Looking for P Truth

Andrés CORDÓN-FRANCO
Miguel Angel GUTIÉRREZ-NARANJO
Mario J. PÉREZ-JIMÉNEZ

Research Group on Natural Computing
Department of Computer Science and Artificial Intelligence
University of Sevilla
Avda. Reina Mercedes s/n, 41012 Sevilla, Spain
E-mail: {acordon,magutier,marper}@us.es

Abstract. In a general sense, Logic studies how to derive new pieces of information from previous ones. In this paper we explore the analogies between P systems, where new configurations can be obtained from previous ones by using a set of rules, and the derivation of new theorems from previous ones or from axioms by using inference rules.

1 Introduction

In P systems [5], the information about a configuration is stored in multisets associated with a like-tree structure of membranes. A given configuration produces, in a discrete amount of time (a *evolution step*), a new configuration by using a set of rules in parallel maximal mode. Usually, the set of rules can be applied in a non deterministic way; so it is more convenient to consider, instead of a single configuration, the set of all the configurations that can be obtained from a previous one in one step by using the corresponding set of rules of the P system.

This schema of generating new pieces of information from previous ones by using a set of rules has been widely studied both in Logic and in Computer Science. Production systems and systems based on rewriting rules are only examples of systems where *if-then* rules are applied. From a Computer Science point of view, *if-then* rules provide a general framework for several programming paradigms. In fact, we can find specific languages as CLIPS [6] or Prolog [1, 7] where these rules are the basis of the system of representation and reasoning.

In this paper we initiate the study of the similarities between inference models based upon first–order logic and P systems. We focus on the definition of a fixpoint semantic for P systems. In the literature, a good amount of examples of fixpoint semantics for programming languages can be found and they provide evidence that this kind of semantics constitutes an adequate tool for motivating and justifying methods for obtaining properties of programs.

The paper is organized as follows. Section 2 describes how the language of first–order logic can be used to represent the configurations of a P system. In section 3 the analogies between P systems and Logic are shown in a more abstract level. An *Evolution Operator* is

presented and a classical result in Logic from fixpoint semantics is adapted to P systems. Finally, in section 4 we briefly discuss some lines for the future work.

2 Logic and P Systems

If we consider P systems as systems of representation of knowledge and reasoning, the relation with Logic can be settled at several levels.

In a first approach, we can consider first–order logic as an abstract system to represent objects and relations among them. In this way we can consider a set of variables and constant, function and predicate symbols to represent the information of a P system as a set of formulae.

In fact, this has been done in [2, 3] where a conjunction of one–literal clauses is used to describe the information contained in one configuration. This set of clauses is a Prolog program, which can be handled by another Prolog program which produces the Prolog codification of the configuration(s) resulting of one evolution step.

The first–order language used in [2, 3] to describe a configuration for a P system with active membranes consists of one binary predicate symbol `::`, four binary function symbols[1]: `at_time, with, at, ec` and as many constants as elements belong to the P system alphabet joint with three distinguished symbols `1,-1` and `0` to denote, respectively, positive, negative and neutral charge and a suitable set of constant symbols to describe the labels. For example, the configuration

$$[_s\,[_e\,[_r\ ab\ ]_r^+\,[_f\,c\,]_f^0\,z_1\,z_2\,]_e^0\,d]_s^-$$

of the P system `p1` at time 2 can be represented as the following first–order formula:

$$\begin{aligned}
& \texttt{::(p1,at_time(with(at(ec(s,-1),[]),[d]),2))} \\
\wedge\ & \texttt{::(p1,at_time(with(at(ec(e,0),[1]),[}z_1\texttt{, }z_2\texttt{]),2))} \\
\wedge\ & \texttt{::(p1,at_time(with(at(ec(r,1),[1,1]),[a, b]),2))} \\
\wedge\ & \texttt{::(p1,at_time(with(at(ec(f,0),[2,1]),[c]),2)),}
\end{aligned}$$

with the intended interpretation of the atoms

`::(p1,at_time(with(at(ec(`*Label*`,`*Charge*`),`*Position*`),`*Multiset*`),`*Step*`))`

Notice that, by using infix operators *x f y* instead of prefix operators *f(x,y)* and an appropriate definition of precedence among symbols, the formula can be written in a more suggestive manner:

$$\begin{aligned}
& \texttt{p1 :: s ec -1 at [] with [d] at_time 2} \\
\wedge\ & \texttt{p1 :: e ec 0 at [1] with [}z_1\texttt{, }z_2\texttt{] at_time 2} \\
\wedge\ & \texttt{p1 :: r ec 1 at [1,1] with [a,b] at_time 2} \\
\wedge\ & \texttt{p1 :: f ec 0 at [2,1] with [c] at_time 2}
\end{aligned}$$

Now we adopt a higher abstraction level and point out natural connections between P systems and usual inference mechanisms for logic. Suppose that the formula ϕ_n represents one configuration of the P system at time n. Our purpose is to define an inference rule which takes the formula ϕ_n and the set of rules of the P system (these rules can also be expressed by means of a set of first–order formulae) as antecedents and outputs the formula ϕ_{n+1} representing the configuration(s) of the P system at time $n+1$. This is the starting point of next section.

[1] Here we do not describe the representation of a list as a logic term; for details, see [1].

3 The Evolution Operator

Roughly speaking, a computation of a P system is a sequence of transitions, in each of which we get a new configuration from a previous one. At the beginning we have some information about membranes and objects in these membranes expressed in the initial configuration of the P system. Besides we have a set of rules which indicates how to obtain new pieces of information from previous ones. After the first step of evolution we get new information codified in the form of one (or more than one, if the considered P system is not deterministic) configuration. The process goes on by generating more and more configurations from previous ones. In Logic, the initial pieces of information correspond to the *axioms* of the system. From initial information, we obtain new pieces of information, *theorems*, by using the corresponding *inference rules*. These theorems are also used as starting points to generate newer ones and so on.

Previous arguments naturally lead to establish connections between the evolution process of a P system and usual theorem proving procedures. This analogy can be summarized as in Figure 1.

Using this analogy, a fixpoint semantics for P systems based on classic results can be developed.

Notice that for P systems with input, the initial configuration must capture this additional information, consequently, in the analogy described above, there does not exist a single set of axioms for the P system, but the set of axioms changes with the input data.

First we define an operator T which associates to every configuration C the set $\{C_1, \ldots, C_n\}$ of all configurations reachable from C in one evolution step. A configuration[2] is a description of the membrane structure together with the multisets associated with each membrane and other extra information (labels, electrical charges, ...) if necessary. Formally, we fix a P system, Π, and we denote by $\mathcal{C}$ the set of all configurations for the alphabet associated with Π. Then we define (as usual, $2^{\mathcal{C}}$ denotes the power set of $\mathcal{C}$).

Axioms	$\Leftrightarrow$	Initial configuration
Inference rules	$\Leftrightarrow$	Rules of the P system
Theorems	$\Leftrightarrow$	(Reachable) Configurations

Figure 1: Analogy between Logic and P systems

Definition 3.1 *Let T be a map from $\mathcal{C}$ into $2^{\mathcal{C}}$ defined as follows:*

$$\begin{array}{rccl} T: & \mathcal{C} & \rightarrow & 2^{\mathcal{C}}, \\ & C & \mapsto & T(C) = \{C_1, \ldots, C_n\}. \end{array}$$

From a general point of view, the value $T(C)$ encodes all the information that can be obtained from C by using the set of rules of the P system Π. The Operator T can be generalized to another one, T^*, which outputs all the information obtained from a *set* of configurations. The *Evolution Operator* is defined as follows.

Definition 3.2 ***(The Evolution Operator)*** *Let T^* be a map from $2^{\mathcal{C}}$ into $2^{\mathcal{C}}$ defined as follows:*

[2]Let us remark that we assume that configurations are codified in an adequate formal language. We skip technical details in this work and instead we focus on motivating our ideas.

$$\begin{array}{rccl} T^*: & 2^{\mathcal{C}} & \to & 2^{\mathcal{C}}, \\ & \mathcal{S} & \mapsto & T^*(\mathcal{S}) = \bigcup_{C \in \mathcal{S}} T(C) \cup \mathcal{S}. \end{array}$$

The set of configurations $T^*(\mathcal{S})$ represents all the information obtained from $\mathcal{S}$ in one step of evolution of the P system Π. The information contained in $\mathcal{S}$ does not disappear in the evolution step but it is contained in $T^*(\mathcal{S})$ as well. Note that $T^*(\mathcal{S})$ can be equal to $\mathcal{S}$ if no new information is generated.

Next result, although straightforward, states the main property (in the present context) of the operator T^*.

Lemma 3.1 *$T^*(\mathcal{S})$ is a monotone operator.*

A well known result in the theory of fixpoints allows us to ensure the existence of fixpoints for the Evolution Operator. Namely, we have the following theorem

Theorem 3.1 *The operator T^* has a least fixpoint.*

Proof. The result follows by the facts that the power set of the set of all configurations, $2^{\mathcal{C}}$, is a complete lattice and T^* is a monotone operator on it. □

Now we face the problem of defining a (suitable) semantics for P systems (let us recall that in Logic the semantics of a formal language deals with the interpretation of the language and includes such notions as meaning, models and truth). To this end, we adapt the classical definition of model based on set theory.

Definition 3.3 *A model of a P system (eventually with input D) is a set of configurations containing its initial configuration (together with D), and closed under the rules of the P system.*

The main result in this work is the following theorem stating that the Evolution Operator captures our notion of truth for a P system.

Theorem 3.2 *The least fixpoint of the Evolution Operator, T^*, associated with a P system is the intersection of all the models of the P system, and is equal to the set of all the reachable configurations of the P system.*

The proof of this theorem is an easy modification of the corresponding classical result in the study of first–order clausal logic[3]. Moreover, if the Evolution Operator is interpreted as the syntax and the fixpoint semantics as semantics for a P system, then the previous result establishes the equivalence between syntax and semantics. Consequently, it becomes a special case of the Completeness Theorem.

[3]See, for example, [4] for details.

4 Final Remarks

In this paper we have addressed the analogies between the evolution of P systems and the inference of new theorems in Logic. From an abstract point of view, we have shown that they are comparable but, of course, the similarities studied here are far from be the unique possible ones and further work has to be done in this direction. For example, in the approach presented above, all reachable configurations are considered as *theorems* in our analogy, regardless whether or not they are or can lead to *halting* configurations. Another approach where only halting configurations are considered relevant information is possible. In this sense, a new definition of truth (in the sense of *logic truth*) has to be looked for.

Acknowledgements. The support of this research through the project TIC2002-04220-C03-01 of the Ministerio de Ciencia y Tecnología of Spain, cofinanced by FEDER funds, is gratefully acknowledged.

References

[1] I. Bratko: *PROLOG Programming for Artificial Intelligence*, Third edition. Addison-Wesley, 2001.

[2] A. Cordón-Franco, M.A. Gutiérrez-Naranjo, M. Pérez-Jiménez, A. Riscos-Núñez, and F. Sancho-Caparrini: A Prolog simulator for deterministic P systems with active membranes, *New Generation Computing*. In press.

[3] A. Cordón-Franco, M.A. Gutiérrez-Naranjo, M. Pérez-Jiménez, A. Riscos-Núñez, and F. Sancho-Caparrini: Implementing in Prolog an effective cellular solution to the Knapsack Problem, *Membrane Computing WMC 2003, Lecture Notes in Computer Science* **2933**. Springer-Verlag, 2004, 140–152.

[4] J.W. Lloyd: *Foundations of Logic Programming*. (2nd ed.) Springer-Verlag, Berlin, 1987.

[5] Gh. Păun: Computing with membranes, *Journal of Computer and System Sciences*, **61**, 1 (2000), 108–143.

[6] CLIPS – A Tool for Building Expert Systems, `http://www.ghg.net/clips/CLIPS.html`.

[7] Logic Programming: `http://www.afm.sbu.ac.uk/logic-prog/`.