Concurrent Extensions of Membrane Computing
SRTP Project Proposal

Hao Bai, Arko Dey Kowshik, Keyi Shen

Zhejiang University, UIUC Institute

1 Project Description

The goal of the project is to investigate extensions to the model of P-systems used for the processing in membrane computing. The envisioned extensions are threefold: (1) turn P-systems into concurrent systems, i.e. assume that processing of rules can be handled autonomously and independent from other rules; (2) turn P-systems into ambient systems depending on their environment and permit changes to the processing rules; (3) leverage the notion of state of a super-cell to include structure rather than only strings.

1.1 Background

Natural computing is an umbrella for various computing paradigms taking processes in nature as models [8]. It captures neural networks, genetic programming, DNA computing and membrane computing [6]. Membrane computing exploits hierarchically nested membrane structures as in cells, each associated with multisets of objects [7]. These associations are manipulated by so-called P-systems, which associate rules with each membrane. Rules are executed in parallel leading to the creation, elimination and wandering of objects through membranes as well as the manipulation of the membrane structure as such [4]. As elaborated in [6] membrane computing has been shown to be useful for many applications, in particular in the context of bio-medical systems [5].

A formal specification of membranes and P-systems has been sketched in [10] using Abstract State Machines [3]. This has led to several open questions concerning limitations of the computation model with respect to concurrency, adaptivity and the used notion of super-cells. While a maximal fireable set of rules is used in P-systems, it appears more natural to assume that the different membranes act more independently and the rule sets are applied in an asynchronous way. In particular, if we think about the rules expressing chemical reactions on molecules as objects, such asynchronous behaviour much better the time-consuming nature of such processes. Foundations of asynchronous concurrent computation have been handled by concurrent ASMs in [2].

Concerning adaptivity, it is already known that membrane behaviour should depend on ambiences. This suggest to consider also ambient ASMs [1] in an extension of the formal model. More generally, as super-cells in membrane computing change, it even seems possible to change the behaviour as well, i.e. to change the rules. This lead to linguistic reflection, which is formally covered by reflective ASMs [9]. Reflection allows a machine to modify its self representation in every step, and to use the amended behaviour specification in the next step.

1.2 Objectives

The objectives of the project are as follows:

O1. Extend P-systems into concurrent systems thereby allowing time-consuming behaviour of rules.
O2. Extend P-systems to become ambient and reflective, i.e. make rules dependent on the ambient environment and subject to change by rules.
O3. Extend the notion of super-cell in membrane computing by considering arbitrary Tarski structures in membranes rather than just multisets of characters taken from a fixed, finite alphabet.

1.3 Methodology

The project will approach the three envisioned extensions one-by-one. In all cases the starting point is the formalisation of membranes and P-systems by ASMs in [10]. In a P-systems a rule is a pair $\ell \to r$, where the stimulator ℓ on the left-hand side is just a multiset of objects $o \in O$ using a fixed alphabet O, and the action r on the right-hand side is a multiset of object/location pairs or a specific *dissolve* action. In addition, there exists a priority order on the set of rules. A P system selects a maximal set of rules that can fire simultaneously, then removes ℓ and adds r for each of the selected rules.

As the processing is strictly sequential, it suggests an asynchronous alternative based on concurrent ASMs [2]. For this we can group rules and associate abstract agents with these groups. As rule execution involves the removal of objects from membranes, while their presence indicates that another rule may still fire, it will be necessary to add a blocking mechanism. We will provide a modified concurrent ASM specification for this and approach an implementation of asynchronous P-systems together with an illustration by meaningful examples.

Concerning adaptation we proceed analogously, this time extending the specification by ambients and reflection. It will not be necessary to exploit the tree algebra for reflective machines, as rules are already part of the specification of P-systems, and the excution engine refers to these rules. Finally, our extension will modify the definition of super-cell using signatures and structures defined over them rather than just multisets of objects. Then the left hand side of a rule can be replaced by an arbitrary condition to be evaluated on a membrane in a generalised super-cell, and the action part can become an almost arbitrary ASM rule subject to the restriction that updates can only be local, membranes can at best be created inside the given membrane or dissolved, and the blocking of other rules is preserved. Again, we will provide a modified specification for this, an implementation of extended P-systems, and an illustration by meaningful examples.

2 Project Team

Project Members:	Bai Hao	Sophomore student of Electrical and Computer Engineering email: Haob.19@intl.zju.edu.cn
	Kowshik, Arko Dey	Junior student of Electrical and Computer Engineering email: Arko.18@intl.zju.edu.cn
	Keyi Shen	Sophomore student of Electrical and Computer Engineering email: Keyi.19@intl.zju.edu.cn
Supervisor:	Schewe, Klaus-Dieter	Professor, Zhejiang University – UIUC Institute email: kd.schewe@intl.zju.edu.cn University id: H219026

All team members are knowledgeable in software systems design and algorithms, which have been handled in the courses on Discrete Mathematics and Data Structures. They are also skilled in programming.

3 Pseudo Code

Algorithm 1: Algorithm of Data Enhancement

```
   Data: 3 pieces of data_3d_std of size [n, x, 32, 3(x, y, z)]
   Result: 1 piece of data_3d_std of size [n, x, 32, 3(x, y, k)]
   // random functioning
 1 data_3d_std = random_translation(data_3d_std);
 2 data_3d_std = random_rotation(data_3d_std);
   // collision elimination
 3 data_3d_std = collision_elimination_std(data_3d_std);
   // camera generation
 4 camera_std, intrinsic_matrix = camera_generator();
   // cento
 5 data_3d_std = transform(camera_std, intrinsic_matrix,
    data_3d_std)
 6 return 1 piece of data_3d_std;
```

Algorithm 2: Algorithm of Random Functioning

```
   Data: 3 pieces of data_3d_std
   Result: 1 piece of data_3d_std
   // randomly shift the timeline
 1 set frame as a shift value;
 2 array = array[ frame : ];
   // randomly translationally move the 3 pieces of
      data_3d_std
 3 set array as a random-shift-value matrix;
   // randomly rotationally move the 3 pieces of
      data_3d_std
 4 if user choose to add randomly rotational movements then
 5 |   find the random rotation_matrix;
 6 else
 7 |   return 0;
 8 end
   // apply changes and combine the 3 pieces of
      data_3d_std
 9 apply array, rotation_matrix to data_3d_std;
10 combine 3 pieces of data_3d_std;
11 return 0;
```

Algorithm 3: Sequential Approach For Eliminating Individual Collisions (with the frame-skipping method)

```
Data: Tensor of shape $(n, x, 17, 3)$ where there are $n$ people, each
      with $x$ frames and 17 vertices (each of which contains 3
      correponds).
Result: Tensor of shape $(n, x, 17, 3)$, where each person has an
        own shift vector through all the frames.
for i in n do
    skip the first person;
    for frame in x (step 3) do
        if the frame has collision with any other person then
            shift_vector_list(for i) append find_shift_vector();
        else
            shift_vector_list(for i) append $(0, 0, 0)$;
        end
    end
    assign max_shift_vector(for i) =
        find_the_maximum_shift_vector(shift_vector_list(for i));
    broadcast add all frames in x of i +=
        max_shift_vector(for i)
end
return the new tensor;
```

Algorithm 4: Find the shift vector of the $frame$ on person i

```
Data: Person $i$, given $frame$, 17 vertices, each with 2
      corresponds
Result: The shift vector for this frame
shift_vector = (0, 0, 0);
while True do
    flag_collision = False;
    for another_person in $(i - 1)$ do
        if is_overlapped(i, another_person) then
            flag_collision = True;
            shift_vector += decide_offset_of_i(i, another_person);
            preview applying shift_vector to data_3d_std[i];
            continue;
        end
    end
    if flag_collision is False then
        break;
    end
end
return shift_vector;
```

References

1. E. Börger, A. Cisternino et V. Gervasi : Ambient abstract state machines with applications. *J. Comput. Syst. Sci.*, 78(3):939–959, 2012.
2. E. Börger et K.-D. Schewe : Concurrent abstract state machines. *Acta Informaticae*, 53(5):469–492, 2016.
3. E. Börger et R. Stärk : *Abstract State Machines.* Springer-Verlag, Berlin Heidelberg New York, 2003.
4. C. Martín-Vide, G. Păun et A. Rodríguez-Patón : On P systems with membrane creation. *The Computer Science Journal of Moldova*, 9(2):134–145, 2001.
5. I. Petre et L. Petre : Mobile ambients and P-systems. *J. UCS*, 5(9):588–598, 1999.
6. G. Păun : *Membrane Computing: An Introduction.* Natural computing series. Springer, 2002.
7. G. Păun : Membrane computing. *In* R. A. Meyers, éd. : *Encyclopedia of Complexity and Systems Science*, p. 5523–5535. Springer, 2009.
8. G. Păun, G. Rozenberg et A. Salomaa : *DNA Computing - New Computing Paradigms.* Texts in Theoretical Computer Science. An EATCS Series. Springer, 1998.
9. K.-D. Schewe : Concurrent reflective Abstract State Machines. *In* T. Jebelean *et al.*, éds : *19th Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC 2017)*, p. 30–35. IEEE, 2018.

10. K.-D. Schewe, L. Tec et Q. Wang : Capturing membrane computing by asms. *In* M. J. Butler *et al.*, éds : *6th International Conference on Abstract State Machines, Alloy, B, TLA, VDM, and Z (ABZ 2018)*, vol. 10817 de *Lecture Notes in Computer Science*, p. 380–385. Springer, 2018.