P Systems and Finite Automata

Xian Xu
BASICS
Department of Computer Science and Technology
Shanghai Jiao Tong University
800 Dong Chuan Road (200240), Shanghai, China
Email: xuxian@sjtu.edu.cn

***Abstract*— In this paper, we integrate the traditional finite-state automata (words, or string based) into the membrane computing paradigm, as previous work prevalently concentrated on multiset based automata. We apply P systems with string objects (worms) to implement finite automata, that is, simulating their running, showing that P systems with string objects can properly hold the computability of finite automata. We give the concept of P system with string objects and finite automata, describe the implementation details, and finally make some future work expectation.**

I. Introduction

We implement (or simulate) the traditional finite(-state) automata [6][7], which receive (ordered) strings (finite symbol sequence, or words) as input and stop after reading the input with computing by transition relation, within the paradigm of membrane computing. That is, we use P systems with string objects (or worms) [11][12][13][15] to realize the running of finite automata, which, as far as we are concerned, is not an research area that has been heavily covered in membrane computing, though some work similar (but not the same essentially, they are multisets based automata) does exist [8][9][10]. On the other hand, albeit some results on universality in computational power have been investigated and proved [14][16][17], the direct and detailed construction of some machines of relatively smaller complexity has not been studied. We consider it also important to work out direct constructions of these specific machines, as these implementation (of for instance finite automata) offers some relatively simple and elementary but useful apparatus (gadget) in computation (also from practical viewpoint). And those things such as the relationship with the cited references will be properly explained in the main body of this paper, for example the difference of P systems we use from those in related work.

Motivation

In fact, the P automata introduced and analyzed in [8][9] and other related articles, based on multisets of objects, are not consistent with the traditional concept of finite automata, which are based on words. The difference lies in that P automata recognizes "strings" different from that recognized by traditional finite automata. By "string", we generally assume that there is an order among the symbols constituting the string, that is, the same symbol in different position of the string should be considered different. For example, in string $abad$ the two a's are different. When the order is neglected, the "string" then degenerates to a symbols bag, or multiset in P systems. Therefore, precisely speaking, P automata recognizes multisets of symbols from the alphabet associated with it, not "strings".

The difference described above renders P automata a special branch in automata theory. However, from either theoretic and practical point of view, string based automata are still an interesting area that deserves attention in membrane computing. P systems with string objects (or worms) make it possible to implement finite automata with P systems. In this variant of P systems, (real) strings are introduced and handled, in the form of *splitting, concatenation, replication* and etc.. By now, the study of P systems with string objects has been mainly on the computational power [14][16], few application of them has been reported.

We propose to combine the two aspects mentioned above, that is, to introduce string objects (worms), with their necessary operations, into the automata theory with respect to P systems. Thus, we can embroil traditional automata in P systems and examine the recognizing power of P systems on strings. We begin by implementing the most basic finite automata with finite words (strings). To be more concrete, in order to achieve this goal, we have to do some melding work. That is, we think it necessary to make use of the operations of two kinds of P systems, one with anti-port rules [3] and the other with string operation rules, to form an integral system that can simulate finite automata. In the rest of this paper, we try to do this.

II. Finite automata

In this section, we give a brief introduction to the concept of finite automata. Here we stick to the nondeterministic version of finite automata, as the deterministic finite automata are special cases of the former, and every nondeterministic finite automata can be converted to a deterministic one. So if we do not state explicitly, all the automata in discussion are nondeterministic.

A Nondeterministic Finite Automata (NFA) is the following structure

$$\mathcal{A} = (\Sigma, Q, s_0, F, \Delta),$$

where

- Σ. The alphabet.
- Q. The set of states.

- $s_0 \in Q$. The initial state.
- $F \subseteq Q$. The set of final states.
- $\Delta \subseteq Q \times \Sigma \times Q$. The transition relation consisting of transition rules of the form:

$$(q, a, p) \in \Delta,$$

which means that in state q, the automata can read a symbol a and then transit to state p.

A running of an automata on some input string can be described as follows. The automata reads the input string's symbol one by one, from the left to the right (you can imagine there is a reading head scanning the input string). On the current symbol and current state of the automata, one of the transition rules is applied to make the state of the automata transit to a new state, and meanwhile the reading head moves right for the next successive symbol in the string. A running process starts with the initial state and the reading head positioned on the leftmost symbol of the string, then continues step by step that is described above, and the whole computation halts when the string is exhausted, that is, the reading head moves beyond the rightmost symbol of the string.

Suppose the automata is $\mathcal{A}$ and the input string is $w \in \Sigma^*$. When one running of $\mathcal{A}$ on w leaves $\mathcal{A}$ in a state belonging to the set F of $\mathcal{A}$, we say that the automata accepts (or recognizes) w. Otherwise, if no running halts in one of the final state, we say that the automata $\mathcal{A}$ rejects w. Moreover, the language, under Σ, that is recognized by an automata $\mathcal{A}$ is defined as

$$L(\mathcal{A}) = \{w \in \Sigma^* \mid \mathcal{A} \; accepts \; w\}.$$

It is well known that the language class recognized by finite automata is the regular language class[7]. More explanation on the basic concept of finite automata can be found in [6].

Example of a finite automaton

We consider a simple example. Let the alphabet be $\Sigma = \{a, b\}$, suppose we want to design a finite automaton that recognizes strings that have an odd number of a's. We can define the automaton as follows.

$$\mathcal{A} = (\Sigma, Q, q_0, q_f, R),$$

where $\Sigma = \{a, b\}$, $Q = \{q_0, q_f\}$, q_0, q_f are the initial state and final state, respectively. R is defined as

$$\begin{array}{l|l} (q_0, a, q_f) & (q_f, a, q_0) \\ (q_0, b, q_0) & (q_f, b, q_f). \end{array}$$

Remark 1:
- One can easily check that the automaton defined above indeed recognizes strings having an odd number of a's, with no regard to the number of b's in them.
- In fact, the automaton is a deterministic one, despite that we define it in the nondeterministic style, because we consider nondeterministic finite automata in this paper. Meanwhile, it is well known that every nondeterministic finite automaton can be converted to a deterministic finite automaton, with some subset construction algorithm [6], so our discussion does not lose generality.

The automaton can be represented graphically as below. In

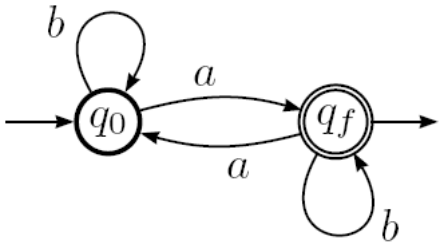

Fig. 1. The automaton recognizing strings having an odd number of a's.

Figure 1, each state is represented by a circle with the name of it in the circle. The initial state has an edge without source pointing to it, and the final state is doubly encircled and has an edge without destination leaving it. Each labeled directed edge corresponds to an element in the transition relation. For example, a labeled edge $q \xrightarrow{a} p$ indicates that there is an element (q, a, p) in the transition relation.

There are a wealth of examples in [6] and [7]. Here we just mention a typical one. For instance, from the example above we can construct finite automata recognizing strings having an even number of a's, an even number of a's and an odd number of b's, a's whose number is $3n(n = 1, 2, ...)$, and etc..

III. P SYSTEMS WITH STRING OBJECTS

Here we give the definition of the P system with string objects. We define it as follows. Note we use $\cdot$ to indicate string concatenation, and most time it is omitted.

Definition 2 (P system with string objects): A P system with string objects has the following structure.

$$\Pi = (V, \mu, A_1, ..., A_m, (R_1, S_1, M_1, C_1), ..., (R_m, S_m, M_m, C_m), i_0),$$

where

- The degree is m.
- V. The alphabet.
- μ. The (embedded) membrane structure. It is of a tree structure.
- $A_i (i = 1, ..., m)$. The initial multiset of string objects in region i.
- $Rule_i : (R_i, S_i, M_i, C_i)(i = 1, ..., m)$. Suppose

$$TAR \triangleq \{here, out\} \cup \{in_j \mid j \text{ is one of the embedded (son) membranes.}\},$$

and $tar, tar1, tar2 \in TAR$. The rules are explained as follows.

- R_i(replication): $r : (a \to u_1 \| u_2; tar1, tar2)$, $a \in V$ and $u_1, u_2 \in V^*$. For any strings $w_1, w_2, w_3 \in V^*$, we write $w_1 \Rightarrow_r (w_2, w_3)$ if $w_1 = x_1 a x_2, w_2 = x_1 u_1 x_2, w_3 = x_1 u_2 x_2$ for some $x_1, x_2 \in V^*$. And w_2, w_3 will be communicated to regions $tar1, tar2$, respectively.
- S_i(splitting): $r : (a \to u_1 | u_2; tar1, tar2)$, $a \in V$ and $u_1, u_2 \in V^*$. For any strings $w_1, w_2, w_3 \in V^*$, we write $w_1 \Rightarrow_r (w_2, w_3)$ if $w_1 = x_1 a x_2, w_2 = x_1 u_1, w_3 = u_2 x_2$ for some $x_1, x_2 \in V^*$. And w_2, w_3 will be communicated to regions $tar1, tar2$, respectively.
- M_i(mutation): $r : (a \to u; tar)$, $a \in V$ and $u \in V^*$. It is in fact a context-free rewriting rule. For any strings $w_1, w_2 \in V^*$, we write $w_1 \Rightarrow_r w_2$ if $w_1 =$

$x_1ax_2, w_2 = x_1ux_2$for some $x_1, x_2 \in V^*$. And w_2 will be communicated to region tar.

- C_i(crossing-over): $r : \ (z; tar1, tar2)$, $z \in V^+$. For any strings $w_1, w_2, w_3, w_4 \in V^*$, we write $(w_1, w_2) \Rightarrow_z (w_3, w_4)$ if $w_1 = x_1zx_2, w_2 = y_1zy_2, w_3 = x_1zy_2$, $w_4 = y_1zx_2$ for some $x_1, x_2, y_1, y_2 \in V^*$. And w_3, w_4 will be communicated to regions $tar1, tar2$, respectively.

Remark 3: Note that we make a little extension to the rules (especially the splitting rule) in comparison with the definition in [11]. That is, we allow empty string in some place of the rules (the u_i's and w_i's). For example, in the splitting rule, u_1 and u_2 belong to V^*, not V^+; so do w_1, w_2, w_3. This extension won't do any harm to the computational power of the system, since the extended rules are no less powerful than the original ones. What is more, they offer some flexibility in application, only on the condition that we should keep an eye on the obviously meaningless cases, which we think will not bring much complexity.

We describe the successful computation of the P system defined above as follows.

Definition 4 (Successful computation): P systems defined as in Definition 2 performs a successful computation if

It halts, that is there is no further rules applicable in any region of the system, with the output possibly in i_0, if defined.

Otherwise, the computation fails, that is the computation never halts and no output will be available.

More explanation on the characteristics of the P system defined above can be found in [11][12][13].

IV. Simulating finite automata

In this section, we give the detail of the simulation of finite automata using P systems with string objects. To achieve this, we need to make a little extension to the P system defined in Definition 2, that is, we need to introduce condition to the four rules in it. Now the rules on strings are of the following form:

$$\langle \text{rule of one of the four types; condition } \mathcal{F} \rangle,$$

where the condition can be a multiset of string objects, indicating that the rule can only be applied when objects in the condition $\mathcal{F}$ exist in the current membrane.

For example, the conditioned splitting rule: $r : \ (a \rightarrow u_1|u_2; tar1, tar2; \mathcal{F})$, $a \in V$ and $u_1, u_2 \in V^*, F \subseteq V^*$. It means that the rule can be applied only in presence of the string objects in $\mathcal{F}$.

Two steps comprise the simulation:

- Use splitting to read input
- Use conditioned rules to simulate transition rules of the finite automata.

Suppose the finite automata is $\mathcal{A} = (\Sigma, Q, q_0, F, \Delta)$. We define the following P system to simulate it.

$$\Pi_{\mathcal{A}} = (V, \mu, A_1, ..., A_m, (R_1, S_1, M_1, C_1), ..., (R_m, S_m, M_m, C_m), i_0),$$

where

- The degree is $m = 4$.
- $V = \Sigma \cup Q \cup \{a' \mid a \in \Sigma\}$. The alphabet.
- $\mu = [_1 \, [_2 \, [_3 \,]_3 \, [_4 \,]_4 \,]_2 \,]_1$. 1 is the place to hold the input string. 2 is the main computation place, where the transition rules are applied. 3 is the place to collect some intermediate objects. 0 is the environment. And 4 is the output region.
- $A_i (i = 1, ..., m)$. The initial multiset of string objects in region i. Here A_1 holds the input string. A_2 holds the initial state (q_0) of the automata. A_0, A_3, A_4 are all empty.
- $Rule_i : (R_i, S_i, M_i, C_i) (i = 1, ..., m)$. Here the rules are all conditioned rules.

We give the detail of the rules in each region below.

Here we explain the rules in each region. First we make some assumption.

- Rules are applied in a sequential manner on a single string, that is, only one rule can be applied to the string at a time. This is from the fact that usually an object takes part in one reaction at a time.
- We suppose that rules of all the types (R_i, S_i, M_i, C_i) function in the following way:

  If one rule finds more than one point in a string suitable for the application of it, it always strictly effects on the first it meets in the string (note strings are ordered from left to right)

  This is a natural premise, since usually a certain reaction will choose the first required object (or a point/site) it encounters in a sequence (string), as in most gene regulation cases. This is economic and effective from practical point of view.

Now suppose rule $(p, a, q) \in \Delta$ is the rule to be simulated. The rules are defined below.

- Rules in region 1. Rules here read in the leftmost unprocessed symbol (that under the reading head, or the current symbol) in the input string and send it to region 2 for further computation. We need a derived rule, Rd, which reads in the current symbol and removes it from the input string. It can be defined as follows

  $$Rd : \ \Sigma^* \rightarrow \Sigma \cup \{\lambda\} (\lambda \text{ indicates empty string}).$$

  In fact, it can be deemed as a kind of $macro$ of rules. Now

  $$(Rd \,;\, in_2)$$

  suffices to serve as the read-in step. Below we define the Rd rule. And we find it sufficient to use the splitting rule.

  $$Rd : \ a \rightarrow (a \mid \lambda \,;\, in_2, here), \quad a \in \Sigma,$$

  that is, if $w = x_1ax_2 (x_1, x_2 \in \Sigma^*)$, then

  $$w \Rightarrow_{Rd} (x_1a, x_2),$$

  with x_1a communicated to region 2 and x_2 remaining in region 1.

  The desired case is that $x_1 = \lambda$, so that the leftmost symbol of w is read out and sent to region 2 for further computation, which forms a fragment of the successful computation path. Note that when $x_1 \neq \lambda$, the computation will never halt, due to the rules in region 2, thus fail.
- Rules in region 2. $Rule_2$ can be divided into two parts.
  - Making the transition:

    $$\begin{array}{ll} (p \rightarrow q \,;\, here \,;\, a), & (p, a, q) \in \Delta \\ (a \rightarrow a' \,;\, in_3 \,;\, \emptyset), & a \in \Sigma \end{array}$$

  - Halting control:

    $$\begin{array}{ll} (p \rightarrow p \,;\, here \,;\, \emptyset), & p \in Q \\ (p_f \rightarrow p_f \,;\, in_4 \,;\, \emptyset) & \end{array}$$

    The two rules in halting control ensure that the transition are made exactly in terms of that of the automata being simulated, which will leave no state symbol here, otherwise, the first rule in the halting control here will be applied forever.

- Rules in region 3. $Rule_3 = \emptyset$.
- Rules in region 4. $Rule_4 = \emptyset$.

Remark 5:
- Only one state symbol appears in region 2 at a certain moment.
- The splitting rule (see Definition 2 for its definition) in region 1 ensures that the input string is stripped one symbol by another (from left to right) to the end, that is no symbol left in region 1 (all have been sent to region 2), when Rd cannot be applied any more and nothing happens in region 1 henceforth.
- After the symbol read in by Rd is processed (some transition rule has been applied), it is moved to region 3, which in a sense can be treated as a garbage collector. We do not send it out to the environment, because in region 3 some more manipulation is possible if needed, while it is not in the environment.
- The P system runs in a nondeterministic way. That is, wrong computational paths may exist, and the simulation succeeds whenever there is at least one computational path that successfully simulates the automata correctly. If we want to make the simulation deterministic, we have to extend the rules to make them probably more powerful.
- The input string is destroyed during the computation. This is not important from computational viewpoint but can be improved.

From the construction above, we can obtain the main result below.

Result

First we give the definition of the successful simulation of a finite automata.

Definition 6 (Successful simulation): A P system with string objects Π (successfully) simulates a finite automata $\mathcal{A}$, whose recognizing language is denoted by $L(\mathcal{A})$, if the following holds

$$\forall w \in \Sigma^*, \quad w \in L(\mathcal{A}) \quad \text{iff} \quad \Pi \text{ halts on input } w,$$

where how the input is made depends on the P system.

Then we can arrive at the main result of this paper. Note in the P system we constructed above, the input is put in region 1 initially.

Theorem 7: Given a finite automata $\mathcal{A}$, the P system $\Pi_{\mathcal{FA}}$ as defined in Section IV simulates it, in the way of Definition 6.

Actually, the result is obvious from the definition and the explanation above. The proof is straightforward. By now, we have finished our major task, the finite automata have been simulated using P systems with string objects.

V. Future work

Since the finite automata can be implemented by P systems with string-objects, we bravely think that other traditional automata can be implemented too, with the powerful operations on strings in such P systems, and in fact this computing approach comes probably from DNA computing, where different models of computation has been studied and formalized (such as [18]). Therefore, we think this work is somewhat meaningful and necessary, since they can serve as the basis of the computational power in the whole membrane computing paradigm. And the candidate is not difficult to think of, for example pushdown automata. Though a little ambitious, we still consider it a relatively practical goal, since there has been some work on the computational power of string-based P systems, and the universality has been obtained [16][17]. We hope that these work can be finished in the near future. But obstacles are not absent in this work. For example, to implement the pushdown automata we need to simulate a stack, which is a data structure that has not been realized before. So more details have to be settled.

VI. Conclusion

In conclusion, we have done several things. We first explained the motivation for our work, that is, the current state of the research in automata in P systems may need some extension to finite automata in that string (or words) should be considered. Second, we introduced the variant of P systems, the string-based P systems, which we make use of to implement finite automata. Then we give the detailed definition of the implementation of finite automata using P systems with string objects. Finally, we make some discussion on the topic we work on here. We think our work above contributes to the field of membrane computing in that it serves as a direct construction of traditional automata in membrane computing.

References

[1] Păun, Gh.. *Computing with membranes*. Turku Centre for Computer Science-TUCS Research Report No 208, 1998. Also in Journal of Computer and System Sciences, 61(1),108-143, 2000.

[2] Păun, Gh.. *Membrane Computing. An Introduction*. Springer, Berlin, 2002.

[3] A. Alhazov, R. Freund and Y. Rogozhin. *Computational power of symport/antiport: history, advances, and open problems*. In [4], 44-78.

[4] R. Freund, G. Lojka, M. Oswald, Gh. Păun (eds.): Pre-Proceedings of the 6th International Workshop on Membrane Computing (WMC6), Vienna Technological University, 2005.

[5] *P Systems Web Page*. http://psystems.disco.unimib.it.

[6] H. Lewis and C.H. Papadimitriou. *Elements of the Theory of Computation* (second edition). Prentice-Hall, September 1997.

[7] J. Hopcroft, R. Motwani and J. Ullman. *Introduction to Automata Theory, Languages, and Computation* (second edition). Addison-Wesley, 2001.

[8] E. Csuhaj-varju and G. Vaszil. *P Automata*. In Gh.Păun and C. Zandron(ed.), Pre-proceedings of the Second Workshop on Membrane Computing(WMC-CdeA2002), 2002.

[9] E. Csuhaj-Varju and G. Vaszil. *P automata or purely communicating accepting P systems*. In A. Salomaa, Gh.Păun, G. Rozenberg and C. Zandron(eds.), Membrane Computing, International Workshop WMC 2002, LNCS 2597,pp. 219-233, 2002. Springer-Verlag, Berlin, 2003.

[10] E. Csuhaj-Varjú. *P automata: Models Results and Research Topics*. In G. Mauri, Gh. Paun and C. Zandron (Eds.), Pre-proceedings of the Fifth Workshop on Membrane Computing, 2004.

[11] C. Martín-Vide and Gh. Păun. *String-Objects in P Systems*. From [5]. 2000.

[12] Gh. Păun, J. Castellanos and A. Rodriguez-Paton. *Computing with Membranes: P Systems with Worm-Objects*. Technical Report CDMTCS-123, Centre for Discrete Mathematics and Theoretical Computer Science, 2000.

[13] Gh. Păun, J. Castenallos and A. Rodriguez-Paton. *P systems with worm objects*. In IEEE 7th. International Conference on String Processing and Information Retrieval, SPIRE 2000, pages 64-74, La Coruna, Spain, 2000.

[14] S.N. Krishna and R. Rama. *P systems with replicated rewriting*. Journal of Automata, Languages Combinator, 6(3):345-350, 2001.

[15] S. Marcus.*Membrane versus DNA*. Proceedings of Membrane Computing (WMC-CdeA2001) (Gh. Păun et al. Eds.). Fundamenta Informaticae, 49(1-3):223-227, 2002.

[16] A. Rodriguez-Paton and J.L. Mate. *On the Power of P Systems with DNA-Worms*. Pre-proceedings of the Workshop on Membrane Computing, 2001.

[17] A.Păun. *P Systems with String-Objects: Universality Results*. Proceedings of Workshop on Membrane Computing (WMC-CdeA 2001),229-242, 2001.

[18] S. Roweis, E. Winfree, R. Burgoyne, N. V. Chelyapov, M. F. Goodman, P. W. K. Rothemund and L. M. Adleman. *A Sticker Based model for DNA Computation*. Proceedings of the Second Annual Meeting on DNA Based Computers, pages 1-27, 1996.