Maximally Parallel Multiset-Rewriting Systems: Browsing the Configurations

Artiom Alhazov

Institute of Mathematics and Computer Science
Academy of Science of Moldova
Str. Academiei 5, Chişinău, MD 2028, Moldova

Research Group on Mathematical Linguistics
Rovira i Virgili University
Pl. Imperial Tàrraco 1, 43005 Tarragona, Spain
E-mail: artiom@math.md, artiome.alhazov@estudiants.urv.es

Summary. The aim of this research is to produce an algorithm for the software that would let a researcher to observe the evolution of maximally parallel multiset-rewriting systems with permitting and forbidding contexts, browsing the configuration space by following transitions like following hyperlinks in the World-Wide Web.

The relationships of maximally parallel multiset-rewriting systems with other rewriting systems are investigated, such as Petri nets, different kinds of P systems, Lindenmayer systems, grammar systems, regulated grammars.

1 Introduction

The study of many formal rewriting systems can be simplified by a *simulator* – a software tool computing the evolution of the system. There are numerous examples of simulators for deterministic and non-deterministic rewriting systems, that are sequential or parallel. In this paper we will focus on the non-deterministic parallel ones. Recall that in the process of the evolution, the system non-deterministically branches (chooses the next configuration among the possible ones).

The need for writing yet another simulator appeared because most of the simulators of *P systems with symbol-objects* (i.e., of maximally parallel distributive multiset-rewriting systems) either simulate just the deterministic/confluent systems or resolve the non-determinism in the random way, see [4, 6, 7, 11, 12, 13, 19]. Clearly, for checking the theorems proved by construction of a system (by testing the examples), it is desired that either all branches of the needed number of steps of the evolution are explored, or one branch is selected by the user from the list of all possible branches at each step. Since the size of the solutions for the first approach may be too big to be studied without further tools, we focus on the latter one.

Thus, the main problem is to compute the set of all possible transitions given the rewriting system with the current configuration. The meaning of the word "browsing" in the title is analogous to its meaning in the World-Wide Web: following the transitions (hyperlinks) between the configurations (pages, documents), optionally remembering the path (history) to be able to come back.

2 Maximally Parallel Multiset Rewriting

For a finite set V (called the alphabet), we will denote the set of all words over V by V^*, the empty word by λ and $V - \{\lambda\}$ by V^+. We refer the reader to [18] for the formal language preliminaries. Without restricting the generality, we can assume that the alphabet is ordered: $V = \{a_1, \cdots, a_k\}$.

A (finite) *multiset* over V is a mapping $M : V \to \mathbb{N}$. For $a \in V$, the number $M(a)$ is called the multiplicity of a in M. In this paper we will usually represent multisets by strings, e.g., M can be represented by $w = \prod_{i=1}^{k} (a_i{}^{M(a_i)})$ (or by any permutation of w since the order of the symbols is not important): the multiplicity $M(a)$ of each symbol a is represented by the number $|w|_a$ of occurrences of the symbol a in w.

A *multiset-rewriting system* is defined as a tuple $G = (V, R, w)$, where V is the alphabet, R is a finite set of rules of the form $r : u \to v$, where $u \in V^+$, $v \in V^*$, r is called the label of the rule and $w \in V^*$ is the initial configuration. The label uniquely defines the rule and the set of all labels is denoted by $Lab(R)$. A *multiset of rules* from R can be represented by a word over $Lab(R)$. Without restricting the generality, we can assume that the rule set is ordered: $R = \{r_1 : u_1 \to v_1, \cdots, r_m : u_m \to v_m\}$.

We now proceed to defining the parallel evolution step. It is said that $\rho = \prod_{j=1}^{m} r_j{}^{m(j)} \in Lab(R)^*$ is applicable for the configuration $w = \prod_{i=1}^{k} (a_i{}^{M(a_i)})$ if $\sum_{j=1}^{m} |u_j|_{a_i} * m(j) \leq M(a_i)$ for any $a_i \in V$, i.e., there is enough symbols in the configuration to perform all the rules in ρ with a corresponding multiplicity. The result of applying ρ on w is

$$w' = \delta(w, \rho) = \prod_{i=1}^{k} (a_i{}^{M(a_i) + \sum_{j=1}^{m} (|v_j|_{a_i} - |u_j|_{a_i}) * m(j)}).$$

In words, w' was obtained from w by removing u_j for $m(j)$ times for all rules r_j, and then inserting v_j for $m(j)$ times for all rules r_j. In other words, symbols from u_j were independently replaced by those from v_j, multiple times.

Notice that w' consists of the symbols that did not react and of the symbols from the right-hand sides of the rules from R. The multiset of rules represented by ρ is *maximal* if it is applicable, and no rule $r_k \in R$ can be applied to the remaining symbols in the same step, i.e., for any $r_k \in R$ there is

$$a_i \in V \text{ with } M(a_i) - \sum_{j=1}^{m} |u_j|_{a_i} * m(j) < |u_k|_{a_i}.$$

The evolution of the system is non-deterministic: it evolves (in one step) from w by any maximal applicable multiset of rules to the corresponding configuration w', denoted by $w \Rightarrow^{\rho} w'$. The superscript ρ may be omitted, and $\Rightarrow^*$ is the reflexive and transitive closure of $\Rightarrow$. For the rewriting system $G = (V, R, w)$, we define the set of sentential forms $SF(G)$ as $\{x \in V^* \mid w \Rightarrow^* x\}$, and we denote by $SF_n(G)$ the set of configurations that can be obtained from w in exactly n steps.

Example 1. Let us consider the grammar $G = (\{S, A, B, C\},\ \{p : \mathbf{SA} \to SAB, q : \mathbf{AB} \to ABC\},\ SAA)$, as well as the following derivations with respect to it:

$$\begin{aligned}
&\mathbf{SA}A \Rightarrow^{p} \mathbf{SAAB} \Rightarrow^{pq} SAABBC\\
&\mathbf{SAAB}BC \Rightarrow^{pq} \mathbf{SAAB}BBCC \Rightarrow^{pq} SAABBBBCCC \Rightarrow^{\cdots}\\
&\mathbf{SAAB}BC \Rightarrow^{pq} S\mathbf{AABB}BCC \Rightarrow^{qq} SAABBBCCCC \Rightarrow^{\cdots}\\
&S\mathbf{AABB}C \Rightarrow^{qq} \mathbf{SAAB}BCCC \Rightarrow^{pq} SAABBBCCCC \Rightarrow^{\cdots}\\
&S\mathbf{AABB}C \Rightarrow^{qq} S\mathbf{AABB}CCC \Rightarrow^{qq} SAABBCCCCC \Rightarrow^{\cdots}.
\end{aligned}$$

Hence, $SF_0(G) = \{SAA\}$, $SF_1(G) = \{SAAB\}$, $SF_{n+2}(G) = \{SA^2B^{2+n-k}C^{1+n+k} \mid 0 \le k \le n\}$, and at any step except the first two either both p and q are applied once, or q is applied twice.

Finally, to increase the power of these systems, let us add promoters and inhibitors to the system, see [10] (called in the regulated rewriting theory permitting and forbidding contexts, respectively, see [14]). The general form of the rules is extended to $r : u \to v|_{p,\neg q}$. The behavior of the system is defined as in the usual case, except for a given configuration w, instead of the whole set R of rules only the set of "active" (promoted and not inhibited) rules is considered. The rule r is promoted if $|w|_a \ge |p|_a$ for all $a \in V$. The rule r is inhibited if $|w|_a \ge |q|_a$ for all $a \in alph(q)$. A rule p without the inhibitor ($q = \lambda$) is never inhibited. A rule without the promoter ($p = \lambda$) is promoted by the definition.

3 The Simulator: Goals and Applications

As it was mentioned in the introduction, we need a method to compute the set of all possible transitions (configurations after 1 step), as a kernel for the configuration browser.

The systems we are considering are the maximally parallel multiset-rewriting systems with promoters/inhibitors. They can be viewed as the *Petri nets* with inhibitor arcs, with 2 differences: promoter arcs are added and the parallelism is maximal, in the sense defined in the previous section. (Notice that the maximality of parallelism can be easily avoided by adding rules $a \to a$ for all $a \in V$).

The maximally parallel multiset-rewriting can also be viewed as non-distributive variant of cooperative P systems with promoters/inhibitors. Encoding

"being in a region i" in object a as, e.g., a_i, one obtains the (distributive) *transitional P systems*. Using the same idea, the *communicative P systems* are reduced to the cooperative multiset-rewriting systems[1]. The behavior of *P systems with active membranes* can be simulated by encoding the polarization of each membrane in one object, which is a 3-stable catalyst for the communication rules and a promoter for the evolution rules.

If we restrict the rules to be non-cooperative (context-free, $|u| = 1$), then we obtain a generalization of $0L$ systems; one can easily use promoters to simulate the behavior of $ET0L$ systems, modulo the order of the symbols. A similar reasoning would establish a link between our systems and *grammar systems*, except in this case the control symbol is a (multi-stable) catalyst, not a promoter. The same is true for simulating the behavior of sequential systems, for example of some *grammars with regulated derivation*. One can find links with other rewriting systems.

The software can be used as a simulator for the systems, a debugger for the theorems proved in a constructive way, a browser of the configurations, a tool for the researcher, or a toy for a student. It was used as an engine of the communicative P systems simulator by Vladimir Rogozhin, to check the theorems in [15] and [1].

4 Problem Reformulation

4.1 Vector representation

In the simulator of the maximally parallel multiset-rewriting systems, the multisets were represented by vectors. A multiset M over $V = \{a_1, \cdots, a_k\}$ can be represented by a vector $(M(a_1), \cdots, M(a_k)) \in \mathbb{N}^k$.

Example 2. The rewriting system

$$G = (\{S, A, B, C\},\ \{p : SA \to SAB, q : AB \to ABC\},\ SAABBC)$$

with $SAABBC \Rightarrow^{pq} SAABBBCC$, $SAABBC \Rightarrow^{qq} SAABBCCC$ can be written in space $\mathbb{N}^4$ as

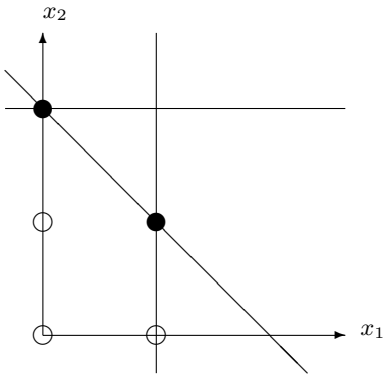
$$V = (4, R, (1, 2, 2, 1)),$$
$$R = \{p : (1, 1, 0, 0) \to (1, 1, 1, 0),\ q : (0, 1, 1, 0) \to (0, 1, 1, 1)\},$$

with $(1, 2, 2, 1) \Rightarrow^{pq} (1, 2, 3, 2)$, $(1, 2, 2, 1) \Rightarrow^{qq} (1, 2, 2, 3)$.

[1] It deserves some attention that the objects E present in the environment in infinite multiplicities (let us denote the set of all objects by O) are not represented in configurations (that are finite), except their copies that are inside the system. Instead, a rule $(ux, out; vy, in)$ assigned to the skin membrane (with label 1), where $u, v \in E^*$ and $x, y \in (O - E)^*$, is converted to a cooperative rewriting rule $h_1(ux)h_0(y) \to h_0(x)h_1(vy)$, where h_i are region-encoding morphisms: $h_i(a) = a_i$, $a \in O$ for $0 \leq i \leq 1$.

4.2 Simplex

A simplex is a part of a finitely dimensional space, which is a set of solutions of a system of linear equations and inequalities, i.e., any intersection of a finite number of hyperplanes and semi-spaces.



$$\begin{cases} x_1 & \leq & 1 \\ x_1 + x_2 & \leq & 2 \\ x_2 & \leq & 2 \\ 0 & \leq & 1 \\ x_1 & \to & max \\ x_2 & \to & max \end{cases}$$

Fig. 1. Multi-criteria problem.

All *multisets* of applicable rules form a simplex.

Example 3. For applying $p^{x_1}q^{x_2}$ to a in $p : (1,1,0,0) \to b,\ q : (0,1,1,0) \to c$, $a = (1,2,2,1)$, where $b, c \in \mathbb{N}^4$, we obtain the following conditions ($x_1 \geq 0,\ x_2 \geq 0$ are assumed):

$$\begin{cases} x_1 & \leq 1 \\ x_1 + x_2 & \leq 2 \\ x_2 & \leq 2 \\ 0 & \leq 1 \end{cases}$$

4.3 Solutions. Maximality

The resulting strategies are the *integer* efficient (Pareto–optimal) solutions of the multi-criteria problem in Figure 1.

5 Program. Examples

5.1 Recursive Approach

We start with the first rule ($r = 1$)

- Calculate *maximal* possible applicable multiplicity x_r
  (1 in the example)

- For *all* numbers from x_r down to 0
  - Calculate *remaining* objects
    (in the example for $x_1 = 1$, $(0, 1, 2, 1)$ remains)
  - Proceed with the *next* rule (if $r < m$)
  - Otherwise, check the *maximality*
    (no more rules applicable to the remaining objects).
    If so, then *display*.

5.2 Drawback. Optimization attempts

Because all space of applicable solutions is searched, the search should be done in a way as efficient as possible. Here are a few ideas in this respect.

Improvement 1: For the *last* symbol, the number x_m is maximal, no loop "for all numbers from x_m down to 0" is needed. The speed-up obtained in this way is minor.

Improvement 2: First calculate the necessary *minimum* of application for all rules, then proceed with the *other* objects. The idea is to consider all possible *total week priorities* (in our example $p \succ q$ and $q \succ p$), calculate the number of applications, consider the minimum.

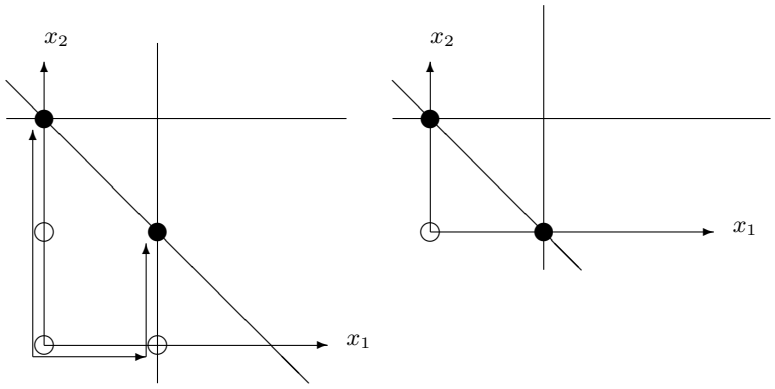


Fig. 2. Reducing the problem to a smaller one.

In this case, the minimal application vector is $(0, 1))$ and the problem $(a = (1, 2, 2, 1))$ is reduced to a smaller problem $(a = (1, 1, 1, 1)$, see Figure 2, right.

A problem with this idea is the fact that the minimum may be not reached by any total week priority.

Example 4. Let us consider the grammar

$$G = (\{a, b, c, d\}, R, a^2b^2c^2d^3)),$$
$$R = \{p : abd \to \lambda,\ q : acd \to \lambda,\ r : bcd \to \lambda,\ s : d \to d\}.$$

Any priority with $r_1 \succ \{r_2, r_3, s\}$ ($\{r_1, r_2, r_3\} = \{p, q, r\}$) will result in applying r_1 twice and s once. Any priority with $s \succ \{p, q, r\}$ will result in applying s 3 times, Thus, the minimum number of applications of s over all total priorities is 1. However, one can apply $p^1q^1r^1$ and then no s is applicable.

Improvement 3: Divide and Conquer – *Independent* Symbols. The steps to follow are the next ones:

- Select *applicable* rules
  (having enough objects, promoted and not inhibited).
- Consider the *dependency graph*, whose nodes are the symbols, with two nodes connected if the corresponding symbols are in the *lhs* of the same applicable rule.
- Consider the *components* of connectedness
  (maximal connected subgraphs). They correspond to the *independent* parts of the problem.
- Solve each subproblem *separately*: the solution vector set is the *direct sum* of the vector sets of the subproblems.

5.3 Sample Look

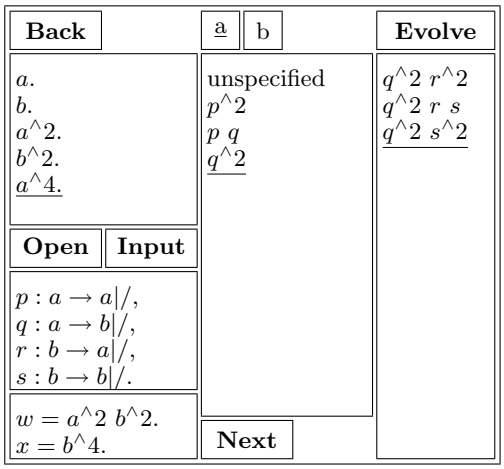


Fig. 3. The main interface.

Figure 3 shows how the program interface looks like. It displays the current configuration w and the next configuration x in the bottom-right corner. Above it,

the system rules are displayed. The button **Open** allows to load another system from a file. The button **Input** allows to manually enter the current configuration. Above these buttons, the history (list of previous configurations) is displayed, and the last item is now selected. The button **Back** allows to return to the selected configuration.

In the middle-top part one can see the tabs corresponding to the independent sub-systems, and the first one is now selected. In the center, the list of choices of evolution is shown, for the symbols corresponding to the selected tab. The button **Next** allows to go to the next tab. At the right, the list of the maximal multisets of rules applicable to w is shown. Notice that making a choice for some sub-system acts as a filter for that list. Once some multiset is selected in the list on the right, the simulator re-computes x. The button **Evolve** allows to add w to the history, make x the current configuration and re-compute the independent sub-problems and the maximal multisets of applicable rules.

6 Concluding Remarks

This paper describes the ideas behind the construction of a simulator for non-deterministic parallel rewriting systems, namely, for maximally parallel multiset-rewriting systems with context.

Section 3 describes the applications of such a simulator and relates the systems it simulates to other parallel rewriting systems, such as Petri nets, transitional P systems, communicative P systems, P systems with active membranes, Lindenmayer systems, grammar systems, regulated grammars.

Various optimizations of the algorithm of computing the set of possible transitions has been done before, for certain parallel rewriting systems with restricted forms of cooperation. It remains an open question how to compute the set of maximal multisets of rules applicable for a given configuration in a more efficient way in the general case. An interesting question would be to study the problem in case of general cooperation of a small degree. A future work would be to implement simulators of different parallel rewriting systems as interfaces for this engine, and to use them in research.

Acknowledgements

The author is supported by the project TIC2002-04220-C03-02 of the Research Group on Mathematical Linguistics, Tarragona. The author acknowledges the Moldovan Research and Development Association (MRDA) and the U.S. Civilian Research and Development Foundation (CRDF), Award No. MM2-3034 for providing a challenging and fruitful framework for cooperation.

References

1. A. Alhazov, M. Margenstern, V. Rogozhin, Yu. Rogozhin, S. Verlan: Communicative P systems with minimal cooperation. In *Membrane Computing. International Workshop WMC5, Milan, Italy, 2004. Revised Papers* (G. Mauri, Gh. Păun, M.J. Pŕez–Jiménez, G. Rozenberg, A. Salomaa, eds.), LNCS 3365, Springer-Verlag, Berlin, 2005, 162–178.
2. A. Alhazov, C. Martín-Vide, Gh. Păun, eds.: *Pre-proceedings of the Workshop on Membrane Computing*, Tarragona, 2003.
3. A. Alhazov, D. Sburlan: (Ultimately confluent) Parallel multiset-rewriting systems with permitting context. In *Pre-proceedings of the Fifth Workshop on Membrane Computing*, Milano, 2004, 95–103.
4. I.I. Ardelean, M. Cavaliere: Modelling biological processes by using a probabilistic P system software. *Natural Computing* 2, 2 (2003), 173–197.
5. F. Arroyo, A.V. Baranda, J. Castellanos, C. Luengo, L.F. Mingo: A recursive algorithm for describing evolution in transition P systems. In *Pre-Proceedings of Workshop on Membrane Computing*, Curtea de Argeş, 2001, GRLMC Report 17/01, Rovira i Virgili University, Tarragona, 2001, 19–30.
6. F. Arroyo, C. Luengo, A.V. Baranda, L.F. de Mingo: A software simulation of transition P systems in Haskell. In *Membrane Computing. International Workshop WMC02, Curtea de Arges 2002, Revised Papers* (Gh. Păun, G. Rozenberg, A. Salomaa, C. Zandron, eds.), LNCS 2597, Springer-Verlag, Berlin, 2003, 19–32.
7. D. Balbotin–Noval, M.J. Pérez–Jiménez, F. Sancho–Caparrini: A MzScheme implementation of transition P systems. In *Membrane Computing. International Workshop WMC02, Curtea de Arges, Revised Papers* (Gh. Păun, G. Rozenberg, A. Salomaa, C. Zandron, eds.), LNCS 2597, Springer-Verlag, Berlin, 2003, 58–73.
8. A.V. Baranda, J. Castellanos, F. Arroyo, R. Gonzalo: Towards an electronic implementation of membrane computing: A formal description of nondeterministic evolution in transition P systems. In *DNA7 Proceedings* (N. Jonoska, N.C. Seeman, eds.), Tampa, 2001, 273–282.
9. A. Binder, R. Freund, G. Lojka, M. Oswald: Implementation of catalytic P systems. *CIAA 2004*, Kingston, 2004, 24–33.
10. P. Bottoni, C. Martín–Vide, Gh. Păun, G. Rozenberg: Membrane systems with promoters/inhibitors. *Acta Informatica*, 38, 10 (2002), 695–720.
11. G. Ciobanu, D. Paraschiv: Membrane software. A P system simulator. *Fundamenta Informaticae* 49, 1-3 (2002), 61–66.
12. G. Ciobanu, G. Wenyuan: A parallel implementation of transition P systems. In [2], 169–184.
13. A. Cordón–Franco, M.A. Gutiérrez–Naranjo, M.J. Pérez–Jiménez, F. Sancho-Caparrini: A Prolog simulator for deterministic P systems with active membranes. *Brainstorming Week on Membrane Computing* (M. Cavaliere, C. Martín–Vide, Gh. Păun, eds.), GRLMC Report 26/03, Tarragona, 2003, 141–154.
14. J. Dassow, Gh. Păun: *Regulated Rewriting in Formal Language Theory*. Springer-Verlag, Berlin, 1989.
15. M. Margenstern, V. Rogozhin, Y. Rogozhin, S. Verlan: About P systems with minimal symport/antiport rules and four membranes. *Pre-proceedings of the Fifth Workshop on Membrane Computing*, Milano, 2004, 283–294.
16. I.A. Nepomuceno–Chamorro: A Java simulator for basic transition P systems. *Journal of Universal Computer Science*, 10, 5 (2004), 620–619.

17. Gh. Păun: *Membrane Computing. An Introduction*. Springer-Verlag, Berlin, 2002.
18. A. Salomaa, G. Rozenberg, eds.: *Handbook of Formal Languages*. Springer-Verlag, Berlin, 1997.
19. A. Syropoulos, E.G. Mamatas, P.C. Allilomes, K.T. Sotiriades: A distributed simulation of P systems. In [2], 455–460.
20. The P Systems Web Page: `http://psystems.disco.unimib.it/`.