High Performance Simulations of Kernel P Systems

Mehmet E. Bakir, Savas Konur, Marian Gheorghe
Department of Computer Science
University of Sheffield
Sheffield, UK
Email: mebakir1@sheffield.ac.uk

Ionut Niculescu, Florentin Ipate
Department of Computer Science
University of Bucharest
Bucharest, Romania
Email: ionutmihainiculescu@gmail.com

***Abstract*—The paper presents the use of a membrane computing model for specifying a synthetic biology pulse generator example and discusses some simulation results produced by the tools associated with this model and compare their performances. The results show the potential of the simulation approach over the other analysis tools like model checkers.**

I. Introduction

Nature inspired computing, including biological and chemical (or molecular) computing, has become a very intensively investigated research area, with a consistent body of theoretical research, and with many interesting applications and challenging problems [1]. One of the most recently introduced area of natural computing is *membrane computing*. This has been conceived as a computational paradigm inspired by the structure and behaviour of the living cell [2]. Many models have been considered and studied, and substantial theoretical results related to the computational power and complexity aspects have been obtained [3]. These models are called *membrane systems* or *P systems*. Interesting applications in systems and synthetic biology have been provided, using a set of methods and tools based on this computational model [4]. In the last years there have been attempts to create more general membrane computing models, which allow the specification of various classes of problems defined with different membrane computing models and providing mechanisms to analyse such systems and simulate their behaviour. One such model, called *kernel P systems* [5], [6], has been recently introduced and some tools for the simulation and verification of the systems specified with this formalism have been built.

In this paper, we present the use of kernel P systems for specifying a synthetic biology pulse generator example and discuss some simulation results produced by the tools associated with this model and compare their performances. The results show the potential of the simulation approach over the other analysis tools like model checkers, which heavily suffer from the well-known *state explosion* problem.

In Section II, we very briefly describe kernel P systems. In Section III, we present the simulation frameworks, supporting the simulation of kernel P system models. Section IV describes the synthetic pulse generator, and presents the experimental results. Section V draws some conclusions and provides some future research directions.

II. Kernel P Systems

Kernel P systems (kP systems for short) are multiset transformation mechanisms consisting of *compartments* linked by some communication channels; each compartment contains *multisets of objects* and *rewriting and communication rules* which transform the multisets of objects and send them to neighbour compartments. The system evolves in steps and at each step, in each compartment the rules are applied in accordance with a certain execution strategy. For the example considered in this paper, in each compartment a rule is executed per step, non-deterministically chosen from those applicable at that moment. The system starts having in each compartment some initial multiset of objects. A formal definition of these models is available from [5], [6].

Kernel P systems are supported by a software framework, called KPWorkbench [7], [8], which integrates a set of tools enabling simulation and model checking of kP systems (e.g. genetic Boolean gates [9]). The KPWorkbench tool implements several translations that connect several target specifications employed for kP system models. In [7], the model checker Spin is utilised in order to verify properties of the kP system models. The KPWorkbench also consists of a native simulator which allows the execution of the models written with the kP system formalism. Recently a translator of kP system models to Flame has been produced, based on a method that allows the expression of kP systems as a set of communicating X-machines [10].

III. Simulation Frameworks for KP Systems

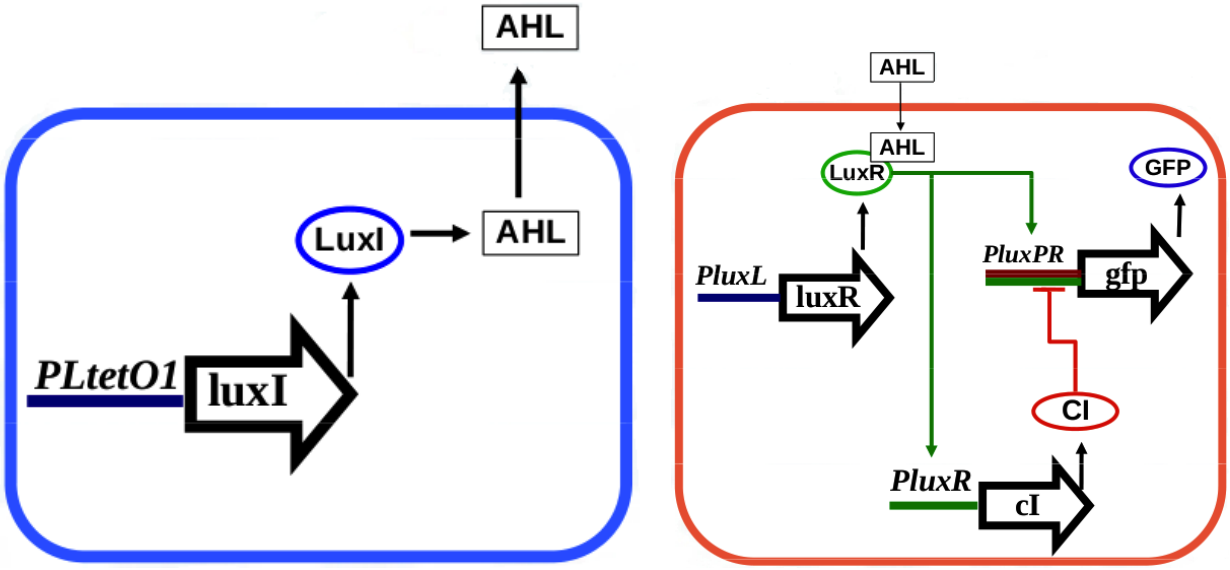
In [11] we have shown the benefits of using a model checker for verifying various properties of the system and for checking the validity of the model. However, we face the well-known problem of state explosion for such approaches and we can only model check very simple systems with very few compartments. The simulation allows us to look at much larger systems and check various results, either final or intermediary ones.

Kernel P system models are specified and represented by a simple and intuitive modelling language, called *kP–Lingua* [12]. kP–Lingua provides for a kP system model a representation into a machine readable format. It also has its own syntax and specific ways of creating compartment types, their instances and connections between them.

KPWorkbench integrates a simulation tool, KPWorkbench Simulator, and provides mechanisms to translate kP–Lingua specifications to Flame.

A. KPWorkbench Simulator

KPWorkbench Simulator is a custom simulation tool, implemented in C# programming language.The tool requires a



(a) Sender cell.

(b) Pulsing cell.

Fig. 1: Two cell types of the pulse generator system (taken from [11]).

kP system model specified in kP–Lingua as input and provides traces of execution for a kP system model. This is translated into an internal data structure, which allows to represent compartments, containing multisets of objects and rules, and their connections with other compartments. The execution strategy in each compartment is interpreted step by step. The simulator provides a command line user interface displaying the current configuration (the content of each compartment) at each step. The output can be printed on command prompt or can be redirected to a file. Depending on the starting step and the granularity of the output, the amount of the printing data will change, which can significantly affect the execution time. The tool is particularly useful for quickly running sanity check on a kP system model, for checking the temporal evolution of the system and for inferring useful information from the simulation results.

B. FLAME

FLAME [13] is a general purpose agent based framework, built on top of the X-machine formalism, a state based model with transformation functions associated to the transitions of the model. It represents the structure of the state machine using an XML format and the transformation functions in standard C. FLAME has become very popular and widely used for numerous applications. The latest developments of FLAME have been focussing on developing variants for high performance computers [13].

The current translator from kP–Lingua maps kP systems into FLAME agents with internal behaviour consisting of rule rewriting and communication. The FLAME environment then executes the model the requested number of steps, storing intermediary results that can be afterwards analysed or interpreted.

A kP system is transformed into a communicating X-machine system by constructing, for each membrane, a communicating X-machine [10] that simulates its behaviour. An additional X-machine, that helps the synchronization of the others, is also built. Each execution strategy of the membrane corresponds to a transition in the communicating X-machine. In FLAME, the communicating X-machines are transformed into agents. Here, the additional X-machine is no longer needed since the synchronization is achieved through message passing.

IV. PERFORMANCE COMPARISON

In this section, we will evaluate the performances of two simulators, integrated into the KPWORKBENCH platform. The performances are very similar in small models. Thus, the evaluation should be performed in large systems. We therefore choose a model from synthetic biology, because synthetic biology models can be very large and it will be a good test case for the simulators. Here, we choose the synthetic *pulse generator*.

A. Pulse Generator

The *pulse generator* [14] is a synthetically constructed colony of bacteria, containing two types of cells: *sender* and *pulsing* (see Figure 1). The sender cells synthesise a signalling protein, transmitted through the pulsing cells. The pulsing cells express the green fluorescent protein (`GFP`) triggered by the signalling molecules, and propagate the excess signalling molecules to the neighbouring cells. The biological process illustrated in Figure 1 can be summarised as follows [11]:

> "Sender cells contain the gene `luxI` from *Vibrio fischeri*. This gene codifies the enzyme `LuxI` responsible for the synthesis of the molecular signal `3OC12HSL` (`AHL`). The `luxI` gene is expressed constitutively under the regulation of the promoter

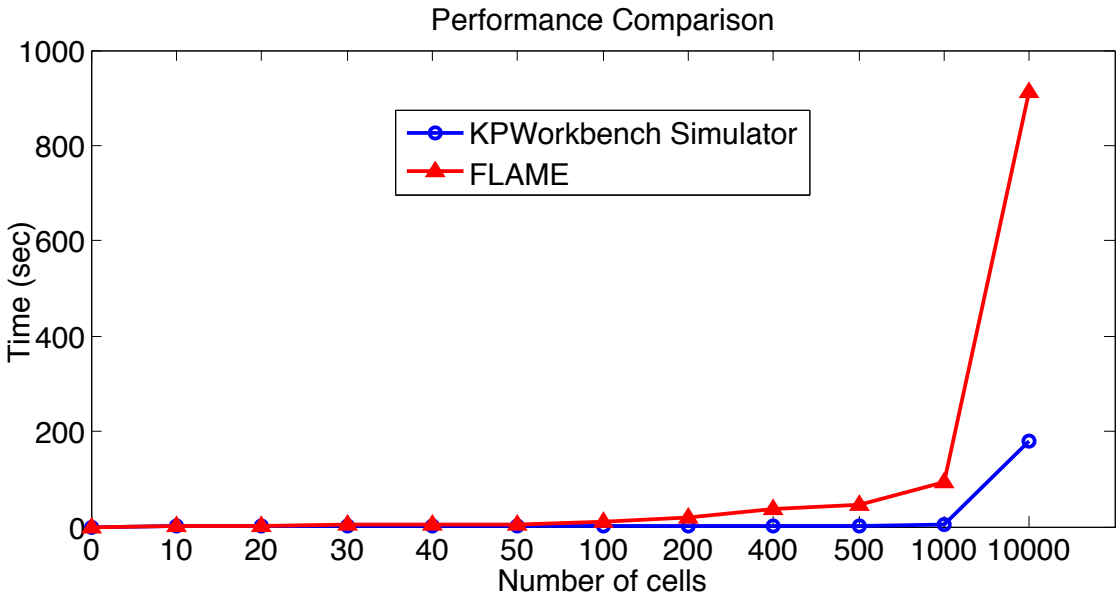


Fig. 2: The comparative simulation results for KPWORKBENCH and FLAME

`PLtetO1` from the tetracycline resistance transposon."

"Pulsing cells contain the `luxR` gene from *Vibrio fischeri* that codifies the `3OC12HSL` receptor protein `LuxR`. This gene is under the constitutive expression of the promoter `PluxL`. It also contains the gene cI from lambda phage codifying the repressor `CI` under the regulation of the promoter `PluxR` that is activated upon binding of the transcription factor `LuxR_3OC12`. Finally, this bacterial strain carries the gene `gfp` that codifies the green fluorescent protein under the regulation of the synthetic promoter `PluxPR` combining the `Plux` promoter (activated by the transcription factor `LuxR_3OC12`) and the `PR` promoter from lambda phage (repressed by the transcription factor `CI`)."

The bacterial strains above are distributed in a specific spatial distribution as a lattice with n rows and m columns. The first two columns consist of sender cells, whereas the rest are pulsing cells. The behaviour of each sender cell is described by 7 rewriting and 1 communication rules and that of the pulsing cell by 34 rewriting and 1 communication rules. The entire model is described in [11] where different properties of the system, both quantitative and qualitative, are verified for small size lattices. Here we consider these two cell types, with the above mentioned rules, but with various lattices which are described in the next section.

The pulse generator is a challenging example, as it is compartmental by design and the dynamic behaviour of each bacterial strain is governed by a large number of kinetic rules. When the number of compartments are increased, the size of the model grows very sharply and the execution of simulations becomes demanding. This makes the pulse generator a good test case for our simulators.

B. Experiments

We will consider a simpler lattice with only a sender and a pulsing cell, but this system will be multiplied by 10, 20, 30, 40, 50, 100, 200, 400, 500, 1000, and 10000 times. For the purpose of these experiments these systems are equivalent to lattices with sizes in the range 10 .. 10000, which are significantly more complex than those described in [11]. Each case will be executed 5 times and the average time calculated. These will be executed with the native KPWORKBENCH simulator and with the FLAME simulator. We note that the system model is described as a kP system model, which is automatically translated into the FLAME simulator. Also, the KPWORKBENCH simulator accepts kP system models, as input.

The results of the simulations on both the KPWORKBENCH simulator and FLAME are comparatively presented in Figure 2. The x axis gives the number of send-pulse pairs of cells, while the y axis indicates the time in seconds. The experiments were performed on a PC with the following configuration: Intel(R) Core(TM)2 Quad CPU - Q6600 2,4Ghz, 4GB RAM. In what follows we explain the performance difference between the two simulators.

In the KPWORKBENCH SIMULATOR, each membrane of the kP system is represented by an instance of a class, transformed from the kP–Lingua language. This approach makes the simulation to be performed in a single memory space, that scales according to the number of membranes used in the model and the number of objects resulting from applying the rules in each simulation step.

In FLAME each agent is represented by an acyclic X-machine (no loops are allowed in order to ensure the completion of the execution of the agent). The agent is executed by passing from one state to another in the X-machine and processing data using functions that are attached to the transitions. When the X-machine reaches the final state, the data

is written to the hard disk and it is then used as input for the next iteration. An important characteristic of FLAME is that it first reads the input data, stored in XML format files, from the hard drive and writes it back at the end of each iteration.

In FLAME, each membrane of the kP system is represented by an agent. The rules are stored together with the membrane multiset as agent data. For each type of membrane from the kP system, a type of agent is defined, and for each execution strategy of the membrane, states are created in the X-machine. Transitions between the two states are represented by C functions that are executed in FLAME when passing from one state to another. Each type of strategy defines a specific function that applies the rules according to the execution strategy.

Given the way the agents for simulating a kP system in FLAME are defined, the volume of data increases with the number of types of membranes, the number of their instances and the size of their multisets. (Note that, since there are no structural rules in our model, the number and structure of membranes remain unchanged throughout the simulation, so the execution time will depend only on the size of the processed multisets.) Consequently, the more data we have, the more time for reading and writing data from or to the hard disk is required. This explains the higher execution time in the case of the FLAME simulator than for the KPWORKBENCH SIMULATOR. It is expected that at least for systems of the same type, i.e., using one single rule per step, the behaviour of the two platforms will be similar. A better correlation between the type of the system, the number of compartments and number of rules, and the behaviour of these platforms will be investigated in a forthcoming paper.

On the other hand, the distributed architecture of Flame allows the simulation to be run on parallel supercomputers with great performance improvements [13]. (For the moment, the implementation of kP Workbench is not suitable for running on parallel computers, but this issue may be considered at a later stage.) Significant performance gains could also be obtained by using solid-state drives (SSDs) for data storage, with lower access time than traditional HDDs, but this is beyond the scope of this paper.

V. CONCLUSION

In this paper we have presented the simulation results of a synthetic biology model coded as a kernel P system, a nature inspired computational paradigm, executed under two different simulation environments. The results presented show the capability of the simulation environments to deal with large scale models - up to 10000 components, each with more than 30 transitions - and consequently with the benefits of a complementary approach to model verification methods, already used for such systems.

The experiments performed show some expected behaviour, whereby a specialised simulation tool, KPWORKBENCH SIMULATOR, provides better results, in terms of execution time, then a general purpose simulation environment, namely FLAME. Both tools produce the same behaviour starting from the same specification, kP–Lingua description, and the translation process is obtained in an automatic way.

In the long term, we aim to show the way the results of the simulation and those of formal verification complement each other for a better understanding of the system behaviour.

ACKNOWLEDGMENT

IN, FI and MG are supported by a grant of the Romanian National Authority for Scientific Research, CNCS-UEFISCDI (project number: PN-II-ID-PCE-2011-3-0688). SK and MG acknowledge the support provided for the synthetic biology research by EPSRC ROADBLOCK (project number: EP/I031812/1). MB is supported by a PhD studentship provided by the Turkey Ministry of Education.

REFERENCES

[1] G. Rozenberg, T. Bäck, and J. N. Kok, Eds., *Handbook of Natural Computing*. Springer, 2012.

[2] G. Păun, "Computing with membranes," *Journal of Computer and System Sciences*, vol. 61, no. 1, pp. 108–143, 2000.

[3] G. Păun, G. Rozenberg, and A. Salomaa, Eds., *The Oxford Handbook of Membrane Computing*. Oxford University Press, 2010.

[4] P. Frisco, M. Gheorghe, and M. J. Pérez-Jiménez, Eds., *Applications of Membrane Computing in Systems and Synthetic Biology*. Springer, 2014.

[5] M. Gheorghe, F. Ipate, and C. Dragomir, "Kernel P systems," pp. 153–170, 2012.

[6] M. Gheorghe, F. Ipate, C. Dragomir, L. Mierlă, L. Valencia-Cabrera, M. García-Quismondo, and M. J. Pérez-Jiménez, "Kernel P systems - Version 1," pp. 97–124, 2013.

[7] C. Dragomir, F. Ipate, S. Konur, R. Lefticaru, and L. Mierlă, "Model checking kernel P systems," in *Proc. 14th Conference on Membrane Computing*, ser. LNCS, vol. 8340. Springer, 2014, pp. 151–172.

[8] M. E. Bakir, F. Ipate, S. Konur, L. Mierlă, and I. Niculescu, "Extended simulation and verification platform for kernel p systems," in *15th International Conference on Membrane Computing*, 2014, p. To Appear.

[9] S. Konur, M. Gheorghe, C. Dragomir, F. Ipate, and N. Krasnogor, "Conventional verification for unconventional computing: a genetic XOR gate example," *Fundamenta Informaticae*, p. To Appear, 2014.

[10] I. Niculescu, F. I. M. Gheorghe, and A. Stefanescu, "From kernel P systems to X-machines and FLAME," *Journal of Automata, Languages and Combinatorics*, to appear.

[11] J. Blakes, J. Twycross, S. Konur, F. Romero-Campero, N. Krasnogor, and M. Gheorghe, "Infobiotics workbench: A P systems based tool for systems and synthetic biology," in *Applications of Membrane Computing in Systems and Synthetic Biology*, ser. Emergence, Complexity and Computation. Springer International Publishing, 2014, vol. 7, pp. 1–41.

[12] M. Gheorghe, F. Ipate, C. Dragomir, L. Mierla, L. Valencia-Cabrera, M. García-Quismondo, and M. J. Pérez-Jiménez, "Kernel P systems," *Eleventh Brainstorming Week on Membrane Computing*, pp. 97–124, 2013.

[13] S. Coakley, M. Gheorghe, M. Holcombe, S. Chin, D. Worth, and C. Greenough, "Exploitation of high performance computing in the flame agent-based simulation framework," in *High Performance Computing and Communication 2012 IEEE 9th International Conference on Embedded Software and Systems (HPCC-ICESS), 2012 IEEE 14th International Conference on*, June 2012, pp. 538–545.

[14] S. Basu, R. Mehreja, S. Thiberge, M.-T. Chen, and R. Weiss, "Spatiotemporal control of gene expression with pulse-generating networks," *PNAS*, vol. 101, no. 17, pp. 6355–6360, 2004.