Contents lists available at ScienceDirect

SoftwareX

journal homepage: www.elsevier.com/locate/softx

Original software publication

kPWorkbench: A software suit for membrane systems

Savas Konur[a,*], Laurenţiu Mierlă[b], Florentin Ipate[c], Marian Gheorghe[a]

[a] Department of Computer Science, University of Bradford, Horton Building, Bradford BD7 1DP, UK
[b] Department of Computer Science, University of Pitesti, Str. Targul din Vale, nr.1, 110040 Pitesti, Arges, Romania
[c] Department of Computer Science, University of Bucharest, Str. Academiei nr. 14, 010014, Bucharest, Romania

ARTICLE INFO

Article history:
Received 9 August 2019
Received in revised form 22 January 2020
Accepted 23 January 2020

Keywords:
Membrane computing
Modelling
Simulation
Agent-based simulation
High performance simulation
Verification
Synthetic biology

ABSTRACT

Membrane computing is a new natural computing paradigm inspired by the functioning and structure of biological cells, and has been successfully applied to many different areas, from biology to engineering. In this paper, we present kPWorkbench, a software framework developed to support membrane computing and its applications. kPWorkbench offers unique features, including *modelling*, *simulation*, *agent-based high performance simulation* and *verification*, which allow modelling and computational analysis of membrane systems. The kPWorkbench formal verification component provides the opportunity to analyse the behaviour of a model and validate that important system requirements are met and certain behaviours are observed. The platform also features a property language based on natural language statements to facilitate property specification.

Code metadata

Current code version	v1.0
Permanent link to code/repository used for this code version	https://github.com/ElsevierSoftwareX/SOFTX_2019_254
Code Ocean compute capsule	https://bit.ly/33ArxOl
Legal Code License	MIT License
Code versioning system used	git
Software code languages, tools, and services used	.Net Core, C#, Spin, Promela, NuSMV, FLAME, ANTLR
Compilation requirements, operating environments & dependencies	https://git.io/JeRDD
If available Link to developer documentation/manual	
Support email for questions	l.m.mierla@bradford.ac.uk

Software metadata

Current software version	kPWorkbench v1.0, kPWorkbench UI v1.0
Permanent link to executables of this version	https://git.io/Je2Fa
Legal Software License	MIT License
Computing platforms/Operating Systems	kPWorkbench: Linux x64, OS X x64, Windows x64; kPWorkbench UI: Windows x64
Installation requirements & dependencies	https://git.io/JeRDD
If available, link to user manual - if formally published include a reference to the publication in the reference list	https://git.io/JeRDD, https://git.io/Je2Fr
Support email for questions	l.m.mierla@bradford.ac.uk

* Corresponding author.
E-mail addresses: s.konur@bradford.ac.uk (S. Konur), l.m.mierla@bradford.ac.uk (L. Mierlă), florentin.ipate@ifsoft.ro (F. Ipate), m.gheorghe@bradford.ac.uk (M. Gheorghe).

https://doi.org/10.1016/j.softx.2020.100407

1. Motivation and significance

In order to go beyond the boundaries of conventional computer science and to be able to address more challenging problems, researchers have extensively worked on introducing new computational models and algorithms inspired from biological, physical, and chemical systems. *Membrane computing* [1] is a branch of nature inspired computing, aiming to develop computational models, methods and techniques by studying biological systems.

The central modelling formalisms within this paradigm are called *membrane systems* or *P systems*, and are inspired by the functioning and structure of biological cells. P systems provide a mapping between biological cells and membranes, which are the computational units of the formalism. Several variants of P systems have been introduced and studied to model and analyse different problems, e.g., systems and synthetic biology [2,3], synchronisation of distributed systems [4], optimisations and graphics [5].

In the last twenty years, apart from introducing and studying many variants of membrane systems [6], a number of tools have been built in order to support both theoretical investigations, but also practical applications – a thorough overview of most of the tools in this area is presented in some papers (see, for example, [7]). Amongst the most important tools produced so far are those widely used in applications. One such tool is P-Lingua [8,9] covering a broad range of membrane systems and using an integrated platform, called MeCoSim [10]; the tool is specifically utilised in modelling and simulation of various biological systems. Another tool used to model biological systems is Meta PLab [11] which deals with a special class of P systems, called MP systems [12]. Numerical P systems, used in modelling and simulation of robot controllers benefit from a tool called SNUPS [13]. Also hybrid P systems are based on an object oriented platform accepting an amalgamation of features of various P systems, and supported by a simulator, called UPSimulator [14].

While the introduction of new variants allowed modelling different sets of problems, the ad-hoc addition of new features has caused an abundance of P system variants, with a lack of a coherent integrating view and well-defined framework that would allow us to analyse, verify and validate the systems behaviour.

To address these issues, we have introduced *kernel P systems (kP systems)* [15,16] to create more general membrane computing models, integrating the most used concepts from P systems. The expressive power and efficiency of the newly introduced kP systems have been illustrated by a number of representative case studies [15,17–19]. In this respect, we have also introduced a modelling language, called *kP–Lingua*, allowing one to write kP system models. The theoretical aspects of the methods and techniques developed for kP systems have been discussed in [20–23].

To provide a tool support for this framework, we have developed the kPWORKBENCH platform (available and downloadable from its website [24]), which permits modelling and computational analysis of membrane systems through its unique features, *modelling*, *simulation*, *agent-based high performance simulation* and *verification*. To assist users in verification process, which is a very cumbersome process for non-experts, the platform also features a user friendly property language based on *natural language* statements, which makes the property specification a much easier task. These unique features make kPWORKBENCH the only available tool supporting the non-probabilistic modelling and analysis of membrane systems using various computational approaches. The usability and novelty of our approach have been illustrated by some case studies from systems and synthetic biology [17,18] to some engineering problems [19,25].

2. Software description

This paper presents the first stable software release of kPWORKBENCH, a software platform that integrates a set of tools and methods, allowing one to *model* membrane systems and to analyse them through *simulation*, *agent-based high-performance simulation* and *verification*.

2.1. Software architecture

Fig. 1 depicts an overview of the kPWORKBENCH system architecture, which consists of three modules:

1. The **kernel P (kP)** module takes a kP system model specified in kP–Lingua, which can be created or edited using a dedicated model editor, as input. The grammar of the kP–Lingua language is written in ANTLR (ANother Tool for Language Recognition) [26], automatically generating the necessary syntactical and semantic analysers. The *kP Model* module accommodates the corresponding data structures of the input model, comprising compartment types, execution strategies, rules, multiset of objects and connections between compartments. The kP–Lingua module instantiates a kP Model object and maps the AST (abstract syntax tree) generated by ANTLR to that object. This object is used as Data Transfer Object (DTO) between different modules of the framework. This separation helps developers easily adding new components to the framework.

2. The **Simulation** module consists of two components, kPWORKBENCH simulator and FLAME agent-based simulator [27]. Both require the kP Model object and simulator parameters, e.g. number of steps, as input. The kPWORKBENCH Simulator component is a custom simulator, which processes the multisets of objects of the input model with respect to its execution strategies and rules. The FLAME Translator transforms the kP Model object into a FLAME Model object that aggregates agent, function, input, condition and output classes. It assigns each compartment to an agent, and the rules and the multiset of objects are stored as agent data. It creates a specific function for each type of execution strategy. In addition, it creates C functions that represent the system behaviour (they are executed by FLAME when the agent makes a transition from one state to another). The FLAME Translator uses the ANTLR template group feature to produce the FLAME simulator specifications from the FLAME Model object.

3. The **Verification** module contains three components: the SPIN [28] and NUSMV [29] translators and the *kP Queries* module:

The SPIN Translator has two main components: *Translator* and *Promela* (SPIN's specification language). The Translator maps the kP Model object to a *Promela* object using the following procedure [20]:

(i) A compartment type is translated into a data type definition with the multiset of objects and links to other compartments, and also with temporary storage variables that provide the parallelism of P systems.
(ii) Multiset of objects is assigned to an integer array where an index denotes the object and its value represents the multiplicity of the object.
(iii) The set of rules are organised according to the execution strategies mapped by a *Proctype* definition – a Promela process.
(iv) Maximal parallelism and arbitrary execution strategies are mapped to the *Do* statement, and choice execution strategy is mapped to *If* statement.

After the mapping process, the *Translator* component translates the Promela object to the corresponding Promela model,

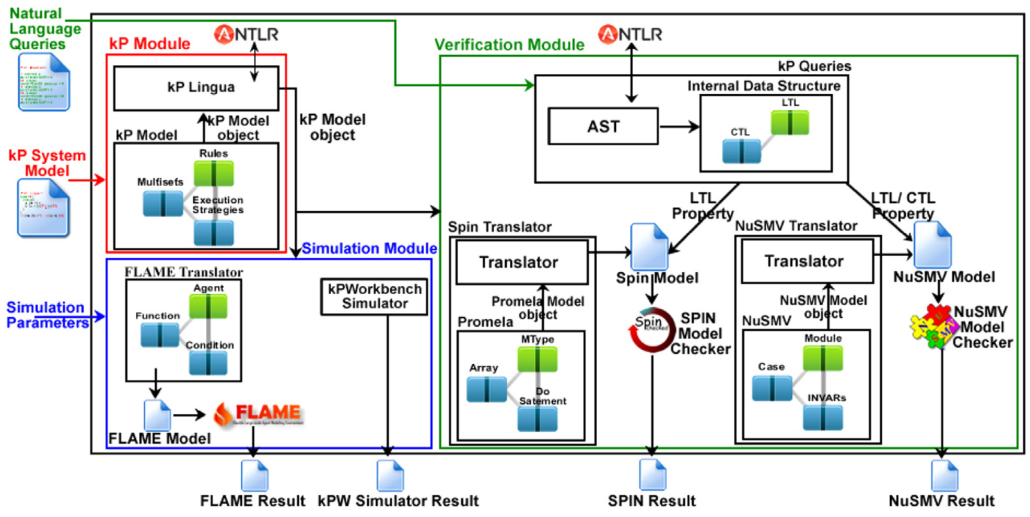

Fig. 1. The overview architecture of kPWorkbench framework.

used by the Spin model checker. More details about the translation from kP System model to the Spin model checker specification can be found in [20].

Similarly, the NuSMV Translator translates the kP Model object to the corresponding NuSMV representation (NuSMV's specification language). The translator has two main components: *Translator* and NuSMV. The NuSMV component consists of subcomponents representing the NuSMV language objects. The *Translator* maps the kP Model object to the NuSMV object as follows:

(i) Each compartment is translated into a module.
(ii) The content of compartments is translated into variables.
(iii) The initial multisets of the compartment are assigned into module parameters.
(iv) Rules and guards are translated into the case statements.
(v) The behaviour of execution strategies and the parallelism of P systems are achieved by introducing custom variables.

After the mapping process, the Translator component generates the NuSMV model from the NuSMV object.

The *kP-Queries* module receives a property, a natural language based statement, as input. The user can build properties from the property language editor. The editor interacts with the kP–Lingua model, and permits accessing the native model elements, which simplifies the property building process. The kP-Queries' domain language has its own grammar, which is independent from and much simpler than the target model checking languages. The DSL (domain specific language) of the property language is written in ANTLR, receiving the EBNF grammar as input and generates the corresponding syntactic and semantic analysers as well as the corresponding AST. We use the *Visitor design pattern* approach, which enables the kP-Queries module to translate every node of the internal presentation of property into the target model checker's corresponding property specification language.

2.2. Software functionalities

Modelling

kPWorkbench accepts kP system models specified in an intuitive modelling language, *kP–Lingua*. kP systems accumulate the most important aspects of various P system variants, so kP–Lingua provides a generic language to model various membrane systems. kPWorkbench features a graphical model editor, permitting to create new model files and editing existing files.

Simulation

kPWorkbench offers two different approaches to simulate kP systems. In both approaches, a kP–Lingua model is provided as an input, and the execution traces of the model are returned as an output. These traces permit exploring the dynamics of the system and observing how the system evolves over time.

In the first approach, we have developed a custom simulation tool [22], which recreates the system dynamics as a set of simulation runs in a *sequential way*. The tool translates a kP–Lingua specification into an internal data structure, which permits representing compartments, containing multisets of objects and rules, and their connections with other compartments.

In the second approach, we have integrated the Flame simulator [27,30], a general purpose large scale agent based simulation environment. Flame is based on the X-machine formalism [31], a type of extended finite state machine whose transitions are labelled by processing functions that operate on a (possibly infinite) set called memory, that models the system data.

In order to simulate kernel P system models in a *parallel way* using the Flame framework, an automated model translation has been implemented for converting the kP–Lingua specification into communicating X-machines [31]. One of the main advantages of this approach is the high scalability degree and efficiency for simulating large scale models.

Verification

Verification, in particular model checking, has been widely applied to the analysis of various systems [32–36]. Verification checks if system in question meets user requirements, expressed in a formal logic [37–40], by exhaustively analysing all possible execution paths verification.

Utilising a comprehensive, integrated and automated verification approach is a very challenging task in the context of membrane computing. For example, it is very difficult to transform some complex features, e.g. membrane division, dissolution and link creation/destruction, into suitable abstractions in verification tools.

We have successfully addressed these issues, and developed a verification environment for kPWorkbench, integrating some state of the art model checking tools, e.g. Spin [28] and NuSMV [29]. The translations from a kP–Lingua representation to the corresponding Spin and NuSMV inputs (i.e. Promela and Smv, respectively) are automatically performed.

In order to facilitate the property specification task (a very difficult process for non-experts who are not familiar with verification languages) kPWorkbench features a user friendly property

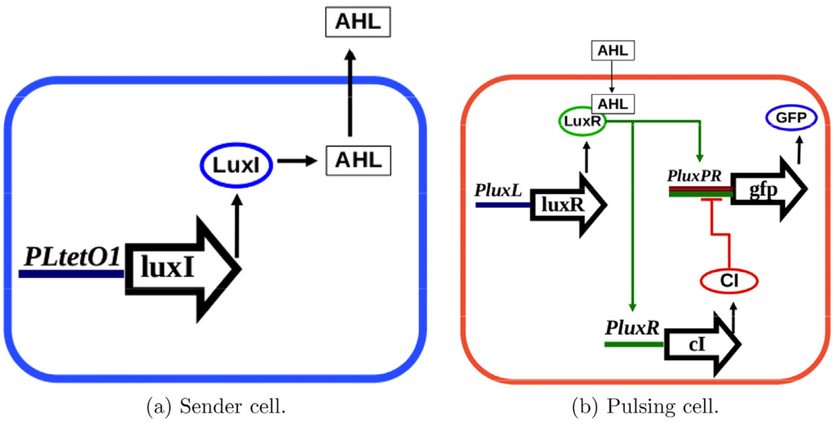

Fig. 2. Two cell types of the pulse generator system.

language, *kP-Queries*, based on *natural language* statements. The language also provides a list of property patterns (templates), generated from most commonly used queries. The property language permits specifying the target logic (i.e. LTL and CTL) for different properties without placing a requirement on a specific model checker. In this way, we can use the same set of properties in various verification experiments.

3. Illustrative examples

In this section, we show the usability of kPWorkbench on a synthetic biology case study [18,41]. The *pulse generator* [42] is a synthetic biology system, composed of two types of bacterial strains: *sender* and *pulsing* cells (see Fig. 2). The sender cells produce a signal (AHL) and propagates it through the pulsing cells, which express the green fluorescent protein (GFP) upon sensing the signal. The excess of the signalling molecules are propagated to the neighbouring cells. Sender cells synthesise the signalling molecule AHL through the enzyme LuxI, expressed under the constitutive expression of the promoter PLtetO1. Pulsing cells express GFP under the regulation of the PluxPR promoter, activated by the LuxR_AHL_2 complex. The LuxR protein is expressed under the control of the PluxL promoter. The GFP production is repressed by the transcription factor CI, codified under the regulation of the promoter PluxR that is activated upon binding of the transcription factor LuxR_AHL_2.

For this case study, we have designed a membrane system, which is a lattice comprising 1 sender cell and 10 pulsing cells in a sequential order. Through simulation and verification experiments, we have observed the propagation of the signalling molecules from the sender cells to the pulsing cells. The system is modelled in kernel P systems using the kP–Lingua language. The actual kP–Lingua model can be accessed at https://github.com/Kernel-P-Systems/kPWorkbench/wiki/Case-Studies

Simulation

The simulation components of kPWorkbench are used to explore the temporal evolution of the system and to infer various information from the simulation results. By analysing the execution steps, we can explore the dynamic behaviour of the system in question.

Table 1 presents the production and availability of the signalling molecules in the sender cell (i.e. $\mathtt{sender}_1$) and the transmission of the signalling molecules and the production of the green florescent protein in the furthest pulsing cell (i.e. $\mathtt{pulsing}_{10}$). The simulation results show that the signalling molecule can be produced and transmitted by the sender cell on average within 10,000 steps. The furthest pulsing cell will eventually receive these signalling molecules (between 10,001 and 20,000 steps), and can use the signal for the production of GFP in later steps (after 137,179 steps), which confirms the propagation behaviour.

The simulation presented in Table 1 is a sequential process, which is not very efficient for large systems, e.g. biological systems. The Flame simulator provides a much faster and efficient alternative for large systems as it uses parallel algorithms, run in high performance environments. Fig. 3 compares the performance of the kPWorkbench native simulator (sequential simulation) and Flame simulator (parallel simulation). As the number of cells increases, Flame runs significantly faster than the kPWorkbench simulator.

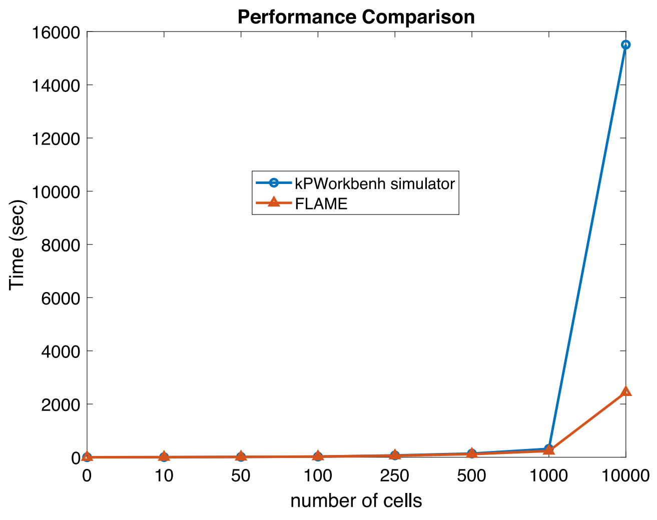

Fig. 3. The comparative simulation results for kPWorkbench and Flame.

Table 1
Simulation results.

Step interval	$\mathtt{sender}_1$	$\mathtt{pulsing}_{10}$	
	AHL	AHL	GFP
0–10,000	Exist	None	None
10,001–20,000	Exist	Exist	None
20,001–30,000	Exist	Exist	None
...	...	...	...
80,001–137,178	Exist	Exist	None
137,179–150,000	Exist	None	Exist

Table 2
Verified properties.

Property	Description	kP-Query	Result
1	**AHL** is eventually propagated to $\textbf{pulsing}_{10}$	**ctl:** `eventually puls10.signalAHL` > 0	T
2	**GFP** is eventually produced in $\textbf{pulsing}_{10}$	**ctl:** `eventually puls10.proteinGFP` > 0	T
3	**AHL** in $\textbf{pulsing}_{1}$ is followed by **AHL** in $\textbf{pulsing}_{10}$	**ctl:** `puls1.signalAHL` > 0 `followed-by puls10.signalAHL` > 0	T
4	**AHL** in $\textbf{sender}_{1}$ is followed by **GFP** in $\textbf{pulsing}_{10}$	**ctl:** `send1.signalAHL` > 0 `followed-by puls10.proteinGFP` > 0	T
5	**GFP** in $\textbf{pulsing}_{10}$ is preceded by **AHL** in $\textbf{pulsing}_{9}$	**ctl:** `puls10.proteinGFP` > 0 `preceded-by puls9.signalAHL` > 0	T

Verification

The verification component allows us to check if a model satisfies the system requirements. In order to observe the propagation behaviour of the pulse generator system, we have written some properties in kP-Queries and verified them on the kP–Lingua model.

Table 2 shows the verification results. The first column shows the property number; the second column describes the properties informally; the third column shows the formal properties expressed in kP-Queries (which are then automatically translated into CTL in NuSMV syntax); and the last column presents the verification results. Property 1 and 2, respectively, verify that AHL is eventually propagated to the furthest pulsing cell, which then produces GFP upon sensing the signalling molecules. Property 3 verifies that AHL propagates through the lattice. Property 4 proves that the sender cell starts the expression of AHL, which then triggers the production GFP in the pulsing cells. Finally, Property 5 proves that AHL should be sensed in the previous neighbouring cell before GFP is produced in the next cell.

4. Impact and conclusions

In this paper, we have discussed a nature inspired computing paradigm, membrane computing, and the kPWorkbench software platform developed to support it.

The need to build tools supporting the membrane computing research has been identified quite early and presented in [6]. The current state of the art development of membrane computing tools [7] refers to a broad spectrum of topics covered. The vast majority of these tools concentrate on a specific variant or a set of variants of membrane systems. Compared with the rest of membrane computing tools, kernel P systems provide more general membrane computing models, integrating the most used concepts from P systems. This is an important aspect of the research within the membrane computing community, as outlined in a survey paper [43].

kPWorkbench integrates several state-of-the-art simulation and verification tools and methods. Featuring multiple simulators, using a native process and agent-based approaches relying on sequential and high performance execution, is very unique in this field. These features allow kPWorkbench to efficiently express problems studied with other classes of membrane systems, consequently analyse them via simulation and verification and decide on the best solutions.

The formal verification feature, which does not exist in any other membrane computing tools, provides the opportunity to analyse the system behaviour and check if certain properties about the system specification are verified, which cannot be done using the conventional simulation approach. Another unique feature is the user-friendly property specification process using natural language statements, making the property specification very easy for non-experts. These features make kPWorkbench the only available integrated toolset permitting non-deterministic analysis of membrane systems.

kPWorkbench has been used in computational modelling and analysis of various systems (such as systems and synthetic biology [17,18], cruise control system [19], sorting [44], 3-Col problem [15]), providing insights into their behaviour. This helps formulating new research questions and addressing them within an efficient, robust and rigorous environment.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

The work of SK is supported by EPSRC, United Kingdom (EP/R043787/1). FI and MG are supported by a grant of the Romanian National Authority for Scientific Research, CNCS-UEFISCDI, project number PN-III-P4-ID-PCE-2016-0210.

References

[1] Păun Gh. Computing with membranes. J Comput System Sci 2000;61(1). http://dx.doi.org/10.1006/jcss.1999.1693.
[2] Frisco P, Gheorghe M, Pérez-Jiménez MJ, editors. Applications of membrane computing in systems and synthetic biology. Springer; 2014, http://dx.doi.org/10.1007/978-3-319-03191-0.
[3] Sanassy D, Fellermann H, Krasnogor N, Konur S, Mierlă L, Gheorghe M, Ladroue C, Kalvala S. Modelling and stochastic simulation of synthetic biological boolean gates. In: 16th IEEE international conference on high performance computing and communications. 2014, p. 404–8. http://dx.doi.org/10.1109/HPCC.2014.68.
[4] Dinneen MJ, Yun-Bum K, Niculescu R. Faster synchronization in P systems. Nat Comput 2012;11(4):637–51. http://dx.doi.org/10.1007/s11047-011-9271-z.
[5] Gimel'farb GL, Niculescu R, Ragavan S. P system implementation of dynamic programming stereo. J Math Imaging Vision 2013;47(1–2):13–26. http://dx.doi.org/10.1007/s10851-012-0367-6.
[6] Păun Gh, Rozenberg G, Salomaa A, editors. The oxford handbook of membrane computing. Oxford University Press; 2010.
[7] Valencia-Cabrera L, Orellana-Martín D, del Amor MAM, Pérez-Hurtado I, Pérez-Jiménez MJ. From super-cells to robotic swarms: two decades of evolution in the simulation of p systems. Bull Int Membr Comput Soc 2017;4(2):66–88.
[8] P-Lingua website, URL: http://www.p-lingua.org.
[9] García-Quismondo M, Gutíerrez-Escudero R, Pérez-Hurtado I, Pérez-Jiménez MJ, nez AR-N. An overview of p-lingua 2.0. In: 11th Int. conference on membrane computing. LNCS, vol. 5957, Springer; 2010, p. 264–88. http://dx.doi.org/10.1007/978-3-642-11467-0_20.
[10] MeCoSim website, URL: http://www.p-lingua.org/mecosim/.
[11] Meta PLab website, URL: http://mplab.scienze.univr.it/index.html.
[12] Marchetti L, Manca V. MpTheory Java library: a multi-platform java library for systems biology based on the Metabolic P theory. Bioinformatics 2015;31(8):1328–30. http://dx.doi.org/10.1093/bioinformatics/btu814.
[13] Arsene O, Buiu C, Popescu N. SNUPS - a simulator for numerical membrane computing. Int J Innovative Comput Inf Control 2011;7:3509–22.
[14] Guo P, Quan C, Ye L. UPSimulator: A general P system simulator. Knowl-Based Syst 2019;170:20–5. http://dx.doi.org/10.1016/j.knosys.2019.01.013.
[15] Gheorghe M, Ipate F, Lefticaru R, Pérez-Jiménez MJ, Ţurcanu A, Valencia-Cabrera L, García-Quismondo M, Mierlă L. 3-col problem modelling using simple kernel p systems. Int J Comput Math 2012;90(4):816–30. http://dx.doi.org/10.1080/00207160.2012.743712.
[16] Gheorghe M, Ceterchi R, Ipate F, Konur S, Lefticaru R. Kernel P systems: from modelling to verification and testing. Theoret Comput Sci 2018;724:45–60. http://dx.doi.org/10.1016/j.tcs.2017.12.010.
[17] Konur S, Gheorghe M, Dragomir C, Ipate F, Krasnogor N. Conventional verification for unconventional computing: a genetic XOR gate example. Fund Inform 2014;134:97–110. http://dx.doi.org/10.3233/FI-2014-1093.

[18] Konur S, Gheorghe M, Dragomir C, Mierlă L, Ipate F, Krasnogor N. Qualitative and quantitative analysis of systems and synthetic biology constructs using P systems. ACS Synth Biol 2015;4(1):83–92. http://dx.doi.org/10.1021/sb500134w.
[19] Lefticaru R, Bakir ME, Konur S, Stannett M, Ipate F. Modelling and validating an engineering application in kernel p systems. In: Gheorghe M, Rozenberg G, Salomaa A, Zandron C, editors. Membrane computing. Cham: Springer International Publishing; 2018, p. 183–95. http://dx.doi.org/10.1007/978-3-319-73359-3_12.
[20] Dragomir C, Ipate F, Konur S, Lefticaru R, Mierlă L. Model checking kernel p systems. In: 14th int. conference on membrane computing. LNCS, vol. 8340, Springer; 2013, p. 151–72. http://dx.doi.org/10.1007/978-3-642-54239-8_12.
[21] Gheorghe M, Konur S, Ipate F, Mierla L, Bakir ME, Stannett M. An integrated model checking toolset for kernel P systems. In: Membrane computing. Springer; 2015, p. 153–70. http://dx.doi.org/10.1007/978-3-319-28475-0_11.
[22] Bakir ME, Konur S, Gheorghe M, Niculescu I, Ipate F. High performance simulations of kernel P systems. In: 16th IEEE int. conference on high performance computing and communications. 2014, p. 409–12. http://dx.doi.org/10.1109/HPCC.2014.69.
[23] Bakir ME, Ipate F, Konur S, Mierlă L, Niculescu I. Extended simulation and verification platform for kernel P systems. In: Gheorghe M, Rozenberg G, Salomaa A, Sosík P, Zandron C, editors. Membrane computing. Cham: Springer International Publishing; 2014, p. 158–78. http://dx.doi.org/10.1007/978-3-319-14370-5_10.
[24] kPWorkbench website: https://github.com/kernel-p-systems/kpworkbench.
[25] Lefticaru R, Konur S, Yildirim Ü, Uddin A, Campean F, Gheorghe M. Towards an integrated approach to verification and model-based testing in system engineering. In: The international workshop on engineering data- & model-driven applications (EDMA-2017). 2017, p. 131–8. http://dx.doi.org/10.1109/iThings-GreenCom-CPSCom-SmartData.2017.25.
[26] ANTLR website, URL: http://www.antlr.org.
[27] Coakley S, Gheorghe M, Holcombe M, Chin S, Worth D, Greenough C. Exploitation of high performance computing in the FLAME agent-based simulation framework. In: Proceedings of 14th IEEE int. conference on high performance computing and communications, 2012, pp. 538–545..
[28] Holzmann GJ. The model checker SPIN. IEEE Trans Soft Eng 1997;23(5):275–95. http://dx.doi.org/10.1109/32.588521.
[29] Cimatti A, Clarke E, Giunchiglia E, Giunchiglia F, Pistore M, Roveri M, Sebastiani R, Tacchella A. Nusmv version 2: An open source tool for symbolic model checking. In: Roc. int. conference on computer-aided verification (CAV 2002). LNCS, vol. 2404, Springer; 2002, p. 359–64. http://dx.doi.org/10.1007/3-540-45657-0_29.
[30] FLAME website, URL: http://flame.ac.uk.
[31] Holcombe M. X-machines as a basis for dynamic system specification. Softw Eng J 1988;3(2):69–76. http://dx.doi.org/10.1049/sej.1988.0009.
[32] Abbink H, et al. Automated support for adaptive incident management, In: Proc. of the 1st int. workshop on information systems for crisis response and management, ISCRAM 04. Brussels, 2004, pp. 153–170.
[33] Arapinis M, Calder M, Denis L, Fisher M, Gray P, Konur S, Miller A, Ritter E, Ryan M, Schewe S, Unsworth C, Yasmin R. Towards the verification of pervasive systems. In: Electronic communications of the EASST 22. 2009.
[34] Konur S, Fisher M, Dobson S, Knox S. Formal verification of a pervasive messaging system. Form Asp Comput 2014;26(4):677–94. http://dx.doi.org/10.1007/s00165-013-0277-4.
[35] Konur S, Gheorghe M. A property-driven methodology for formal analysis of synthetic biology systems. IEEE/ACM Trans Comput Biol Bioinform 2015;12:360–71. http://dx.doi.org/10.1109/TCBB.2014.2362531.
[36] Bakir ME, Konur S, Gheorghe M, Krasnogor N, Stannett M. Automatic selection of verification tools for efficient analysis of biochemical models. Bioinformatics 2018;34(18):3187–95. http://dx.doi.org/10.1093/bioinformatics/bty282.
[37] Konur S. An interval logic for natural language semantics. In: Proceedings of the seventh conference on advances in modal logic, Nancy, France, 9-12 2008, 2008, pp. 177–191.
[38] Konur S. Real-time and probabilistic temporal logics: An overview, CoRR abs/1005.3200. arXiv:1005.3200.
[39] Konur S. A survey on temporal logics for specifying and verifying real-time systems. Front Comput Sci 2013;7(3):370–403. http://dx.doi.org/10.1007/s11704-013-2195-2.
[40] Konur S. Specifying safety-critical systems with a decidable duration logic. Sci Comput Program 2014;80(PB):264–87. http://dx.doi.org/10.1016/j.scico.2013.07.012.
[41] Blakes J, Twycross J, Konur S, Romero-Campero F, Krasnogor N, Gheorghe M. Infobiotics workbench: A P systems based tool for systems and synthetic biology. In: Applications of membrane computing in systems and synthetic biology. Emergence, complexity and computation, vol. 7, Springer; 2014, p. 1–41. http://dx.doi.org/10.1007/978-3-319-03191-0_1.
[42] Basu S, Mehreja R, Thiberge S, Chen M-T, Weiss R. Spatio-temporal control of gene expression with pulse-generating networks. PNAS 2004;101(17):6355–60. http://dx.doi.org/10.1073/pnas.0307571101.
[43] Gheorghe M, Gh Păun, Pérez-Jiménez MJ, Rozenberg G. Research frontiers of membrane computing: Open problems and research topics. Internat J Found Comput Sci 2013;24:547–624. http://dx.doi.org/10.1142/S0129054113500202.
[44] Gheorghe M, Ceterchi R, Ipate F, Konur S. Kernel P systems modelling, testing and verification - sorting case study. In: Leporati A, Rozenberg G, Salomaa A, Zandron C, editors. Membrane computing. Cham: Springer International Publishing; 2017, p. 233–50. http://dx.doi.org/10.1007/978-3-319-54072-6_15.