Verifying P Systems with Costs by Using Priced-Timed Maude

Bogdan Aman and Gabriel Ciobanu

Romanian Academy, Institute of Computer Science, Iaşi, Romania
E-mail: bogdan.aman@gmail.com, gabriel@info.uaic.ro

Summary. We consider P systems that assigns storage costs per step to membranes, and execution costs to rules. We present an abstract syntax of the new class of membrane systems, and then deal with costs by extending the operational semantics of P systems with promoters, inhibitors and registers. We use Priced-Timed Maude to implement the P systems with costs. By using such a rewriting engine which corresponds to the semantics of membrane systems with costs, we are able to prove the operational correctness of this implementation. Based on such an operational correspondence, we can analyze properly the evolutions of the P systems with costs, and verify several reachability properties, including the cost of computations that reach a given membrane configuration. This approach opens the way to various optimization problems related to membrane systems, problems making sense in a bio-inspired model which now can be verified by using a complex software platform.

1 Introduction

Membrane computing is introduced in [10] and represents now a well known branch of natural computing that aims to abstract computing ideas and formal models from the structure and functioning of living cells, as well as from the organization of cells in tissues, organs or other higher order structures such as colonies of cells [11]. Membrane systems (known also as P systems) are parallel and distributed models working with multisets of symbols in cell-like compartmental architectures. The existing results in membrane computing refer mainly to the P systems characterization of Turing computability, providing also some polynomial solutions to NP-complete problems by using an exponential workspace created in a "biological way".

Time was introduced and studied in the framework of membrane systems [2]. However, time is not the only quantitative notion of interest; other quantities such as energy [8] or accumulated cost can be included in such systems. The notions of energy and cost are connected to (evolution) time, because the longer the system evolves, the higher the energy and costs are. For simplicity, in this paper we study

only the (evolution) costs in a membrane system. A membrane system with costs is essentially a simple membrane system in which object storage costs per evolution step are assigned to membranes, and execution costs are assigned to rules. Notice that here we consider the cost only as an external/observer variable, and thus whether a rule is applicable only depends on available resources (not cost value). In this paper we present an abstract syntax of the membrane systems with costs, and then define a structural operational semantics of P systems with costs. We use a rewriting engine called Priced-Timed Maude to implement these P systems. After proving an operational correctness of this implementation, we can analyze properly the evolutions of the P systems involving costs, We look at the cost of computations reaching a given membrane configuration. This paper represents a first step towards a more detailed analysis of various costs in the context of membrane systems.

This class of P systems with costs differs from energy-based P systems [8], a model of membrane systems whose computations occur by manipulating the energy associated to the objects, as well as the free energy units occurring inside the regions of the system. In [8] the energy units are used to transform objects, while in this paper the costs are used only to compute the evolution cost, and eventually to return an optimal evolution with respect to its cost.

2 Membrane Systems with Costs

Before describing in a formal way the evolution of a P system with costs, we present first an inductive definition of the membrane structure, the sets of configurations, and a definition for the corresponding transition systems.

Configurations are states of a transition system, and a computation consists of sequences of transitions between configurations terminating (if it terminates) in a final configuration. A sequence of transition steps represents a *computation*. A computation is successful if this sequence is finite, namely there is no rule applicable to the objects present in the last committed configuration. In a halting committed configuration, the result of a successful computation is the total number of objects present either in the membrane considered as the output membrane, or in the outer region.

In general,operational semantics provides a framework for defining a formal description of a computing system. It is intuitive and flexible, and it becomes more attractive during the years by the developments presented in [12] and [9]. In basic P systems, a computation is regarded as a sequence of parallel applications of rules in various membranes, followed by a communication step and a dissolving step. The operational semantics of the P systems emphasises the deductive nature of the membrane computing by describing the transition steps by using a set of inference rules [3]. The operational semantics of P systems is implemented by using the cost extension of the rewriting system called Maude [7]. The relationship between the operational semantics of P systems and Maude rewriting is given by certain operational correspondence results.

Let O be a finite alphabet of objects over which we consider the *free commutative monoid* O_c^*, whose elements are *multisets*. The empty multiset is denoted by *empty*. Objects can be enclosed in messages together with a target indication. We have *here* messages of typical form $(w, here)$, *out* messages (w, out), and *in* messages (w, in_L). For the sake of simplicity, hereinafter we consider that the messages with the same target indication merge into one message:
$\prod_{i\in I}(v_i, here) = (w, here)$, $\prod_{i\in I}(v_i, in_L) = (w, in_L)$, $\prod_{i\in I}(v_i, out) = (w, out)$, with $w = \prod_{i\in I} v_i$, I a non-empty set, and $(v_i)_{i\in I}$ a family of multisets over O.

We use the mapping rules to associate to a membrane label the set of evolution rules: $\mathsf{rules}(L_i) = R_i$, and the projections L, w and c which return from a membrane its label, its current multiset, and its cost, respectively.

The set $\mathcal{M}(\Pi)$ *of membranes for a P system with costs* Π, and *the membrane structures* are inductively defined as follows:

- if L is a label, c is a cost and w is a multiset over $O \cup (O \times \{here\}) \cup (O \times \{out\}) \cup \{\delta\}$, then $\langle L; c | w \rangle \in \mathcal{M}(\Pi)$; $\langle L; c | w \rangle$ is called *simple (or elementary) membrane*, and it has the structure $\langle\rangle$;
- if L is a label, c is a cost and w is a multiset over $O \cup (O \times \{here\}) \cup (O \times \{in_{L(M_j)} | \, j \in [n]\}) \cup (O \times \{out\}) \cup \{\delta\}$, $M_1, \ldots, M_n \in \mathcal{M}(\Pi)$, $n \geq 1$, where each membrane M_i has the structure μ_i, then $\langle L; c | w; M_1, \ldots, M_n \rangle \in \mathcal{M}(\Pi)$; $\langle\, L\,;c\,|\,w\,;\, M_1, \ldots, M_n\,\rangle$ is called *a composite membrane* having the structure $\langle \mu_1, \ldots, \mu_n \rangle$.

We conventionally suppose the existence of a set of sibling membranes denoted by $NULL$ such that $M, NULL = M = NULL, M$ and $\langle L | w; NULL \rangle = \langle L | w \rangle$. The use of $NULL$ significantly simplifies several definitions and proofs. Let $\mathcal{M}^*(\Pi)$ be the free commutative monoid generated by $\mathcal{M}(\Pi)$ with the operation $(\_, \_)$ and the identity element $NULL$. We define $\mathcal{M}^+(\Pi)$ as the set of elements from $\mathcal{M}^*(\Pi)$ without the identity element. Let M_+, N_+ range over non-empty sets of sibling membranes, M_i over membranes, M_*, N_* range over possibly empty multisets of sibling membranes, and L over labels. The membranes preserve the initial labelling, cost and evolution rules in all subsequent configurations. Therefore in order to describe a membrane we consider its label, its cost and the current multiset of objects together with its structure.

A *configuration* for a P system with costs Π is a skin membrane which has no messages and no dissolving symbol δ, i.e., the multisets of all regions are elements in O_c^*. We denote by $\mathcal{C}(\Pi)$ the set of configurations for Π.

An *intermediate configuration* is an arbitrary skin membrane in which we may find messages or the dissolving symbol δ. We denote by $\mathcal{C}^{\#}(\Pi)$ the set of intermediate configurations. We have $\mathcal{C}(\Pi) \subseteq \mathcal{C}^{\#}(\Pi)$.

Each P system with costs has an *initial configuration* which is characterized by the initial multiset of objects for each membrane and the initial membrane structure of the system. For two configurations C_1 and C_2 of Π, we say that there is a *transition* from C_1 to C_2, and write $C_1 \Rightarrow C_2$, if the following *steps* are executed in the given order:

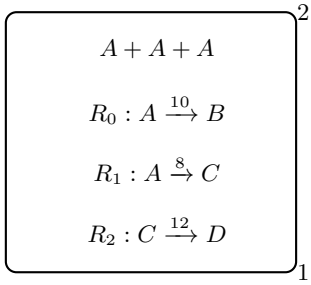
1. *maximal parallel rewriting step* as in [6];

2. *parallel communication of objects through membranes*;
3. *parallel membrane dissolving* of the membranes containing δ.

The last two steps take place only if there are messages or δ symbols resulting from the first step, respectively. If the first step is not possible, then neither are the other two steps; we say that the system has reached a *halting configuration*.

To illustrate these notions we give in Figure 1 a small example containing only of a single membrane labelled by 1 and with associated cost 2. This membrane contains three objects A and three rules with different assigned costs (rule R_0 with cost 10, rule R_1 with cost 8 and rule R_2 with cost 12). A possible evolution leads in just one step to a configuration consisting of three objects B and with the cost of evolution of 30 resulting from applying three times rule R_0.

Fig. 1. A Small P System with Costs



3 Implementing P Systems with Costs by using Maude

Reasoning about the accumulated cost (of energy usage, for instance) during behaviours is crucial in biological and embedded systems (e.g., wireless sensor networks) where minimizing overall consumed resources is critical. Generally, by using a rewriting engine called Maude, a formal specification of a system can be automatically transformed into an interpreter. Moreover, Maude provides an useful new extension called Priced-Timed Maude [4] supporting the formal specification and analysis of systems in which the cost of performing actions plays a significant role. The tool offers a search command, a semi-decision procedure for finding failures of safety properties, and also a model checker. Since the P systems with costs combine the power of parallel rewriting in various locations (compartments), the power of local and contextual evolution and the use of rewriting costs, it is natural to use a rewriting engine and a rewrite theory.

Roughly speaking, a rewrite theory is a triple $(\Sigma, E, \mathcal{R}, \mathcal{L})$, where (Σ, E) is an equational theory used for implementing the deterministic computation, therefore (Σ, E) should be terminating and Church-Rosser, $\mathcal{R}$ is a set of rewrite rules with costs used to implement nondeterministic and/or concurrent computations, and $\mathcal{L}$ is a set of tick rules which can model the time elapse in the system. Therefore we find rewriting logic suitable for implementing these membrane systems.

For simplicity, in this section we consider only costs added to the evolution rules. A P system consists of a maximal parallel application of the evolution rules, the (repeated) steps of internal evolution, communication, and dissolving. This sequence of steps uses a kind of synchronization. A P system has a tree like structure with the skin as its root, the composite membranes as its internal nodes and the elementary membranes as its leaves. The order of the children of a node is not important due to the associativity and commutativity properties of the concatenation operation $\_,\_$ of membranes.

In what follows we extend with costs the operational semantics of membrane systems with promoters, inhibitors [5] and registers [1]. We define an operational semantics of membrane systems by means of three sets of inference rules corresponding to maximal parallel rewriting, sending messages and dissolving. The notation $\mathcal{R} \vdash t \to t'$ is used to express that $t \to t'$ is provable in the theory $\mathcal{R}$ using the inference rules of rewriting logic. We use the syntax of the rewriting engine Maude extended for systems with costs [4] to describe a rewriting theory which corresponds faithfully to the semantics of membrane systems with costs.

In rewriting logic we describe a multiset of objects and messages as consisting of four "bags" of which three are multisets of objects (standing for objects which are actually in the membrane, objects with message *here*, objects with message *out*), and the fourth containing a multiset of pairs of objects and labels i which stand for objects with message in_i. This representation facilitates the rewriting logic specification because in this way there is no need for additional sorts with respect to messages. We first consider the following sorts:

```
sorts Obj ObjMultiset ObjAddressMultiset Label Rule RuleSet .
subsort Obj < ObjMultiset . subsort Rule < RuleSet .
```

By `emptyMO` and `emptyMAO` we denote the empty multiset of objects, respectively of objects with labels, and use `+` to denote the addition on both `ObjMultiset` and `ObjAddressMultiset`.

The multiset of objects with addresses is constructed through the operator
`op in : ObjMultiset Label -> ObjAddressMultiset`.
A rule is constructed through the operator

```
op _->_|_|_|_|_|_ : ObjMultiset ObjMultiset ObjMultiset
ObjAddressMultiset ObjMultiset ObjMultiset Cost -> Rule [ctor] .
```

The first slot is for the objects to be consumed (it is the left hand side of the rule); the second slot is for the objects produced with label "here"; the third slot is for the objects produced with label "out"; the fourth slot is for the objects produced with label "in_child"; the next two slots are for promoters respectively inhibitors. The last slot is used to give the cost of applying the rule. The operators which are used to manipulate the components of a rule are

```
ops lhs rhsHere rhsOut promoter inhibitor : Rule -> ObjMultiset .
op rhsIn : Rule -> ObjAddressMultiset .
op costOf : Rule -> Cost .
```

`rulesIn : Label -> RuleSet` is used to present the rules inside a membrane.

We work with register membranes even when implementing message passing and dissolving. This does not modify in any way the semantics. In what follows, all the membranes are register membranes, even when not explicitly stated.
A register membrane is constructed through the operator

```
op <_‘[_|_|_|_‘]_>_ : Label ObjMultiset ObjMultiset ObjMultiset
ObjAddressMultiset MembraneSet ObjMultiset -> Membrane [ctor] .
```

The first slot is for the label; the second slot is for the objects inside the membrane; the third slot is for the objects with label "here"; the fourth slot is for the objects with label "out"; the fifth slot is for the objects with label "in_child"; the sixth slot is for the set of children membranes; the last slot is for the register. The operators which are used to manipulate the components of a rule are

```
op labelOf : Membrane -> Label .
ops register here : Membrane -> ObjMultiset .
ops content out : MembraneSet -> ObjMultiset .
op inChildren : Membrane -> ObjAddressMultiset .
op children : Membrane -> MembraneSet .
```

Other operators are `_isIn_` which evaluates whether a multiset is contained in another multiset, `mprIrred`, `msgIrred`, `dissIrred`, `eraseDelta`, `emptyOut` and `emptyReg` whose names are self-explaining. We also use `labelsOf` to gather the membrane labels which appear in the right hand side of a rule, for the same purpose `membraneSetLabels` with respect to the membrane sets, and `subsetOf` to compare them. These last three functions are used only when evaluating whether a pair formed of a membrane M and a rule R is valid:

```
op valid : Membrane Rule -> Bool .
ceq valid(M, R) = true if lhs(R) isIn content(M) /\ promoter(R)
  isIn (content(M) + register(M)) /\ labelsOf(rhsIn(R)) subsetOf
  membraneSetLabels(children(M)) /\ if (inhibitor(R) =/= emptyMO)
  then (inhibitor(R) isIn (content(M) + register(M)) == false)
  else true fi .
eq valid(M, R) = false [owise] .
```

To separate the three stages of evolution of a membrane we use four tags:

```
sorts evolutionType State .
ops mpr msg diss end : -> evolutionType [ctor] .
op _;_ : MembraneSet evolutionType -> State [ctor] .
```

where `end` is used to stop the rewriting once the membrane has stopped evolving.

The maximal parallel rewriting of a membrane is given by the following rules, where the second one is executed with the cost of the corresponding rule:

```
crl [1] : M , MM ; mpr => M1 , MM ; mpr if
  MM =/= null /\ M ; mpr => M1 ; mpr /\ M =/= M1 .

crl [2] : < L [ W1 | W2 | W3 | A ] MM > W4 ; mpr =>
  < L [ W1 - lhs(R) | W2 + rhsHere(R) | W3 + rhsOut(R) | A
  + rhsIn(R) ] MM > (W4 + lhs(R) ) ; mpr with cost costOf(R)
```

```
  if mprIrred(MM) /\ R RR := rulesIn(L)
  /\ valid(< L [ W1 | W2 | W3 | A ] MM > W4, R) .

crl [3] : < L [ W1 | W2 | W3 | A ] MM > W4 ; mpr =>
  < L [ W1 | W2 | W3 | A ] MM1> W4 ; mpr if
  mprIrred(MM) == false /\ MM ; mpr => MM1 ; mpr /\ MM =/= MM1 .
```

These rules impose the following evolution: if in a membrane there is some mpr-reducible child membrane, then the membrane is replaced by a similar membrane which has that child rewritten (rules `crl [3]` and `crl [1]`); if a membrane has only mpr-irreducible children, all valid rules are applied one by one (rule `crl [2]`). When even the *skin* membrane is mpr-irreducible, the following rule is applied

```
crl [4] : M ; mpr => emptyReg(M) ; msg if labelOf(M) == 1
  /\ mprIrred(M) .
```

in order to empty the register and to begin the next evolution stage, that of the message sending.

This message sending stage is governed by the following rules:

```
crl [5] : M , MM ; msg => M1 , MM ; msg if
  MM =/= null /\ M ; msg => M1 ; msg /\ M =/= M1 .

crl [6] : < L [ W1 | W2 | W3 | A ] MM > W4 ; msg => if L == 1 then
  < L [ W1 + W2 + out(MM1) | emptyMO | emptyMO | emptyMAO ]
  emptyOut(sendIn(A, MM1)) > W4 ; msg else
  < L [ W1 + W2 + out(MM1) | emptyMO | W3 | emptyMAO ]
  emptyOut(sendIn(A, MM1)) > W4 ; msg fi
  if msgIrred(MM) == false /\ MM ; msg => MM1 ; msg /\ msgIrred(MM1) .

crl [7] : < L [ W1 | W2 | W3 | A ] MM > W4 ; msg => if L == 1 and
  W3 =/= emptyMO then < L [ W1 + W2 + out(MM) | emptyMO
  | emptyMO | emptyMAO ] emptyOut(sendIn(A, MM)) > W4;msg else
  < L [ W1 + W2 + out(MM) | emptyMO | W3 | emptyMAO ]
  emptyOut(sendIn(A, MM)) > W4 ; msg fi if msgIrred(MM)
  /\ (A =/= emptyMAO) or (W2 =/= emptyMO) or out(MM) =/= emptyMO .
```

In this stage a membrane evolves in a single rewriting step: if the set MM of children membranes is msg-reducible, then MM rewrites to a msg-irreducible $MM1$(rule `crl [5]`); the membrane M with objects $W1$ which contains MM is rewritten to the membrane $M1$ with objects $W1 + W2 + out(MM1)$ (i.e. the objects with messages of form $(a, here)$ are transformed in objects of form a, and the objects sent out by the set $MM1$ of membranes are added), and children $emptyOut(sendIn(A, MM1))$ (i.e. the objects of form (a, in_j) are sent into the membrane with label j and then the objects with messages of form (a, out) are erased from every child membrane). The result is msg-irreducible, because the only objects with messages are in the membrane $M1$, and they are of the form (a, out) (if $M1$ is the *skin* not even those objects remain). If the set MM of children membranes is msg-irreducible, then the same process takes place, except that instead of $MM1$ it is still MM(rule [7]).

Rules `crl [5]`, `crl [6]` and `crl [7]` correspond to inference rules $msg1$ and $msg2$. In defining the transition relation T_{msg} we treat the case of an elementary membrane separately, since we prefer to avoid extending T_{msg} to sets of membranes. Although rules `crl [6]` and `crl [7]` look almost identical, we cannot include them in a single rule with the conditional part `if MM ; msg => MM1 ; msg` because it would lead to an infinite loop of identical rewritings. This happens because $MM; msg \rightarrow MM; msg$ is provable in rewriting logic.

When the entire membrane system is msg-irreducible, the rule

```
crl [8] : M ; msg => M ; diss if labelOf(M) == 1 /\ msgIrred(M) .
```

is applied. This rule starts the next evolution stage, that of dissolving.

The rules for dissolving membranes are:

```
crl [9] : M , MM ; diss => M1 , MM ; diss if
  MM =/= null /\ M ; diss => M1 ; diss /\ M =/= M1 .

crl [10] : < L [ W1 | W2 | W3 | A ] MM > W4 ; diss =>
  < L [ W1 + eraseDelta(content(M)) | W2 | W3 | A ]
  children(M) , MM1 > W4 ; diss if dissIrred(MM) /\ M , MM2 := MM
  /\ delta isIn content(M) /\ MM1 := children(M) , MM2 .

crl [11] : < L [ W1 | W2 | W3 | A ] MM > W4 ; diss =>
  < L [ W1 | W2 | W3 | A ] MM1 > W4 ; diss
  if dissIrred(MM) == false /\ MM ; diss => MM1 ; diss
  /\ dissIrred(MM1) .
```

If the set MM of children membranes for a membrane M is diss-reducible and it rewrites to a diss-irreducible set of membranes $MM1$, then M is rewritten to the similar membrane $M1$ which has children membranes $MM1$ (rules `crl [9]` and `crl [11]`). When the set MM of children membranes is diss-irreducible and at least one of the membranes in MM contains the special symbol δ, then all the membranes from MM which contain δ are dissolved (rule `crl [10]`). Note that a top membrane M does not dissolve even when it does contain δ. This happens because the rewriting rules are given with the purpose of describing the evolution of the *skin* membrane, which can never dissolve. Rules `crl [9]`, `crl [10]` and `crl [11]` correspond to inference rules $msg1$ and $msg2$. Again, we have used the first rule in this group as a stepping stone towards the rewriting of a set of sibling membranes, while avoiding to include the rewriting of a set of sibling membranes in the transition relation T_{diss}.

When the *skin* membrane is diss-irreducible but is mpr-reducible, the rule

```
crl [12] : M ; diss => M ; mpr if labelOf(M) == 1
  /\ dissIrred(M) /\ mprIrred(M) == false .
```

is applied; it starts once more the maximal parallel rewriting stage of the evolution. However, if the *skin* membrane is also mpr-irreducible, rule

```
crl [13] : M ; diss => M ; end if   labelOf(M)==1
  /\dissIrred(M)/\mprIrred(M).
```

is applied; in this case it ends the rewriting. We do not need to evaluate the msg-irreducibility of the *skin* membrane, because the dissolving stage can only be reached by msg-irreducible membranes.

The correspondence between the operational semantics given by the transition relation $\Rightarrow$ on one hand, and the rewriting logic implementation on the other hand is is given by a mapping $\psi : \Pi \rightarrow \mathsf{State}$ defined by the natural encoding presented above. By $\mathcal{R}_\diamond$ we denote the rewrite theory defined by the rewrite rules `[1]` ... `[13]` together with the operators and equations defining them. The next theorem emphasizes the correspondence between the dynamics of the membrane systems with costs and the rewrite theory.

Theorem 1. $M \stackrel{c}{\Rightarrow} N$ *iff* $\mathcal{R}_D \vdash \psi(M) \Rightarrow^* \psi(N)$ *with cost* c.

4 Analyzing and Verifying P Systems With Costs

Using the previous operational correspondence provided by Theorem 1, the software experiments done in Priced-Timed Maude reflect exactly the evolution of the encoded membrane systems with costs. In this section we use a simple example of a membrane system with costs, example that is described in rewriting logic in the following form:

```
eq R0 = A -> B | emptyMO | emptyMAO | emptyMO | emptyMO | 10 .
eq R1 = A -> C | emptyMO | emptyMAO | emptyMO | emptyMO | 8 .
eq R2 = C -> D | emptyMO | emptyMAO | emptyMO | emptyMO | 12 .
eq Q = < 1 [ A + A + A | emptyMO | emptyMO | emptyMAO ] null > emptyMO .
eq rulesIn(1) = R0 R1 R2 .
eq S = Q ; mpr .
```

When entering the rewrite command

```
(ptfrew {S} in time <= 0 with cost <= 70 .)
```

Maude presents the following output:

```
Result PricedTimedSystem :
 {< 1[B + B + B | emptyMO | emptyMO | emptyMAO]null > emptyMO ; end}
    in time 0 with cost 30
```

We use priced-time Maude to check if certain configurations of a system can be reached (reachability problem).

```
(ptsearch {S} =>* {X:StateStop} with no limits .)
```

We use the *ptsearch* command to answer the question: starting from the initial membrane system S, what are the reachable final states (the ones containing the *end* tag)? This is done by searching for states which match a corresponding pattern. In this example, we use the $=> *$ symbol, meaning that we are searching for several steps. If one is interested in a bounded number of reachable final states, the command $ptsearch[n]$ can be used to obtain systems reachable in n steps. In our case, the output is

```
Priced-timed search in EXAMPLE
    {S} =>* {X:StateStop}
with no time or cost limit and with mode default time increase 10 :

Solution 1
TIME_ELAPSED:Time --> 0 ; TOTAL_COST_INCURRED:Cost --> 30 ;
  X:StateStop --> < 1[B + B + B | emptyMO | emptyMO | emptyMAO]null
  > emptyMO ; end

Solution 2
TIME_ELAPSED:Time --> 0 ; TOTAL_COST_INCURRED:Cost --> 40 ;
  X:StateStop --> < 1[B + B + D | emptyMO | emptyMO | emptyMAO]null
  > emptyMO ; end

Solution 3
TIME_ELAPSED:Time --> 0 ; TOTAL_COST_INCURRED:Cost --> 50
  X:StateStop --> < 1[B + D + D | emptyMO | emptyMO | emptyMAO]null
  > emptyMO ; end

Solution 4
TIME_ELAPSED:Time --> 0 ; TOTAL_COST_INCURRED:Cost --> 60 ;
  X:StateStop --> < 1[D + D + D | emptyMO | emptyMO | emptyMAO]null
  > emptyMO ; end

No more solutions
```

It should be noticed that after only two steps, the cost of the reachable configurations is very different depending on the rules applied.

In addition to these commands, Priced-Timed Maude allows to find optimal results such as the earliest state matching a pattern, as well as the cheapest evolution to reach a given configuration. In our case, the earliest reachable states containing the evolution type `end` can be found using the following command

```
(priced find earliest { Q ; mpr} =>* {X:MembraneSet ; end}
   with no cost limit .)
```

that returns the result:

```
Priced find earliest {X:MembraneSet ; end} in EXAMPLE such that
   {Q ; mpr} =>* {X:MembraneSet ; end}
with no cost limit with mode default time increase 10 :

Result: {< 1[B + B + B | emptyMO | emptyMO | emptyMAO]null >
   emptyMO ; end} in time 0 with cost 30
```

Using the command `find cheapest`, it is possible to detect the cheapest evolution (as cost) to reach a given configuration.

```
(find cheapest { Q ; diss} =>* {X:MembraneSet ; mpr}
   with no time limit . )
```

This command verifies that indeed reaching a configuration ready to apply maximal from a configuration ready to apply dissolution rules takes 10 time units with cost 0.

```
Find cheapest in EXAMPLE
   {Q ; diss} =>* {X:MembraneSet ; mpr}
with no time limit time and with mode default time increase 10 :

Solution
TIME_ELAPSED:Time --> 10 ; TOTAL_COST_INCURRED:Cost --> 0 ;
   X:MembraneSet
   --> < 1[A + A + A | emptyMO | emptyMO | emptyMAO]null > emptyMO
```

5 Conclusions and Future Work

We defined P systems with costs by assigning storage costs to membranes, as well as and execution costs to rules. We used the Priced-Timed Maude rewriting engine to implement these P systems with costs. By using such a rewriting engine corresponding to the semantics of membrane systems with costs, we proved the operational correctness of this implementation. Based on such an operational correspondence, we can analyze the P systems with costs and verified several interesting properties.

As a future work we plan to deal with Cost Problems in the framework of membrane systems by considering two variants of the cost problem, namely the Cost-Threshold Problem (can we obtain an evolution cost under a certain threshold value) and the Cost-Optimality Problem (compute the minimal evolution cost). We also intend to study how different evolution strategies influence the computed cost of reaching a desired configuration.

References

1. O. Agrigoroaiei, G. Ciobanu. Rewriting Logic Specification of Membrane Systems with Promoters and Inhibitors. *Electronic Notes in Theoretical Computer Science* **238**(3), 5–22 (2009).
2. B. Aman, G. Ciobanu. Time Delays in Membrane Systems and Petri Nets. *Electronic Proceeding in Theoretical Computer Science* **57**, 47–60 (2011).
3. O. Andrei, G. Ciobanu, D. Lucanu. A Rewriting Logic Framework for Operational Semantics of Membrane Systems. *Theoretical Computer Science* **373**, 163–181 (2007).
4. L. Bendiksen, P.C. Ölveczky. The Priced-Timed Maude Tool. *Lecture Notes in Computer Science* **5728**, 443–448 (2009).
5. P. Bottoni, C. Martín-Vide, Gh. Paun, G. Rozenberg. Membrane Systems With Promoters/Inhibitors. *Acta Informatica* **38**, 695–720 (2002).
6. G. Ciobanu, S. Marcus, Gh. Păun. New Strategies of Using the Rules of a P System in a Maximal Way: Power and Complexity. *Romanian Journal of Information Science and Technology* **12**(1), 157–173 (2009).
7. M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, J.F. Quesada. Maude: Specification and Programming in Rewriting Logic. *Theoretical Computer Science* **285**, 187–243 (2002).
8. A. Leporati, C. Zandron, G. Mauri. Simulating the Fredkin Gate with Energy-based P Systems. *Journal of Universal Computer Science* **10**(5), 600–619 (2004).

9. R. Milner. Operational and Algebraic Semantics of Concurrent Processes. *Handbook of Theoretical Computer Science* **B**, 1201–1242, Elsevier (1990).
10. Gh. Păun. Computing with membranes. *Journal of Computer and System Sciences* **61**, 108–143 (2000).
11. Gh. Păun. *Membrane Computing. An Introduction.* Springer (2002).
12. G. Plotkin. Structural Operational Semantics. *Journal of Logic and Algebraic Programming* **60**, 17–140 (2004).