An Alternative P-System using Persistent Turing Machine

Mahmoud Abdelaziz, Amr Badr and Ibrahim Farag

Computer Science Department, Faculty of Computers and Information, Cairo University, Egypt.
mahmoud_m_aziz@yahoo.com; a.badr.fci@gmail.com; i.farag@fci-cu.edu.eg

Abstract: P system or membrane computation created by Paun [1998] is a branch of Natural computing which is a research filed that investigates both the computation designed by human being and computation taking place in nature. This model of computation is based on the processes done by the living cells and its computations are defined as applications of rules and transitions of objects. In this way, we have the same behavior of computational power as Turing Machine. This kind of systems computes by passing from a configuration to another by applying rules and the computation is considered complete when it halts. This paper investigates an alternative system to the P system model using the persistent Turing Machine (PTM). The proposed system likewise P-systems has two phases of computation application and communication phases. We introduce the design of the application phase and leave the communication phase for another research. Our design considers each membrane in the system membrane structure as a machine and the design of this machine will be explained using state transition graphs. Our design has been tested and simulated on a computer application. This design may open a new possibility, in the future to design a hardware that can simulate the process done by P-systems.
[Mahmoud Abdelaziz, Amr Badr and Ibrahim Farag. **An Alternative P-System using Persistent Turing Machine.** *Life Sci J* 2015;12(5):122-126]. (ISSN:1097-8135). http://www.lifesciencesite.com. 14

Keywords: Membrane computing, P System, Turing machine, Persistent Turing Machines.

1. Introduction

Natural computing has three major areas which are Neural Networks, Genetic Algorithms and Bio-molecular Computation. Membrane computing is a branch of bio-molecular computation and P System is a computational model within the field membrane computing. P system computes through two parallel phases of computations which are application phase and communication phase. Application phase is massively parallel in applying rules inside membranes where the communication phase is the communication among these membranes. For more comprehensive information about membrane computing see [1-3].

Persistent Turing Machine (PTM) is a nondeterministic 3-tape Turing machine (TM) with a read-only input tape, a read-write work tape, and a write-only output tape. Upon receiving an input taken from its environment on its input tape, a PTM computes for a while and then outputs the result to the environment on its output tape. For the models of TM and PTM [4-7].

2. Why Turing Machine

Both P system and Turing Machine are rewriting systems; they perform their computations through transforming their configuration from a configuration to another by applying rules (as in p system) or function (as in Turing machine). Also, P-systems are computability models of the same type as Turing machines or other classic representations of algorithms, and, as a consequence, they can be easily simulated on computers [8].

TM is a hypothetical device representing a machine designed to do a specific task so it has a good performance. Its model used in designing the computer and also used to test any computation that can be performed by any other computing device. One of the most important uses of the TM is that it helps us in understanding the limits of any mechanical computation.

3. The Proposed Alternative system

P systems have two phases of computation application and communication phase, the proposed alternative system has the same two phases where

- Application phase: a finite set of distributed PTMs, each PTM processes its own data by applying its own rules in localized manner. PTM can accept a stream of input from its environment and produces a stream of output to its environment while it computes (work), so this set of machines form a system of parallel distributed machines.
- Communication phase: is a PTM machine to transfer data between the set of application phase machines and synchronizes process among these set of machines which perform.

This paper explains the design of application phase especially the execution of the application rules and we shall leave the communication phase for another research.

Since PTM works in a sequential manner, the alternative system applies rules under a condition which is the rules applied in an ascending order according to the rules execution priorities. According to this view, the P Systems has the same relevance of

PTM, and each membrane in the P system can be considered as the simplest machine in the Transition P Systems.

In order to explain the design we suggest, the structure of this work is organized as follow; define the system structure of the p system and the alternative system, rules representation, rule selection and rule execution, stating the designed machine and its constrains, related work, results and future work, conclusion.

4. System Structure

The structure of the system is composed of a set of PTM machines, each machine considered as a membrane in the structure of p system. Each PTM machine has three tapes, the first for input data, the second for output and the third for work. All computation occurred on the work tape and the machine interacts with its environment through the input/output tapes. All symbols represented as 8 bits of 0s and 1s on the machine tapes, table (1) shows a part of symbols codes.

The Design of this machine is complicated; however its basic principles are as simple as standard Turing Machine principles. Therefore, it is better to breakdown the design into sub-machines and each sub-machine could have smaller sub-machines, each of which can be simpler than the whole machine design. The results of executing all these sub-machines are forming the full operations of the P System computation under the previous conditions. The main sub-machine for the design is the sub-machines which apply and execute the rules which we shall explain its design in this paper.

Since all computation is done on the machine work tape, so we partitioned the work tape into four parts separated by a specific 8 bits of code for each portion to make the machine specify which part it reads

5. Rules

Table.1 a part of code for symbols on the work tape

Symbol	Code
rule types symbols	
Active rule	00001000
Applicable rule	00010000
Not applicable rule	00011000
separators symbols	
Begin of work tape	**B**
Begin of rule zone	10000000
Begin of data zone	11100000
End of work tape	B
data object symbols	
A	00000001
B	00000010
C	00000011
D	00000100

The main core of computation process in the application phase of p system is the applying and executing rules. The computation has three types of rewriting rules application, communication and structure rules.

Each set of rules is different from the others in its behavior. The simplest one of them is the application rules because its application needs only deleting data that used to execute the rule from its work tape for this machine then writing the output data produced by its execution on the work tape.

5.1 Rule representation

Rules are represented on a part of work tape start with the code 10000000, each rule on this part has a rule number and rule state as shown in fig.(1) these rules are sorted on the tape according to its execution priority in a descending order where the rule has the high priority appears first. Each rule has a rule state where rules state is classified as Active, Applicable and Not applicable.

Active: the rule has the turn to executed and it has (00001000) code.
Applicable: the rule could be executed when it has the turn to be executed and it has (00010000) code.
Not applicable: the rule can't be executed and it has (00011000) code.

10000000 Rule state Rule no ---

Fig.1 Rule part on the work tape

5.2 Rule Applying

The rules are applied in two stages: the first stage is the "select" rule stage and the second stage is "test and apply" rule stage.

Select a rule stage: Machine starts reading work tape until it finds the beginning of rule part (the part of work tape has the code 10000000), the sub-machine which finds the start of rule part is called *Go-to-Rule-Part* sub-machine. Executing of *Go-to-Rule-Part* sub-machine means that the head of the work tape is on the first cell of rule part, at this situation the machine starts selecting an active or applicable rule to apply according to the rule priority and the rule state, the sub-machine which selects an active or applicable rule called *Get-Act-or-App-Rule*. After that the machine changes the rule state by using other sub machine called *Change-rule-state* which changes the rule state into active state and get the rule number. Changing the rule sate into active state notifies the machine with the last rule used and repeats execution of the rule until the rule doesn't found the data needed in this situation, the rule state changes into not applicable state. At this time the machine searches for the next applicable rule and so on until the machine doesn't find any applicable rule; at this time the

machine halts. The design of the sub machines *Go-to-Rule-Part*, *Get-Act-or-App-Rule* and *Change-Rule-State* are shown in figures (2) and(3).

Test and apply rule stage: after the machine selects a rule to apply it must test the execution of that rule. At the end of this stage there are two possibilities. The first, the rule executes its task where the design makes the machine executes the rule n times until the rule fails the execution test, which is the second possibility where the rule doesn't apply and fails in the execution test, the machine changes the rule state into not applicable state and goes to *select a rule stage* in an iterative manner until a halting configuration is reached.

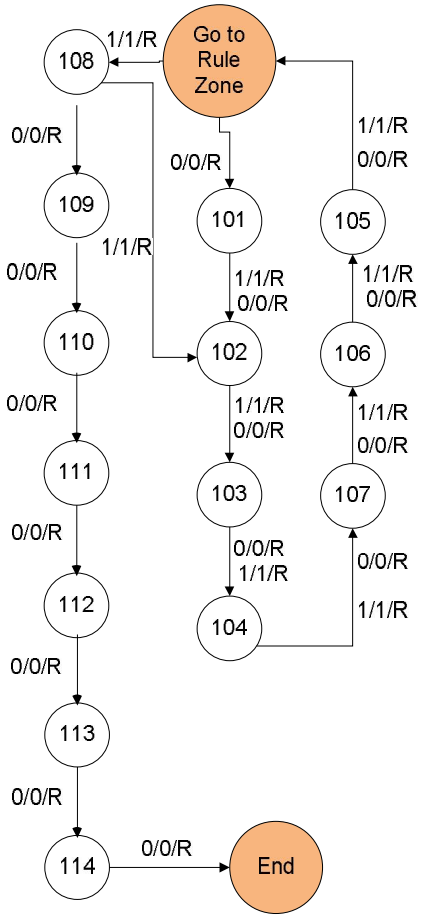

Fig.(2) a state transition graph for Go-to-Rule-Part submachine

The execution test depends on the rule type. If the rule type is application rule like a→b the execution test searches on the work tape data part for the data which the rule needs to execute its task, else, if the rule is a communication or a structure rule the execution test searches on the communication part of the work tape for the existence of the machine number which will receive the data produced when the rule executes. If the machine number is not found the execution test fails, else the execution test searches for the data which the rule needs to execute its task. An example for rule execution can be seen in Fig.(4) which tests and executes an application rule a→b.

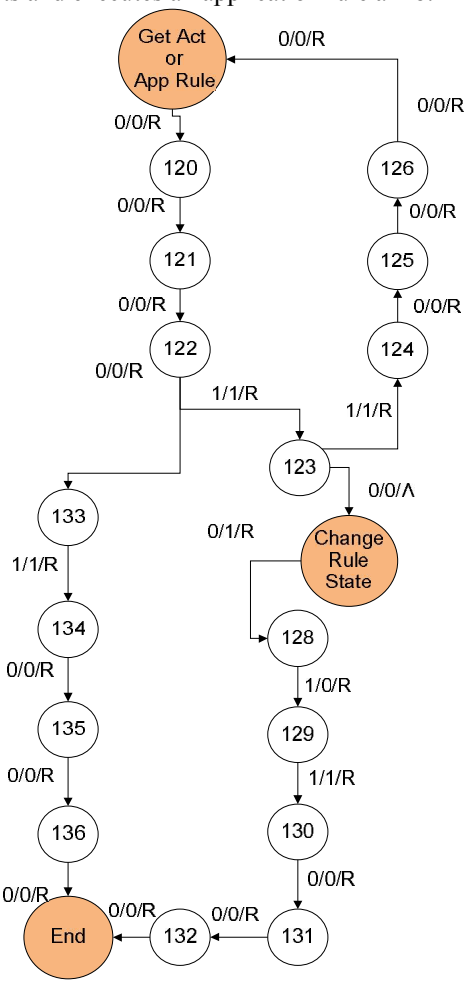

Fig.(3) a state transition graph for Get-Act-or-App-Rule submachine

6. Related Work

There is a few proposed related work like "A Reconfigurable Hardware Membrane System" [9] this paper presents a parallel hardware implementation of a special class of membrane systems. The implementation is based on a universal membrane hardware component that allows running membrane systems on specialized hardware.

The author explains that he has chosen the FPGAs to implement its hardware, since FPGAs are fully reprogrammable. The author uses sets of registers of 8-bit to store the instances of objects, rules, rules priority and membranes labels.

There are some drawbacks in this design that we can summarize as follows:

- Registers which used to store the instances of objects have a limit size, so they limit the number of objects instances that can be stored in, on the other hand, this drawback doesn't exist in the Turing Machine since its tape has unlimited size.
- Rule size limitation on the other hand, there is no limitation of the rules size since the rule embedded in the design of the machine.
- Limitation of the number of rules.

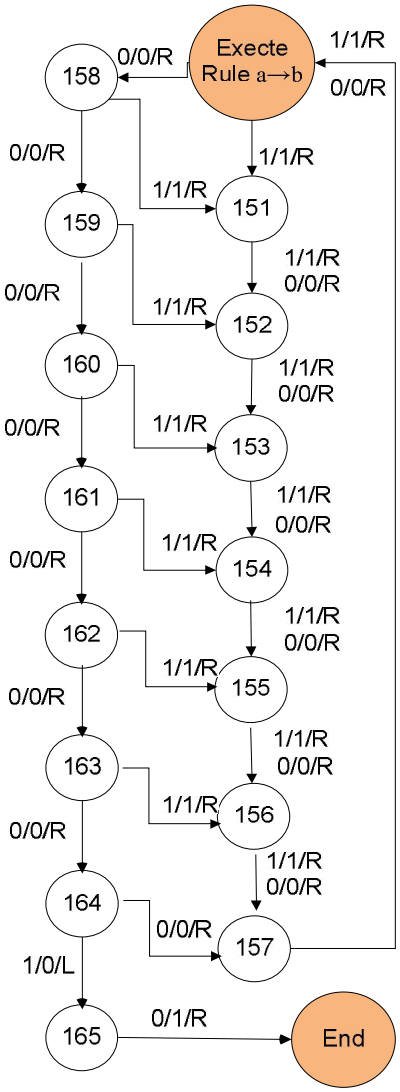

Fig.4 a state transition graph for a→b Rule

7. Results

The proposed design has been simulated and evaluated by developing an application using visual basic which was used to measure the average time for executing one rule. The program follows our design where each membrane in the system structure of membranes is a machine that has its own data and rules. So each machine has a set of sequential code to select and execute its rules. Each sequential code assigned to different thread, according to that concept all machines work in parallel. We supposed that there is a coordination set of code to coordinate the data passing between machines. Also, it follows the changes in the structure of the system which happened as a result of executing the structure rules.

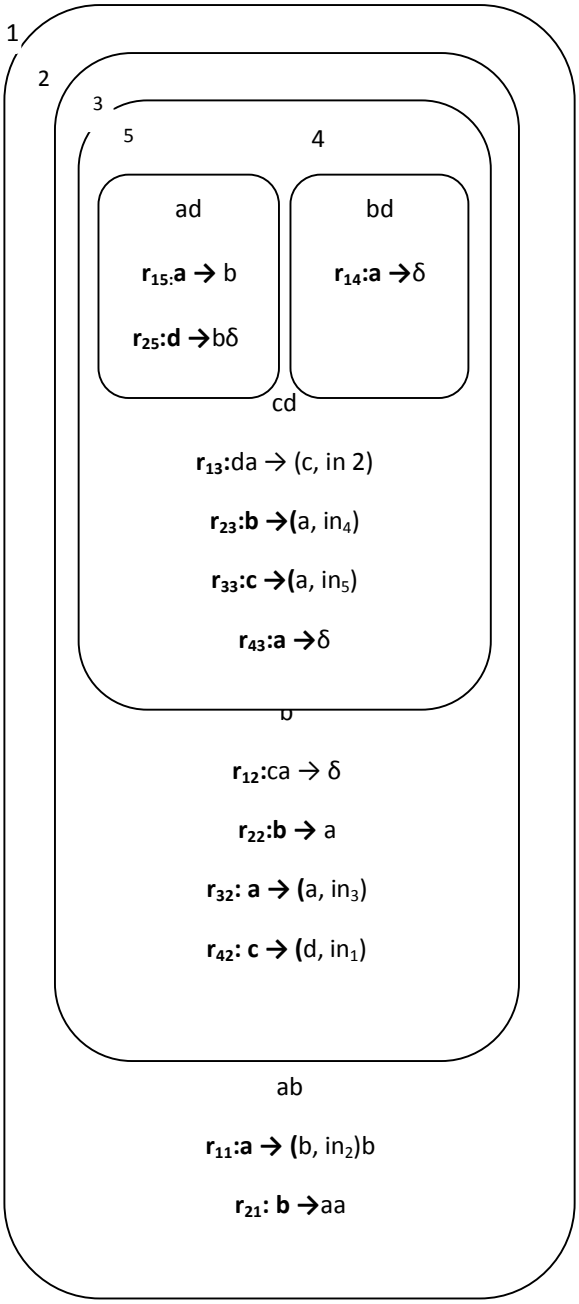

Fig.(5) a system structure of a five membrane

The application runs the exampled system shown in Fig.(5) and measures the average time for executing one rule and comparing the results obtained from applying rule in the two way (apply one by one

rule at a time and apply a power set of rules at a time) Table (2) and Fig(6) show the obtained results. The experiment is done on a multiple of data shown in example and measured the average time to execute one rule in two ways for applying rules.

As shown in the table and graph of the result, the way one by one applying rule is better when the data are few; on the other hand, the power set in applying rules is better when the data processed is huge.

Table(2) the average execution time for a rule

Multiple number of objects	One by one Appling rules	Power set Appling rules
1	0.01	0.012909091
5	0.006583333	0.008
10	0.006583333	0.00427907
20	0.006416667	0.00412766

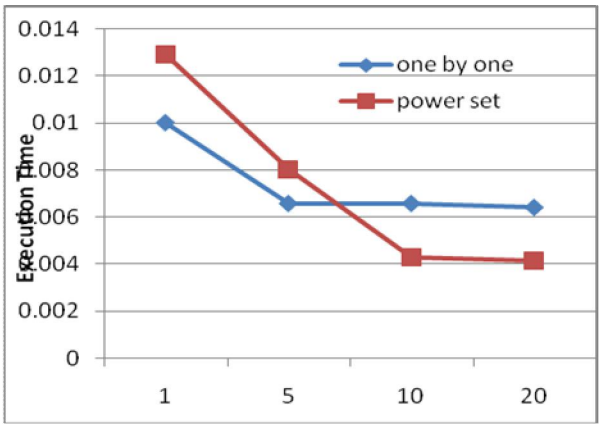

Fig (6) the average execution time for a rule

8. Conclusion

We investigated an alternative system to the p system; the main idea of the proposed system that can be seen as a system computes through two phases

Application phase: which can be seen as a distributed PTMs processing their data in localized manner and it cloud be said that each machine deals with its environment as supplier or consumer of data, and each machine has been broken down into submachine's and we focused on the design of submachine's which apply and execute rules.

Communication phase: we consider this phase as a machine coordinating the machines in the application phase and transferring the data among these machines; the design of this machine will be explained in a further research.

References

1. Gheorghe Paun, "Computing with membranes" In Turku University Computer Science Research Report No. 208, 1998.
2. Gheorghe Paun, "A quick introduction to membrane computing" The Journal of Logic and Algebraic Programming, 79, 2010, 291-294
3. Rozenberg G., B¨ack T., Kok J. N., "Handbook of Natural Computing", volume II. Springer eds. (2011).
4. Dina Goldin, Peter Wegner, "Persistent Turing Machines", Brown University Technical Report, 1998.
5. James L. Hein, "Theory of computation, by Jones and artlet Publisher eds. (1996)
6. J.L., J. D´ıaz, and J. G´abarro. "Structural complexity", Vol. I, 2nd edition, Springer-Verlag, Berlin, 1995.
7. J.E., J.D. Ullman. "Introduction to Automata Theory", Lan- guages, and Computation, Addison-Wesley Publishing Company, Read-ing, Mass., 1979.
8. Gabriel Ciobanu, G. Păun, Mario J.Perez "Applications of Membrane Computing " Springer-Verlag Berlin Heidelberg, 2006.
9. Biljana Petreska and Christof Teuscher "A Reconfigurable Hardware Membrane System" Membrane Computing Lecture Notes in Computer Science Volume 2933, 2004, pp 269-285.

4/28/2015