of P systems, for instance, powerful enough to simulate the behavior of complex biological systems.

[*]Address: Department of Computer Science, Babes-Bolyai University Cluj-Napoca, cchira@cs.ubbcluj.ro.

[†]Address:Department of Computer Science, University of Illinois at Urbana-Champaign, tserban2@illinois.edu.

[‡]Corresponding author. Address: Department of Computer Science, University of Bucharest, Str. Academiei nr.14, sector 1, C.P. 010014, Bucuresti, Romania. gheorghe@funinf.cs.unibuc.ro.

The introduced model of P system with control nuclei is described in this paper (and illustrated by several examples) in the context of static membrane structures as well as for dynamic membranes. In the latter case, the behaviour of the extended P system model dealing with operations such as membrane dissolution, membrane migration and membrane division is discussed. Several approaches to membrane dissolution (identified as the most problematic operation for P systems with control nuclei) are presented. In this context, P systems with control nuclei using multiple parallel programs are defined as an extension of the introduced model able to deal with the membrane dissolution problem. Furthermore, the functionality of P systems with nuclei is emphasized for modeling cell growth and division in biological systems.

2 P system with control nuclei: static membrane structure

In this section we describe the extension of P systems with control nuclei in the restricted case of a static membrane structure. In such a case, no membrane dissolution or division is allowed. When this restriction is in place, the proposed extension to resulting P systems is easier as no combination of control programs from different nuclei is needed. The next section tackles the case of dynamic membrane structures (where the above restriction is missing) discussing the difficult decision problem of dealing with forking programs or combining programs from different nuclei and introducing the general concept of P systems with control nuclei.

2.1 P system with control nuclei

We present the extension for a particular class of P systems, known as classical transition P system [8]. The extension with control nuclei of other classes of P systems is similar and it is not included in this paper.

Definition 2.1 (Restricted transition P systems) A *restricted transition P system*, of degree $m \geq 1$, is formally defined by a tuple

$$\Pi = (O, C, \mu, w_1, \ldots, w_m, R_1, \ldots, R_m, i_o),$$

where: (1) O is an alphabet of *objects*; (2) $C \subseteq O$ is the set of catalysts; (3) μ is a membrane structure (with the membranes bijectively labeled by natural numbers $1, \ldots, m$); (4) $w_1, \ldots, w_m$ are multisets over O associated with the regions $1, \ldots, m$ of μ, represented by strings from O^* unique up to permutations; (5) $R_1, \ldots, R_m$ are finite sets of rules associated with the membranes $1, \ldots, m$; the rules are of the form

$u \to \overline{v}$
with $u \in O^+$ and $\overline{v} \in (O \times \mathit{Tar})^*$, where $\mathit{Tar} = \{\mathit{here}, \mathit{in}, \mathit{out}\}$

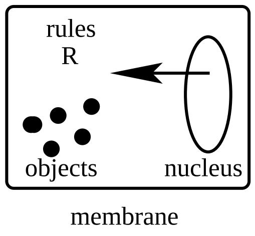

Figure 1: A membrane enriched with a control nucleus

(6) i_o is the label of the output membrane, an elementary one in μ; alternatively, i_o may be 0 indicating that the collecting region is the environment. □

This class of P systems explicitly excludes rules as

$$u \to \overline{v}\delta.$$

When δ is present in such a rule, its application leads to the dissolution of the membrane and to the abolishment of the rules associated with the membrane just dissolved. In the next section, we show how to deal with P systems using this type of rules and how to develop a rich set of options for combining the rules of the dissolved membrane with the ones of the parent membrane and not simply ignoring them.

Definition 2.2 (Semantics of P systems) A membrane is denoted by $[_h\ ]_h$. By convention, $[_h u]_h$ denotes a membrane with u present in the solution (among other objects). Starting from the *initial configuration*, which consists of μ and $w_1, \ldots, w_m$, the system passes from one configuration to another by applying a transition, i.e., the rules from each set R_i in a non-deterministic and maximally parallel way. A sequence of transitions is called a *computation*; a computation is *successful* if and only if it halts. With a successful computation one associates a *result*, in the form of the number of objects present in membrane i_o in the halting configuration. □

Figure 1 contains a schematic illustration of a membrane enriched with a control nucleus. The informal definition of P system with control nuclei follows. To get a more formal definition, one has to select a particular programming language for generating the rules, which is outside of the scope of this paper. However, it is worth mentioning here that a nucleus program may have global knowledge on the status of the whole P system. For instance, it may use tests on the presence or absence of specific objects in the regions of specific membranes when it decides on the rules to be exported in a next step in its own membrane region. (For more on this topics see the comments after Example 2.7.)

Definition 2.3 (Restricted P system with control nuclei) A restricted P system with control nuclei (rPCN, for short) is an enriched restricted P system, where each membrane has an associated nucleus with a program responsible for generating the rules to be used within the membrane region.

The semantics of a rPCN is simple: (1) start from the initial configuration; (2) in each running step, a set of rules is generated in each membrane by its nucleus program, the usual maximal-parallel rule for P systems is applied (synchronously, in all membranes), and finally the rules are deleted; (3) the procedure for one step, described in (2), is repeated until the halting condition is fulfilled. □

With this semantics, the nucleus programs will wait on each other until they all reach the `export` stage, hence these `export` statements act as a barrier, as well. A consequence of the synchronous behaviour of P systems is the end of the computation when one of the nucleus programs fails to reach a next `export` statement. A good programming technique is to use acyclic components of appropriate and balanced computation complexity in between consecutive `export` statements in all nucleus programs.

Below we present a sample of P systems with nucleus programs, written in a conventional imperative programming language, but enriched with an `export` statement and a running procedure similar to that used by coroutines. Namely, a program is executed from its current control point until it reaches an `export` statement. In such a point, the program is stopped and the generated rules are exported and applied in the region of its associated membrane. When this object transformation process is finished, the rules are discarded from the membrane region and the nucleus program is reactivated, starting from its last control point; it runs again until a new `export` statement is reached and this process is repeated until a halting condition is fulfilled.

Example 2.4 (Standard transition P systems)

```
Pa:: (code for membrane i)
  while(true) {
    export Ri;
  }
```

In this first example, we consider a P system with m membranes $1, \ldots, m$. The nucleus program `Pa`, described above and used by the i-th membrane in the P system, constantly produces the same set of rules `Ri` for its membrane i. As one can easily see, a rPCN using such nucleus programs is actually a classical transitional P system. The simplicity of the nucleus programs used for the class of P in this example suggests that general nucleus programs may add (very) powerful controlling mechanisms to classical P systems, indeed. □

Example 2.5 (Priorities)

```
Pb:: (code for membrane i)
  while(true) {
    export prim(Ri_1);
    ...
    export prim(Ri_ni);
    export Unprim;
  }
```

This second example, using the code in `Pb`, illustrates how to deal with priority strategies when applying the rules. Before going into details, some explanations on the notation are necessary: by `prim(R)` we denote the set of rules obtained from `R` by a decoration with $'$ (prim) of its right-hand side terms; `Unprim` is the rule which strip out the prim decoration of all terms. The program in `Pb` acts as follows: it first generates and applies the rules in `Ri_1`; when no such rules may be applied, the rules in `Ri_2` are applied, and so on; by using prim-unprim decorations we constrain the rules `Ri_1, Ri_2, ...` to be applied to the original elements in the membrane region, prohibiting the use of the newly produced values in subsequent rules.

When all membranes have the same number of rules (i.e., all `ni`'s are equal), the above method works well. Otherwise, to achieve a required global synchronization for a single step in all membranes of the original P system with priorities, one simply has to add appropriate number of `export` statements with empty sets of rules in the nucleus programs of the membranes with smaller number of rules.

Notice that this model is for rules with priorities in their weak interpretation. In this weak interpretation of the priority, rules are applied in decreasing order of their priorities, namely a lower priority rule can only be applied after all higher priority rules have been applied. (The case of strong interpretation, where a lower priority rule cannot be applied at all, if a higher priority rule was applied, is not considered here.) □

Example 2.6 (Synchronized P systems)

```
Pc:: (code of membrane i)
  x = i mod n;
  goto x;
  while(true) {
    0  : export R0;
    ...
    n-1: export R(n-1);
  }
```

The third example, using the program `Pc`, illustrates a kind of pipelined synchronous execution in a P system. Let us suppose that the P system has `m` membranes (denoted from 0 to `m-1`) and `n` sets of rules `R0,...,R(n-1)`. Each membrane infinitely repeats a cyclic execution of the rules `R0,...,R(n-1)`, but starting with a different rule. For instance, if `m=4` and `n=3`, the system uses the following rules in its membranes `0,...,3` at steps 1, 2, 3, etc.:

```
step 1:  (R0,R1,R2,R0);
step 2:  (R1,R2,R0,R1);
step 3:  (R2,R0,R1,R2);
etc.
```

□

Example 2.7 (Prime numbers) This last example illustrates how global knowledge on the state of a P system can be used by the nucleus programs. We give a more informal presentation of the system and the programs here.

This P system checks in parallel whether a given number is prime or not. To this aim, p membranes $1, 2, \ldots, p$ are used, with membrane 1 including membranes $2, \ldots, p$. The master membrane 1 proposes a number n and sends copies to all other membranes (using a rule $a \rightarrow (a, here)(a, in_2) \ldots (a, in_p))$. Every slave membrane i in $\{2, \ldots, p\}$ keeps a copy of the number n using n copies of the object a. In parallel, each slave membrane i starts checking the divisibility with value i; this is done by copying the number using b's (by a rule $a \rightarrow ab$) and then exporting the rule $b^i \rightarrow \lambda$. After this step, the master membrane tests if one membrane in $\{2, \ldots, p\}$ has no b's, in which case the computation stops with the conclusion that the number is not prime. It also checks if the square of the greatest current testing number (i.e., the one currently used in the membrane p) exceeds n, in which case the computation stops with the conclusion that the number is prime. Otherwise, each slave membrane i deletes all b's, increases its testing number i by $p - 1$, and repeats the above procedure. □

It is worth noticing that the spirit of P systems is to avoid such global knowledge, i.e., to limit the access of the nucleus programs to the local information within their membranes only and to use explicit passing of objects from a membrane region to another membrane region to convey information.

These programs are simple examples used to illustrate the concept of control nuclei. Actually, any kind of nucleus programs may be used in rPCNs. A particular benefit of the implementation of P systems in K, described in [13], is that it can be easily adapted to use such powerful nucleus programs - just mix the P systems implementation in [13] with the known representation of several common programming languages in K.

Remark 2.8 (add/remove/replace rules) The procedure of deleting the current rules and inserting new rules into a membrane region by passing from one computation step to the next is just one of many other possibilities. The `export` statement in the nucleus programs may be decorated with additional flags to specify the dynamics of the rules. For instance, one can use the following keywords associated to the `export` statement:

- `add R` - the rules in R are added to the rules already existing into the membrane region;
- `remove R` - the rules in R are removed from the membrane region;
- `replace R,S` - the rules in R are removed from the membrane region, while the rules in S are added to the rules in the membrane region (to avoid conflicts, `R` and `S` are supposed to be disjoint).

These options may be derived from the default case, provided the programming language for the nucleus programs is rich enough to allow keeping track of the exported rules during the runs of the program. □

Remark 2.9 The rules generated by the same nucleus program might differ from one context to another when the nucleus behaviour is implemented using an agent-based approach. In such a case, the nucleus program itself can act in a proactive manner modifying the output results, influenced by changes that occur in its environment, by messages received from other programs or by learning new information (agent-based behaviour similar to that of multi-agent systems where agents are able to autonomously act and take intiative for reaching an objective [12, 16]). □

2.2 A quantitative version

The definition of P systems with control nuclei in the previous section is further generalized to P systems with quantitative rules. This means that:

- Each rule exported by a nucleus in its membrane region may have a spatial multiplicity factor and it can be applied at most that number of times in a given step. The multiplicity factor is a natural number or infinity (∞). The former version of the model is the particular case of this extended version when the spatial multiplicity factor of all rules is equal to ∞.
- An orthogonal temporal quantitative version may be developed where each rule persists into the membrane region for a given number of steps (a natural number or infinity). In standard P systems this value is ∞, while in the rPCN's introduced in the previous subsection the value is 1.

Combinations of quantitative specifications for both the spatial and the temporal extensions of the exported rules may be used, leading to a powerful control mechanism of the P systems behaviour, especially useful for simulation purposes.

2.3 An implementation

A quick implementation comes almost for free, thanks to:

- an implementation of P systems (with structural data) in K (subsequently implemented as an extension of Maude), see [13];
- the definitions (and the resulting automatic implementations) of various programming languages in K.

For instance, one can use the concise, but powerful concurrent, object-oriented programming language KOOL with the syntax defined in Table 1. Its simple K-implementation may be found in [10].

Program	$P ::=$	$C^* \ E$
Class	$C ::=$	class X is D^* M^* end $\mid$ class X extends X' is D^* M^* end
Decl	$D ::=$	var $\{X,\}^+$;
Method	$M ::=$	method X is D^* S end $\mid$ method X $(\{X',\}^+\ )$ is D^* S end
Expression	$E ::=$	$X \mid I \mid F \mid B \mid Ch \mid Str \mid (E)$ $\mid$ new X $\mid$ new X $(\{E,\}^+)$ $\mid$
		self $\mid$ $E\ X_{op}\ E'$ $\mid$ $E.X(())^?$ $\mid$ $E.X(\{E,\}^+)$ $\mid$ super() $\mid$
		super.$X(())^?$ $\mid$ super.$X(\{E,\}^+)$ $\mid$ super($\{E,\}^+$) $\mid$ primInvoke($\{E,\}^+$)
Statement	$S ::=$	E <- E'; $\mid$ begin D^* S end $\mid$ if E then S (else S')$^?$ fi $\mid$
		try S catch X S end $\mid$ throw E ; $\mid$ while E do S od $\mid$
		for X <- E to E' do S od $\mid$ break; $\mid$ continue; $\mid$
		return $(E)^?$; $\mid$ S S' $\mid$ E; $\mid$ assert E; $\mid$ X: $\mid$ spawn E ; $\mid$
		acquire E ; $\mid$ release E ; $\mid$ typecase E of Cs^+ (else S)$^?$ end
Case	$Cs ::=$	case X of S

$$X \in \mathit{Name}, I \in \mathit{Integer}, F \in \mathit{Float},\ B \in \mathit{Boolean}, Ch \in \mathit{Char}, Str \in \mathit{String},$$
$$X_{op} \in \mathit{Operator\ Names}$$

Table 1: KOOL Syntax

3 P system with control nuclei: dynamic membrane structure

A particular technical issue, avoided in the presentation in the previous section by restricting the class of transition P systems, is the impact of the application of a rule which dissolves the membrane to the nucleus programs: What happens with the nucleus program of the dissolved membrane? The opposite event of membrane division seems is less complicate due to the high level of abstraction used in our nucleus programming model: just copy the program in the daughter membranes (for implementation, one can use a kind of Unix `fork` statement to copy the program, i.e., copy its code, its current control point, and the memory variables with their current values). As we will discuss in the next section, when a less abstract model is to be used for nucleus programs, as for instance the one suggested by the control mechanisms engaged in the biological cell growth and division processes, membrane division may be equally problematic.

3.1 Membrane dissolution

A few options to deal with membrane dissolution are sketched in this section, grouped in two classes:

1. a further extension of PCNs to have more nuclei (and nucleus programs) associated to a membrane;

2. the application of certain rules to combine the nucleus program of the dissolved membrane with the nucleus program of the parent membrane;

3. ignore the nucleus program of the disolved membrane, as in traditional P systems.

Classical transition P systems also use membrane dissolution rules

$u \to \overline{v}\delta$
with $u \in O^+$ and $\overline{v} \in (O \times Tar)^*$, where $Tar = \{here, in, out\}$

Traditionally [8], the presence of δ in these rules indicates that the application of such a rule leads to the dissolution of the membrane and to the abolishment of the rules associated with the membrane just dissolved. We can better handle this case using our model with control nuclei and provide a rich set of options for the semantics of the dissolution rule.

One solution to this problem is to further extend the previously introduced rPCNs by allowing multiple programs in a nucleus, working in parallel to generate the rules for its membrane region.

Definition 3.1 (P system with control nuclei using multiple parallel programs) P systems with control nuclei using parallel programs (PCN-Par's, for short) is the extension of restricted P systems with control nuclei (rPCN's), where each membrane has an associated nucleus with a set of programs acting in parallel for generating the rules to be used within its membrane region.

The semantics of a PCN-Par is similar with that of PCN's, but now the set of rules to be generated for a membrane region by its nucleus is obtained using the programs working in parallel in the nucleus. □

When sets of programs working in parallel are used in the nuclei, the default meaning is the synchronous behavior: all programs have to reach the export stage and the collected rules are exported. (Interleaving semantics may be considered, as well, where one program is activate at each round and only its `export` statement contributes to the rules to be exported in the membrane region in that round.)

Another approach to the membrane dissolution problem is sketched below. Instead of using multiple programs obtained by adding the nucleus program(s) of the dissolved membrane to the nucleus program(s) of the parent membrane, one can combine the programs to get a unique program. This way, the model with a single nucleus program in the previous section is preserved, provided the language for nucleus programs is rich enough to accommodate for the new combinations of programs required.

This option opens a full range of possibilities to combine nucleus programs. The composition of the parent and daughter programs may be done using various operators, including the following:

- sequential composition - the program of the dissolved membrane is added to the top or the bottom of the parent membrane program;

- interleaving composition - the programs are composed by interleaving, i.e., at each step nondeterministically one of the programs is activated to generate the rules;
- a unique composite program is created, but having multiple control points acting in parallel.

3.2 Other active membrane operations

Other classes of interesting P systems bring an important additional feature: the possibility to dynamically change the membrane structure. The membranes can evolve themselves, either changing their characteristics or getting divided. In this subsection we briefly discuss the impact of two such features when combined with control nuclei: membrane division and membrane migration.

In a PCN, membrane division mainly reduces to the duplication of the membrane contents [or, depending on the rule, the partition of its contents] and a replication of the nucleus program (a `fork`-like replication: each child membrane has an identical copy of the parent membrane nucleus program, with the same code, the same position of the control point, and the same variables and with the same values as in the parent membrane).

The migration operator (endocytosis/exocytosis or gemmation, see [8]) is also unproblematic for PCNs, as during such a transformation a nucleus program remains attached to the same membrane. While the membrane can be moved from one place to another (with possible changes of the contents), the impact on the system is reflected in the nucleus programs - these programs become aware on the new membrane structure when generating rules.

4 Modeling cell normal and abnormal development

In this section, we briefly describe how PCNs can be used to model cell growth and division in biological systems, both in normal and abnormal (tumoral) developments. The reader is directed to [7] for further explanation and details about the terms, concepts, and phenomena used in this section.

The abstract development of PCNs in the previous sections, based on programs written in conventional programming languages, is better suited for "in silico" models. To tackle "in vivo" systems, we present a particular low-level DNA-based biological representation of the nucleus programs and of the transformation rules. The resulting systems are named biological P systems with control nuclei (BPCNs for short).

The transformation rules associated to membranes in BPCNs include rules with the following format

$a \rightarrow p(c)$ if c.

Such a rule represents an abstract formulation of a part of the transcription process where the DNA/RNA code c is used to transform the aminoacids in a into $p(c)$, the protein represented by c. It should be noted that depending on whether the quantitative or qualitative PCN model is used, a single c can be used for only one or for several transformations of a's in $p(c)$'s, respectively. To be consistent with the PCN semantics previously developed, the code c has to become inactive at the end of the transformation step, either being degraded or moved in a trash/inactive area (for instance in the nucleus). From a biological perspective, it is not clear why a code c is to be lost at the end of a transformation step, and perhaps a variation of the model where a code c is allowed to be active during several transformation steps is more appropriate.

Like in standard P systems, the result $p(c)$ of a rule can migrate into another membrane. Unlike standard P systems, in BPCNs a protein $p(c)$ can also migrate into control nuclei and can be attached to specific spots of the DNA, with an activation or inhibition result of the related gene. From the nucleus program point of view, this situation is nothing more than the ability of the program to use tests: depending on the value of a particular variable (e.g., a protein sitting in the cell or in the nucleus) its behavior follows a specific path from a set of possible continuations.

The nucleus consists of a strand of DNA, a sequence of basic nucleotides separated in "genes," each gene codifying a protein. On the DNA strand, several proteins are attached at specific spots. In a current configuration, the DNA strand has one or more control points where active genes are copied and exported into the membrane region for transcription. The specific program (or mechanism) used to get the position of the control points for the active genes used in a next transcription step is left unspecified at this moment. (A simple option could be that each control point travels along the DNA strand and stops at the first active gene. However, this is an oversimplification, as it does not take into account the dynamics and the timing of the attachment of the proteins to the DNA strand, or the insertion or deletion of new control points.)

While in P systems one could use an expensive abstract rule to duplicate a membrane and its contents, in BPCNs one could develop a more detailed mechanism for cell growth and division, closer to the real processes seen in biological systems. The payoff for this effort of having a detailed representation of the division process is that one could also model and study abnormal (tumoral) development of the cells.

According to [7], the cell cycle consists of the following phases (see Figure 2):

- (G0) - a commitment is taken towards a division process;
- (G1) - this is a growth stage where RNA and proteins are synthesized;
- (S) - this phase contains DNA replication;
- (G2) - during this period, the cell gets two complete diploid sets of chromosomes;

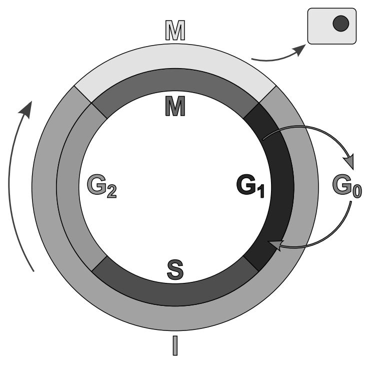

Figure 2: The cell cycle

- (M-mitosis) - here the nucleus is dissolved and the daughter cells are created.

Abstract versions of most of these processes can be modeled in BPCNs as follows:

- (G0) - a starting control point is inserted in a code at a point where the division is codified;
- (G1) - a duplication rule $x \rightarrow x\ x$, where each membrane component gets a copy, may be used for this growth stage;
- (S) - as transformation rules take place in membrane regions only, the DNA duplication is slightly complicated: the full DNA code is exported into the membrane region, duplicated there, and finally moved back into the nucleus;
- (G2) - the abstract version of this stage is not completely clear at this moment - it may have to deal with the need to have the same "control points" in both copies of the DNA strands (as in Unix processes obtained by the `fork` command);
- (M) - in this stage the nucleus is divided into two (this is opposite to the previously mentioned process of joining two nuclei); finally, use a rule to divide a membrane in two membranes, with an even separation of its contents into the daughter membranes.

The described BPCNs for development of the cells and their division could be easily adapted to take into account tumor attacks. The result of a DNA tumor viral infection, roughly falls into two categories:

- (1) permissive cells allow for multiplication of the DNA virus, then the cell dies and the viral DNA is spread into the neighboring cells;

- (2) nonpermissive cells may sometimes be infected by the insertion of the viral DNA into the nucleus DNA, changing the cell phenotype (in particular, after cell division, the daughter cells inherit an infected nucleus).

To conclude, P systems with control nuclei, in both their abstract and more biologically motivated forms, promise to be a good candidate for modeling, simulating, and understanding the evolution of complex (including biological) systems.

5 Concluding remarks

At a first glance it looks like this paper rises more questions than it solves. In our view, the main contribution of our approach is the introduction of a powerful high level modeling and programming environment where experiments with various options with inspiration from biology can be easily done. Together with interested researchers from biology or related area we hope to find and study a more detailed model able to cope with the complexity of real biological phenomena.

References

References

[1] Alberts, B., Bray, D., Lewis, J., Raff, M., Roberts, K., Watson J.D.: Molecular Biology of the Cell, 3rd ed. Garland Publishing, New York (1994).

[2] Andrei, O., Ciobanu, G., Lucanu, D.: Expressing Control Mechanisms of Membranes by Rewriting Strategies. In: Proc. Workshop on Membrane Computing 2006, LNCS 4361, 154–169. Springer (2006).

[3] Clavel, M., Durn, F., Eker, S., Lincoln, P., Mart-Oliet, N., Meseguer, J., Talcott, C.: All About Maude - A High-Performance Logical Framework, LNCS 4350. Springer (2007).

[4] Drăgoi, C., Ștefănescu, G.: AGAPIA v0.1: A programming language for interactive systems and its typing systems. In: Proc. Foundation of Interactive Computation, ETAPS 2007. Electronic Notes in Theoretical Computer Science, 203, 69–94 (2008).

[5] Frisco, P.: The Conformon-P System: A Molecular and Cell Biology-Inspired Computability Model. Theoretical Computer Science, 312, 295–319(2004)

[6] Hills, M., Roșu, G.: KOOL: An Application of Rewriting Logic to Language Prototyping and Analysis. In: Proc. RTA 2007, LNCS 4533, 246–256. Springer (2007).

[7] Lewin, B.: Genes VIII. Oxford University Press (2004).

[8] Păun, G.: Computing with membranes. Journal of Computer and System Sciences, 61, 108–143 (2000).

[9] Păun, G.: Introduction to Membrane Computing. 12th Estonian Winter School in Computer Science, 2007. http://psystems.disco.unimib.it/download/MembIntro2004.pdf

[10] Roșu, G.: K: A Rewriting-Based Framework for Computations – Preliminary version. Technical Report UIUCDCS-R-2007-2926, Department of Computer Science, University of Illinois (2007). http://fsl.cs.uiuc.edu/k.

[11] Roșu, G., Șerbănuță, T.F.: An Overview of the K Semantic Framework. J. of Logic and Algebraic Programming, 79(6), 397–434 (2010)

[12] Russel, S., Norvig, P.: Artificial Intelligence: A Modern Approach. Prentice Hall, 2nd edition, (2002).

[13] Șerbănuță, T., Ștefănescu, G., Roșu, G.: Defining and Executing P systems with Structured Data in K. In: Proc. Workshop on Membrane Computing 2008, LNCS 5391, pp374–393. Springer, Berlin (2009).

[14] Ștefănescu, G.: Interactive systems with registers and voices. Fundamenta Informaticae, 73, 285-306 (2006).

[15] URL: The Web Page of Membrane Computing: http://ppage.psystems.eu/

[16] Wooldrige, M.: An Introduction to Multiagent Systems. Wiley & Sons (2002).