Testing based on P systems – an overview

Marian Gheorghe[1,2] and Florentin Ipate[2]

[1] Department of Computer Science, The University of Sheffield
Regent Court, Portobello Street, Sheffield S1 4DP, UK
Email: M.Gheorghe@dcs.shef.ac.uk

[2] Department of Computer Science, Faculty of Mathematics and Computer Science
The University of Pitesti
Str Targu din Vale 1, 110040 Pitesti
Email: florentin.ipate@ifsoft.ro

Abstract. In this extended abstract there are surveyed various testing approaches utilised so far for applications based on P systems.

Keywords: P systems, grammars, finite state machines, model based testing

1 Introduction

All software applications, as any engineering products, irrespective of their nature and purpose, are thoroughly tested before being released, installed and used. Testing appears everywhere, is part of any technology, and does not have a substitute. In many hardware or software systems testing is conducted together with formal verification, especially when a certain formal model is utilised. In software industry testing is a necessary mechanism to increase the confidence in the product correctness and to make sure it works properly.

P systems area, initially introduced by [11], has been under an intensive investigation in the last decade. It covers a broad range of aspects, from theoretical investigations into the computational power and descriptional complexity of various mechanisms, to applications in modelling different natural or engineered systems, and from interactions with other computational models to implementations of various problems utilising either certain tools or general purpose programming languages. An account of the various developments of the field, mostly at the theoretical level, is provided in [13], [12]; applications of P systems are presented in [2]. The most recent research aspects of this field are reported in [14].

Testing P systems has been so far considered by using certain coverage principles. More often the rule coverage is utilised, by taking into account different contexts. In order to reveal the usage of the rules, grammar and automaton based methods are derived from P systems specifications. These two types of testing methods are reviewed in the following sections. Some specific test set generation methods are analysed and discussed.

2 Grammar based methods

In the context of grammar testing it is assumed that for a given specification defined as a grammar, an implementation of it exists and this will be tested. In order to test the implementation, a test set is built, as a finite set of sequences containing references to rules.

Although there are similarities between context-free grammars utilised in grammar testing and basic P systems, that we aim to consider, there are also major differences that pose new problems in defining testing methods and strategies to obtain tests sets. Some of the difficulties encountered when some grammar-like testing procedures are introduced, are related to: the hierarchical compartmentalisation of the P system model, parallel behaviour, communication mechanisms, the lack of a non-terminal alphabet and the use of multisets of objects instead of sets of strings.

The rule coverage criteria discussed will be illustrated for one compartment P system, i.e., $\Pi = (V, \mu, w, R)$, where $\mu = [_1]_1$. The simplest and most basic rule coverage criterion, called *rule coverage*, is defined in such a way that every rule from R is covered by a certain computation; i.e., for each rule $r \in R, r : a \rightarrow v$, there is a multiset u_r over V which *covers* r (there is a computation $w \Longrightarrow^* xay \Longrightarrow x'vy' \Longrightarrow^* u_r$; $w, x, y, v, u_r \in V^*$, $a \in V$). Some more complex coverage criteria can be considered (see [4], [5]).

Let us consider the following one compartment P system, $\Pi_1 = (V_1, \mu_1, w_1, R_1)$, where $V_1 = \{s, a, b, c\}$; $\mu_1 = [_1]_1$ - i.e., one compartment, denoted by 1; $w_1 = s$; $R_1 = \{r_1 : s \rightarrow ab, r_2 : a \rightarrow c, r_3 : b \rightarrow bc, r_4 : b \rightarrow c\}$. Each multiset w, will be denoted by a vector of non-negative integer numbers $(|w|_s, |w|_a, |w|_b, |w|_c)$. Test sets for Π_1 satisfying the rule coverage criterion are

- $T_{1,1} = \{(0, 1, 1, 0), (0, 0, 1, 2), (0, 0, 0, 2)\}$ and
- $T_{1,2} = \{(0, 1, 1, 0), (0, 0, 1, 2), (0, 0, 0, 3)\}$.

3 Finite state machine based methods

Finite state machine based testing is widely used for software testing. It provides very efficient and exhaustive testing strategies and well investigated methods to generate test sets. In this case it is assumed that a model of the system under test is provided in the form of a finite state machine. In our case we will consider a way to obtain such a machine from a partial computation in a P system. More precisely we will consider computations of at most k steps, for a given integer k, starting from the initial multisets. These can be considered paths in an automaton defining partial computations of no more than k steps. Either this automaton or a minimal one covering it can be now utilised as a model to generate test sets (see [5], [7]).

A different aproach can be also considered by using a special class of state machines, called X-machines. Given that the relationships between various classes of P systems and these machines are well studied ([14], [1]) and the X-machine based testing is well developed, standard techniques for generatig tests sets based on X-machines can be adapted to the case of P systems [6].

Specific coverage criteria can be defined in the case of finite state machine based testing. One such criterion, called *transition coverage*, aims to produce a test set in such a way that every single transition of the model is covered.

If we build a finite state machine associated with the previous P system, Π_1, for partial computations of length at most 4, then a test set satisfying the transition cover principle is
$T_{1,s} = \{(1,0,0,0),(0,1,1,0),(0,0,1,2),(0,0,0,2),(0,0,1,3),(0,0,0,3)\}$.

The transition cover criterion, however, does not only depend on the rules applied, but also on the state reached by the system when a given rule has been applied.

4 Generating test sets using model checking

The generation of different test sets, according to certain coverage criteria, can be done by utilising some specific algorithms or by applying some tools that indirectly will generate test sets. Such tools, like model checkers, can be used to verify some general properties of a model and when these are not fulfilled then some counter-examples are produced, which act as test sets in certain circumstances.

In the case of P systems an encoding based on a Kripke structure associated with the system is provided for model checkers like NuSMV [9] or SPIN [10]. This relies on certain operations defined in [3] and encapsulates the main features of a P system, including maximal parallelism and communication, but within a finite space of values associated with the objects present in the system. The rule coverage principle is expressed by using temporal logics queries available in such contexts. By negating specific coverage criteria, counter-examples are generated. For instance the rule coverage set $T_{1,1}$ can be obtained in this way.

5 Conclusions

P systems based testing methods are reviewed and some coverage principles presented. Two main classes of methods, based on grammars and finite state machines, are introduced and specific test generation tools based on model checking techniques are mentioned. Apart from these methods some other approaches have been considered when mutation techniques have been employed [8].

Acknowledgements. The research of MG and FI is supported by CNCSIS grant no.643/2009, *An integrated evolutionary approach to formal modelling and testing.*

References

1. J. Aguado, T. Balanescu, A. Cowling, M. Gheorghe, M. Holcombe, F. Ipate, (2002) P systems with replicated rewriting and stream X-machines (Eilenberg machines), *Fundamenta Informaticae,* 49, 17–33.

2. G. Ciobanu, M.J. Pérez-Jiménez, Gh. Păun, eds., (2006) *Applications of membrane computing*, Natural Computing Series, Springer.
3. Z. Dang, O.H. Ibarra, C. Li, G. Xie, (2006) Decidability of model-checking P systems, *Journal of Automata, Languages and Combinatorics,* 11, 179-198.
4. M. Gheorghe, F. Ipate, (2008) On testing P systems, in D. W. Corne, P. Frisco, Gh. Păun, G. Rozenberg, A. Salomaa (eds.), *9th Workshop on Membrane Computing,* Lecture Notes in Computer Science, 5391, 204–216.
5. M. Gheorghe, F. Ipate, R. Lefticaru, C. Dragomir (2010) An integrated approach to P systems formal verification, in *Proceedings of the 11th Conference on Membrane Computing,* (accepted).
6. F. Ipate, M. Gheorghe, (2008) Testing non-deterministic stream X-machine models and P systems, *Electronic Notes in Theoretical Computer Science,* 227, 113–226.
7. F. Ipate, M. Gheorghe, (2009) Finite state based testing of P systems, *Natural Computing,* 8, 833–846.
8. F. Ipate, M. Gheorghe, (2009) Mutation based testing of P systems, *International Journal of Computers, Communications & Control,* 4, 253–262.
9. F. Ipate, M. Gheorghe, R. Lefticaru, (2010) Test generation from P systems using model checking, *Journal of Logic and Algebraic Programming,* (in press).
10. F. Ipate, R. Lefticaru, C. Tudose (2010) Formal verification of P systems using SPIN, *International Journal of Foundations of Computer Science,* (submitted).
11. Gh. Păun, (2000) Computing with membranes, *Journal of Computer and System Sciences*, 61, 108–143.
12. Gh. Păun, (2002) *Membrane computing. An introduction,* Springer, Berlin.
13. Gh. Păun, G. Rozenberg, (2002) A guide to membrane computing, *Theoretical Computer Science*, 287, 73–100.
14. Gh. Păun, G. Rozenberg, A. Salomaa, eds., (2009) *The Oxford handbook of membrane computing,* The Oxford University Press, Oxford.