Open Problems

B. Aman, G. Ciobanu
Institute of Computer Science, Romanian Academy
Bld. Carol I, no.8, 700505, Iaşi Branch, România

1 Type Systems in Membrane Computing

Type theory is fundamental both in logic and computer science. In computer science, type theory refers to the design, analysis and study of type systems. Generally, a type system is used to prevent the occurrences of errors during the evolution of a system. A type inference procedure determines the minimal requirements to accept a system or a component as well-typed.

Membrane systems consider cells as mechanisms working in a maximal parallel and non-deterministic manner. However, the living cells do not work in such a way: a chemical reaction takes place only if certain quantitative constraints are fulfilled. In order to cope with such constraints, membrane systems should be enriched by adding a quantitative type discipline, and making use of type inference and principal typing [2]. We associate to each reduction rule a minimal set of constraints that must be satisfied in order to assure that by the application of this rule to a well-formed membrane system, we get a well-formed membrane system as well. A first step in this direction was done in [1] where a type system for membrane system with symport/antiport rules is given.

Some restrictions can be imposed to membrane systems using type systems:

- limiting the volume of cell/number of objects present in a membrane;
- limiting the types of objects that are allowed to stay in/on a membrane;
- limiting the amount of energy a neuron can receive.

Other restrictions can be designed for specific classes of membrane systems.

2 Verification of Some Behavioural Properties

There are numerous decidability problems that have to be answered when using membrane systems for modelling. In what follows we present some of them.

Reachability is the problem of deciding whether a system can reach a given configuration during its evolution. This problem is useful in the automated verification of systems, by checking if a system can reach an undesired state. It also has the property that many other problems can be reduced to it. When

a biological system is modelled using membrane systems, reachability is also useful. Assume we are modelling a virus attack on an organism. One can be interested in knowing if, during evolution, the virus successfully infects some parts of the organism, namely if the membrane system model reaches some undesired configuration.

Boundedness is a property of systems whose resources may be bounded (production and consumption are limited during a finite period of time). From a biological point of view, boundedness can be interpreted as a storage limitation: e.g., in a cell only a finite amount of chemical components are allowed to be stored since the cell cannot accumulate more than a finite amount of material.

A liveness property asserts that a system always progresses: e.g., if the system requests a resource, then its request is eventually granted. More precisely, when proving the liveness property of a biological system, we check that the time progresses from each reachable configuration. A dead marking is a marking in which no transitions are enabled.

As a consequence, tools and techniques developed for membrane systems should provide beside description and analysis, also automated verification of behavioural properties (qualitative properties (reachability) and quantitative properties (boundedness, liveness)) of membrane systems, and in particular for the investigation of the ongoing behaviour of membrane systems. This would complement and broaden the standard approach to the analysis of membrane systems which concentrates primarily on the ultimate results of "successful" or halting computations of membrane systems, and on the computational power (including aspects of complexity) of different variants of the main model.

References

[1] B. Aman, G. Ciobanu. Typed Membrane Systems. *Lecture Notes in Computer Science*, vol. 5957, 169–181, 2010.

[2] J. Wells. The Essence of Principal Typings. *Lecture Notes in Computer Science*, vol.2380, Springer, 913–925, 2002.