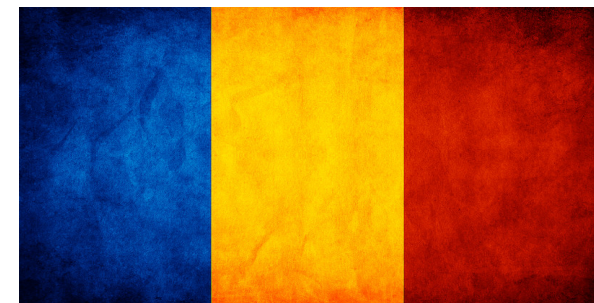


Second Rodin User and Developer Workshop,
Dusseldorf, 20-22 september 2010

Event-B models of P systems

Florentin IPATE
Adrian TURCANU



Romania



Summary

- Generalities about P systems
- A simple example
- A first Event-B model
- About Kripke structures
- The corresponding Kripke structure of a P system
- A refinement
- LTL
- On P system testing using model checking
- Conclusions and future work



Generalities about P systems

Membrane computing, the research field initiated by Gheorghe Paun in 1998, aims to define computational models, called P systems, which are inspired by the behavior and structure of the living cell.

A n – membrane P system is a tuple $\Pi = (V, \mu, w_1, \dots, w_n, R_1, \dots, R_n)$, where :

- V is a finite alphabet
- μ defines a membrane structure which is a hierarchical arrangement of n compartments called regions delimited by membranes
- w_i represents the initial multiset of objects (strings over V) occurring in region i
- R_i denotes the set of processing rules applied in region i .



Rules in each region have the form $u \rightarrow (a_1, t_1), \dots, (a_m, t_m)$, where $u \in V^*$, $a_i \in V$, $t_i \in \{in, out, here\}$. When such a rule is applied to a multiset u in the current region :

- u is replaced by the symbols a_i with $t_i = here$
- symbols a_i with $t_i = out$ are sent to the outer region or outside the system
- symbols a_i with $t_i = in$ are sent into one of the regions contained in the current one, arbitrarily chosen.

The rules are applied in a maximally parallel mode : they are used in all the regions in the same time and in each region all the symbols that may be proceed must be.



A configuration of Π is a tuple $c = (u_1, \dots, u_n)$, where $u_i \in V^*$ is the multiset associated with the region $i = \overline{1, n}$.

A configuration is called terminal if none of its components can be further derived.

The output of the computation is given by the objects in a specified output membrane.

A simple example

We give as a first example a very simple P system, with one membrane, having the alphabet $V = \{s, a, b, c\}$, the initial multiset $w = s$ and the rules :

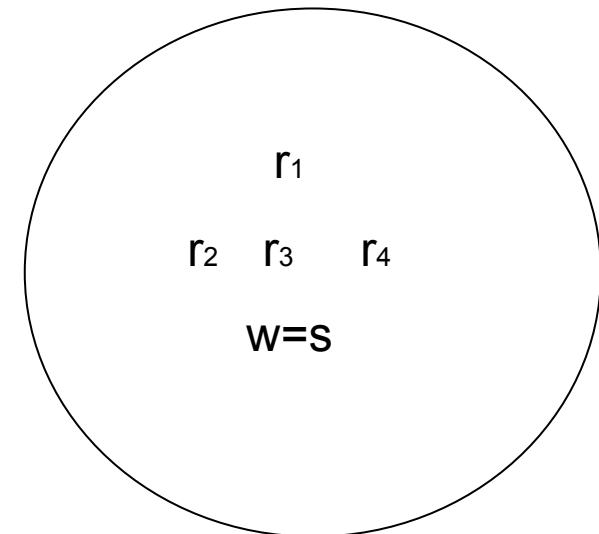
$$r_1 : s \rightarrow ab$$

$$r_2 : a \rightarrow c$$

$$r_3 : b \rightarrow bc$$

$$r_4 : b \rightarrow c$$

We will use it for our first Event - B models.





A first Event-B model

We will give some general ideas about modeling an 1 - membrane P system using the Event - B formalism.

For each object we introduce a variable representing the number of objects of that type that can be consumed.

Also, for any object from the right side of any rule we will introduce an auxiliary variable representing the number of produced objects of that type. It will be used in order to be incremented as a consequence of the application of different rules at one step.

We will introduce an event for each rule.

Finally, when no rule can be further applied, we will introduce an actualization event : every variable a will receive the value of a_1 ($a := a_1$) and a_1 resets to 0.

Next, we apply these general ideas to the first example of P system, presented before.

We use 7 variables, all natural numbers :

- s for the number of occurrences of object s ;
- a, a_1 for the number of occurrences of object a ;
- b, b_1 for the number of occurrences of object b ;
- c, c_1 for the number of occurrences of object c ;



We use the following events :

– event *initialization* : $s = 1, a = b = c = a_1 = b_1 = c_1 = 0$ end.

$r_1 : s \rightarrow ab$

– event *rule_1* : when $s > 0$ then $s := s - 1$

$a_1 := a_1 + 1$

$b_1 := b_1 + 1$


end.

$r_2 : a \rightarrow c$

– event *rule_2* : when $a > 0$ then $a := a - 1$

$c_1 := c_1 + 1$

end.



$r_3 : b \rightarrow bc$

– event *rule_3* : when $b > 0$ then $c_1 := c_1 + 1$ end.

$r_4 : b \rightarrow c$

– event *rule_4* : when $b > 0$ then $b := b - 1$

$c_1 := c_1 + 1$

end.

– event *actualization* : when $s = a = b = 0$ then $a := a_1$

$b := b_1$

$c := c_1$

$a_1 = b_1 = c_1 = 0$

end.

So, the first model is done.

The 21 POs are easy to check and are automatically.

About Kripke structures

A Kripke structure over a set of atomic propositions AP is a four tuple $M = (S, I, H, L)$, where :

- S is a finite set of states
- $I \subseteq S$ is an initials state set
- $H \subseteq S \times S$ is a left- total transition relation
- $L : S \rightarrow 2^{AP}$ is a labelling function that maps each state into the set of atomic propositions that hold in that state (i.e. $L(s) = \{p \in AP \mid p \text{ is true in state } s\}$).

A path in a this Kripke structure M is an infinite sequence of states $\pi = s_0 s_1 \dots$ such that $(s_i, s_{i+1}) \in H$, for every $i \geq 0$. We say that path π starts in state s if $s_0 = s$. A finite path is a prefix of an infinite path.

The following notations are used :

$Paths(M, s)$ = the set of paths of M that start in state s .

$Path(M) = \bigcup_{s \in I} Paths(M, s)$ i.e. the set of all paths starting in an initial state.

$FPath(M)$ = the set of all finite paths from initial states.


The corresponding Kripke structure of a P system

We consider an 1 – membrane P system $\Pi = (V, \mu, w, R)$, where $R = \{r_1, \dots, r_m\}$. Each rule r_i , $i = \overline{1, m}$, is of the form $u_i \rightarrow v_i$, where u_i and v_i are multisets over an alphabet V .

In order to define the Kripke structure equivalent to Π we use two predicates :

- $MAXPARAL(u, v, n_1, \dots, n_m)$, $u \in N^k$, $n_1, \dots, n_m \in N$ with the following signification : a derivation of the configuration u in maximally parallel mode is obtained by applying rules r_1, \dots, r_m , respectively n_1, \dots, n_m times, and
- $APPLY(u, v, n_1, \dots, n_m)$, denoting that v is obtained from u by applying rules r_1, \dots, r_m , respectively n_1, \dots, n_m times.

Remark that $MAXPARAL(u, v, 0, \dots, 0)$ represents that u is a terminal configuration of Π .



In order to keep the number of configurations under control we will assume that each component of a configuration cannot exceed an established upper bound, denoted MAX , and each rule can only be applied for at most a given number of times, denoted SUP .

We will use the following notations :

- $u \leq MAX$, if all the components of the configuration u doesn't exceed MAX ;
- $(n_1, \dots, n_m) \leq SUP$ if $n_i \leq SUP$ for every $i = \overline{1, m}$;
- $N_{MAX}^k = \{u \in N^k \mid u \leq MAX\}$
- $N_{SUP}^m = \{(n_1, \dots, n_m) \in N^m \mid (n_1, \dots, n_m) \leq SUP\}$.

The system is assumed to crash whenever $u \leq MAX$ or $(n_1, \dots, n_m) \leq SUP$ does not hold.

Obviously, the normal termination occurs when $u \leq MAX$ and $(n_1, \dots, n_m) \leq SUP$ and no rule can be further applied.

The corresponding Kripke structure $M = (S, H, I, L)$ to the P -system Π is defined as follows: $S = N_{MAX}^k \cup \{Halt, Crash\}$, $Halt, Crash \notin N_{MAX}^k$, $Halt \neq Crash$, $I = w$ and the left-total relation H is defined by:

- $(u, v) \in H$, $u, v \in N_{MAX}^k$ if $\exists (n_1, \dots, n_m) \in N_{SUP}^m \setminus \{(0, \dots, 0)\}$ such that $MAXPARAL(\psi u_1, v_1, n_1, \dots, u_m, v_m, n_m) \wedge APPLY(u, v, u_1, v_1, n_1, \dots, u_m, v_m, n_m)$;
- $(u, Halt) \in H$, $u \in N_{MAX}^k$ if $MAXPARAL(\psi u_1, v_1, 0, \dots, u_m, v_m, 0)$;
(no rule can be further applied)
- $(u, Crash) \in H$, $u \in N_{MAX}^k$ if $\exists (n_1, \dots, n_m) \in N_{SUP}^m$ $v \in N^k$ such that $\neg((n_1, \dots, n_m) \leq SUP \wedge v \leq MAX) \wedge MAXPARAL(\psi u_1, v_1, n_1, \dots, u_m, v_m, n_m) \wedge APPLY(u, v, u_1, v_1, n_1, \dots, u_m, v_m, n_m)$;
- $(Halt, Halt) \in H$;
- $(Crash, Crash) \in H$.



A refinement

We will try now to give some ideas about modeling a P system using the corresponding Kripke structure. We will use again a 1 – membrane P system $\Pi = (V, \mu, w, R)$ with rules of the form $u_i \rightarrow v_i$.

First, we modify the set of states, $S = N_{MAX}^k \cup \{Halt, Crash\}$, in order to avoid a state explosion by introducing one single state for all the values from N_{MAX}^k denoted by *Running*. Hence, the Event - B model will have the set of states $S = \{Halt, Crash, Running\}$


We see this second model as a refinement of the first. We have to add :

- a variable n_i for each rule $r_i \in V$ showing the number of applications of it;
- one variable $state \in S$, for the current state of the model;
- the two constants *MAX* and *SUP* and the set of states and for this we will need a context

All the events introduced before has to be refined and happens in the state *Running*.

In order to finish our refinement we have to model :

- the transition from the state *Running* to the state *Crash* and
- the transition from the state *Running* to the state *Halt*



We consider that the transition from *Running* to *Crash* happens when we can still have objects to consume using some rule, but if we do that we exceed the upper bound for a rule counter.

For our previous example, we have the new event :

event *RunToCrash*: when $state = Running \wedge$
 $(s > 0 \wedge n_1 = SUP) \vee (a > 0 \wedge n_2 = SUP)$
 $\vee (b > 0 \wedge (n_2 = SUP \vee n_3 = SUP))$
then $state = Crash$
end.

The transition from *Running* to *Halt* will be considered when we are reaching a terminal configuration. Our example corresponds the following event :

event *RunToHalt*: when $state = Running \wedge s = 0 \wedge a + a_1 = 0 \wedge b + b_1 = 0$
then $state = Halt$
end.

The other events are easy to refine and so the refinement is done.



Rodin

Linear Temporal Logic (LTL)

Linear temporal logic (LTL) is a modal logic with special operators for time. An LTL formula consists of atomic propositions, boolean operators (\neg , \wedge , \vee , \rightarrow , \equiv) and temporal operators (X , U , G , F). The meaning of the temporal operators are the following:

- Xa : a has to be true in the next state (X = “next”)
- aUb : a has to hold from the current state up to a state where b is true (U = “until”)
- G : a condition has to hold at all states of a path (G = “always”)
- F : a condition eventually holds at some time in the future (F = “eventually”)

In LTL the only path quantifier is A (for every path), i.e. we can describe only one path properties per formula and the only state subformulas permitted are atomic propositions.

The BNF definition of LTL formula is given by:

$$\Phi ::= true \mid f \text{ als } a \in AP \mid \neg \Phi \mid \Phi_1 \wedge \Phi_2 \mid \Phi_1 \vee \Phi_2 \mid \Phi_1 \rightarrow \Phi_2 \mid \Phi_1 \equiv \Phi_2 \mid \Phi_1 U \Phi_2 \mid X\Phi \mid G\Phi \mid F\Phi$$

On P system testing using model checking

Model checking is a verification technique that explores all possible system states in a brute force manner. The idea of testing with model checkers is to interpret counterexamples as test cases.

For a 1-membrane P system $\Pi = (V, \mu, w, R)$ and its equivalent Kripke structure $M = (S, H, I, L)$ we can define, for example, the following coverage criteria :

A finite path (s_0, \dots, s_n) in M is called a test case which covers a rule r_i if there exists $p \leq n-1$ for which, $s_p, s_{p+1} \in N_{MAX}^k$ and exists $(n_1, \dots, n_m) \in N_{SUP}^m$, $n_i \geq 1$ such that $MAXPARAL(s_p, u_1, v_1, n_1, \dots, u_m, v_m, n_m) \wedge APPLY(s_p, s_{p+1}, u_1, v_1, n_1, \dots, u_m, v_m, n_m)$.

In this case a finite prefix of a counterexample for the *LTL* specification $G((n_i \geq 1) \wedge (state = other))$ considered for a model of M is a test case which covers rule r_i .

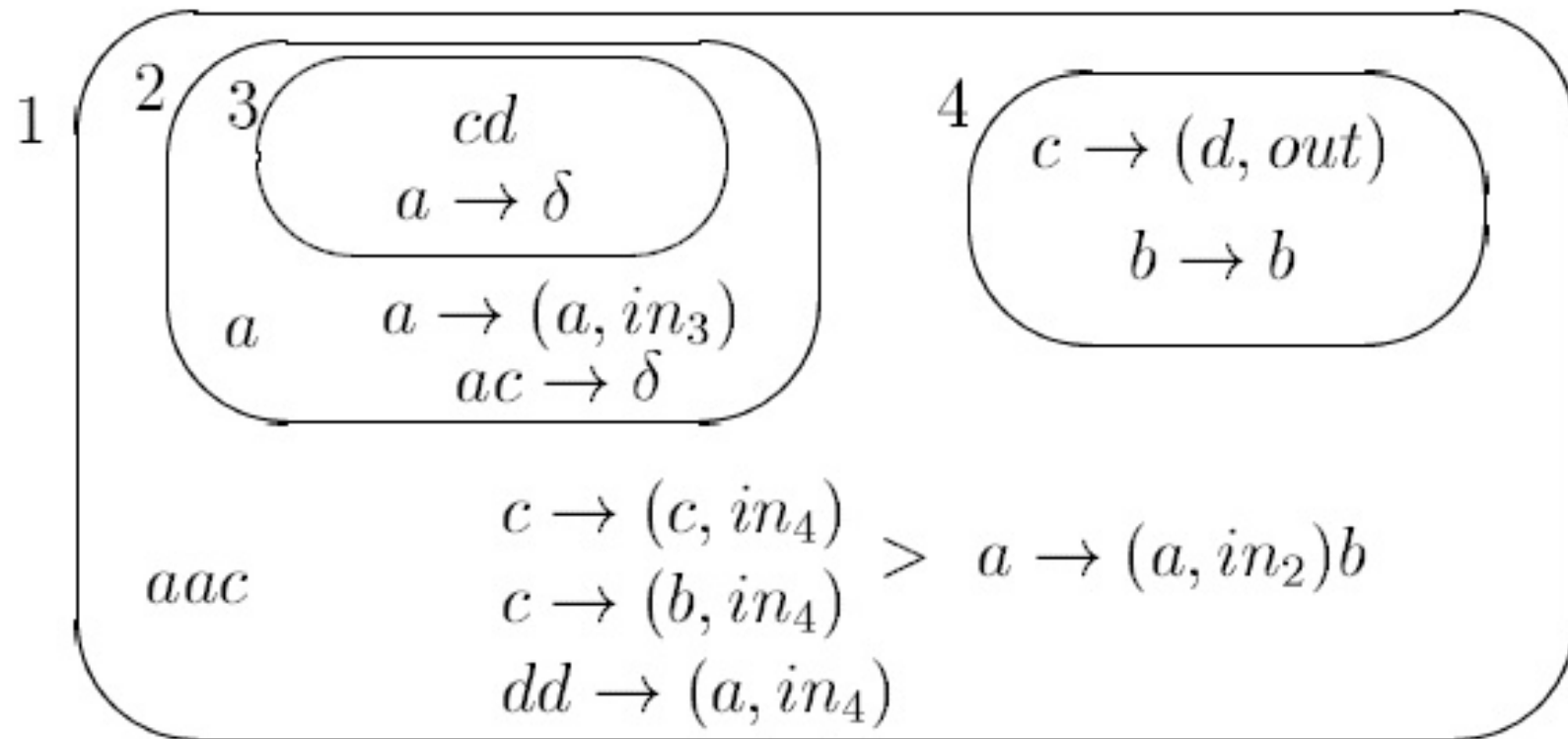
For more details see Ipate, F., Gheorghe, M., Lefticaru, R., Test Generation from P Systems Using Model Checking, 2010.



Conclusions and future work

- We made this presentation a simple model of P system using Event-B syntax.
- Some ideas about testing P system using model checking were presented.
- Our future work will concentrate on modeling more complicated P systems.
- We plan to use the model checker ProB on this models to generate test cases

Model a P system like this...



A dream? I hope not...



Event EndOfPresentation:

when

“you were paying attention to my presentation”

then

Thanks!