Computer Science Journal of Moldova, vol.**20**, no.2(59), 2012

Self-Stabilization in Membrane Systems*

Artiom Alhazov Marco Antoniotti Rudolf Freund
Alberto Leporati Giancarlo Mauri

Abstract

In this paper we study a notion of self-stabilization, inspired from biology and engineering. Multiple variants of formalization of this notion are considered, and we discuss how such properties affect the computational power of multiset rewriting systems.

1 Introduction

Membrane systems, also called P systems, are a framework for (bioinspired) computational models, see [9], [10] and [14]. In this paper we consider a one-region rewriting model with symbol objects. In this case, membrane computing can be considered as (maximally parallel or sequential) multiset processing. In general, a computation is a sequence of transitions between configurations. Configurations are multisets, and the transitions are induced by rules, defined by reactants, products and control (additional applicability conditions, if any), viewed as formal computational systems (generating/accepting numeric/vector sets, or computing functions).

We will call a property dynamic if it depends on the behaviour of a system and cannot be easily derived from its description (as opposed to syntactic properties). Given any finite computation, we assume that the property is easily verifiable. The two usual sources of undecidability are a) that we do not always know whether we are dealing with finite

* The first author acknowledges the project RetroNet by the Lombardy Region of Italy under the ASTIL Program (regional decree 6119, 20100618). The work of the second, the fourth and the fifth author was partially supported by Università degli Studi di Milano-Bicocca, Fondo di Ateneo per la Ricerca (FAR) 2011.

or infinite computations, and b) that some properties are defined on infinite number of computations (due to non-determinism, to the initial input or to some other parameter). In the case of this paper, another source of potential undecidability is the finite set to be reached as given in the definitions below.

Since in this paper we will deal with reachability issues, we would also like to mention the connection with temporal logic [5].

Self-stabilization is a known concept in conventional distributed computing, introduced by E. Dijkstra in [4], as well as in systems biology, but not yet considered in the framework of membrane computing. Other works on self-stabilization are [12], [1], [7], [13], [6], [3], to name a few. It has been recalled by Jacob Beal during the Twelfth Conference in Membrane Computing, CMC12, and an attempt to formalize it in the membrane computing framework has been done in [2]. The underlying idea is the tolerance of natural and engineering systems to perturbations. The formulation from [15] says:

> A system is self-stabilizing if and only if:
>
> 1. Starting from any state, it is guaranteed that the system will eventually reach a correct state (convergence).
>
> 2. Given that the system is in a correct state, it is guaranteed to stay in a correct state, provided that no fault happens (closure).

In case of inherently non-deterministic systems, "with probability 1" should be added. Based on this concept, we propose to consider a few formal properties, following the discussion below.

In this paper we consider fully cooperative multiset rewriting, possibly with promoters/inhibitors/priorities, operating either in the maximally parallel or the sequential mode. We consider a single working region only, for two reasons. First, the properties of interest are unaffected by flattening the static membrane structure. Second, we would currently like to avoid the discussion about reachability related to "arbitrary configurations" with dynamic membrane structure.

2 Definitions

We assume the reader to be familiar with the basics of formal language theory, e.g., we refer to [11].

An *alphabet* is a finite non-empty set V of abstract *symbols*. The free monoid generated by V under the operation of concatenation is denoted by V^*; the *empty string* is denoted by λ, and $V^* \setminus \{\lambda\}$ is denoted by V^+. The family of all finite (recursive, recursively enumerable) sets of positive integers is denoted by $NFIN$ ($NREC$, NRE, respectively).

2.1 Membrane systems

A one-region (rewriting) membrane system is a tuple

$$\Pi = (O, w, R)\,,$$

where O is a finite alphabet, $w \in O^*$ is a string representing the initial multiset, and R is a set of rules of the form $r : u \to v$, $u \in O^+$, $v \in O^*$.

A configuration of the system is represented by a multiset of objects from O contained in the region, and a rule $r : u \to v$ is applicable if the current configuration contains the multiset specified by u. Furthermore, applicability may be controlled by promoters ($r : u \to v|_a$), inhibitors ($r : u \to v|_{\neg b}$), or priorities ($r' > r$). Throughout the paper, we will use the word *control* to mean that at least one of these three features is allowed. In such cases, in addition to the availability of u for a rule r to be applicable, the promoter a must be present in the current configuration, the inhibitor b has to be absent in the current configuration, and no rule r' with higher priority than r is allowed to be applicable, respectively.

A computation step in the sequential mode consists of the nondeterministic application of one applicable rule, replacing its left side with its right side. In the maximally parallel mode, multiple applicable rules have to be applied multiple times, to disjoint submultisets, in a non-deterministic way, possibly leaving some objects idle, under the condition that no further rule is applicable to them. The computation step is denoted by the binary relation $\Rightarrow$. A computation halts when no rule is applicable to the current configuration (halting configuration).

For a generating system, the result of a halting computation is the total number of objects in the system when it halts. The set of numbers generated by a P system is the set of results of its computations. An accepting system is described as (O, Σ, w, R), where Σ is an input alphabet: instead of w, the computation starts with wx, $x \in \Sigma^*$, and its result is $|x|$ if it halts. The set of numbers accepted by a P system is the set of results of its computations for all $x \in \Sigma^*$.

2.2 Self-stabilization and related properties

We now resume the discussion started at the end of the Introduction.

Clearly, "a correct state" should be rephrased as "a configuration in the set of correct configurations". Moreover, we would like to eliminate the set of correct states, let us denote it by S, as a parameter. We say that our property holds if there exists some finite set S of configurations satisfying the conditions 1 and 2 above. Since membrane systems are inherently non-deterministic, we additionally propose two weaker degrees of such a property: possible (there exists a computation satisfying the conditions), almost sure (the conditions are satisfied with probability 1 with respect to non-determinism). Finally, if condition 2 is not required, we call the corresponding property (finite) set-convergence instead of self-stabilization. We now give the formal definitions from [2].

Definition 1 *A P system* Π *is possibly converging to a finite set* S *of configurations iff for every configuration* C *of* Π *there exists a configuration* $C' \in S$ *such that* $C \Rightarrow^* C'$.

Definition 2 *A P system* Π *is (almost surely) converging to a finite set* S *of configurations iff for every configuration* C *of* Π *the computations starting in* C *reach some configuration in* S *(with probability 1, respectively).*

Definition 3 *A P system* Π *is possibly closed with respect to a finite set* S *iff for every non-halting configuration* $C \in S$ *there exists a configuration* $C' \in S$ *such that* $C \Rightarrow C'$.

Definition 4 *A P system* Π *is closed with respect to a finite set* S *iff for every non-halting configuration* $C \in S$ $C \Rightarrow C'$ *implies* $C' \in S$.

We say that a system is **(possibly, almost surely) set-converging** if it is (possibly, almost surely, respectively) converging to some finite set of configurations.

We say that a system is **possibly self-stabilizing** if it is possibly converging to some finite set S of configurations and if it is possibly closed with respect to S.

We say that a system is **(almost surely) self-stabilizing** if it is (almost surely, respectively) converging to some finite set S of configurations and if it is closed with respect to S.

The examination of computational aspects of these properties motivates us to add "weakly" to the properties proposed in [2] – *(possibly, almost surely) converging, (possibly) closed, (possibly, almost surely) set-converging, (possibly, almost surely) self-stabilizing* – if the corresponding conditions over configurations C only span the **reachable non-halting** ones.

Another comment we can make on "almost sure" is that such a property may depend on how exactly the transition probability is defined. The easiest way is to assign equal probabilities to all transitions from a given configuration. Alternatively, to a transition via a multiset of rules $r_1^{n_1} \cdots r_m^{n_m}$ we may assign the weight of a multinomial coefficient $\binom{n_1+\cdots+n_m}{n_1,\cdots,n_m} = \frac{(n_1+\cdots+n_m)!}{n_1!\cdots n_m!}$, which will make the corner cases less probable than the average ones. There can be other ways to define transition probabilities, but we would like to discuss the properties of interest without fixing a specific way. We assume the transition probabilities in an independent subsystem are the same as if it were the entire system.

An important assumption we impose on the probability distribution is that *the probability of each transition is uniquely determined by the associated multiset of rules and by the set of all applicable multisets of rules*, yet it does not depend on the objects that cannot react, or by the previous history of the computation.

2.3 Register machines

In what follows we will need to simulate register machines; here we briefly recall their definition and some of their computational properties. A *register machine* is a tuple $M = (m, B, l_0, l_h, P)$, where m is the number of registers, P is the set of instructions bijectively labeled by elements of B, $l_0 \in B$ is the initial label, and $l_h \in B$ is the final label. The instructions of M can be of the following forms:

- $l_1 : (ADD(j), l_2, l_3)$, with $l_1 \in B \setminus \{l_h\}$, $l_2, l_3 \in B$, $1 \leq j \leq m$
  Increase the value of register j by one, and non-deterministically jump to instruction l_2 or l_3. This instruction is called *increment*.
- $l_1 : (SUB(j), l_2, l_3)$, with $l_1 \in B \setminus \{l_h\}$, $l_2, l_3 \in B$, $1 \leq j \leq m$
  If the value of register j is zero then jump to instruction l_3, otherwise decrease the value of register j by one and jump to instruction l_2. The two cases of this instruction are called *zero-test* and *decrement*, respectively.
- $l_h : HALT$. Stop the execution of the register machine.

A register machine is *deterministic* if $l_2 = l_3$ in all its ADD instructions. A *configuration* of a register machine is described by the contents of each register and by the value of the program counter, which indicates the next instruction to be executed. Computations start by executing the first instruction of P (labelled with l_0), and terminate with reaching a $HALT$-instruction.

Register machines provide a simple universal computational model [8]. Register machines can be used as *accepting* or as *generating* as well as as decision devices. In accepting register machines, a vector of non-negative integers is accepted if and only if the register machine halts having it as input. Usually, without loss of generality, we may assume that the instruction $l_h : HALT$ always appears exactly once in P, with label l_h. In the generative case, we start with empty registers and take as results of all possible halting computations. Being used as decision devices, register machines may halt in an accepting state with label l_{yes} or in a rejecting state l_{no}, respectively. In the following,

we shall call a specific model of P systems *computationally complete* if and only if for any register machine M we can effectively construct an equivalent P system Π of that type simulating each step of M in a bounded number of steps and yielding the same results.

3 Results

3.1 Accepting systems

For the following theorem we consider any computationally complete model of P systems as defined above, e.g., a model with maximally parallel multiset rewriting or with controlled sequential multiset rewriting.

Theorem 1 *If a model of P systems yields a computationally complete class, then the weakly self-stabilizing subclass accepts exactly $NREC$.*

Proof. For any recursive number set there is a register machine M with one accepting state q_{yes} and one rejecting state q_{no}, deciding it. We modify the register machine in order to obtain a register machine M' which, once the decision is made, i.e., q_{yes} or q_{no} has been reached, erases the workspace and then enters q'_{yes} or q'_{no} respectively, thereby halting in q'_{yes} if and only if the input is accepted or performing an infinite loop with $q'_{no} : (SUB\,(1)\,, q'_{no}, q'_{no})$ if and only if the input x is rejected. This register machine M' now can be simulated with a P system Π, which by construction starts with a configuration representing the input x and will either end with halting in a configuration representing the state q'_{yes} or else looping in a configuration representing the state q'_{no}, i.e., Π is weakly self-stabilizing.

Conversely, consider a self-stabilizing P system Π, i.e., for each input x, Π performs a computation that ends up in a configuration from a finite set S and then cannot reach any other configuration outside S. Now consider the derivation graph for all possible computations of Π on the input x, i.e., the nodes of this directed graph represent the configurations and the edges indicate the derivation steps from one configuration to the next one during one of these computations. As the number of configurations directly derivable from any configuration

in Π is finite, this derivation graph is a connected directed graph with finite degree (from each node, only a finite number of edges is leaving); moreover, this graph cannot have a simple path (a path visiting each node at most once) which is infinite, as every computation in Π has to reach a configuration (node) from S and then cannot leave the set of configurations S any more. Due to König's lemma[1], the total number of nodes (configurations) in the derivation graph must be finite. Hence, even without knowing the set S, the bruteforce algorithm computing all possible transitions from the initial configuration, but halting as soon as the system halts or a configuration already passed previously is reached, yields a decision procedure for the set accepted by Π. □

Strengthening this result by removing "weakly" is problematic, even if more powerful P systems are used. Indeed, self-stabilization also from unreachable configurations would need to handle not only the configurations without any state or with multiple states (which could be handled with the joint power of maximal parallelism and priorities), but also configurations representing a situation with only one state which is not the initial state of the underlying register machine. We have to leave this question open.

Theorem 2 *If a model of P systems yields a computationally complete class, then the weakly almost surely self-stabilizing P systems of this class accept exactly NRE.*

Proof. We start with the construction from Theorem 1. We want to show that relaxing the property "weakly self-stabilizing" to "almost surely" leads from recursiveness to computational completeness. It suffices to handle the case when the system rejects the input by never halting. We modify the underlying register machine as follows: add a non-deterministic transition from every state $p \in Q$ to a new state e, from e erase the contents of all registers and then jump

[1]König's lemma: Let G be a connected graph with finite degree. If G contains an infinite number of nodes, then it contains an infinite simple path.

back to e; this will not affect the accepting power, but it will provide a self-stabilizing path from any reachable non-halting configuration. The transition from p to e can be done by $p : (ADD(j), e, e)$, since the registers then are emptied anyway. Furthermore, the basic model of register machines does not allow non-determinism other than $p : (ADD(j), q, r)$. The branching at ADD instructions may be done by assuming the original computation to be deterministic and replacing $p : (ADD(j), q, q)$ by $p : (ADD(j), q, e)$. The branching at a SUB instruction $p : (SUB(j), q, r)$ may be done by the sequence of rules $p : (ADD(j), e, p')$, $p' : (SUB(j), p'', p'')$, $p'' : (SUB(j), q, r)$.

The probability that the computation does not self-stabilize for more than k steps decreases exponentially with respect to k. Indeed, the simulation of a register machine by P system has bounded parallelism, each instruction is simulated in a bounded number of steps, and at least one path leads to self-stabilization. Moreover, there only exists a finite number of different sets of applicable multisets containing a branching from the simulation into the self-stabilization path, so the minimum probability for this self-stabilization path is strictly positive. These observations conclude the proof. □

Theorem 3 *If a model of P systems yields a computationally complete class, then the class of all almost surely self-stabilizing maximally parallel/sequential P systems with priorities accepts exactly NRE.*

Proof. Given a set L from NRE, we first construct a P system Π simulating a register machine M accepting L and then extend Π to a P system Π' even fulfilling the condition of almost surely self-stabilizing.

Let $M = (m, B, l_0, l_h, P)$ a deterministic register machine accepting L. We now construct the P system $\Pi = (O, l_0, R, >)$ with priorities accepting L:

$$\begin{aligned} O &= B \cup \{a_i \mid 1 \leq i \leq m\}, \\ R &= \{l_1 \to a_j l_2 \mid l_1 : (ADD(j), l_2) \in P\} \\ &\cup \{a_j l_1 \to l_2, l_1 \to l_3 \mid l_1 : (SUB(j), l_2, l_3) \in P\} \\ > &= \{a_j l_1 \to l_2 > l_1 \to l_3 \mid l_1 : (SUB(j), l_2, l_3) \in P\}. \end{aligned}$$

The contents of a register i, $1 \leq i \leq m$, is represented by the number of symbols a_i in Π. The state l of the register machine is represented by the corresponding symbol l in Π, too. When M halts in l_h with all registers being empty, Π also halts with the configuration $\{l_h\}$. Obviously, Π accepts L, both in the sequential as well as in the maximally parallel mode.

To strengthen the result to even non-weak almost sure self-stabilization, we have to take into account the non-reachable configurations, too. The almost surely self-stabilizing P system $\Pi' = (O', l_0, R', >')$ with priorities accepting L is constructed as follows:

$$
\begin{aligned}
O' &= B \cup \{a_i \mid 1 \leq i \leq m\} \cup \{e\}, \\
R' &= \{l_1 \to a_j l_2 \mid l_1 : (ADD(j), l_2) \in P\} \\
&\cup \{a_j l_1 \to l_2, l_1 \to l_3 \mid l_1 : (SUB(j), l_2, l_3) \in P\} \\
&\cup \{a_i \to e \mid 1 \leq i \leq m\} \cup \{ex \to e \mid x \in O'\} \cup \{e \to e\} \\
&\cup \{l \to e \mid l \in B \setminus \{l_h\}\} \cup \{ll' \to e \mid l, l' \in B\}, \\
>' &= \{a_j l_1 \to l_2 > l_1 \to l_3 \mid l_1 : (SUB(j), l_2, l_3) \in P\} \\
&\cup \{ex \to e > r, ll' \to e > r \mid l, l' \in B, x \in O', r \in R\} \\
&\cup \{l \to e > a_i \to e \mid l \in B \setminus \{l_h\}, 1 \leq i \leq m\} \\
&\cup \{r > e \to e \mid r \in R' \setminus \{e \to e\}\}.
\end{aligned}
$$

In addition to the idea of the construction given in the proof of Theorem 2 using the exit e by applying a rule $l \to e$, $l \in B \setminus \{l_h\}$, it suffices to self-stabilize from the configurations with no state and from the configurations with multiple states of the register machine. Multiple states can be reduced by the rules $ll' \to e$, $l, l' \in B$. If no state symbol is present, then we may exit with one of the rules $a_i \to e$, $1 \leq i \leq m$. All remaining cases can be captured by the rules $ex \to e$, $x \in O'$. By construction, the self-stabilizing set S equals $\{\{l_h\}, \{e\}, \emptyset\}$. The whole construction again is valid for the sequential as well as the maximally parallel mode. □

An open question is whether priorities in Theorem 3 can be replaced by promoters or inhibitors.

3.2 Generating systems

Theorem 4 *Any finite set M of numbers can be generated by some self-stabilizing membrane system without control.*

Proof. Consider a P system $\Pi = (\{s, a\}, s, R)$, where

$$R = \{s \to a^n \mid n \in M\} \cup \{a^{\max(M)+1} \to \lambda, ss \to s\}.$$

It is not difficult to see that Π generates M and (taking $S = \{a^n \mid n \leq \max(M)\} \cup \{s\}$) it is self-stabilizing. □

Since self-stabilization implies set-convergence and closure, and relaxing either property (to possibly, almost surely and/or weakly) does not compromise the construction of the P system descibed in the proof of Theorem 4, the lower bound on the generative power of associated systems restricted to any property we have defined, is at least $NFIN$.

Lemma 1 *A possibly finite set-converging system only generates finite sets.*

Proof. By Definition 1, for a system possibly converging to a set S, S contains all halting configurations. Since S is finite, so is the set of all the halting configurations. Hence, at most $NFIN$ is generated. □

Theorem 5 *Any of the following classes of P systems* $\mathbf{dp}OP^{\mathbf{m}}(\mathbf{c})$ *generate exactly $NFIN$:*

d *is possibly/almost surely/ -*
p *is self-stabilizing/finite set-converging*
m *is maximally parallel/sequential*
c *is uncontrolled/with promoters/with inhibitors/with priorities.*

Proof. The claims directly follow from Theorems 4 and 5. □

We now proceed to weak properties of generative systems.

Theorem 6 *Weakly almost surely self-stabilizing P systems generate exactly $NFIN$.*

Proof. The lower bound is shown by Theorem 4. Now take a weakly self-stabilizing P system Π, and its associated set S from the definition of the property. Consider an arbitrary halting computation of Π. Let C be the configuration of Π one step before the halting. Interpreting finite set-convergence for C implies that the halting configuration belongs to S. Since the halting computation has been arbitrarily chosen, the set of all halting configurations is a subset of S, and hence it is finite. Therefore, the set generated by Π is finite, too. □

Theorem 7 *If a model of P system yields a computationally complete class, then weakly possibly self-stabilizing subclass generates NRE.*

Proof. Consider the construction from Theorem 2, but for a generative P system. The simulation of the underlying register machine is carried out until some point. Unless the P system has already halted, it always has a choice to self-stabilize and loop. □

4 Conclusions

We have presented some results (some of them summarized in Table 1) concerning the notion of self-stabilization, recently proposed for membrane computing. Its essence is reachability and closure of a finite set.

Property	comput. complete	(sequ/maxpar) +pri	Thm
self stabilizing	acc. $?/F$		-/5
almost surely s.s.	acc. $?/F$	acc. NRE/F	3/5
possibly s.s.	acc. $?/F$	acc. NRE/F	3/5
weakly s.s.	acc. $NREC/F$		1/6
weakly almost surely s.s.	acc. NRE/F		2/6
weakly possibly s.s.	acc. NRE/gen. NRE		2/7

Table 1. Results (letter F stands for "generate exactly $NFIN$").

One of the questions we proposed is whether priorities may be replaced by promoters or inhibitors in Theorem 2. Another open question is the power of accepting with unrestricted self-stabilization, even if maximal parallelism is combined with priorities (a comment after Theorem 1 and the first question mark in the table above). The other open questions are also marked with question marks in the table above. Any system in the corresponding classes must (besides doing the actual computation) converge (definitely, in probability or possibly) to some finite set from anywhere, without using the joint power of maximal parallelism and control.

We mention two topics that we did not deal with here. One is considering the finite set as a parameter, possibly leading to a discussion in model checking. The other one concerns reachability questions in dynamic membrane structures.

References

[1] Special Issue on Self-Stablization (A. Arora, Sh. Dolev, W.-P. de Roever, Eds.). *Distributed Computing* **20(1)**, 2007, 93pp.

[2] A. Alhazov: Properties of Membrane Systems. *Membrane Computing*, 12th International Conference, CMC 2011, Fontainebleau, Revised Selected Papers (M. Gheorghe, Gh. Păun, G. Rozenberg, A. Salomaa, S. Verlan, Eds.), *Lecture Notes in Computer Science* **7184**, 2012, 1–13.

[3] Z. Collin, Sh. Dolev: Self-stabilizing Depth-first Search. *Information Processing Letters* **49(6)**, 1994, 297-301.

[4] E.W. Dijkstra: Self-stabilizing Systems in Spite of Distributed Control. *Communication of the ACM* **17(11)**, 1974, 643-644.

[5] E.A. Emerson: Temporal and Modal Logic. In: *Handbook of Theoretical Computer Science*, Chapter 16, the MIT Press, 1990.

[6] T. Herman: Probabilistic Self-stabilization. *Information Processing Letters* **35(2)**, 1990, 63–67.

[7] Special Issue on Self-Stabilization (M. Merritt, Ed.). *Distributed Computing* **7(1)**, 1993, 66pp.

[8] M.L. Minsky: *Finite and Infinite Machines*. Prentice Hall, Englewood Cliffs, New Jersey, 1967.

[9] Gh. Păun: *Membrane Computing. An Introduction*. Springer, 2002.

[10] Gh. Păun, G. Rozenberg, A. Salomaa: *The Oxford Handbook of Membrane Computing*, Oxford University Press, 2010.

[11] G. Rozenberg, A. Salomaa: *Handbook of Formal Languages*, 3 vol., Springer, 1997.

[12] M. Schneider: Self-stabilization. *ACM Computing Surveys* **25(1)**, 1993, 45–67.

[13] T. Weis, A. Wacker: Self-stabilizing Automata. *Biologically-Inspired Collaborative Computing*, IFIP International Federation for Information Processing **268**, 2008, Springer, 59–69.

[14] P systems webpage: `http://ppage.psystems.eu`

[15] `http://en.wikipedia.org/wiki/Self-stabilization`

Information about authors:

A. Alhazov[1,2], M. Antoniotti[1],
R. Freund[3], A. Leporati[1], G. Mauri[1],

[1] Università degli Studi di Milano-Bicocca
Dipartimento di Informatica, Sistemistica e Comunicazione
Viale Sarca 336, 20126 Milano, Italy
E-mail: {`artiom.alhazov`, `marco.antoniotti`,
`alberto.leporati`, `giancarlo.mauri`}`@unimib.it`

[2] Institute of Mathematics and Computer Science
Academy of Sciences of Moldova
Academiei 5, Chişinău MD-2028 Moldova
E-mail: `artiom@math.md`

[3] Faculty of Informatics, Vienna University of Technology
Favoritenstr. 9, 1040 Vienna, Austria
E-mail: `rudi@emcc.at`

Received May 8, 2012