Formal Verification of P Systems with Active Membranes through Model Checking

Florentin Ipate[1], Raluca Lefticaru[1], Ignacio Pérez-Hurtado[2], Mario J. Pérez-Jiménez[2], Cristina Tudose[1]

[1]University of Pitesti

[2]University of Sevilla

Outline

- Background
- Theoretical basis of the approach
- Case study
- Conclusions

Background

• **Model checking**: automated technique for verifying if a *model* meets a given property (specified in temporal logic)

• **Model checker**: tool that takes as *input* the specification of a system (model) + temporal logic formulae. *Output*: "*true*" or "*false*" for each formula.

• If a property violation is discovered then a **counterexample** is returned.

• Properties to be checked: *safety* (something bad never happens), *liveness* (something good will eventually happen), *fairness* (does, under certain conditions, an event occur repeatedly?), invariants etc.

Previous work

- Decidability of model-checking properties for P systems

- Model verification:
  - Maude LTL model checker (using rewriting logic)
  - Prism (for stochastic systems)
  - NuSMV (symbolic model checker)
  - Spin (models written in Promela)
  - ProB (models written in B/Event-B)

- Model property extraction using Daikon

- Only P systems with static structure have been considered

- In this paper P systems with active membranes (in particular with cell division) are considered.

Steps for P system model checking

Given a P system, Π with active membranes build up the following steps:

- Kripke structure – M_Π associated with Π; translating the rules and the semantics of the Π to M_Π

- implement – M'_Π in Spin; (M_Π and M'_Π are not functionally equivalent)

- formulate properties – properties regarding the system are formulated as LTL formulae on M_Π

- transform – LTL formulae on M_Π into LTL formulae on M'_Π

Kripke structure associated with a P system (1)

Given a P system Π with rules R = $\{r_1, \ldots r_m\}$

• **states** of M_Π: configurations of Π plus two special states (***Crash*** and ***Halt***)

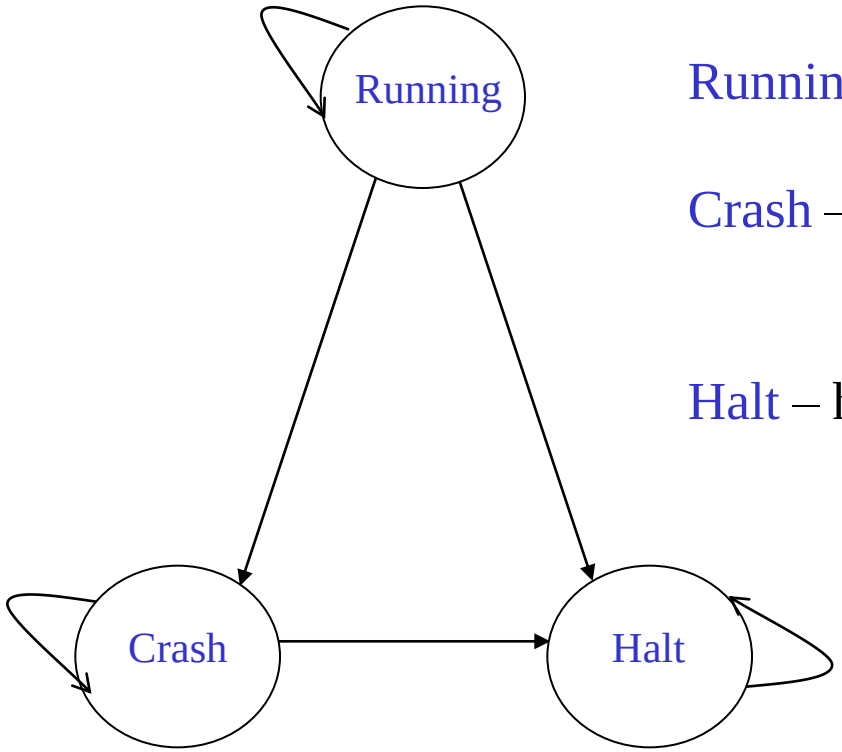
• **transitions** of M_Π: there is a transition from configuration *c* to configuration *d* if $\forall$membrane *i* in *c* $\exists n_1^i, ..., n_m^i$, such that:

- ***d* is obtained from *c*** by applying rules $r_1, \ldots, r_m, n_1^i, \ldots, n_m^i$ times, respectively, for every membrane *i*;
- **at least one rule** is applied;
- any membrane can be subject of only **one communication/ dissolution/ division rule**;
- a computation from *c* develops in **maximally parallel** mode.

Kripke structure associated with a P system (2)

- In order to keep the number of states finite:
  - **upper bound Max** on the number of symbols of any type in any membrane;
  - **upper bound Sup** on the number of applications of any rule;
  - whenever (at least) one of these upper bounds is exceeded, extra transitions to ***Crash*** are added.
- The halting configurations of the P system (i.e. in which no rule can be applied) are represented by extra transitions to ***Halt***.

M_{Π} diagram



Running (set of states) – normal behaviour;

Crash – abnormal behaviour
(upper bounds exceeded);

Halt – halting configurations.

Transforming LTL formulae (1)

- Transition between the states of M_{Π} are defined using "$\exists$"
- Most modelling languages (Promela included) do not support the existential (or the universal) quantifier
- A transition involving $\exists$ is normally implemented as a sequence of transitions (e.g. a "do - od" loop in Promela)
- Therefore M'_{Π} (the implementation of M_{Π}) will contain additional (intermediary) states
- LTL formulae on M_{Π} need to be translated into LTL formulae on M'_{Π}

Transforming LTL formulae (2)

Property	LTL specification
G p	[] (p || !$pInS$)
F p	<> (p && $pInS$)
p U q	(p || !$pInS$) U (q && $pInS$)
X p	X (!$pInS$ U (p && $pInS$))
p R q	(p && $pInS$) V (q || !$pInS$)

• pInS is a predicate which holds in the original (non-intermediary) states.

• e.g., "*Globally b>0*" $\Rightarrow$ "*Globally b > 0 or not pInS*" (b > 0 only for configurations corresponding to the P system, but not for the intermediary states).

Case study: Subset Sum problem (1)

Given a finite set $A = \{a_1, \ldots, a_n\}$, of n elements, where each element a_i has an associated weight, w_i, and a constant $k \in N$, it is requested to determine whether or not there exists a subset $B \subseteq A$ such that $w(B) = k$, where $w(B) = \sum_{a_i \in B} w_i$

Case study: Subset Sum problem (2)

- well-known NP-complete problem
- solved in linear time and in an uniform way using P systems with membrane division rules

  Pérez-Jiménez, M.J., Riscos-Núñez, A.: Solving the Subset-Sum problem by P systems with active membranes. New Generation Computing 23(4), 339-356 (2005)

- improved solution: the total cost is logarithmic in one variable and linear in the rest.

  Díaz-Pernil, D., Gutiérrez-Naranjo, M.A., Pérez-Jiménez, M.J., Riscos-Núñez, A.: A logarithmic bound for solving Subset Sum with P systems. In: Eleftherakis, G., Kefalas, P., Păun, G., Rozenberg, G., Salomaa, A. (eds.) Workshop on Membrane Computing. Lecture Notes in Computer Science, vol. 4860, pp. 257-270. Springer (2007)

Case study: Subset Sum problem (3)

- *Generation stage*: for every subset of *A*, a (single) membrane is generated via membrane division.
- *Calculation stage*: in each membrane the weight of the associated subset is calculated.
- *Checking stage*: in each membrane it is checked whether the weight of its associated subset is exactly *k*.
- *Output stage*: the system sends out the answer to the environment, according to the result of the checking stage.

Case study: Subset Sum problem (4)

$$\prod(\langle n,k\rangle) = (\Gamma(\langle n,k\rangle), \{e,s\}, \mu, w_e, w_s, R, i(n,k)), where:$$

- $\Gamma(\langle n,k\rangle) = \{x_0, x_1, \ldots, x_n\} \cup \{\overline{a}_0, \overline{a}, a_0, a, d_1, e_0, \ldots, e_n, q, q_0, \ldots, q_{2k+1}, z_0, \ldots, z_{2n+2k+2}, Yes, No_0, No, \#\}$ is the alphabet;
- $\mu = [_s [_e]_e^0]_s^0$ is the membrane structure;
- $w_s = z_0, w_e = e_0 a^k$ are the initial multisets;
- $i(n,k) = e$ and contains the code $x_1^{w_1} \ldots x_n^{w_n}$;

Case study: Subset Sum problem (5)

The set of rules, *R*:

(1) $[e_i]_e^0 \to [q]_e^-[e_i]_e^+, 0 \le i \le n;$
$[e_i]_e^+ \to [e_{i+1}]_e^0[e_{i+1}]_e^+, 0 \le i \le n.$ For each subset of *A* a membrane is generated.

(2) $[x_0 \to \bar{a}_0]_e^0; [x_0 \to \lambda]_e^+; [x_i \to x_{i-1}]_e^+,$ *for* $1 \le i \le n.$

The code from the input membrane is built in such a way that the multiplicity of x_j represents the weight of $a_j \in A$. These three rules calculate in $\bar{a}_0$ the weight of a subset.

(3) $[q \to q_0]_e^-; [\bar{a}_0 \to a_0]_e^+; [\bar{a} \to a]_e^-.$

The rules mark the beginning of the checking stage; the weight of the subset is now coded by the multiplicity of a_0.

(4) $[a_0]_e^- \to []_e^0\#; [a]_e^0 \to []_e^-\#.$

The number of occurrences of a_0 and a are compared in a checking loop.

Case study: Subset Sum problem (6)

(5) $[q_{2j} \to q_{2j+1}]_e^-, 0 \le j \le k; [q_{2j+1} \to q_{2j+2}]_e^0, 0 \le j \le k-1.$

Objects q_i are utilised as counters of the checking loop.

(6) $[q_{2k+1}]_e^- \to []_e^0 Yes; [q_{2k+1}]_e^0 \to []_e^0 \#; [q_{2j+1}]_e^- \to []_e^- \#; 0 \le j \le k-1.$

These rules provide an answer to the checking loop given that there are the same number of a_0 and a, more a_0 objects, or more a objects, respectively.

(7) $[z_i \to z_{i+1}]_s^0, 0 \le i \le 2n+2k+1; [z_{2n+2k+2} \to d_1 No_0]_s^0.$

Objects z_i control the checking stage in all membranes. When the checking stage is over d_1 and No_0 are released into the skin membrane.

(8) $[d_1]_s^0 \to []_s^+ d_1; [No_0 \to No]_s^+; [Yes]_s^+ \to []_s^0 Yes; [No]_s^+ \to []_s^0 No.$

In the final stage either *Yes* or *No* is sent out into the environment.

Experimental results (1)

- We have considered a set A with 2 and 3 elements, weights between 1 and 3, and k between 2 and 4.
- Two examples reported
  - (1): n = 3; k = 4; w = [1; 2; 2]
  - (2): n = 3; k = 3; w = [2; 2; 2]

Experimental results (2)

Property	**LTL specification**	Result 1	Result 2
Generally, there is not YES in the environment.	`[](env.Yes == 0 || !pInS)`	false	true
Generally, there is not NO in the environment.	`[](env.No == 0 || !pInS)`	true	false
Eventually, there is YES in the environment.	`<>(env.Yes == 1 && pInS)`	true	false
Eventually, there is NO in the environment.	`<>(env.No == 1 && pInS)`	false	true

Experimental results (3)

When e_1 appears in a specific membrane 1, with electrical charge 0, then a q will appear with a negative electrical charge.	`[]( ( membr[1].e[1]==1 && membr[1].charge==0 -> <> (membr[1].q==1 && membr[1].charge==-1)) || !pInS)`	true	true
When e_1 appears in a specific membrane 1, with electrical charge 0, then a q will appear with a positive electrical charge.	`[]( ( membr[1].e[1]==1 && membr[1].charge==0 -> <> (membr[1].q==1 && membr[1].charge==1)) || !pInS)`	false	false
For all i, j, if e_i appears in membrane j, with electrical charge 0, then q will appear in the same membrane with a negative electrical charge.	`[]((i>=0 && i<3 && j>=0 && j<10 && (membr[j].e[i]==1 && membr[j].charge==0 -> <> (membr[j].q==1 && membr[j].charge==-1 && pInS))) ||!pInS)`	true	true

Experimental results (4)

- Comparative simulations with both P-Lingua and the Promela code: the translation from P-Lingua to Promela does not introduce much overhead into the system.
- The maximum number of membranes obtained during the P system computation was 16 (for n = 3): Spin produced a response in at most 10 seconds (less than 5 seconds for simple queries).
- Much slower responses (several minutes) for larger n and complex queries.

Future experiments

- Test the limit for which Spin can produce a response in a reasonable time interval
- Investigate how our implementation can be improved in order to cope with the well-known state explosion
- Address more complex properties
- Identify invariants of various stages
- Automate the process of transforming the P system specification, given as a P-Lingua file, into a Promela model
- Other challenging, real life examples

Conclusions

- Significant advances in the area of model checking of P systems
- P systems with active membranes - P systems with cell division rules, having a bounded number of produced membranes
- Limitations due to the underlying nature of model checking
- Further work needed to improve performance and automate the process

Questions?