Computer Science Journal of Moldova, vol.**18**, no.2(53), 2010

Simulator of P-Systems with String Replication Developed in Framework of P-Lingua 2.1*

Veaceslav Macari, Galina Magariu, Tatiana Verlan

Abstract

In this paper we present beta version of simulator for P-systems with string replication rules. This simulator is developed according to P-Lingua ideology and principles of the P-Lingua 2.1 development environment. Format for presentation of rules with replications in P-Lingua language is proposed. The already known solutions by means of P-systems with string replication for two problems are used to demonstrate the work with the simulator: the SAT problem and inflections generation problem.

Keywords: P-system, string object, string replication rule, simulator, P-Lingua

1 Introduction

Membrane computing is a rapidly developing scientific trend. The scientists, working in the domain and those who applies results obtained in the domain for practical problems solution, need some tools for verification, demonstration and proof of their theoretical ideas and results. Taking into consideration fast development of the domain and constant appearance of new types of P-systems, the idea to create the development environment, which can be extended as new types of P-systems appear, seems to be successful [1]. Simulators developed in the framework of P-Lingua 2.1 support several types of P-systems: transition P-system model, symport/antiport P-system model, active

*The authors acknowledge the support of the Science and Technology Center in Ukraine, project 4032 "Power and efficiency of natural computing: neural-like P (membrane) systems".

membranes P-system model (with membrane division rules and with membrane creation rules), probabilistic P-system model, stochastic P-system model. As new models have been included, new simulators have been developed inside the pLinguaCore library, providing at least one simulator for each supported model [2]. P-systems working with string objects using replication rules are not covered by the P-Lingua 2.1 soft, but considered to be powerful means for solution of some problems in different domains (e.g. SAT problem, Hamiltonian Path Problem, some linguistic problems, etc. [3], [4], [5]). So, in this paper we present beta version of simulator for P-systems with string replication rules. This simulator is developed according to P-Lingua ideology and principles of the P-Lingua 2.1 development environment.

2 Model of P-System with String Replication Rules

The formal definition of membrane P-systems with replications is given in [6]. A P-system with string objects and input is a construct

$$\Pi = (O, \Sigma, \mu, M_{l_1}, \ldots, M_{l_p}, R_{l_1}, \ldots, R_{l_p}, i_0), \tag{1}$$

where
O – a finite alphabet;
Σ – a sub-alphabet, $\Sigma \subseteq O$;
μ – a membrane structure defined as a rooted tree with nodes labeled $1, \ldots, p$, the interior of each membrane defines a region;
M_{l_i} – multiset of strings, initially present in region l_i, $1 \leq i \leq p$;
R_{l_i} – set of rules of region l_i, $1 \leq i \leq p$;
l_{i_0} – label of input region, $1 \leq i_0 \leq p$.

A replication rule has the following structure:

$$a \rightarrow (u_1, t_1)||(u_2, t_2)|| \ldots ||(u_k, t_n), \tag{2}$$

where
$a \in O^+$,

$u_j \in O^*$, $1 \leq j \leq n$,
$t_j \in \{out, here\} \cup \{in_{l_i} \mid 1 \leq i \leq p\}$, $1 \leq j \leq n$.

It is the string rewriting rule with string replication and target indication.

Application of a rule $a \to (u_1, t_1)||(u_2, t_2)|| \ldots ||(u_k, t_n)$ from the set of rules R_{l_k} transforms any string of the form $w_1 a w_2$ from region l_k into n strings $w_1 u_1 w_2, w_1 u_2 w_2, \ldots, w_1 u_n w_2$, where $w_1, w_2 \in O^*$. The resulting string $w_1 u_j w_2$ should be sent to the region specified by t_j:

- if $t_j = here$, the resulting string remains in the region l_k;
- if $t_j = out$, the resulting string is sent out of the region associated with membrane with label l_k to the region immediately outside;
- if $t_j = in_{l_i}$, the resulting string is sent into the region associated with membrane with label l_i, which has to be immediately nested into the region l_k.

(If $k = 1$ we have the usual string rewriting rule with target indication.)

The initial configuration contains the input string(s) over Σ in region i_0 and the multisets of strings M_i in regions i. The rules of the system are applied in parallel to all strings in the system. It may occur that several rules are applicable to a string. But really only one of them can be applied. The rule which will be applied, is determined in non-deterministic way. The computation consists in non-deterministic application of the rules to the strings in regions. If some string can be involved into the computing process, it has to be involved.

The computation halts when no rules are applicable. The result of the computation is the set of all words sent out of the outermost region into environment.

3 P-Lingua Format for P-Systems with String Replication Rules

Since the P-Systems with String Replication had not been implemented within the scope of P-Lingua 2.1, the format for replication rules rep-

resentation have been developed by the authors. When developing the format, the authors tried to keep principles and notations accepted in the domain and in the framework of P-Lingua 2.1. The authors acknowledge A.Alhazov for fruitful discussions on the question. The developed format is described below.

3.1 Special Symbols

There is a set of symbols of keyboard and reserved words in P-Lingua 2.1, which are used for specific aim when a P-system is described in P-Lingua format [2] (e.g. @mu, def, $call$, #). We use the special symbols predefined in P-Lingua with the same purpose as it was designed by the developers of P-Lingua. But, for the P-system with string replication there was a need to introduce some additional symbols and additional functionality for the existing ones.

Thus, the following notations are admitted:

- symbol "$'$" is used now to present not only the label of membrane, but the target membrane as well;
- symbol "$\|$" is introduced as the sign for operation of replication;
- the new reserved word "*here*" shows, that the resulting string will remain in the current region;
- the new reserved word "*out*" shows, that the resulting string will be sent out of the current region into the immediately encompassing region.

3.2 Rules with Replication

P-systems with replications operate with strings. To represent a string one has to use a sequence of alphabet symbols which are tied by sign of concatenation and are bracketed in $<$ and $>$. For example, $< c.A\{1,2\}.Beta >$ is the string from three alphabet symbols c, $A_{1,2}$, $Beta$. To write the empty string, we use $< \# >$.

3.2.1 Membrane to Which a Rule Belongs and Target Membrane

To indicate membrane to which a rule belongs, we write symbol "`'` " and membrane label in the left part of the rule after the string. For example, if a rule belongs to membrane with label 1, we write:

```
<a.b.c>'1 -->
```

To indicate the target membrane, we use the same scheme for the right part of the rule if the target is in_{l_i}. When the target is *out* or *here*, we write (after the string) the symbol "`'` " and the respective reserved word *out* or *here*. For example, the following rule for membrane with label 1

$$a \to (abc, here)||(dd, in_2)||(f, out) \tag{3}$$

we present as

```
<a>'1 --> <a.b.c>'here  || <d.d>'2 || <f> 'out
```

3.2.2 Rules Presentation

According to general view of a replication rule $a \to (u_1, t_1)||(u_2, t_2)|| \ldots \ldots ||(u_k, t_k)$, it consists of a left part and a list of right parts with the symbol of replication "||" between them.

$$Left\ part \to Right\ part\ ||\ Right\ part\ ||\ \ldots\ ||\ Right\ part \tag{4}$$

Left part has the following format:

$$s'h, \tag{5}$$

where
s – a non-empty string, e.g. `<a.alpha.bb>`;
h – the label of the membrane to which the rule belongs.

Each Right part can have one of the following three formats:

1)

$$s'here, \tag{6}$$

where

– s is a string, possibly empty, e.g. `<a.beta.abd>` or $< \# >$;

– the reserved word "*here*" shows, that the resulting string will remain in the current region;

2)

$$s'out, \tag{7}$$

where

– s is a string, possibly empty, e.g. `<a.beta.abd>` or $< \# >$;

– the reserved word "*out*" shows, that the resulting string will be sent out of the current region into the immediately encompassing region;

3)

$$s'h, \tag{8}$$

where

– s is a string, possibly empty, e.g. `<a.beta.abd>` or $< \# >$;

– h is the label of target membrane, which specifies the region into which the resulting string will be placed and which has to be immediately nested into the current membrane.

Examples of rules:

1. Suppose that we have the following rule for membrane 1:

$$\beta \cdot e \rightarrow (\phi_1, here) \mid\mid (\phi_2, in_3) \mid\mid (\lambda, out) \tag{9}$$

In Plingua format it looks like:

```
<beta.e>'1 --> <fi{1}>'here || <fi{2}>'3 || <#>'out;
```

2. Suppose that we have the following rule for membrane 2:

$$d \rightarrow (\mu \cdot k, out) \tag{10}$$

In Plingua format it looks like:

```
<d>'2 --> <mu.k>'out;
```

4 Examples of Solution Implementation to Some Problems

4.1 A Solution to SAT

Satisfiability problem (SAT) definition is the following: Given a Boolean expression E in conjunctive normal form (CNF), to decide if there is some assignment to the variables in E such that E is true.

Let us consider a solution to the SAT problem, using P-systems with string replication, given in [4].

Suppose we are given a fomula $E = C_1 \wedge C_2 \wedge \ldots \wedge C_m$ where C_1, C_2, ..., C_m are disjunctions, and the variables involved are x_1, x_2, ..., x_n. The following P-system with string replication is proposed for the problem solution:

$$\Pi = (V, V, \mu, M_1, M_2, \ldots, M_{m+1}, R_1, R_2, \ldots, R_{m+1}), \qquad (11)$$

where
$V = \{a_i, t_i, f_i \mid 1 \leq i \leq n\}$,
$\mu = [_1\ [_2 \ldots\ [_{m+1}\ ]_{m+1}\ ]_2\ ]_1$;
$M_{m+1} = \{a_1\}$, $M_i = \{\lambda\}$, $1 \leq i \leq m$,
$R_{m+1} = \{a_i \rightarrow (t_i a_{i+1}, here) \mid\mid (f_i a_{i+1}, here) \mid 1 \leq i \leq n-1\}$
$\cup\{a_n \rightarrow (t_n, out) \mid\mid (f_n, out)\}$,
$R_j = \{t_i \rightarrow (t_i, out) \mid x_i$ is present in $C_j, 1 \leq i \leq n\}$
$\cup\{f_i \rightarrow (f_i, out) \mid \neg x_i$ is present in $C_j, 1 \leq i \leq n\}$, $1 \leq j \leq m$.

For demonstration simplicity we take the case, when the values for m and n are not so large: $m = 3$, $n = 4$, $E = (x_1 + \neg x_2)(\neg x_2 + x_3 + \neg x_4)\neg x_3$.

Code of the program for this problem solution written in P-Lingua (file with extension *.pli*) is the following:

```
@model<string_replication>

def SAT() {

@mu = [ [ [ [ ]'4]'3]'2]'1;
```

```
@ms(4) = <a{1}>;
/* rules for membrane 1 */
<t{1}>'1 --> <t{1}>'out;
<f{2}>'1 --> <f{2}>'out;
/* rules for membrane 2 */
<t{3}>'2 --> <t{3}>'out;
<f{2}>'2 --> <f{2}>'out;
<f{4}>'2 --> <f{4}>'out;
/* rules for membrane 3 */
<f{3}>'3 --> <f{3}>'out;
/* rules for membrane 4 */
<a{1}>'4 --> <t{1}.a{2}>'here || <f{1}.a{2}>'here;
<a{2}>'4 --> <t{2}.a{3}>'here || <f{2}.a{3}>'here;
<a{3}>'4 --> <t{3}.a{4}>'here || <f{3}.a{4}>'here;
<a{4}>'4 --> <t{4}>'out || <f{4}>'out;
}

def main() {
call SAT();
}
```

The solution is got in 7 steps ($m + n$ steps).

The initial configuration is shown in Annex 1, Fig. 1. There is the only string a_1 in the region associated with label 4. During the first 4 steps all possible sets of values for variables x_1, x_2, x_3, x_4 are generated in the region associated with label 4.

Step 1. (See Annex 1, Fig. 2.) Due to replication two strings are generated in the region associated with label 4 (for different values of the variable x_1: t_1 – for the value *true* and f_1 – for the value *false*): strings $< t_1.a_2 >$, $< f_1.a_2 >$.

Step 2. (See Annex 1, Fig.3.) Four strings are generated in the region associated with label 4 for different values of the variables x_1 and x_2: $< t_1.t_2.a_3 >$, $< f_1.t_2.a_3 >$, $< t_1.f_2.a_3 >$, $< f_1.f_2.a_3 >$.

Step 3. (See Annex 1, Fig. 4.) Eight strings are generated in the region associated with label 5 for different values of the variables x_1,

x_2 and x_3.

Step 4. (See Annex 1, Fig. 5.) 16 strings are generated for different values of all 4 variables x_1, x_2, x_3, x_4 and they are sent out of the region associated with label 4 into the region associated with label 3.

Step 5. (See Annex 1, Fig. 6.) During the steps 5 – 7 the sets of variables values are filtered: only those sets leave the region, for which the given expression gets the value *true*. At the step 5 the sets are filtered by possible values of the third disjunction C_3 which in our case is negation of the variable x_3: only sets with value *false* for the variable x_3 leave the region associated with label 3.

Step 6. (See Annex 1, Fig. 7.) At the step 6 the sets from the region associated with label 2 are filtered by possible values of the second disjunction C_2 which in our case is $\neg x_2 + x_3 + \neg x_4$: only sets, for which this disjunction is equal to *true*, are sent out of the region associated with label 2.

Step 7. (See Annex 1, Fig. 8.) At the step 7 the sets from the region associated with label 1 are filtered by possible values of the first disjunction C_1 which in our case is $x_1 + \neg x_2$: only the sets, for which this disjunction is equal to *true*, are sent out of the region associated with label 1. And after that the computation halts: the resulting sets, for which the formula $E = (x_1 + \neg x_2)(\neg x_2 + x_3 + \neg x_4)\neg x_3$ gets the value *true*, are in the environment.

4.2 A Solution to the Problem of Inflections Generation in Romanian Language

Inflection in natural language means a change in the form of a word, usually modification or affixation, signalling change in such grammatical functions as tense, voice, mood, person, gender, number, or case. The inflection process goes on according to some set of rules (different rules for different groups of words). The number of such rules varies for different natural languages. The rules can be algorithmized, so the process of inflections generation gets computer aiding.

When modeling the process of inflections generation by P-systems, there appears the possibility to construct in parallel way all the inflec-

tions not only for one word, but all the inflections for some group of words which have specific common characteristics, i.e. belong to one inflectional model (e.g. neuter noun, in Romanian, inflectional model 3).

In the article [5] there is defined the P-system with string replication performing the inflection process (including vowel/consonant alternation with the assumption that alternating subword is present in the input word in just one occurrence). According to this definition we construct the P-system for the words of one inflectional model for nouns in Romanian – masculine, inflectional model 5. Below we demonstrate the work of the P-system on the example of two nouns – *"brad"* (engl. *"firtree"*) and *"caid"* (engl. *"kaid"*), which belong to this inflectional model. For better understanding of P-system rules, let us consider the list of inflected words for the noun *"brad"*. Taking into account that the Romanian forms for nominative and accusative cases coincide, as well as for the genitive and dative ones, we consider the reduced paradigm:

brad, brad, brad, bradul, bradului, bradule,
brazi, brazi, brazi, brazii, brazilor, brazilor.

Since one part of inflections is formed without alternation and another part – with alternation, we have two sublists of endings for the nouns of inflectional model 5:

$F_1 = \{, , , ul, ului, ule\}$ and $F_2 = \{i, i, i, ii, ilor, ilor\}$.

So in this case we have the number of sublists of endings $s = 2$; the number of alternations is $m - 1 = 1$, then we have $m = 2$.

Now we can construct the P-system which models the inflection process for considered inflection group according to the definition given in [5].

The words *"brad"* and *"caid"* (followed by the symbol "#") we place into the input region. The P-system looks like the following:

$$\Pi = (O, \Sigma, \mu, brad\#, \lambda, \lambda, R_1, R_2, R_3, 1), \tag{12}$$

where
$\Sigma = V \cup \{\#\}$,

$O = \Sigma \cup E$,
$\mu = [\ [\ ]_2\ [\ ]_3\ ]_1$,
$E = \{\#_2\} \cup \{A_{11}, A_{12}, A_{21}, A_{22}\}$,
$V = \{a, b, \ldots, z\}$,
$R_1 = \{\# \to A_{12} \mid\mid (\#_2, in_2)\} \cup \{A_{21} \to (\lambda, in_3)\} \cup$
$\{A_{12} \to (\lambda, out)||(\lambda, out)||(\lambda, out)||(ul, out)||(ului, out)||(ule, out)\} \cup$
$\{A_{22} \to (i, out)||(i, out)||(i, out)||(ii, out)||(ilor, out)||(ilor, out)\}$
$R_2 = \{d \to (zA_{21}, out\}$,
$R_3 = \{\#_2 \to (A_{22}, out)\}$.

The P-system has $1 + (s-1)m$ membranes, then there are 3 membranes in our case. Membrane with label 1 is intended for endings adding. The inflections corresponding to subset F_1 are generated in 2 steps. At the second step they are sent out to the environment. Additionally, in the P-system there are m membranes for each other subset (these membranes are intended for alternations implementation). In our case there are two membranes intended for implementation of single alternation: membranes with labels 2 and 3. The number of steps necessary for performing one alternation is equal to 2. When there are several alternations, they are implemented subsequently inside a word but in parallel for different words. So, the solution is got in $2m + 1$ steps regardless of the number of words placed in the input region; then there are 5 steps for our case.

Code of the program for this problem solution written in P-Lingua (file with extension *.pli*) is the following:

```
@model<string_replication>

def Inflection() {
@mu = [ []'2 []'3]'1;
@ms(1) = <b.r.a.d.diez>,<c.a.i.d.diez>;

 /* rules for membrane 1 */

/* <diez> -> <A{1,2}> || <diez{2}> in 2; */
      <diez>'1 --> <A{1,2}>'1 || <diez{2}>'2;
```

```
/* <A{2,1}> -> <#> in 3; */
      <A{2,1}>'1 --> <#>'3;
/* <A{1,2}> -> <#> out || <#> out || <#> out || <u.l> out
|| <u.l.u.i> out || <u.l.e> out; */
      <A{1,2}>'1 --> <#>'out || <#>'out || <#>'out ||
      <u.l>'out || <u.l.u.i>'out || <u.l.e>'out;
/* <A{2,2}> -> <i> out || <i> out || <i> out || <i.i> out
|| <i.l.o.r> out || <i.l.o.r> out; */
      <A{2,2}>'1 --> <i>'out || <i>'out || <i>'out ||
      <i.i>'out || <i.l.o.r>'out || <i.l.o.r>'out;

/* rule for membrane 2 */
/* <d> -> <z.A{2,1}> out;  */
      <d>'2 --> <z.A{2,1}>'out;

/* rule for membrane 3 */
/* <diez{2}> -> <A{2,2}> out;  */
      diez{2}>'3 --> <A{2,2}>'out;
}

def main() {
     call Inflection();

}
```

In Annex 2, Fig. 9 there is the initial configuration of the P-system as it looks in the simulator console.

Step 1. (See Annex 2, Fig. 10.) Since we have two sublists of endings, for each input word two strings are generated – two strings stay in the region 1 and two strings enter the region 2:

- strings in the region 1 are responsible for generation of inflections with endings from subset F_1, which are supposed to be formed without alternation;

- strings in the region 2 are responsible for generation of inflections with endings from subset F_2. One can see that the string is

marked by the special symbol $diez_2$. It has index equal to 2 that shows that the string corresponds to subset F_2 (the subset with index equal to 2).

Step 2. (see Annex 2, Fig. 11.) For strings from region 1 the replicative substitutions are performed and the generated inflections with endings from subset F_1 are sent out to the environment. For strings from region 2 the alternation is carried out (letter *"d"* is changed by letter *"z"*), the marked letter $A_{2,1}$ is added to show that the first alternation was carried out and the resulting strings go to region 1.

Step 3. (see Annex 2, Fig. 12.) The marked letter $A_{2,1}$ in the strings in region 1 is dropped and the resulting strings are placed in region 3.

Step 4. (see Annex 2, Fig. 13.). The symbol $diez_2$ in the strings from region 3 is replaced by the marked letter $A_{2,2}$ and the resulting strings are sent out to region 1. The second index of the marked letter is equal to m, it means that all alternations are carried out and now the endings have to be added.

Step 5. (see Annex 2, Fig. 14.) For strings from region 1 the replicative substitutions are performed and generated inflections with endings from subset F_2 are sent out to the environment. The computation halts: the resulting strings corresponding to all inflections of the input words are present in the environment.

Due to massive parallelism the process of all inflections generation for the words *"brad"* and *"caid"* was implemented in 5 steps. The user is able to place into the input region several words from this inflection group – the number of steps for all inflections generation will be the same.

5 Conclusion and Future Work

The simulator for P-systems with String replications is being developed according to the ideology and in the framework of pLinguaCore. So a set of new classes and methods were added to pLinguaCore in order to support string objects as they are used in P-systems with replication

rules. At the same time a set of classes and methods already existing in PLinguaCore were modified to fit string replication P-systems.

The main idea of P-Lingua extension was to use already implemented ideology and to make a separate program flow at the same time. Therefore, new P-system implementation uses base types of P-Lingua framework but the code is separated starting from parsing of P-Lingua program, XML generation, XML loading and simulation with intermediary results and final solution output.

For the implementation of P-Lingua strings replication model, changes in parsing of string objects and rules were made.

A new model definition – "string_replication" – was added in order to make the P-Lingua soft to start working with strings replication P-system.

As far as when applying the rules there is an active work with string objects in P-systems with replications, the internal representation of string objects was changed (compared with that as it is made in PLingua 2.1). String object is represented as a list of objects (not as a simple string), each of which containing: alphabet object name, alphabet object indexes, alphabet object multiplicity. Due to this the work with rules and membrane content during simulation is essentially simplified.

The following classes were added to realize rules with replication: StringsCellLikePsystem, ReplicationLeftHandRule, StringsReplicationRightHandRule, StringsReplicationCellLikeRule, ReadStringsReplicationRule (for reading replication rules from XML file).

The subsequent work supposes implementation of other types of rules for string objects, first of all, splicing operation. Moreover, the more sophisticated means are planned to be offered for user of the simulator: for visualization of P-system (for these models) current configuration and for control of evolving process. Undoubtedly that our experience in the work on strings replication P-systems simulator will serve the good base for elaboration of simulators for other types of P-systems.

Annex 1

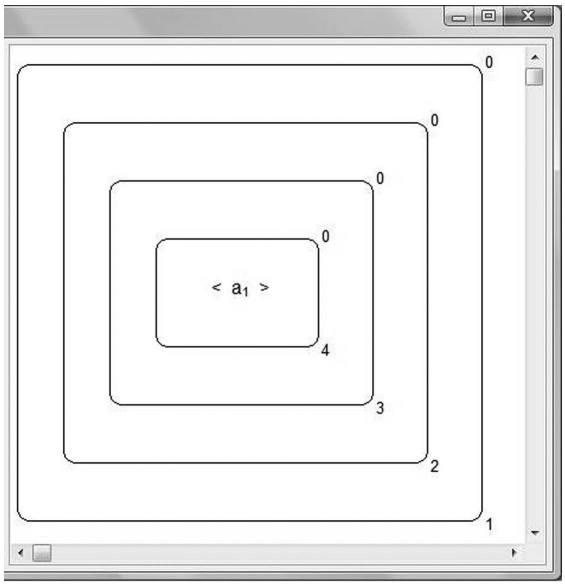

Figure 1. The initial configuration of the P-system for problem SAT solution

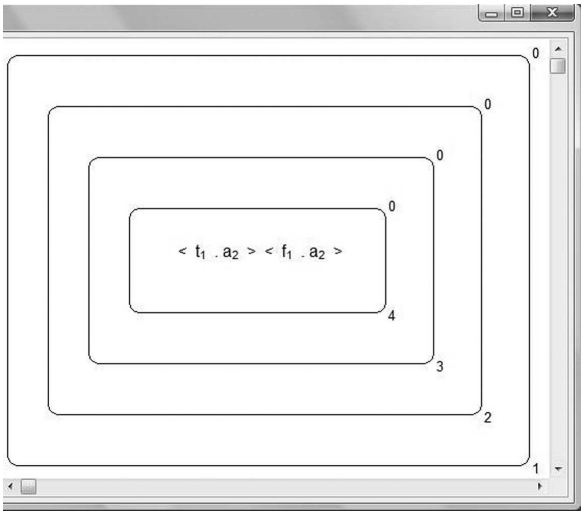

Figure 2. Configuration of the system after Step 1

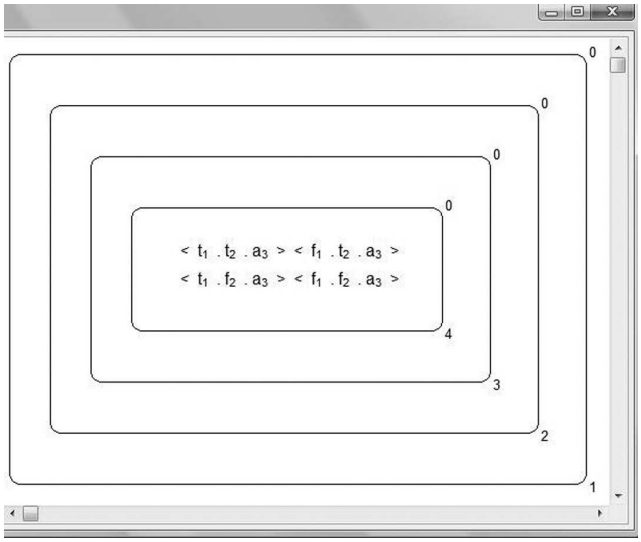

Figure 3. Configuration of the system after Step 2

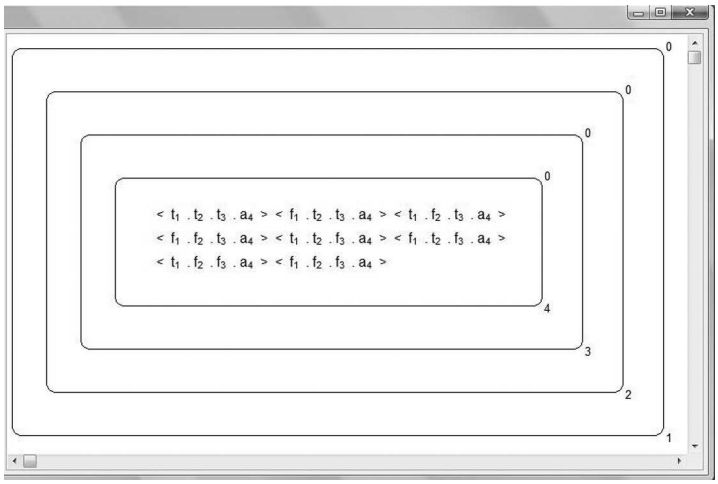

Figure 4. Configuration of the system after Step 3

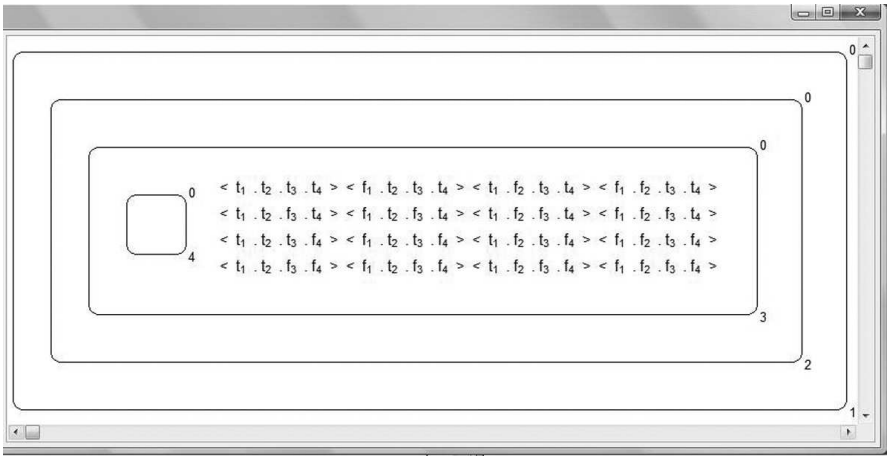

Figure 5. Configuration of the system after Step 4

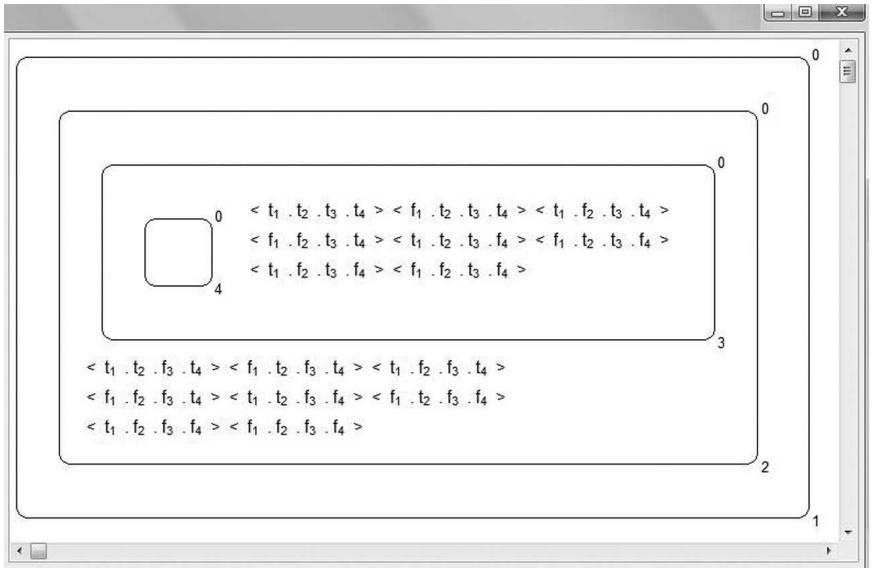

Figure 6. Configuration of the system after Step 5

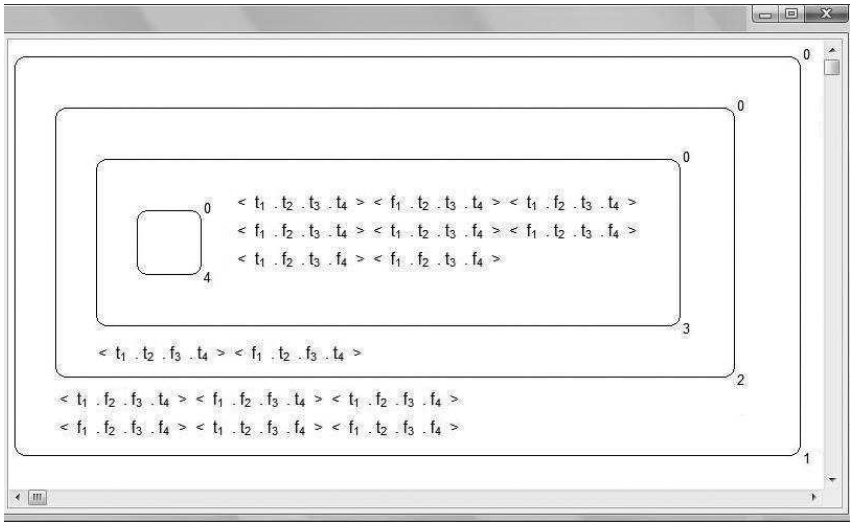

Figure 7. Configuration of the system after Step 6

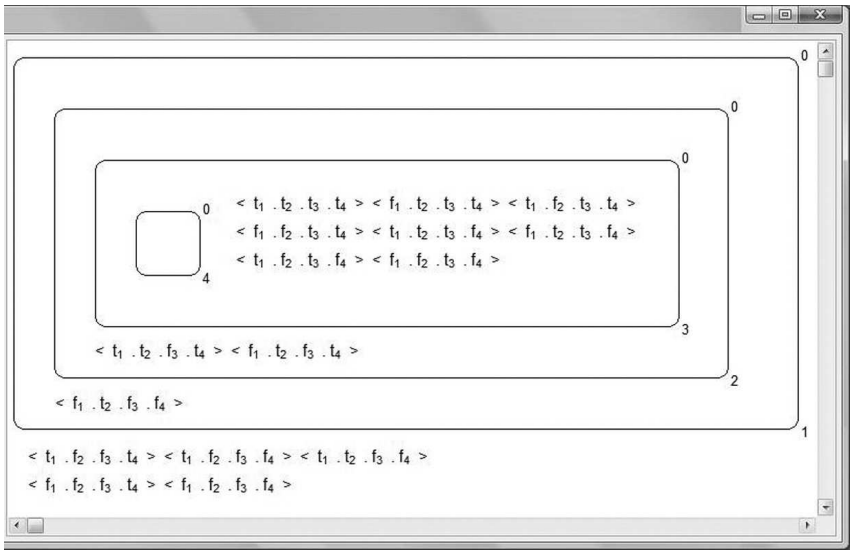

Figure 8. Configuration of the system after Step 7

Annex 2

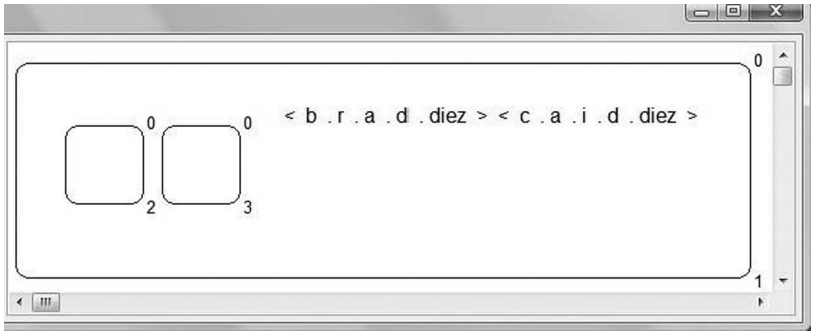

Figure 9. The initial configuration of the P system for inflections generation

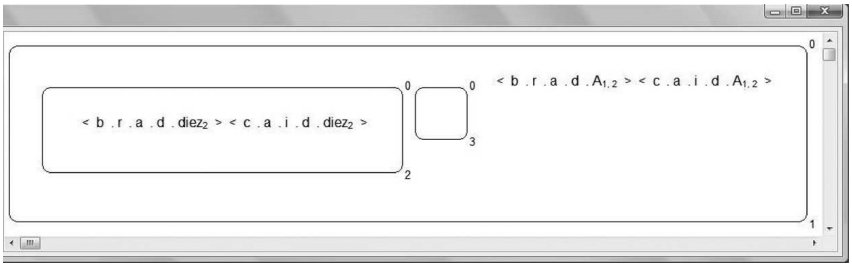

Figure 10. Configuration of the system after Step 1

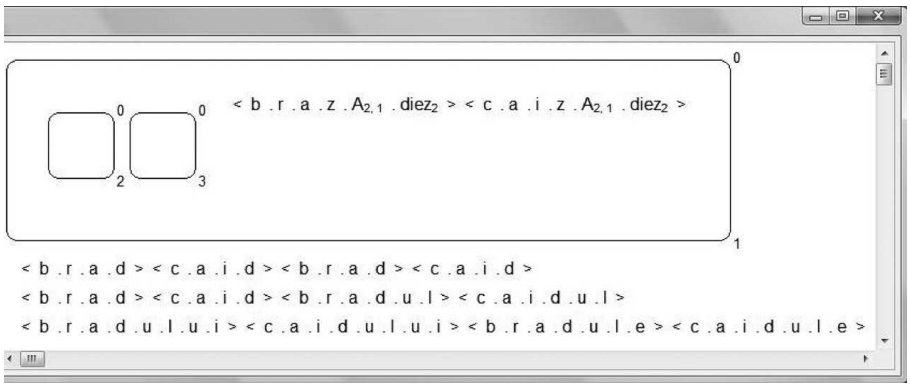

Figure 11. Configuration of the system after Step 2

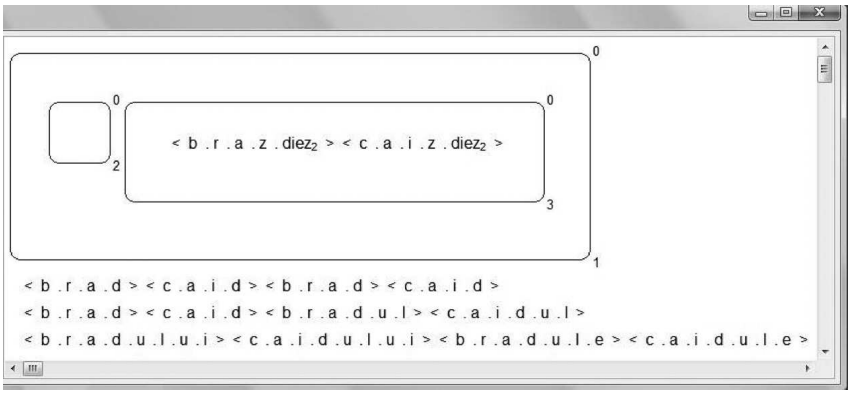

Figure 12. Configuration of the system after Step 3

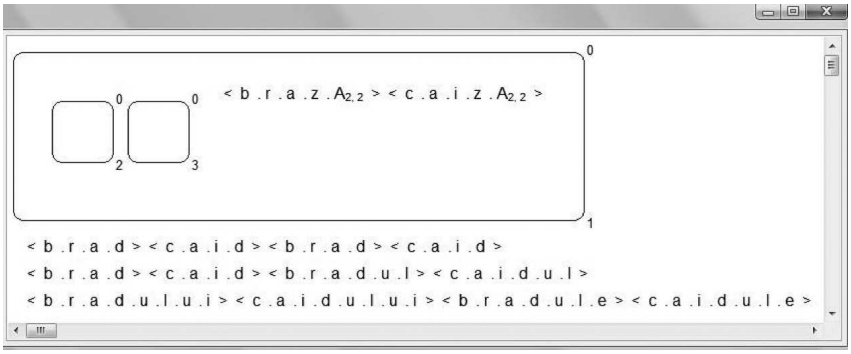

Figure 13. Configuration of the system after Step 4

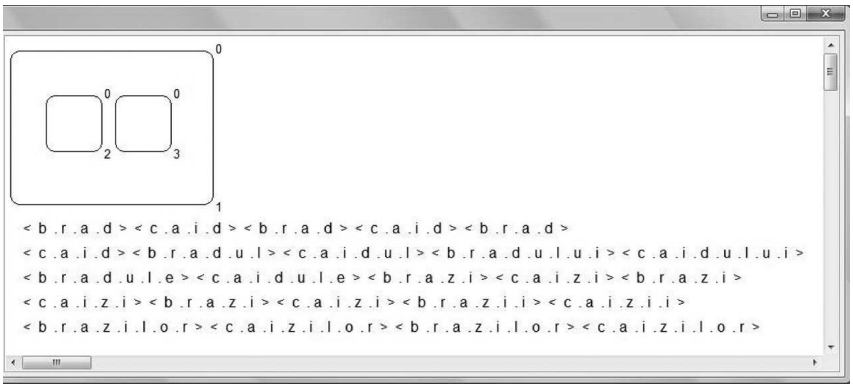

Figure 14. Configuration of the system after Step 5

References

[1] D. Díaz-Pernil, I. Pérez-Hurtado, M.J. Pérez-Jiménez, A. Riscos-Núñez. *A P-lingua programming environment for Membrane Computing*, Proceedings of the 9th Work-shop on Membrane Computing, pp. 155–172 (2008)

[2] M. García-Quismondo, R. Gutiérrez-Escudero, I. Pérez-Hurtado, M.J. Pérez-Jiménez, A. Riscos-Núñez. *An Overview of P-Lingua 2.0*, Tenth Workshop on Membrane Computing (WMC10), Curtea de Arges, Romania, August 24-27, pp. 240–264 (2009)

[3] V. Manca, C. Martín-Vide, Gh. Păun. *On the Power of P Systems with Replicated Rewriting*. Journal of Automata, Languages, and Combinatorics, 6, 3, pp. 359–374 (2001)

[4] Sh.N. Krishna, R. Rama. *P Systems with Replicated Rewriting*. Journal of Automata, Languages and Combinatorics 6(3): pp. 345–350 (2001).

[5] A. Alhazov, E. Boian, S. Cojocaru, Y. Rogozhin. *Modelling Inflections in Romanian Language by P Systems with String Replication*. Computer Science Journal of Moldova, V.17, N.2(50), pp. 160–178 (2009)

[6] Gh. Păun. *Membrane Computing: an Introduction*. Springer (2002).

[7] D. Díaz-Pernil, I. Pérez-Hurtado, M.J. Pérez-Jiménez, A. Riscos-Núñez. *P-Lingua: A Programming Language for Membrane Computing*. In D. Díaz, C. Graciani, M.A. Gutiérrez, Gh. Păun, I. Pérez-Hurtado, A. Riscos (eds.) Proceedings of the Sixth Brainstorming Week on Membrane Computing, Report RGNC 01/08, Fénix Editora, pp. 135–156 (2008)

V. Macari, G. Magariu, T. Verlan, Received August 22, 2010

Institute of Mathematics and Computer Science,
Academy of Sciences of Moldova,
Academiei 5, Chişinău MD-2028 Moldova
E–mail: *vmacari@gmail.com*, *gmagariu@math.md*, *tverlan@math.md*