Active membrane systems without charges and using only symmetric elementary division characterise P

Niall Murphy[1] and Damien Woods[2]

[1]Department of Computer Science, NUI Maynooth, Ireland.
Funded by the Irish Research Council for Science Engineering and Technology

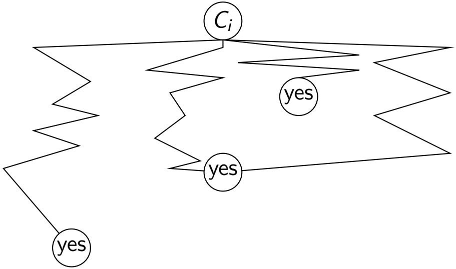
[2]Cork Complex Systems Group, CS Dept, University College Cork, Ireland.
Funded by the Science Foundation of Ireland

June 25, 2007

Recogniser membrane systems

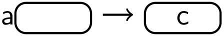
- Decide language problems
- Non-deterministic
- Maximally parallel
- (Semi-)Uniform by polynomial time deterministic Turing machines
- Have an *input* membrane
- Produce the correct `yes` or `no` response to the given problem in polynomial time
- Computation is confluent

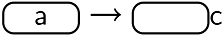
Confluence, all computation paths are valid



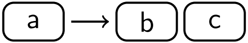
The evolution rules of active membranes with out charges

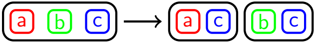
- Object evolution, type (a), 

- Communication in, type (b), 

- Communication out, type (c), 

- Dissolution, type (d), 

- Elementary division, type (e), 

- Non-elementary division, type (f), 

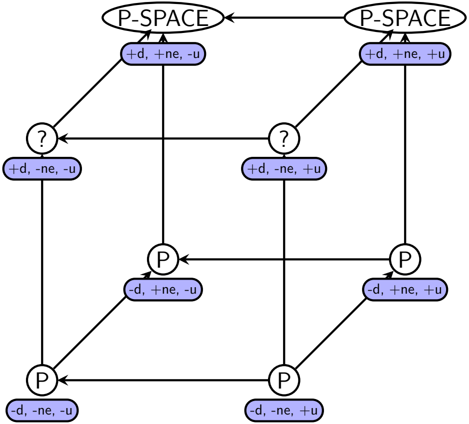
Symmetric and Asymmetric division

Rules of type (e) [a] $\longrightarrow$ [b] [c]

Rules of type (e_s) [a] $\longrightarrow$ [b] [b]

$\mathbf{PMC}^{S}_{\mathcal{EAM}^0_{-a}}$

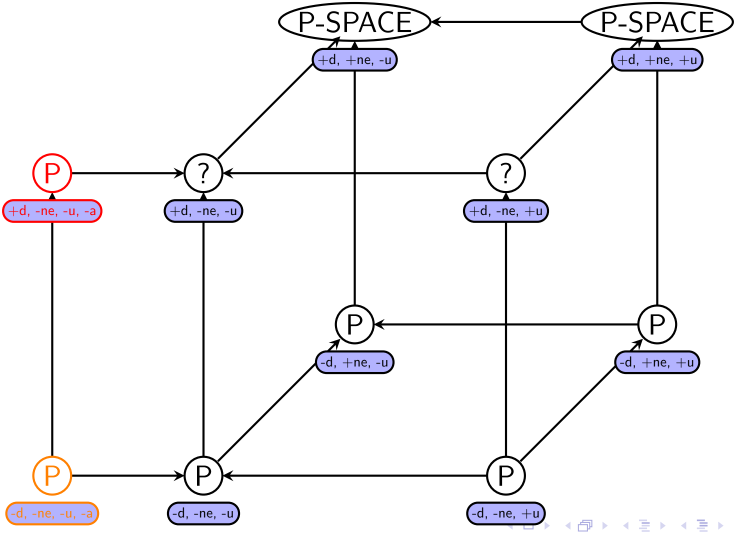
What is currently known



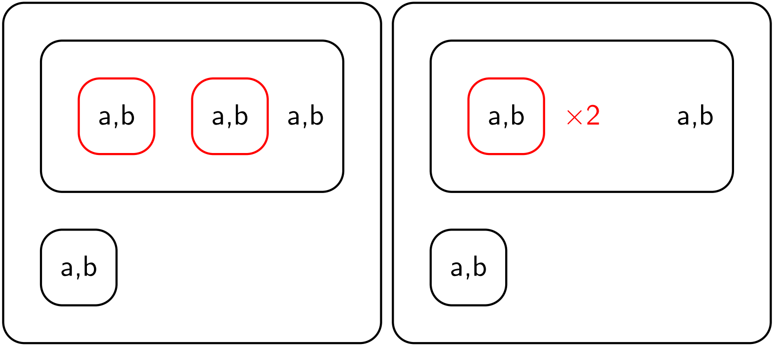
Artiom Alhazov and Mario Pérez-Jiménez. 2006

Miguel Gutiérrez-Naranjo, Mario Pérez-Jiménez, Agustín Riscos-Núñez, and Francisco Romero-Campero. 2006

Our result


P-SPACE
P-SPACE
+d, +ne, -u
+d, +ne, +u
P
?
?
+d, -ne, -u, -a
+d, -ne, -u
+d, -ne, +u
P
P
-d, +ne, -u
-d, +ne, +u
P
P
P
-d, -ne, -u, -a
-d, -ne, -u
-d, -ne, +u

We define an *equivalence class* in membrane systems

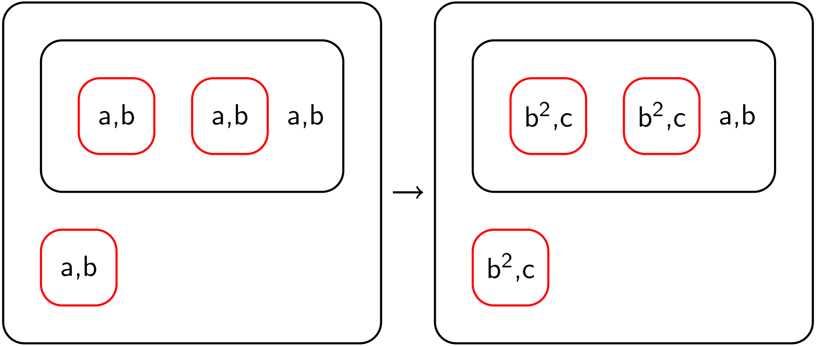


- Membranes with the same contents
- Membranes with the same parent

We have a deterministic simulation

- We sort rules, equivalence classes and objects
- Simulation is the same every time
- Confluence!

Object evolution rules, $[a] \rightarrow [b, c]$

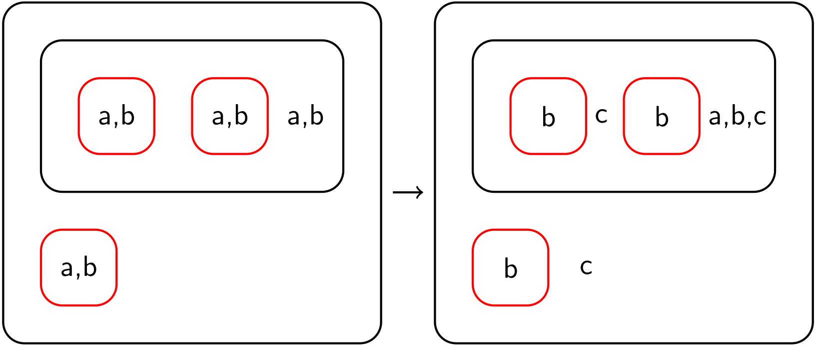


EC's at t_i: 4

EC's at t_{i+1}: 4

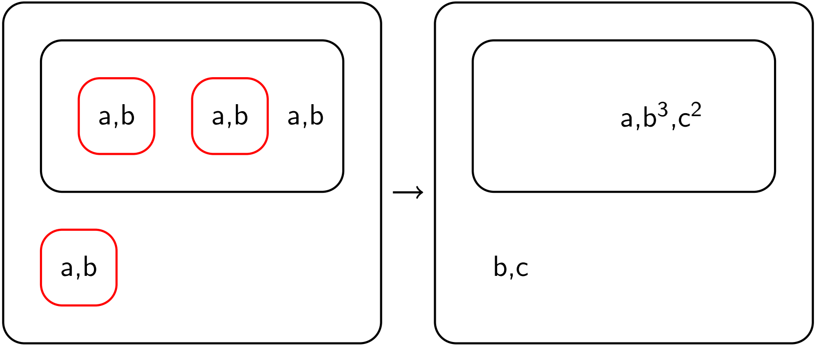
Increase: 0

Communication out rules, $[a] \rightarrow [\,]c$



EC's at t_i: 4
EC's at t_{i+1}: 4
Increase: 0

Dissolution rules, $[a] \longrightarrow c$

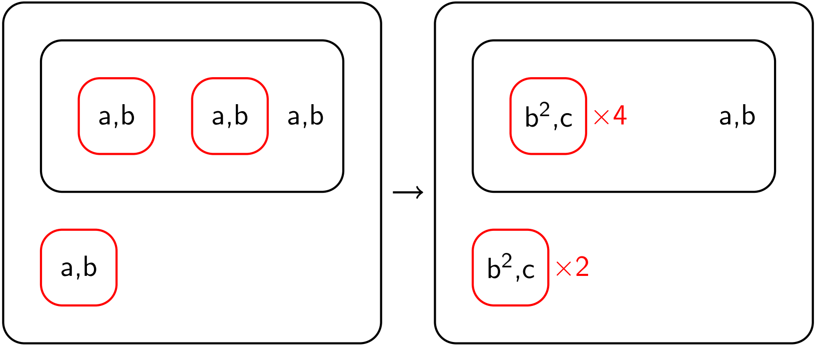


EC's at t_i: 4

EC's at t_{i+1}: 2

Increase: 0

Symmetric elementary division rules, $[a] \rightarrow [b][b]$

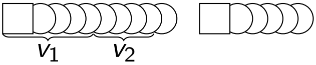


EC's at t_i: 4

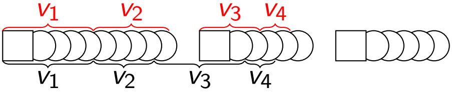
EC's at t_{i+1}: 4

Increase: 0

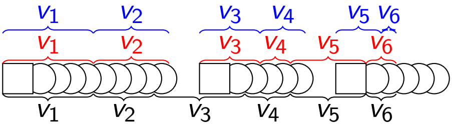
For (*b*) rules in the case $\mathbb{V} < \mathbb{M}$, the increase is $|V|$



EC's at t_i: 2
EC's at t_{i+1}: 4
Increase: 2

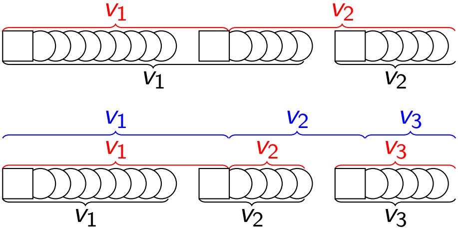


EC's at t_i: 3
EC's at t_{i+1}: 7 or 6
Increase: 4 or 3



EC's at t_i: 3
EC's at t_{i+1}: 9 or 8 or 7
Increase: 6 or 5 or 4

For (b) rules in the case $\mathbb{V} \geq \mathbb{M}$, the increase is $|V| - 1$

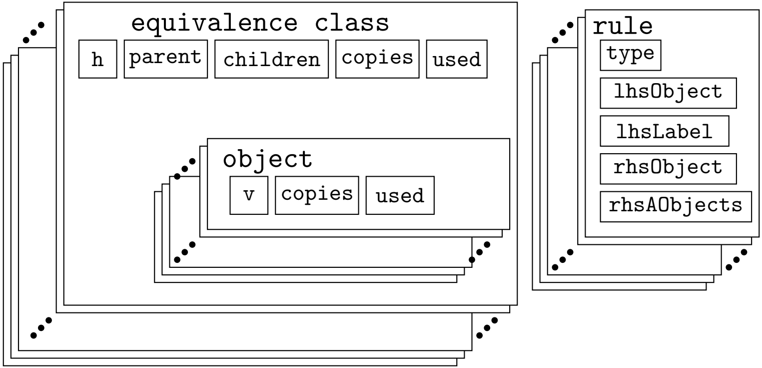


EC's at t_i: 2
EC's at t_{i+1}: 2
Increase: 0

EC's at t_i: 3
EC's at t_{i+1}: 3 or 4
Increase: 0 or 1

EC's at t_i: 3
EC's at t_{i+1}: 3 or 4 or 5
Increase: 0 or 1 or 2

The register structures



Loop algorithm

```
             Input: equivalence_class
             Output: equivalence_class after one timestep of computation
             b_rules ← ∅;
             b_ecs ← ∅;
             b_objs ← ∅;
O(|R|)       forall rule in sortedRules do
                 if rule.label matches equivalence_class.label and rule is not type (b) then
O(|V|)               forall object in sortedObjects do
                         if not all copies of object have been used then
                             if rule is type (a) then
O(|V|)                           Apply_a_rule(equivalence_class, object, rule);
                             else if rule is type (c) then
O(1)                             Apply_c_rule(equivalence_class, object, rule);
                             else if rule is type (d) then
O(|V|)                           Apply_d_rule(equivalence_class, object, rule);
                             else if rule is type (e_s) then
O(1)                             Apply_e_rule(equivalence_class, object, rule);
                             end
                         end
                     end
                 end
                 if rule is type (b) then
O(|E|)               forall child_ec in equivalence_class do
                         if child_ec.label = rule.lhsLabel and object.used ≥ 1 then
                             append child_ec to b_ecs ;
                             append object to b_objs ;
O(|V||E|)                    Apply_b_rule(b_ecs, b_objs, rule)
                         end
                     end
                 end
             end
O(|V| × |E|) reset all used counters to 0;
```

Function ApplyRules(*equivalence_class*) Applies all applicable rules for an equivalence class for one timestep

Algorithm for (b) rules

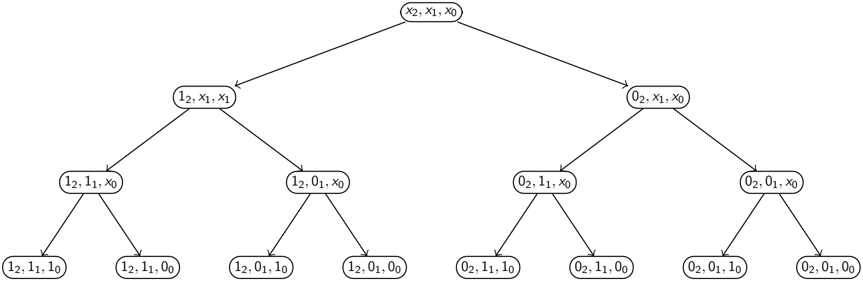
```
Input: membrane
Output: membrane after (b) rules have been applied
b_objects_sorted ← sort(b_objects);
b_equivalence_classes_sorted ← sort(b_equivalence_classes);
O(|V|)  forall object in b_objects_sorted do
O(|E|)      forall equivalence_class in b_equivalence_classes_sorted do
                if object.multiplicity < equivalence_class.multiplicity then
                    copy equivalence_class to new_equiv_class ;
                    subtract object.multiplicity from new_equiv_class.multiplicity ;
                    equivalence_class.multiplicity ← object.multiplicity ;
                    equivalence_class.used ← equivalence_class.multiplicity ;
                    increment equivalence_class.object.multiplicity ;
                    increment equivalence_class.object.used ;
                end
                else if object.multiplicity ≥ equivalence_class.multiplicity then
                    increment equivalence_class.object.multiplicity ;
                    increment equivalence_class.object.used ;
                    equivalence_class.used ← equivalence_class.multiplicity ;
                    subtract equivalence_class.multiplicity from object.multiplicity ;
                end
            end
        end
```

Function Apply_b_rules(*b_equivalence_classes*, *b_objects*, *b_rules*). Total time complexity $O(|V||E|)$.

Total simulation time

$$O(t|R||E|^2|V|)$$

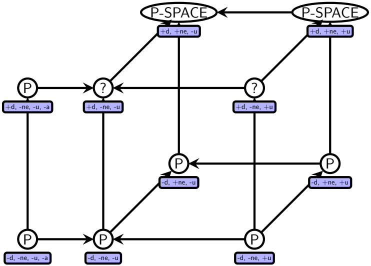
Asymmetric elementary division can create an exponential number of equivalence classes



$$[x_i] \rightarrow [1_i]\,[0_i]$$

The result

- $\mathbf{PMC}^{S}_{\mathcal{EAM}^0_{-a}}$ allows dissolution rules but has a **P** upper bound.
- We simulate in polynomial time a model which uses exponential numbers of membranes and objects



Future work

- Upper bound or lower bound on the asymmetric case
- Symmetric non-elementary division
- log space uniformity