5th Workshop on Membrane Computing and Biologically Inspired Process Calculi

MeCBIC 2011

Fontainebleau, France
23rd August 2011

Editor: Gabriel Ciobanu

Table of Contents

Biologically Inspired Process Calculi, Petri Nets and Membrane Computing

Gabriel Ciobanu
Romanian Academy, Institute of Computer Science
Iaşi, Romania
gabriel@iit.tuiasi.ro

This volume represents the proceedings of the 5th Workshop on Membrane Computing and Biologically Inspired Process Calculi (MeCBIC 2011), held together with the 12th International Conference on Membrane Computing on 23rd August 2011 in Fontainebleau, France.

MeCBIC is usually devoted to membrane computing and biologically inspired process calculi (ambients, brane calculi). This year we also attracted papers dealing with bio-inspired Petri nets in order to promote collaboration between the Petri nets and membrane computing communities.

Biological membranes play a fundamental role in the complex reactions which take place in cells of living organisms. Membrane systems were introduced as a class of distributed parallel computing devices inspired by the observation that any biological system is a complex hierarchical structure, with a flow of biochemical substances and information that underlies their functioning. The modelling and analysis of biological systems has also attracted considerable interest from both the Petri nets and the process calculi research communities. A deeper investigation of the relationships between these formalisms is interesting, providing valuable cross fertilization of these research areas. Membrane computing deals with the computational properties, making use of automata, formal languages, and complexity results. Certain process calculi, such as mobile ambients and brane calculi, work also with notions of compartments and membranes. Petri nets are used to model and analyze the biological systems.

The submitted papers describe biologically inspired models and calculi, biologically inspired languages, properties of biologically inspired models and languages, theoretical links and comparison between different models. All submitted papers were reviewed by three or four referees. We thank the authors and reviewers for doing an excellent job; without their enthusiastic work this volume would not have been possible. We are indebted to the members of the Programme Committee:

Bogdan Aman	Jean-Louis Giavitto	Gethin Norman	Jason Steggles
Roberto Barbut	S.N. Krishna	Andrew Phillips	Angelo Troina
Marco Bernardo	Jean Krivine	G. Michele Pinna	Sergey Verlan
Paola Bonizzoni	Paolo Milazzo	Franck Pommereau	Gianluigi Zavattaro.
Gabriel Ciobanu (chair)			

We express our gratitude to the invited speakers Jetty Kleijn and Cosimo Laneve for their very interesting talks. The first talk presents similarities and differences between Petri nets and membrane systems, and how to enhance the Petri nets in order to faithfully model the dynamics of the biological phenomena represented by membrane systems and reaction systems. The second talk deals with reversibility in massive concurrent systems; reversible structures for massive concurrent systems are introduced and studied, and an equivalence on computations that abstracts away from the order of causally independent reductions is defined.

© Gabriel Ciobanu
All the rights to the paper remain with the authors.

The main aim of the workshop is to bring together researchers working on membrane computing, in biologically inspired process calculi, and in Petri nets in order to present their recent results and to discuss new ideas concerning these formalisms, their properties and relationships. Many thanks to Sergey Verlan and Maciej Koutny for their help in organizing the workshop, and to Bogdan Aman for his help in preparing this volume.

Reversibility in Massive Concurrent Systems

Luca Cardelli
Microsoft Research, Cambridge
luca@microsoft.com

Cosimo Laneve
Università di Bologna
laneve@cs.unibo.it

Reversing a (forward) computation history means undoing the history. In concurrent systems, undoing the history is not performed in a deterministic way but in a causally consistent fashion, where states that are reached during a backward computation are states that could have been reached during the computation history by just performing independent actions in a different order. In RCCS, Danos and Krivine achieve this by attaching a memory m to each process P, in the monitored process construct $m : P$. Memories in RCCS are stacks of information needed for processes to backtrack. Alternatively, Phillips and Ulidowski propose a technique for reversing process calculi without using memories. In this technique, the structure of processes is not destroyed and the progress is noted by underlining the actions that have been performed. In order to tag the communicating processes, they generate unique identifiers on-the-fly during the communications.

These foundational studies of reversible and concurrent computations have been largely stimulated by areas such as chemical and biological systems – called *massive concurrent systems* in the following – where operations are reversible, and only an appropriate injection of energy and/or a change of entropy can move the computational system in a desired direction.

However there is a mismatch between chemical and biological systems and the above concurrent formalisms. In the latter ones, reversibility means *desynchronizing processes that actually interacted in the past* while, in massive concurrent systems, reversibility means *reversibility of configurations*. In order to make massive concurrent systems reversible with the process calculus meaning, one has to remember the position and momentum of each molecule, which is precisely contrary to the well-mixing assumption of biochemical soups, namely that the probability of collision between two molecules is independent of their position (*cf.* Gillespie's algorithm).

To comply with the well-mixing assumption, notions of causality and independence of events need to be adapted to reflect the fundamental fact that different processes of the same species are indistinguishable. Their interactions can cause effects, but not to the point of being able to identify the precise molecule that caused an effect. We introduce an algebra for massive concurrent systems, called *reversible structures*, and, following Lévy, we define an equivalence on computations that abstracts away from the order of causally independent reductions – the *permutation equivalence*. Because of multiplicities this abstraction does not always exchange independent reductions. For example, two reductions that use a same signal cannot be exchanged because one cannot grasp whether the two reductions are competing on a same signal or are using two different occurrences of it. Notwithstanding this inadequacy, permutation equivalence in reversible structures yields a standardization theorem that allows one to remove converse reductions from computations.

Reversible structures may implement significant CCS-style interaction patterns (Cardelli already noticed this by studying a class of reversible systems – the DNA chemical systems). Consider for example a binary operator that takes two input molecules and produces one unrelated output molecule when (and only when) both inputs are present. It is too difficult to engineer the input machinery in order to any possible pattern of interaction, and to produce the output molecule out of their own structure. This operator

© Luca Cardelli and Cosimo Laneve
All the rights to the paper remain with the authors.

is therefore implemented by an artifact that binds the two inputs one after the other and then releases the output out of its own structure. Of course, if the second input never comes it must release the first input, because the first input may be legitimately used by some other operator. This means that the binding of the first input must be reversible, and the natural reversibility of reversible structures is exploited to achieve the correctness.

In order to bridge the gap between reversible process calculi and massive concurrent systems, we consider reversible structures where multiplicities are dropped (terms have multiplicity one) – the *coherence* constraint. Coherence in this strong sense is not realizable in well-mixed chemical solutions, but may become realizable in the future if we learn how to control individual molecules. We demonstrate that coherent reversible structures implement the *asynchronous* fragment of RCCS.

The exact distance between coherent and uncoherent reversible structures (that is, between reversible process calculi and massive systems) is manifested by the computational complexity of the reachability problem (verifying whether a configuration is reachable from an initial one). We demonstrate that reachability in coherent reversible structures has a computational complexity that is quadratic with respect to the size of the structures, a problem that is EXPSPACE-complete in generic structures.

Our study prompts a thorough analysis of reversible calculi where processes have multiplicities and the causal dependencies between copies may be exchanged. Open questions are (i) What synchronization schemas can be programmed in massive concurrent systems? (ii) Are there other constraints, different than coherence, such that relevant bio-chemical properties retains better algorithms than in standard structures? (iii) What is the theory of massive (reversible) systems *with irreversible operators* and what is the relationship with standard programming languages?

Petri Nets and Bio-Modelling

and how to benefit from their synergy

Jetty Kleijn
LIACS, Leiden University, 2300 RA, The Netherlands
kleijn@liacs.nl

Maciej Koutny
School of Computing Science,Newcastle University,
Newcastle upon Tyne, NE1 7RU, United Kingdom
maciej.koutny@ncl.ac.uk

Grzegorz Rozenberg
Leiden Center for Natural Computing, Leiden University
Niels Bohrweg 1, 2333 CA Leiden, The Netherlands
rozenber@liacs.nl

Petri nets are a general, well-established model of concurrent and distributed computation featuring a wealth of tools for the analysis and verification of their behavioural properties. On the other hand, to understand specific biological processes different formalisations have been proposed. Examples here are membrane systems and reaction systems which are close abstractions of the functioning of the living cell. Membrane systems are a computational model inspired by the way chemical reactions take place in cells that are divided by membranes into compartments. The central idea behind reaction systems is that the functioning of a living cell is based on interactions between (a large number of) individual reactions, and moreover these interactions are regulated by two main mechanisms: facilitation and inhibition.

In this talk we are concerned with the intrinsic similarities and differences between Petri nets on the one hand, and membrane systems and reaction systems on the other hand. In particular, we are interested in the benefits that can result from establishing strong semantical links between the latter two models and Petri nets. Our aim is to enhance the Petri net model in order to faithfully model the dynamics of the biological phenomena/processes represented by membrane systems and reaction systems.

After introducing Petri nets, we will outline how to understand and formalise their causality and concurrency semantics.

Then we turn to membrane systems. Like membrane systems, Petri nets are in essence multiset rewriting systems. Using this key commonality we describe a faithful translation from basic membrane systems to Petri nets. To capture the compartmentalisation of membrane systems, the Petri net model has to be extended with localities which in turn leads to the idea of locally synchronised executions. In the thus extended model the standard causality semantics is no longer sound, and we will discuss possible solution to this problem.

Next we describe reaction systems which are a recently proposed model aimed at investigating processes carried by biochemical reactions. Now, the resulting computational model is remarkably different since in reaction systems, biochemical reactions are modeled using a qualitative rather than a quantitative approach. As a consequence, counting — and hence the multiset based calculus implemented in Petri nets — is no longer appropriate. This insight leads to a new class of Petri nets, called set-nets, a novel and challenging class of nets with intriguing (and yet to be discovered) properties.

We conclude the talk by demonstrating how in turn set-nets with localities correspond to membrane systems with qualitative evolution rules.

Altogether this talk aims to demonstrate the fruitful two-way interaction between biological models and Petri nets. Both membrane systems and reaction systems have inspired the introduction of new and

© J. Kleijn, M. Koutny and G. Rozenberg
All the rights to the paper remain with the authors.

relevant extensions to the basic net model, whereas having a Petri net semantics opens the way for a new understanding, analysis and synthesis techniques for biologically inspired systems.

The presentation is essentially self-contained; in particular, all the necessary details concerning the three models will be provided.

References

[1] A.Ehrenfeucht, G. Rozenberg. Reaction Systems. *Fundamenta Informaticae* 75, 1-18, 2007.

[2] J. Kleijn, M. Koutny. Synchrony and Asynchrony in Membrane Systems. *Lecture Notes in Computer Science* 4361, 66-85, 2006.

[3] J. Kleijn, M. Koutny. Petri Nets with Localities and Testing. *Lecture Notes in Computer Science* 6128, 19-38, 2010.

[4] J. Kleijn, M. Koutny. Petri nets and membrane computing (Chapter 15). In [10], 389-412, 2010.

[5] J. Kleijn, M. Koutny. Membrane systems with Qualitative Evolution Rules. to appear.

[6] J. Kleijn, M. Koutny, G. Rozenberg. Towards a Petri Net Semantics for Membrane Systems. *Lecture Notes in Computer Science* 3850, 292-309, 2006.

[7] J. Kleijn, M. Koutny, G. Rozenberg. Process Semantics for Membrane Systems. *Journal of Automata, Languages and Combinatorics* 11, 321-340, 2006.

[8] J. Kleijn, M. Koutny, G. Rozenberg. Modelling Reaction Systems with Petri Nets. In: *Proceedings of the International Workshop on Biological Processes & Petri Nets (BioPPN-2011) (M. Heiner, H. Matsuno, eds.), CEUR Workshop Proceedings* 724, 36-52, 2011.

[9] G. Păun. *Membrane Computing, An Introduction*. Springer-Verlag, 2002.

[10] G. Păun, G. Rozenberg, A. Salomaa. *The Oxford Handbook of Membrane Computing*. Oxford University Press (2009).

[11] W. Reisig, G. Rozenberg (eds.) Lectures on Petri Nets I and II. *Lecture Notes in Computer Science* 1491 and 1492. Springer-Verlag, 1998.

A Testing Framework for P Systems

Roberto Barbuti
Dipartimento di Informatica, University of Pisa
Largo Pontecorvo 3, 56127 Pisa, Italy
barbuti@di.unipi.it

Diletta Romana Cacciagrano
School of Science and Technology, University of Camerino
Via Madonna delle Carceri 9, 62032 Camerino (MC), Italy
diletta.cacciagrano@unicam.it

Andrea Maggiolo-Schettini
Dipartimento di Informatica, University of Pisa
Largo Pontecorvo 3, 56127 Pisa, Italy
maggiolo@di.unipi.it

Paolo Milazzo
Dipartimento di Informatica, University of Pisa
Largo Pontecorvo 3, 56127 Pisa, Italy
milazzo@di.unipi.it

Luca Tesei
School of Science and Technology, University of Camerino
Via Madonna delle Carceri 9, 62032 Camerino (MC), Italy
luca.tesei@unicam.it

Testing equivalence was originally defined by De Nicola and Hennessy in a process algebraic setting (CCS) with the aim of defining an equivalence relation between processes being less discriminating than bisimulation and with a natural interpretation in the practice of system development. Finite characterizations of the defined preorders and relations led to the possibility of verification by comparing an implementation with a specification in a setting where systems were seen as black boxes with input and output capabilities, thus neglecting internal undetectable behaviours.

In this paper, we start defining a porting of the well-established testing theory into membrane computing, in order to investigate possible benefits in terms of inherited analysis/verification techniques and interesting biological applications. P Algebra, a process algebra for describing P Systems, is used as a natural candidate for the porting since it enjoys the desirable property of being compositional and comes with other observational equivalences already defined and studied.

We consider P Systems with multiple membranes, dissolution, promoters and inhibitors. Notions as *observable* and *test* are conveniently rephrased in the membrane scenario, where they lack as native notions and have a not so obvious mean. At the same time, concepts as promoters, inhibitors, membrane inclusion and dissolution are emphasized and exploited in the attempt of realizing a testing machinery able to formalize several features, which are proper of membranes and, as a consequence, worth being highlighted as basic observables for P Systems. The new testing semantics framework inherits from the original one the ability to define qualitative system properties. Moreover, it results to be suitable also to express *quantitative* aspects, a feature which turns out to be very useful for the biological domain and, at the same time, puts in evidence an expected high expressive power of the framework itself.

1 Introduction

Membrane computing, the research field initiated by Gheorghe Păun [22, 20], aims to define computational models, called *P Systems*, which are inspired by the behaviour and structure of the living cell. Since its introduction, the P System model has been intensively studied and developed: many variants of membrane systems have been proposed and regular collective volumes are annually edited.

The most investigated membrane computing topics are related to the computational power of different variants, their capabilities to solve hard problems, decidability, complexity aspects and hierarchies of classes of languages produced by these devices.

© R. Barbuti, D. R. Cacciagrano, A. Maggiolo-Schettini
P. Milazzo and L. Tesei
All the rights to the paper remain with the authors.

In the last years, there have also been significant developments in using the P systems paradigm for modelling, simulation and formal verification [11]. Although such topics have been exercised to different classes of P Systems [3], testing has been quite neglected in this context.

1.1 Testing and P Systems

Testing P Systems has been so far considered by using certain coverage principles. More often the rule coverage is utilised, by taking into account different contexts. Such contexts - typically grammar, automaton and model checking techniques - are described in depth in [14, 15]:

- (*Grammar-based methods*) In order to test an implementation developed from a P System specification in a grammar-based method, a test set is built, in a black box manner, as a finite set of sequences containing references to rules. Although there are similarities between context-free grammars utilised in grammar testing and basic P Systems, there are also major differences that pose new problems in defining testing methods and strategies to obtain test sets. Some of the difficulties encountered when some grammar-like testing procedures are introduced, are related to the hierarchical compartmentalisation of the P System model, parallel behaviour, communication mechanisms, the lack of a non-terminal alphabet and the use of multisets of objects instead of sets of strings.

- (*Finite-state machine methods*) Finite state machine-based testing is widely used for software testing. It provides very efficient and exhaustive testing strategies and well investigated methods to generate test sets. In this case it is assumed that a model of the system under test is provided in the form of a finite state machine. In the P System model case, such a machine is typically obtained from a partial computation in a P System.

  A different approach uses a special class of state machines, called *X-machines*. Given that the relationships between various classes of P Systems and these machines are well studied [1] and the X-machine-based testing is well developed, standard techniques for generating test sets based on X-machines can be adapted to the case of P Systems [16].

  Specific coverage criteria are defined in the case of finite state machine-based testing. One such criterion, called *transition coverage*, aims to produce a test set in such a way that every single transition of the model is covered.

- (*Model checking-based methods*) The generation of different test sets, according to certain coverage criteria, can be done by utilising some specific algorithms or by applying some tools that indirectly will generate test sets. Such tools, like model checkers, can be used to verify some general properties of a model and when these are not fulfilled then some counter-examples are produced, which act as test sets in certain circumstances.

  In the case of P Systems, an encoding based on a Kripke structure associated with the system is provided for model checkers like NuSMV [17] or SPIN [18]. This relies on certain operations defined in [12] and encapsulates the main features of a P System, including maximal parallelism and communication, but within a finite space of values associated with the objects present in the system.

  The rule coverage principle is expressed by using temporal logics queries available in such contexts. By negating specific coverage criteria, counter-examples are generated.

1.2 Our contribution: a Process Algebra-based testing machinery for P Systems

The community of Process Algebra taught us that the usefulness of formalisms for the description and the analysis of reactive systems is closely related to the underlying notion of *behavioural equivalence*. Such an equivalence should formally identify behaviours that are informally indistinguishable from each other, and at the same time distinguish between behaviours that are informally different. The authors of [4] go toward such a direction, proposing some observational equivalences on a suitable algebraic notation of P Systems.

One way of determining behavioural equivalences is by observing the systems we are interested in, experimenting on them, and drawing conclusions about the behaviour of such systems based on what we see, e.g., testing the system. Such an approach has been formalized in the Process Algebra setting by a suitable testing machinery [13], pivoting on a restricted form of context, called *observer*.

The way to exercise (evaluate) a process on a given observer is done by letting the considered process and the given observer to run *in parallel* and by looking at the computations which the *running test* can perform.

It is worth noting that internal actions of the process under test do not affect, but the case in which they lead to divergence, the satisfaction of the test: they are not observable as input or outputs, thus they cannot be perceived by an external user that is experimenting on the system. This idea is typical of a testing framework: systems are considered black boxes and only observables matter in their comparison. This characteristic is imported in the testing notion introduced in this paper. Internal production and migration of objects in the P system under test will not be seen by the observer: only the objects injected into the system by the observer and the objects that are returned from the system to the observer will matter.

Another typical characteristic of the testing framework, worth underlining, is that the observer must have the ability to force the system under test to follow certain paths among all the possible ones. This is in order to investigate, for instance, what the system can do after a quite specific sequence of inputs, or after a predefined sequence of inputs/outputs. In order to guarantee this possibility, and thus giving the equivalence a discriminating power similar to the one in the original setting, in the testing framework we introduce in this paper we exploit promoters and inhibitors. Without them the intrinsic nature of P system behaviour, in particular maximal parallelism, would have prevented this central feature, thus invalidating the porting.

The characteristics discussed above are central in the idea of testing equivalence. Bisimulation-based equivalences, even the weak one, are highly discriminating and do not reflect a practical view of "testing" a system: usually, and *this is always the case for biological systems*, the whole internal structure/dynamics of the system is not known, but it is required to check bisimulation. The only way to study such systems is to interact with them and analyse what can be observed from experiments, with the means that are available. Along this idea, we start with this work the definition of a testing framework with the characteristics above, giving initial theoretical results and some simple examples of tests, without any particular biological impact. However, we intend as future work to investigate and exploit the analogy of the defined notion of testing with biological experiments in order to give more evidence of the biological relevance of the work. We can devise, for instance, techniques for experimental planning that could be of great interest for experimental biologists, along the same line of the techniques proposed in [2].

In [13], different equivalence relations (e.g., *may* and *must* testing equivalences) between systems are defined. Two systems are considered equivalent if they pass exactly the same set of observers. Such equivalences are further broken down into preorder relations on systems, i.e., relations that are reflexive and transitive (though not necessarily symmetric). Formally, given a process P and an observer o,

- P **may** o means that there exists a successful computation from the test $P\,|\,o$ (where $|$ is the parallel operator, and successful means that there is a state where the special action ω is enabled);
- P **must** o means that every maximal computation from the test $P\,|\,o$ is successful;
- The preorder $P \leq_{sat} Q$ means that for any observer o, $P\,\mathbf{sat}\,o$ implies $Q\,\mathbf{sat}\,o$, where **sat** denotes **may** or **must**;
- The equivalence $P \approx_{sat} Q$ means $P \leq_{sat} Q$ and $Q \leq_{sat} P$.

In [23] and in [19] a new testing semantics was proposed to incorporate the fairness notion: the *fair*-testing (aka *should*-testing) semantics. In contrast to the classical *must*-testing (semantics), *fair*-testing abstracts from certain divergences, e.g., infinite loops of τ (invisible) actions. This is achieved by stating that the observer o is satisfied if success always remains within reach in the system under observer. In other words, P **fair** o holds if in every maximal computation from $P\,|\,o$ every state can lead to success after finitely many interactions.

On the basis of *P Algebra*, the algebraic notation of P Systems introduced in [4] (see Section 2), we adapt for P Systems the testing machinery defined in [13] (see Section 3).

We introduce the concept of *context* in case of P Systems expressed as P Algebra terms. Using this concept, we then define what we consider an *observer*, which is again a P Algebra term with certain characteristics. This leads naturally to the definition of computations of a *running test*, i.e. a tested system running together with an observer. Then, following the classical definition of [13], we define the success of a running test in the two well-known versions of *may* and *must*[1]. Finally, the testing preorders are introduced, together with the induced equivalence relations. More in detail:

- An observer consists of a membrane structure in which the skin membrane contains several membranes (with possibly sub-membranes) one of which is a hole, i.e., a place where another fully defined P System can be placed and run. The skin membrane of this tested P System instantiates the hole membrane and becomes a full component of the running test.
- P Systems that are observers are distinguished from P Systems that are normal, testable processes similarly to the classical testing approach, where a particular action, called ω, is used to denote the success of a test and, if the running test is able to perform this action, then the computation under consideration is a successful one. Similarly, in our framework this is easily translated introducing a fresh, particular object ω that, when sent out of the skin membrane of the running test, denotes the success of the computation.
- As usual in testing frameworks, we consider only the behaviours of the running test in which no output is produced (this corresponds to considering only invisible action (e.g., τ) computations in a CCS-like Process Algebra). This is needed to explore all possible behaviours of the tested system while running together with the observer.

An important result consists of the fact that, differently from the original testing semantics framework, the one proposed here for P Systems results to be suitable to define both qualitative and (above all) *quantitative* system properties. This is mainly because the formalism of P system is expressive enough to express both qualitative and quantitative aspects. However, these features are crucial for the biological domain. In Section 4, we put quantitative capabilities in evidence defining examples of quantitative tests, both concerning time and number of individuals, of a system modelling a population of individuals that

[1]In this paper we do not consider *fair* testing.

can reproduce both sexually and asexually. Note that these examples are to be intended only as explanation of the concepts introduced in the paper and not as examples of verifications in the model-checking style. The main direction of continuing this work towards verification is to find finite characterizations of the resulting testing equivalence, possibly with respect to suitable classes of observers, and thus using them to compare the expected behaviour of a finite state system with its actual behaviour.

2 Background

We first briefly recall the definition of P Systems [22, 20]. Then, we give the definition of the syntax and the semantics of the P Algebra as it was presented in [4], where a class of P Systems including rule promoters and inhibitors [10] was considered. The original formulation of the P Algebra, without promoters and inhibitors, can be found in [7, 6] that we refer as a more detailed presentation of the semantics.

2.1 P Systems with Promoters and Inhibitors

A P System consists of a *hierarchy of membranes*, each of them containing a multiset of *objects*, representing molecules, a set of *evolution rules*, representing chemical reactions, and possibly other membranes. Each evolution rule consists of two multisets of objects, describing the reactants and the products of the chemical reaction. A rule in a membrane can be applied only to objects in the same membrane. Some objects produced by the rule remain in the same membrane, others are sent *out* of the membrane, others are sent *into* the inner membranes (assumed to exist) which are identified by their labels.

In the original definition of P Systems, rules are applied with *maximal parallelism*, namely it cannot happen that a rule is not applied when the objects needed for its triggering are available. Here, we assume that at each step at least one evolution rule in the whole system is applied, and also that more than one rule and several occurrences of the same rule can be applied at the same step (to different objects). In other words, we assume that at each step a multiset of evolution rule instances is non-deterministically chosen and applied in each membrane, such that in the whole system at least one rule is applied. This is a general form of parallelism that is better suited than the maximal one to describe events in biological systems.

In P Systems with promoters and inhibitors an evolution rule in a membrane may have some *promoters* and some *inhibitors*. Promoters are objects that are required to be present and inhibitors are objects that are required to be absent in the membrane m in order to enable the application of the rule. Promoters will be denoted simply as objects, namely $a,b,c,\ldots$, while inhibitors will be denoted as objects preceded by a negation symbol, namely $\neg a, \neg b, \neg c, \ldots$.

We denote with $\mathscr{D}_V$ the set of all possible promoters and inhibitors symbols that can be obtained from an alphabet V, namely $\mathscr{D}_V = V \cup \neg V$. Given a set of promoter and inhibitor symbols D, we denote with D^+ and D^- the sets of objects containing all the objects occurring in D as promoters and all the objects occurring in D as inhibitors, respectively. We remark that D^+ and D^- are sets of objects, hence elements on D^- will not be preceded by $\neg$. Moreover, with $\neg D$ we denote the set obtained by transforming each promoter in D into an inhibitor and vice versa. As an example, if $D = \{a, \neg b, \neg c, d\}$ we have $D^+ = \{a,d\}$, $D^- = \{b,c\}$ and $\neg D = \{\neg a, b, c, \neg d\}$.

We assume that all evolution rules have the following form, where $u, v_h, v_o, v_1, \ldots, v_n$ are multisets of objects, $\{l_1, \ldots, l_n\}$ is a set of membrane labels in $\mathbb{N}$, and D is a set of promoters and inhibitors:

$$u \to (v_h, here)(v_o, out)(v_1, in_{l_1}) \ldots (v_n, in_{l_n})|_D \, .$$

A rule can be applied only if requirements expressed by D are satisfied. When a rule is applied, the multiset of objects u is replaced by v_h, multiset v_o is sent to the parent membrane, and each v_i is sent to inner membrane l_i. Promoters are not consumed by the application of the corresponding evolution rule and a single occurrence of a promoter may enable the application of more than one rule in each evolution step. Similarly, a single occurrence of an inhibitor forbids the application of all the evolution rules in which it appears. We assume that the set of promoters and inhibitors D of an evolution rule does not contain the same object both as a promoter and as an inhibitor, namely $D^+ \cap D^- = \varnothing$, and that consumed objects u are not mentioned among inhibitors, namely $u \cap D^- = \varnothing$.

Definition 1. *A* P System Π *is a tuple* $(V, \mu, w_1, \ldots, w_n, R_1, \ldots, R_n)$ *where:*

- V *is an* alphabet *whose elements are called* objects*;*
- $\mu \subset \mathbb{N} \times \mathbb{N}$ *is a* membrane structure*, such that* $(l_1, l_2) \in \mu$ *denotes that the membrane labelled by* l_2 *is contained in the membrane labelled by* l_1*;*
- w_j *with* $1 \leq j \leq n$ *are multisets of objects in* V *associated with the membranes* $1, \ldots, n$ *of* μ*;*
- R_j *with* $1 \leq j \leq n$ *are finite sets of* evolution rules *associated with the membranes* $1, \ldots, n$ *of* μ.

2.2 The P Algebra: Syntax and Semantics

In this section we recall the *P Algebra*, the algebraic notation of P Systems we have introduced in [7], with slight modifications introduced in [4]. We assume V to be an alphabet of objects and we adopt the usual string notation to represent multisets of objects in V. For instance, to represent $\{a, a, b, b, c\}$ we may write either $aabbc$, or a^2b^2c, or $(ab)^2c$. We denote with $\mathsf{Set}(u)$ the support of multiset u, namely the set of all the objects occurring in u. We denote multiset (and set) union as string concatenation, hence we write u_1u_2 for $u_1 \cup u_2$. Moreover, we shall write $u(a)$ for the number of occurrences of a in multiset u. For the sake of legibility, we shall write $u \to v_h v_o \{v_{l_i}\}|_D$ for the generic evolution rule $u \to (v_h, here)(v_o, out)(v_1, in_{l_1}) \ldots (v_n, in_{l_n})|_D$.

The abstract syntax of the P Algebra is defined as follows.

Definition 2 (P Algebra)**.** *The abstract syntax of* membrane contents c*,* membranes m*, and* membrane systems ms *is given by the following grammar, where* l *ranges over* $\mathbb{N}$ *and* a *over* V*:*

$$\begin{aligned} c &::= (\varnothing, \varnothing) \;\big|\; (u \to v_h v_o \{v_{l_i}\}|_D, \varnothing) \;\big|\; (\varnothing, a) \;\big|\; c \cup c \\ m &::= [_l\, c\,]_l \\ ms &::= ms \mid ms \;\big|\; \mu(m, ms) \;\big|\; F(m) \end{aligned}$$

A membrane content c represents a pair $(\mathscr{R}, u)$, where $\mathscr{R}$ is a set of evolution rules and u is a multiset of objects. A membrane content is obtained through the union operation $\_\cup\_$ from constants representing single evolution rules and single objects, and can be plugged into a membrane with label l by means of the operation $[_l\, \_\,]_l$ of membranes m. Hence, given a membrane content c representing the pair $(\mathscr{R}, u)$ and $l \in \mathbb{N}$, $[_l\, c\,]_l$ represents the membrane having l as label, $\mathscr{R}$ as evolution rules and u as objects.

Membrane systems ms have the following meaning: $ms_1 \mid ms_2$ represents the juxtaposition of ms_1 and ms_2, $\mu(m, ms)$ represents the hierarchical composition of m and ms, namely the containment of ms in m, and $F(m)$ represents a *flat membrane*, namely it states that m does not contain any child membrane. Juxtaposition is used to group sibling membranes, namely membranes all having the same parent in a membrane structure. This operation allows hierarchical composition μ to be defined as a binary operator on a single membrane (the parent) and a juxtaposition of membranes (all the children).

Note that every P System has a corresponding membrane system in the P Algebra, and that there exist membrane systems which do not correspond to any P System.

In what follows we will often write $[\![_l\, c]\!]_l$ for $F([_l\, c]_l)$. We shall also often write $(\mathscr{R},u)$ where $\mathscr{R} = \{r_1,\dots,r_n\}$ is a set of rules and $u = o_1 \dots o_m$ a multiset of objects rather than $(r_1,\varnothing)\cup\dots\cup(r_n,\varnothing)\cup(\varnothing,o_1)\cup\dots\cup(\varnothing,o_m)$. Moreover, we shall often omit parentheses around membrane contents.

The semantics of the P Algebra is given as a labelled transition system (LTS). The labels of the LTS can be of the following forms:

- $(u,v,v',D,I,O^{\uparrow},O^{\downarrow})$, describing a computation step performed by a membrane content c, where:
  - u is the multiset of objects consumed by the application of evolution rules in c, as it results from the composition, by means of $\_\cup\_$, of the constants representing these evolution rules.
  - v is the multiset of objects in c offered for the application of the evolution rules, as it results from the composition, by means of $\_\cup\_$, of the constants representing these objects. When operation $[_l\, \_]_l$ is applied to c, it is required that v and u coincide.
  - v' is the multiset of objects in c that are not used to apply any evolution rule and, therefore, are not consumed, as it results from the composition, by means of $\_\cup\_$, of the constants representing these objects.
  - D is a set of promoters and inhibitors required to be present and absent, respectively, by the application of evolution rules in c. More precisely, D^- contains all the inhibitors of the applied evolution rules in c, whereas D^+ is a subset of the promoters of those rules. Such a subset contains only those objects that are not present in the multiset of objects of c.
  - I is the multiset of objects received as input from the parent membrane and from the child membranes.
  - $O^{\uparrow}$ is the multiset of objects sent as an output to the parent membrane.
  - $O^{\downarrow}$ is a set of pairs (l_i,v_{l_i}) describing the multiset of objects sent as an output to each child membrane l_i.
- $(\mathscr{I}^{\downarrow},I^{\uparrow},O^{\uparrow},O^{\downarrow},app)$, describing a computation step performed by a membrane m, where: $\mathscr{I}^{\downarrow}$ is a set containing only the pair (l,I) where l is the label of m and I is the multiset of objects received by m as input from the parent membrane, $I^{\uparrow}$ is the multisets of objects received from the child membranes of m, and $O^{\uparrow}$ and $O^{\downarrow}$ are as in the previous case. Finally, $app \in \{0,1\}$ is equal to 0 if no rule has been applied in m in the described computation step, and it is equal to 1 otherwise.
- $(\mathscr{I}^{\downarrow},O^{\uparrow},app)$, describing a computation step performed by a membrane system ms, where $\mathscr{I}^{\downarrow}$, $O^{\uparrow}$ and app are as in the previous cases.

For the sake of legibility, in transitions with labels of the first form we shall write the first four elements of the label under the arrow denoting the transition and the other elements over the arrow. Now, LTS transitions are defined through SOS rules [21]. We give here a very short explanation of such rules. Please, refer to [7] for more details.

We start by giving in Fig. 1 the transition rules for membrane contents. Rule $(mc1_n)$ describes n simultaneous applications of an evolution rule for any $n \in \mathbb{N}$. Rule $(mc2)$ describes the case in which an evolution rule is not applied because a subset D' of the promoters and inhibitors in D it requires to be present and absent, respectively, are assumed not to satisfy the requirements. Rules $(mc3),(mc4)$ and $(mc5)$ describe the transitions performed by membrane contents consisting of a single object and the transitions performed by an empty membrane content.

Rule $(u1)$ describes the behaviour of a union of membrane contents. In this transition rule we use some auxiliary notations. Given two sets $O_1^{\downarrow}$ and $O_2^{\downarrow}$ representing two outputs to inner membranes,

$$\frac{I \in V^* \qquad n \in \mathbb{N}}{(u \to v_h v_o\{v_{l_i}\}|_D, \varnothing) \xrightarrow[u^n,\varnothing,\varnothing,D]{I,v_o^n,\{(l_i,v_{l_i}^n)\}} (u \to v_h v_o\{v_{l_i}\}|_D, I v_h^n)} \qquad (mc1_n)$$

$$\frac{I \in V^* \qquad D' \subseteq \neg D \qquad D' \neq \varnothing}{(u \to v_h v_o\{v_{l_i}\}|_D, \varnothing) \xrightarrow[\varnothing,\varnothing,\varnothing,D']{I,\varnothing,\varnothing} (u \to v_h v_o\{v_{l_i}\}|_D, I)} \qquad (mc2)$$

$$\frac{I \in V^*}{(\varnothing, a) \xrightarrow[\varnothing,a,\varnothing,\varnothing]{I,\varnothing,\varnothing} (\varnothing, I)} \qquad (mc3) \qquad\qquad \frac{I \in V^*}{(\varnothing, a) \xrightarrow[\varnothing,\varnothing,a,\varnothing]{I,\varnothing,\varnothing} (\varnothing, Ia)} \qquad (mc4)$$

$$\frac{I \in V^*}{(\varnothing, \varnothing) \xrightarrow[\varnothing,\varnothing,\varnothing,\varnothing]{I,\varnothing,\varnothing} (\varnothing, I)} \qquad (mc5)$$

$$\frac{\begin{array}{c} x_1 \xrightarrow[u_1,v_1,v_1',D_1]{I_1,O_1^{\uparrow},O_1^{\downarrow}} y_1 \qquad x_2 \xrightarrow[u_2,v_2,v_2',D_2]{I_2,O_2^{\uparrow},O_2^{\downarrow}} y_2 \\ (D_1^- \cup D_2^-) \cap \mathsf{Set}(v_1 v_1' v_2 v_2') = \varnothing \qquad D_1 \cap \neg D_2 = \varnothing \qquad D = (D_1 D_2) \setminus \mathsf{Set}(v_1 v_1' v_2 v_2') \end{array}}{x_1 \cup x_2 \xrightarrow[u_1 u_2, v_1 v_2, v_1' v_2', D]{I_1 I_2, O_1^{\uparrow} O_2^{\uparrow}, O_1^{\downarrow} \cup_{\mathbb{N}} O_2^{\downarrow}} y_1 \cup y_2} \qquad (u1)$$

Figure 1: Transition rules for membrane contents and unions of membrane contents.

we write $O_1^{\downarrow} \cup_{\mathbb{N}} O_2^{\downarrow}$ to denote the set $\{(l,uv) \mid (l,u) \in O_1^{\downarrow} \wedge (l,v) \in O_2^{\downarrow}\} \cup \{(l,u) \mid (l,u) \in O_1^{\downarrow} \wedge \nexists v.(l,v) \in O_2^{\downarrow}\} \cup \{(l,v) \mid (l,v) \in O_2^{\downarrow} \wedge \nexists u.(l,u) \in O_1^{\downarrow}\}$.

In Fig. 2 we give transition rules for individual membranes, juxtaposition and hierarchical composition. Note that from the transition label of the membrane content we have no information about the objects that have been produced by the applied evolution rules. Rules $(m1)$ and $(m2)$ describe the transitions performed by a membrane with label l. In particular, $(m1)$ describes the case in which no objects are received as an input from the external membrane, while $(m2)$ describes the case in which a multiset of objects $I_1 \neq \varnothing$ is received. In these rules app is set to zero if no evolution rule is applied $(u = \varnothing)$, and it is set to one if at least one rule is applied $(u \neq \varnothing)$. Rule $(fm1)$ allows us to infer the behaviour of a flat membrane $[\![_l\, c]\!]_l = F([_l\, c]_l)$ from the behaviour of membrane $[_l\, c]_l$. Rule $(jux1)$ allows us to infer the behaviour of a juxtaposition of two membrane structures from the behaviours of the two structures. Finally, rule $(h1)$ describes the behaviour of a hierarchical composition of membranes. In this rule we assume $\doteq$ to be an equivalence relation on sets of pairs (l,u) with $l \in \mathbb{N}$ and $u \in V^*$, such that, given two such sets $\mathscr{I}_1$ and $\mathscr{I}_2$, then $\mathscr{I}_1 \doteq \mathscr{I}_2$ holds if and only if $(\mathscr{I}_1 \setminus \{(l,\varnothing) \mid l \in \mathbb{N}\}) = (\mathscr{I}_2 \setminus \{(l,\varnothing) \mid l \in \mathbb{N}\})$. In the last two rules app is set to one if at least one between app_1 and app_2 is equal to one, namely $app = max(app_1, app_2)$. This means that at least one rule has been applied in the whole composition.

We conclude by defining a *system trace* as a sequence of internal information given by an execution of a P Algebra term. We assume that the system can send objects out of the outmost membrane, but cannot receive objects from outside. This requirement corresponds to the fact that in a P System objects cannot be received by the outmost membrane from the external environment. Note that executions containing steps in which no rule is applied, namely those with 0 as last element of the label, are not considered.

Definition 3 (Trace). *A trace of a membrane system ms with alphabet V is a (possibly infinite) sequence w of outputs such that, for any $O_i^{\uparrow}$ and ms_i with $i \in \mathbb{N}^+$*

$$\frac{x \xrightarrow[u,u,v',D]{I,O^{\uparrow},O^{\downarrow}} y \qquad app = \begin{cases} 0 & \text{if } u = \varnothing \\ 1 & \text{otherwise} \end{cases} \qquad D^{+} = \varnothing}{[_l\, x]_l \xrightarrow{\varnothing,I,O^{\uparrow},O^{\downarrow},app} [_l\, y]_l} \quad (m1)$$

$$\frac{x \xrightarrow[u,u,v',D]{I_1I_2,O^{\uparrow},O^{\downarrow}} y \qquad app = \begin{cases} 0 & \text{if } u = \varnothing \\ 1 & \text{otherwise} \end{cases} \qquad D^{+} = \varnothing \qquad I_1 \neq \varnothing}{[_l\, x]_l \xrightarrow{\{(l,I_1)\},I_2,O^{\uparrow},O^{\downarrow},app} [_l\, y]_l} \quad (m2)$$

$$\frac{x \xrightarrow{\mathscr{I}^{\downarrow},\varnothing,O^{\uparrow},\varnothing,app} y}{F(x) \xrightarrow{\mathscr{I}^{\downarrow},O^{\uparrow},app} F(y)} \quad (fm1) \qquad \qquad \frac{x_1 \xrightarrow{\mathscr{I}_1,O_1^{\uparrow},app_1} y_1 \qquad x_2 \xrightarrow{\mathscr{I}_2,O_2^{\uparrow},app_2} y_2}{x_1|x_2 \xrightarrow{\mathscr{I}_1\mathscr{I}_2,O_1^{\uparrow}O_2^{\uparrow},max(app_1,app_2)} y_1|y_2} \quad (jux1)$$

$$\frac{x_1 \xrightarrow{\mathscr{I}_1^{\downarrow},I_1^{\uparrow},O_1^{\uparrow},O_1^{\downarrow},app_1} y_1 \qquad x_2 \xrightarrow{\mathscr{I}_2,O_2^{\uparrow},app_2} y_2 \qquad O_1^{\downarrow} \bumpeq \mathscr{I}_2 \qquad O_2^{\uparrow} = I_1^{\uparrow}}{\mu(x_1,x_2) \xrightarrow{\mathscr{I}_1^{\downarrow},O_1^{\uparrow},max(app_1,app_2)} \mu(y_1,y_2)} \quad (h1)$$

Figure 2: Rules for individual membranes and hierarchical composition of membranes

- $w = O_1^{\uparrow}O_2^{\uparrow}\cdots O_n^{\uparrow}$ *and* $ms \xrightarrow{\varnothing,O_1^{\uparrow},1} ms_1 \xrightarrow{\varnothing,O_2^{\uparrow},1} \dots \xrightarrow{\varnothing,O_n^{\uparrow},1} ms_n \not\xrightarrow{\varnothing,O^{\uparrow},1}$

  or

- $w = O_1^{\uparrow}O_2^{\uparrow}\cdots O_n^{\uparrow}\cdots$ *and* $ms \xrightarrow{\varnothing,O_1^{\uparrow},1} ms_1 \xrightarrow{\varnothing,O_2^{\uparrow},1} ms_2 \xrightarrow{\varnothing,O_3^{\uparrow},1} \dots\dots$

We denote with $\mathscr{T}$ *the set of all traces.*

3 Testing framework

We first introduce the concept of *context* in case of P Systems expressed as P Algebra terms. Using this concept, we then define what we consider an *observer*, which is again a P Algebra term with certain characteristics. This leads naturally to the definition of computations of a running test, i.e. a tested system running together with an observer. Then, following the classical definition of [13], we define the success of a test in the two well-known versions of *may* and *must*. Finally, the testing preorder is introduced, together with the induced equivalence relation.

3.1 Contexts and test satisfaction

At a first look, a natural candidate for context of a P System is another P System, which we call *observer*, consisting of a membrane structure in which the skin membrane contains several membranes (with possibly sub-membranes) one of which is a "hole", i.e., a place where another fully defined P System can be placed and run. The skin membrane of this *tested* P System instantiates[2] the "hole" membrane and becomes a full component of the *running test*.

[2]This instantiation process may require some trivial modifications of the tested P System, such as α-conversion of the numbers assigned to the skin and inner membranes.

However, in [6], a result regarding flattening P Systems in P Algebra is presented. A similar result can be found in [8]. The flattening process of [6] reduces any P System, specified in P Algebra with promoters and inhibitors, into a flat one (i.e., with no internal membranes) that is bisimilar to the original one. The notion of bisimulation is the one, based on computation steps, defined in [7]. These results suggested us to simplify, without loss of generality, the notion of context we are defining. For this reason, instead of considering an observer of the form $\mu(m, ms_1 \mid ms_2 \mid \cdots \mid ms_n \mid \Box)$, we will always consider the equivalent observer $\mu(m', \Box)$ where m' results from the flattening process of $\mu(m, ms_1 \mid ms_2 \mid \cdots \mid ms_n)$.

Another ingredient of the testing framework is needed to distinguish formally P Systems that are *observers* from P Systems that are normal, *testable* processes. In classical testing a particular action, usually called ω, is used to denote the "success" of a test, that is to say, if the *running test* is able to perform this action then the computation under consideration is a successful one. This easily translates in our framework: we introduction a fresh, particular object, $\omega \notin V$ that, when sent out of the skin membrane of the running test, denotes the "success" of the computation.

Definition 4 (Observer)**.** *Let V be the alphabet of objects and let $\omega \notin V$ be a particular object. An* observer system*, or simply a* test*, is a P Algebra term of the form $\mu(m, \Box)$ where m is the skin membrane $[_1\, c\,]_1$, $\Box$ is an unspecified membrane ms numbered 2 and each rule of m is of the form $u \rightarrow v_h \omega_o \{v_{l_2}\}|_D$. In other words, m communicates with its only child membrane 2 and can send out only ω objects.*

Definition 5 (Running test)**.** *Let V be the alphabet of objects and let $\omega \notin V$ be a particular object. Let $\mu(m, \Box)$ be an observer system and let ms be term in P Algebra denoting a closed membrane (i.e., of the form $F(-)$ or $\mu(-,-)$). The* running test *is the P Algebra term $\mu(m, ms')$ where ms' is the term ms in which the former skin membrane (numbered 1) is re-labelled in 2 and the other internal membranes numbers are α-converted in order not to collide with 1 and 2 (the re-labelling of the membrane names of course implies also applying the substitutions to all the references to the membrane names in the term.)*

Definition 6 (Computations)**.** *Let $\mu(m, ms)$ be a running test. A computation c of $\mu(m, ms)$ is any sequence of the form:*

- *c is finite: $\mu(m, ms) = ms_0 \xrightarrow{\varnothing,\varnothing,1} ms_1 \xrightarrow{\varnothing,\varnothing,1} \ldots \xrightarrow{\varnothing,\varnothing,1} ms_n \overset{\varnothing,\varnothing,1}{\not\longrightarrow}$*

  or

- *c is infinite: $\mu(m, ms) = ms_0 \xrightarrow{\varnothing,\varnothing,1} ms_1 \xrightarrow{\varnothing,\varnothing,1} ms_2 \xrightarrow{\varnothing,\varnothing,1} \ldots.$*

A computation is called successful *if there are $k \in \mathbb{N}$ and $n \in \mathbb{N}^+$ such that $ms_k \xrightarrow{\varnothing,\{\omega^n\},1} ms_{k+1}$, i.e., at least a success symbol can be sent out of the skin membrane along the run. A membrane like ms_k, from which a transition can be taken that sends out at least one ω object is called* success *membrane. We may also write $ms_k \xrightarrow{\omega}$ to indicate that ms_k is a success membrane.*

Note that, as usual in testing frameworks, we consider only the behaviours of the running test in which no output is produced (this corresponds to considering only τ computations in a CCS-like Process Algebra). This is needed to explore all possible behaviours of the tested system while running together with the observer. Note that an unsuccessful computation, i.e. one with no success state along it, can be either finite or infinite. In testing theories *divergence* must have a delicate treatment because it can lead to different testing preorders [13, 9]. For simplicity we port here the original formulation of [13]. Thus, we define a unary predicate $ms \uparrow$ meaning that, with respect to success, the "state" ms along a computation is "underdefined" because it leads to a divergent computation.

Definition 7 (Divergence)**.** *Let c be a computation of a running test $\mu(m,ms)$. c is* divergent*, which we denote by $c \Uparrow$ iff:*

- *c is unsuccessful, or*
- *c contains a membrane ms such that $ms \uparrow$ and there is not a success membrane ms' preceding ms in c.*

We now have all the ingredients to define the success of a test. Following the classical approach, such a definition comes into version: a *may* satisfaction, weaker, and a *must* satisfaction, stronger.

Definition 8 (Satisfaction of a test)**.** *Let $\mu(m,\Box)$ be a test and ms be a closed membrane. Then we say:*

- *ms* **may** *$\mu(m,\Box)$ iff there exists a computation c of $\mu(m,ms)$ and $k \in \mathbb{N}$ such that*

$$c : \mu(m,ms) = ms_0 \xrightarrow{\varnothing,\varnothing,1} ms_1 \xrightarrow{\varnothing,\varnothing,1} \cdots \xrightarrow{\varnothing,\varnothing,1} ms_k$$

  and $ms_k \xrightarrow{\omega}$.

- *ms* **must** *$\mu(m,\Box)$ iff for each computation c of $\mu(m,ms)$,*

$$c : \mu(m,ms) = ms_0 \xrightarrow{\varnothing,\varnothing,1} ms_1 \xrightarrow{\varnothing,\varnothing,1} ms_2 \xrightarrow{\varnothing,\varnothing,1} \cdots$$

  the following conditions hold:

  (i) there is $n \in \mathbb{N}$ such that $ms_n \xrightarrow{\omega}$

  (ii) if there is $k \in \mathbb{N}$ such that $ms_k \uparrow$, then there exists $k' \leq k$ such that $ms_{k'} \xrightarrow{\omega}$.

Note that, in case of *must* satisfaction, computations can be infinite, but it is required that a success state is present just at the beginning of divergence, or before.

3.2 Testing preorders and testing equivalences

Using the definitions of satisfaction of a test, we naturally derive preorders between membrane systems.

Definition 9. *Let ms_1 and ms_2 two closed membrane systems. We define two relations $\sqsubseteq_T^m$ and $\sqsubseteq_T$ between closed membrane systems as follows:*

- *$ms_1 \sqsubseteq_T^m ms_2$* ***iff*** *for each observer $\mu(m,\Box)$, ms_1* **may** *$\mu(m,\Box) \Rightarrow ms_2$* **may** *$\mu(m,\Box)$*
- *$ms_1 \sqsubseteq_T ms_2$* ***iff*** *for each observer $\mu(m,\Box)$, ms_1* **must** *$\mu(m,\Box) \Rightarrow ms_2$* **must** *$\mu(m,\Box)$*

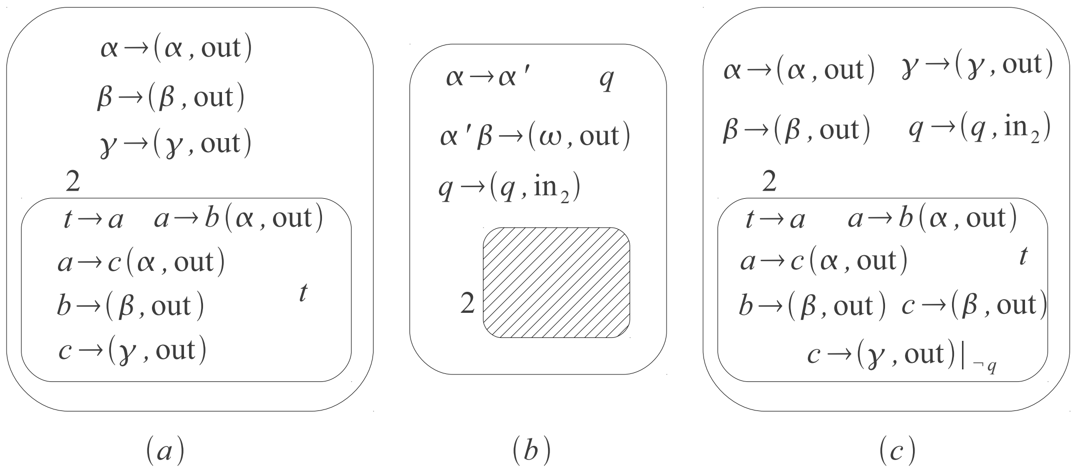
Proposition 1. *The relations $\sqsubseteq_T^m$ and $\sqsubseteq_T$ are preorders.*

Definition 10 (Testing equivalence)**.** *We say that ms_1 is* **may** *testing equivalent to ms_2, and write $ms_1 \approx_T^m ms_2$, iff $ms_1 \sqsubseteq_T^m ms_2$ and $ms_2 \sqsubseteq_T^m ms_1$.*
Analogously, we say that ms_1 is **must** *testing equivalent to ms_2, and write $ms_1 \approx_T ms_2$, iff $ms_1 \sqsubseteq_T ms_2$ and $ms_2 \sqsubseteq_T ms_1$.*

Since the defined relations are kernels of preorders, it is easy to conclude that they are equivalence relations.

In [7] some equivalence relations are defined between the terms of P Algebra. Among them, we consider bisimulation, denoted by $\approx$, and trace equivalence, denoted by $\approx_{Tr}$. We show that the relationships between these two equivalences with the *must* testing equivalence are the same that hold when classical process calculi, as CCS, are considered [13].

Proposition 2. *Let $\approx, \approx_T$ and $\approx_{Tr}$ as above. Then, $\approx \subsetneq \approx_T \subsetneq \approx_{Tr}$.*



$\alpha \to (\alpha, \text{out})$
$\beta \to (\beta, \text{out})$
$\gamma \to (\gamma, \text{out})$
2
$t \to a$ $a \to b(\alpha, \text{out})$
$a \to c(\alpha, \text{out})$
$b \to (\beta, \text{out})$
t
$c \to (\gamma, \text{out})$
$\alpha \to \alpha'$ q
$\alpha' \beta \to (\omega, \text{out})$
$q \to (q, \text{in}_2)$
2
$\alpha \to (\alpha, \text{out})$ $\gamma \to (\gamma, \text{out})$
$\beta \to (\beta, \text{out})$ $q \to (q, \text{in}_2)$
2
$t \to a$ $a \to b(\alpha, \text{out})$
$a \to c(\alpha, \text{out})$ t
$b \to (\beta, \text{out})$ $c \to (\beta, \text{out})|$
$c \to (\gamma, \text{out})|_{\neg q}$
(a)
(b)
(c)

Figure 3: Two membrane systems, (a) and (c), that are trace equivalent, but not test equivalent, together with the observer (b) that distinguishes them.

Proof. The two set inclusions can be proved rephrasing the argument used for CCS [13]. Regarding the strictness of the inclusions, Figure 3 shows two systems that are trace equivalent, but can be distinguished by the shown observer. Not that both systems perform the set of traces $\{\varnothing(\alpha)(\beta), \varnothing(\alpha)(\gamma)\}$. However, considering the *must* testing, the observer shown in Figure 3(b) is not satisfied by the system (a) because in one computation, after α, only γ can be produced, thus that computation is unsuccessful. Conversely, for the other system (c) all computations are successful. Note that inhibitors and promoters are used in this settings to select only specific paths in the system. In the classical setting of testing for synchronous calculi this role is played by the parallel operator together with the restriction on observing only τ-computations.

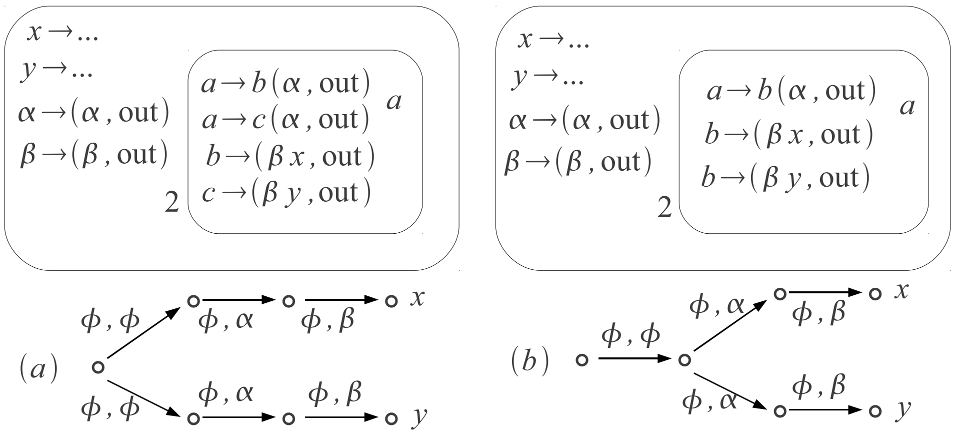
Figure 4 shows two systems that are test equivalent, but not bisimilar. Below each system the graph of possible transitions[3] is depicted, in order to show that bisimulation does not hold. □

4 Example of testing scenario

In this section we model a population of individuals that can reproduce both sexually and asexually. We define different observers to show the expressiveness of the defined testing framework. It is worth noting that *quantitative* aspects of systems can be easily expressed. Note that the defined observers are very specific and are intended to only show the capabilities of the framework introduced above. In particular, they should not be intended as examples of verification, because this analysis is to be done by checking the testing equivalence between the system and its expected behaviour, modelled as a simpler "specification" P system.

Most animal species use sexual reproduction to produce offspring, while a minority of species reproduce asexually by producing clones of the mother. Both strategies have advantages and weaknesses.

[3]For the sake of legibility, we omit the third field in the transition labels.



$x \to ...$
$y \to ...$
$\alpha \to (\alpha, \text{out})$
$\beta \to (\beta, \text{out})$
2
$a \to b(\alpha, \text{out})$
$a \to c(\alpha, \text{out})$
$b \to (\beta x, \text{out})$
$c \to (\beta y, \text{out})$
a
$x \to ...$
$y \to ...$
$\alpha \to (\alpha, \text{out})$
$\beta \to (\beta, \text{out})$
2
$a \to b(\alpha, \text{out})$
$b \to (\beta x, \text{out})$
$b \to (\beta y, \text{out})$
a
(a)
ϕ, ϕ
ϕ, α
ϕ, β
x
ϕ, ϕ
ϕ, α
ϕ, β
y
(b)
ϕ, ϕ
ϕ, α
ϕ, β
x
ϕ, α
ϕ, β
y

Figure 4: Two membrane systems that are test equivalent, but not bisimilar, together with their graph of transitions.

During sexual reproduction genes from two individuals are combined in the offspring that receives genetic material from both parents involving, in diploid populations, recombination among genes. Recombination can break up favourable sets of genes accumulated by selection. Moreover, asexual populations composed by only females can reproduce twice as fast in each generation than sexual populations, because there is no need to produce males for ongoing reproduction.

Despite its considerable cost, sexual reproduction it is still by far the most frequent mode of reproduction in vertebrates. Asexual reproduction has only been described in less than 0.1% of vertebrate species. In general, it is assumed that sexually reproducing populations harbour more genetic variation than asexually reproducing populations, and a high level of genetic variation allows perpetual adaptation to changing environments.

Particularly, populations in heterogeneous habitats, threatened by various parasites or under strong competition, have been shown to have greater genetic variation. For the above reasons a variety of species (essentially among invertebrates) adopted a mixed strategy which tries to combine the advantages of both methods [24].

In this example we model a simple organism able to reproduce either sexually or asexually. We consider the individual of the species as diploid with only a locus (gene), thus each genotype is composed by a pair (a_1, a_2) of alleles which the two chromosomes have for the gene. Moreover, we consider the sex of individuals, thus each of them is represented by a pair of alleles together with the symbol, f or m, of the sex. The rules controlling the evolution of the population are *reproduction rules*, either sexual reproduction rules or asexual, and *death rules*. Each rule has an inhibitor; when the inhibitor is present the rule cannot be applied.

Consider a set of alleles (values for the single gene) of k elements $\{v_1, v_2, \dots, v_k\}$. In the following a_1, a_2, a_3, a_4 belong to $\{v_1, v_2, \dots, v_k\}$. The reproduction rules are the following:

1. $a_1a_2m \ \ a_3a_4f \to a_1a_2m \ \ a_3a_4f \ \ a_ia_js \mid_{\neg no\_sex\_repr} \ \ (i \in \{1,2\}, j \in \{3,4\}, s \in \{m, f\})$
2. $a_1a_2f \to a_1a_2f \ \ a_1a_2f \mid_{\neg no\_asex\_repr}$

Note that each rule has its inhibitor *no_sex_repr* or *no_asex_repr*. The death rules are the following:

$$\begin{array}{ll} 3. & a_1a_2m \rightarrow \lambda\ |_{\neg no\_male\_death} \\ 4. & a_1a_2f \rightarrow \lambda\ |_{\neg no\_female\_death} \end{array}$$

We add also rules for females and males which simply survive, without reproducing or dying:

$$\begin{array}{ll} 5. & a_1a_2m \rightarrow a_1a_2m\ |_{\neg no\_male\_life} \\ 6. & a_1a_2f \rightarrow a_1a_2f\ |_{\neg no\_female\_life} \end{array}$$

Finally we use a rule for sending out individuals from the membrane in which the population evolved. This rule is *promoted* by the promoter *send_out*:

$$\begin{array}{ll} 7. & a_1a_2m \rightarrow (a_1a_2m,\ out)\ |_{send\_out} \\ 8. & a_1a_2f \rightarrow (a_1a_2f,\ out)\ |_{send\_out} \end{array}$$

In the following examples, we consider the membrane system defined above as the system under test. In each example we define a specific observer that is meant to test the evolution forcing certain situations. The observer controls the system by sending into it both the initial individuals and the promoters/inhibitors for constraining the population dynamics.

Example 1. *Let us consider a population in which the possible alleles for the single locus are* $\{0,1\}$ *and an initial population composed by four males of genotype* (00) *and four females of genotype* (01)*. Let us control the population dynamics by inhibiting both the sexual reproduction and the death of females. The observer we define, using the* **must** *version, is able to analyse the following property of the system: "After two time units, no female in the population can differ from the initial ones, and the number of such females is greater than or equal to the initial female number."*

Note that naturally the tests express quantitative *aspects, both on time and on numbers of individuals. Assume that membrane* 1*, i.e. the observer, initially contains the element a and that Inh is the set of all inhibitors, the rules in membrane* 1 *are the following:*

$$\begin{array}{ll} 1_0. & a \rightarrow 1\ ((00m)^4, (01f)^4, no\_sex\_repr, no\_female\_death,\ in_2) \\ 2_0. & 1 \rightarrow 2 \\ 3_0. & 2 \rightarrow 3\ (Inh \cup \{send\_out\},\ in_2) \\ 4_0. & 3 \rightarrow 4 \\ 5_0. & a_1a_2f \rightarrow fail \quad (a_1 \neq 0 \vee a_1 \neq 1) \\ 6_0. & 4 \rightarrow 5 \\ 7_0. & 5\ (01f)^4 \rightarrow (\omega,\ out)\ |_{\neg fail} \end{array}$$

Rule 1_0 *sends into the membrane under test the initial population (four males and four females) and the inhibitors for sexual reproduction and death of females. Rule* 2_0 *waits for a time unit, and, after that, Rule* 3_0 *sends, during the second step of the populations evolution, all the inhibitors together with the promoter send_out. Rule* 4_0 *waits for a time unit to allow the inner membrane to send out all the individuals. Afterwards, Rule* 5_0 *produces a* $fail$ *if a female different from the initial ones is present. At the same time Rule* 6_0 *increases the counter. Finally, Rule* 7_0 *sends out the* ω *symbol only if* $fail$ *is absent.*

Example 2. *Consider the initial population of Example 1, under the same conditions. Again using a* **must** *test, we consider the following property: "Given any* $k \in \mathbb{N}$*, we can check that it is not possible, in n time units, for all* $n \in [1,k]$*, to produce a female different from the initial ones." Note that this* *property is not a for-all statement: we count until the given k for checking it. This is of course weaker than checking the for-all statement.*

The rules in membrane 1 *are the following:*

$1_0.$	$a \rightarrow 1\ block\ ((00m)^4, (01f)^4, no\_sex\_repr, no\_female\_death,\ in_2)$
$2_0.$	$1 \rightarrow 2$
$3_0.$	$2 \rightarrow 3$
	$\dots$
$k_0.$	$k-1 \rightarrow k$
$(k+1)_0.$	$block \rightarrow block\ \|_{\neg k}$
$(k+2)_0.$	$block \rightarrow 1'\ (Inh \cup \{send\_out\},\ in_2)$
$1'_0.$	$1' \rightarrow 2'$
$2'_0.$	$a_1a_2f \rightarrow fail \quad (a_1 \neq 0 \vee a_1 \neq 1)$
$3'_0.$	$2' \rightarrow 3'$
$4'_0.$	$3' \rightarrow (\omega,\ out)\ \|_{\neg fail}$

Rules from 2_0 *to* k_0 *increase the counter until k. At each time unit either Rule* $(k+2)_0$ *can be executed, stopping the evolution of the population, or Rule* $(k+1)_0$ *can be fired, allowing the population to evolve for one more step. Rule* $(k+1)_0$ *is inhibited by k, thus when the counter reaches k the evolution must terminate. Rules from* $1'_0$ *to* $4'_0$ *produce a* ω *if and only if, in the final populations there are only females equal to the initial ones.*

Example 3. *Consider the initial population of Example 1. This time let us consider a* **may** *test expressing the following: "Without initial conditions it is possible to have recombination (offspring with different genotypes with respect to the initial population) after k steps."*

The rules are the following:

$1_0.$	$a \rightarrow 1\ block\ ((00m)^4, (01f)^4,\ in_2)$
$2_0.$	$1 \rightarrow 2$
$3_0.$	$2 \rightarrow 3$
	$\dots$
$k_0.$	$k-1 \rightarrow k$
$(k+1)_0.$	$k \rightarrow 1'\ (Inh \cup \{send\_out\},\ in_2)$
$1'_0.$	$1' \rightarrow 2'$
$2'_0.$	$a_1a_2f \rightarrow (\omega,\ out) \quad (a_1 \neq 0 \vee a_1 \neq 1)$
$2'_0.$	$a_1a_2m \rightarrow (\omega,\ out) \quad (a_1 \neq 0 \vee a_1 \neq 1)$

Example 4. *Consider again the initial population of Example 1. Let us define a* **may** *test expressing: "By allowing only the asexual reproduction for k steps, and then allowing only the sexual reproduction for the following k steps, it is possible to have recombination in the final population."*

For this example we need the concept of antidote. An antidote is a symbol able to remove the effect of an inhibitor, usually for an inhibitor x, the antidote is denoted by anti_x. The effect of an antidote is described by particular rules, the antidote rules, which have the form $anti\_x\ x \rightarrow \lambda$*. In this example we assume that, in the membrane under test, there are the antidote rules for the inhibitors no_sex_repr and* *no_sex_repr. The rules are the following:*

1_0.	$a \rightarrow 1\ block\ ((00m)^4, (01f)^4, no\_sex\_repr,\ in_2)$
2_0.	$1 \rightarrow 2$
	$\dots$
k_0.	$k-1 \rightarrow k\ (anti\_no\_sex\_repr, no\_asex\_repr,\ in_2)$
$(k+1)_0$.	$k \rightarrow k+1$
$(2k)_0$.	$2k-1 \rightarrow 1'\ (Inh \cup \{send\_out\},\ in_2)$
$1'_0$.	$1' \rightarrow 2'$
$2'_0$.	$a_1 a_2 f \rightarrow (\omega,\ out) \quad (a_1 \neq 0 \vee a_1 \neq 1)$
$2'_0$.	$a_1 a_2 m \rightarrow (\omega,\ out) \quad (a_1 \neq 0 \vee a_1 \neq 1)$

5 Conclusions

The testing machinery defined in [13] and the P Algebra proposed in [4] inspired us a suitable Process Algebra-based testing environment for P Systems. On the one hand, the new testing environment shares with the original one the concepts of observer, running test, successful and unsuccessful computation, testing preorders/equivalences, allowing us to define qualitative system properties. On the other hand, differently from the original one, it results to be suitable also to express *quantitative* aspects. Such a feature puts in evidence an expected high expressive power of the framework itself, which needs to be formally studied.

The natural continuation of this work is to find finite decidable characterizations of testing equivalence of finite state P Algebra terms in order to perform verification by comparing a system with its expected behaviour. Moreover, we plan to extend the testing environment also studying a suitable version of *fair* testing semantics for P Systems, as well as rephrasing the testing environment for Spatial P Systems [5], with the aim of expressing quantitative properties involving spatial information, being crucial in the biological (and not only) domain.

On the biological side, we intend to show as future work the potentials of the testing framework of having a practical impact, for instance on planning both in-silico and wet-lab experiments.

References

[1] J. Aguado, T. Balanescu, T. Cowling, M. Gheorghe, M. Holcombe, and F. Ipate. P Systems with Replicated Rewriting and Stream X-machines (Eilenberg Machines). *Fundam. Inf.*, 49:17–33, January 2002.

[2] J. Ahmad, G. Bernot, J. Comet, D. Lime, and O. Roux. Hybrid Modelling and Dynamical Analysis of Gene Regulatory Networks with Delays. *ComPlexUs*, 3(4):231–251, 2006.

[3] O. Andrei, G. Ciobanu, and D. Lucanu. A Rewriting Logic Framework for Operational Semantics of Membrane Systems. *Theor. Comput. Sci.*, 373(3):163–181, 2007.

[4] R. Barbuti, A. Maggiolo-Schettini, P. Milazzo, and D. Gruska. A Notion of Biological Diagnosability Inspired by the Notion of Opacity in Systems Security. *Fundamenta Informaticae*, 102(1):19–34, 2010.

[5] R. Barbuti, A. Maggiolo-Schettini, P. Milazzo, G. Pardini, and L. Tesei. Spatial P Systems. *Natural Computing*, 2010. Received: 26 October 2009 Accepted: 24 February 2010 Published online: 24 March 2010.

[6] R. Barbuti, A. Maggiolo-Schettini, P. Milazzo, and S. Tini. A P Systems Flat Form Preserving Step-by-step Behaviour. *Fundamenta Informaticae*, 87(1):1–34, 2008.

[7] R. Barbuti, A. Maggiolo-Schettini, P. Milazzo, and S. Tini. Compositional Semantics and Behavioral Equivalences for P Systems. *Theoretical Computer Science*, 395(1):77–100, 2008.

[8] L. Bianco and V. Manca. Encoding-Decoding Transitional Systems for Classes of P systems. In *Proc. of WMC 2005, 6th International Workshop on Membrane Computing*, number 3850 in LNCS, pages 134–143. Springer-Verlag, 2006.

[9] M. Boreale and R. Pugliese. Basic Observables for Processes. *Information and Computation*, 149(1):77–98, 1999.

[10] P. Bottoni, C. Martín-Vide, G. Păun, and G. Rozenberg. Membrane Systems with Promoters/Inhibitors. *Acta Informatica*, 38(10):695–720, 2002.

[11] G. Ciobanu, M. J. Pérez-Jiménez, and G. Paun, editors. *Applications of Membrane Computing*. Natural Computing Series. Springer, 2006.

[12] Z. Dang, C. Li, O. H. Ibarra, and G. Xie. On the Decidability of Model-checking for P Systems. *J. Autom. Lang. Comb.*, 11:279–298, January 2006.

[13] R. De Nicola and M. Hennessy. Testing Equivalences for Processes. *Theoretical Computer Science*, 34(1-2):83–133, 1984.

[14] M. Gheorghe and F. Ipate. On Testing P Systems. In D. Corne, P. Frisco, G. Paun, G. Rozenberg, and A. Salomaa, editors, *Membrane Computing*, volume 5391 of *Lecture Notes in Computer Science*, pages 204–216. 2009.

[15] M. Gheorghe and F. Ipate. Testing Based on P Systems - An Overview. In *Proceedings of the 11th International Conference on Membrane Computing*, CMC'10, pages 3–6, Berlin, Heidelberg, 2010. Springer-Verlag.

[16] F. Ipate and M. Gheorghe. Testing Non-deterministic Stream X-machine Models and P systems. *Electron. Notes Theor. Comput. Sci.*, 227:113–126, January 2009.

[17] F. Ipate, M. Gheorghe, and R. Lefticaru. Test Generation From P systems Using Model Checking. *Journal of Logic and Algebraic Programming*, 79(6):350–362, 2010. Membrane computing and programming.

[18] F. Ipate, R. Lefticaru, and C. Tudose. Formal Verification of P Systems using SPIN. *International Journal of Foundations of Computer Science*, 22(1):133–142, 2011.

[19] V. Natarajan and R. Cleaveland. Divergence and Fair Testing. In *Proceedings of the 22nd International Colloquium on Automata, Languages and Programming*, ICALP '95, pages 648–659, 1995.

[20] G. Păun. *Membrane Computing. An Introduction*. Springer-Verlag, 2002.

[21] G. D. Plotkin. A Structural Approach to Operational Semantics. *J. Log. Algebr. Program.*, 60-61:17–139, 2004.

[22] G. Păun and G. Rozenberg. A Guide to Membrane Computing. *Theoretical Computer Science*, 287(1):73–100, 2002.

[23] A. Rensink and W. Vogler. Fair testing.

[24] I. Schön, K. Martens, and P. van Dijk. *Lost Sex: The Evolutionary Biology of Parthenogenesis*. Springer Verlag, 2009.

A Spatial Calculus of Wrapped Compartments*

Livio Bioglio[1], Cristina Calcagno[1,2], Mario Coppo[1], Ferruccio Damiani[1],
Eva Sciacca[1], Salvatore Spinella[1], Angelo Troina[1]
[1]Dipartimento di Informatica, Università di Torino
[2]Dipartimento di Biologia Vegetale, Università di Torino

The Calculus of Wrapped Compartments (CWC) is a recently proposed modelling language for the representation and simulation of biological systems behaviour. Although CWC has no explicit structure modelling a spatial geometry, its compartment labelling feature can be exploited to model various examples of spatial interactions in a natural way. However, specifying large networks of compartments may require a long modelling phase. In this work we present a surface language for CWC that provides basic constructs for modelling spatial interactions. These constructs can be compiled away to obtain a standard CWC model, thus exploiting the existing CWC simulation tool. A case study concerning the modelling of Arbuscular Mychorrizal fungi growth is discussed.

1 Introduction

Several complex biological phenomena include aspects in which space plays an essential role, key examples are the growth of tissues and organisms, embryogenesis and morphogenesis processes or cell proliferation. This has encouraged, in recent years, the development of formal models for the description of biological systems in which spatial properties can be taken into account [9, 4, 19], as required by the emerging field of *spatial systems biology* [26] which aims at integrating the analysis of biological systems with spatial properties

The Calculus of Wrapped Compartments (CWC) [12, 11, 10] is a calculus for the description of biochemical systems which is based on the notion of a compartment which represents, in some sense, the abstraction of a region with specific properties (characterized by a *label*, a *wrap* and a *content*). Biochemical transformations are described via a set of stochastic reduction rules which characterize the behaviour of the represented system.

In a recent work [7] we have have shown how CWC can be used to model spatial properties of biological systems. The idea is to exploit the notion of compartment to represent spatial regions (with a fixed, two-dimensional topology) in which the labels plays a key role in defining the spatial properties. In this framework, the movement and growth of system elements are described, via specific rules (involving adjacent compartments) and the functionalities of biological components are affected by the spatial constraints given by the sector in which they interact with other elements. CWC allows to model several spatial interactions in a very natural way. However, when the complexity of simulation scenarios increases, the specification of large networks of compartments each one having its own peculiar behaviour and initial state may require a long and error prone modelling phase.

In this paper we introduce a surface language for CWC that defines a framework in which the notion of space is included as an essential component of the system. The space is structured as a square grid, whose dimension must be declared as part of the system specification. The surface language provides

*This research is funded by the BioBITs Project (*Converging Technologies* 2007, area: Biotechnology-ICT), Regione Piemonte.

Proceedings of 5th Workshop on Membrane Computing
and Biologically Inspired Process Calculi (MeCBIC 2011)
Pages 25–39, 2011.

© L. Bioglio, C. Calcagno, M. Coppo, F. Damiani,
E. Sciacca, S. Spinella and A. Troina
All the rights to the paper remain with the authors.

basic constructs for modelling spatial interactions on the grid. These constructs can be compiled away to obtain a standard CWC model, thus exploiting the existing CWC simulation tool.

A similar approach can be found in [19] where the topological structure of the components is expressed via explicit links which require ad-hoc rules to represent movements of biological entities and a logic-oriented language to flexibly specify complex simulation scenarios is provided. In order to deal with larger biological systems, we are planning to extend the CWC to the spatial domain incrementally. At this early stage we neglected to consider problems related to the increase of the spatial rules with the increasing dimension of the grid[1]. A partial solution to this problem is the use of appropriate data structures to represent entities scattered on a grid. A further step in this direction should be that of allowing the definition of different topological representations for spatial distributions of the biological entities, like in [7]. This requires, obviously, that also the surface language be enriched with primitives suitable to express different spatial topology and related concepts (like the notion of proximity of locations and that of movement in space). The right spatial topology could also help to minimize the number of spatial rules needed for modeling phenomena. A more ambitious goal will be that of providing a basis for computational geometry to our simulator, in order to identify spatially significant events for the simulation. This will requires however a much bigger implementation effort.

Organisation of the Paper Section 2 recalls the CWC framework. Section 3 presents the surface language needed to describe spatial terms and rules. Section 4 presents a case study concerning some spatial aspects in the modelling of Arbuscular Mychorrizal fungi. Section 5 concludes the paper by briefly discussing related work and possible directions for further work. The Appendix presents the software module implementing the surface language.

2 The Calculus of Wrapped Compartments

The Calculus of Wrapped Compartments (CWC) (see [12, 10, 11]) is based on a nested structure of ambients delimited by membranes with specific proprieties. Biological entities like cells, bacteria and their interactions can be easily described in CWC.

2.1 Term Syntax

Let $\mathscr{A}$ be a set of *atomic elements* (*atoms* for short), ranged over by a, b, ..., and $\mathscr{L}$ a set of *compartment types* represented as *labels* ranged over by $\ell, \ell', \ell_1, \ldots$ A *term* of CWC is a multiset $\bar{t}$ of *simple terms* where a simple term is either an atom a or a compartment $(\overline{a} \rfloor \overline{t'})^{\ell}$ consisting of a *wrap* (represented by the multiset of atoms $\overline{a}$), a *content* (represented by the term $\overline{t'}$) and a *type* (represented by the label ℓ).

As usual, the notation $n * t$ denotes n occurrences of the simple term t. We denote an empty term with $\bullet$. An example of CWC term is $2{*}a\ b\ (c\ d \rfloor e\ f)^{\ell}$ representing a multiset (multisets are denoted by listing the elements separated by a space) consisting of two occurrences of a, one occurrence of b (e.g. three molecules) and an ℓ-type compartment $(c\ d \rfloor e\ f)^{\ell}$ which, in turn, consists of a wrap (a membrane) with two atoms c and d (e.g. two proteins) on its surface, and containing the atoms e (e.g. a molecule) and f (e.g. a DNA strand). See Figure 1 for some other examples with a simple graphical representation.

[1]note that in a 2D model the space-related rules grow according to the square of the grid dimension

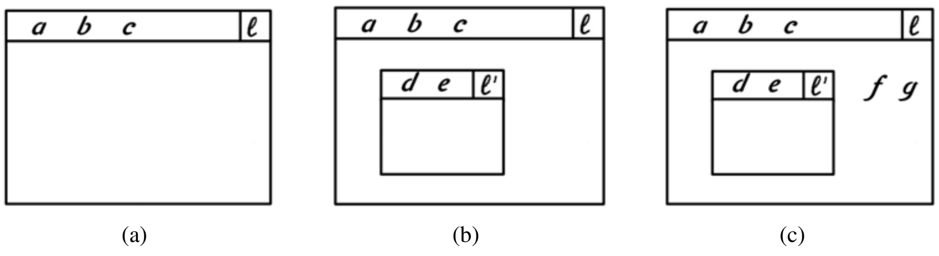


Figure 1: **(a)** represents $(a\ b\ c \rfloor \bullet)^{\ell}$; **(b)** represents $(a\ b\ c \rfloor (d\ e \rfloor \bullet)^{\ell'})^{\ell}$; **(c)** represents $(a\ b\ c \rfloor (d\ e \rfloor \bullet)^{\ell'}\ f\ g)^{\ell}$

2.2 Rewriting Rules

System transformations are defined by rewriting rules, defined by resorting to CWC terms that may contain variables. We call *pattern* the l.h.s. component $\overline{p}$ of a rewrite rule and *open term* the r.h.s. component $\overline{o}$ of a rewrite rule, defined as multiset of *simple patterns* p and *simple open terms* o given by the following syntax:

$$\begin{array}{rcl|l}
p & ::= & a & (\overline{a}\ x \rfloor \overline{p}\ X)^{\ell} \\
o & ::= & a & (\overline{q} \rfloor \overline{o})^{\ell} \quad \Big| \quad X \\
q & ::= & a & x
\end{array}$$

where $\overline{a}$ is a multiset of atoms, $\overline{p}$ is a pattern (a, possibly empty, multiset of simple patterns), x is a *wrap variable* (can be instantiated by a multiset of atoms), X is a *content variable* (can be instantiated by a CWC term), $\overline{q}$ is a multiset of atoms and wrap variables and $\overline{o}$ is an open term (a, possibly empty, multiset of simple open terms). Patterns are intended to match, via substitution of variables with ground terms (containing no variables), with compartments occurring as subterms of the term representing the whole system. Note that we force *exactly* one variable to occur in each compartment content and wrap of our patterns and simple patterns. This prevents ambiguities in the instantiations needed to match a given compartment.[2]

A *rewrite rule* is a triple $(\ell, \overline{p}, \overline{o})$, denoted by $\ell : \overline{p} \longmapsto \overline{o}$, where $\overline{p}$ and $\overline{o}$ are such that the variables occurring in $\overline{o}$ are a subset of the variables occurring in $\overline{p}$. The application of a rule $\ell : \overline{p} \longmapsto \overline{o}$ to a term $\overline{t}$ is performed in the following way: 1) Find in $\overline{t}$ (if it exists) a compartment of type ℓ with content u and a substitution σ of variables by ground terms such that $u = \sigma(\overline{p}\ X)$[3] and 2) Replace in $\overline{t}$ the subterm u with $\sigma(\overline{o}\ X)$. We write $\overline{t} \longmapsto \overline{t'}$ if $\overline{t'}$ is obtained by applying a rewrite rule to $\overline{t}$. The rewrite rule $\ell : \overline{p} \longmapsto \overline{o}$ can be applied to any compartment of type ℓ with $\overline{p}$ in its content (that will be rewritten with $\overline{o}$).

For instance, the rewrite rule $\ell : a\ b \longmapsto c$ means that in all compartments of type ℓ an occurrence of $a\ b$ can be replaced by c

While the rule does not change the label ℓ of the compartment where the rule is applied, it may change all the labels of the compartments occurring in its content. For instance, the rewrite rule $\ell : (a\ x \rfloor X)^{\ell_1} \longmapsto (a\ x \rfloor X)^{\ell_2}$ means that, if contained in a compartment of type ℓ, all compartments of type ℓ_1 and containing an a in their wrap can change their type to ℓ_2.

For uniformity reasons we assume that the whole system is always represented by a term consisting of a single compartment with distinguished label $\top$ and empty wrap, i.e., any system is represented by a

[2] The linearity condition, in biological terms, corresponds to excluding that a transformation can depend on the presence of two (or more) identical (and generic) components in different compartments (see also [20]).

[3]The implicit (distinguished) variable X matches with all the remaining part of the compartment content.

term of the shape $(\bullet \rfloor \bar{t})^\top$, which will be also written as $\bar{t}$, for simplicity.

2.3 Stochastic Simulation

A stochastic simulation model for biological systems can be defined along the lines of the one presented by Gillespie in [14], which is, *de facto*, the standard way to model quantitative aspects of biological systems. The basic idea of Gillespie's algorithm is that a rate function is associated with each considered chemical reaction which is used as the parameter of an exponential distribution modelling the probability that he reaction takes place. In the standard approach this reaction rate is obtained by multiplying the kinetic constant of the reaction by the number of possible combinations of reactants that may occur in the region in which the reaction takes place, thus modelling the law of mass action. For rules defining spatial movement the kinetic constant can be interpreted as the speed of the movement. In [12], the reaction rate is defined in a more general way by associating to each reduction rule a function which can also define rates based on different principles as, for instance, the Michaelis-Menten nonlinear kinetics.

For simplicity, in this paper, we will follow the standard approach in defining reaction rates. Each reduction rule is then enriched by the kinetic constant k of the reaction that it represents (notation $\ell : \overline{p} \overset{k}{\longmapsto} \overline{o}$). For instance in evaluating the application rate of the stochastic rewrite rule $R = \ell : a\ b \overset{k}{\longmapsto} c$ (written in the simplified form) to the term $\bar{t} = a\ a\ b\ b$ in a compartment of type ℓ we must consider the number of the possible combinations of reactants of the form $a\ b$ in $\bar{t}$. Since each occurrence of a can react with each occurrence of b, this number is 4. So the application rate of R is $k \cdot 4$.

2.4 The CWC simulator

The CWC simulator [1] is a tool under development at the Computer Science Department of the Turin University, based on Gillespie's direct method algorithm [14]. It treats CWC models with different rating semantics (law of mass action, Michaelis-Menten kinetics, Hill equation) and it can run independent stochastic simulations over CWC models, featuring deep parallel optimizations for multi-core platforms on the top of FastFlow [2]. It also performs online analysis by a modular statistical framework.

3 A Surface Language

In this section we embed CWC into a surface language able to express, in a synthetic form, both spatial (in a two-dimensional grid) and biochemical CWC transformations. The semantics of a surface language model is defined by translation into a standard CWC model.

We distinguish between two kind of compartments:

1. *Standard* compartments (corresponding to the usual CWC compartments), used to represent entities (like bacteria or cells) that can move through space.

2. *Spatial* compartments, used to represent portions of space. Each spatial compartment defines a location in a two dimensional grid through a special atom, called *coordinate*, that occurs on its wrap. A coordinate is denoted by `row.column`, where `row` and `column` are intergers. Spatial compartments have distinguished labels, called *spatial labels*, that can be used to provide a specific characterisation of a portion of space.

For simplicity we assume that the wraps of each spatial compartment contains only the coordinate. Therefore, spatial compartment differentiations can be expressed only in terms of labels.[4]

For example, the spatial compartment $(\mathtt{1.2} \rfloor 2 * b)^{soil}$ represents the cell of the grid located in the first row and the second column, and has type *soil*, the spatial compartment $(\mathtt{2.3} \rfloor 3 * b\ c)^{water}$ represents a *water*-type spatial compartment in position `2.3`. In our grid we assume that molecules can float only through neighbor cells: all the rules of interaction between spatial compartments must obviously contain the indexes of their location. For example, the rule $\top : (\mathtt{1.2}\ x \rfloor a\ X)^{water}(\mathtt{2.2}\ y \rfloor Y)^{soil} \stackrel{k}{\longmapsto} (\mathtt{1.2}\ x \rfloor X)^{water}(\mathtt{2.2}\ y \rfloor a\ Y)^{soil}$ moves the molecule a from the *water* compartment in position `1.2` to the *soil* compartment in position `2.2` with a rate k representing in this case, the speed of the movement of a in downwards direction from a cell of *water*-type to a cell of *soil*-type.

Let `R` and `C` denote the dimensions of our $\mathtt{R} \times \mathtt{C}$ grid defined by `R` rows and `C` columns. To increase the expressivity of the language we define a few structures to denote portions (i.e. sets of cells) of the grid. With Θ we denote a set of coordinates of the grid and we use the notion $\mathtt{r.c} \in \Theta$ when the coordinate `r.c` is contained in the set Θ. We define rectangles by `rect[r.c,r'.c']` where `r.c,r'.c'` represent the edges of the rectangle. We project rows and columns of our grid with the constructions $\mathtt{row}[i]$ and $\mathtt{col}[j]$ respectively.

Example 3.1 *The set* $\Theta = \{\mathtt{6.6}\} \cup \mathtt{rect}[\mathtt{1.1},\mathtt{3.2}] \cup \mathtt{col}[5]$ *represents the set of coordinates*

$$\Theta = \{\mathtt{6.6}\} \cup \{\mathtt{1.1},\mathtt{2.1},\mathtt{3.1},\mathtt{1.2},\mathtt{2.2},\mathtt{3.2}\} \cup \{\mathtt{i.5} \mid \forall i \in [1,\mathtt{R}]\}.$$

Note that $\mathtt{row}[i]$ is just a shorthand for `rect[i.1,i.C]`. Similarly for columns.
We use `[*]` as shorthand to indicate the whole grid (i.e. `rect[1.1,R.C]`).

We also define four *direction* operators, `N, W, S, E` that applied to a range of cells shift them, respectively, up, left, down and right. For instance `E(1.1) = 1.2`. In the intuitive way, we also define the four diagonal movements (namely, `NW`, `SW`, `NE`, `SE`). With Δ we denote a set of directions and we use the special symbol $\diamond$ to denote the set containing all eight possible directions.

We convene that when a coordinate, for effect of a shit, goes out of the range of the grid the corresponding point is eliminated from the set.

3.1 Surface Terms

We define the initial state of the system under analysis as a set of compartments modelling the two-dimensional grid containing the biological entities of interest.

Let Θ denote a set of coordinates and ℓ_s a spatial label. We use the notation:

$$\Theta, \ell_s \boxplus \bar{t}$$

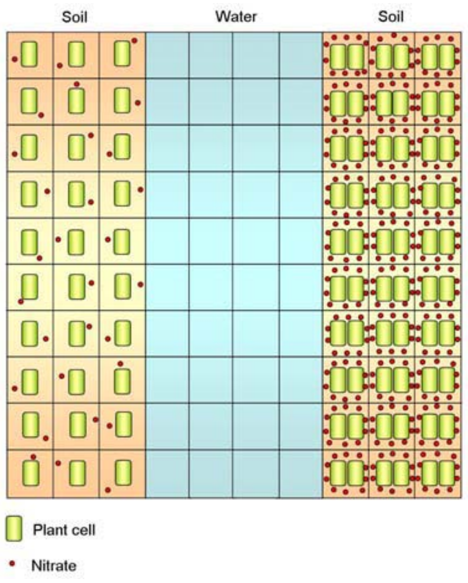
to define a set of cells of the grid. Namely $\Theta, \ell_s \boxplus \bar{t}$ denotes the top level CWC term:

$$(\bullet \rfloor (\mathtt{r_1.c_1} \rfloor \bar{t})^{\ell_s} \ \dots \ (\mathtt{r_n.c_n} \rfloor \bar{t})^{\ell_s})^{\top}$$

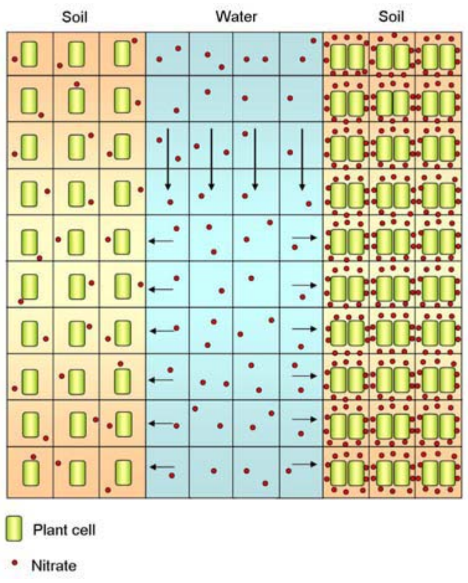
where $\mathtt{r_i.c_j}$ range over all elements of Θ.

A spatial CWC term is thus defined by the set of grid cells covering the entire grid.

[4]Allowing the wrap of spatial compartments to contain other atoms, thus providing an additional mean to express spatial compartment differentiations, should not pose particular technical problems (extend the rules of the surface language to deal with a general wrap content also for spatial compartments should be straightforward).



(a) Initial state



(b) Spatial events

Figure 2: Graphical representations of the grid described in the Example 3.2.

Example 3.2 *The CWC term obtained by the three grid cell constructions:*

$\texttt{rect}[1.1,10.3], soil \boxplus\ nitr\ (receptors \rfloor cytoplasm\ nucleus)^{PlantCell}$
$\texttt{rect}[1.4,10.7], water \boxplus\ \bullet$
$\texttt{rect}[1.8,10.10], soil \boxplus\ 10 * nitr\ 2 * (receptors \rfloor cytoplasm\ nucleus)^{PlantCell}$
builds a 10×10 *grid composed by two portions of soil (the right-most one reacher of nitrates and plant cells) divided by a river of water (see Figure 2(a)).*

3.2 Surface Rewrite Rules

We consider rules for modelling three kind of events.
Non-Spatial Events: are described by standard CWC rules, i.e. by rules of the shape:

$$\ell : \overline{p} \stackrel{k}{\longmapsto} \overline{o}$$

Non-spatial rules can be applied to any compartment of type ℓ occurring in any portion of the grid and do not depend on a particular location.

Example 3.3 *A plant cell might perform its activity in any location of the grid. The following rules, describing some usual activities within a cell, might happen in any spatial compartment containing the plant cells under considerations:*
$PlantCell : nucleus \stackrel{k_1}{\longmapsto} nucleus\ mRNA$
$PlantCell : mRNA\ cytoplasm \stackrel{k_2}{\longmapsto} mRNA\ cytoplasm\ protein.$

Spatial Events: are described by rules that can be applied to specific spatial compartments. These rules allow to change the spatial label of the considered compartment. Spatial events are described by rules of the following shape:

$$\Theta \triangleright \ell_s : \overline{p} \xmapsto{k} \ell'_s : \overline{o}$$

Spatial rules can be applied only within the spatial compartments with coordinates contained in the set Θ and with the spatial label ℓ_s. The application of the rule may also change the label of the spatial compartments ℓ_s to ℓ'_s. This rule is translated into the CWC set of rules:
$\top : (\mathtt{r_i.c_i}\ x \rfloor \overline{p}\ X)^{\ell_s} \xmapsto{k} (\mathtt{r_i.c_i}\ x \rfloor \overline{o}\ X)^{\ell'_s} \quad \forall \mathtt{r_i.c_i} \in \Theta$.

Note that spatial rules are analogous to non spatial ones. The only difference is the explicit indication of the set Θ which allows to write a single rule instead of a set of rule (one for each element of Θ).

Example 3.4 *If we suppose that the river of water in the middle of the grid defined in Example 3.2 has a downward streaming, we might consider the initial part of the river (framed by the first row* `rect[1.4,1.7]`*) to be a source of nitrates (as they are coming from a region which is not modelled in the actual considered grid). The spatial rule:*
$\mathtt{rect[1.4,1.7]} \triangleright water : \bullet \xmapsto{k_3} water : nitr$
models the arrival of nitrates at the first modeled portion of the river (in this case it does not change the label of the spatial compartment involved by the rule).

Spatial Movement Events: are described by rules considering the content of two adjacent spatial compartments and are described by rules of the following shape:

$$\Theta \triangleleft \Delta \triangleright \ell_{s_1}, \ell_{s_2} : \overline{p_1}, \overline{p_2} \xmapsto{k} \ell'_{s_1}, \ell'_{s_2} : \overline{o_1}, \overline{o_2}$$

This rule changes the content of two adjacent (according to the possible directions contained in Δ) spatial compartments and thus allows to define the movement of objects. The pattern matching is performed by checking the content of a spatial compartment of type ℓ_{s_1} located in a portion of the grid defined by Θ and the content of the adjacent spatial compartment of type ℓ_{s_2}. Such a rule could also change the labels of the spatial compartments. This rule is translated into the CWC set of rules:

$$\top : (\mathtt{r_i.c_i}\ x \rfloor \overline{p_1}\ X)^{\ell_{s_1}} (dir(\mathtt{r_i.c_i})\ y \rfloor \overline{p_2}\ Y)^{\ell_{s_2}} \xmapsto{k} (\mathtt{r_i.c_i}\ x \rfloor \overline{o_1}\ X)^{\ell'_{s_1}} (dir(\mathtt{r_i.c_i})\ y \rfloor \overline{o_2}\ Y)^{\ell'_{s_2}}$$

for all $\mathtt{r_i.c_i} \in \Theta$ and for all $dir \in \Delta$.

Example 3.5 *We assume that the flux of the river moves the nitrates in the water according to a downward direction in our grid and with a constant speed in any portion of the river with the following rule:*
$\mathtt{rect[1.4,9.7]} \triangleleft \{\mathtt{S}\} \triangleright water, water : nitr, \bullet \xmapsto{k_4} water, water : \bullet, nitr$
when nitrates reach the down-most row in our grid they just disappear (non moving event):

$$\mathtt{rect[10.4,10.7]} \triangleright water : nitr \xmapsto{k_4} water : \bullet.$$

Moreover, nitrates streaming in the river may be absorbed by the soil on the riverside with the rule:
$\mathtt{rect[1.4,10.7]} \triangleleft \{\mathtt{W,E}\} \triangleright water, soil : nitr, \bullet \xmapsto{k_5} water, soil : \bullet, nitr.$
A graphical representation of these events is shown in Figure 2(b). Other rules can be defined to move the nitrates within the soil etc.

4 Case Study: A Growth Model for AM Fungi

In this section we illustrate a case study concerning the modelling of Arbuscular Mychorrizal fungi growth.

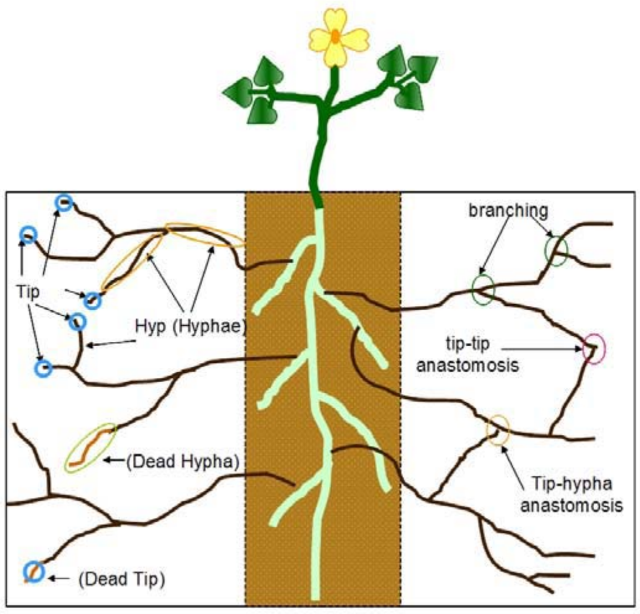


Figure 3: Extraradical mycelia of an arbuscular mycorrhizal fungus.

4.1 Biological Model

The arbuscular mycorrhizal (AM) symbiosis is an example of association with high compatibility formed between fungi belonging to the Glomeromycota phylum and the roots of most land plants[15]. AM fungi are obligate symbionts, in the absence of a host plant, spores of AM fungi germinate and produce a limited amount of mycelium. The recognition between the two symbionts is driven by the perception of diffusible signals and once reached the root surface the AM fungus enters in the root, overcomes the epidermal layer and it grows inter-and intracellularly all along the root in order to spread fungal structures. Once inside the inner layers of the cortical cells the differentiation of specialized, highly branched intracellular hyphae called arbuscules occur. Arbuscules are considered the major site for nutrients exchange between the two organisms. The fungus supply the host with essential nutrients such as phosphate, nitrate and other minerals from the soil. In return, AM fungi receive carbohydrates derived from photosynthesis in the host.

Simultaneously to intraradical colonization, the fungus develops an extensive network of hyphae which explores and exploits soil microhabitats for nutrient acquisition. AM fungi have different hyphal growth patterns, anastomosis and branching frequencies which result in the occupation of different niche in the soil and probably reflect a functional diversity [18] (see Figure 3). The mycelial network that develops outside the roots is considered as the most functionally diverse component of this symbiosis. Extraradical mycelia (ERM) not only provide extensive pathways for nutrient fluxes through the soil, but also have strong influences upon biogeochemical cycling and agro-ecosystem functioning [22]. The mechanisms by which fungal networks extend and function remain poorly characterized. The functioning of ERM presumably relies on the existence of a complex regulation of fungal gene expression with regard to nutrient sensing and acquisition. The fungal life cycle is then completed by the formation, from the external mycelium, of a new generation of spores able to survive under unfavourable conditions.

Investigations on carbon (C) metabolism in AM fungi have proved useful to offer some explanation for their obligate biotrophism. As mentioned above, an AM fungus relies almost entirely on the host plant for its carbon supply. Intraradical fungal structures (presumably the arbuscules) are known to take up photosynthetically fixed plant C as hexoses. Unfortunately, no fungal hexose transporter-coding gene has been characterized yet in AM fungi.

In order to quantify the contribution of arbuscular mycorrhizal (AM) fungi to plant nutrition, the development and extent of the external fungal mycelium and its nutrient uptake capacity are of particular importance. Shnepf and collegues [25] developed and analysed a model of the extraradical growth of AM fungi associated with plant roots considering the growth of fungal hyphae from a cylindrical root in radial polar coordinates.

Measurements of fungal growth can only be made in the presence of plant. Due to this practical difficulty experimental data for calibrating the spatial and temporal explicit models are scarce. Jakobsen and collegues [16] presented measurements of hyphal length densities of three AM fungi: *Scutellospora calospora* (Nicol.& Gerd.) Walker & Sanders; *Glomus* sp. associated with clover (*Trifolium subterraneum* L.);these data appeared suitable for comparison with modelled hyphal length densities.

The model in [25] describes, by means of a system of Partial Differential Equations (PDE), the development and distribution of the fungal mycelium in soil in terms of the creation and death of hyphae, tip-tip and tip-hypha anastomosis, and the nature of the root-fungus interface. It is calibrated and corroborated using published experimental data for hyphal length densities at different distances away from root surfaces. A good agreement between measured and simulated values was found for the three fungal species with different morphologies associated with *Trifolium subterraneum* L. The model and findings are expected to contribute to the quantification of the role of AM fungi in plant mineral nutrition and the interpretation of different foraging strategies among fungal species.

4.2 Surface CWC Model

In this Section we describe how to model the growth of arbuscular mycorrhyzal fungi using the surface spatial CWC. We model the growth of AM fungal hyphae in a soil environment partitioned into 13 different layers (spatial compartments with label *soil*) to account for the distance in centimetres between the plant root and the fungal hyphae where the soil layer at the interface with the plant root is at position 1.1. We describe the mycelium by two atoms: the hyphae (atom *Hyp*) related to the length densities (number of hyphae in a given compartment) and the hyphal tips (atom Tip). The plant root (atom *Root*) is contained in the *soil* compartment at position 1.1.

The tips and hyphae at the root-fungus interface proliferate according to the following spatial events:

$$\{\mathtt{1.1}\} \triangleright soil : Root \overset{\tilde{a}}{\longmapsto} soil : Root\ Hyp$$

$$\{\mathtt{1.1}\} \triangleright soil : Root \overset{a}{\longmapsto} soil : Root\ Tip$$

where $\tilde{a}$ and a is the root proliferation factor for the hyphae and tips respectively.

Hyphal tips are important, because growth occurs due to the elongation of the region just behind the tips. Therefore, the spatial movement event describing the hyphal segment created during a tip shift to a nearby compartment is:

$$[*] \triangleleft \{\mathtt{E},\mathtt{W}\} \triangleright soil, soil : Tip, \bullet \xmapsto{v} soil, soil : Hyp, Tip$$

where v is the rate of tip movement. The hyphal length is related to tips movement, i.e. an hyphal trail is left behind as tips move through the compartments. We consider hyphal death to be linearly proportional to the hyphal density, so that the rule describing this spatial event is:

$$[*] \triangleright soil : Hyp \xmapsto{d_H} soil : \bullet$$

where d_H is the rate of hyphal death.

Mycorrhizal fungi are known to branch mainly apically where one tip splits into two. In the simplest case, branching and tip death are linearly proportional to the existing tips in that location modelled with the following spatial events:

$$[*] \triangleright soil : Tip \xmapsto{b_T} soil : 2 * Tip$$

$$[*] \triangleright soil : Tip \xmapsto{d_T} soil : \bullet$$

where b_T is the tip branching rate and d_T is the tip death rate.

Alternatively, if we assume that branching decreases with increasing tip density and ceases at a given maximal tip density, we employ the spatial event:

$$[*] \triangleright soil : 2 * Tip \xmapsto{c_T} soil : \bullet$$

where $c_T = \frac{b_T}{T_{max}}$. From a biological point of view, this behaviour take into account the volume saturation when the tip density achieves the maximal number of tips T_{max}.

The fusion of two hyphal tips or a tip with a hypha can create interconnected networks by means of anastomosis:

$$[*] \triangleright soil : 2 * Tip \xmapsto{a_1} soil : Tip$$

$$[*] \triangleright soil : Tip\ Hyp \xmapsto{a_2} soil : Tip$$

where a_1 and a_2 are the tip-tip and tip-hypha anastomosis rate constants, respectively.

The initial state of the system is given by the following grid cell definition:

$$\{\mathtt{1.1}\},\ soil \boxplus Root\ T_0 * Tip\ H_0 * Hyp$$

$$\mathtt{rect}[\mathtt{1.2}, \mathtt{1.13}],\ soil \boxplus \bullet$$

where T_0 and H_0 are the initial number of tips and hyphae respectively at the interface with the plant root.

4.3 Results

We run 60 simulations on the model for the fungal species *Scutellospora calospora* and *Glomus* sp. Figure 4 show the mean values of hyphae (atoms *Hyp*) of the resulting stochastic simulations in function of the elapsed time in days and of the distance from the root surface. The rate parameters of the model are taken from [25].

The results for *S. calospora* are in accordance with the linear PDE model of [25] which is characterized by linear branching with a relatively small net branching rate and both kinds of anastomosis are

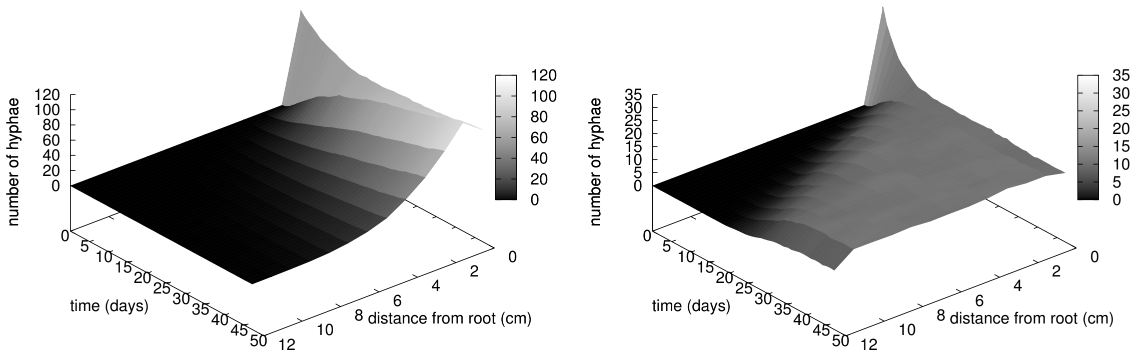


Figure 4: Mean values of 60 stochastic simulations of hyphal growth (atoms *Hyp*) results for S. calospora and Glomus sp. fungi.

negligible when compared with the other species. This model imply that the fungus is mainly growing and allocating resources for getting a wider catchment area rather than local expoloitation of mineral resources via hyphal branching.

The model for *Glomus* sp. considers the effect of nonlinear branching due to the competition between tips for space. The results obtained for *Glomus* sp. are in accordance with the non–linear PDE model of [25] which imply that local exploitation for resources via hyphal branching is important for this fungus as long as the hyphal tip density is small. Reaching near the maximum tip density, branching decreases. Symbioses between a given host plant and different AM fungi have been shown to differ functionally [23].

5 Conclusions and Related Works

For the well-mixed chemical systems (even divided into nested compartments) often found in cellular biology, interaction and distribution analysis are sufficient to study the system's behaviour. However, there are many other situations, like in cell growth and developmental biology where dynamic spatial arrangements of cells determines fundamental functionalities, where a spatial analysis becomes essential. Thus, a realistic modelling of biological processes requires space to be taken into account [17].

This has brought to the extension of many formalisms developed for the analysis of biological systems with (even continuous) spatial features.

In [9], Cardelli and Gardner develop a calculus of processes located in a three-dimensional geometric space. The calculus, introduces a single new geometric construct, called *frame shift*, which applies a three-dimensional space transformation to an evolving process. In such a work, standard notions of process equivalence give rise to geometric invariants.

In [5], a variant of P-systems embodying concepts of space and position inside a membrane is presented. The objects inside a membrane are associated with a specific position. Rules can alter the position of the objects. The authors also define *exclusive* objects (only one exclusive object can be contained inside a membrane). In [3], an spatial extension of CLS is given in a 2D/3D space. The spatial terms of the calculus may move autonomously during the passage of time, and may interact when the constraints on their positions are satisfied. The authors consider a *hard-sphere* based notion of space: two objects, represented as spheres, cannot occupy the same space, thus conflicts may arise by moving objects. Such conflicts are resolved by specific algorithms considering the forces involved and appropriate pushing among the objects.

BioShape [6] is a spatial, particle-based and multi-scale 3D simulator. It treats biological entities of different size as geometric 3D *shapes*. A shape is either basic (polyhedron, sphere, cone or cylinder) or composed (aggregation of shapes glued on common surfaces of contact). Every element involved in the simulation is a 3D process and has associated its physical motion law.

Adding too many features to the model (e.g., coordinates, position, extension, motion direction and speed, rotation, collision and overlap detection, communication range, etc.) could heavily rise the complexity of the analysis. To overcome this risk, a detailed study of the possible subsets of these features, chosen to meet the requirements of particular classes of biological phenomena, might be considered.

In this paper we pursued this direction by extending CWC with a surface language providing a framework for incorporating basic spatial features (namely, coordinates, position and movement). In future work we plan to extend the surface language to deal with three dimensional spaces and to investigate the possibility to incorporate other spatial features to the CWC simulation framework.

Notably, the framework presented in this paper could also be applied to other calculi which are able to express compartmentalisation (see, e.g., BioAmbients [24], Brane Calculi [8], Beta-Binders [13], etc.).

References

[1] M. Aldinucci, M. Coppo, F. Damiani, M. Drocco, E. Giovannetti, E. Grassi, E. Sciacca, S. Spinella & A. Troina (2010): *CWC Simulator*. Dipartimento di Informatica, Università di Torino. `http://cwcsimulator.sourceforge.net/`.

[2] M. Aldinucci & M. Torquati (2009): *FastFlow website*. FastFlow. `http://mc-fastflow.sourceforge.net/`.

[3] R. Barbuti, A. Maggiolo-Schettini, P. Milazzo & G. Pardini (2009): *Spatial Calculus of Looping Sequences*. Electr. Notes Theor. Comput. Sci. 229(1), pp. 21–39. Available at `http://dx.doi.org/10.1016/j.entcs.2009.02.003`.

[4] R. Barbuti, A. Maggiolo-Schettini, P. Milazzo & G. Pardini (2011): *Spatial Calculus of Looping Sequences*. Theoretical Computer Science .

[5] R. Barbuti, A. Maggiolo-Schettini, P. Milazzo, G. Pardini & L. Tesei (2011): *Spatial P systems*. Natural Computing 10(1), pp. 3–16. Available at `http://dx.doi.org/10.1007/s11047-010-9187-z`.

[6] F. Buti, D. Cacciagrano, F. Corradini, E. Merelli & L. Tesei (2010): *BioShape: A Spatial Shape-based Scale-independent Simulation Environment for Biological Systems*. Procedia CS 1(1), pp. 827–835. Available at `http://dx.doi.org/10.1016/j.procs.2010.04.090`.

[7] C. Calcagno, M. Coppo, F. Damiani, M. Drocco, E. Sciacca, S. Spinella & A. Troina (To Appear): *Modelling Spatial Interactions in the AM Symbiosis using CWC*. In: CompMod 2011.

[8] L. Cardelli (2004): *Brane Calculi*. In: Proc. of CMSB'04, LNCS 3082, Springer, pp. 257–278.

[9] L. Cardelli & P. Gardner (2010): *Processes in Space*. In: Proc. of the 6th international conference on Computability in Europe, CiE'10, Springer-Verlag, pp. 78–87.

[10] M. Coppo, F. Damiani, M. Drocco, E. Grassi, M. Guether & A. Troina (2011): *Modelling Ammonium Transporters in Arbuscular Mycorrhiza Symbiosis*. Transactions on Computational Systems Biology XIII, pp. 85–109.

[11] M. Coppo, F. Damiani, M. Drocco, E. Grassi, E. Sciacca, S. Spinella & A. Troina (2010): *Hybrid Calculus of Wrapped Compartments*. In: Proceedings Compendium of the Fourth Workshop on Membrane Computing and Biologically Inspired Process Calculi (MeCBIC'10), 40, EPTCS, pp. 103–121.

[12] M. Coppo, Damiani. F., M. Drocco, Grassi. E. & A. Troina (2010): *Stochastic Calculus of Wrapped Compartments*. In: 8th Workshop on Quantitative Aspects of Programming Languages (QAPL'10), 28, EPTCS, pp. 82–98.

[13] P. Degano, D. Prandi, C. Priami & P. Quaglia (2006): *Beta-binders for Biological Quantitative Experiments*. Electr. Notes Theor. Comput. Sci. 164(3), pp. 101–117. Available at `http://dx.doi.org/10.1016/j.entcs.2006.07.014`.

[14] D. Gillespie (1977): *Exact Stochastic Simulation of Coupled Chemical Reactions*. J. Phys. Chem. 81, pp. 2340–2361.

[15] M.J. Harrison (2005): *Signaling in the Arbuscular Mycorrhizal Symbiosis*. Annu. Rev. Microbiol. 59, pp. 19–42.

[16] I. Jakobsen, LK Abbott & AD Robson (1992): *External Hyphae of Vesicular-arbuscular Mycorrhizal Fungi Associated with Trifolium Subterraneum L. 1. Spread of Hyphae and Phosphorus Inflow into Roots*. New Phytologist 120(3), pp. 371–379.

[17] B. Kholodenko (2006): *Cell-signalling Dynamics in Time and Space*. Nature Reviews Molecular Cell Biology 7, pp. 165–176.

[18] H. Maherali & J.N. Klironomos (2007): *Influence of Phylogeny on Fungal Community Assembly and Ecosystem Functioning*. Science 316(5832), p. 1746.

[19] S. Montagna & M. Viroli (2010): *A Framework for Modelling and Simulating Networks of Cells*. Electr. Notes Theor. Comput. Sci. 268, pp. 115–129. Available at `http://dx.doi.org/10.1016/j.entcs.2010.12.009`.

[20] N. Oury & G. Plotkin (2011): *Multi-Level Modelling via Stochastic Multi-Level Multiset Rewriting*. Draft submitted to MSCS.

[21] T. Parr et al.: *ANTLR website*. `http://www.antlr.org/`.

[22] S. Purin & M.C. Rillig (2008): *Parasitism of Arbuscular Mycorrhizal Fungi: Reviewing the Evidence*. FEMS Microbiology Letters 279(1), pp. 8–14.

[23] S. Ravnskov & I. Jakobsen (1995): *Functional Compatibility in Arbuscular Mycorrhizas Measured as Hyphal P transport to the Plant*. New Phytologist 129(4), pp. 611–618.

[24] A. Regev, E. M. Panina, W. Silverman, L. Cardelli & E. Y. Shapiro (2004): *BioAmbients: An Abstraction for Biological Compartments*. Theor. Comput. Sci. 325(1), pp. 141–167.

[25] A. Schnepf, T. Roose & P. Schweiger (2008): *Growth Model for Arbuscular Mycorrhizal Fungi*. Journal of The Royal Society Interface 5(24), p. 773.

[26] A. Spicher, O. Michel & J.-L. Giavitto (2011): *Interaction-Based Simulations for Integrative Spatial Systems Biology*. In: Understanding the Dynamics of Biological Systems, Springer New York, pp. 195–231.

Appendix: Implementation of the Surface Language

This Section presents a software module implementing the translation of a surface language model into the corresponding standard CWC model that can be executed by the CWC simulator (cf. Sec. 2.4). The module is written in Java by means of the ANTLR parser generator [21]. The input syntax of the software is defined as following.

Patterns, Terms and Open Terms: pattern, terms and open terms follow the syntax of CWC. In the definition of a compartment, its label is written in braces, as the first element in the round brackets. The symbol $\rfloor$ is translated into |, and the empty sequence $\bullet$ into `\e`. If a pattern, term or open term is repeated several times, we write the number of repetitions before it.

Grid Coordinates: the row and the column of a grid coordinate are divided by a comma. All the constructions of the surface language are implemented. The components of a set of coordinates are divided by a blank space.

Directions: for the directions we use the same keywords of the surface language, plus the special identifiers `+`, `x`, `*` to identify all the orthogonal directions (`N`,`S`,`W`,`E`), all the diagonal directions (`NW`,`SW`,`NE`,`SE`) and all the directions, respectively.

Model Name: the name of the model is defined following the syntax

$$\texttt{model}\ \mathit{string}\ \texttt{;}$$

where *string* is the name of the model.

Grid Dimensions: the dimensions of the grid are expressed with the syntax

$$\texttt{grid}\ r\ \texttt{,}\ c\ \texttt{;}$$

where r and c are the number of rows and columns of the grid, respectively.

Grid Cell Construction: the notation of a grid cell $\Theta, \ell_s \boxplus \bar{t}$ is translated into the code line

$$\texttt{cell} < \Theta > \{\ \ell_s\ \}\ \bar{t}\ \texttt{;}$$

the module writes as many CWC compartments as the number of coordinates in Θ: each of these copies has the same label ℓ_s and the same content $\bar{t}$, but a different coordinate in the wrap.

Non Spatial Events: the notation of a non spatial event $\ell : \overline{p} \overset{k}{\longmapsto} \overline{o}$ is translated into the code line

$$\texttt{nse}\ \{\ \ell\ \}\ \overline{p}\ \texttt{[}\ k\ \texttt{]}\ \overline{o}\ \texttt{;}$$

the module translates this line in a unique CWC rule.

Spatial Events: the notation of a spatial event $\Theta \triangleright \ell_s : \overline{p} \overset{k}{\longmapsto} \ell'_s : \overline{o}$ is translated into the code line

$$\texttt{se} < \Theta > \{\ \ell_s\ \}\ \overline{p}\ \texttt{[}\ k\ \texttt{]}\ \{\ \ell'_s\ \}\ \overline{o}\ \texttt{;}$$

As shortcuts, the absence of $< \Theta >$ indicates the whole grid, and the absence of $\{\ \ell'_s\ \}$ indicates that the label of the spatial compartment does not change. The module writes a CWC rule for each coordinate in Θ: a rule differs from the others only in the coordinate written in its wrap.

Spatial Movement Events: the notation of a spatial movement event $\Theta \triangleleft \Delta \triangleright \ell_{s_1}, \ell_{s_2} : \overline{p_1}, \overline{p_2} \overset{k}{\longmapsto} \ell'_{s_1}, \ell'_{s_2} : \overline{o_1}, \overline{o_2}$ is translated into the code line

$$\texttt{sme} < \Theta > \texttt{[}\ \Delta\ \texttt{]}\ \{\ \ell_{s_1}\ \}\ \overline{p_1}\ \{\ \ell_{s_2}\ \}\ \overline{p_2}\ \texttt{[}\ k\ \texttt{]}\ \{\ \ell'_{s_1}\ \}\ \overline{o_1}\ \{\ \ell'_{s_2}\ \}\ \overline{o_2}\ \texttt{;}$$

As shortcuts, the absence of $< \Theta >$ indicates the whole grid, and the absence of $\{\ \ell'_{s_1}\ \}$ or $\{\ \ell'_{s_2}\ \}$ indicates that the label of the spatial compartment does not change. In case of absence of $\{\ \ell'_{s_2}\ \}$, an underscore is used to separate $\overline{o_1}$ and $\overline{o_2}$. For each coordinate in Θ, the module writes as many CWC rules as the number of directions in Δ; in case of a coordinate on the edge of the grid, the module writes a CWC rule for a direction only if this one identifies an adjacent spatial compartment on the grid. The number of CWC rules is therefore less or equal to $|\Theta| \times |\Delta|$.

Monitors: a monitor permits to expose what pattern we need to monitor: at the end of simulation, all the states of this pattern are written in a log file. The syntax to design a monitor is the following:

$$\texttt{monitor}\ \mathit{string} < \Theta > \{\ \ell_s\ \}\ \overline{p}\ \texttt{;}$$

where *string* is a string describing the monitor and $\overline{p}$ is the pattern, contained into a spatial compartment labelled by ℓ_s and the coordinates Θ, to monitor. As shortcut, the absence of $< \Theta >$ indicates the average of the monitors in the whole grid, and the absence of $\{\ \ell_s\ \}$ indicates to write a monitor for each label defined in the model. The module writes a monitor for each coordinate in Θ; in case of the absence of $\{\ell_s\}$, the module writes a monitor for each combination of coordinates in Θ and spatial labels defined in the model.

The construction of a model follows the order used to describe the translator syntax: first the model name and the grid dimensions, then the rules of the model. After the rules, we define the grid cell, and finally the monitors.

Listings 1 and 2 show the input file for the CWC Surface Language software to model the *S. calospora* and *Glomus* sp. fungi growth.

Listing 1: Input file for the Surface Language parser to model the *S. calospora* fungus growth

```
model "AM Fungi Growth Model S. Calospora";

grid 1,13;

//  Tips and hyphae proliferation at the root-fungus interface
se <0,0> {grid} Root [2.52] Root Tip;
se <0,0> {grid} Root [3.5] Root Hyp;

// Tips branching and death
se {grid} Tip [0.02] 2 Tip;
se {grid} Tip [0.0052] \e;

// Hyphae death
se {grid} Hyp [0.18] \e;

// Hyphal creation during a tip shift to a nearby compartment
sme [E W] {grid} Tip {grid} \e [0.125] Hyp _ Tip;

//  Initial state
cell <0,0> {grid} Root 97 Tip 115 Hyp;
cell <rect[0,1 0,12] >  {grid} \e;

monitor "Hyp" <rect[0,0 0,12] > {grid} Hyp;
```

Listing 2: Input file for the Surface Language parser to model the *Glomus* sp. fungus growth

```
model "AM Fungi Growth Model Glomus sp.";

grid 1,13;

//  Tips and hyphae proliferation at the root-fungus interface
se <0,0> {grid} Root [2.24] Root Tip;
se <0,0> {grid} Root [1.04] Root Hyp;

// Tips branching and death
se {grid} Tip [1.91] 2 Tip;
se {grid} Tip [0.15] \e;

// Tips saturation
se {grid} 2 Tip [0.15] \e;

// Hyphae death
se {grid} Hyp [0.28] \e;

// Hyphal creation during a tip shift to a nearby compartment
sme [E W] {grid} Tip {grid} \e [0.065] Hyp _ Tip;

//  Initial state
cell <0,0> {grid} Root 84 Tip 35 Hyp;
cell <rect[0,1 0,12] >  {grid} \e;

monitor "Hyp" <rect[0,0 0,12] > {grid} Hyp;
```

Brane Calculi Systems: A Static Preview of their Possible Behaviour

Chiara Bodei
Dipartimento di Informatica, Università di Pisa
chiara@di.unipi.it

Linda Brodo
Dipartimento di Scienze dei Linguaggi, Università di Sassari
brodo@uniss.it

We improve the precision of a previous Control Flow Analysis for Brane Calculi [4], by adding information on the context and introducing causality information on the membranes. This allows us to prove some biological properties on the behaviour of systems specified in Brane Calculi.

1 Introduction

In [6] Cardelli introduced a family of process calculi, called *Brane Calculi*, endowed with dynamically nested membranes, focussing on the interactions that happen *on* membranes rather than inside them. Brane calculi offer a suitable and formal setting for investigating the behaviour of the specified systems, in order to establish the biological properties of interest. Nevertheless, since the behaviour of a system is usually given in terms of its transition system, whose size can be huge, especially when modelling complex biological systems, its exploration can be computationally hard. One possible solution consists in resorting to static techniques to extract information on the dynamic behaviour and to check the related dynamic properties, without actually running the corresponding program. The price is a loss in precision, because these techniques can only provide approximations of the behaviour. However, we can exploit static results to perform a sort of preliminary and not too much expensive screening of *in silico* experiments. In the tradition [13] of applying static techniques to process calculi used in modelling biological phenomena, we present here a contextual and less approximate extension of the Control Flow Analysis for Brane Calculi introduced in [4]. Control Flow Analysis (CFA) is a static technique, based on Flow Logic [12], that provides a variety of automatic and decidable methods and tools for analysing properties of computing systems. One of the advantages of the CFA is that the obtained information on the behaviour are quite general. As a consequence, a single analysis can suffice for verifying a variety of properties: different inspections of the CFA results permit to check different properties, with no need of re-analysing it several times. Only the values of interest tracked for testing change accordingly and the definitions of the static counterparts of the dynamic properties must be provided. Control Flow Analysis provides indeed a *safe over-approximation* of the *exact* behaviour of a system, in terms of the possible reachable configurations. That is, at least all the valid behaviours are captured. More precisely, all those events that the analysis does not consider as possible will *never* occur. On the other hand, the set of events deemed as possible may, or may not, occur in the actual dynamic evolution of the system. To this end we have improved the precision of the CFA in [4], by adding information on the context (along the lines of [15]) and introducing causality information on the membranes. Also, this extra-information allows us to refine the static checking of properties related to the spatial structure of membranes. Furthermore, we focus on causality, since we believe it plays a key role in the understanding of the behaviour of biological systems, in our case specified in a process algebra like the Brane one. In order to investigate the possibilities of our CFA to capture some kinds of causal dependencies arising in the MBD version of Brane Calculi, we follow [5] and its classification, by applying the analysis to the same key examples.

Proceedings of 5th Workshop on Membrane Computing
and Biologically Inspired Process Calculi (MeCBIC 2011)
Pages 41–56, 2011.

© Chiara Bodei and Linda Brodo
All the rights to the paper remain with the authors.

We observe that the analysis is able to capture some of these dependencies. This is a small improvement in the direction of giving some causal structure to the usually flat CFA results. The gain in precision is paid in terms of complexity: the presented analysis is rather expensive from a computational point of view.

The paper gets in the research stream dedicated to the application of static techniques and, in particular, Control Flow Analysis to bio-inspired process calculi e.g., [13, 3]. Similar to ours are the works devoted to the analysis of BioAmbients [18]. In particular, [15], where the authors introduce a contextual CFA and [16] where a pathway analysis is exploited for investigating causal properties. BioAmbients are analysed using instead Abstract Interpretation in [7, 8, 9]. The analysis presented in [7] records information on the number of occurrences of objects and therefore is able to capture quantitative and causal aspects, necessary to reason on the temporal and spatial structure of processes. In [8], in a different context, the behaviour of processes is safely approximated and the properties of a fragment of Computation Tree Logic is preserved. This makes it possible to address temporal properties and therefore some kinds of causality. Finally, [9] presents a static analysis that computes an abstract transition systems for BioAmbients processes, able to validate temporal properties. Our choice of the Brane calculi depends on the fact they have resulted to be particularly useful for modelling and reasoning about a large class of biological systems, such as the one of the eukaryotic cells that, differently from the prokaryotes, possess a set of internal membranes. Among the first formalisms used to investigate biological membranes there are the P Systems [14], introduced by Păun, which formalise distributed parallel computations biologically-inspired: a biological system is seen as a complex hierarchical structure of nested membranes inspired by the structure of living cells. Finally, besides Brane, there are other calculi of interest for our approach, that have been specifically defined for modelling biological structures such as compartments and membranes, e.g., an extension [11] of κ-calculus [10], Beta Binders [17] and the Calculus of Looping Sequences [2].

The rest of the paper is organised as follows. In Section 2, we present the MBD version of Brane Calculi. We introduce the Control Flow Analysis in Section 3. In Section 4, we exploit our analysis to check some properties related to the hierarchical structure of Brane processes. In Section 5, we discuss on which kind of causal information our CFA can capture. In Section 6, the static treatment of Brane PEP action is added and the whole analysis is applied to a model of infective cycle of the Semliki Forest Virus. Section 7 presents some concluding remarks. Proofs of theorems and lemmata presented throughout the paper are collected in Appendix A.

2 An overview on Brane Calculi

The Brane Calculi [6] are a family of calculi defined to describe the interaction amongst membraned component. Specifically, the membrane interactions are explicitly described by means of a set of membrane-based interaction capabilities. A system consists of nested membranes, as described by the following syntax, where n is taken from a countable set Λ of names.

$$\begin{array}{rll} P,Q ::= & \diamond \mid P \circ Q \mid !P \mid \sigma\langle P\rangle^{\mu} & \text{systems } \Pi \\ \sigma,\tau ::= & 0 \mid \sigma|\tau \mid !\sigma \mid a.\sigma & \text{membrane processes } \Sigma \\ a,b ::= & mate_n \mid mate_n^{\perp} \mid bud_n \mid bud_n^{\perp}(\rho) \mid drip(\rho) & \text{MBD actions } \Xi_{MBD} \end{array}$$

The basic structure of a system consists of (sub-)system composition, represented by the monoidal operator $\circ$ (associative, commutative and with $\diamond$ as neutral element). Replication ! is used to represent the composition of an unbounded number of systems or membrane processes. $\sigma\langle P\rangle^{\mu}$ is a *membrane* with

$(\mathscr{S}/_{\equiv}, \circ, \diamond)$ is a commutative monoid	$(\mathscr{B}/_{\equiv}, \vert, 0)$ is a commutative monoid
$!\diamond \equiv \diamond$	$!0 \equiv 0$
$!(P \circ Q) \equiv !P \circ !Q$	$!(\sigma \vert \tau) \equiv !\sigma \vert !\tau$
$!!P \equiv !P$	$!!\sigma \equiv !\sigma$
$!P \equiv P \circ\, !P$	$!\sigma \equiv \sigma \vert !\sigma$
$0\langle\diamond\rangle^{\mu} \equiv \diamond$	
$\sigma \equiv \tau \Rightarrow \sigma \vert \rho \equiv \tau \vert \rho$	$P \equiv Q \Rightarrow P \circ R \equiv Q \circ R$
$\sigma \equiv \tau \Rightarrow !\sigma \equiv !\tau$	$P \equiv Q \Rightarrow !P \equiv !Q$
$\sigma \equiv \tau \Rightarrow a.\sigma \equiv a.\tau$	$P \equiv Q \wedge \sigma \equiv \tau \Rightarrow \sigma\langle P\rangle^{\mu} \equiv \tau\langle Q\rangle^{\mu}$

Table 1: Structural Congruence for Brane Calculi.

content P and interaction capabilities represented by the process σ. Note that, following [4], we annotate membranes with a unique label μ so as to distinguish the different syntactic occurrences of a membrane. Note that these labels have no semantic meaning, but they are useful for our CFA. We refer to $\mu \in \mathbf{M}$ as the identity of the membrane $\sigma\langle P\rangle^{\mu}$, where $\mathbf{M}$ is the finite set of membrane identities. We assume that each considered system is contained in an ideal outermost membrane, identified by a distinguished element $* \in \mathbf{M}$.

Membranes exhibit interaction capabilities, like the MBD set of actions that model membrane fusion and splitting. The former is modelled by the *mating* operation, the latter can be rendered both by *budding*, that consists in splitting off exactly one internal membrane, and *dripping*, that consists in splitting off one empty membrane. For the sake of simplicity, we focus here on the fragment of the calculus without communication primitives and molecular complexes, and with only the MBD actions. The treatment of the alternative set of PEP actions is analogous and it is postponed to Section 5, where it is briefly introduced.

Membrane processes σ consist of the empty process 0, the parallel composition of two processes, represented by the monoidal | operator with 0 as neutral element, the replication of a process and of the process that executes an interaction *action a* and then behaves as another process σ. Actions for mating ($mate_n$) and budding (bud_n) have the corresponding co-actions ($mate_n^{\perp}$, $bud_n^{\perp}$ resp.) to synchronise with. Here n, which identifies a pair of complementary action and co-action that can interact, is taken from a countable set Λ of names. The actions $bud_n^{\perp}(\rho)$ and $drip(\rho)$ are equipped with a process ρ associated to the membrane that will be created when performing budding and dripping actions.

The semantics of the calculi is given in terms of a transition system defined up to a structural congruence and reduction rules. The standard *structural congruence* $\equiv$ on systems Π and membranes Ξ is the least congruence satisfying the clauses in Table 1. Reduction rules complete the definition of the interleaving semantics. They consist of the basic reaction rules, valid for all brane calculi (upper part of Table 2) and by the reaction axioms for the MBD version (lower part of Table 2). We use the symbol $\rightarrow^*$ for the reflexive and transitive closure of the transition relation $\rightarrow$.

They are quite self-explanatory and we make only a few observations about the labels treatment. Given a system, the set of its membrane identities is finite. Indeed, the structural congruence rule imposes that $!\sigma\langle P\rangle^{\mu} \equiv \sigma\langle P\rangle^{\mu} \circ !\sigma\langle P\rangle^{\mu}$, i.e. no new identity label μ is introduced by recursive calls. A distinguished membrane identity is needed each time a new membrane is generated as a consequence of a performed action, e.g. the new membrane obtained by the fusion of two membranes after a mate synchronisation. To determine such labels we exploit the functions $\mathbf{MI_{mate}}$, $\mathbf{MI_{bud}}$, and $\mathbf{MI_{drip}}$ that return fresh and distinct membrane identities, depending on the actions and on their syntactic contexts [4]. Recall that the number of needed membrane identities is finite, as finite are the possible combinations of actions and contexts. Therefore, we choose these functions in such a way that, given an action and

$$(Par)\ \frac{P \to Q}{P \circ R \to Q \circ R} \qquad (Brane)\ \frac{P \to Q}{\sigma\langle P\rangle^{\mu} \to \sigma\langle Q\rangle^{\mu}} \qquad (Struct)\ \frac{P \equiv P' \ \wedge\ P' \to Q' \ \wedge\ Q' \equiv Q}{P \to Q}$$

$(Mate)$ $mate_n.\sigma|\sigma_0\langle P\rangle^{\mu_P} \circ mate_n^{\perp}.\tau|\tau_0\langle Q\rangle^{\mu_Q} \to \sigma|\sigma_0|\tau|\tau_0\langle P \circ Q\rangle^{\mu_{PQ}}$
where $\mu_{PQ} = \mathbf{MI}_{\mathbf{mate}}(mate_n, \mu_P, mate_n^{\perp}, \mu_Q, \mu_{gp}, \mu_p, \mu)$,
μ identifies the closest membrane surrounding μ_P and μ_Q in the context $\mu_{gp}\mu_p$

(Bud) $bud_n^{\perp}(\rho).\tau|\tau_0\langle bud_n.\sigma|\sigma_0\langle P\rangle^{\mu_P} \circ Q\rangle^{\mu_Q} \to \rho\langle\sigma|\sigma_0\langle P\rangle^{\mu_P}\rangle^{\mu_R} \circ \tau|\tau_0\langle Q\rangle^{\mu_Q}$
where $\mu_R = \mathbf{MI}_{\mathbf{bud}}(bud_n, \mu_P, bud_n^{\perp}, \mu_Q, \mu_{gp}, \mu_p, \mu)$,
μ identifies the closest membrane surrounding μ_Q in the context $\mu_{gp}\mu_p$

$(Drip)$ $drip(\rho).\sigma|\sigma_0\langle P\rangle^{\mu_P} \to \rho\langle\rangle^{\mu_R} \circ \sigma|\sigma_0\langle P\rangle^{\mu_P}$
where $\mu_R = \mathbf{MI}_{\mathbf{drip}}(drip(\rho), \mu_P, \mu_{gp}, \mu_p, \mu)$,
μ identifies the closest membrane surrounding μ_P in the context $\mu_{gp}\mu_p$

Table 2: Reduction Semantics for Brane Calculi.

the identities of the membranes on which the action (and the corresponding co-action, if any) reside, the function includes the membrane identity needed to identify the membrane obtained by firing that action.

3 A Contextual CFA for Brane Calculi

We present an extension of the Control Flow Analysis (CFA), introduced in [4] for analysing system specified in Brane Calculi. The analysis over-approximates all the possible behaviour of a top-level system P_*. In particular, the analysis keeps track of the possible contents of each membrane, thus taking care of the possible modifications of the containment hierarchy due to the dynamics. The new analysis, following [15], incorporates context in the style of 2CFA, thus increasing the precision of the approximations w.r.t. [4]. Furthermore, the analysis exploits some causality information to further reduce the degree of approximation. A localised approximation of the contents of a membrane or *estimate* $\mathscr{I}$ is defined as follows:

$$\mathscr{I} \subseteq \mathbf{M} \times \mathbf{M} \times \mathbf{M} \times (\mathbf{M} \cup \Xi_{MBD})$$

Here, $\mu_s \in \mathscr{I}(\mu_{gp}, \mu_p, \mu)$ (that is $(\mu_{gp}, \mu_p, \mu, \mu_s) \in \mathscr{I}$) means that the membrane identified by μ may surround the membrane identified by μ_s, whenever μ is surrounded by μ_p and μ_p is surrounded by μ_{gp}. The outermost membranes $\mu_{gp}\mu_p$ represent what is called the *context* and that amounts to $**$ when the analysed membrane is at top-level. Moreover, $a \in \mathscr{I}(\mu_{gp}, \mu_p, \mu)$ means that the action a may reside on and affect the membrane identified by μ, in the context $\mu_{gp}\mu_p$. Furthermore, the analysis collects two types of some causality information:

- An approximation of the possible causal circumstances in which a membrane can arise:

$$\mathscr{C} \subseteq (\Xi_{MBD} \times \mathbf{M} \times \Xi_{MBD} \times \mathbf{M} \times \mathbf{M} \times \mathbf{M} \times \mathbf{M}) \uplus (\Xi_{MBD} \times \mathbf{M} \times \mathbf{M} \times \mathbf{M} \times \mathbf{M})$$

  Here $(a_n, \mu_P, a_n^{\perp}, \mu_Q, \mu_{gp}, \mu_p, \mu) \in \mathscr{C}(\mu_c)$ means that the membrane μ_c can be *causally derived* by the firing of the action a_n in μ_P and the coaction $a_n^{\perp}$ in μ_Q, in the context $\mu_{gp}\mu_p, \mu$. Similarly, $(a_n, \mu_P, \mu_{gp}, \mu_p, \mu) \in \mathscr{C}(\mu_c)$ for an action a_n like *drip*, without a co-action.

- An approximation of the possible membrane incompatibilities:

$$\mathscr{R} \subseteq (\mathbf{M} \times \mathbf{M} \times \mathbf{M}) \times (\mathbf{M} \times \mathbf{M} \times \mathbf{M})$$

  Here, $((\mu_{gp}, \mu_p, \mu), (\mu'_{gp}, \mu'_p, \mu)) \in \mathscr{R}$ means that the membrane μ in the context $\mu_{gp}\mu_p$ cannot interact with the membrane μ in the context $\mu'_{gp}\mu'_p$, because the second membrane is obtained from the first and the first one is dissolved.

Note that $\mathscr{C}$ and $\mathscr{R}$ are two strict order relations, thus only transitivity property holds. To validate the correctness of a proposed estimate $\mathscr{I}$, we state a set of clauses operating upon judgements like $\mathscr{I}, \mathscr{C}, \mathscr{R} \models^{\mu_{gp}\mu_p\mu} P$. This judgement expresses that when the subprocess P of P_* is enclosed within a membrane identified by $\mu \in \mathbf{M}$, in the context $\mu_{gp}\mu_p \in \mathbf{M} \times \mathbf{M}$, then $\mathscr{I}$ correctly captures the behaviour of P, i.e. the estimate is valid also for all the states Q passed through a computation of P.

The analysis is specified in two phases. First, it checks that $\mathscr{I}$ describes the initial process. This is done in the upper part of Table 3, where the clauses amount to a syntax-driven structural traversal of process specification. The clauses rely on the auxiliary function A that collects all the actions in a membrane process σ and that is reported at the beginning of Table 3. Note that the actions collected by A, e.g., in $\sigma = \sigma_0.\sigma_1$ are equal to the ones in $\sigma' = \sigma_0|\sigma_1$, witnessing the fact that here the analysis introduces some imprecision and approximation. he clause for membrane system $\sigma\langle P\rangle^{\mu_s}$ checks that whenever a membrane μ_s is introduced inside a membrane μ, in the context μ_{gp}, μ_p the relative hierarchy position must be reflected in $\mathscr{I}$, i.e. $\mu_s \in \mathscr{I}(\mu_{gp}, \mu_p, \mu)$. Furthermore, the actions in σ that affect the membrane μ_s and that are collected in $A(\sigma)$, are recorded in $\mathscr{I}(\mu_p, \mu, \mu_s)$. Finally, when inspecting the content P, the fact that the enclosing membrane is μ_s in the context $\mu_p\mu$ is recorded, as reflected by the judgement $\mathscr{I}, \mathscr{C}, \mathscr{R} \models^{\mu_p\mu\mu_s} P$. The rule for $\diamond$ does not restrict the analysis result, while the rules for parallel composition $\circ$, and replication ! ensure that the analysis also holds for the immediate subsystems, by ensuring their traversal. In particular, note that the analysis of $!P$ is equal to the one of P. This is another source of imprecision.

Secondly, the analysis checks that $\mathscr{I}$ also takes into account the dynamics of the process under consideration; in particular, the dynamics of the containment hierarchy of membranes. This is expressed by the closure conditions in the lower part of Table 3 that mimic the semantics, by modelling, without exceeding the precision boundaries of the analysis, the semantic preconditions and the consequences of the possible actions. More precisely, each precondition checks whether a pair of complementary actions could possibly enable the firing of a transition according to $\mathscr{I}$. The conclusion imposes the additional requirements on $\mathscr{I}$ that are necessary to give a valid prediction of the analysed action.

Consider e.g., the clause for $(Mate)$ (the other clauses are similar). If (i) there exists an occurrence of a mate action: $mate_n \in \mathscr{I}(\mu_p, \mu, \mu_P)$; (ii) there exists an occurrence of the corresponding co-mate action: $mate_n^{\perp} \in \mathscr{I}(\mu_p, \mu, \mu_Q)$; (iii) the corresponding membranes are siblings: $\mu_P, \mu_Q \in \mathscr{I}(\mu_{gp}, \mu_p, \mu)$, (iv) the redexes are not incompatible, i.e. the corresponding membranes can interact: $((\mu_p, \mu, \mu_P), (\mu_p, \mu, \mu_Q)) \notin \mathscr{R}$ then the conclusion of the clause expresses the effects of performing the transition $(Mate)$. In this case, we have that $\mathscr{I}$ must reflect that (i) there may exist a membrane μ_{PQ} inside μ, in the context $\mu_{gp}\mu_p$, at the same nesting level of the membranes μ_P and μ_Q; and (ii) the contents of μ_P and of μ_Q, their children and their grandchildren, may also be included in μ_{PQ}. Note that the contribution changes depending on whether we consider μ_P (μ_Q, resp.), their children or their grandchildren. With the inclusion $\mathscr{I}(\mu, \mu_P) \subseteq \mathscr{I}(\mu, \mu_{PQ})$ we mean that for each μ_s in the context μ, μ_P, all the elements in $\mathscr{I}(\mu, \mu_P, \mu_s)$ are included in $\mathscr{I}(\mu, \mu_{PQ}, \mu_s)$. Similarly, with $\mathscr{I}(\mu_P) \subseteq \mathscr{I}(\mu_{PQ})$ we mean that for each μ_{gs} in the context $\mu_P\mu_s$, and in turn for each μ_s in the context μ, μ_P, all the elements in $\mathscr{I}(\mu_P, \mu_s, \mu_{gs})$ belong to $\mathscr{I}(\mu_{PQ}, \mu_s, \mu_{gs})$. We use a similar notation for the relation $\mathscr{R}$. (iii) The membrane μ_{PQ} is the result of the transition $(Mate)$,

$$A(0)=\emptyset \qquad A(a.\sigma)=\{a\}\cup A(\sigma) \qquad A(!\sigma)=A(\sigma) \qquad A(\sigma_0|\sigma_1)=A(\sigma_0)\cup A(\sigma_1)$$

$$\begin{array}{lll}
\mathscr{I},\mathscr{C},\mathscr{R} \models^{\mu_{gp}\mu_p\mu} \diamond & \text{iff} & true \\
\mathscr{I},\mathscr{C},\mathscr{R} \models^{\mu_{gp}\mu_p\mu} P\circ Q & \text{iff} & \mathscr{I},\mathscr{C},\mathscr{R} \models^{\mu_{gp}\mu_p\mu} P \wedge \mathscr{I},\mathscr{C},\mathscr{R} \models^{\mu_{gp}\mu_p\mu} Q \\
\mathscr{I},\mathscr{C},\mathscr{R} \models^{\mu_{gp}\mu_p\mu} !P & \text{iff} & \mathscr{I},\mathscr{C},\mathscr{R} \models^{\mu_{gp}\mu_p\mu} P \\
\mathscr{I},\mathscr{C},\mathscr{R} \models^{\mu_{gp}\mu_p\mu} \sigma\langle P\rangle^{\mu_s} & \text{iff} & \mu_s \in \mathscr{I}(\mu_{gp},\mu_p,\mu) \wedge A(\sigma)\subseteq \mathscr{I}(\mu_p,\mu,\mu_s) \wedge \mathscr{I},\mathscr{C},\mathscr{R} \models^{\mu_p,\mu,\mu_s} P
\end{array}$$

$$\begin{array}{ll}
(Mate) & mate_n \in \mathscr{I}(\mu_p,\mu,\mu_P) \wedge mate_n^{\perp} \in \mathscr{I}(\mu_p,\mu,\mu_Q) \wedge \mu_P,\mu_Q \in \mathscr{I}(\mu_{gp},\mu_p,\mu) \wedge \\
& ((\mu_p,\mu,\mu_P),(\mu_p,\mu,\mu_Q)) \notin \mathscr{R} \\
& \Rightarrow \mu_{PQ} \in \mathscr{I}(\mu_{gp},\mu_p,\mu) \text{ where } \mu_{PQ} = \mathbf{MI_{mate}}(mate_n,\mu_P,mate_n^{\perp},\mu_Q,\mu_{gp},\mu_p,\mu) \wedge \\
& \mathscr{I}(\mu_p,\mu,\mu_P) \subseteq \mathscr{I}(\mu_p,\mu,\mu_{PQ}) \wedge \mathscr{I}(\mu,\mu_P) \subseteq \mathscr{I}(\mu,\mu_{PQ}) \wedge \mathscr{I}(\mu_P) \subseteq \mathscr{I}(\mu_{PQ}) \\
& \mathscr{I}(\mu_p,\mu,\mu_Q) \subseteq \mathscr{I}(\mu_p,\mu,\mu_{PQ}) \wedge \mathscr{I}(\mu,\mu_Q) \subseteq \mathscr{I}(\mu,\mu_{PQ}) \wedge \mathscr{I}(\mu_Q) \subseteq \mathscr{I}(\mu_{PQ}) \\
& \wedge (mate_n,\mu_P,mate_n^{\perp},\mu_Q,\mu_{gp},\mu_p,\mu) \in \mathscr{C}(\mu_{PQ}) \wedge \\
& ((\mu_p,\mu,\mu_P),(\mu_p,\mu,\mu_{PQ})),((\mu_p,\mu,\mu_Q),(\mu_p,\mu,\mu_{PQ})) \in \mathscr{R} \wedge \\
& ((\mu,\mu_P),(\mu,\mu_{PQ})),((\mu,\mu_Q),(\mu,\mu_{PQ})) \in \mathscr{R} \\
\\
(Bud) & bud_n \in \mathscr{I}(\mu,\mu_Q,\mu_P) \wedge bud_n^{\perp}(\rho) \in \mathscr{I}(\mu_p,\mu,\mu_Q) \wedge \mu_P \in \mathscr{I}(\mu_p,\mu,\mu_Q) \wedge \mu_Q \in \mathscr{I}(\mu_{gp},\mu_p,\mu) \\
& \Rightarrow \mu_R \in \mathscr{I}(\mu_{gp},\mu_p,\mu) \text{ where } \mu_R = \mathbf{MI_{bud}}(bud_n,\mu_P,bud_n^{\perp}(\rho),\mu_Q,\mu_{gp},\mu_p,\mu) \wedge \\
& A(\rho) \subseteq \mathscr{I}(\mu_p,\mu,\mu_R) \wedge \mu_P \in \mathscr{I}(\mu_p,\mu,\mu_R) \wedge \\
& \mathscr{I}(\mu,\mu_Q,\mu_P) \subseteq \mathscr{I}(\mu,\mu_R,\mu_P) \wedge \mathscr{I}(\mu_Q,\mu_P) \subseteq \mathscr{I}(\mu_R,\mu_P) \\
& \wedge (bud_n,\mu_P,bud_n^{\perp}(\rho),\mu_Q,\mu_{gp},\mu_p,\mu) \in \mathscr{C}(\mu_R) \\
& \wedge ((\mu,\mu_Q,\mu_P),(\mu,\mu_R,\mu_P)) \in \mathscr{R} \wedge ((\mu_Q,\mu_P),(\mu_R,\mu_P)) \in \mathscr{R} \\
\\
(Drip) & drip(\rho) \in \mathscr{I}(\mu_p,\mu,\mu_P) \wedge \mu_P \in \mathscr{I}(\mu_{gp},\mu_p,\mu) \\
& \Rightarrow \mu_R \in \mathscr{I}(\mu_{gp},\mu_p,\mu) \text{ where } \mu_R = \mathbf{MI_{drip}}(drip(\rho),\mu_P,\mu_{gp},\mu_p,\mu) \wedge A(\rho) \subseteq \mathscr{I}(\mu_p,\mu,\mu_R) \\
& \wedge (drip(\rho),\mu_P,\mu_{gp},\mu_p,\mu) \in \mathscr{C}(\mu_R)
\end{array}$$

Table 3: Analysis for Brane Processes

performed by the two membranes μ_P and μ_Q, in the context $\mu_{gp}\mu_p\mu$, as witnessed by the corresponding entry in the component $\mathscr{C}$; (iv) the new membrane μ_{PQ} is *incompatible* with the μ_P and μ_Q, because μ_{PQ}, derived by the transition (*Mate*), follows both μ_P and μ_Q. Note the similar incompatibility between the membrane μ_P in the context $\mu\mu_Q$ before the (*Bud*) transition and the derived one μ_P in the context $\mu\mu_R$. The above requirements correspond to the application of the semantic rule (*Mate*) that would result in the fusion of the two membranes.

Note that, since the new membrane μ_{PQ} inherits the prefix actions that affected the membranes μ_P and μ_Q, it inherits also $mate_n$ and $mate_n^{\perp}$ (we write in red this kind of *imprecise* inclusions). This is due to over-approximation, even though it is harmless: the two prefix actions cannot be further used to predict a communication because they both occur in $\mathscr{I}(\mu_p,\mu,\mu_{PQ})$. Still, the presence of both $mate_n \in \mathscr{I}(\mu_p,\mu,\mu_P)$ and $mate_n^{\perp} \in \mathscr{I}(\mu_p,\mu,\mu_{PQ})$ could lead to predict another interaction that is impossible at run time. Thanks to $\mathscr{R}$, we can safely exclude it, thus gaining precision. This gain is obtained in general: $\mathscr{R}$ collects indeed pairs of capabilities that could be syntactically compatible with an interaction, but that cannot really interact, because they dynamically occur in membranes that are not simultaneously present.

$\mu_{P_0},\mu_{P_1} \in \mathscr{I}(*,*,\mu_P), \mu_P,\mu_Q \in \mathscr{I}(*,*,*)$					$\mu_{P_0},\mu_{P_1} \in \mathscr{I}(*,*,\mu_{PQ}), \mu_{PQ} \in \mathscr{I}(*,*,*)$
$\mu^i_{R1} \in \mathscr{I}(*,*,*), \mu_{P_0} \in \mathscr{I}(*,*,\mu^i_{R1})\ (i=0,1)$					$\mu_{R2} \in \mathscr{I}(*,*,*), \mu_{P_1} \in \mathscr{I}(*,*,\mu_{R2})$
$mate_n$	$\in$	$\mathscr{I}(*,*,\mu_P)$,	$\mathscr{I}(*,*,\mu_{PQ})$,		
$mate^\perp_n$	$\in$	$\mathscr{I}(*,*,\mu_Q)$,	$\mathscr{I}(*,*,\mu_{PQ})$,		
bud_m	$\in$	$\mathscr{I}(*,\mu_P,\mu_{P_0})$,	$\mathscr{I}(*,\mu_{PQ},\mu_{P_0})$,	$\mathscr{I}(*,\mu^i_{R1},\mu_{P_0})$,	
$bud^\perp_m(\rho_1)$	$\in$	$\mathscr{I}(*,*,\mu_P)$,	$\mathscr{I}(*,*,\mu_{PQ})$,		
bud_o	$\in$	$\mathscr{I}(*,\mu_P,\mu_{P_1})$,	$\mathscr{I}(*,\mu_{PQ},\mu_{P_1})$,	$\mathscr{I}(*,\mu_{R2},\mu_{P_1})$,	
$bud^\perp_o(\rho_2)$	$\in$	$\mathscr{I}(*,*,\mu_Q)$,	$\mathscr{I}(*,*,\mu_{PQ})$,		
$((*,*,\mu_P),(*,*,\mu_{PQ})) \in \mathscr{R}$					$((*,*,\mu_Q),(*,*,\mu_{PQ})) \in \mathscr{R}$
$((*,\mu_P,\mu_{P_0}),(*,\mu_{PQ},\mu_{P_0})) \in \mathscr{R}$					$((*,\mu_P,\mu_{P_1}),(*,\mu_{PQ},\mu_{P_1})) \in \mathscr{R}$
$((*,\mu_P,\mu_{P_0}),(*,\mu^i_{R1},\mu_{P_0}) \in \mathscr{R}$					$((*,\mu_P,\mu_{P_1}),(*,\mu_{R2},\mu_{P_1})) \in \mathscr{R}$
$(mate_n,\mu_P,mate^\perp_n,\mu_Q,*,*,*) \in \mathscr{C}(\mu_{PQ})$					$(bud_0,\mu_{P_1},bud^\perp_o,\mu_{PQ},*,*,*) \in \mathscr{C}(\mu_{R2})$
$(bud_m,\mu_{P_0},bud^\perp_m,\mu_{PQ},*,*,*) \in \mathscr{C}(\mu^i_{R1})$					

Table 4: Some entries of the Example 1 Analysis

The gain in precision is paid in terms of complexity: the presented analysis is rather expensive from a computational point of view, due to the introduction of contexts and to the possibly high number of different membrane names. Both these features may lead to an explosion of the possible reachable configurations.

Example 1. *To illustrate how our CFA work we use two simple examples. The emphasis is on the process algebraic structures and not on their biological expressiveness. We first report an application of it to a simple process P, illustrated in [4] (and in turn taken from [5]). We consider P and the following possible computations, where ρ_1 and ρ_2 are not specified as they are not relevant here.*

$$P = (mate_n|bud^\perp_m(\rho_1))\langle bud_m\langle\rangle^{\mu_{P_0}} \circ bud_o\langle\rangle^{\mu_{P_1}}\rangle^{\mu_P} \circ (mate^\perp_n|bud^\perp_o(\rho_2))\langle\rangle^{\mu_Q} \xrightarrow{mate_n}$$
$$P_1 = (bud^\perp_m(\rho_1)|bud^\perp_o(\rho_2))\langle bud_m\langle\rangle^{\mu_{P_0}} \circ bud_o\langle\rangle^{\mu_{P_1}} \circ \diamond\rangle^{\mu_{PQ}} \xrightarrow{bud_m}$$
$$P_2 = \rho_1\langle\langle\rangle^{\mu_{P0}}\rangle^{\mu^0_{R1}} \circ bud^\perp_o(\rho_2))\langle bud_o\langle\rangle^{\mu_{P_1}} \circ \diamond\rangle^{\mu_{PQ}} \xrightarrow{bud_o}$$
$$P_3 = \rho_1\langle\langle\rangle^{\mu_{P0}}\rangle^{\mu^0_{R1}} \circ \rho_2\langle\langle\rangle^{\mu_{P1}}\rangle^{\mu_{R2}} \circ \langle\diamond\rangle^{\mu_{PQ}}$$

$$P = (mate_n|bud^\perp_m(\rho_1))\langle bud_m\langle\rangle^{\mu_{P_0}} \circ bud_o\langle\rangle^{\mu_{P_1}}\rangle^{\mu_P} \circ (mate^\perp_n|bud^\perp_o(\rho_2))\langle\rangle^{\mu_Q} \xrightarrow{bud_m}$$
$$P'_1 = \rho_1\langle\langle\rangle^{\mu_{P0}}\rangle^{\mu^1_{R1}} \circ mate_n\langle\langle bud_o\rangle^{\mu_{P_1}}\rangle^{\mu_P} \circ (mate^\perp_n|bud^\perp_o(\rho_2))\langle\rangle^{\mu_Q} \xrightarrow{mate_m}$$
$$P'_2 = \rho_1\langle\langle\rangle^{\mu_{P0}}\rangle^{\mu^1_{R1}} \circ bud^\perp_o(\rho_2)\langle\langle bud_o\rangle^{\mu_{P_1}}\rangle^{\mu_{PQ}} \xrightarrow{bud_o}$$
$$P'_3 = \rho_1\langle\langle\rangle^{\mu_{P0}}\rangle^{\mu^1_{R1}} \circ \rho_2\langle\langle\rangle^{\mu_{P1}}\rangle^{\mu_{R2}} \circ \langle\diamond\rangle^{\mu_{PQ}}$$

The main entries of the analysis are reported in Table 4, where $**$ identifies the ideal outermost context in which the system top-level membranes are. We write in red the entries due to approximations, but not reflecting the dynamics. Furthermore, we pair the inclusions of actions and of the corresponding co-actions, in order to emphasise which are the pairs of prefixes that lead to the prediction of a possible communication. It is easy to check that $\mathscr{I}$ is a valid estimate by following the two stage procedure explained above.

To understand in which way the $\mathscr{R}$ component refines the analysis, note that since the analysis entries include $mate_n \in \mathscr{I}(*,*,\mu_{PQ})$ and $mate^\perp_n \in \mathscr{I}(*,*,\mu_Q)$, without the check on the $\mathscr{R}$ component, we can predict a transition between the two membranes μ_{PQ} and μ_P. This transition is not possible instead, because μ_{PQ} is causally derived by μ_P.

Note that although the CFA offers in general an over-approximation of the possible dynamic behaviour, in this example the result is rather precise. The transition $mate_n$ is predicted as possible, since its precondition requirements are satisfied. Indeed, we have that $mate_n \in \mathscr{I}(*,*,\mu_P)$ $mate^\perp_n \in \mathscr{I}(*,*,\mu_Q)$,

$$\boxed{\begin{array}{llll}
\multicolumn{4}{l}{\mu_P, \mu_Q, \mu_{PQ} \in \mathscr{I}(*,*,*),} \\
\multicolumn{4}{l}{\mu_{P_0}, \mu_{P_1} \in \mathscr{I}(*,*,\mu_P), \mathscr{I}(*,*,\mu_{PQ}), \mu_{Q_0} \in \mathscr{I}(*,*,\mu_Q), \mathscr{I}(*,*,\mu_{PQ})} \\
\multicolumn{4}{l}{\mu_{P_0Q_0} \in \mathscr{I}(*,*,\mu_{PQ}), \mu_{P_0Q_0P_1} \in \mathscr{I}(*,*,\mu_{PQ}),} \\
\multicolumn{4}{l}{\mu_{P_0P_1} \in \mathscr{I}(*,*,\mu_{PQ}), \mu'_{P_0P_1} \in \mathscr{I}(*,*,\mu_P), \mathscr{I}(*,*,\mu_{PQ}),} \\
\multicolumn{4}{l}{\mu_{P_0P_1Q_0} \in \mathscr{I}(*,*,\mu_{PQ}), \mu'_{P_0P_1Q_0} \in \mathscr{I}(*,*,\mu_{PQ}),} \\
\quad mate_n \in & \mathscr{I}(*,*,\mu_P), & & \\
\quad mate_n^{\perp} \in & \mathscr{I}(*,*,\mu_Q), & & \\
\quad mate_m \in & \mathscr{I}(*,\mu_P,\mu_{P_0}), & \mathscr{I}(*,\mu_{PQ},\mu_{P_0}), & \mathscr{I}(*,\mu_{PQ},\mu_{P_0P_1}), \\
\quad mate_m^{\perp} \in & \mathscr{I}(*,\mu_Q,\mu_{Q_0}), & \mathscr{I}(*,\mu_{PQ},\mu_{Q_0}), & \mathscr{I}(*,\mu_{PQ},\mu_{Q_0}), \\
\quad mate_m \in & \mathscr{I}(*,\mu_P,\mu'_{P_0P_1}), & \mathscr{I}(*,\mu_{PQ},\mu'_{P_0P_1}), & \\
\quad mate_m^{\perp} \in & \mathscr{I}(*,\mu_Q,\mu_{Q_0}), & \mathscr{I}(*,\mu_{PQ},\mu_{Q_0}), & \\
\quad mate_o \in & \mathscr{I}(*,\mu_P,\mu_{P_0}), & \mathscr{I}(*,\mu_{PQ},\mu_{P_0Q_0}), & \mathscr{I}(*,\mu_{PQ},\mu_{P_0}), \\
\quad mate_o^{\perp} \in & \mathscr{I}(*,\mu_P,\mu_{P_1}), & \mathscr{I}(*,\mu_{PQ},\mu_{P_1}) & \mathscr{I}(*,\mu_{PQ},\mu_{P_1}), \\
\multicolumn{4}{l}{((*,*,\mu_P),(*,*,\mu_{PQ})) \in \mathscr{R}} \\
\multicolumn{4}{l}{((*,*,\mu_Q),(*,*,\mu_{PQ})) \in \mathscr{R}} \\
\multicolumn{4}{l}{((*,\mu_P,\mu_{P_i}),(*,\mu_{PQ},\mu_{P_i})) \in \mathscr{R}\ i = 0,1} \\
\multicolumn{4}{l}{(mate_n, \mu_P, mate_n^{\perp}, \mu_Q, *,*,*) \in \mathscr{C}(\mu_{PQ})} \\
\multicolumn{4}{l}{(mate_m, \mu_{P_0}, mate_n^{\perp}, \mu_{Q_0}, *,*,\mu_{PQ}) \in \mathscr{C}(\mu_{P_0Q_0})} \\
\multicolumn{4}{l}{(mate_m, \mu_{P_0P_1}, mate_m^{\perp}, \mu_{Q_0}, *,*,\mu_{PQ}) \in \mathscr{C}(\mu_{P_0P_1Q_0})} \\
\multicolumn{4}{l}{(mate_o, \mu_{P_0Q_0}, mate_o^{\perp}, \mu_{P_1}, *,*,\mu_{PQ}) \in \mathscr{C}(\mu_{P_0Q_0P_1})} \\
\multicolumn{4}{l}{(mate_o, \mu_{P_0}, mate_o^{\perp}, \mu_{P_1}, *,*,\mu_{PQ}) \in \mathscr{C}(\mu_{P_0P_1})} \\
\multicolumn{4}{l}{(mate_o, \mu_{P_0}, mate_o^{\perp}, \mu_{P_1}, *,*,\mu_P) \in \mathscr{C}(\mu'_{P_0P_1})}
\end{array}}$$

Table 5: Some entries of the Example 2 Analysis

and μ_P and μ_Q are sibling and causally compatible membranes. Also the transition on bud_m is initially possible and this result is actually predicted by the analysis, since $bud_m \in \mathscr{I}(*,\mu_P,\mu_{P_0})$ and $bud_m^{\perp} \in \mathscr{I}(*,*,\mu_P)$, with $\mu_{P_0} \in \mathscr{I}(*,*,\mu_P)$, i.e. P is the father of P_0. Instead, we can observe that the transition on bud_o cannot be performed in the initial system. Indeed, bud_o resides on the membrane μ_{P_1} in the context $*\mu_P$, while the coaction $bud_o^{\perp}$ resides on μ_Q that is not the father of μ_{P_1}. The transition on bud_o can be performed instead in the membrane μ_{P_1} in the context $*\mu_{PQ}$, that is the membrane introduced by the previous $mate_n$ transition.

Example 2. *We now apply our CFA to another process P, taken from [5]. We consider P and the following possible computations.*

$$\begin{array}{l}
P = mate_n\langle (mate_m|mate_o)\langle\rangle^{\mu_{P_0}} \circ mate_o^{\perp}\langle\rangle^{\mu_{P_1}}\rangle^{\mu_P} \circ mate_n^{\perp}.\langle mate_m^{\perp}\langle\rangle^{\mu_{Q_0}}\rangle^{\mu_Q} \xrightarrow{mate_n} \\
\quad P_1 = \langle (mate_m|mate_o)\langle\rangle^{\mu_{P_0}} \circ mate_o^{\perp}\langle\rangle^{\mu_{P_1}} \circ mate_m^{\perp}\langle\rangle^{\mu_{Q_0}}\rangle^{\mu_{PQ}} \xrightarrow{mate_m} \\
\quad P_2 = \langle mate_o\langle\rangle^{\mu_{P_0Q_0}} \circ mate_o^{\perp}\langle\rangle^{\mu_{P_1}}\rangle^{\mu_{PQ}} \xrightarrow{mate_0} P_3 = \langle\langle\rangle^{\mu_{P_0Q_0P_1}}\rangle^{\mu_{PQ}} \\
\quad P_1 = \langle (mate_m|mate_o)\langle\rangle^{\mu_{P_0}} \circ mate_o^{\perp}\langle\rangle^{\mu_{P_1}} \circ mate_m^{\perp}\langle\rangle^{\mu_{Q_0}}\rangle^{\mu_{PQ}} \xrightarrow{mate_o} \\
\quad P_2' = \langle mate_m\langle\rangle^{\mu_{P_0P_1}} \circ \langle mate_m^{\perp}\langle\rangle^{\mu_{Q_0}}\rangle^{\mu_{PQ}} \xrightarrow{mate_m} P_3' = \langle\langle\rangle^{\mu_{P_0P_1Q_0}}\rangle^{\mu_{PQ}} \\
P = mate_n\langle (mate_m|mate_o)\langle\rangle^{\mu_{P_0}} \circ mate_o^{\perp}\langle\rangle^{\mu_{P_1}}\rangle^{\mu_P} \circ mate_n^{\perp}.\langle mate_m^{\perp}\langle\rangle^{\mu_{Q_0}}\rangle^{\mu_Q} \xrightarrow{mate_o} \\
\quad P_1'' = mate_n\langle mate_m\langle\rangle^{\mu'_{P_0P_1}}\rangle^{\mu_P} \circ mate_n^{\perp}.\langle mate_m^{\perp}\langle\rangle^{\mu_{Q_0}}\rangle^{\mu_Q} \xrightarrow{mate_n} \\
\quad P_2'' = \langle mate_m\langle\rangle^{\mu'_{P_0P_1}} \circ mate_m^{\perp}\langle\rangle^{\mu_{Q_0}}\rangle^{\mu_{PQ}} \xrightarrow{mate_m} P_3'' = \langle\langle\rangle^{\mu'_{P_0P_1Q_0}}\rangle^{\mu_{PQ}}
\end{array}$$

The main entries of the analysis are reported in Table 5, where we do not include the entries due to approximations, but not reflecting the dynamics. As before, we pair the inclusions of actions and of the corresponding co-actions, in order to emphasise which are the pairs of prefixes that lead to the prediction *of a possible communication. This motivates some redundancies in the entries. Also in this example, the CFA result is rather precise.*

Semantic Correctness Our analysis is semantically correct with respect to the given semantics, i.e. a valid estimate enjoys the following subject reduction property with respect to the semantics.

Theorem 1. ***(Subject Reduction)***
If $P \to Q$ and $\mathscr{I}, \mathscr{C}, \mathscr{R} \models^{\mu_{gp}\mu_p\mu} P$ then also $\mathscr{I}, \mathscr{C}, \mathscr{R} \models^{\mu_{gp}\mu_p\mu} Q$.

This result depends on the fact that analysis is invariant under the structural congruence, as stated below.

Lemma 1. ***(Invariance of Structural Congruence)*** *If $P \equiv Q$ and we have that $\mathscr{I}, \mathscr{C}, \mathscr{R} \models^{\mu_{gp}\mu_p\mu} P$ then also $\mathscr{I}, \mathscr{C}, \mathscr{R} \models^{\mu_{gp}\mu_p\mu} Q$.*

Moreover, it is possible to prove that there always exists a least estimate (see [4] for a similar statement and proof).

4 CFA for Spatial Structure Properties

Control Flow Analysis provides indeed a *safe over-approximation* of the *exact* behaviour of a system, that is, at least all the valid behaviours are captured. More precisely, all those events that the analysis does not consider as possible will *never* occur. On the other hand, the set of events deemed as possible may, or may not, occur in the actual dynamic evolution of the system. The 2CFA gains precision w.r.t the 0CFA presented in [4] and the incompatibility relation $\mathscr{R}$ increases this gain. In the next section, we will discuss on the contribution of the component $\mathscr{C}$.

We can exploit our analysis to check *spatial structure* properties, of the membranes included in the system under consideration. In particular, because of over-approximation, we can ask negative questions like whether: (i) a certain interaction capability *c never affects* the membrane labelled μ, i.e. it never occurs in the membrane process of the membrane labelled μ; (ii) the membrane labelled μ *never ends up* in the membrane labelled μ'.

Suppose we have all the possible labels μ of the possible membranes arising at run time. Then we can precisely define the above informally introduced properties. We first give the definition of the dynamic property, then the corresponding static property and, finally, we show that the static property implies the dynamic one. For each static property, we check for a particular content in the component $\mathscr{I}$.

Definition 1 (Dynamic: c never on μ)**.** *Given a process P including a membrane labelled μ, we say that the capability c never affects the membrane labelled μ if there not exists a derivative Q such that $P \to^* Q$, in which the capability c can affect the membrane labelled μ.*

Definition 2 (Static: c never on μ)**.** *Given a process P including a membrane labelled μ, we say that the capability c never appears on the membrane labelled μ if and only if there exists an estimate $(\mathscr{I}, \mathscr{R}, \mathscr{C})$ such that: $c \notin \mathscr{I}(\mu_{gp}, \mu_p, \mu)$ for each possible context $\mu_{gp}\mu_p$.*

Theorem 2. *Given a process P including a membrane labelled μ, then if c never appears on the membrane labelled μ, then the capability c never affects the membrane labelled μ.*

Definition 3 (Dynamic: μ' never inside μ)**.** *Given a process P including a membrane labelled μ and a membrane labelled μ', we say that the membrane μ' never ends up inside the membrane labelled μ if there not exists a derivative Q such that $P \to^* Q$, in which μ' occurs inside the membrane μ.*

Definition 4 (Static: μ' never inside μ)**.** *Given a process P including a membrane labelled μ, we say that μ' never appears inside the membrane labelled μ if and only if there exists an estimate $(\mathscr{I},\mathscr{R},\mathscr{C})$ such that: $\mu' \notin \mathscr{I}(\mu_{gp},\mu_p,\mu)$ for each possible context $\mu_{gp}\mu_p$.*

Theorem 3. *Given a process P including a membrane labelled μ and a membrane labelled μ', then if μ' never appears inside the membrane labelled μ, then the membrane μ' never ends up inside the membrane labelled μ.*

Back to our first running example, we can prove, for instance, that the capability $bud_o^{\perp}$ never affects the membrane labelled μ_P. This can be checked by looking in the CFA entries, for the content of $\mathscr{I}(*,*,\mu_P)$, that indeed does not include $bud_o^{\perp}$. Intuitively, this explains the fact that the bud_o synchronisation is not syntactically possible in the context μ_P, whose sub-membrane μ_{P_0} is affected by bud_o.

In our second running example, we can prove instead, for instance, that the membrane $\mu_{P_0Q_0}$ never ends up inside the membrane labelled μ_P, where $\mu_{P_0Q_0} = \mathbf{MI}_{\mathbf{mate}}(mate_m,\mu_{P_0},mate_n^{\perp},\mu_{Q_0},*,*,\mu_{PQ})$. Indeed, by inspecting the CFA results, we have that $\mu_{P_0Q_0} \notin \mathscr{I}(*,*,\mu_P)$. Intuitively, this corresponds to the fact that the $mate_m$ synchronisation is not syntactically possible in the context μ_P, because μ_{P_0} and μ_{Q_0} are not siblings, while it is in the context μ_{PQ}.

Similarly, we can mix ingredients and introduce new properties, e.g. one can ask whether two membranes labelled μ' and μ'', *never end up (occur) together* in the same membrane μ. On the static side, this amounts to checking whether there exists an estimate $(\mathscr{I},\mathscr{R},\mathscr{C})$ such that: for all possible context $\mu_{gp}\mu_p$, $\mu' \in \mathscr{I}(\mu_{gp},\mu_p,\mu) \wedge \mu'' \notin \mathscr{I}(\mu_{gp},\mu_p,\mu)$ or $\mu' \notin \mathscr{I}(\mu_{gp},\mu_p,\mu) \wedge \mu'' \in \mathscr{I}(\mu_{gp},\mu_p,\mu)$. Note that a single analysis can suffice for verifying all the above properties: only the values of interest tracked for testing change.

5 Discussion on Causal Information

Understanding the causal relationships between the actions performed by a process is a relevant issue for all process algebras used in Systems Biology. Although our CFA approximates the possible reachable configurations, we are able to extract some information on the causal relations among these configurations. To investigate these possibilities of our CFA, we follow [5] where different kinds of causal dependencies are described and classified, by applying our analysis to the same key examples.

The first kinds are called *structural causality* and *synchronisation causality* and are typical of all process algebras. Structural causality arises from the prefix structure of terms, as in

$$P = drip(\sigma).drip(\rho)\langle\rangle^{\mu_P} \xrightarrow{drip} \sigma\langle\rangle^{\mu_R} \circ drip(\rho)\langle\rangle^{\mu_P} \xrightarrow{drip} \sigma\langle\rangle^{\mu_R} \circ \rho\langle\rangle^{\mu_R'} \circ \langle\rangle^{\mu_P}$$

where the action on $drip(\rho)$ depends on the one on $drip(\sigma)$, since the second action is not reachable until the first has fired. Synchronisation causality arises when an action depends on a previous synchronisation as in:

$$P = drip(\sigma_1).mate_n.drip(\tau_1)\langle\rangle^{\mu_{P_0'}} \circ drip(\sigma_2).mate_n.drip(\tau_2)\langle\rangle^{\mu_{P_1'}} \xrightarrow{drip}^{2}$$
$$P' = \sigma_1\langle\rangle^{\mu_{R1}} \circ \sigma_2\langle\rangle^{\mu_{R2}} \circ mate_n.drip(\tau_1)\langle\rangle^{\mu_{P_0'}} \circ mate_n.drip(\tau_2)\langle\rangle^{\mu_{P_1'}} \xrightarrow{mate_n}$$
$$P'' = \sigma_1\langle\rangle^{\mu_{R1}} \circ \sigma_2\langle\rangle^{\mu_{R2}} \circ \langle\rangle^{\mu_{P_0'P_1'}}$$

where, the mate action is possible only when both $drip(\sigma_1)$ and $drip(\sigma_2)$ have been performed, and the following $drip(\tau_1)$ and $drip(\tau_2)$ depend on the previous mate synchronisation. Our CFA is not able to capture these kinds of dependencies, because of the $A()$ function definition, according to which $A(\sigma.\tau) = A(\sigma) \cup A(\tau)$. In other words, the CFA disperses the order between prefixes.

According to [5], when an action is performed on a membrane it impacts only on its continuation and not on the whole process on the membrane, e.g., in:

$$P = (mate_n|drip(\sigma))\langle\rangle^{\mu_P} \circ mate_n^{\perp}\langle\rangle^{\mu_Q} \xrightarrow{mate_n} drip(\sigma)\langle\rangle^{\mu_{PQ}} \xrightarrow{drip} \sigma\langle\rangle^{\mu_R} \circ \langle\rangle^{\mu_{PQ}}$$

$$P = (mate_n|drip(\sigma))\langle\rangle^{\mu_P} \circ mate_n^{\perp}\langle\rangle^{\mu_Q} \xrightarrow{drip} \sigma\langle\rangle^{\mu'_R} \circ (mate_n)\langle\rangle^{\mu_P} \circ mate_n^{\perp}\langle\rangle^{\mu_Q} \xrightarrow{mate_n} \sigma\langle\rangle^{\mu'_R} \circ \langle\rangle^{\mu_{PQ}}$$

the drip operation can be considered causally independent form the mate operation, because it can be executed regardless of the fact that the mate interaction has been performed.

Our analysis reflects this, because we have $\mathscr{C}(\mu_R) \ni (drip(\sigma), \mu_P, *, *, *)$ and also that $\mathscr{C}(\mu'_R) \ni (drip(\sigma), \mu_{PQ}, *, *, *)$.

When considering MBD actions and, in particular, the mate action, we have to do with another kind of causality called *environmental* in [5], due to the fact that the interaction possibilities of the child membranes are increased by the mate synchronisation.

Examples of this kind of causality can be observed in our running examples. In the first, for instance, the bud_o depends on the $mate_n$, as reflected by the CFA entries: $\mathscr{C}(\mu_{R2}) \ni (bud_o, \mu_P, bud_o^{\perp}, \mu_Q, *, *, \mu_{PQ})$, where $\mathscr{C}(\mu_{PQ}) \ni (mate_n, \mu_P, mate_n^{\perp}, \mu_Q, *, *, *)$.

In the second, we can observe that the synchronisation on $mate_m$ cannot be performed before a synchronisation on $mate_n$, as captured by the following CFA entries: $\mathscr{C}(\mu_{P_0Q_0}) \ni (mate_m, \mu_{P_0}, mate_m^{\perp}, \mu_{Q_0}, *, *, \mu_{PQ})$, and $\mathscr{C}(\mu_{P_0P_1Q_0}) \ni (mate_m, \mu_{P_0P_1}, mate_m^{\perp}, \mu_{Q_0}, *, *, \mu_{PQ})$, where $(mate_n, \mu_P, mate_n^{\perp}, \mu_Q, *, *, *)$ belongs to $\mathscr{C}(\mu_{PQ})$.

Finally, in [5], a casual dependency generated by Bud (and Drip) is discussed on the following example:

$$P = bud_n^{\perp}(drip(\sigma))\langle bud_n\langle\rangle^{\mu_{P0}}\rangle^{\mu_P} \rightarrow_{bud} drip(\sigma)\langle\langle\rangle^{\mu_{P0}}\rangle^{\mu_R} \circ \langle\rangle^{\mu_P} \rightarrow_{drip} \sigma\langle\rangle^{\mu_{R\sigma}} \circ \langle\langle\rangle^{\mu_{P0}}\rangle^{\mu_R} \circ \langle\rangle^{\mu_P}$$

The bud action generates a new membrane and the corresponding actions are caused by the new membrane, as captured by the CFA entries: $drip(\sigma) \in \mathscr{I}(*, *, \mu_R)$ and $\mathscr{C}(\mu_{R_\sigma}) \ni (drip(\sigma), \mu_R, *, *, *)$.

These considerations encourage us to further investigate and to formalise the static contribution of the CFA in establishing causal relationships.

6 The Analysis at Work: Viral Infection

We illustrate our approach by applying it to the abstract description of the infection cycle of the Semliki Forest Virus, shown in Figure 1, as specified in [6]. The Semliki Forest Virus is one of the so-called "enveloped viruses". We focus just on the first stage of the cycle and we report the analysis as given in [4]. The virus, specified in Table 8, consists of a capsid containing the viral RNA (the nucleocapsid). The nucleocapsid is surrounded by a membrane, similar to the cellular one, but enriched with a special protein. The virus is brought into the cell by phagocytosis, thus wrapped by an additional membrane layer. An endosome compartment is merged with the wrapped-up virus. At this point, the virus uses its special membrane protein to trigger the exocytosis process that leads the naked nucleocapsid into the cytosol, ready to damage it. By summarising, if the *cell* gets close to a *virus*, then it evolves into an infected cell. The complete evolution of the viral infection is reported in Table 8, while the main analysis entries are in Table 9. The specification includes the PEP version of Brane calculus, whose syntax and reduction semantics is reported in Table 6. This further set of PEP actions (Ξ_{PEP}) are inspired by endocytosis and exocytosis processes. The first indicates the process of incorporating external material into a cell, by engulfing it with the cell membrane, while the second one indicates the reverse process. Endocytosis is rendered by two more basic operations: *phagocytosis* (denoted by *phago*), that consists in engulfing just one external membrane, and *pinocytosis* (denoted by *pino*), consists in engulfing zero external membranes; *exocytosis* is instead denoted by *exo*. The CFA for the calculus can be straightforwardly extended

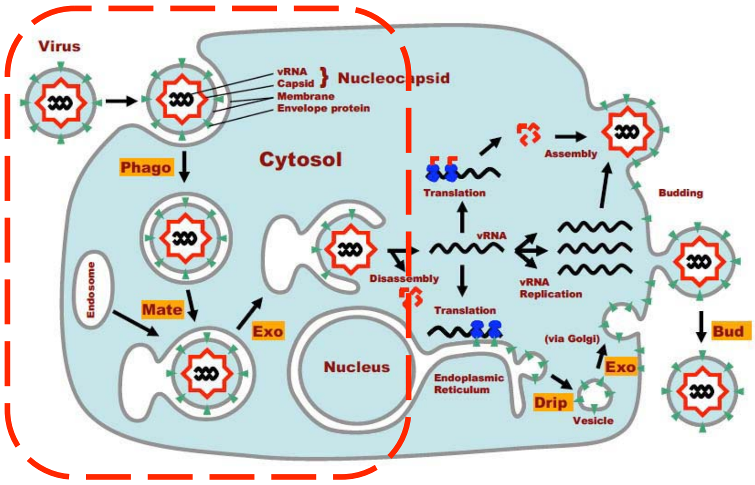


Figure 1: *Viral Infection (highlighted part) and Reproduction. [Adapted from [6] and [1]]*

to deal with the Phago/Exo/Pino (PEP) actions, as shown in Table 7.

	$a ::= \quad phago_n \mid phago_n^{\perp}(\rho) \mid exo_n \mid exo_n^{\perp} \mid pino(\rho) \qquad \Xi_{PEP}$
$(Phago)$	$phago_n.\sigma \mid \sigma_0\langle P\rangle^{\mu_P} \circ phago_n^{\perp}(\rho).\tau \mid \tau_0\langle Q\rangle^{\mu_Q} \to \tau \mid \tau_0\langle \rho\langle \sigma \mid \sigma_0.\langle P\rangle^{\mu_P}\rangle^{\mu_R} \circ Q\rangle^{\mu_Q}$
	where $\mu_R = \mathbf{MI_{phago}}(phago_n, \mu_P, phago_n^{\perp}(\rho), \mu_Q, \mu_{gp}, \mu_p, \mu)$
	and μ identifies the closest membrane surrounding μ_P in the context $\mu_{gp}\mu_p$
(Exo)	$exo_n^{\perp}.\tau \mid \tau_0\langle exo_n.\sigma \mid \sigma_0\langle P\rangle^{\mu_P} \circ Q\rangle^{\mu_Q} \to P \circ \sigma \mid \sigma_0 \mid \tau \mid \tau_0\langle Q\rangle^{\mu_Q}$
$(Pino)$	$pino(\rho).\sigma \mid \sigma_0\langle P\rangle^{\mu_P} \to \sigma \mid \sigma_0\langle \rho\langle\rangle^{\mu_R} \circ P\rangle^{\mu_P}$
	where $\mu_R = \mathbf{MI_{pino}}(pino(\rho), \mu_P, \mu_{gp}, \mu_p, \mu)$
	and μ identifies the closest membrane surrounding μ_P in the context $\mu_{gp}\mu_p$

Table 6: Syntax and Reduction Rules for PEP Actions.

Roughly, the analysis results allow us to predict the effects of the infection. Indeed, the inclusion $\mu_{nucap} \in \mathscr{I}(*,*,\mu_{memb})$ reflects the fact that, at the end of the shown computation, *nucap* is inside *membrane* together with *cytosol'* that is equivalent to *cytosol*, apart from the label $\mu_{ph\text{-}endo}$ that decorates the enclosed membrane *endosome*. Furthermore, we can check our properties in this systems. As far as the spatial structure properties, we can prove here, e.g., that (i) the capability $exo^{\perp}$ never affects the membrane labelled μ_{ph} (as $exo^{\perp} \notin \mathscr{I}(*,\mu_{memb},\mu_{ph})$); and that (ii) the membrane μ_{virus} never ends up inside the membrane labelled μ_{endo} (as $\mu_{virus} \notin \mathscr{I}(*,\mu_{memb},\mu_{endo})$). Furthermore, we can observe that the CFA captures the dependency of the synchronisation on *mate* on the synchronisation on *phago*, since we have that $(mate, \mu_{ph}, mate^{\perp}, \mu_{endo}, *, *, \mu_{memb}) \in \mathscr{C}(\mu_{ph\text{-}endo})$, and μ_{ph} is such that we have that $(phago, \mu_{virus}, phago^{\perp}, \mu_{memb}, *, *, *) \in \mathscr{C}(\mu_{ph})$.

$(Phago)$	$phago_n \in \mathscr{I}(\mu_p, \mu, \mu_P) \wedge phago_n^{\perp}(\rho) \in \mathscr{I}(\mu_p, \mu, \mu_Q) \wedge \mu_P, \mu_Q \in \mathscr{I}(\mu_{gp}, \mu_p, \mu) \wedge ((\mu_p, \mu, \mu_P), (\mu_p, \mu, \mu_Q)) \notin \mathscr{R}$
	$\Rightarrow A(\rho) \subseteq \mathscr{I}(\mu, \mu_Q, \mu_R) \wedge \mu_R \in \mathscr{I}(\mu_p, \mu, \mu_Q) \wedge \mu_P \in \mathscr{I}(\mu, \mu_Q, \mu_R)$
	where $\mu_R = \mathbf{MI_{phago}}(phago_n, \mu_P, phago_n^{\perp}(\rho), \mu_Q, \mu_{gp}, \mu_p, \mu)$
(Exo)	$exo_n \in \mathscr{I}(\mu, \mu_Q, \mu_P) \wedge exo_n^{\perp} \in \mathscr{I}(\mu_p, \mu, \mu_Q) \wedge \mu_P \in \mathscr{I}(\mu_p, \mu, \mu_Q) \wedge \mu_Q \in \mathscr{I}(\mu_{gp}, \mu_p, \mu)$
	$\Rightarrow A(\sigma), A(\sigma_0) \subseteq \mathscr{I}(\mu_p, \mu, \mu_Q) \wedge \mathscr{I}(\mu, \mu_Q, \mu_P) \subseteq \mathscr{I}(\mu_{gp}, \mu_p, \mu)$
$(Pino)$	$pino(\rho) \in \mathscr{I}(\mu_p, \mu, \mu_P) \wedge \mu_P \in \mathscr{I}(\mu_{gp}, \mu_p, \mu)$
	$\Rightarrow A(\rho) \subseteq \mathscr{I}(\mu, \mu_P, \mu_R) \wedge \mu_R \in \mathscr{I}(\mu_p, \mu, \mu_P)$
	where $\mu_R = \mathbf{MI_{pino}}(pino(\rho), \mu_P, \mu_{gp}, \mu_p, \mu)$

Table 7: Closure Rules for PEP Actions

$virus$	$\overset{def}{=}$	$phago.exo\langle nucap\rangle^{\mu_{virus}}$	$membrane$	$\overset{def}{=}$	$!phago^{\perp}(mate)\vert !exo^{\perp}$
$nucap$	$\overset{def}{=}$	$!bud\vert X\langle vRNA\rangle^{\mu_{nucap}}$	$cytosol$	$\overset{def}{=}$	$endosome \circ Z$
$cell$	$\overset{def}{=}$	$membrane\langle cytosol\rangle^{\mu_{memb}}$	$endosome$	$\overset{def}{=}$	$!mate^{\perp}\vert !exo^{\perp}\langle\rangle^{\mu_{endo}}$

$$virus \circ cell$$

$$\equiv (phago.exo)\langle nucap\rangle^{\mu_{virus}} \circ (!phago^{\perp}(mate)|!exo^{\perp})\langle cytosol\rangle^{\mu_{memb}} \xrightarrow{phago}$$
$$(!phago^{\perp}(mate)|!exo^{\perp})\langle mate\langle exo\langle nucap\rangle^{\mu_{virus}}\rangle^{\mu_{ph}} \circ (!mate^{\perp}|!exo^{\perp})\langle\rangle^{\mu_{endo}} \circ Z\rangle^{\mu_{memb}}$$
$$\xrightarrow{mate} (!phago^{\perp}(mate)|!exo^{\perp})\langle(!mate^{\perp}|!exo^{\perp})\langle exo\langle nucap\rangle^{\mu_{virus}}\rangle^{\mu_{ph\text{-}endo}} \circ Z\rangle^{\mu_{memb}} \xrightarrow{exo}$$
$$(!phago^{\perp}(mate)|!exo^{\perp})\langle(!mate^{\perp}|!exo^{\perp})\langle\rangle^{\mu_{ph\text{-}endo}} \circ nucap \circ Z\rangle^{\mu_{memb}} \equiv$$
$$membrane\langle nucap \circ cytosol'\rangle^{\mu_{memb}}$$

Table 8: Viral Infection System and its Evolution

$$\mu_{nucap} \in \mathscr{I}(*, *, \mu_{virus}), \mu_{endo} \in \mathscr{I}(*, *, \mu_{memb}), \mu_{virus}, \mu_{memb} \in \mathscr{I}(*, *, *)$$
$$phago, exo \in \mathscr{I}(*, *, \mu_{virus}), phago^{\perp}(mate), exo^{\perp} \in \mathscr{I}(*, *, \mu_{memb})$$
$$mate \in \mathscr{I}(*, \mu_{memb}, \mu_{ph}), \mu_{ph} \in \mathscr{I}(*, *, \mu_{memb}), \mu_{virus} \in \mathscr{I}(*, \mu_{memb}, \mu_{ph}),$$
$$mate^{\perp}, exo^{\perp} \in \mathscr{I}(*, \mu_{memb}, \mu_{endo})$$
$$\mu_{nucap} \in \mathscr{I}(\mu_{memb}, \mu_{ph}, \mu_{virus}),$$
$$\mu_{ph\text{-}endo} \in \mathscr{I}(*, *, \mu_{memb}), \mu_{virus} \in \mathscr{I}(*, \mu_{memb}, \mu_{ph\text{-}endo}),$$
$$mate^{\perp}, exo^{\perp} \in \mathscr{I}(*, \mu_{memb}, \mu_{ph\text{-}endo})$$
$$\mu_{nucap} \in \mathscr{I}(\mu_{memb}, \mu_{ph\text{-}endo}, \mu_{virus}),$$
$$\mu_{nucap} \in \mathscr{I}(*, *, \mu_{memb}),$$
$$\mu_{ph-endo} \in \mathscr{I}(*, *, \mu_{memb})$$
$$(phago, \mu_{virus}, phago^{\perp}, \mu_{memb}, *, *, *) \in \mathscr{C}(\mu_{ph})$$
$$(mate, \mu_{ph}, mate^{\perp}, \mu_{endo}, *, *, \mu_{memb}) \in \mathscr{C}(\mu_{ph\text{-}endo})$$

Table 9: Viral Infection Analysis Results

7 Conclusions

We have presented a refinement of the CFA for the Brane calculi [4], based on contextual and causal information. The CFA provides us with a verification framework for properties of biological systems modelled in Brane, such as properties on the spatial structure of processes, in terms of membrane hierarchy. We plan to formalise new properties like the ones introduced here.

We have found that the CFA is able to capture some kinds of causal dependencies [5] arising in the MBD version of Brane Calculi. As future work, we would like to investigate thoroughly and formally the static contribution of the CFA in establishing causal relationships between the Brane interactions.

Acknowledgments. We wish to thank Francesca Levi for our discussion on a draft of our paper and our anonymous referees for their useful comments.

References

[1] B. Alberts, D. Bray, J. Lewis, M. Raff, K. Roberts, and J.D. Watson. "Molecular Biology of the Cell". Third Edition, Garland.

[2] R. Barbuti, G. Caravagna, A. Maggiolo-Schettini, P. Milazzo, and G. Pardini. *The calculus of looping sequences*. In *Proc. of SFM'08*, LNCS 5016 (2008), 387–423.

[3] C. Bodei. *A Control Flow Analysis for Beta-binders with and without Static Compartments*. Theoretical Computer Science 410(33-34): 3110-3127, 2009, Elsevier.

[4] C. Bodei, A. Bracciali, and D. Chiarugi. *Control Flow Analysis for Brane Calculi*. In Proc. of MeCBIC'08, ENTCS 227, pp. 59-75, 2009.

[5] N. Busi. *Towards a Causal Semantics for Brane Calculi*. In *What is it About Government that Americans Dislike*, 1945–1965, University Press, 2007.

[6] L. Cardelli. *Brane calculi - interactions of biological membranes*. In *Proc. of Computational Methods in Systems Biology (CMSB'04)*, LNCS 3082, 257–280, 2005.

[7] R. Gori and F. Levi. A New Occurrence Counting Analysis for BioAmbients. *Proc. of APLAS'05*, LNCS 3780:381-400, 2005.

[8] R. Gori and F. Levi. An Analysis for Proving Temporal Properties of Biological Systems. *Proc. of APLAS'06*, LNCS 4279:234–252, 2006.

[9] R. Gori and F. Levi. *Abstract interpretation based verification of temporal properties for BioAmbients*, Inf. Comput.(8): 869-921, 2010.

[10] V. Danos and C. Laneve. *Graphs for core molecular biology*. In *Proc. of CMSB'03*, LNCS 2602 (2003), 34–46, Springer.

[11] C. Laneve and F. Tarissan. *A Simple Calculus for Proteins and Cells*, In ENTCS 171 (2), pp. 139-154, 2007.

[12] H. R. Nielson and F. Nielson. Flow logic: A multi-paradigmatic approach to static analysis. In *The Essence of Computation*, LNCS 2566, pp. 223–244. Springer, 2002.

[13] F. Nielson, H. Riis Nielson, C. Priami, and D. Schuch da Rosa. *Control Flow Analysis for BioAmbients*. ENTCS 180(3), 65–79, 2007, Elsevier.

[14] G. Paun. *Computing with membranes (P systems): A variant*. Int. J. Found. Comput. Sci., 11(1) (2000).

[15] H. Pilegaard, F. Nielson, H. Riis Nielson. *Context Dependent Analysis of BioAmbients*. In *Proc. of Emerging Aspects of Abstract Interpretation'06*, 2006.

[16] H. Pilegaard, F. Nielson, H. Riis Nielson. *Pathway analysis for BioAmbients*. In *The Journal of Logic and Algebraic Programming*, 2008.

[17] C. Priami and P. Quaglia. *Beta binders for biological interactions*. In *Proc. of CMSB'04*, LNCS 3082 (2005).

[18] A. Regev, E.M. Panina, W. Silverman, L. Cardelli, and E.Y. Shapiro. BioAmbients: An abstraction for biological compartments. *Theoretical Computer Science* 325(1): 141-167. 2004, Elsevier.

A Proofs

This appendix restates the lemmata and theorems presented earlier in the paper and gives the proofs of their correctness. To establish the semantic correctness, the following auxiliary results are needed.

Proposition 1. *If $\mathscr{I},\mathscr{C},\mathscr{R} \models^{\mu_p\mu\mu_1} P$ and $\mathscr{I}(\mu_p,\mu,\mu_1) \subseteq \mathscr{I}(\mu_p,\mu,\mu_2)$, then $\mathscr{I},\mathscr{C},\mathscr{R} \models^{\mu_p\mu\mu_2} P$.*

Proof. By structural induction on P. We show just one case.
Case $P = \sigma\langle P'\rangle^{\mu_s}$. We have that $\mathscr{I},\mathscr{C},\mathscr{R} \models^{\mu_p\mu\mu_1} P$ is equivalent to $\mu_s \in \mathscr{I}(\mu_p,\mu,\mu_1) \ \wedge\ A(\sigma) \subseteq \mathscr{I}(\mu,\mu_1,\mu_s) \ \wedge\ \mathscr{I},\mathscr{C},\mathscr{R} \models^{\mu\mu_1\mu_s} P'$. Now, $\mu_s \in \mathscr{I}(\mu_p,\mu,\mu_1)$ and $\mathscr{I}(\mu_p,\mu,\mu_1) \subseteq \mathscr{I}(\mu_p,\mu,\mu_2)$ and $A(\sigma) \subseteq \mathscr{I}(\mu,\mu_1,\mu_s)$ imply $\mu_s \in \mathscr{I}(\mu_p,\mu,\mu_2)$ and $A(\sigma) \subseteq \mathscr{I}(\mu,\mu_2,\mu_s)$. Therefore, by induction hypothesis, we have that $\mathscr{I},\mathscr{C},\mathscr{R} \models^{\mu_p\mu\mu_2} P$. □

Proposition 2. *If $\sigma \equiv \tau$ then $A(\sigma) = A(\tau)$.*

Proof. The proof amounts to a straightforward inspection of each of the clauses defining the structural congruence clauses relative to membranes. We only show two cases, the others are similar.
Case $\sigma_0|\sigma_1 \equiv \sigma_1|\sigma_0$. We have that $A(\sigma_0|\sigma_1) = A(\sigma_0) \cup A(\sigma_1) = A(\sigma_1|\sigma_0)$.
Case $\sigma \equiv \tau \Rightarrow \sigma|\rho \equiv \tau|\rho$. We have that $A(\sigma|\rho) = A(\sigma) \cup A(\rho)$. Now, since $\sigma \equiv \tau$, we have that $A(\sigma) = A(\tau)$ and therefore $A(\sigma|\rho) = A(\tau) \cup A(\rho)$, from which the required $A(\tau|\rho)$. □

Lemma 4.1 (Invariance of Structural Congruence) *If $P \equiv Q$ and we have that $\mathscr{I},\mathscr{C},\mathscr{R} \models^{\mu_{gp}\mu_p\mu} P$ then also $\mathscr{I},\mathscr{C},\mathscr{R} \models^{\mu_{gp}\mu_p\mu} Q$.*

Proof. The proof amounts to a straightforward inspection of each of the clauses defining the structural congruence clauses. We only show two cases, the others are similar.
Case $P_0 \circ P_1 \equiv P_1 \circ P_0$. We have that $\mathscr{I},\mathscr{C},\mathscr{R} \models^{\mu_{gp}\mu_p\mu} P_0 \circ P_1$ is equivalent to $\mathscr{I},\mathscr{C},\mathscr{R} \models^{\mu_{gp}\mu_p\mu} P_0 \wedge \mathscr{I},\mathscr{C},\mathscr{R} \models^{\mu_{gp}\mu_p\mu} P_1$, that is equivalent to $\mathscr{I},\mathscr{C},\mathscr{R} \models^{\mu_{gp}\mu_p\mu} P_1 \wedge \mathscr{I},\mathscr{C},\mathscr{R} \models^{\mu_{gp}\mu_p\mu} P_0$ and therefore to $\mathscr{I},\mathscr{C},\mathscr{R} \models^{\mu_{gp}\mu_p\mu} P_1 \circ P_0$.
Case $P \equiv Q \ \wedge\ \sigma \equiv \tau \Rightarrow \sigma\langle P\rangle^{\mu_s} \equiv \tau\langle Q\rangle^{\mu_s}$. We have that $\mathscr{I},\mathscr{C},\mathscr{R} \models^{\mu_{gp}\mu_p\mu} \sigma\langle P\rangle^{\mu_s}$ is equivalent to $\mu_s \in \mathscr{I}(\mu_{gp},\mu_p,\mu) \ \wedge\ A(\sigma) \subseteq \mathscr{I}(\mu_p,\mu,\mu_s) \ \wedge\ \mathscr{I},\mathscr{C},\mathscr{R} \models^{\mu_p,\mu,\mu_s} P$. By Proposition 2, $A(\tau) \subseteq \mathscr{I}(\mu_p,\mu,\mu_s)$, and by induction hypothesis, we have that $\mathscr{I},\mathscr{C},\mathscr{R} \models^{\mu_p,\mu,\mu_s} Q$. As a consequence, we can conclude that $\mathscr{I},\mathscr{C},\mathscr{R} \models^{\mu_{gp}\mu_p\mu} \tau\langle Q\rangle^{\mu_s}$. □

Theorem 4.2 (Subject Reduction)
If $P \to Q$ and $\mathscr{I},\mathscr{C},\mathscr{R} \models^{\mu_{gp}\mu_p\mu} P$ then also $\mathscr{I},\mathscr{C},\mathscr{R} \models^{\mu_{gp}\mu_p\mu} Q$.

Proof. The proof is by induction on $P \to Q$. The proofs for the rules (Par) and $(Brane)$ are straightforward, using the induction hypothesis and the clauses in Table 3. The proof for the $(Struct)$ uses instead the induction hypothesis and Lemma 4.1. The proofs for the basic actions in the lower part of Table 2 are straightforward, using the clauses in Table 3.
Case (Par). Let P be $P_0 \circ P_1$ and Q be $P_0' \circ P_1$, with $P_0 \to P_0'$. We have to prove that $\mathscr{I},\mathscr{C},\mathscr{R} \models^{\mu_{gp}\mu_p\mu} Q$. Now $\mathscr{I},\mathscr{C},\mathscr{R} \models^{\mu_{gp}\mu_p\mu} P$ is equivalent to $\mathscr{I},\mathscr{C},\mathscr{R} \models^{\mu_{gp}\mu_p\mu} P_0 \ \wedge\ \mathscr{I},\mathscr{C},\mathscr{R} \models^{\mu_{gp}\mu_p\mu} P_1$. By induction hypothesis, we have that $\mathscr{I},\mathscr{C},\mathscr{R} \models^{\mu_{gp}\mu_p\mu} P_0'$, and from $\mathscr{I},\mathscr{C},\mathscr{R} \models^{\mu_{gp}\mu_p\mu} P_0' \ \wedge\ \mathscr{I},\mathscr{C},\mathscr{R} \models^{\mu_{gp}\mu_p\mu} P_1$ we obtain the required $\mathscr{I},\mathscr{C},\mathscr{R} \models^{\mu_{gp}\mu_p\mu} Q$.
Case (Brane). Let P be $\sigma\langle P_0\rangle^{\mu_s}$ and Q be $\sigma\langle P_0'\rangle^{\mu_s}$. We have to prove that $\mathscr{I},\mathscr{C},\mathscr{R} \models^{\mu_{gp}\mu_p\mu} \sigma\langle P_0'\rangle^{\mu_s}$. Now $\mathscr{I},\mathscr{C},\mathscr{R} \models^{\mu_{gp}\mu_p\mu} P$ is equivalent to have that $\mu_s \in \mathscr{I}(\mu_{gp},\mu_p,\mu) \ \wedge\ A(\sigma) \subseteq \mathscr{I}(\mu_p,\mu,\mu_s) \ \wedge\ \mathscr{I},\mathscr{C},\mathscr{R} \models^{\mu_p,\mu,\mu_s} P_0$. By induction hypothesis, we have that $\mathscr{I},\mathscr{C},\mathscr{R} \models^{\mu_p,\mu,\mu_s} P_0'$. We can therefore conclude that $\mathscr{I},\mathscr{C},\mathscr{R} \models^{\mu_{gp}\mu_p\mu} Q$.

Case (Struct). Let $P \equiv P_0$, with $P_0 \to P_1$ such that $P_1 \equiv Q$. By Lemma A, we have that $\mathscr{I},\mathscr{C},\mathscr{R} \models^{\mu_{gp}\mu_p\mu} P_0$, by induction hypothesis $\mathscr{I},\mathscr{C},\mathscr{R} \models^{\mu_{gp}\mu_p\mu} P_1$ and, again by Lemma A, $\mathscr{I},\mathscr{C},\mathscr{R} \models^{\mu_{gp}\mu_p\mu} Q$.
Case (Mate). Let P be $mate_n.\sigma|\sigma_0\langle P_0\rangle^{\mu_0} \circ mate_n^{\perp}.\tau|\tau_0\langle P_1\rangle^{\mu_1}$ and Q be $\sigma|\sigma_0|\tau|\tau_0\langle P_0 \circ P_1\rangle^{\mu_{01}}$. Then, $\mathscr{I},\mathscr{C},\mathscr{R} \models^{\mu_{gp}\mu_p\mu} P$ amounts to $\mathscr{I},\mathscr{C},\mathscr{R} \models^{\mu_{gp}\mu_p\mu} mate_n.\sigma|\sigma_0\langle P_0\rangle^{\mu_0}$ and $\mathscr{I},\mathscr{C},\mathscr{R} \models^{\mu_{gp}\mu_p\mu} mate_n^{\perp}.\tau|\tau_0\langle P_1\rangle^{\mu_1}$ and, in turn, to $\mu_0,\mu_1 \in \mathscr{I}(\mu_{gp},\mu_p,\mu)$, $\{mate_n\} \cup A(\sigma) \cup A(\sigma_0) \subseteq \mathscr{I}(\mu_p,\mu,\mu_0)$, $\{mate_n^{\perp}\} \cup A(\tau) \cup A(\tau_0) \subseteq \mathscr{I}(\mu_p,\mu,\mu_1)$, and $\mathscr{I},\mathscr{C},\mathscr{R} \models^{\mu_p\mu\mu_0} P_0$ and $\mathscr{I},\mathscr{C},\mathscr{R} \models^{\mu_p\mu\mu_1} P_1$. Note that, $((\mu_p,\mu,\mu_0),(\mu_p,\mu,\mu_1))$ does not belong to $\mathscr{R}$. Because of the closure conditions, from the above, we have, amongst the several implied conditions, that $\exists\mu_{01} = \mathbf{MI_{mate}}(mate_n,\mu_0,mate_n^{\perp},\mu_1,\mu_{gp},\mu_p,\mu)$ such that $\mu_{01} \in \mathscr{I}(\mu_{gp},\mu_p,\mu) \wedge \mathscr{I}(\mu_p,\mu,\mu_0) \cup \mathscr{I}(\mu_p,\mu,\mu_1) \subseteq \mathscr{I}(\mu_p,\mu,\mu_{01})$. From $\mathscr{I}(\mu_p,\mu,\mu_i) \subseteq \mathscr{I}(\mu_p,\mu,\mu_{01})$ for $i = 0,1$, we have that $A(\sigma) \cup A(\sigma_0) \subseteq \mathscr{I}(\mu_p,\mu,\mu_{01})$ and $A(\tau) \cup A(\tau_0) \subseteq (\mu_p,\mu,\mu_{01})$ and, by Proposition 1, we have that $\mathscr{I},\mathscr{C},\mathscr{R} \models^{\mu_p\mu\mu_{01}} P_0$ and $\mathscr{I},\mathscr{C},\mathscr{R} \models^{\mu_p\mu\mu_{01}} P_1$, and hence the required $\mathscr{I},\mathscr{C},\mathscr{R} \models^{\mu_p\mu\mu_{01}} Q$.
Case (Bud). Let P be $bud_n^{\perp}(\rho).\tau|\tau_0\langle bud_n.\sigma|\sigma_0\langle P_0\rangle^{\mu_0} \circ P_1\rangle^{\mu_1}$ and Q be the process $\rho\langle\sigma|\sigma_0\langle P_0\rangle^{\mu_0}\rangle^{\mu_R} \circ \tau|\tau_0\langle P_1\rangle^{\mu_1}$. Now, $\mathscr{I},\mathscr{C},\mathscr{R} \models^{\mu_{gp}\mu_p\mu} P$ is equivalent to $\mu_1 \in \mathscr{I}(\mu_{gp},\mu_p,\mu)$, $\{bud_n^{\perp}(\rho)\} \cup A(\tau) \cup A(\tau_0) \subseteq \mathscr{I}(\mu_p,\mu,\mu_1)$, and, moreover, $\mathscr{I},\mathscr{C},\mathscr{R} \models^{\mu_p\mu\mu_1} bud_n.\sigma|\sigma_0\langle P_0\rangle^{\mu_0}$ and $\mathscr{I},\mathscr{C},\mathscr{R} \models^{\mu_p\mu\mu_1} P_1$, from which we have that $\mu_0 \in \mathscr{I}(\mu_p,\mu,\mu_1)$, $\{bud_n\} \cup A(\sigma) \cup A(\sigma_0) \subseteq \mathscr{I}(\mu,\mu_1,\mu_0)$, and $\mathscr{I},\mathscr{C},\mathscr{R} \models^{\mu\mu_1\mu_0} P_0$. Because of the closure conditions, from above, we have that $\exists\mu_R = \mathbf{MI_{bud}}(bud_n,\mu_0,bud_n^{\perp},\mu_1,\mu_{gp},\mu_p,\mu)$ such that $\mu_R \in \mathscr{I}(\mu_{gp},\mu_p,\mu)$, $A(\rho) \subseteq \mathscr{I}(\mu_p,\mu,\mu_R)$, $\mu_0 \in \mathscr{I}(\mu_p,\mu,\mu_R)$, and ($\mathscr{I}(\mu,\mu_1,\mu_0) \subseteq \mathscr{I}(\mu,\mu_R,\mu_0)$ and $\mathscr{I}(\mu_1,\mu_0) \subseteq \mathscr{I}(\mu_R,\mu_0)$ (cond 2)). We have that $\mathscr{I},\mathscr{C},\mathscr{R} \models^{\mu_{gp}\mu_p\mu} Q$ is equivalent to have that $\mathscr{I},\mathscr{C},\mathscr{R} \models^{\mu_{gp}\mu_p\mu} \rho\langle\sigma|\sigma_0\langle P_0\rangle^{\mu_0}\rangle^{\mu_R}$ (1) and that $\mathscr{I},\mathscr{C},\mathscr{R} \models^{\mu_{gp}\mu_p\mu} \tau|\tau_0\langle P_1\rangle^{\mu_1}$ (2). For (1), we have to prove that $\mu_R \in \mathscr{I}(\mu_{gp},\mu_p,\mu)$, $A(\rho) \subseteq \mathscr{I}(\mu_p,\mu,\mu_R)$ and $\mathscr{I},\mathscr{C},\mathscr{R} \models^{\mu_p\mu\mu_R} \sigma|\sigma_0\langle P_0\rangle^{\mu_0}$, that is equivalent to $\mu_0 \in \mathscr{I}(\mu_p,\mu,\mu_R)$, $A(\sigma) \cup A(\sigma_0) \subseteq \mathscr{I}(\mu,\mu_R,\mu_0)$ and $\mathscr{I},\mathscr{C},\mathscr{R} \models^{\mu\mu_R\mu_0} P_0$. From the hypotheses, we have that $\mu_0 \in \mathscr{I}(\mu_p,\mu,\mu_R)$. Since $A(\sigma) \cup A(\sigma_0) \subseteq \mathscr{I}(\mu,\mu_1,\mu_0)$ and (cond 2) we have $A(\sigma) \cup A(\sigma_0) \subseteq \mathscr{I}(\mu,\mu_R)$. From $\mathscr{I},\mathscr{C},\mathscr{R} \models^{\mu\mu_1\mu_0} P_0$, because of (cond 2) and Proposition 1, we have that $\mathscr{I},\mathscr{C},\mathscr{R} \models^{\mu\mu_R\mu_0} P_0$. For (2), we have to prove that $\mu_1 \in \mathscr{I}(\mu_{gp},\mu_p,\mu)$, $A(\tau) \cup A(\tau_0) \subseteq \mathscr{I}(\mu_p,\mu,\mu_1)$ and $\mathscr{I},\mathscr{C},\mathscr{R} \models^{\mu_p\mu\mu_1} P_1$. All these conditions are satisfied (see above). Therefore, we obtain the required $\mathscr{I},\mathscr{C},\mathscr{R} \models^{\mu_{gp}\mu_p\mu} Q$.
Case (Drip). Let P be $drip(\rho).\sigma|\sigma_0\langle P_0\rangle^{\mu_0}$ and Q be $\rho\langle\rangle^{\mu_R} \circ \sigma|\sigma_0\langle P_0\rangle^{\mu_0}$. We have that $\mathscr{I},\mathscr{C},\mathscr{R} \models^{\mu_{gp}\mu_p\mu} P$ is equivalent to $\mu_0 \in \mathscr{I}(\mu_{gp},\mu_p,\mu)$, $\{drip(\rho)\} \cup A(\sigma) \cup A(\sigma_0) \subseteq \mathscr{I}(\mu_p,\mu,\mu_0)$, and $\mathscr{I},\mathscr{C},\mathscr{R} \models^{\mu_p\mu\mu_0} P_0$. Because of the closure conditions, from the above, $\exists\mu_R = \mathbf{MI_{drip}}(drip(\rho),\mu_0,\mu_{gp},\mu_p,\mu)$ such that $\mu_R \in \mathscr{I}(\mu_{gp},\mu_p,\mu)$, $A(\rho) \subseteq \mathscr{I}(\mu_p,\mu,\mu_R)$. We have that $\mathscr{I},\mathscr{C},\mathscr{R} \models^{\mu_{gp}\mu_p\mu} Q$ is equivalent to both $\mathscr{I},\mathscr{C},\mathscr{R} \models^{\mu_{gp}\mu_p\mu} \rho\langle\rangle^{\mu_R}$ and $\mathscr{I},\mathscr{C},\mathscr{R} \models^{\mu_{gp}\mu_p\mu} \sigma|\sigma_0\langle P_0\rangle^{\mu_0}$. The first condition is verified, because $\mu_R \in \mathscr{I}(\mu_{gp},\mu_p,\mu)$ and $A(\rho) \subseteq \mathscr{I}(\mu_p,\mu,\mu_R)$. The second amounts to $\mu_0 \in \mathscr{I}(\mu_{gp},\mu_p,\mu)$ and $A(\sigma) \cup A(\sigma_0) \subseteq \mathscr{I}(\mu_p,\mu,\mu_0)$ and it is satisfied as well. We therefore obtain the required $\mathscr{I},\mathscr{C},\mathscr{R} \models^{\mu_{gp}\mu_p\mu} Q$.
$\square$

Theorem 5.2 *Given a process P including a membrane labelled μ, then if c never appears on the membrane labelled μ, then the capability c never affects the membrane labelled μ.*

Proof. First of all, we observe that if c affects μ in P, then we have a contradiction, since it implies that $c \notin \mathscr{I}(\mu_{gp},\mu_p,\mu)$ for some context $\mu_{gp}\mu_p$. We now show that there exist no Q, Q' such that $P \to^* Q \to Q'$ such that c does not affect μ in Q, while it does in Q'. The only case in which this can happen is when a (*Bud*) or a (*Drip*) is performed with parameter ρ including c. Indeed, the firing of such an action lets arise a new membrane affected by the corresponding parameter. We focus on the second one. Suppose we have in Q a sub-process $drip(\rho).\sigma|\sigma_0\langle Q_0\rangle^{\mu}$ and that c occurs in ρ. This amounts to have that c can affect μ in Q'. By theorem 1, we have that $(\mathscr{I},\mathscr{R},\mathscr{C})$ is an estimate also for Q. Nevertheless this implies that $c \in \mathscr{I}(\mu_{gp},\mu_p,\mu)$, thus leading to a contradiction. $\square$

Synchronization of P Systems with Simplex Channels

(Work in Progress)

Florentin Ipate
Department of Computer Science, University of Piteşti,
Târgul din Vale 1, Piteşti, Romania
florentin.ipate@ifsoft.ro

Radu Nicolescu
Department of Computer Science, University of Auckland,
Private Bag 92019, Auckland, New Zealand
r.nicolescu@auckland.ac.nz

Ionuţ Mihai Niculescu
Department of Computer Science, University of Piteşti,
Târgul din Vale 1, Piteşti, Romania
ionutmihainiculescu@gmail.com

Cristian Ştefan
Department of Computer Science, University of Piteşti,
Târgul din Vale 1, Piteşti, Romania
liviu.stefan@yahoo.com

We solve the Firing Squad Synchronization Problem (FSSP), for P systems based on digraphs with simplex channels, where communication is restricted by the direction of structural arcs. Previous work on FSSP for P systems focused exclusively on P systems with duplex channels, where communication between parents and children is bidirectional. Our P solution, the first for simplex channels, requires cell IDs, strongly connected digraphs and some awareness of the local topology (such as each cell's outdegree)—we argue that these requirements are necessary. Compared to the known solutions for cellular automata, our solution is substantially simpler and faster.

Keywords: P systems, digraphs, strongly connected, simplex channels, firing squad synchronization, cellular automata.

1 Introduction

The *Firing Squad Synchronization Problem* (FSSP), originally proposed by Myhill in 1957 [14], is one of the best studied problems for cellular automata. Essentially, the problem involves programming a network of cellular automata, so that, after the *firing* order is given by the *general*, after some finite time, all the cells in the system enter a designated *firing* state, *simultaneously* and *for the first time*.

Versions of FSSP have been proposed and studied for variety of network structures, from simple linear graphs to rings, trees, or general connected graphs; see, for example, [2, 4, 10, 11, 12, 15, 17, 23, 25, 26]. However, most of these versions, require *bidirectional* communication (i.e. *duplex* channels): only a few notable exceptions consider the more restricted *unidirectional* communication (i.e. *simplex* channels), starting with Kobayashi [12]. Later, Even, Litman and Winkler [9] proposed improved solutions for arbitrary *undirectional strongly-connected digraph*, working in $O(N^2)$ steps, where N is the digraph *size* (number of cells). Ostrovsky and Wilkerson [18] improved this further, to a solution which runs in $O(ND)$ steps, where D is the digraph *diameter* (typically smaller than N)— this still seems to be the best solution available.

Several FSSP solutions have recently been studied in the framework of P systems, although with somehow different formulations, stemming from their different computing capabilities. P solutions were proposed: for *trees*, by Bernardini et al. [3] and Alhazov et al. [1]; and for arbitrary *connected graphs*, by Dinneen at al. [6, 7, 8]. All these P solutions require *duplex* channels and follow the typical pattern of a *wave* algorithm [24], using three phases:

© F. Ipate, R. Nicolescu, I.M. Niculescu and C. Ştefan
All the rights to the paper remain with the authors.

1. a *first broadcast*—which follows all shortest paths from the *general* and builds a *virtual BFS tree* (or *dag*);
2. a *convergecast*—which helps determine the *general's eccentricity*;
3. a *second broadcast*—which carries the actual firing command (with a countdown counter).

The best P solutions need $e_g + k$ steps for each of the three phases, for a total of $3e_g + k$ steps, where e_g is the general's eccentricity (height for trees, if the general is at the root).

Obviously, while the two broadcasts, of phases (1) and (3), would also work with simplex channels, duplex channels are essential for the convergecast of phase (2), where children need to talk back to their parents. At first sight, the convergecast seems impossible for simplex channels. However, children can still talk back to their parents, if the digraph is *strongly-connected*, albeit on a typically longer *path*. Moreover, if messages cannot be confused, all children can send messages to their parents, in parallel (overlapping in time without problems), achieving this way a *virtual convergecast*.

Based on ideas from Ostrovsky and Wilkerson [18], we propose a *first* FSSP solution for P systems with *simplex* channels, based on arbitrary *strongly-connected digraphs*. Our solution runs in $O(e_gD)$ steps, specifically: (1) e_g steps for the first broadcast; (2) $O(e_gD)$ steps for the virtual convergecast (maximally parallelized); (3) e_g steps for the second broadcast. Thus, in terms of execution time, our P solution compares favourably with the best known solution for cellular automata. Taking into account the different problem constraints and different computing capabilities of P systems vs. cellular automata, we were expecting a simpler and faster solution, but not necessarily such a substantial speed improvement. However, for the reasons mentioned, any performance comparison must be viewed with a grain of salt.

The actual design seems challenging and requires careful selection of the most adequate ingredients, some of which are available in cellular automata, but not typically available in P systems. Specifically, we argue that the P solution requires: (1) reified cell IDs; (2) reified local network information, such as the number of outgoing arcs (this information *is* available in cellular automata); and (3) high-level generic rules, if we want to have a fixed rule set, independent of the actual network size.

2 Preliminaries

We assume that the reader is familiar with the basic terminology and notations, such as relations, graphs, nodes (vertices), edges, directed graphs (digraphs), directed acyclic graphs (dags), arcs, alphabets, strings and multisets.

A *P system* is a parallel and distributed computational model, inspired by the structure and interactions of cell membranes. This model was introduced by Păun in 1998–2000 [19]. An in-depth overview of this model can be found in Păun et al. [22].

In this paper, we consider an *ad-hoc* definition of P systems, based on our definition of *simple P module* [5], which extends earlier versions of tissue and neural P systems [13, 20]. However, here we intentionally *restrict* rule transfer mode to broadcast to all children, $\downarrow_\forall$.

Definition 1 *A P system of order n with simplex channels is a system* $\Pi = (O, K, \delta)$*, where:*

1. *O is a finite alphabet of elementary symbols; strings over O are interpreted as multisets;*
2. $K = \{\sigma_1, \sigma_2, \dots, \sigma_n\}$ *is a finite set of cells; where each cell is a system* $\sigma_i = (Q_i, R_i)$*, with* Q_i *a finite set of states and* R_i *a finite set of rewriting rules over O, further detailed below.*

3. *δ is an irreflexive binary relation on K, which represents a set of structural arcs between cells, with unidirectional communication capabilities, strictly from parents to children.*

4. *Each R_i is a finite linearly ordered set of multiset rewriting rules with promoters, of the form: $S\,x \rightarrow_\alpha S'\,x'\,(y)_{\downarrow_\forall} \cdots |_z$, where $S, S' \in Q_i$, $x, x', y \in O^*$, $z \in O^*$ is the promoter, $\alpha \in \{\mathtt{min}, \mathtt{max}\}$ is a rewriting operator and $\downarrow_\forall$ is a transfer operator, here restricted to send y messages from a parent to all its children.*

As usually, each cell, $\sigma_i \in K$, starts from an *initial configuration* (S_{i0}, w_{i0}), where $S_{i0} \in Q_i$ is its *initial state* and $w_{i0} \in O^*$ is its *initial content*. A cell *evolves* by applying one or more rules, which can change its *current configuration*, i.e. its current state and current content, and send symbols to its children.

The application of a rule transforms the *current state S* to the *target state S'*, rewrites multiset x as x' and sends multiset y by replication to all its children. Note that, multisets x' and y will not be visible to further rules in this same step, but they will become visible after no more rules are applicable, i.e. they will be available since next step only. *Promoters* are symbols which enable rules, but are not consumed by the rules' application.

When an applicable rule is applied, its rewriting operator α indicates how many times it is actually applied: *once*, if $\alpha = \mathtt{min}$; or *as many times as possible*, if $\alpha = \mathtt{max}$.

As used here, rules have *priorities* and are applied in *weak priority* order [21], with special attention to target state compatibility: (1) higher priority applicable rules are applied before lower priority applicable rules, and (2) a lower priority applicable rule is applied only if it indicates the same target state as the previously assigned rules (if any).

All cells evolve *synchronously* in one global *step*. An *evolution* of a P system is a sequence of steps, where each cell starts from its initial configuration. An execution *halts* if no cell can evolve.

2.1 Further P systems extensions

We let each cell, σ_i, start with its own unique *cell ID* symbol, ι_i. We thus *reify* the conceptual cell index, i, into an internal symbol, which is accessible to the rules, exclusively as an *immutable promoter* [16].

We enhance our vocabulary by recursive composition of *elementary symbols* from O into a simple form of *complex symbols* [16]. Such complex symbols can be viewed as complex molecules, consisting of elementary atoms or other molecules.

Further, complex symbols let us process our multisets with high-level *generic rules*, using *free variable* matching. To explain these additional ingredients, consider this hypothetical rule (which uses an additional transfer mode, targeted to a specific child, not considered in the definition used here):

$$S\;a\;n_j \rightarrow_{\mathtt{min.min}} S'\;b\;(c_i)_{\downarrow_j}\;|\iota_i.$$

This is a *generic* rule, which uses an *extended rewriting mode*, with complex symbols, c_i and n_j, where i and j are free variables. In fact, c_i and n_j are just shorthands for *tuples* (c, i) and (n, j), or, equivalently, for *compound terms* $c(i)$ and $n(j)$. If needed, we can build more complex symbols by recursive composition; e.g., we could have complex symbols such as $d(e, i, f(j))$. Generally, a free variable could match anything, including another complex symbol. However, in this rule, i and j are constrained to match cell ID indices only:

1. i—because it also appears as the cell ID of the current cell, ι_i;

2. j—because it also indicates the target of the transfer mode, $\downarrow_j$.

A generic rule is identified by using an *extended* version of the "classical" rewriting mode, in fact, by a combined *instantiation* and *rewriting* mode. Our sample rule uses the extended mode `min.min`, where the two `min` operators have distinct semantics: the first `min` operator is new and describes the generic instantiation; the second `min` is the classical operator, which describes the rule application. Briefly:

1. according to the first `min`, this rule is *instantiated once*, for one of the existing n_j symbols (if any), while promoter, ι_i, constrains i to the cell ID index of the current cell, σ_i;
2. according to the second `min`, the instantiated rule is *applicable once*, i.e. if applied, it consumes one a and one n_j, produces one b and sends one c_i to child σ_j (if this exists).

As a further example, consider the scenario in which the current cell, σ_1, contains the multiset $n_2n_3n_3$. Here, our sample generic rule instantiates (randomly) *one* of the following two lower-level rules, which is then applied in the classical way (in the `min` rewriting mode):

$$S\ a\ n_2 \rightarrow_{\mathtt{min}} S'\ b\ (c_1)_{\downarrow_2}.$$
$$S\ a\ n_3 \rightarrow_{\mathtt{min}} S'\ b\ (c_1)_{\downarrow_3}.$$

We consider four basic combinations of the instantiation and rewriting modes, `min.min`, `min.max`, `max.min`, `max.max`; their semantics is:

- `min.min` indicates that the generic rule is (randomly) instantiated *once*, if possible, and the instantiated rule is applied *once*, if possible.
- `min.max` indicates that the generic rule is (randomly) instantiated *once*, if possible, and the instantiated rule is applied as *many* times as possible.
- `max.min` indicates that the generic rule is instantiated as *many* times as possible, without superfluous instances (i.e. without duplicates or instances which are not applicable), and each one of the instantiated rules is applied *once*, if possible.
- `max.max` indicates that the generic rule is instantiated as *many* times as possible, without superfluous instances (i.e. without duplicates or instances which are not applicable), and each one of the instantiated rules is applied as *many* times as possible.

All instantiations are ephemeral, created when rules are tested for applicability and disappearing at the end of the step.

3 FSSP problem for P systems with simplex channels

We are required to find:

1. an alphabet O;
2. a *cell prototype* $\sigma = (Q,R)$, where
   (a) R is a set of rules over O;
   (b) Q contains two distinguished states:
      - S_0: a *quiescent* state, i.e. if σ is in state S_0 and *empty*, then there are no applicable rules;
      - S_f: a *final* state, i.e. if σ is in state S_f, then there are no applicable rules.

such that, given:

1. *any* finite set of σ copies, $K = \{\sigma_1, \sigma_2, \ldots, \sigma_n\}$, $\sigma_i = \sigma$;

2. connected via *any* strongly-connected digraph δ;

the P system $\Pi = (O, K, \delta)$ with *simplex channels* will evolve according to the following specification:

1. all cells start from quiescent state S_0: $S_{i0} = S_0$;
2. except a distinguished cell σ_g, called the *general*, all cells start with a restricted initial content, containing, the reified cell ID and a reified count of the cell's outdegree: $w_{i0} \subseteq \{\iota_i, c^{\texttt{outdegree}(\sigma_i)}\}, \forall i \neq g$;
3. the evolution terminates and, during its last step: all cells enter state s_f *simultaneously* and *for the first time*.

Remark 1 *Our formulation has different constraints than the original problem for cellular automata. In the cellular automata formulation, there is a given fixed bound on the number of input and output connections of each cell (bounded indegree and outdegree). In our formulation, there is no bound on the number of input and output connections that a cell may have. However, this is compensated by the fact that in our formulation there are no size bounds on messages or cells' internal memory. These trade-offs, as well as different computing capabilities, suggests that performance comparisons must be viewed with a grain of salt.*

Remark 2 *We argue that both the reified cell ID and the reified children count, or equivalent information, are necessary, definitely for our approach, and, likely, for any other approach. Note that children counts are implicit in the cellular automata version, where unconnected channels (out of the fixed sized pool) can be detected.*

Remark 3 *There is no constraint on the cells' final contents. However, if needed, any left-over garbage could be collected in one extra step.*

Remark 4 *Practically, we are only required to design the rule set, R, because this implies the alphabet, O, and the state set, Q.*

Remark 5 *Note the rule set, R, must be fixed and applicable to any structural digraph. This is a strong requirement: we require a rule set which is independent of the size and structure of the actual system.*

4 FSSP solution for P systems with simplex channels

Our solution runs in three phases (conceptually similar to the duplex case):

1. First phase: a *first broadcast* from the general. This phase builds the *virtual-dag* (the virtual BFS dag) and, for each cell: (a) records its *virtual-dag-parent(s)*; and (b) successively computes its *depth* attribute, which represents this cell's depth level in the virtual-dag (the same as this cell's digraph distance from the general). A first phase broadcast message is a complex symbol, $x_{k,i}$, where i is the sender's ID and k is the next *depth* level (σ_i's own *depth* plus one).
2. Second phase: a *virtual convergecast* from the virtual-dag leaves. For each cell, this phase successively computes the *max-depth* attribute, which represents the maximum depth over all descendant cells in the virtual-dag. In the end, the general's *max-depth* is its eccentricity.

   This virtual convergecast simulates impossible direct virtual-dag-child to virtual-dag-parent messages, by broadcasting them over the digraph (using ad-hoc BFS dags). A convergecast message is a complex symbol, $a_{j,i,k}$, where i is the sender's ID, j is its virtual-dag-parent ID and k is σ_i's own *max-depth*. In addition to virtual-dag leaves, a pseudo-convergecast message is also sent by a digraph cell to its digraph parents with larger *depth* attributes (if any). Because each message is uniquely identified by both it sender and its destination, any number of such convergecasts can run in parallel, without creating confusions.

Note that a cell *needs* to know its digraph outdegree—to detect when it receives its last outstanding convergecast message and to start its own convergecast. However, a cell does *not* know the identities of its digraph children, or how long a message from any one of them will take to reach it.

3. Third and last phase: a *second broadcast* from the general, with a countdown to firing. A last phase broadcast message is a complex symbol, f_k, where k is the next countdown counter (σ_i's own countdown minus one).

As possible extensions, not discussed here, we can extract from our solution a more general subprogram, to send any message from any cell to any other cell, which can run in parallel, without creating confusions. Also, we can consolidate the routing information, to speedup future messages with the same destination; however, this feature is not required here (each convergecast is performed exactly once).

Figures 1–8 show bird's eye views of the evolution of Π: a sample P system, with simplex channels, based on a strongly connected digraph.

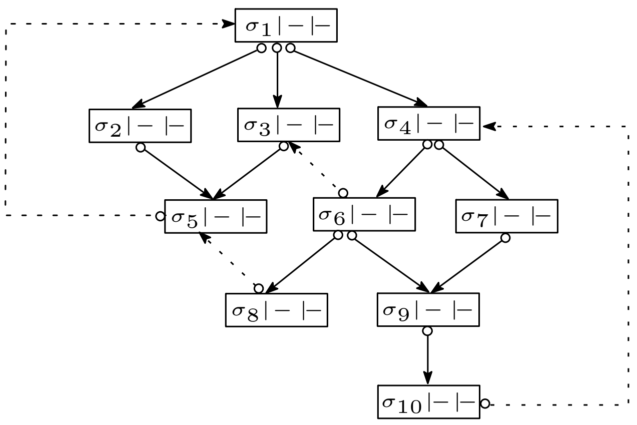


Figure 1: Π: Initial configuration of a sample P system with simplex channels, based on a strongly connected digraph. Normal arrows are arcs in the virtual dag, created by a BFS broadcast, started from the general, σ_1. The remaining digraph arcs are dotted arrows. Each cell knows its digraph outdegree, indicated by small circles at outgoing arrows' tails. Each cell blob shows three attributes, in order: (1) its cell ID; (2) its *depth* attribute (computed by the first broadcast); and (3) its *max-depth* attribute (computed by the virtual convergecast). At this stage, the *depth* and *max-depth* attributes are still indeterminate and all small circles are white, indicating still outstanding convergast messages.

4.1 Rule set R

0. Rules in state S_0:
    1. $S_0\ a \to_{\mathtt{min.min}}$ $S_1\ g\ n_0\ m_0\ (x_{1,i})_{\downarrow\forall}\ |\iota_i$
    2. $S_0\ x_{k,j} \to_{\mathtt{max.min}}$ $S_1\ l_k\ p_j\ (x_{k+1,i})_{\downarrow\forall}\ |\iota_i$
    3. $S_0\ x_{k,j} \to_{\mathtt{max.max}} S_1\ p_j\ |\iota_i$
1. Rules in state S_1:
    1. $S_0\ l_k \to_{\mathtt{max.min}} S_1\ n_k\ m_k$
    2. $S_0\ l_k \to_{\mathtt{max.max}} S_1$
    3. $S_1\ x_{k,j} \to_{\mathtt{max.min}} S_1\ y_{j,0}$
    4. $S_1\ x_{k,j} \to_{\mathtt{max.max}} S_1$
    5. $S_1\ s_k \to_{\mathtt{max.min}} Max\ w\ m_k$
    6. $S_1\ y_{j,k} \to_{\mathtt{max.min}}$ $S_1\ v_{j,i}\ (a_{j,i,k})_{\downarrow\forall}\ |\iota_i$
    7. $S_1\ a_{j,k,l} \to_{\mathtt{max.max}} S_1\ |v_{j,k}$
    8. $S_1\ a_{i,j,k} \to_{\mathtt{max.min}} S_1\ v_{i,j}\ s_k\ |\iota_i$
    9. $S_1\ a_{j,k,l} \to_{\mathtt{max.min}}$ $S_1\ v_{j,k}\ (a_{j,k,l})_{\downarrow\forall}$
2. Rules in state S_2:
    1. $S_2\ t \to_{\mathtt{min}} S_5\ (t)_{\downarrow\forall}$
    2. $S_2\ s_k \to_{\mathtt{max.min}} Max\ w\ m_k$
    3. $S_2\ y_{j,k} \to_{\mathtt{max.min}}$ $S_2\ v_{j,i}\ (a_{j,i,k})_{\downarrow\forall}\ |\iota_i$
    4. $S_2\ a_{j,k,l} \to_{\mathtt{max.max}} S_2\ |v_{j,k}$
    5. $S_2\ a_{i,j,k} \to_{\mathtt{max.min}} S_2\ v_{i,j}\ s_k\ |\iota_i$
    6. $S_2\ a_{j,k,l} \to_{\mathtt{max.min}}$ $S_2\ v_{j,k}\ (a_{j,k,l})_{\downarrow\forall}$
3. Rules in state S_3:
    1. $S_3\ cc \to_{\mathtt{min}} S_1\ ce$
    2. $S_3\ c \to_{\mathtt{min}} S_4\ eb$
3. Rules in state S_4:
    1. $S_4\ \to_{\mathtt{max.min}}$ $S_2\ t\ f_k\ (t)_{\downarrow\forall}\ |g\ b\ m_k$
    2. $S_4\ \to_{\mathtt{max.min}} S_2\ y_{j,k}\ |p_j\ m_k$
4. Rules in state Max (needs refinement):
    1. $Max\ m_k \to_{\mathtt{max}} S_3\ |m_{k+l}$
5. Rules in state S_5:
    1. $S_5\ v_{k,l} \to_{\mathtt{max.max}} S_f\ |f_0$
    2. $S_5\ p_k \to_{\mathtt{max.max}} S_f\ |f_0$
    3. $S_5\ n_k \to_{\mathtt{max.max}} S_f\ |f_0$
    4. $S_5\ m_k \to_{\mathtt{max.max}} S_f\ |f_0$
    5. $S_5\ e \to_{\mathtt{min.max}} S_f\ |f_0$
    6. $S_5\ b \to_{\mathtt{min.max}} S_f\ |f_0$
    7. $S_5\ \to_{\mathtt{min.min}} S_f\ |f_0$
    8. $S_5\ v_{k,l} \to_{\mathtt{max.max}} S_5\ |f_k$
    9. $S_5\ p_k \to_{\mathtt{max.max}} S_5\ |f_k$
    10. $S_5\ n_k \to_{\mathtt{max.max}} S_5\ |f_k$
    11. $S_5\ m_k \to_{\mathtt{max.max}} S_5\ |f_k$
    12. $S_5\ e \to_{\mathtt{min.max}} S_5\ |f_k$
    13. $S_5\ b \to_{\mathtt{min.max}} S_5\ |f_k$
    14. $S_5\ f_k \to_{\mathtt{max.min}}$ $S_5\ f_{k-1}\ (f_{k-1})_{\downarrow\forall}$
    15. $S_5\ f_k \to_{\mathtt{max.max}} S_5$
    16. $S_5\ t \to_{\mathtt{max}} S_5$

4.2 Alphabet O, elementary and complex symbols

- ι_i: reified cell ID;
- a: starts the process from the cell which next assume the general role;
- g: marks the general;
- $x_{j,k}$: complex symbol broadcasted in the first phase;

- n_k: indicates the *depth* attribute, k;
- m_k: indicates the *max-depth* attribute, k;
- l_k: auxiliary symbol used to compute *depth* attribute, n_k;
- p_k: pointer to a parent cell;
- $y_{i,k}$: initiates the sending of message k to σ_i;
- $a_{i,j,k}$: complex symbol broadcasted in the convergecast phase, sent from σ_j to σ_i and carrying payload k, representing σ_j's *max-depth* (if known, otherwise it is a pseudo-convergcast sent by a cell with lower *depth*);
- $v_{i,j}$: records the passage of a message from σ_j to σ_i;
- s_k: auxiliary symbol used to compute *max-depth* attribute, m_k, via maximum;
- c: used to count the children which have not yet sent their convergecast messages (initially cell's outdegree);
- e: used to count the children which have already sent their convergecast messages (initially zero);
- b : marks a cell which has performed its convergecast;
- t: auxiliary symbol used in the countdown to firing;
- f_k: complex symbol broadcasted in the last phase, carrying the countdown to firing;

4.3 Brief description

1. State S_0: initiates the first broadcast and computes *depth* attributes.
2. State S_1: continues the first broadcast and builds the virtual-dag.
3. State S_2: performs the actual convergecast.
4. State S_3: decides if a non-general cell is ready to start its convergecast, i.e. if it has received its all outstanding convergecast messages, from all its digraph children.
5. State S_4: decides if the general is ready to start the second broadcast (countdown to firing), i.e. if it has received convergecast messages from all its children.
6. State *Max*: prepares the convergecast, by determining the *max-depth* attribute.
7. State S_5: last broadcast, counts down to firing and erases unnecessary symbols.

5 Assessment

Theorem 1 *The synchronization time of the FSSP solution for digraph-based P systems with simplex channels is* $e_g + e_g D + e_g$*, i.e. bounded by* $O(e_g D)$*.*

Theorem 2 *The digraph must be strongly connected, otherwise, there is no solution.*

Theorem 3 *Cells must know the number of their outgoing arcs, otherwise, there is no solution.*

Theorem 4 *Cell IDs must be reified, otherwise, there is no solution.*

6 Experimental results

Besides our earlier example, we have empirically validated our solution in several test scenarios, with digraphs of different shapes and sizes, for example:

- A simple ring (Figure 9).
- A main ring linking a series of smaller rings of size two (Figure 10).
- A main ring linking a series of smaller rings of size three (Figure 11).
- A main ring linking a series of smaller rings of increasing size (Figure 12).
- A set of 11 random directed graphs, with up to 70 nodes each, generated by the standard `networkx` package.

The results support our claim for correctness and performance.

7 Conclusions

In this paper, we explicitly presented a first solution to the FSSP for synchronous digraph-based P systems with *simplex* channels. Our design suggests, but does not need, ways to consolidate routing information in such systems—this can be a topic for further study.

Our solution runs in $O(eD)$ steps and compares favourably, i.e. it is simpler and faster, than the best known solution for cellular automata [18], which runs in $O(ND)$ steps. Taking into account the different problem constraints and different computing capabilities of P systems vs cellular automata, we were expecting a simpler and a faster solution, but not necessarily such a substantial speed improvement. However, as noted before, any performance comparison must be viewed with a grain of salt.

Our solution used a fixed size high-level rule set, independent of the number of cells in the actual system and of its structure. This supports the case for reified cell IDs, complex symbols and generic rules and suggests that such ingredients could be useful or even essential in any distributed or just large system.

Acknowledgments

The authors wish to acknowledge the contribution of Michael Dinneen, John Morris and Yun-Bum Kim, and the support of the University of Auckland FRDF grant 9843/3626216. The work of Florentin Ipate and Ionuţ Mihai Niculescu was supported by CNCSIS-UEFISCSU, project number PNII-IDEI 643/2008. The work of Cristian Ştefan was supported by SOP-HRD grant 52826.

References

[1] A. Alhazov, M. Margenstern, and S. Verlan. Fast Synchronization in P Systems. In D. W. Corne, P. Frisco, G. Păun, G. Rozenberg, and A. Salomaa, editors, *Workshop on Membrane Computing*, volume 5391 of *Lecture Notes in Computer Science*, pages 118–128. Springer, 2008.

[2] R. Balzer. An 8-state Minimal Time Solution to the Firing Squad Synchronization Problem. *Information and Control*, 10(1):22–42, 1967.

[3] F. Bernardini, M. Gheorghe, M. Margenstern, and S. Verlan. How to Synchronize the Activity of All Components of a P System? *Int. J. Found. Comput. Sci.*, 19(5):1183–1198, 2008.

[4] A. Berthiaume, T. Bittner, L. Perkovic, A. Settle, and J. Simon. Bounding the Firing Synchronization Problem on a Ring. *Theor. Comput. Sci.*, 320(2-3):213–228, 2004.

[5] M. J. Dinneen, Y.-B. Kim, and R. Nicolescu. Edge- and node-disjoint paths in P systems. *Electronic Proceedings in Theoretical Computer Science*, 40:121–141, 2010.

[6] M. J. Dinneen, Y.-B. Kim, and R. Nicolescu. Synchronization in P Modules. In C. S. Calude, M. Hagiya, K. Morita, G. Rozenberg, and J. Timmis, editors, *Unconventional Computation*, volume 6079 of *Lecture Notes in Computer Science*, pages 32–44. Springer-Verlag, Berlin Heidelberg, 2010.

[7] M. J. Dinneen, Y.-B. Kim, and R. Nicolescu. An Adaptive Algorithm for P System Synchronization. In *Proceedings of the Twelfth International Workshop on Membrane Computing, (CMC11), Fontainebleau/Paris, France*, pages 1–26, in press, 2011.

[8] M. J. Dinneen, Y.-B. Kim, and R. Nicolescu. Faster synchronization in P systems. *Journal of Natural Computing*, pages 1–17, in press, 2011.

[9] S. Even, A. Litman, and P. Winkler. Computing with Snakes in Directed Networks of Automata. *Journal of Algorithms*, 24:740–745, 1990.

[10] E. Goto. A Minimal Time Solution of the Firing Squad Problem. Course notes for Applied Mathematics 298, pages 52–59, Harvard University, 1962.

[11] J. J. Grefenstette. Network Structure and the Firing Squad Synchronization Problem. *J. Comput. Syst. Sci.*, 26(1):139–152, 1983.

[12] K. Kobayashi. The Firing Squad Synchronization Problem for a Class of Polyautomata Networks. *J. Comput. Syst. Sci.*, 17(3):300–318, 1978.

[13] C. Martín-Vide, G. Păun, J. Pazos, and A. Rodríguez-Patón. Tissue P systems. *Theor. Comput. Sci.*, 296(2):295–326, 2003.

[14] E. F. Moore. The Firing Squad Synchronization Problem. *Moore, E.F. (ed.) Sequential Machines, Selected Papers*, pages 213–214, 1964.

[15] F. R. Moore and G. G. Langdon. A Generalized Firing Squad Problem. *Information and Control*, 12(3):212–220, 1968.

[16] R. Nicolescu and H. Wu. BFS solution for disjoint paths in P systems. In C. Calude, J. Kari, I. Petre, and G. Rozenberg, editors, *Unconventional Computation*, volume 6714 of *Lecture Notes in Computer Science*, pages 164–176. Springer Berlin / Heidelberg, 2011.

[17] Y. Nishitani and N. Honda. The Firing Squad Synchronization Problem for Graphs. *Theor. Comput. Sci.*, 14:39–61, 1981.

[18] R. Ostrovsky and D. S. Wilkerson. Faster Computation On Directed Networks of Automata. In *In Proceedings of the Fourteenth Annual ACM Symposium on Principles of Distributed Computing*, pages 38–46. ACM, 1995.

[19] G. Păun. Computing with Membranes. *Journal of Computer and System Sciences*, 61(1):108–143, 2000.

[20] G. Păun. *Membrane Computing: An Introduction*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2002.

[21] G. Păun. Introduction to Membrane Computing. In G. Ciobanu, M. J. Pérez-Jiménez, and G. Păun, editors, *Applications of Membrane Computing*, Natural Computing Series, pages 1–42. Springer-Verlag, 2006.

[22] G. Păun, G. Rozenberg, and A. Salomaa. *The Oxford Handbook of Membrane Computing*. Oxford University Press, Inc., New York, NY, USA, 2010.

[23] H. Schmid and T. Worsch. The Firing Squad Synchronization Problem with Many Generals For One-Dimensional CA. In J.-J. Lévy, E. W. Mayr, and J. C. Mitchell, editors, *IFIP TCS*, pages 111–124. Kluwer, 2004.

[24] G. Tel. *Introduction to Distributed Algorithms*. Cambridge University Press, 2000.

[25] H. Umeo, N. Kamikawa, K. Nishioka, and S. Akiguchi. Generalized Firing Squad Synchronization Protocols for One-dimensional Cellular Automata — A Survey. *Acta Physica Polonica B Proceedings Supplement*, 3(2):267–289, 2010.

[26] A. Waksman. An optimum solution to the firing squad synchronization problem. *Information and Control*, 9(1):66–78, 1966.

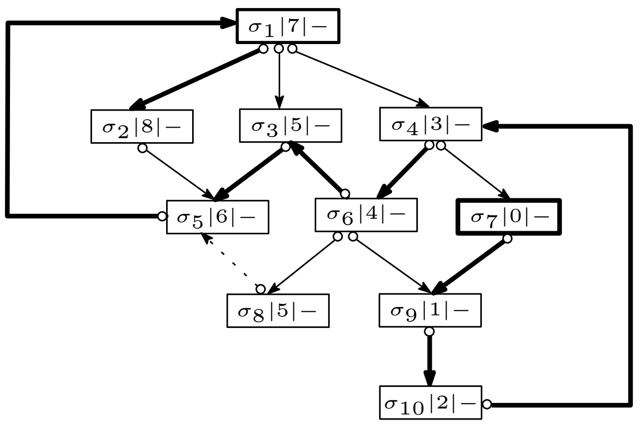


Figure 2: Π: route from σ_7 to σ_2, indicated via thick arrows.

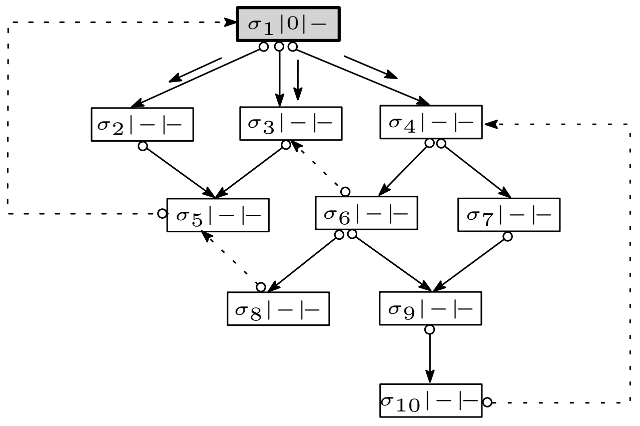


Figure 3: Π, step 1: The general, σ_1, starts the first phase, by broadcasting the complex symbol $a_{1,1}$.

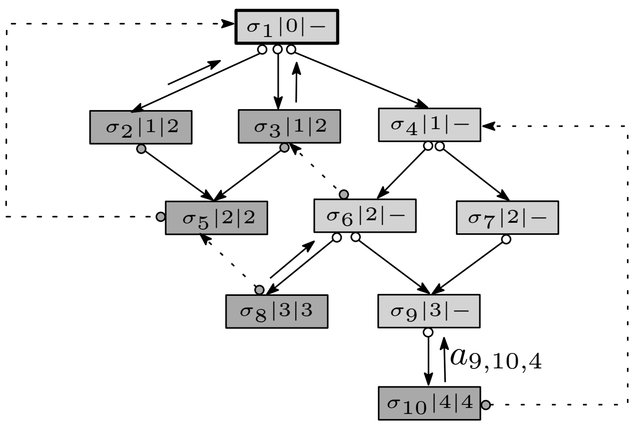


Figure 4: Π, step 9: Cell σ_{10} has just learned that it is a virtual-dag leaf, by receiving cell's σ_4's pseudo-convergecast message, the complex symbol $a_{10,4,-}$. Cell σ_{10} starts now its virtual convergecast, by broadcasting the complex symbol $a_{9,10,4}$, towards its virtual-dag-parent, σ_9. At this stage, each cell knows its virtual-dag-parent(s) and its own *depth* attribute. All other cells have already initiated their virtual convergecasts. All leaves, including σ_{10}, and some other cells already know their *max-depth* attribute: exactly, if they have received all their outstanding convergecast messages, or a lower bound, otherwise. Message $a_{9,10,4}$ will take three more steps: via paths $\sigma_{10}.\sigma_4.\sigma_6.\sigma_9$ and path $\sigma_{10}.\sigma_4.\sigma_7.\sigma_9$. Three other, earlier stared, virtual convergecasts run in parallel: σ_2 to σ_1, σ_3 to σ_1, σ_8 to σ_6. Smaller arrows near structural arcs indicate virtual convergecasts.

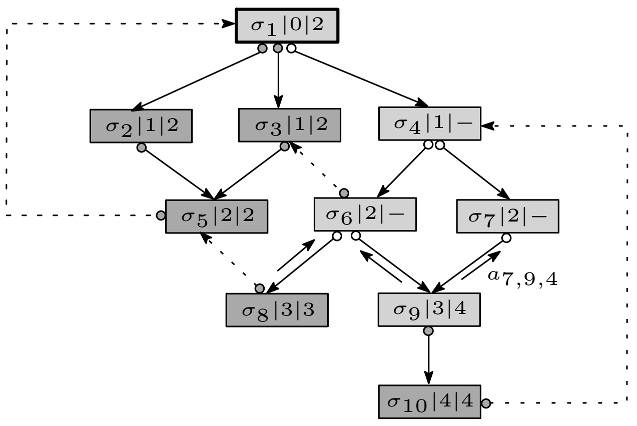


Figure 5: Π, step 12: After receiving its single expected convergecast message, cell σ_9 starts its own virtual convergecasts towards its parents, σ_6 and σ_7, by broadcasting complex symbols, $a_{6,9,4}$ and $a_{7,9,4}$, respectively. Each of these messages will take three more steps, via path $\sigma_9.\sigma_{10}.\sigma_4.\sigma_6$ and via path $\sigma_9.\sigma_{10}.\sigma_4.\sigma_7$, respectively. These two convergecasts run in parallel with another virtual convergecast: σ_8 to σ_6.

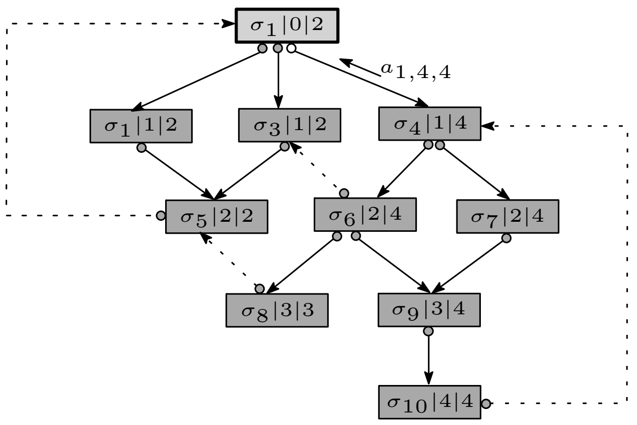


Figure 6: Π, step 18: The convergecast phase is almost completed. Cell σ_4 starts its convergecast towards its parent, σ_1, by broadcasting the complex symbol $a_{1,4,4}$. This convergecast will take four more steps, via paths $\sigma_4.\sigma_6.\sigma_3.\sigma_5.\sigma_1$ and path $\sigma_4.\sigma_6.\sigma_8.\sigma_5.\sigma_1$. This is cell σ_1's last oustanding convergecast.

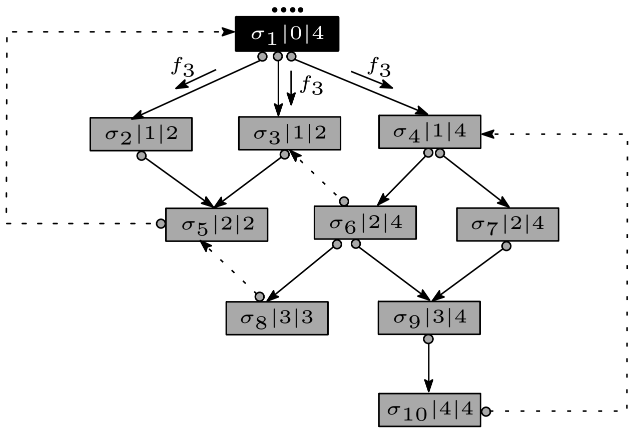


Figure 7: Π, step 22: General σ_1 starts the last phase, by broadcasting the complex symbol f_3, carrying the countdown to firing. Small dots above a cell indicate the cell's countdown counter, which is also broadcasted to all its digraph children.

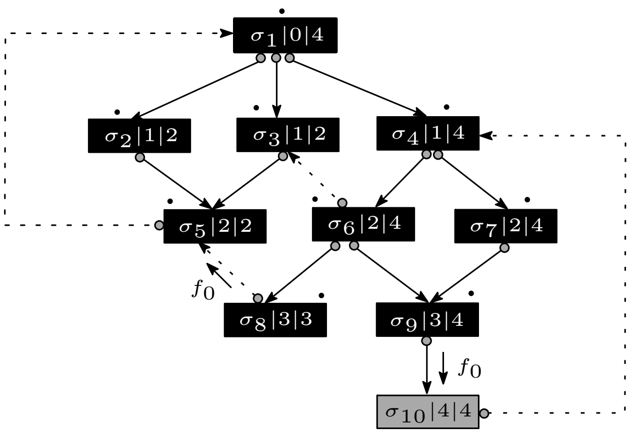
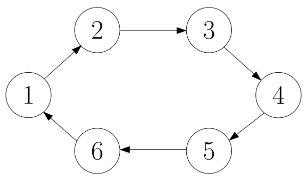


Figure 8: Π, step 25: The last phase is almost complete. Cell σ_9 forwards the complex symbol f_0 to σ_{10}. This the last step before firing (not illustrated here).

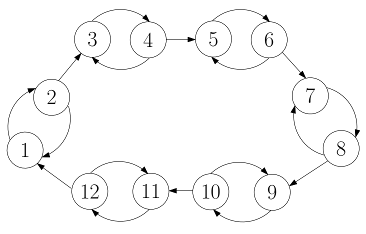


(a) Sample network, $N = 6$.

N	e_g	D	Steps
2	1	1	18
3	2	2	29
4	3	3	42
5	4	4	57
6	5	5	74
7	6	6	93
8	7	7	114
9	8	8	137
10	9	9	162
11	10	10	189
12	11	11	218
13	12	12	249
14	13	13	282
15	14	14	317

(b) Results.

Figure 9: Ring networks.

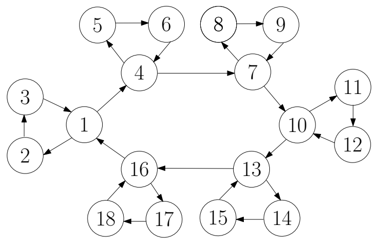


(a) Sample network, $N = 10$.

N	e_g	D	Steps
2	1	1	18
4	3	3	40
6	5	5	62
8	7	7	90
10	9	9	122
12	11	11	158
14	13	13	198
16	15	15	242
18	17	17	290
20	19	19	342

(b) Results.

Figure 10: Ring networks of size 2 rings.

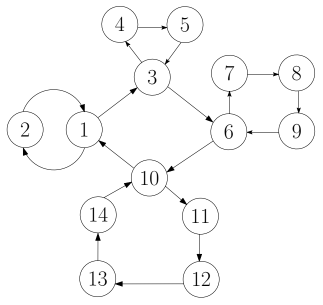


(a) Sample network, $N = 18$.

N	e_g	D	Steps
5	4	4	57
10	5	5	67
15	6	6	78
20	7	7	90
25	8	8	101

(b) Results.

Figure 11: Ring networks of size 3 rings.



(a) Sample network, $N = 14$.

N	e_g	D	Steps
2	1	1	18
5	3	3	40
9	5	5	63
14	7	7	90
20	9	9	121
27	11	11	156

(b) Results.

Figure 12: Ring networks of increasing size rings.

Further Results on Languages of Membrane Structures

Rama Raghavan
Department of Mathematics
Indian Institute of Technology Chennai
Chennai, India
ramar@iitm.ac.in

H. Ramesh
Department of Mathematics
Indian Institute of Technology Guwahati
Guwahati, India
ramesh_h@iitg.ernet.in

Marian Gheorghe
Department of Computer Science
The University of Sheffield
Sheffield, UK
M.Gheorghe@dcs.shef.ac.uk

Shankara Narayanan Krishna
Department of Computer Science and Engineering
Indian Institute of Technology Bombay
Powai, Mumbai, India
krishnas@iitb.ac.in

In [3], P systems with active membranes were used to generate languages, in the sense of languages associated with the structure of membrane systems. Here, we analyze the power of P systems with membrane creation and dissolution restricted to elementary membranes, P systems without membrane dissolution operating according to certain output modes. This leads us to characterizations of recursively enumerable languages.

1 Introduction

In [3], an alternative approach to generate languages by means of P systems was proposed. An appropriate representation for a string was built by means of a membrane structure and then the string is generated by visiting the membrane structure according to a well-specified strategy. P systems with active membranes were considered, allowing membrane creation or division or duplication and dissolution, where the output of a computation may be obtained either by visiting the tree associated with the membrane structure, or by following the traces of a specific object, called traveller, or sending out the objects. For each of these approaches, characterizations of recursively enumerable languages were provided based on P systems that use different sets of operations for modifying the membrane structure.

The output of a computation was considered not as a single entity, which is either sent out of the system or collected in a specific membrane. Instead the output is given by catenating the content or the labels of each region of the whole configuration reached by the system at the end of a computation. They considered a general class of P systems with active membranes equipped with membrane division, creation, duplication and dissolution operations. Membrane duplication means that, starting from an existing membrane, we can create a new membrane which encloses the existing one. Then three different approaches for collecting the output of a computation was given namely visiting the tree associated with the membrane structure, following the traces of a special object (traveller traces), and sending out the objects (external mode). The trace mode and external mode were investigated earlier in the literature. For the external mode, the main difference with respect to this approach is that, before sending out the objects, we need to prepare an appropriate membrane structure where the output objects are supposed to be distributed according to a specific strategy.

The approach presented in [3], is related to the problem of finding alternative ways to define the output of the computation in membrane systems. In fact, this method puts emphasis on the structure of the membranes whose role is important in successful computations.

Proceedings of 5th Workshop on Membrane Computing
and Biologically Inspired Process Calculi (MeCBIC 2011)
Pages 73–82, 2011.

© R. Raghavan, H. Ramesh, M. Gheorghe and S.N. Krishna
All the rights to the paper remain with the authors.

In this paper, we investigate the computational power of P systems with active membranes equipped with membrane creation and membrane dissolution restricted to elementary membranes operating according to all four output modes. Also we analyse the power of P systems without membrane dissolution operating according to the three identified output modes. We need the label changing feature of *in* type rules to obtain the universality in the second case.

The paper is organized as follows. Section 3 recalls the definition of P systems with active membranes together with the definition of three different output modes. In section 3.1 we state the results from [3] concerning the power of P systems with active membranes generating languages of membrane structures. In sections 4 and 5, we prove characterizations of recursively enumerable languages by means of P systems with active membranes equipped with the membrane creation and dissolution operations.

2 Some Prerequisites

In this section we introduce some formal language theory notions which will be used in this paper; for further details, refer to [8].

For an alphabet V, we denote by V^* the set of all strings over V, including the empty one, denoted by λ. By *RE* we denote the family of recursively enumerable languages.

In our proofs in the following sections we need the notion of a *matrix grammar with appearance checking*. Such a grammar is a construct $G = (N,T,S,M,F)$, where N,T are disjoint alphabets, $S \in N$, M is a finite set of sequences of the form $(A_1 \to x_1,\ldots,A_n \to x_n), n \geq 1$, of context free rules over $N \cup T$, and F is a set of occurrences of rules in M (N is the nonterminal alphabet, T is the terminal alphabet, S is the axiom, while the elements of M are called matrices).

For $w,z \in (N \cup T)^*$ we write $w \Rightarrow z$ if there is a matrix $(A_1 \to x_1,\ldots,A_n \to x_n)$ in M and the strings $w_i \in (N \cup T)^*, 1 \leq i \leq n+1$, are such that $w = w_1, z = w_{n+1}$, and, for all $1 \leq i \leq n$, either (1) $w_i = w_i^{'}Aw_i^{''}$, $w_{i+1} = w_i^{'}x_iw_i^{''}$, for some $w_i^{'}, w_i^{''} \in (N \cup T)^*$, or (2) $w_i = w_{i+1}, A_i$ does not appear in w_i, and the rule $A_i \to x_i$ appears in F. (The rules of a matrix are applied in order, possibly skipping the rules in F if they cannot be applied - one says that these rules are applied in the *appearance checking mode*).

The language generated by G is defined by $L(G) = \{w | w \in T^*, S \Rightarrow^* w\}$. The family of languages of this form is denoted by MAT_{ac}. It is known that $MAT_{ac} = RE$.

We say that a matrix grammar with appearance checking $G = (N,T,S,M,F)$ is in the *Z-binary normal form* if $N = N_1 \cup N_2 \cup \{S,Z,\#\}$, with these sets mutually disjoint, the matrices of type 3 can also be of the form $(X \to Z, A \to \#)$, and the only matrix of type 4 (terminal matrix) is of the form $(Z \to \lambda)$.

According to Lemma 1.3.7 in [4], for each matrix grammar there is an equivalent matrix grammar in the binary normal form.

Next we define a computing device which is equivalent in power with Turing machine. Such a machine runs a program consisting of numbered instructions of several simple types. Several variants of register machines with different number of registers and different instruction sets were shown to be computationally universal (e.g., see [5]).

An *n-register machine* is a construct $M = (n,H,l_0,l_h,I)$, where:

- n is the number of registers,
- H is the set of instruction labels,
- l_0 is the initial label,
- l_h is the final label, and

- I is a set of labelled instructions of the form $l_i : (op(r), l_j, l_k)$, where $op(r)$ is an operation on register r of M, l_i, l_j, l_k are labels from the set H (which labels the instructions in a one-to-one manner),

The machine is capable of the following instructions:

> $(ADD(r), l_j, l_k)$: Add one to the contents of register r and proceed to instruction l_j or to instruction l_k; in the deterministic variants usually considered in the literature we demand $l_j = l_k$.
>
> $(SUB(r), l_j, l_k)$: If register r is not empty, then subtract one from its content and go to instruction l_j, otherwise proceed to instruction l_k.
>
> *halt*: Stop the machine. This additional instruction can only be assigned to the final label l_h.

When considering the generation of languages, we use the model of a *register machine with output tape* (e.g., see [2]) , which also uses a tape operation:

- $l : (write(a), l'')$: Write symbol a on the the output tape and go to l''.

We then also specify the output alphabet T in the description of the register machine with output tape, i.e., we write $M = (n, T, H, l_0, l_h, I)$. Let $L \subseteq V^*$ be a recursively enumerable language. Then L can be generated by a register machine with output tape and with 2 registers.

3 Languages of Membrane Structures

We consider a general class of P systems with active membranes equipped with membrane division, creation, duplication and dissolution operation. These operations represent abstractions of cellular biology processes of mitosis and membrane formation through self-assembling lipid bilayers [1]. We recall the definition of this P system from [3].

Definition 1 *A P system with active membranes is a construct*

$$\Pi = (V, K \cup \{0\}, \mu, w_0, w_1, \ldots, w_{m-1}, R)$$

where

1. *V is an alphabet; its elements are called objects;*
2. *K is an alphabet; its elements are called labels; the symbol $0 \notin K$ is the label of the skin membrane;*
3. *μ is a membrane structure containing $m \geq 1$ membranes; the skin membrane is labelled by 0 and all other membranes are labelled with symbols in K;*
4. *w_0 is the multiset associated with the skin membrane;*
5. *$w_i \in V^*$, for $1 \leq i \leq m-1$, is a multiset of objects associated with the membrane i;*
6. *R is a finite set of rules of the form:*
   a) *$[_i a \rightarrow v]_i$ with $a \in V, v \in V^*$, and $i \in K \cup \{0\}$ (inside a membrane i an object a is replaced by a multiset v),*
   b) *$[_i a]_i \rightarrow b[_i]_i$ with $a, b \in V$, and $i \in K \cup \{0\}$ (an object is sent out from a membrane, maybe modified),*
   c) *$a[_i]_i \rightarrow [_i b]_i$ with $a, b \in V$, and $i \in K \cup \{0\}$ (an object is moved into a membrane, the object may be modified),*

d) $[_i a \rightarrow [_j b]_j]_i$ *with* $a,b \in V$, $i \in K \cup \{0\}$, *and* $j \in K$ *(membrane creation: inside a membrane i, starting from an object a, a new elementary membrane j is created, which contains an object b),*

e) $[_i a]_i \rightarrow [_k b]_k [_j c]_j$ *with* $a,b,c \in V$, $i,j,k \in K$ *(membrane division: the membrane i, in the presence of an object a, is divided into two new membranes labelled by k and j, and the content (objects and sub-membranes) of the membrane i is copied into each new membrane where the object a is respectively replaced by b or c),*

f) $[_i a]_i \rightarrow [_k b [_j c]_j]_k$ *with* $a,b \in V$, $i,j,k \in K$ *(membrane duplication the membrane i, in the presence of an object a, is duplicated, that is, the label i is changed into j, the object a is replaced by c, and a new upper membrane labelled by k is created, which contains an object b),*

g) $[_i a]_i \rightarrow a$ *with* $a \in V$, *and* $i \in K$ *(membrane dissolution: in the presence of an object a, the membrane i is dissolved and its content (objects and sub-membranes) is released in the directly upper region).*

In the above system we have: an initial membrane structure with m membranes that contain m multisets associated with the regions, and a finite set of evolution rules. Moreover, as usual in P systems with active membranes, we also consider a distinct alphabet K which is used to label the membranes and is necessary to precisely identify the rules that can be applied inside every membrane. In general, in a P system with active membranes, the number of membranes can be increased and decreased arbitrarily and there can be many different membranes with the same label, which can be distinguished from each other only by the objects they contain. Thus, the labels from K make possible to keep finite the representation of a P systems by specifying a set of "types", each one with its own set of rules, for the membrane possibly present in the system at any time. A membrane with no further membrane inside is called an *elementary membrane*.

The set R contains rules for modifying both the number and the distribution of objects inside the system and the number and type of membranes which define the structure of the system. The former rules are expressed in the form of transformation and communication rules (rules of type (a), (b) and (c)) whereas the latter ones (rules of type (d), (e), (f) and (g)) comprise the operations of: membrane creation, membrane division, membrane duplication and membrane dissolution, respectively.

Remark 1 *Here, we do not consider the feature of membrane polarization for P systems with active membranes as reported in the literature. However, in the above definition, rules of more general forms are used that are able to change the labels of the membranes involved.*

As usual, P systems with active membranes evolve according to a non-deterministic maximal parallel strategy. Rules of type (a), (b), (c), and (d) are applied to all the objects which they can be applied to. Rules of type (e), (f), (g) are applied to all the membranes which they can be applied to. Obviously, in each step, the same membrane cannot be used by more than one rule of type (e), (f), (g) (i.e., a membrane cannot simultaneously be divided, duplicated and dissolved). More precisely, we assume that, in each step, the objects first evolve by means of rules of type (a), (b), (c), (d), and then the membranes evolve according to rules of type (e), (f), (g).

A computation is obtained by applying rules of R starting from the initial configuration. A computation is said successful if it reaches a configuration where no more rules can be applied.

We illustrate the application of rules (d) - (g) by examples. In the following examples, P,Q are possible contents of membranes and a,b,c are objects from V. The effect of the rules on some membrane structures is below:

(1) $[_iP\ a]_i$
rule of type (d) : We get $[_iP\ [_jb]_j]$
rule of type (e) : We get $[_jP\ b]_j\ [_kP\ c]_k$
rule of type (f) : We get $[_kb\ [_jP\ c]_j]_k$

(2) $[_j[_iP\ a]_i\ Q]_j$
rule of type (g) : We get $[_jPQ\ a]_j$

The result of a computation may be considered in various forms, which are called output modes.

- *Visiting the tree.* The result of a computation is the set of strings obtained by visiting the tree associated with the membrane structure in the final configuration. The resulting set of strings is obtained by concatenating either the labels of the membranes or the objects inside these membranes, in the order they are visited. If a membrane contains more than one object, then we consider all the possible permutations of these objects. When we collect the labels, we do not consider the skin membrane, which is always labelled by 0. This output mode is denoted either by *lab*, if we collect the labels, or by *obj* if we collect the objects.
- *Traveller traces.* We assume that the initial configuration contains a special object t, called the *traveller*, inside some membrane. The traveller t can be moved by using rules of type (b) or (c), but it cannot be modified by any rule. The resulting string is obtained as follows: initially we start with the empty string associated with the initial configuration, then whenever the object t crosses a membrane labelled by i, we add the symbol i at the rightmost side of the current string. This output mode is denoted by *traces*.
- *External mode.* The resulting set of strings is defined as follows: we start initially with an empty string outside of the membrane system; whenever an object is sent out of the skin membrane, we add such an object to the rightmost end of each current string. If some objects are sent out from the skin membrane at the same time, we consider the string formed by all the permutations of these objects. This output mode is denoted by *ext*.

We denote by $LOP_{m,n}(Op,l)$, with $Op \subseteq \{a,b,c,d,e,f,g\}$, $l \in \{obj, lab, traces, ext\}$, the family of languages generated by P systems with active membranes with at most m membranes in the initial configuration that use at most n different labels for the membranes (the cardinality of $K \cup \{0\}$ is at most n), that apply rules of the forms specified in Op, and has the output mode l. As usual, if the value of m, or the value of n, is not bounded, it is replaced by the symbol $*$. Moreover, when the rules of type $(e),(f)$, and (g) are allowed only for elementary membranes, the corresponding operations are denoted by e', f', and g'. Also, when the rules of type (b) and (c) are allowed to change the label of the membrane, the corresponding operations are denoted by b' and c'.

3.1 The power of membrane creation and membrane division

In this section, we present some results from [3]. The universality of P systems with membrane division and membrane dissolution with respect to the output modes *lab, obj* was given by the following theorem.

Theorem 1 $LOP_{1,*}(\{a,b,c,d,g\},v) = RE$, *for* $v \in \{lab, obj\}$.

A similar result holds for P systems with membrane division and membrane dissolution restricted to elementary membranes.

Theorem 2 $LOP_{2,*}(\{a,b,c,e,g'\},v) = RE$, *for* $v \in \{lab, obj\}$.

In the proofs of the above theroem, the final configuration has membrane structure of depth 2. So a "predefined" order among the membranes is necessary to get a suitable representation for the strings of a language. Such an approach does not work well in the case of *traces* and *external* output modes. The following theorem shows that such a problem can be avoided by considering the operation of *membrane duplication*.

Theorem 3 $LOP_{2,*}(\{a,b,c,f,g\},v) = RE$, *for* $v \in \{lab, obj, ext\}$ *and* $LOP_{3,*}(\{a,b,c,f,g\}, traces) = RE$.

The following cases were left open in [3]:
The computational power of

1. P systems with membrane creation and membrane dissolution (or membrane division and membrane dissolution) operating according to the external mode or the traces mode;

2. P systems without membrane dissolution operating according to any of the three identified output modes;

3. P systems with membrane creation and membrane dissolution restricted to elementary membranes.

We settle some of the above cases in the coming sections.

4 Universality with Membrane Creation and Dissolution

The following is a characterization of recursively enumerable languages by means of P systems with active membranes equipped with the membrane creation and dissolution operation restricted to elementary membranes operating according to all output modes.

Theorem 4 $LOP_{1,*}(\{a,b,c,d,g'\},v) = RE$, *for* $v \in \{lab, obj, ext\}$ *and* $LOP_{2,*}(\{a,b,c,d,g'\}, traces) = RE$.

Proof: The proof is based on the simulation of a register machine $M = (2, T, P, l_0, l_h)$. We construct a P system with active membranes that simulates the register machine M such that

$$\Pi = (V, K \cup \{0\}, [_0]_0, l_0', R)$$

where

$$\begin{aligned}
V &= T \cup \{a_1, a_2, b_1, b_2\} \cup \{l', l'' \mid l : (ADD(r), l', l'') \in P\} \\
&\cup \ \{l_0, l_0', l_0'', 1', 2', \$' \mid l_0 \text{ is the initial label of M } \} \\
&\cup \ \{l_i, l_i', l_i'', l_i''', l_i^{iv}, l', l'' \mid l : (SUB(i), l', l'', i = 1, 2\} \\
&\cup \ \{(a, l'), \overline{(a, l')} \mid l : (WRITE(a), l') \in P, a \in T\} \\
&\cup \ \{1', 2', \$'\} \\
K &= T \cup \{1, 2, 3, 4, \$\}
\end{aligned}$$

$$
\begin{aligned}
R &= \{[_0 l'_0 \rightarrow 1'2'\$' l''_0]_0, [_0 l''_0 \rightarrow l_0] \mid l_0 \text{ is the initial label of M}\} \\
&\cup \{[_0 1' \rightarrow [_1]_1]_0, [_0 2' \rightarrow [_2]_2]_0, [_0 \$' \rightarrow [_\$]_\$]_0\} \\
&\cup \{[_0 l \rightarrow b_i l']_0, [_0 l \rightarrow b_i l'']_0 \mid l : (ADD(i), l', l''), i = 1,2\} \\
&\cup \{b_i [_i]_i \rightarrow [_i a_i]_i \mid i = 1,2\} \\
&\cup \{l \rightarrow l_i, l_i [_i]_i \rightarrow [_i l_i]_i \mid l : (SUB(i), l', l'', i = 1,2\} \\
&\cup \{a_i [_3]_3 \rightarrow [_3 a_i]_3, [_3 l'_i \rightarrow l'']_3 \mid l : (SUB(i), l', l'', i = 1,2\} \\
&\cup \{[_3 a_i]_3 \rightarrow \lambda, [_3 l''_i \rightarrow l'''_i]_3, [_3 l'''_i]_3 \rightarrow l^{iv}_i \mid l : (SUB(i), l', l'', i = 1,2\} \\
&\cup \{[_i l'''_i]_i \rightarrow l', [_i l^{iv}_i]_i \rightarrow l'' \mid l : (SUB(i), l', l'', i = 1,2\} \\
&\cup \{[_0 l \rightarrow (a, l')]_0 \mid l : (WRITE(a), l')\} \\
&\cup \{(a, l')[_b]_b \rightarrow [_b (a, l')]_b \mid b \in T \cup \{\$\}\} \\
&\cup \{[_\$ (a, l')]_\$ \rightarrow \overline{(a, l')}, [_b \overline{(a, l')} \rightarrow [_a l' \$]_a]_b \mid b \in T\} \\
&\cup \{[_a \$ \rightarrow [_\$]_\$]_a \mid a \in T\} \\
&\cup \{l_h [_1]_1 \rightarrow [_1 l_h]_1, [_1 l_h]_1 \rightarrow l'_h, l'_h [_2]_2 \rightarrow [_2 l'_h]_2, [_2 l'_h]_2 \rightarrow l''_h\} \\
&\cup \{[_0 a_i \rightarrow \lambda]_0 \mid i = 1,2\} \\
&\cup \{l''_h [_4]_4 \rightarrow [_4 l''_h]_4, [_4 l''_h]_4 \rightarrow \lambda\}
\end{aligned}
$$

Let us see how the P system Π works. Initially, we have the configuration $[_0[_4 t]_4 l'_0]_0$. We apply the first 5 rules to produce the configuration $[_0 l_0 [_1]_1 [_2]_2 [_\$]_\$]_0$. The value of the two registers $i = 1,2$, are represented by the number of objects a_i inside the corresponding membrane i. The membrane labelled \$ is used to prepare an appropriate membrane structure where the output objects are supposed to be distributed according to a specific strategy.

The add instruction $l : (ADD(i), l', l'')$ is simulated as follows. We use the rule $l \rightarrow b_i l'$ or $l \rightarrow b_i l''$ to create an object b_i corresponding to the register i. Now the object b_i changes to a_i while entering inside membrane i.

In order to simulate a subtract instruction $l : (SUB(i), l', l'')$, we send the object l_i into the membrane i and then proceed in the following way: The object l_i creates a membrane with label 3 and an object l'_i. If the register i is not empty, then the object a_i will enter membrane 3 and dissolve it; otherwise the object l'''_i dissolves membrane 3 there by changing to l^{iv}_i. If the register i is not empty, then we have l'''_i in membrane i; otherwise l^{iv}_i. Now we will send l' or l'' to the skin membrane depending upon the presence of l'''_i or l^{iv}_i in membrane i respectively. This will end the simulation of the SUB instruction.

The simulation of the instruction $l : (WRITE(a), l')$ is done as follows. First we use the rule $l \rightarrow (a, l')$ in the skin membrane. The object (a, l') travels deep inside the nested membrane structure until it reaches the membrane $[_\$]_\$$. In membrane \$, the object (a, l') changes to $\overline{(a, l')}$ and dissolves the membrane. Now the object $\overline{(a, l')}$ will create a membrane labelled a which contains the objects l' and \$. The object l' moves toward the skin membrane whereas the object \$ will create a membrane $[_\$]_\$$ inside the membrane $[_a]_a$. The object l' starts the simulation of the instruction labelled l' after reaching the skin membrane.

The presence of object l_h in the skin membrane will start the clean-up process. It will remove both the membranes 1, 2 and the objects inside them. Finally the object l''_h dissolves membrane 4 which contains the object t.

At last, we have a configuration of the form $[_0 t [_{x_1} [_{x_2} \ldots [_{x_h} [_\$]_\$]_{x_h} \ldots]_{x_2}]_{x_1}]_0$ with $x_1 x_2 \ldots x_h \in L(M)$, for some $h \geq 1$. Now we move the traveller t by using rules of the form $t[_a]_a \rightarrow [_a t]_a$, with $a \in T$, and in this way we generates exactly the string $x_1 x_2 \ldots x_h \in L(M)$.

External mode: For this mode we consider a P systems Π whose initial configuration is $[_0 l_0']_0$, where l_0 is the starting label of the register machine M. We simulate the register machine M in the same way as described above for the traveller traces, and during the clean-up process the object l_h' changes to f and dissolves membrane 2 instead of changing to l_h''. Thus, we have a configuration $[_0 f [_{x_1} [_{x_2} \dots [_{x_h} [_\$]_\$]_{x_h} \dots]_{x_2}]_{x_1}]_0$, and we can generate the string $x_1 x_2 \dots x_h \in L(M)$ by using the following rules:

- $f[_a]_a \to [_a f']_a$ with $a \in T$
- $[_a f' \to af]_a$ with $a \in T$
- $[_b a]_b \to a[_b]_b$ with $a, b \in T$
- $[_0 a]_0 \to a[_0]_0$ with $a \in T$

By applying these rules, we can send the objects out of the skin membrane in the right order

We can easily modify the above system Π to obtain a final configuration of the form $[_0 [_{x_1} x_1 [_{x_2} x_2 \dots [_{x_h} x_h [_\$]_\$]_{x_h} \dots]_{x_2}]_{x_1}]_0$ for some $h \geq 1$, and $x_1 x_2 \dots x_h \in L(M)$. If we visit the tree associated with this membrane structure either by collecting the labels or by collecting the objects, then we get $x_1 x_2 \dots x_h \in L(M)$ in both cases. $\square$

Remark 2 *The universality of P systems with membrane division and membrane dissolution restricted to elementary membranes with respect to the traces and external output modes can be proved in a similar fashion provided the rules of type endocytosis were allowed. Because a combination of rules of type division and endocytosis can simulate rules of type creation.*

5 Universality with only Membrane Creation

A similar result holds for P systems that use only the membrane creation operation avoiding the operation membrane dissolution for all output modes except *traveller traces*. But we need the label changing feature for *in* type rules to obtain universality.

Theorem 5 *$LOP_{1,*}(\{a,b,c',d\},v) = RE$, for $v \in \{lab, obj, ext\}$.*

Proof: Let $G = (N,T,S,M,F)$, with $N = N_1 \cup N_2 \cup \{S, Z, \#\}$, be a matrix grammar with appearance checking in Z-binary normal form where the matrix of type 1 is $(S \to XA)$, the matrices of type 2 are labelled, in one to one manner, by $m_1, \dots, m_k$, and matrices of type 3 by $m_{k+1}, \dots, m_n$. We construct a P system with active membranes that simulates the matrix grammar G as follows:

$$\Pi = (V, K \cup \{0\}, [_0]_0, S, R)$$

where

$$\begin{aligned} V &= N_1 \cup N_2 \cup T \cup \{Z, \#\} \cup \{Y_i, Y_i' \mid 1 \leq i \leq n, Y \in N_1\} \\ &\cup \ \{Y_{i,B} \mid Y \in N_1, B \in N_2, 1 \leq i \leq k\} \cup \{Y_{i,\$} \mid Y \in N_1, 1 \leq i \leq k\} \\ K &= N_2 \cup T \cup \{\$\} \end{aligned}$$

$$\begin{array}{rcl}
R & = & \{[_0 S \to [_A X_A]_A]_0, [_A X_A \to [_\$ X'_A]_\$]_A \mid (S \to XA) \in M\} \\
& \cup & \{[_\$ X'_A]_\$ \to X'_A[_\$]_\$, [_A X'_A]_A \to X[_A]_A \mid (S \to XA) \in M\} \\
& \cup & \{[_0 X \to Y_i]_0 \mid X, Y \in N_1, 1 \leq i \leq n\} \\
& \cup & \{Y_i[_y]_y \to [_y Y_i]_y \mid y \in N_2 \cup T, y \neq A, m_i : (X \to Y, A \to x) \in M, 1 \leq i \leq k\} \\
& \cup & \{Y_i[_\$]_\$ \to [_\# \#]_\# \mid Y \in N_1, 1 \leq i \leq k\} \\
& \cup & \{Y_i[_A]_A \to [_a Y'_i]_a \mid m_i : (X \to Y, A \to a), 1 \leq i \leq k\} \\
& \cup & \{[_y Y'_i]_y \to Y'_i[_y]_y \mid y \in N_2 \cup T \cup \{\$\}, 1 \leq i \leq n\} \\
& \cup & \{[_0 Y'_i \to Y]_0 \mid Y \in N_1, 1 \leq i \leq n\} \\
& \cup & \{Y_i[_A]_A \to [_{a_1} Y_{i,a_2}]_{a_1} \mid m_i : (X \to Y, A \to a_1a_2), 1 \leq i \leq k\} \\
& \cup & \{Y_{i,B}[_C]_C \to [_B Y_{i,C}]_B \mid B, C \in N_2 \cup T \cup \{\$\}, 1 \leq i \leq k\} \\
& \cup & \{[_B Y_{i,\$} \to [_\$ Y'_i]_\$]_B \mid Y \in N_1, B \in N_2, 1 \leq i \leq k\} \\
& \cup & \{Y_i[_A]_A \to [_\# \#]_\#, Y_i[_\$]_\$ \to [_\$ Y'_i]_\$ \mid m_i : (X \to Y, A \to \#), k+1 \leq i \leq n\} \\
& \cup & \{[_\# \# \to \#]_\#, [_0 Z \to \lambda]_0\}
\end{array}$$

Initially, we have the configuration $[_0 S]_0$. We simulate the unique matrix of type 1 in G by applying the first 4 rules to produce the configuration $[_0 X[_A[_\$]_\$]_A]_0$. We need the membrane labelled by \$ in order to identify the end of the string.

Assume that we have a configuration of the form $[_0 X[_{x_1}[_{x_2}\cdots[_{x_h}[_\$]_\$]_{x_h}\cdots]_{x_2}]_{x_1}]_0$ after some steps, where $h \geq 1$, and $Xx_1x_2\ldots x_h$ is a sentential form of G, with $X \in N_1, x_i \in N_2 \cup T$. Now we apply the rule $[_0 X \to Y_i]_0$, for some $1 \leq i \leq n$. We have two cases according to the values of i.

Case 1: $1 \leq i \leq k$. In this case, we are simulating a matrix of type 2, i.e., $m_i : (X \to Y, A \to x)$. By using the rule $Y_i[_y]_y \to [_y Y_i]_y$, we move Y_i deeper inside the nested membranes. If there is no membrane labelled by A in the current configuration, then we use the rule $Y_i[_\$]_\$ \to [_\# \#]_\#$. The symbol # generates an infinite computation by means of the rule $[_\# \# \to \#]_\#$. If some membrane labelled A is present in the current configuration, then we have two cases.

Case a): $|x| = 1$, i.e., $x = a \in N_2 \cup T$.
In this case, we use the rule $Y_i[_A]_A \to [_a Y'_i]_a$. The above rule changes the object Y_i into Y'_i and also changes the label A into a. Now the object Y'_i travel towards the skin membrane. Once it reaches the skin, it becomes Y.

Case b): $|x| = 2$, i.e., $x = a_1a_2 \in N_2 \cup T$.
In this case, we use the rule $Y_i[_A]_A \to [_{a_1} Y_{i,a_2}]_{a_1}$. Here the object Y_i changes to Y_{i,a_2} and the label A changes to a_1. Further we use the rule $Y_{i,B}[_C]_C \to [_B Y_{i,C}]_B$ to move the object $Y_{i,B}$ towards the membrane labelled \$. While moving the objects $Y_{i,B}$, we change the labels of the membrane remembering the previous label in their second component. Once we got the object $Y_{i,\$}$ in the innermost membrane, we use it to create a membrane labelled \$ containing the object Y'_i. After this we move Y'_i towards the skin membrane. It will become Y once it reaches the skin membrane.

Case 2: $k+1 \leq i \leq n$. That is we are simulating a matrix of type 3 ($m_i : (X \to Y, A \to \#)$). In this case, we use the object Y_i to check for the presence of a membrane labelled A in the current configuration. If there exists a membrane labelled A, the object Y_i is moved inside by the rule $Y_i[_A]_A \to [_\# \#]_\#$. This will lead to an infinite computation that yields no result. Otherwise, the object Y_i becomes Y'_i after reaching the innermost membrane labelled by \$. Now we move the object Y'_i towards the skin membrane where it changes to Y.

Finally, we erase the symbol Z in the skin membrane, once it was introduced and the computation halts. Now by applying rules as in the previous theorem, we send out the objects in the right order. $\square$

6 Conclusion

This paper explores the idea of defining membrane systems that are able to build up a membrane structure that encodes some meaningful information proposed by [3]. We investigated the computational power of P systems with membrane creation and dissolution rules operating according to the external and traces mode. Also we proved the universality of P systems with membrane creation alone, but we allow the label changing feature for *in* type rules. At the moment, we are unable to characterize the power of P systems with active membranes equipped with membrane creation alone.

Acknowledgements. M. Gheorghe and R. Rama are grateful to the Royal Academy of Engineering which through a grant supporting *Exchanges with India and China* (2010), partially funded this research. MG has been also supported by CNCSIS grant no. 643/2009, *An integrated evolutionary approach to formal modelling and testing*. The authors would like to thank the anonymous reviewers for the comments made on an early version of this paper.

References

[1] B. Alberts et al., *Molecular Biology of the Cell*, Garland Science, 2002.

[2] A. Alhazov, R. Freund and A. Riscos-Núñez, One and Two Polarizations, Membrane Creation and Objects Complexity in P Systems, *Proc. of SYNASC 2005*, Timisoara, Romania, 385–394.

[3] F. Bernardini and M. Gheorghe, Languages Generated by P Systems with Active Membranes, *New Generation Computing*, 22(4), 2004, 311–329.

[4] J. Dassow and Gh. Păun, *Regulated Rewriting in Formal Language Theory*, Springer-Verlag, 1989.

[5] M.L. Minsky, *Computation: Finite and Infinite Machines*, Prentice Hall, 1967.

[6] Gh. Păun, *Membrane Computing: An Introduction*, Springer-Verlag, Berlin, 2002.

[7] R. Rama and H. Ramesh, On Generating Trees by P Systems with Active Membranes, *Proc. of SYNASC 2005*, Timisoara, Romania, 462–466.

[8] G. Rozenberg and A. Salomaa (eds.), *Handbook of Formal Languages* (3 volumes), Springer-Verlag, 1997.

Generalized Communicating P Systems Working in Fair Sequential Mode

Antoine Spicher
LACL, Département Informatique
Université Paris Est
61 av. gén. de Gaulle, 94010, Créteil, France
antoine.spicher@u-pec.fr

Sergey Verlan
LACL, Département Informatique
Université Paris Est
61 av. gén. de Gaulle, 94010, Créteil, France
verlan@u-pec.fr

In this article we consider a new derivation mode for generalized communicating P systems (GCPS) corresponding to the functioning of population protocols (PP) and based on the sequential derivation mode and a fairness condition. We show that PP can be seen as a particular variant of GCPS. We also consider a particular stochastic evolution satisfying the fairness condition and obtain that it corresponds to the run of a Gillespie's SSA. This permits to further describe the dynamics of GCPS by a system of ODEs when the population size goes to the infinity.

1 Introduction

The notion of a *generalized communicating P system* was introduced in [22], with the aim of providing a common generalization of various purely communicating models in the framework of P systems.

A generalized communicating P system, or a *GCPS* for short, corresponds to a hypergraph where each node is represented by a cell and each edge is represented by a rule. Every cell contains a multiset of objects which – by communication rules – may move between the cells. The form of a *communication rule* is $(a,i)(b,j) \to (a,k)(b,l)$ where a and b are objects and i,j,k,l are labels identifying the input and the output cells. Such a rule means that an object a from cell i and an object b from cell j move synchronously to cell k and cell l, respectively. In this respect, the model resembles the Petri Net formalism [18] where tokens from various input places come along together to fire a given transition and then fork out to destination places, see [22, 4] for more details.

Depending on the communication rules form, several *restrictions on communication rules* (modulo symmetry) can be introduced. Due to the simplicity of their rules, the generative power of such restricted systems is of particular interest and it has been studied in detail. In [22, 8, 7, 17] it was proved that eight of the possible nine restricted variants (with respect to the form of rules) are able to generate any recursively enumerable set of numbers; in the ninth case only finite sets of singletons can be obtained. Furthermore, these systems even with relatively small numbers of cells and simple underlying (hypergraph) architectures are able to achieve this generative power. In [7] a further restriction is introduced by considering that the alphabet of objects is a singleton (like in Petri Nets) and it is shown that the computational completeness can be achieved in four of the restricted variants.

Population protocols (PP) have been introduced in [1] (see [3] for a survey) as a model of sensor networks consisting of very limited mobile agents with no control over their own movement. A population protocol corresponds to a collection of anonymous agents, modelled by finite automata, that interact with one another to carry out computations, by updating their states, using some rules. Their computational power has been investigated under several hypotheses in most of the cases restricted to finite size populations. In particular, predicates stably computable in the original model have been characterized as those

© Antoine Spicher and Sergei Verlan
All the rights to the paper remain with the authors.

definable in Presburger arithmetic. The article [5] studies the convergence of PP when the population size goes to the infinity.

The evolution of a PP follows a particular fairness condition: an execution is *fair* if for all configurations C that appear infinitely often in the execution, if C is predecessor of a configuration C', then C' appears infinitely often in the execution. We consider such a condition in the case of GCPS systems and obtain a new derivation mode which we call *fair sequential mode* (fs-mode). We further study the dynamic behaviour of the system in this mode. Among several possible evolution strategies we consider a stochastic strategy that satisfies the fairness condition and we obtain that the evolution of the system corresponds to a run of the Gillespie stochastic simulation algorithm (SSA). Using the correspondence between SSA and ODEs (assuming mass-action kinetics) we show that the dynamics of the system can be represented by a system of ODEs when the population size goes to the infinity. We also consider the converse problem and we give sufficient conditions for a system of ODEs to be represented by a GCPS system working in concentration-depended stochastic implementation of the fs-mode. We consider several examples of GCPS simulating Lotka-Volterra (predator-prey) behaviour or computing approximations of algebraic numbers.

2 Background

In this section we recall some basic notions and notations used in membrane computing, formal language theory and computability theory. For further details and information the reader is referred to [16, 17, 19].

An alphabet is a finite non-empty set of symbols. For an alphabet V, we denote by V^* the set of all strings over V, including the empty string, λ. The *length* of the string $x \in V^*$ is the number of symbols which appear in x and it is denoted by $|x|$. The number of occurrences of a symbol $a \in V$ in $x \in V^*$ is denoted by $|x|_a$. If $x \in V^*$ and $U \subseteq V$, then we denote by $|x|_U$ the number of occurrences of symbols from U in x.

A finite multiset over V is a mapping $M : V \longrightarrow \mathbb{N}$; $M(a)$ is said to be the multiplicity of a in M ($\mathbb{N}$ denotes the set of non-negative integers.) A finite multiset M over an alphabet V can be represented by all permutations of a string $x = a_1^{M(a_1)} a_2^{M(a_2)} \dots a_n^{M(a_n)} \in V^*$, where $a_j \in V$, $1 \leq j \leq n$; x represents M in V^*. If no confusion arises, we also may use the customary set notation for denoting multisets. The size of a finite multiset M, represented by $x \in V^*$ is defined as $\Sigma_{a \in V} |x|_a$.

2.1 P Systems

Next we recall the basic definitions concerning generalized communicating P systems [22].

Definition 1. A *generalized communicating P system* (a *GCPS*) of degree n, where $n \geq 1$, is an $(n+4)$-tuple $\Pi = (O, E, w_1, \dots, w_n, R, h)$ where

1. O is an alphabet, called the *set of objects* of Π;
2. $E \subseteq O$; called the *set of environmental objects* of Π;
3. $w_i \in O^*$, $1 \leq i \leq n$, is the multiset of objects *initially associated with cell* i;
4. R is a finite set of *interaction rules* (or *communication rules*) of the form $(a,i)(b,j) \to (a,k)(b,l)$, where $a, b \in O$, $0 \leq i, j, k, l \leq n$, and if $i = 0$ and $j = 0$, then $\{a,b\} \cap (O \setminus E) \neq \emptyset$; *i.e.*, $a \notin E$ and/or $b \notin E$;
5. $h \in \{1, \dots, n\}$ is the *output cell.*

The system consists of n cells, labelled by natural numbers from 1 to n, which contain multisets of objects over O; initially cell i contains multiset w_i (the initial contents of cell i is w_i). We distinguish an additional special cell, labelled by 0, called the *environment*. The environment contains objects of E in an *infinite number of copies*.

The cells interact by means of the rules $(a,i)(b,j) \to (a,k)(b,l)$, with $a,b \in O$ and $0 \le i,j,k,l \le n$. As the result of the application of the rule, object a moves from cell i to cell k and b moves from cell j to cell l. If two objects from the environment move to some other cell or cells, then at least one of them must not appear in the environment in an infinite number of copies. Otherwise, an infinite number of objects can be imported in the system in one step.

A *configuration* of a GCPS Π, as above, is an $(n+1)$-tuple $(z_0,z_1,\dots,z_n)$ with $z_0 \in (O \setminus E)^*$ and $z_i \in O^*$, for all $1 \le i \le n$; z_0 is the multiset of objects present in the environment in a finite number of copies, whereas, for all $1 \le i \le n$, z_i is the multiset of objects present inside cell i. The *initial configuration of* Π is the tuple $(\lambda, w_1,\dots,w_n)$.

Given a multiset of rules $\mathscr{R}$ over R and a configuration $u = (z_0,z_1,\dots,z_n)$ of Π, we say that $\mathscr{R}$ is *applicable* to u if all its elements can be applied simultaneously to the objects of multisets $z_0,z_1,\dots,z_n$ such that every object is used by at most one rule. Then, for a configuration $u = (z_0,z_1,\dots,z_n)$ of Π, a new configuration $u' = (z'_0,z'_1,\dots,z'_n)$ is obtained by applying the rules of R in a non-deterministic maximally parallel manner: taking an applicable multiset of rules $\mathscr{R}$ over R such that the application of $\mathscr{R}$ results in configuration $u' = (z'_0,z'_1,\dots,z'_n)$ and there is no other applicable multiset of rules $\mathscr{R}'$ over R which properly contains $\mathscr{R}$.

It is also possible to replace the maximally parallel strategy of rule application by other strategies, called *derivation modes* (in the context of the present paper, the terms *mode* and *strategy* are used indifferently). A derivation mode lies in the heart of the semantics of P systems and it permits to specify which multiset among different possible applicable multisets of rules can be applied. When P systems were introduced, only the maximally parallel derivation mode was considered which states that corresponding multisets should be maximal, *i.e.*, non-extensible. With the appearance of the minimal parallel derivation mode [6] the concept of the derivation mode had to be precisely defined and [10] presents a framework that permits to easily define different derivation modes.

One application of a multiset of rules satisfying the conditions of a derivation mode represents a *transition* in Π from configuration u to configuration u'. A transition sequence is said to be a *successful generation* by Π if it starts with the initial configuration of Π and ends with a *halting configuration*, *i.e.*, with a configuration where no further transition step can be performed.

We say that Π *generates a non-negative integer n* if there is a successful generation by Π such that n is the size of the multiset of objects present inside the output cell in the halting configuration. The *set of non-negative integers generated* by a GCPS Π in this way is denoted by $N(\Pi)$. It is also possible to use GCPS as acceptors, in this case an input multiset is accepted if the system halts on it.

In [22] it is shown that GCPS are able to generate all recursively enumerable languages. Moreover this result can be obtained by using various restrictions on the type of rules (*i.e.* induced hypergraph structures), on the number of membranes and on the cardinality of the alphabet. We refer to [22, 8, 7] for more details.

If the cardinality of the alphabet O is equal to one, then we refer to the corresponding symbol as a token (denoted by $\bullet$). Hence, we assume that $O = \{\bullet\}$. We observe that such systems are similar to Petri Nets having a restricted topology. This is especially visible if a graphical notation is used. However, the maximal parallelism and the concept of the environment are specific to P systems, so we place this study in the latter framework. A converse study of P systems from the point of view of Petri Nets can be found in [11]. For more details on Petri Nets and membrane computing we also refer to [17].

In this article we shall consider the dynamics of the configuration of GCPS, so we are no more interested in computation (and halting evolutions).

2.2 Population Protocols

We give below the definition as it appears in [5]. A protocol is given by $(Q, \Sigma, \imath, \omega, \delta)$ with the following components. Q is a finite set of states. Σ is a finite set of input symbols. $\imath : \Sigma \to Q$ is the initial state mapping, and $\omega : Q \to \{0,1\}$ is the individual output function. $\delta \subseteq Q^4$ is a joint transition relation that describes how pairs of agents can interact. Relation δ is sometimes described by listing all possible interactions using the notation $(q_1,q_2) \to (q'_1,q'_2)$, or even the notation $q_1q_2 \to q'_1q'_2$, for $(q_1,q_2,q'_1,q'_2) \in \delta$ (with the convention that $(q_1,q_2) \to (q_1,q_2)$ when no rule is specified with (q_1,q_2) in the left hand side.)

Computations of a protocol proceed in the following way. The computation takes place among n agents, where $n \geq 2$. A configuration of the system can be described by a vector of all the agent's states. The state of each agent is an element of Q. Because agents with the same states are indistinguishable, each configuration can be summarized as an unordered multiset of states, and hence of elements of Q.

Each agent is given initially some input value from Σ: each agent's initial state is determined by applying $\imath$ to its input value. This determines the initial configuration of the population.

An execution of a protocol proceeds from the initial configuration by interactions between pairs of agents. Suppose that two agents in state q_1 and q_2 meet and have an interaction. They can change into state q'_1 and q'_2 if (q_1,q_2,q'_1,q'_2) is in the transition relation δ. If C and C' are two configurations, we write $C \to C'$ if C' can be obtained from C by a single interaction of two agents: this means that C contains two states q_1 and q_2 and C' is obtained by replacing q_1 and q_2 by q'_1 and q'_2 in C, where $(q_1,q_2,q'_1,q'_2) \in \delta$. An execution of the protocol is a (potentially infinite) sequence of configurations $C_0, C_1, C_2, \ldots$, where C_0 is an initial configuration and $C_i \to C_{i+1}$ for all $i \geq 0$. An execution is *fair* if for all configurations C that appears infinitely often in the execution, if $C \to C'$ for some configuration C', then C' appears infinitely often in the execution.

At any point during an execution, each agent's state determines its output at that time. If the agent is in state q, its output value is $\omega(q)$. The configuration output is 0 (respectively 1) if all the individual outputs are 0 (respectively 1). If the individual outputs are mixed 0s and 1s then the output of the configuration is undefined.

Let p be a predicate over multisets of elements of Σ. Predicate p can be considered as a function whose range is $\{0,1\}$ and whose domain is the collection of these multisets. The predicate is said to be computed by the protocol if, for every multiset I, and every fair execution that starts from the initial configuration corresponding to I, the output value of every agent eventually stabilizes to $p(I)$.

The following was proved in [1, 2]:

Theorem 1 ([1, 2])**.** *A predicate is computable in the population protocol model if and only if it is semilinear.*

Recall that semilinear sets are known to correspond to predicates on counts of input agents definable in first-order Presburger arithmetic [15].

2.3 Gillespie Algorithm

A usual abstraction in the simulation of biochemical systems consists in considering the system (e.g., a bacterium) as a homogeneous chemical solution where the reactions of the model are taking place. D.T. Gillespie has proposed in [12] an algorithm for producing the trajectories of such a chemical system by computing the *next reaction* and the *elapsed time* since last reaction occurred. Let μ be a chemical reaction. The probability that μ takes place during an infinitesimal time step is proportional to:

- c_μ, the *stochastic reaction constant*[1] of reaction μ;
- h^S_μ, the number of distinct molecular combinations that can activate reaction μ; it depends on the current chemical state S;
- $d\tau$, the length of the time interval.

Gillespie proved that the probability $P(\tau,\mu|S)d\tau$ that, being in a chemical state S, the next reaction will be of type μ and will occur in the time interval $(t+\tau, t+\tau+d\tau)$ is:

$$P(\tau,\mu|S)d\tau = a^S_\mu\, e^{-a^S_0 \tau} d\tau$$

where $a^S_\mu = c_\mu\, h^S_\mu$ is called the *propensity* of reaction μ, and $a^S_0 = \sum_\nu a^S_\nu$ is the combined propensity of all reactions.

This probability leads to the first straightforward Gillespie's *exact stochastic simulation algorithm* (SSA) called the *first reaction method.* From a chemical state S, it consists in choosing an elapsed time τ for each reaction μ according to the probability $P(\tau,\mu|S)$. The reaction with the lowest elapsed time is selected and applied on the system making its state evolve. A new probability distribution is then computed for this new state and the process is iterated.

The Gillespie's SSA gives a way to simulate a continuous-time Markov chain with the states corresponding to the states of the system and with transitions between states corresponding to a single occurrence of a reaction. The probability for a transition between two states S and S' corresponding to the application of rule μ is defined as a^S_μ / a^S_0. In the following, we drop the mention of the current state S in these notations.

3 Fair Sequential Derivation Mode

In this section we are interested in the relation between PP and GCPS. We show that in terms of structure PP and GCPS are quite similar, the main differences concern the environment and the derivation mode. We define a new *fair sequential* mode for GCPS and hence we are able to encode any PP in a GCPS w.r.t. their dynamics. We then remark that GCPS with stochastic and Gillespie-like strategies are part of this new class of GCPS and we propose their use for simulations of population behaviours.

It can be easily seen that both PP and GCPS are particular instances of multiset rewriting. Indeed, in both cases the underlying data structure is multiset (obtained in a direct way for PP and by attaching the indices of membranes to the objects in GCPS) and the evolution rules clearly correspond to multiset rewriting rules with both left hand and right hand sides of size two. So, the translation of a PP to a one-symbol GCPS can be easily done as follows. Given a PP with set of states Q (for convenience we suppose that $Q = \{1,\ldots,n\}$) and transition relation δ in an initial configuration C_0, the corresponding GCPS $\Pi = (O, E, w_1, \ldots, w_n, R, 1)$ can be defined as:

- $O = E = \{\bullet\}$,
- $w_q = \bullet^k$, $k = |C_0|_q$ for any $q \in Q$,
- $R = \{(\bullet, q_1)(\bullet, q_2) \to (\bullet, q'_1)(\bullet, q'_2) \mid q_1 q_2 \to q'_1 q'_2 \in \delta\}$.

[1]Evaluating the stochastic constants is one of the key issues in stochastic simulations of biochemical reactions.

The above system encodes each state q of PP by a token $\bullet$ present in membrane labelled by q. Since we are interested in the dynamics of the system, no output membrane is necessary. The above construction covers the core of PP and to obtain the complete equivalence encoding and decoding functions $\imath$ and ω shall be used in the same way.

It is not possible to do a similar encoding of GCPS with PP because PP always deal with finite multisets and GCPS can use the infinite multiset corresponding to the environment. However, any GCPS having no rule involving the environment can be translated to PP in a similar way.

We remark that the biggest difference between PP and GCPS is given by the evolution step, *i.e.*, by the derivation mode. For GCPS, mainly the maximally parallel derivation mode is investigated with several attempts to investigate asynchronous or minimally parallel derivation mode, see [17] for more details. The derivation mode of PP is very particular – it corresponds to a sequential strategy where only one rule is applied at each step, like in Petri Nets, but with an additional fairness condition.

We can consider such a strategy in GCPS case as well. More precisely we consider *fair* computations: a GCPS computation is *fair*, if for any configuration u that appears infinitely often in the computation, then any configuration u', such that $u \Rightarrow u'$ in sequential application, also appears infinitely often. We shall call such computational strategy a *fair sequential derivation mode* (shortly *fs-mode*).

From these considerations, it is trivial to observe that PP are similar to GCPS in fs-mode with only one symbol in the alphabet. If we consider an encoding function $\imath$ like for PP and the halting condition corresponding to the stabilization of the ω-image of the configuration, then as an immediate consequence of [1, 2], we obtain that any GCPS working in fs-mode and that does not have any rule involving the environment can only accept semilinear sets.

Conversely, we also obtain that any PP working in maximally parallel mode (*i.e.*, a maximally parallel number of interactions can happen at each step) are computationally complete if the number agents in some particular state q_0 is going to the infinity.

From now on, we only speak of PP in terms of their associated one-symbol GCPS in fs-mode.

3.1 FS-Mode and Stochastic Evolution

Although powerful the definition of the fairness remains obscure. Let try to clarify it. When assuming that the number of configurations is finite (this is the case for classical PP for example), the definition can be easily rephrased as follows: a computation $u_0 \Rightarrow u_1 \Rightarrow \dots$ is fair if

- there exists a non-negative integer N such that configuration u_N belongs to a terminal strongly connected component of the state graph[2]; and
- any state of this terminal strongly connected component appears infinitely often in the execution.

There are many possible evolutions of the system in the fs-mode. One example of such an evolution is to choose at each step a rule that leads to a configuration that either never was visited previously or was not visited for some time greater than k, $k > 0$ (if possible).

Among all possible evolutions, Markovian processes feature prominently since they respect the fairness condition (it is well-known that Markovian processes leave non-terminal strongly connected components with probability 1), they do not require any history or global knowledge on the state space, and they provide a modelling tool useful in many domains (like in the simulation of population behaviours or in distributed algorithmics). Such a Markovian process corresponds to a labeling of each state graph arrow $u \Rightarrow^p u'$ by a static probability p that only depends on configuration u. Here are two examples of such Markovian processes:

[2]In this directed graph, nodes correspond to the configurations u, and two nodes u and u' are directly linked if $u \Rightarrow u'$.

1. *Equiprobable evolutions*: $u \Rightarrow^{1/k_u} u'$ where k_u denotes the cardinality of the set $\{u' \mid u \Rightarrow u'\}$.
2. *Concentration-dependent evolutions*: $u \Rightarrow^{p_r} u'$ where r denotes the applied rule and p_r is proportional to h_r, the number of distinct combinations of tokens that activate r, with a proportionality coefficient that only depends on r. Assuming that $r = (\bullet, q_1)(\bullet, q_2) \to (\bullet, q'_1)(\bullet, q'_2)$, the number h_r is given by

$$h_r = \begin{cases} |u|_{q_1}|u|_{q_2} & \text{if } q_1 \neq 0, q_2 \neq 0, q_1 \neq q_2 \\ |u|_{q_1}(|u|_{q_1} - 1) & \text{if } q_1 \neq 0, q_1 = q_2 \\ |u|_{q_1} & \text{if } q_2 = 0 \\ |u|_{q_2} & \text{if } q_1 = 0 \end{cases} \tag{1}$$

The two last cases hold when the environment (containing an infinite number of tokens) is involved in the rule.

3.2 FS-Mode GCPS modelling Population Dynamics

Assuming that for a given rule proportionality coefficients are the same for all configurations, the concentration-dependent strategy directly corresponds to a run of the Gillespie's SSA. Thus, we advocate that GCPS in fs-mode provide a good theoretical tool for studying population behaviours.

A paradigmatic example illustrating how GCPS allows a well suited specification of population behaviours consists of the description of a process inspired by the Lotka-Volterra model.

The Lotka-Volterra Model. The Lotka-Volterra process was introduced by Lotka as a model of coupled auto-catalytic chemical reactions, and was investigated by Volterra as a model for studying an ecosystem of predators and preys [9]. This model specifies how two coupled populations (of chemicals or individuals) Y_1 (the preys) and Y_2 (the predators) behave. In [12], D.T. Gillespie proposes the study of this system derived from the following ODEs

$$\frac{dY_1}{dt} = (c_1 - c_2Y_2)Y_1 \qquad \frac{dY_2}{dt} = (c_2Y_1 - c_3)Y_2 \tag{2}$$

Equivalently, the following chemical reactions

$$Y_1 \xrightarrow{c_1} 2Y_1 \qquad Y_1 + Y_2 \xrightarrow{c_2} 2Y_2 \qquad Y_2 \xrightarrow{c_3} . \tag{3}$$

specify a model whose behaviour is described by ODEs system (2). The dynamics of these reactions is conveniently characterized using the predator-prey interpretation. The first rule states that a prey Y_1 reproduces. The second rule states that a predator Y_2 reproduces after feeding on prey Y_1. Finally, the last rule specifies that predators Y_2 die of natural causes. Coefficients c_i are the rates of the three reactions. The correspondence between the two models relies in the fact that the trajectories of the Gillespie's SSA tend to the solutions of the ODEs system given by the law of mass action on the reactions. This result is due to the particular application of the Kurtz's theorem [13] to chemical systems.

Lotka-Volterra GCPS Definition. The model above does not fill GCPS requirements since the first and last reactions are not pairwise interactions. We propose to extend reactions (3) by considering a *renewable* resource $\overline{X}$ for Y_1 as a third species[3]: the molecular level of X remains *constant* whatever is

[3]We use the same notation as in [12] to express that the food resource X is assumed renewable.

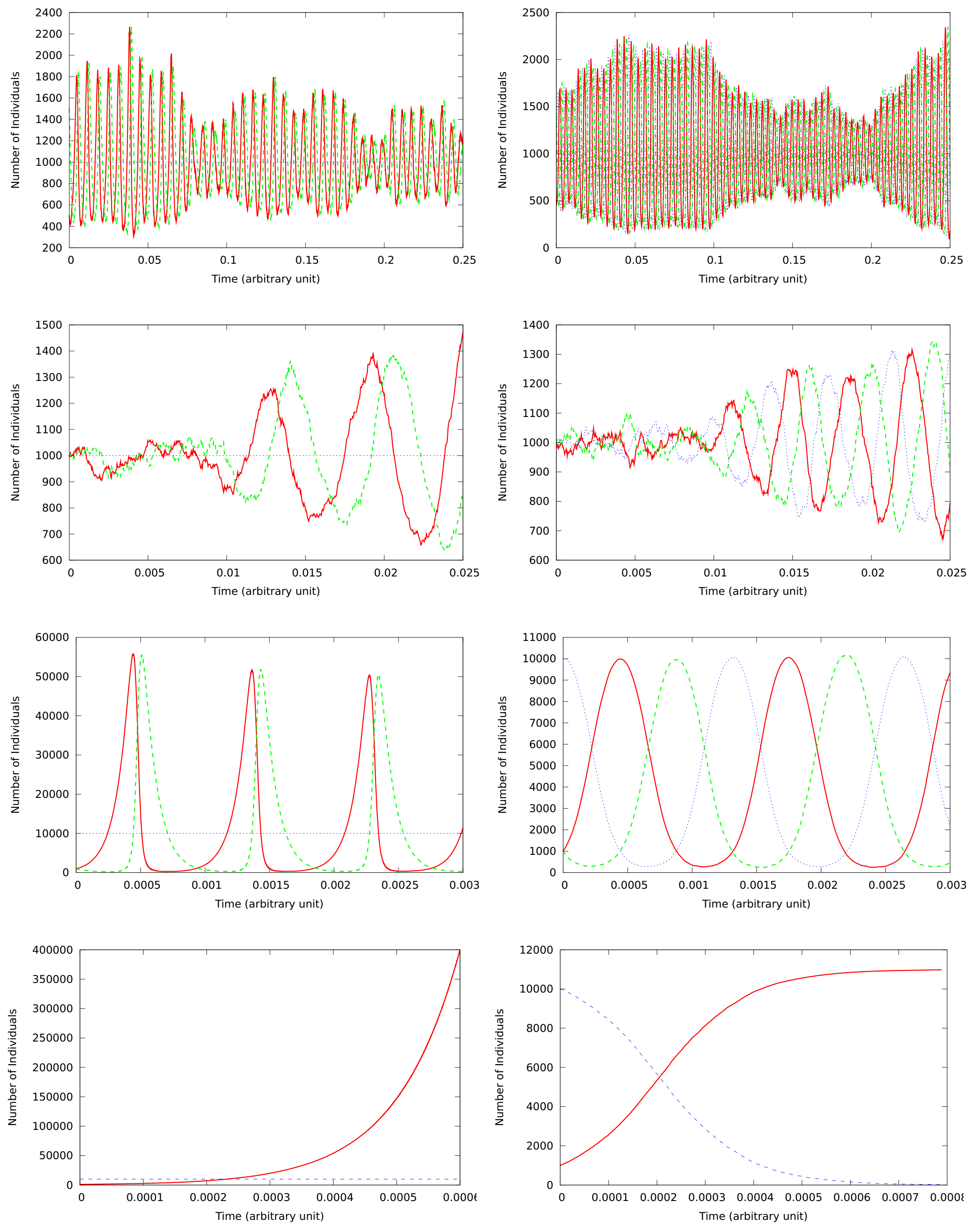


Figure 1: Runs of the Lotka-Volterra model with renewable (left column) and not-renewable (right column) resources for different initial states (kinetics rates equal 1): $Y_1 = 500, Y_2 = 1500, \overline{X} = X = 1000$ (first row), $Y_1 = Y_2 = \overline{X} = X = 1000$ (second row), $Y_1 = Y_2 = 1000, \overline{X} = X = 10000$ (third row), $Y_1 = 1000, Y_2 = 0, \overline{X} = X = 10000$ (fourth row). The solid red line represents preys, the dashed green line predators, and the blue dotted line resources. The two first rows show that both dynamics exhibit the same properties as presented in [12] (particularly, in second row, oscillations raise from an equilibrium initial state for the ODEs). The third row shows the difference in the dynamics when the resource size is ten times larger than the population size. The last row shows the difference in the dynamics when the predator population is empty. The simulations have been done using the general simulation language MGS (`http://mgs.spatial-computing.org`) that allows an easy implementation of all models of the present article [20, 14, 21].

its production or its consumption. The extended system of reactions is:

$$\overline{X} + Y_1 \xrightarrow{c'_1} 2Y_1 \qquad Y_1 + Y_2 \xrightarrow{c'_2} 2Y_2 \qquad Y_2 + \overline{X} \xrightarrow{c'_3} 2\overline{X} \tag{4}$$

The use of a pairwise interaction in the last reaction can be interpreted as a competition between the two predator behaviours: a predator in presence of preys eates and reproduces (second reaction); a predator in absence of prey (represented by the grass) dies (third reaction). Moreover, with the hypothesis that the number of $\overline{X}$ remains constant, the behaviour of this system is exactly described by ODEs (2) with $c_1 = \overline{X}c'_1$, $c_2 = c'_2$ and $c_3 = \overline{X}c'_3$. Thus, systems (3) and (4) are equivalent in terms of dynamics.

System (4) is only composed of pairwise interactions that satisfy condition 4 of Definition 1. Thus, it can be easily translated to a one-symbol GCPS, denoted Π_{LV}, working in fs-mode (in concentration-dependent implementation) with rules R:

$$\begin{array}{ll}
(\bullet,0)(\bullet,1) \to (\bullet,1)(\bullet,1) & \qquad (\bullet,1)(\bullet,0) \to (\bullet,1)(\bullet,1) \\
(\bullet,1)(\bullet,2) \to (\bullet,2)(\bullet,2) & \qquad (\bullet,2)(\bullet,1) \to (\bullet,2)(\bullet,2) \\
(\bullet,2)(\bullet,0) \to (\bullet,0)(\bullet,0) & \qquad (\bullet,0)(\bullet,2) \to (\bullet,0)(\bullet,0)
\end{array}$$

where membrane indices 0, 1 and 2 represent the environment (an infinite source of X), the preys Y_1 and the predators Y_2, respectively.

Let now consider the previously defined concentration-dependent evolution with probabilities $p_\mu = a_\mu/a_0$ for each $\mu \in R$ with the propensity function $a_\mu = c_\mu\, h_\mu$: c_μ is the rate of the corresponding reaction in (3) and h_μ is given by equation (1) accordingly to Π_{LV}. The reader is invited to pay attention that even if the environment is an *infinite* source of X (instead of a *constant* one), the dynamics are well taken into account: rules involving the environment have probabilities that do not depend on the environment size, see equation (1). For example, the propensity of the first reaction is given by $a_1 = c_1\, h_1 = c_1\, Y_1 = c'_1\, \overline{X}\, Y_1$ as expected w.r.t. reactions (4). In this respect, any computation of Π_{LV} represents a run of the Gillespie's SSA of reactions (4). As a consequence, Π_{LV} is an exact model of the original Lotka-Volterra system.

It has to be remarked that Π_{LV} cannot be described by any PP since the environment objects are involved in its definition. A possible specification of the Lotka-Volterra equations may be obtained within a PP by considering X as a *not-renewable* resource. Such a definition has been realized (taking reactions (4) and substituting $\overline{X}$ by X.) However, due to the limitation of resource, this system does not respect the dynamics of equation (2) anymore. For example, without any predators, a population of preys stabilizes in this model, while in the original model it grows exponentially. Figure 1 gives some examples of simulations of the Lotka-Volterra model considering renewable and not-renewable resources.

General Population Dynamics. It is possible to reverse the above method and to give a GCPS system whose population dynamics will correspond to some dynamics given by a system of ODEs, under the following conditions. Let us consider the ODEs system defined on set of variables $\{Y_1, \ldots, Y_N\}$ of the form

$$\frac{dY_i}{dt} = \sum_{j,k} a^i_{jk} Y_j Y_k - \sum_j (b_{ij} + b_{ji}) Y_i Y_j \tag{5}$$

where coefficients a^i_{jk} and b_{ij} satisfy the following conditions:

1. for all i, j, k, $a^i_{jk} \geq 0$ and $b_{ij} \geq 0$;
2. for all j, k such that $b_{jk} \neq 0$, there exists either one index i_0 such that $a^{i_0}_{jk} = 2b_{jk}$, or two distinct indices i_1 and i_2 such that $a^{i_1}_{jk} = a^{i_2}_{jk} = b_{jk}$; for any other index i, $i \neq i_0$ or $i \neq i_1$ and $i \neq i_2$, $a^i_{jk} = 0$.

The above conditions are sufficient to ensure that $\sum_i \frac{dY_i}{dt} = 0$. Then there exists a concentration-dependent fs-mode GCPS without rules involving the environment (*i.e.*, a PP) whose behaviour is exactly described by ODEs (5) when the population size goes to the infinity. Indeed, these equations correspond to the mass-action law of a set of rules such that for any j,k with $b_{jk} \neq 0$

$$(\bullet, j)(\bullet, k) \rightarrow^{b_{jk}} (\bullet, i_0)(\bullet, i_0) \qquad \text{or} \qquad (\bullet, j)(\bullet, k) \rightarrow^{b_{jk}} (\bullet, i_1)(\bullet, i_2)$$

according to the considered possibility of the above condition 2. The reader is invited to pay attention that these equations correspond to a wider range of dynamics than the dynamics of second-order chemical reactions with two products since they allow the specification of ordered interactions (e.g., involving a sender and a receiver as considered in the PP literature). This property also holds in PP and suggests that equations (5) exactly describe PP dynamics when the size of the populations tends to the infinity.

It is obvious that a more general class of population behaviours is captured by concentration-dependent fs-mode GCPS since they have not to be conservative thanks to the environment. Following the idea of equivalence between systems (3) and (4) in terms of dynamics, equations (5) can be extended with the introduction of a renewable variable $\overline{Y_0}$. This wider class of ODEs is supported by concentration-dependent fs-mode GCPS model.

4 Computational Properties

In this section, we focus on the original use of PP as a computational model of algebraic numbers proposed in [5]. This article investigates the case where the computation is independent of the initial contents of the system.

Using the GCPS terminology, the main idea of [5] is to consider the result of a computation as a ratio between the number of tokens in certain membrane and the total number of tokens (without taking care of the environment) when *the population size goes to the infinity* and when *the state of the system converges*. The proposed work relies on the definition of a particular strategy of execution of the PP: a step of execution consists in sampling uniformly and independently of the past two distinct tokens in the membrane and let them interact in a sequential mode. This strategy is fair since it corresponds to a Markov process. The authors of the aforementioned article studied the Markov chain associated with PP and proved its equivalence to some system of ODEs at the limit.

We remark that the same kind of result directly arises from considerations of Section 3.2 since this computational model is captured by one-symbol GCPS working in fs-mode with Gillespie concentration-dependent implementation. Indeed, the above execution strategy exactly corresponds to a Gillespie's SSA run where the stochastic constants equal 1 for all rules. Thus, the study of the model corresponds to the investigation of the sensibility of the associated ODEs system. Let us illustrate this point by considering the running example of [5]

$$\begin{aligned}
(\bullet, p)(\bullet, p) &\rightarrow (\bullet, p)(\bullet, m)\\
(\bullet, p)(\bullet, m) &\rightarrow (\bullet, p)(\bullet, p)\\
(\bullet, m)(\bullet, p) &\rightarrow (\bullet, p)(\bullet, p)\\
(\bullet, m)(\bullet, m) &\rightarrow (\bullet, p)(\bullet, m)
\end{aligned}$$

where symbols p and m identify two membranes. It has been shown that the ratio $\frac{p}{p+m}$, where p (resp. m) is the size of the membrane p (resp. m), converges to $\frac{1}{\sqrt{2}}$ when the population size goes to the infinity.

Accordingly to equations (5), we associate ODEs with this GCPS as follows

$$\frac{dY_p}{dt} = Y_m^2 + 2Y_pY_m - Y_p^2 \qquad\qquad \frac{dY_m}{dt} = -Y_m^2 - 2Y_pY_m + Y_p^2$$

The stable states of this system are obtained when the two equations vanish, that is, when either $Y_m = -(\sqrt{2}+1)Y_p$ or $Y_m = (\sqrt{2}-1)Y_p$. The first solution is incoherent since it involves a negative size of population. The second solution trivially leads to the expected result $\frac{Y_p}{Y_p+Y_m} = \frac{1}{\sqrt{2}}$.

5 Conclusions

In this article we investigated connections between population protocols and generalized communicating P systems. The two models share the same multiset structure and the same type of rules. Traditionally PP are used to study population dynamics in the context of distributed algorithmics while GCPS are investigated for the computational properties.

By incorporating the derivation mode from PP into GCPS framework we obtained a strict inclusion of PP in GCPS working in fs-mode. We then took a particular implementation of the fs-mode corresponding to a run of the Gillespie's SSA and we obtained that the dynamics of the systems can be described by the corresponding system of differential equations. Different questions then could be explored, like the investigation of the conditions ensuring that the system reaches a stable state regardless of its initial state or ensuring that a stable state is never reached for any initial configuration. GCPS are in this sense easier to handle than PP because of the environment that permits to easily simulate the equivalent of creation or degradation reactions. Section 3.2 also considers the converse problem of the construction of a GCPS system exhibiting a particular behaviour given by a systems of ODEs. It would be interesting to see if the given sufficient conditions are also necessary. A mathematical challenge resulting from Section 4 is whether for any algebraic number $x \in [0..1]$ there is a GCPS working in concentration-dependent evolution implementation of the fs-mode that converges to x.

We remark that the presented results hold only in the concentration-dependent implementation of the fs-mode. By taking an equiprobable implementation the results are completely different.

Since Petri Nets can be seen as multiset rewriting, it is clear that the results of this paper can be translated to this domain (for Petri Nets with specific type of rules and an additional fairness strategy).

We think that the fs-mode has interesting properties that should be further explored. As showed in the article, the fairness condition is in some sense similar to a stochastic evolution, so it could be preferable to consider this condition instead of a stochastic behaviour. Another interesting property of the proposed stochastic implementation is that Gillespie's SSA introduces an explicit continuous time and discrete events in the model, which do not appear in a GCPS description.

Acknowledgments. The authors would like to acknowledge the support of ANR project SynBioTIC.

References

[1] D. Angluin, J. Aspnes, Z. Diamadi, M. J. Fischer, and R. Peralta. Computation in Networks of Passively Mobile Finite-State Sensors. *Distributed Computing*, pages 235–253, Mar. 2006.

[2] D. Angluin, J. Aspnes, and D. Eisenstat. Stably Computable Predicates are Semilinear. In *PODC'06: Proceedings of the twenty-fifth annual ACM symposium on Principles of distributed computing*, pages 292–299, New York, NY, USA, 2006. ACM Press.

[3] J. Aspnes and E. Ruppert. An Introduction to Population Protocols. *Bulletin of the Europ. Assoc. for Theor. Comp. Sci.*, 93:98–117, Oct. 2007.

[4] F. Bernardini, M. Gheorghe, M. Margenstern, and S. Verlan. How to Synchronize the Activity of All Components of a P System? *International Journal of Foundations of Computer Science.*, 19(5):1183–1198, 2008.

[5] O. Bournez, P. Chassaing, J. Cohen, L. Gerin, and X. Koegler. On the Convergence of Population Protocols when Population Goes to Infinity. *Applied Mathematics and Computation*, 215(4):1340–1350, 2009.

[6] G. Ciobanu, L. Pan, G. Paun, and M. J. Pérez-Jiménez. P Systems with Minimal Parallelism. *Theor. Comput. Sci.*, 378(1):117–130, 2007.

[7] E. Csuhaj-Varjú, G. Vaszil, and S. Verlan. On Generalized Communicating P Systems with One Symbol. In M. Gheorghe, T. Hinze, and G. Paun, editors, *Proceedings of the Eleventh International Conference on Membrane Computing*, pages 137–154. Verlag ProBusiness Berlin, 2010.

[8] E. Csuhaj-Varjú and S. Verlan. On Generalized Communicating P Systems with Minimal Interaction Rules. *Theor. Comp. Sci.*, 412(1-2):124–135, 2011.

[9] L. Edelstein-Keshet. *Mathematical Models in Biology*. Random House, New York, 1988.

[10] R. Freund and S. Verlan. A Formal Framework for Static (Tissue) P Systems. In G. Eleftherakis, P. Kefalas, G. Păun, G. Rozenberg, and A. Salomaa, editors, *Membrane Computing, 8th International Workshop, WMC 2007, Thessaloniki, Greece, June 25-28, 2007 Revised Selected and Invited Papers*, volume 4860 of *LNCS*, pages 271–284. Springer, 2007.

[11] P. Frisco. *Computing with Cells*. Oxford University Press, 2009.

[12] D. T. Gillespie. Exact Stochastic Simulation of Coupled Chemical Reactions. *J. Phys. Chem.*, 81(25):2340–2361, 1977.

[13] T. Kurtz. A Limit Theorem for Perturbed Operator Semigroups with Applications to Random Evolutions. *J. Funct. Anal.*, 12:55–67, 1973.

[14] O. Michel, A. Spicher, and J.-L. Giavitto. Rule-Based Programming for Integrative Biological modelling – Application to the modelling of the Lambda Phage Genetic Switch. *Natural Computing*, 8(4):865–889, december 2009.

[15] M. Presburger. Uber die Vollstandig-keit eines Gewissen Systems der Arithmetik Ganzer Zahlen, in Welchemdie Addition als Einzige Operation Hervortritt. *Comptes rendus du I Congres des Mathematicians des Pays Slaves*, pages 92–101, 1929.

[16] G. Păun. *Membrane Computing. An Introduction*. Springer–Verlag, 2002.

[17] G. Păun, G. Rozenberg, and A. Salomaa. *The Oxford Handbook Of Membrane Computing*. Oxford University Press, 2009.

[18] W. Reisig. *Petri Nets. An Introduction*. Springer, 1985.

[19] G. Rozenberg and A. Salomaa. *Handbook of Formal Languages, 3 volumes*. Springer, 1997.

[20] A. Spicher, O. Michel, M. Cieslak, J.-L. Giavitto, and P. Prusinkiewicz. Stochastic P Systems and the Simulation of Biochemical Processes with Dynamic Compartments. *BioSystems*, 91(3):458–472, March 2008.

[21] A. Spicher, O. Michel, and J.-L. Giavitto. *Understanding the Dynamics of Biological Systems*, chapter Interaction-Based Simulations for Integrative Spatial Systems Biology, 195–231. Springer, 2011.

[22] S. Verlan, F. Bernardini, M. Gheorghe, and M. Margenstern. Generalized Communicating P Systems. *Theor. Comp. Sci.*, 404(1-2):170–184, 2008.

Abstracting Asynchronous Multi-Valued Networks: An Initial Investigation

L. Jason Steggles
Newcastle University, UK.
L.J.Steggles@ncl.ac.uk

Multi-valued networks provide a simple yet expressive qualitative state based modelling approach for biological systems. In this paper we develop an abstraction theory for asynchronous multi-valued network models that allows the state space of a model to be reduced while preserving key properties of the model. The abstraction theory therefore provides a mechanism for coping with the state space explosion problem and supports the analysis and comparison of multi-valued networks. We take as our starting point the abstraction theory for synchronous multi-valued networks which is based on the finite set of traces that represent the behaviour of such a model. The problem with extending this approach to the asynchronous case is that we can now have an infinite set of traces associated with a model making a simple trace inclusion test infeasible. To address this we develop a decision procedure for checking asynchronous abstractions based on using the finite state graph of an asynchronous multi-valued network to reason about its trace semantics. We illustrate the abstraction techniques developed by considering a detailed case study based on a multi-valued network model of the regulation of tryptophan biosynthesis in *Escherichia coli*.

1 Introduction

Multi-valued networks (MVNs) [25, 34, 35] are an expressive qualitative modelling approach for biological systems (for example, see [35, 7, 28, 3]). They extend the well–known *Boolean network* [17, 18] approach by allowing the state of each regulatory entity to be within a range of discrete values instead of just *true* or *false*. The state of each regulatory entity is influenced by other regulatory entities in the MVN and entities update their state using either a *synchronous update strategy* [18, 39] where all entities simultaneously update their state, or an *asynchronous update strategy* [33, 15, 36] where entities update their state independently using a non-deterministic approach.

While MVNs have shown their usefulness for modelling and understanding biological systems further work is still needed to strengthen the techniques and tools available for MVNs. One interesting area that needs developing is a theory for abstracting MVNs. Abstraction techniques allow a simpler model to be identified which can then be used to provide insight into the more complex original model. Such techniques are well–known in the formal verification community as a means of coping with the complexity of formal models (see for example [9, 6, 10, 13]). The main motivation behind developing such a theory for MVNs can be summarised as follows:

(1) The analysis of MVNs is limited by the well–known problem of state space explosion. Using abstraction is one useful approach which allows analysis results from a simpler approximate model to infer results about the original model.
(2) Often several MVNs are defined at different levels of abstraction when modelling a system. It is therefore clearly important to be able to formally relate these models using an appropriate theory.
(3) An abstraction theory would provide a basis for the step–wise refinement of MVNs.

Proceedings of 5th Workshop on Membrane Computing
and Biologically Inspired Process Calculi (MeCBIC 2011)
Pages 95–111, 2011.

© L. Jason Steggles
All the rights to the paper remain with the authors.

(4) Identifying an abstraction for a complex MVN provides a means of better visualising and understanding the behaviour an MVN, giving greater insight into the system being modelled.

The abstraction theory we present for asynchronous MVNs is based on extending the synchronous abstraction theory presented in [5]. We formulate a notion of what it means for an MVN to be correctly abstracted by a simpler MVN with the same network structure but smaller state space. The idea is to use an abstraction mapping to relate the reduced state space of an abstraction to the original MVN. An abstraction is then said to be correct if its set of traces is within the abstracted traces of the original MVN. This definition of abstraction represents an *under–approximation* [9, 24] since not all of the behaviour of the original MVN is guaranteed to have been captured within the abstraction. We show that this approach allows sound analysis inferences about positive reachability properties in the sense that any reachability result shown on an abstraction must hold on the original model. An important result of this is that it therefore follows that all attractors of an asynchronous abstraction correspond to attractors in the original MVN. Note that an alternative approach commonly used in abstraction is to use an *over–approximation* [9, 24, 10] in which false positives may occur. However, such an approach appears to be problematic for MVNs and we discuss this further in Section 3.

The non-deterministic nature of asynchronous MVNs mean that we encounter additional complications compared to the synchronous case; an asynchronous MVN can have an infinite set of traces which means that directly checking trace inclusion to check a proposed abstraction is infeasible. We overcome these difficulties by constructing a decision procedure for checking asynchronous abstractions that is based on the underlying finite state graph of an MVN. We introduce the idea of *step terms* which are used to denote possible ways to use sets of concrete states to represent abstract states. The decision procedure starts with the set of all possible step terms and then iteratively prunes the set until either a consistent abstract representation has been found or the set of remaining step terms is too small to make it feasible to continue. We provide a detailed proof that shows the decision procedure correctly identifies asynchronous abstractions and discuss the complexity of the decision procedure.

We illustrate the abstraction theory we develop by considering a case study based on modelling the regulatory network that controls the biosynthesis of tryptophan by the bacteria *E. coli* [29, 27]. Tryptophan is essential for the development of *E. coli* and its resource intensive synthesis is carefully controlled to ensure its production only occurs when an external source is not available. We investigate identifying asynchronous abstractions for an existing MVN model of this regulatory mechanism which was developed in [30].

The paper is organized as follows. In Section 2 we provide a brief overview of the MVN modelling framework and present a simple illustrative example. In Section 3 we formulate a notion of abstraction for asynchronous MVNs and consider the analysis properties that can be inferred from an abstraction. In Section 4 we present a decision procedure for checking asynchronous abstractions and provide a detailed proof of correctness for this procedure. In Section 5 we illustrate the theory and techniques developed by a case study based on modelling the regulatory network that controls the biosynthesis of tryptophan by *E. coli*. Finally, in Section 6 we present some concluding remarks and discuss related work.

2 Multi-valued Network Models

In this section, we introduce *multi-valued networks* (MVNs) [25, 34, 35], a qualitative modelling approach which extends the well-known *Boolean network* [17, 18] approach by allowing the state of each regulatory entity to be within a range of discrete values. MVNs can therefore discriminate between the strengths of different activated interactions, something which Boolean networks are unable to capture. MVNs have been extensively studied in circuit design (for example, see [25, 20]) and successfully applied to modelling biological systems (for example, see [35, 7, 28, 3]).

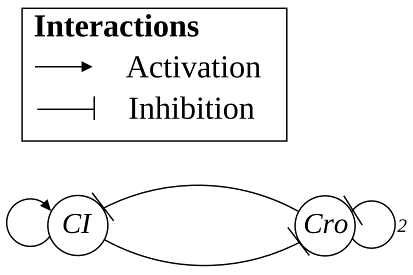
An MVN consists of a set of logically linked entities $G = \{g_1,\dots,g_k\}$ which regulate each other in a positive or negative way. Each entity g_i in an MVN has an associated set of discrete states $Y(g_i) = \{0,\dots,m_i\}$, for some $m_i \geq 1$, from which its current state is taken. Note that a Boolean network is therefore simply an MVN in which each entity g_i has a Boolean set of states $Y(g_i) = \{0,1\}$. Each entity g_i also has a neighbourhood $N(g_i) = \{g_{i_1},\dots,g_{i_{l(i)}}\}$ which is the set of all entities that can directly affect its state. A given entity g_i may or may not be a member of $N(g_i)$ and any entity in which $N(g_i) = \{\}$ is taken to be an input entity whose regulation is outside the current model. The behaviour of each entity g_i based on these neighbourhood interactions is formally defined by a logical next-state function f_{g_i} which calculates the next-state of g_i given the current states of the entities in its neighbourhood.

We can define an MVN more formally as follows.

Definition 1. An MVN *MV* is a four-tuple $MV = (G,Y,N,F)$ where:
i) $G = \{g_1,\dots,g_k\}$ is a non-empty, finite set of entities;
ii) $Y = (Y(g_1),\dots,Y(g_k))$ is a tuple of state sets, where each $Y(g_i) = \{0,\dots,m_i\}$, for some $m_i \geq 1$, is the state space for entity g_i;
iii) $N = (N(g_1),\dots,N(g_k))$ is a tuple of neighbourhoods, such that $N(g_i) \subseteq G$ is the neighbourhood of g_i; and
iv) $F = (f_{g_1},\dots,f_{g_k})$ is a tuple of next-state multi-valued functions, such that if $N(g_i) = \{g_{i_1},\dots,g_{i_n}\}$ then the function $f_{g_i} : Y(g_{i_1}) \times \cdots \times Y(g_{i_n}) \to Y(g_i)$ defines the next state of g_i. □

Consider the following simple example *PL2* of an MVN defined in Figure 1 which models the core regulatory mechanism for the *lysis–lysogeny switch* [34, 23] in the bacteriophage λ (this model is taken from [32]). It consists of two entities *CI* and *Cro*, defined such that $Y(CI) = \{0,1\}$ and $Y(Cro) = \{0,1,2\}$. The next-state functions for each entity are defined using the state transition tables presented in Figure 1.(b) (where $[g_i]$ is used to denote the next state of entity g_i). We can summarise the interactions as follows: entity *Cro* inhibits the expression of *CI* and at higher levels of expression, also inhibits itself; entity *CI* inhibits the expression of *Cro* while promoting its own expression.

In the sequel, let $MV = (G,Y,N,F)$ be an arbitrary MVN. In a slight abuse of notation we let $g_i \in MV$ represent that $g_i \in G$ is an entity in *MV*.



(a) Network structure

CI	*Cro*	$[CI]$	$[Cro]$
0	0	1	1
0	1	0	2
0	2	0	1
1	0	1	0
1	1	0	0
1	2	0	1

(b) State transition tables

Figure 1: The MVN model *PL2* of the core regulatory mechanism for the lysis-lysogeny switch in bacteriophage λ (taken from [32]).

A *global state* of an MVN MV with k entities is represented by a tuple of states $(s_1,\dots,s_k)$, where $s_i \in Y(g_i)$ represents the state of entity $g_i \in MV$. As a notational convenience we often use $s_1 \dots s_k$ to represent a global state $(s_1,\dots,s_k)$. When the current state of an MVN is clear from the context we let g_i denote both the name of an entity and its corresponding current state. The *global state space* of an MVN MV, denoted S_{MV}, is the set of all possible global states $S_{MV} = Y(g_1) \times \cdots \times Y(g_k)$.

The state of an MVN can be updated either *synchronously* (see [18, 39]), where the state of all entities is updated simultaneously in a single update step, or *asynchronously*[1] (see [33, 15]), where entities update their state independently. We define these update strategies more formally as follows:

Definition 2.
1) *Synchronous Update*: Given two states $S_1, S_2 \in S_{MV}$, we let $S_1 \xrightarrow{Syn} S_2$ represent a *synchronous update step* such that S_2 is the state that results from simultaneously updating the state of each entity g_i using its next-state function f_{g_i} and the appropriate states from S_1 as indicated by the neighbourhood $N(g_i)$.

2) *Asynchronous Update*: For any $g_i \in MV$ and any state $S \in S_{MV}$ we let $[S]^{g_i}$ denote the global state that results by updating the state of g_i in S using f_{g_i}. Define the global state function $next^{MV} : S_{MV} \rightarrow \mathscr{P}(S_{MV})$ on any state $S \in S_{MV}$ by

$$next^{MV}(S) = \{[S]^{g_i} \mid g_i \in MV \textit{ and } [S]^{g_i} \neq S\}$$

Given a state $S_1 \in S_{MV}$ and $S_2 \in next^{MV}(S_1)$, we let $S_1 \xrightarrow{Asy} S_2$ represent an *asynchronous update step*. □

Note that given the above definition, only asynchronous update steps that result in a change in the current state are considered (see [15]).

Continuing with our example, consider the global state 12 for *PL2* (see Figure 1) in which *CI* has state 1 and *Cro* has state 2. Then $12 \xrightarrow{Syn} 01$ is a single synchronous update step on this state resulting in the new state 11. Considering an asynchronous update, we have $next^{MV}(12) = \{02,\ 11\}$ and $12 \xrightarrow{Asy} 02$ and $12 \xrightarrow{Asy} 11$ are valid asynchronous update steps.

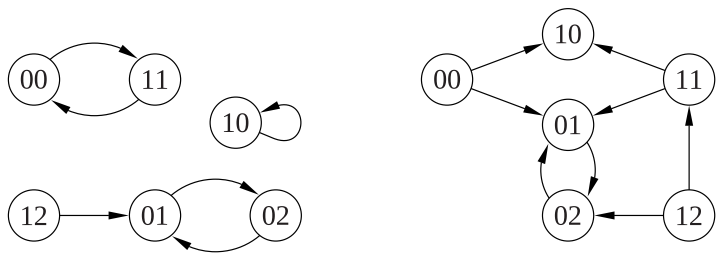
The sequence of update steps from an initial global state through S_{MV} is called a *trace*. In the case of the synchronous update semantics such traces are deterministic and infinite. Given that the global state space is finite, this implies that a synchronous trace must eventually enter a cycle, known formally as an *attractor cycle* [18, 35].

Definition 3. A *synchronous trace* σ is a list of global states $\sigma = \langle S_0, S_1, S_2, \dots \rangle$, where $S_i \xrightarrow{Syn} S_{i+1}$, for $i \geq 0$. □

The set of all synchronous traces, denoted $Tr^S(MV)$, therefore completely characterizes the behaviour of an MVN model under the synchronous semantics and is referred to as the *synchronous trace semantics* of MV. Note that we have one synchronous trace for each possible initial state and so the set of synchronous traces is always finite (see [18, 39]).

In the asynchronous case, traces are non-deterministic and can be finite or infinite. A single initial state can have an infinite number of possible asynchronous traces starting from it and thus in the asynchronous case there can be infinite number of traces.

[1]Note that different variations of the asynchronous semantics have been considered in the literature (see for example [26]) but that we focus on the one most commonly used for MVNs.



(a) Synchronous (b) Asynchronous

Figure 2: The (a) synchronous and (b) asynchronous state graphs for *PL2*.

Definition 4. An *asynchronous trace* σ is either:
i) a finite sequence of global states $\sigma = \langle S_0, S_1, \ldots, S_n \rangle$, where $S_i \xrightarrow{Asy} S_{i+1}$, for $i = 0, \ldots, n-1$, and $next^{MV}(S_n) = \{\}$.
ii) an infinite sequence of global states $\sigma = \langle S_0, S_1, S_2, \ldots \rangle$, where $S_i \xrightarrow{Asy} S_{i+1}$, for $i \geq 0$. □

The set of all asynchronous traces, denoted $Tr^A(MV)$, therefore completely characterizes the behaviour of an MVN model under the asynchronous semantics and is referred to as the *asynchronous trace semantics* of MV. Any state $S \in S_{MV}$ which cannot be asynchronously updated, i.e. $next^{MV}(S) = \{\}$, is referred to as a *point attractor* [34].

In our running example, *PL2* has a state space of size $|S_{PL2}| = 6$ and has the following (finite in this case) set of asynchronous traces:

$$\begin{array}{ll}
\langle 00, 01, 02, 01, 02, \ldots \rangle & \langle 10 \rangle \\
\langle 00, 10 \rangle & \langle 11, 01, 02, 01, 02, \ldots \rangle \\
\langle 01, 02, 01, 02, \ldots \rangle & \langle 11, 10 \rangle \\
\langle 02, 01, 02, 01, \ldots \rangle & \langle 12, 02, 01, 02, 01, \ldots \rangle
\end{array}$$

From the above traces it is clear that state 10 is a point attractor for *PL2*.

The behaviour of an MVN under the synchronous or asynchronous trace semantics can be represented by a state graph (for example, see [36]) in which the nodes are the global states and the edges are precisely the update steps allowed. We let $SG^S(MV) = (S_{MV}, \xrightarrow{Syn})$ and $SG^A(MV) = (S_{MV}, \xrightarrow{Asy})$ denote the corresponding state graphs under the synchronous and asynchronous trace semantics.

The synchronous and asynchronous state graphs for *PL2* are presented in Figure 2.

When analysing the behaviour of an MVN it is important to consider its attractors which can represent important biological phenomena, such as different cellular types like proliferation, apoptosis and differentiation [16]. In the synchronous case all traces are infinite and so must lead to a cyclic sequence of states which are taken as an *attractor* [18, 35, 39]. As an example, consider *PL2* (see Figure 2.(a)) which has the point attractor $10 \rightarrow 10$; and attractors $00 \xrightarrow{Syn} 11 \xrightarrow{Syn} 00$ and $01 \xrightarrow{Syn} 02 \xrightarrow{Syn} 01$ of period 2. In the asynchronous case we have *point attractors* which are states that cannot be updated and also the strongly connected components in an MVN's asynchronous state graph are considered to be *attractors* [36]. Again, considering *PL2* (see Figure 2.(b)) we can see that in the asynchronous case it has two point attractors, 01 and 10, and one attractor $01 \xrightarrow{Asy} 02 \xrightarrow{Asy} 01$.

3 Asynchronous Abstractions

In this section we consider developing a notion of abstraction for asynchronous MVNs. The idea is to formulate what it means for an MVN to be correctly abstracted by a simpler MVN with the same network structure but smaller state space. We take as our starting point the abstraction techniques developed for synchronous MVNs [5] and investigate extending these to the asynchronous case. We show that our approach allows sound analysis inferences about positive reachability properties and that all attractors of an asynchronous abstraction correspond to attractors in the original MVN.

We begin by recalling the notion of a state mapping and abstraction mapping [5] used to reduce an entity's state space.

Definition 5. Let MV be an MVN and let $g_i \in MV$ be an entity such that $Y(g_i) = \{0,\ldots,m\}$ for some $m > 1$. Then a *state mapping* $\phi(g_i)$ for entity g_i is a surjective mapping $\phi(g_i) : \{0,\ldots,m\} \to \{0,\ldots,n\}$, where $0 < n < m$. □

The state mapping must be surjective to ensure that all states in the new reduced state space are used. From a biological viewpoint it may also be reasonable to further restrict the state mappings considered, for example, only considering those mappings which are order-preserving. Note we only consider state mappings with a codomain larger than one, since a singular state entity does not appear to be of biological interest.

As an example, consider entity $Cro \in PL2$ (see Figure 1) which has the state space $Y(Cro) = \{0,1,2\}$. It is only meaningful to simplify Cro to a Boolean entity and so one possible state mapping to achieve this would be:

$$\phi(Cro) = \{0 \mapsto 0, 1 \mapsto 1, 2 \mapsto 1\},$$

which maps state 0 to 0 and merges states 1 and 2 into a single state 1.

In order to be able to simplify several entities at the same time during the abstraction process we introduce the notion of a family of state mappings.

Definition 6. Let $MV = (G,Y,N,F)$ be an MVN with entities $G = \{g_1,\ldots,g_k\}$. Then an *abstraction mapping* $\phi = \langle\phi(g_1),\ldots,\phi(g_k)\rangle$ for MV is a family of mappings such that for each $1 \leq i \leq k$ we have $\phi(g_i)$ is either a state mapping for entity g_i or is the identity mapping $I_{g_i} : Y(g_i) \to Y(g_i)$ where $I_{g_i}(s) = s$, for all $s \in Y(g_i)$. Furthermore, for ϕ to be useful we normally insist that at least one of the mappings $\phi(g_i)$ is a state mapping. □

Note in the sequel given a state mapping $\phi(g_i)$ we let it denote both itself and the corresponding abstraction mapping containing only the single state mapping $\phi(g_i)$.

An abstraction mapping ϕ can be used to abstract an asynchronous trace (see Definition 4) using a similar approach to that detailed for synchronous traces [5]. We begin by defining how an abstraction mapping can be lifted to a global state.

Definition 7. Let $\phi = \langle\phi(g_1)\ldots\phi(g_k)\rangle$ be an abstraction mapping for MV. Then ϕ can be used to abstract a global state $s_1\ldots s_k \in S_{MV}$ by applying it pointwise, i.e. $\phi(s_1\ldots s_k) = \phi(g_1)(s_1)\ldots\phi(g_k)(s_k)$. □

We can apply an abstraction mapping ϕ to an asynchronous trace $\sigma \in Tr^A(MV)$ by applying ϕ to each global state in the trace in the obvious way and and then merging consecutive identical states. Note that removing consecutive identical states is needed since by the definition of an asynchronous trace (see Definition 4) each asynchronous update rule must result in a new global state, i.e. the state of an entity has to change in order for a state transition to occur.

Definition 8. Let $\phi = \langle \phi(g_1) \dots \phi(g_k) \rangle$ be an abstraction mapping for MV and let $\sigma \in Tr^A(MV)$ be either a finite $\sigma = \langle S_0, S_1, \dots, S_n \rangle$ or infinite $\sigma = \langle S_0, S_1, S_2, \dots \rangle$ asynchronous trace. Then $\phi(\sigma)$ is the abstracted trace that results by

i) First apply the abstraction mapping to each state in σ, i.e. in the finite case $\langle \phi(S_0), \phi(S_1), \dots, \phi(S_n) \rangle$ or in the infinite case $\langle \phi(S_0), \phi(S_1), \phi(S_2), \dots \rangle$.
ii) Next merge consecutive identical global states in the trace into a single global state to ensure that no two consecutive states are identical in the resulting abstracted trace, i.e. suppose the result is an infinite trace $\langle \phi(S_0), \phi(S_1), \phi(S_2), \dots \rangle$ then we know that for $i \in \mathbf{N}$ we have $\phi(S_i) \neq \phi(S_{i+1})$. □

We let $\phi(Tr^A(MV)) = \{\phi(\sigma) \mid \sigma \in Tr^A(MV)\}$ denote the set of abstracted traces.

As an example, consider applying the abstraction mapping $\phi(Cro) = \{0 \mapsto 0, 1 \mapsto 1, 2 \mapsto 1\}$ to the *PL2* asynchronous trace $\langle 00, 01, 02, 01, 02, \dots \rangle$. Part i) of Definition 8 above results in the trace $\langle 00, 01, 01, 01, 01, \dots \rangle$; we now merge identical consecutive states to derive the abstracted trace $\langle 00, 01 \rangle$. It is interesting to note that abstracting an infinite trace can result in a finite abstracted trace, as above. The intuition here is that a cyclic set of states have been abstracted to a single point. The complete set of abstracted asynchronous traces of *PL2* using $\phi(Cro)$ are given below:

$$\begin{array}{lcl}
\phi(Cro)(\langle 00, 01, 02, 01, 02, \dots \rangle) & = & \langle 00, 01, \rangle \\
\phi(Cro)(\langle 00, 10 \rangle) & = & \langle 00, 10 \rangle \\
\phi(Cro)(\langle 01, 02, 01, 02, \dots \rangle) & = & \langle 01 \rangle \\
\phi(Cro)(\langle 02, 01, 02, 01, \dots \rangle) & = & \langle 01 \rangle \\
\phi(Cro)(\langle 10 \rangle) & = & \langle 10 \rangle \\
\phi(Cro)(\langle 11, 01, 02, 01, 02, \dots \rangle) & = & \langle 11, 01 \rangle \\
\phi(Cro)(\langle 11, 10 \rangle) & = & \langle 11, 10 \rangle \\
\phi(Cro)(\langle 12, 02, 01, 02, 01, \dots \rangle) & = & \langle 11, 01 \rangle
\end{array}$$

The definition of an asynchronous abstraction is based on its trace semantics and follows along similar lines to that for the synchronous case [5]. We say an asynchronous abstraction is correct if its set of traces is within the abstracted traces of the original MVN. This definition of abstraction represents an *under–approximation* [] since not all of the behaviour of the original MVN is guaranteed to have been captured within the abstraction (we discuss the implications of this below).

Definition 9. Let $MV_1 = (G_1, Y_1, N_1, F_1)$ and $MV_2 = (G_2, Y_2, N_2, F_2)$ be two MVNs with the same structure, i.e. $G_1 = G_2$ and $N_1(g_i) = N_2(g_i)$, for all $g_i \in MV_1$. Let ϕ be an abstraction mapping from MV_2 to MV_1. Then we say that MV_1 *asynchronously abstracts* MV_2 *under* ϕ, denoted $MV_1 \lhd_A^{\phi} MV_2$, if, and only if, $Tr^A(MV_1) \subseteq \phi(Tr^A(MV_2))$. □

As an abstraction example, consider the MVN *APL2* defined in Figure 3 which has the same structure as *PL2* (see Figure 1) but is a Boolean model. Then given the abstraction mapping $\phi(Cro) = \{0 \mapsto 0, 1 \mapsto 1, 2 \mapsto 1\}$ we can see that $Tr^A(APL2) \subseteq \phi(Cro)(Tr^A(PL2))$ holds and so *APL2* is an abstraction of *PL2*, i.e. $APL2 \lhd_A^{\phi(Cro)} PL2$ holds. Note that *APL2* has two point attractors: 01 and 10 which correspond

CI	Cro	$[CI]$	$[Cro]$
0	0	1	1
0	1	0	1
1	0	1	0
1	1	0	0

$\langle 00,01\rangle$	$\langle 10\rangle$
$\langle 00,10\rangle$	$\langle 11,01\rangle$
$\langle 01\rangle$	$\langle 11,10\rangle$

Figure 3: State transition tables defining *APL2* and associated asynchronous trace semantics $Tr^A(APL2)$.

to the two attractors associated with *PL2* (see Figure 2.(b)) and thus, *APL2* can bee seen to be a good approximation of the behaviour of *PL2*.

Recall that one of the original motivations for developing an abstraction theory was to aid the analysis of complex MVNs. It is therefore important to consider what properties of an asynchronous MVN can be inferred from an abstraction MVN. We consider reachability and the existence of attractors since these are the main properties that are considered when analysing an MVN.

Theorem 10. Let $MV_1 \lhd_A^\phi MV_2$ and let $S_1, S_2 \in S_{MV_1}$. If S_2 is reachable from S_1 in MV_1 then there must exist states $S_1', S_2' \in S_{MV_2}$ such that $\phi(S_1') = S_1$, $\phi(S_2') = S_2$, and S_2' is reachable from S_1' in MV_2.

Proof. Since S_2 is reachable from S_1 there must exist a trace $\sigma \in Tr^A(MV_1)$ which begins with state S_1 and which contains state S_2. From Definition 9, we know that $Tr^A(MV_1) \subseteq \phi(Tr^A(MV_2))$ must hold. Therefore there must exist a trace $\sigma' \in Tr^A(MV_2)$ such that $\phi(\sigma') = \sigma$. From this it is straightforward to see that there must exist the required states S_1' and S_2' in σ' such that $\phi(S_1') = S_1$, $\phi(S_2') = S_2$, and S_2' is reachable from S_1'. □

The above theorem indicates that inferring reachability properties from an abstraction is sound but not complete [13]. The implications of this can be summarised as follows: (i) If one state is reachable from another in an abstraction then a corresponding reachability property must hold in the original model; (ii) However, if one state is not reachable from another in an abstraction then a corresponding reachability property in the original MVN may or may not hold and more analysis will be required. This relates to the fact that our notion of abstraction represents an *under–approximation* [9, 24] of the original model. The alternative approach would be to use an *over–approximation* abstraction model [9, 24, 10] in which false positives can arise and need to be dealt with. It turns out that an over–approximation approach is not well suited to MVNs given that our goal is to find an abstraction model that is a well–defined MVN. To illustrate the potential problems, consider what happens if a point attractor is identified to a non–attractor state by an abstraction mapping. In this case no over–approximation abstraction can exist since such an MVN would need to contain a state that was both a point attractor and also had a successor state. Thus the approach taken here of using an under–approximation appears to be the appropriate approach to use.

Note that a consequence of the above is that all attractors in an abstraction must have corresponding attractors in the original MVN.

Corollary 11. If $MV_1 \lhd_A^\phi MV_2$ then all attractors of MV_1 must represent attractors in MV_2.

Proof. Follows directly from the definition of an attractor and Theorem 10. □

4 A Decision Procedure for Asynchronous Abstractions

Given we have now formulated a definition of an asynchronous abstraction we are now interested in defining a procedure for checking whether a proposed abstraction MV_1 is an asynchronous abstraction of an MVN MV_2. In the synchronous case the approach taken was to simply check that each trace $\sigma \in Tr^S(MV_1)$ was contained within the set of abstracted traces $\phi(Tr^S(MV_2))$. However, in the asynchronous case both sets of traces $Tr^A(MV_1)$ and $\phi(Tr^A(MV_2))$ may be infinite and so such a simple set inclusion check is not feasible. Instead we propose a decision procedure based on using the state graphs that summarise the behaviour of an asynchronous MVN. The idea is to consider all sets of states and associated edges that can be used to model an abstract state. We then iterate through these removing those state sets which can not be represented given the current allowable state sets. If at any point we no longer have any state sets remaining for a particular abstract state then we have shown the abstraction is not valid and we terminate the decision procedure. If, on the other hand, we reach a point at which no more state sets can be removed then we know the abstraction must be valid and we can again terminate the procedure.

In the sequel let MV_1 and MV_2 be MVNs with the same structure and let ϕ be an abstraction mapping from MV_2 to MV_1.

In order to define a decision procedure `checkAsynAbs(`MV_1`,`MV_2`,`ϕ`)` for checking if MV_1 is an asynchronous abstraction under ϕ of MV_2 we begin by formulating some preliminary concepts.

i) *Representing abstract states*: Let $S \in Y(MV_1)$ then we define

$$\phi^{-1}(S) = \{S' \mid S' \in Y(MV_2),\ \phi(S') = S\}$$

to be the set of all states in MV_2 that can represent the abstract state S.

ii) *Set of identical consecutive states*: For any state $S' \in Y(MV_2)$ we define the set $[S']^\phi$ of all *consecutive* reachable states from S' that have the same abstract state $\phi(S')$. Define $[S']^\phi = \bigcup_{i \in \mathbf{N}} [S']^\phi_i$, where $[S']^\phi_i$ is defined recursively: $[S']^\phi_0 = \{S'\}$ and

$$[S']^\phi_{i+1} = \{S'_2 \mid S'_1 \in [S']^\phi_i,\ S'_2 \in next^{MV_2}(S'_1),\ \phi(S') = \phi(S'_2)\}.$$

We now define the notion of a *step term*, an expression which is used to represent one possible way to model an abstract state using a set of original states. Such step terms will form the basis of our decision procedure.

Definition 12. Let $S \in Y(MV_1)$ and suppose $next^{MV_1}(S) = \{S_1, \ldots, S_m\}$. Then for each non-empty set of states $\Gamma \subseteq \phi^{-1}(S)$ we define the *step term* $st(S,\Gamma)$ by

$$st(S,\Gamma) = [S : \Gamma : D(S_1), \ldots, D(S_m)],$$

where $D(S_i) = \{S'_2 \mid S'_1 \in \Gamma,\ S'_2 \in next^{MV_2}(\{S'_1\} \cup [S'_1]^\phi),\ \phi(S'_2) = S_i\}$, and $next^{MV_2}$ has been lifted from taking a single state as input to taking a set of states in the obvious way. Note that the use of $[S'_i]^\phi$ is needed in the above definition to take account of the merging of consecutive identical states that occurs in abstracted traces (see part ii) in Definition 8).

We say a step term $[S : \Gamma : D(S_1), \ldots, D(S_m)]$ is *valid* iff:
i) the states Γ used in a step term have the appropriate connections, i.e. $D(S_i) \neq \{\}$, for $i = 1, \ldots, m$; and

ii) if S is a point attractor in $SG^A(MV_1)$ then it must be modelled by point attractors in $SG^A(MV_2)$ (discounting steps to identical abstracted states), i.e. if $next^{MV_1}(S) = \{\}$ then for each $S' \in \Gamma$ we have $next^{MV_2}([S']^\phi) - [S']^\phi = \{\}$. □

We let $Step(S)$ denote the set of all valid step terms

$$Step(S) = \{st(S,\Gamma) \mid \Gamma \subseteq \phi^{-1}(S),\ st(S,\Gamma) \textit{ is valid}\}.$$

Observe that each valid step term $st(S,\Gamma) \in Step(S)$ must correctly model in MV_2 the connections between $S \in Y(MV_1)$ and its corresponding next states $next^{MV_1}(S)$ in MV_1.

The proposed decision procedure is presented in Figure 4. It works by creating a family $C = \langle C(S) \subseteq Step(S) \mid S \in Y(MV_1)\rangle$ of sets of all valid step terms. It then repeatedly looks at each set of step terms $C(S)$, for each abstract state $S \in Y(MV_1)$, removing those that have next states that are not currently in the remaining stored step terms of C.

```
Algorithm checkAsynAbs($MV_1$,$MV_2$,$\phi$):

/** Initialise valid state terms **/
for each $S \in Y(MV_1)$ do $C(S) = Step(S)$
/** Iteratively check sets of step terms **/
repeat
    done:=true
    for each $S \in Y(MV_1)$ do
        for each $[S:\Gamma:D(S_1),\ldots,D(S_m)] \in C(S)$ do
            for $i$:= 1 to $m$ do
                if $st(S_i,D(S_i)) \notin C(S_i)$ then
                   $C(S) = C(S) - \{[S:\Gamma:D(S_1),\ldots,D(S_m)]\}$
                   done:=false
            if $C(S) = \{\}$ then return false
until (done = true)
return true
```

Figure 4: Decision procedure for checking asynchronous abstractions $MV_1 \lhd^{\phi}_{A} MV_2$.

It is straightforward to show that the decision procedure must always terminate.

Theorem 13. The decision procedure checkAsynAbs(MV_1,MV_2,ϕ) always terminates.

Proof. This follows from that fact we can only ever begin with a finite family of finite sets of step terms, that no step terms can ever be added, and that we must remove at least on step term in order to continue to the next iteration. Therefore the algorithm either terminates when no step terms are removed or continues to iterate until we reach a point where one set $C(S)$ of step terms is empty, again resulting in termination of the algorithm. □

The complexity of the decision procedure in the worst case, when MV_1 is not an asynchronous abstraction of MV_2, can be derived as follows. Assume MV_1 is a Boolean model which has n entities and k is an upper bound on the number of states in MV_2 that can be abstracted to a single state in MV_1, i.e. $k \geq |\phi^{-1}(S)|$, for all $S \in Y(MV_1)$. Note that k can be calculated from the abstraction mapping used and is not dependent on n. The three nested for loops in the decision procedure have an upper bound of $O(2^n \times 2^k \times n)$ where: 2^n is the number of states in MV_1; 2^k is an upper bound on the number of different sets of states that can be mapped to a given abstract state; and n represents the maximum number of states that can be connected to a given state. The outer repeat until loop will iterate round removing a single step term until one of the step term sets is empty. This gives a final upperbound of $O(2^{2(n+k)} \times n^2)$. In practice the decision procedure should perform much better than this. Note that for a given abstraction mapping, k can be seen as a fixed constant which does not increase as entities are added (providing the state of those entities is not abstracted).

Let $[S : \Gamma : D(S_1), \ldots, D(S_m)]$ be a valid step term, let $\alpha_1 \in \Gamma$ and $\alpha_2 \in D(S_i)$, for some $1 \leq i \leq m$. Then note that due to the way consecutive identical states are treated it may not directly hold that $\alpha_1 \xrightarrow{Asy} \alpha_2$ since $\alpha_2 \in next^{MV_2}(\{\alpha_1\} \cup [\alpha_1]^\phi)$. We let $\overline{\alpha_1} = \alpha_1 \xrightarrow{Asy} \alpha_1^1 \xrightarrow{Asy} \cdots \xrightarrow{Asy} \alpha_1^r$, for $\alpha_1^j \in [\alpha_1]^\phi$, for $1 \leq j \leq r$ represent the sequence of identical abstracted states needed such that $\overline{\alpha_1} \xrightarrow{Asy} \alpha_2$ does hold in MV_2.

The following lemma considers how step terms can be chained together and is is needed to prove the main correctness result below.

Lemma 14. Let $C = \langle C(S) \subseteq Step(S) \mid S \in Y(MV_1)\rangle$ be a family of sets of valid step terms such that:
i) For each $S \in Y(MV_1)$ we have $C(S) \neq \{\}$;
ii) The family C is closed under step terms, i.e. for each $S \in Y(MV_1)$ and $[S : \Gamma : D(S_1), \ldots, D(S_m)] \in C(S)$ we have $st(S_i, D(S_i)) \in C(S_i)$, for $1 \leq i \leq m$.

Then every path[2] $\gamma = \gamma_1 \xrightarrow{Asy} \ldots \xrightarrow{Asy} \gamma_p$ in the abstraction state graph $SG^A(MV_1)$ must have a corresponding path $\alpha = \alpha_1 \xrightarrow{Asy} \ldots \xrightarrow{Asy} \alpha_r$, $r \geq p$, in the original state graph $SG^A(MV_2)$ such that $\phi(\alpha) = \gamma$.

Proof.
Let $\gamma = \gamma_1 \xrightarrow{Asy} \cdots \xrightarrow{Asy} \gamma_p$ be a path in the state graph $SG^A(MV_1)$. Then by assumptions i) and ii) it is straightforward to see there must exist a (not necessarily unique) chain of step terms

$$[\gamma_i : \Gamma_i : \ldots, D(\gamma_{i+1}), \ldots] \in C(\gamma_i), \;\; st(\gamma_p, \Gamma_p) \in C(\gamma_p)$$

for $1 \leq i < p$, such that for $j = 2, \ldots, p$ we have $\Gamma_j = D(\gamma_j)$.

We now prove that for any $\alpha_p \in \Gamma_p$ there must exist $\alpha_i \in \Gamma_i$, for $1 \leq i < p$, such that $\alpha = \overline{\alpha_1} \xrightarrow{Asy} \cdots \xrightarrow{Asy} \overline{\alpha_{p-1}} \xrightarrow{Asy} \alpha_p$ is a path in $SG^A(MV_2)$ with $\phi(\alpha) = \gamma$. We prove this using induction on $p \in \mathbf{N}$, $p \geq 2$ as follows.

1) *Induction Base*. Let $p = 2$ and suppose we have a path $\gamma_1 \xrightarrow{Asy} \gamma_2$. Then we know there must exist step terms $[\gamma_1 : \Gamma_1 : \ldots, D(\gamma_2), \ldots] \in C(\gamma_1)$ and $st(\gamma_2, D(\gamma_2)) \in C(\gamma_2)$ (as explained above). Clearly by the definition of step terms we know that for any $\alpha_2 \in D(\gamma_2)$ there must exist $\alpha_1 \in \Gamma_1$ such that $\overline{\alpha_1} \xrightarrow{Asy} \alpha_2$ and

$$\phi(\overline{\alpha_1} \xrightarrow{Asy} \alpha_2) = \gamma_1 \xrightarrow{Asy} \gamma_2.$$

[2]We note that a path differs from a trace in that a trace represents a complete run of an MVN whereas a path is simply a walk through an MVN's state graph.

2) *Induction Step*. Let $p = q+1$, for some $q \in \mathbf{N}$, $q \geq 2$. Suppose we have a path $\gamma = \gamma_1 \xrightarrow{Asy} \cdots \xrightarrow{Asy} \gamma_q \xrightarrow{Asy} \gamma_{q+1}$. Then we know there must exist step terms

$$[\gamma_1 : \Gamma_1 : \ldots, D(\gamma_2), \ldots] \in C(\gamma_1),\ \ [\gamma_i : D(\gamma_i) : \ldots, D(\gamma_{i+1}), \ldots] \in C(\gamma_i),\ \ st(\gamma_{q+1}, D(\gamma_{q+1})) \in C(\gamma_{q+1}),$$

for $2 \leq i \leq q$ (as explained above). Then by the induction hypothesis we know for each $\alpha_q \in D(\gamma_q)$ there must exist $\alpha_i \in \Gamma_i$, for $1 \leq i < q$, such that $\overline{\alpha_1} \xrightarrow{Asy} \cdots \xrightarrow{Asy} \overline{\alpha_{q-1}} \xrightarrow{Asy} \alpha_q$ is a path in $SG^A(MV_2)$ with $\phi(\overline{\alpha_1} \xrightarrow{Asy} \cdots \xrightarrow{Asy} \overline{\alpha_{q-1}} \xrightarrow{Asy} \alpha_q) = \gamma_1 \xrightarrow{Asy} \cdots \xrightarrow{Asy} \gamma_q$. By the definition of step terms it follows that for any $\alpha_{q+1} \in D(\gamma_{q+1})$ there must exist $\alpha_q \in \Gamma_q$ such that $\overline{\alpha_q} \xrightarrow{Asy} \alpha_{q+1}$. Combining this with the induction hypothesis given above shows the existence of the required path in $SG^A(MV_2)$. $\square$

It now remains to show that the decision procedure `checkAsynAbs(`MV_1`,`MV_2`,`ϕ`)` correctly checks for asynchronous abstractions.

Theorem 15. `checkAsynAbs(`MV_1`,`MV_2`,`ϕ`)` returns *true* if, and only if, $MV_1 \lhd_A^{\phi} MV_2$.

Proof.
Part 1) $\Rightarrow$ Suppose `checkAsynAbs(`MV_1`,`MV_2`,`ϕ`)` returns *true*. By inspecting the decision procedure we can see this means that a family $\{C(S) \subseteq Step(S) \mid S \in Y(MV_1)\}$ of non–empty sets of valid step terms must have been found which is closed under step terms. Consider any abstract trace $\sigma \in Tr^A(MV_1)$; then by Lemma 14 and since any trace can be interpreted as a path in $SG^A(MV_1)$ we have that there must exist a path α in $SG^A(MV_2)$ such that $\phi(\alpha) = \sigma$. It is straightforward to see that α must be a well–defined trace for MV_2, i.e. $\alpha \in Tr^A(MV_2)$, by the definition of *valid* step term. This shows that $Tr^A(MV_1) \subseteq \phi(Tr^A(MV_2))$ and so by Definition 9 we have $MV_1 \lhd_A^{\phi} MV_2$.

Part 2) $\Leftarrow$ Suppose $MV_1 \lhd_A^{\phi} MV_2$ then by Definition 9 we know

$$Tr^A(MV_1) \subseteq \phi(Tr^A(MV_2)) \tag{1}$$

Then we show that there must exist a family of sets of valid step terms which are closed under step term inclusion and thus that `checkAsynAbs(`MV_1`,`MV_2`,`ϕ`)` must terminate returning *true*.

Let $X \subseteq Tr^A(MV_2)$ be the set of traces that abstractly correspond to $Tr^A(MV_1)$:

$$X = \{\sigma \mid \sigma' \in Tr^A(MV_2),\ \exists \sigma \in Tr^A(MV_1).\phi(\sigma') = \sigma\}$$

For each $S \in Y(MV_1)$, let $X\langle S\rangle$ denote the set of all states that abstract to S which occur at the start of a trace in X:

$$X\langle S\rangle = \{\sigma'(1) \mid \sigma' \in X,\ \phi(\sigma'(1)) = S\}$$

where $\sigma'(1)$ represents the first state of trace σ'. Let $next^{MV_1}(S) = \{S_1, \ldots, S_m\}$, then using Definition 12 we can define the step term

$$st(S, X\langle S\rangle) = [S : X\langle S\rangle : D(S_1), \ldots, D(S_m)]$$

Clearly, $st(S, X\langle S\rangle)$ must be valid by (1) above. We can now recursively define a set of step terms closed under step term inclusion from $st(S, X\langle S\rangle)$ as follows.

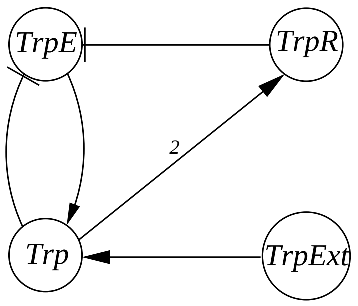
Define $H(X\langle S\rangle) = \bigcup_{i\in\mathbf{N}} H(X\langle S\rangle)_i$, where $H(X\langle S\rangle)_i$ is defined recursively: $H(X\langle S\rangle)_0 = \{st(S, X\langle S\rangle)\}$ and

$$H(X\langle S\rangle)_{i+1} = \{st(V_j, D(V_j)) \mid [V : \Gamma : D(V_1), \ldots, D(V_r)] \in H(X\langle S\rangle)_i,\ V_j \in \{V_1, \ldots, V_r\}\}.$$

Clearly, the set $H(X\langle S\rangle)$ is closed under step term inclusion by construction. Also note that it can only contain valid step terms; this follows from (1) above and the fact that if $st(S,\Gamma)$ is a valid step term then any new step term $st(S, \Gamma \cup \{S'\})$ formed by adding an additional state $S' \in \phi^{-1}(S)$ must also be valid. It therefore follows that for each $S \in Y(MV_1)$ we know that each step term $st(S_i, \Gamma) \in H(X\langle S\rangle)$ must occur in the initial family C of sets of step terms used in the decision procedure, i.e. $st(S,\Gamma) \in C(S)$. Since none of these step terms can be removed from C by the closure property it follows that the decision procedure `checkAsynAbs(`MV_1`,`MV_2`,`ϕ`)` must terminate returning *true*. □

5 Case Study: The Regulation of Tryptophan Biosynthesis

In this section we present a detailed case study which illustrates the abstraction techniques developed in the previous sections. Our case study is based on identifying abstractions for a published MVN model of the regulatory system used to control the biosynthesis of tryptophan in *E. coli* [30]. Tryptophan is an amino acid which is essential for the development of *E. coli*. However, the synthesis of tryptophan is resource intensive and for this reason is carefully controlled to ensure it is only synthesised when no external source is available. The regulatory network that controls the biosynthesis of tryptophan by *E. coli* has been extensively studied (see for example [29, 27]).



TrpE	*TrpExt*	*Trp*	[*Trp*]
0	0	0,1	0
0	0	2	1
0	1	0,1,2	1
0	2	0	1
0	2	1,2	2
1	0,1	0,1,2	1
1	2	0	1
1	2	1,2	2

Trp	[*TrpR*]
0,1	0
2	1

Trp	*TrpR*	[*TrpE*]
0	0	1
0	1	0
1,2	0,1	0

Figure 5: An MVN model *MTRP* of the regulatory mechanism for the biosynthesis of tryptophan in *E. coli* (from [30]). The state transition table for *TrpExt* has been omitted as this is a simple input entity. Note that the state transition tables use a shorthand notation where an entity is allowed to be in any of the states listed for it in a particular row.

Consider the MVN model *MTRP* for tryptophan biosynthesis presented in Figure 5 which is taken from [30]. It consists of four regulatory entities: *TrpE* – a Boolean input entity indicating the presence of the activated enzyme required for synthesising tryptophan; *TrpR* – a Boolean entity indicating if the repressor gene for tryptophan production is active; *TrpExt* – a ternary entity indicating the level of tryptophan in the external medium; and *Trp* – a ternary entity indicating the level of tryptophan within the bacteria. Note the above entity order is used when displaying global states for *MTRP*. We can see from the model that the presence of tryptophan in the external medium *TrpExt* directly affects the level of tryptophan within the bacteria *Trp* and that the activated enzyme *TrpExt* is required to synthesise tryptophan. The presence of tryptophan within the bacteria deactivates the enzyme *TrpE* and at higher-levels also activates the repressor *TrpR* which then acts to inhibit the production of the enzyme *TrpE*.

The state space for the *MTRP* consists of 36 global states and for this reason we do not reproduce its state graph here. Instead we simply note that the asynchronous state graph for *MTRP* comprises three disjoint graphs based on the following three attractors: $0000 \xrightarrow{Asy} 1000 \xrightarrow{Asy} 1001 \xrightarrow{Asy} 0001 \xrightarrow{Asy} 0000$; 0011; and 0122. To identify abstractions for *MTRP* we begin by defining appropriate state mappings for the non-Boolean entities *TrpExt* and *Trp* as follows:

$$\phi(Trp) = \{0 \mapsto 0, 1 \mapsto 1, 2 \mapsto 1\}, \quad \phi(TrpExt) = \{0 \mapsto 0, 1 \mapsto 1, 2 \mapsto 1\}.$$

These can then be combined into an abstraction mapping

$$\phi = \langle I_{TrpE}, I_{TrpR}, \phi(TrpExt), \phi(Trp) \rangle.$$

Following the approach presented in [5], we first apply this abstraction mapping to *MTRP* to produce a set $\phi(MTRP)$ of candidate abstraction models. By analysing $\phi(MTRP)$ we are able to establish that there are 8 possible candidate abstraction models (we have 4 choices for next-state of *TrpR* and 2 choices for *Trp*). After investigating these candidate models we were able to identify one valid asynchronous abstraction *ATRP* (which is presented in Figure 6) for *MTRP* under ϕ using the decision procedure `checkAsynAbs(`*ATRP*, *MTRP*, ϕ`)`. Note that since $Tr^A(ATRP)$ and $\phi(Tr^A(MTRP))$ are in fact finite trace sets in this case we were able to verify the result $ATRP \lhd^{\phi}_{A} MTRP$, by checking that $Tr^A(ATRP) \subseteq \phi(Tr^A(MTRP))$ holds.

Trp	[*TrpR*]
0,1	0

TrpE	*TrpExt*	*Trp*	[*Trp*]
0	0	0,1	0
0	1	0,1	1
1	0,1	0,1	1

Trp	*TrpR*	[*TrpE*]
0	0	1
0	1	0
1	0,1	0

Figure 6: The asynchronous abstraction *ATRP* identified for *MTRP* under the state mappings $\phi(Trp) = \{0 \mapsto 0, 1 \mapsto 1, 2 \mapsto 1\}$ and $\phi(TrpExt) = \{0 \mapsto 0, 1 \mapsto 1, 2 \mapsto 1\}$.

The state graph for *ATRP* consists of two disjoint graphs and has two attractors: $0000 \xrightarrow{Asy} 1000 \xrightarrow{Asy} 1001 \xrightarrow{Asy} 0001 \xrightarrow{Asy} 0000$; and 0011. It therefore successfully captures two of the three attractors present in *MTRP*.

6 Conclusions

In this paper we have developed an abstraction theory for asynchronous MVNs based on extending the ideas developed for synchronous MVNs [5] and defined what it means for an MVN to be correctly abstracted by a simpler MVN with the same network structure but smaller state space. The abstraction approach used is based on an *under–approximation* approach [9, 24] in which an abstraction captures a subset of the behaviour of the original MVN. We showed that this approach allows positive reachability properties of an MVN to be inferred from a corresponding asynchronous abstraction and that all attractors of an asynchronous abstraction correspond to attractors in the original MVN. An alternative approach would be to use an *over–approximation* approach [9, 24, 10] in which false positives can arise. However, the construction of an abstraction model which over–approximates an MVN's behaviour appears to be problematic if we wish to remain within the MVN framework (see Section 3 for a discussion of this).

Directly checking asynchronous abstractions turned out to be problematic given that an asynchronous MVN may have an infinite set of traces which makes it infeasible to directly check trace inclusion. To address this we developed a decision procedure for checking asynchronous abstractions based on the finite state graph of an asynchronous MVN. The decision procedure used *step terms* to denote possible ways to use sets of concrete states to represent abstract states and worked by iteratively pruning the set of step terms until either a consistent abstract representation has been found or the set of remaining step terms is too small to make it feasible to continue. Importantly, we provided a detailed proof that showed the decision procedure worked correctly. Note that as it stands, the decision procedure is inefficient; work is on going to refine this procedure and to use it as a basis of a tool for abstraction checking. Such a tool will provide the support needed to carry out more complex case studies, for example supporting the work currently underway to investigate abstractions for the relatively complex MVN model of the carbon starvation response in *E. coli* presented in [3].

We illustrated the abstraction theory and techniques developed by considering a detailed case study based on identifying a Boolean abstraction for an asynchronous MVN model of the regulatory system used to control the biosynthesis of tryptophan in *E. coli*. The abstraction found proved to faithfully represent the behaviour of the original MVN and in particular, captured two of the three attractors known to exist in the original MVN. The case study illustrates the potential for the abstraction theory presented and in particular, how it allows the balance between the level of abstraction used and the tractability of analysis to be explored.

An alternative approach for abstracting MVNs is to reducie the number of regulatory entities in an MVN while ensuring the preservation of key properties (see [21, 37, 22]). This approach seems to be complimentary to the one developed here and we are currently investigating combining these ideas. Another possible abstraction approach would be to make use of results on modelling MVNs using Petri nets [11, 3, 4, 8] and to then apply Petri net abstraction techniques (see for example [31, 19, 38]). Such an approach appears promising from an analysis point of view but problematic in that the resulting Petri net abstraction may not be interpretable as an MVN and so force the modeller to explicitly use a different modelling formalism.

One interesting area for future work is to investigate automatically constructing abstractions for a given MVN and abstraction mapping. Some initial work on restricting the search space for such abstractions can be found in [5] but more work is needed here. One idea is to consider developing refinement techniques similar to those of *CEGAR (Counterexample Guided Abstraction Refinement)* [10] and other abstraction refinement techniques [24]. Closely linked to this idea is the notion of a maximal abstraction, that is an abstraction which captures the largest possible behaviour of the original MVN with respect to all other possible abstractions for the given abstraction mapping. In future work we intend to investigate developing such a notion and in particular, consider how to automate the construction of such maximal abstractions.

Acknowledgments. We would like to thank Richard Banks and Maciej Koutny for their advice and support during the preparation of this paper. We would also like to thank the anonymous referees for their very helpful comments and suggestions.

References

[1] T. Akutsu, S. Miyano and S. Kuhara, Identification of Genetic Networks from Small Number of Gene Expression Patterns Under the Boolean Network model, *Proc. of Pac. Symp. on Biocomp.*, 4:17–28, 1999.

[2] R. Banks. *Qualitatively Modelling Genetic Regulatory Networks: Petri Net Techniques and Tools*. Ph. D. Dissertation, School of Computing Science, University of Newcastle upon Tyne, 2009.

[3] R. Banks and L. J. Steggles. A High-Level Petri Net Framework for Multi-Valued Genetic Regulatory Networks. *Journal of Integrative Bioinformatics*, 4(3):60, 2007.

[4] R. Banks, V. Khomenko, and L. J. Steggles. Modelling Genetic Regulatory Networks. In: I. Koch, W. Reisig and R. Schreiber (Eds), *Modelling in Systems Biology: the Petri Net Approach*, pages 73-100, Computational Biology Series, Springer Verlag, 2010.

[5] R. Banks and L. J. Steggles. An Abstraction Theory for Qualitative Models of Biological Systems. *Electronic Proceedings in Theoretical Computer Science*, 40:23-38, 2010.

[6] S. Bensalem, Y. Lakhnech, and S. Owre. Computing Abstractions of Infinite State Systems Compositionally and Automatically. In: *Proc. of the 10th Int. Conference on Computer Aided Verification*, Lecture Notes In Computer Science 1427, pages 319–331, Springer-Verlag, 1998.

[7] C. Chaouiya, E. Remy, and D. Thieffry. Petri Net Modelling of Biological Regulatory Networks. *Journal of Discrete Algorithms*, 6(2):165–177, 2008.

[8] C. Chaouiya, A. Naldi, E. Remy, and Thieffry. Petri Net Representation of Multi-Valued Logical Regulatory Graphs. *Natural Computing*, 10(2):727–750, 2011.

[9] E. M. Clarke, O. Grumberg, and D. E. Long. Model Checking and Abstractions. *ACM Transactions on Programming Languages and Systems*, 16(5):1512 - 1542, 1994.

[10] E. M. Clarke, O. Grumberg, S. Jha, Y. Lu and H. Veith. Counterexample-Guided Abstraction Refinement for Symbolic Model Checking. *Journal of the ACM*, 50(5):752–794, 2003.

[11] J. Comet, H. Klaudel, and S. Liazu. Modeling Multi-valued Genetic Regulatory Networks Using High-Level Petri Nets. Lecture Notes in Computer Science, vol. 3536, pagse 208–227, Springer–Verlag, 2005.

[12] B. Drossel, T. Mihaljev, and F. Greil. Number and Length of Attractors in a Critical Kauffman Model with Connectivity One. *Physical Review Letters*, 94(8), 2005.

[13] V. Da Silva, D. Kroening, and G. Weissenbacher. A Survey of Automated Techniques for Formal Software Verification. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 27(7):1165–1178 , 2008

[14] A. Esmaeili and C. Jacob. Evolutionary Exploration of Boolean Networks *Proceedings of the IEEE Congress on Evolutionary Computation*, pages 3396 - 3403, 2008.

[15] I. Harvey and T. Bossomaier. Time Out of Joint: Attractors in Asynchronous Random Boolean Networks. In: P. Husbands and I. Harvey (eds.), *Proc. of ECAL97*, pages 67–75, MIT Press 1997.

[16] S. Huang and D. Ingber. Shape-Dependent Control of Cell Growth, Differentiation, and Apoptosis: Switching Between Attractors in Cell Regulatory Networks. *Experimental Cell Research*, 261(1):91–103, 2000.

[17] S. A. Kauffman. Metabolic Stability and Epigenesis in Randomly Constructed Genetic Nets. *Journal of Theoretical Biology*, 22(3):437-467, 1969.

[18] S. A. Kauffman. *The Origins of Order: Self-Organization and Selection in Evolution.* Oxford University Press, New York, January 1993.

[19] P. Küngas. Petri Net Reachability Checking Is Polynomial with Optimal Abstraction Hierarchies. Lecture Notes in Computer Science, vol. 3607, pages 149–164, Springer–Verlag, 2005. Copyright: 2005

[20] A. Mishchenko and R. Brayton. Simplification of Non-deterministic Multi-Valued Networks. In: *ICCAD '02: Proc. of the 2002 IEEE/ACM Int. Conference on Computer-aided design*, pages 557–562, 2002.

[21] A. Naldi, E. Remy, D. Thieffry, and C. Chaouiya. A Reduction of Logical Regulatory Graphs Preserving Essential Dynamical Properties. In: *Proc. of CMSB '09*, Lecture Notes in Bioinformatics 5688, pages 266 - 280, Springer-Verlag, 2009.

[22] A. Naldi, E. Remy, D. Thieffry, and C. Chaouiya. Dynamically Consistent Reduction of Logical Regulatory Graphs. *Theoretical Computer Science*, 412(21):2207-2218, 2011.

[23] A.B. Oppenheim, O. Kobiler, J. Stavans, D. L. Court, and S. L. Adhya. Switches in Bacteriophage λ Development. *Annual Review of Genetics*, 39:4470–4475, 2005.

[24] R. Pelánek. *Reduction and Abstraction Techniques for Model Checking*. PhD thesis, Masaryk University Brno, 2006.

[25] R. Rudell and A. Sangiovanni-Vincentelli. Multiple-Valued Minimization for PLA Optimization. *IEEE Transactions on Computer-Aided Design*, CAD-6, 1987.

[26] A. Saadatpour, I. Albert, and R. Albert. Attractor Analysis of Asynchronous Boolean Models of Signal Transduction Networks. *Journal of Theoretical Biology*, 266:641–656, 2010.

[27] M. Santillán and M. C. Mackey. Dynamic Regulation of the Tryptophan Operon: A modeling Study and Comparison with Experimental Data *PNAS*, 98(4): 1364-1369, 2001.

[28] M. Schaub, T. Henzinger, and J. Fisher. Qualitative Networks: A Symbolic Approach to Analyze Bio-Logical Signaling Networks. *BMC Systems Biology*, 1:4, 2007.

[29] A. K. Sen and W. Liu. Dynamic Analysis of Genetic Control and Regulation of Amino Acid Synthesis: The Tryptophan Operon in Escherichia Coli. *Biotechnology and Bioengineering*, 35(2):185-194, 1990.

[30] E. Simão, E. Remy, D. Thieffry and C. Chaouiya. Qualitative Modelling of Regulated Metabolic Pathways: Application to the Tryptophan Biosynthesis in E. Coli. *Bioinformatics*, 21: ii190-196, 2005.

[31] I. Suzuki and T. Murata. A Method for Stepwise Refinement and Abstraction of Petri Nets. *Journal of Computer and System Sciences*, 27:51-76, 1983.

[32] D. Thieffry and R. Thomas. Dynamical Behaviour of Biological Regulatory Networks - II. Immunity Control in Bacteriophage Lambda. *Bulletin of Mathematical Biology*, 57:277–295, 1995.

[33] R. Thomas. Boolean Formalization of Genetic Control Circuits. *Journal of Theoretical Biology*, 42:563–585, 1990.

[34] R. Thomas and R. D'Ari. *Biological Feedback*, CRC Press, 1990.

[35] R. Thomas, D. Thieffry and M. Kaufman. Dynamical Behaviour of Biological Regulatory Networks - I. Biological Role of Feedback Loops and Practical use of the Concept of Loop-Characteristic State. *Bulletin of Mathematical Biology*, 57:247–276, 1995.

[36] L. Tournier and M. Chaves. Uncovering Operational Interactions in Genetic Networks Using Asynchronous Boolean Dynamics. *Journal of Theoretical Biology*, 260:196–209, 2009.

[37] A. Veliz–Cuba. Reduction of Boolean Networks.
http://arxiv.org/abs/0907.0285, submitted 2009. (Visited Dec 2010)

[38] H. Wimmel and K. Wolf, Karsten. Applying CEGAR to the Petri Net State Equation. In: *Proc. of TACAS'11/ETAPS'11*, Lecture Notes in Computer Science, vol. 6605, pages 224–238, Springer–Verlag, 2011.

[39] A. Wuensch. Basins of Attraction in Network Dynamics: A Conceptual Framework for Biomolecular Networks, In: G.Schlosser and G.P.Wagner (Eds), *Modularity in Development and Evolution*, pages 288-311, Chicago University Press, 2002.

Multiscale Modelling: A Mobile Membrane Approach

(Work in Progress)

Federico Buti, Massimo Callisto De Donato, Flavio Corradini, Emanuela Merelli, Luca Tesei
School of Science and Technology, Computer Science Division
University of Camerino, Camerino, Italy
{name.surname}@unicam.it

Nowadays, multiscale modelling is recognized as the most suitable way to study biological processes. Indeed, almost every phenomenon in nature exhibits a multiscale behaviour, i.e., it is the outcome of interactions that occur at different spatial and temporal scales. Although several ways to provide "multilayer" models have been proposed, only Complex Automata naturally embed spatial information and realize the multiscale approach with well-established inter-scale integration schemas. Recently, such approach has been restated in terms of Spatial P systems - a variant of P systems with a more geometric concept of space.
In this work we discuss how mobile membranes, a variant of membrane systems inspired by the biological movements of endocytosis and exocytosis, can be efficaciously exploited to define a uniform multiscale coupling scheme relying only on the features of the formalism itself.

1 Introduction

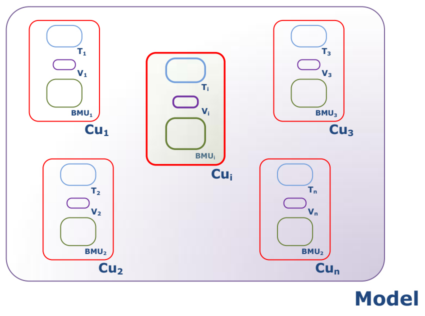
Many biological phenomena are characterized by interactions involving different spatial and temporal scales simultaneously. At each scale, different structures come into play. Consequently, several computational methodologies have been developed for modelling biological processes with different degrees of resolution. Among them, *multiscale* modelling rose up as the most suitable one. In such approach, several models at different spatial and temporal scales, are *coupled* together. How these models are homogenized, i.e, *integrated* is a key aspect to guarantee that the overall model is faithful to the real phenomenon.

Recently, complex Automata (CxA) [4] were introduced as a robust multiscale modelling solution which takes in account coupling issues. In CxA the multiscaling is realised on uniform components, the *Cellular Automata*, (CA) by different and well-established integration schemas. Any CxA consists of a finite grid of CA cells, where each cell has an associated state taken from a finite set of different states. In [2], authors rephrased the CxA approach in term of Spatial P Systems (SPs) [1], a variant of P Systems enriched with a notion of space. Similarly to CxA, SPs represents space as a geometric 2D grid-based set of cells that contain objects as in classical P Systems. As a case study, authors provided a uniform multiscale model of bone remodelling, a common multiscale phenomena, as an aggregation of single SP models at different scales, coupled through functions *external* to the paradigm. Following their approach, we discuss how mobile membranes[3] can be exploited to define a multiscale coupling scheme which relies only on the operations provided from the mobile membrane paradigm itself.

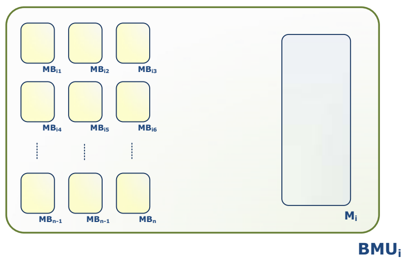
2 Bone remodelling with mobile membranes

Bone remodelling is a biological phenomenon in which mature bone tissue is removed from the skeleton (*resorption*) and new bone tissue is formed (*formation*); such process is multiscale since macroscopic

© F. Buti, M. Callisto De Donato,
F. Corradini, E. Merelli and L. Tesei
All the rights to the paper remain with the authors.



(a) MODEL



(b) BMU

Figure 1: Multiscale model for bone remodelling. Figure (a) represents the whole model containing the coupling membranes CU (red). Figure (b) shows a detailed BMU (cell scale) membrane.

behaviours, e.g. mechanical stimuli at the tissutal level and microstructure (cell scale actors - e.g. hormones, receptors etc.) strongly influence each other. Figure 1(a) shows the proposed model based on mobile membranes. As in [2], we consider two resolution levels: the tissutal and the cellular ones. Tissue level is divided into a series of membranes T_i (cyan), each representing a portion of bone. Each membrane contains, at any moment, a number of objects c proportional to the mineralisation value (expressed as a density in a certain interval) of the tissue. At the cellular scale we consider a series of BMU_i membranes (green), i.e. "Bone Multicellular unit" membranes representing the cellular sites in which remodelling occurs[1]. The integration of the two scales is realised through a coupling membrane CU_i (red) and a simple membrane V_i (violet), that moves between the tissutal and cellular membranes carrying the coupling information. No external function is needed since the paradigm provide the right means to handle the coupling.

3 Future work

We sketched a possible application of Mobile Membranes to the definition of a multiscale coupling scheme which, differently from [2], totally relies on the paradigm itself. In the near future we aim to fully develop this approach. Moreover, we also aim to study the paradigm expressivity w.r.t. CxA, and SPs. To this aim, an extension of Mobile Membranes with an explicit spacial notion is mandatory.

References

[1] R. Barbuti, A. Maggiolo-Schettini, P. Milazzo, G. Pardini & L. Tesei (2010): *Spatial P Systems*. *Natural Computing* Received: 26 October 2009 Accepted: 24 February 2010 Published online: 24 March 2010.

[2] D. Cacciagrano, F. Corradini, E. Merelli & L. Tesei (2010): *Multiscale Bone Remodelling with Spatial P Systems*. In: G. Ciobanu & M. Koutny, editors: *MeCBIC*, *EPTCS* 40, pp. 70–84. Available at `http://dx.doi.org/10.4204/EPTCS.40.6`.

[3] G. Ciobanu (2010): *Membrane Computing and Biologically Inspired Process Calculi*. Editura Universitatii Alexandru Ioan Cuza Din Iassi.

[4] A. G. Hoekstra, J.L. Falcone, A. Caiazzo & B. Chopard (2008): *Multi-scale Modeling with Cellular Automata: The Complex Automata Approach*. In: *ACRI*, pp. 192–199.

[1] A detailed description can be found at: `http://courses.washington.edu/bonephys/physremod.html`.

Modelling of Genetic Regulatory Mechanisms with GReg

Nicolas Sedlmajer
University of Geneva
sedlmaj0@etu.unige.ch

Didier Buchs
University of Geneva
didier.buchs@unige.ch

Steve Hostettler
University of Geneva
steve.hostettler@unige.ch

Alban Linard
University of Geneva
alban.linard@unige.ch

Edmundo Lopez
University of Geneva
edmundo.lopez@unige.ch

Alexis Marechal
University of Geneva
alexis.marechal@unige.ch

Most available tools propose simulation frameworks to study models of biological systems, but simulation only explores a few of the most probable behaviours of the system. On the contrary, techniques such as model checking, coming from IT-systems analysis, explore all the possible behaviours of the modelled systems, thus helping to identify emergent properties. A main drawback from most model checking tools in the life sciences domain is that they take as input a language designed for computer scientists, that is not easily understood by non-expert users. We propose in this article an approach based on DSL. It provides a comprehensible language to describe the system while allowing the use of complex and powerful underlying model checking techniques.

However, investigation through formal models of biological systems is not as widely spread as in other natural sciences such as chemistry and physics. This is partly due to the complexity of the living systems themselves, the strenuosity to formalize biological concepts, the difficulty in performing experiments on living systems in order to test a model, *etc*. Another reason is that formal models require the use of formalisms (*e.g.,* Petri Nets (PNs)), which are usually too complex for non-experts. To circumvent this difficulty, we propose to use the Domain Specific Languages (DSL) approach. A DSL is a language designed to be understandable by a domain expert and at the same time translatable into a formal language. This paper presents Gene Regulation Language (GReg), a DSL for the modelling of genetic regulatory mechanisms through the regulatory network approach [3, 4].

The main idea of regulatory networks is to model inter-biological reactions through a set of interdependent biological rules. This can be seen as a set of discrete modules having strong interconnections. The occurrence of interesting events in the biological system can be represented as logical properties expressed on the states of these modules. This is very similar to the kind of properties computer scientists validate on hardware and software systems (deadlocks, invariants, reachability, . . .).

Among the tools available in this domain, the main analysis approach for regulatory networks is *simulation*. Simulation is generating and analyzing a limited sample of possible system behaviours. This technique is not convenient when the main purpose of the research is to look for rare or abnormal behaviours (*e.g.,* cancer). The main approach in this case is to use *model checking* instead of simulation. Model checking consists in generating and analysing all the possible states of the system. Naturally, this technique suffers from the drawback of the enormous number of possible states of biological systems.

It is interesting to note that this problem is well-known to the model checking community in computer science, where it is called the *state space explosion*. There is a parallel between cellular interactions and software systems in that the state space explosion is partly due to their concurrent nature, but mainly due to the number of molecular species, and their respective populations, that are present in living organisms. Therefore, we can apply techniques that have been developed for the model checking of hardware and

Proceedings of 5th Workshop on Membrane Computing
and Biologically Inspired Process Calculi (MeCBIC 2011)
Pages 115–116, 2011.

© N. Sedlmajer, D. Buchs, S. Hostettler
A. Linard, E. Lopez and A. Marechal
All the rights to the paper remain with the authors.

software systems to biological interactions. Approaches based on a symbolic encoding of the state space [2] are particularly well-suited for this.

In this paper we show a work in progress in our group. We present GReg, a DSL dedicated to the modelling of genetic regulatory mechanisms. GReg is given as an example of the DSL-based verification process. We describe the creation of a language tailored to the understanding of domain experts, and how this language can be translated into a formal model where model checking can be applied. Model checking is a very well known verification technique used in computer science. Its main advantage is the complete exploration of the state space of the model, thus allowing to discover rare but potentially interesting events. A query language to express the properties of such events is embedded with GReg.

Three axes of development lie ahead: improving the expressivity of the modelling and query languages, assessing the usability of the approach and exploring the mitigation of the state space explosion.

Extending the expressivity of GReg Concepts such as time and probabilities play an important role in biology and are therefore good candidates for a language extension. As Algebraic Petri Nets (APNs), the current underlying formalism, do not support these notions, such extension would require to change the target platform. Good examples of target formalisms are timed Petri nets and stochastic Petri nets. The techniques used to create GReg allow changing the target language without changing the language itself.

Improving the usability of our tool. Although highly efficient, textual languages are usually not as intuitive as graphical languages. On the other hand, graphical domain specific languages are especially good in the early phase of the modeling as well as for documentation, but they are often less practical when the model grows. The tools in Eclipse Modelling Project (EMP) that were used to create GReg allow us to define a graphical version of the same language, thus keeping the best of both worlds. Another way to ease the modelling phase is to allow import/export of models from/to other formalisms and standards such as Systems Biology Markup Language (SBML) and to integrate GReg with *Cytoscape* framework through its plug-in mechanism.

Mitigating the state space explosion. So far, we have done little experimentation in this area for real biological processes. This weakness is being worked on as we speak using studies found in the literature and adapted to GReg. We are also working on a more detailed comparison to other tools dedicated to biological problems, *e.g.,* GinSim. Nevertheless, we conducted several studies on usual IT protocols and software models that show that AlPiNA can handle huge state spaces [1]. This suggests promising results in the regulatory mechanisms domain.

References

[1] D. Buchs, S. Hostettler, A. Marechal, and M. Risoldi. AlPiNA: A Symbolic Model Checker. In *Petri Nets'2010: Applications and Theory of Petri Nets, Braga, Portugal*, volume 6128 of *Lecture Notes in Computer Science*, pages 287–296, 2010.

[2] J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill, and L. J. Hwang. Symbolic Model Checking: 10^{20} States and Beyond. *Information and Computation*, 98, 1992.

[3] C. Chaouiya, H. Klaudel, and F. Pommereau. A Modular, Qualitative Modelling of Regulatory Networks Using Petri Nets. In *Modeling in Systems Biology*, volume 16 of *Computational Biology*, pages 253–279. Springer London, 2011.

[4] R. Thomas. Regulatory Networks seen as Asynchronous Automata: a Logical Description. *Journal of Theoretical Biology*, 153:1–23, 1991.

Time-Lengths in Time Petri Net Models for Steady States in Biochemical Systems

Louchka Popova-Zeugmann[1] and Elisabeth Pelz[2]

[1] Department of Computer Science, Humboldt University, Berlin, Germany
popova@informatik.hu-berlin.de
[2] LACL, University Paris Est Créteil, Fac de Sciences, Créteil, France
pelz@u-pec.fr

Abstract

In this paper we are interested in situations of biochemical reaction systems (BioRSs) in which all species have constant concentrations and all reactions have constant rates. Such a situation is called steady state and usually achieved after some time has elapsed since the system's beginning.

We start with a model of a BioRS presented as a stochastic Petri net (SPN). This is originally obtained from a system of ordinary differential equations (ODEs) achieved using simulation and interpolation.

The concentrations and rates modeling the steady state in the BioRS can be established by simulation of the SPN-model. A steady state in the BioRS signifies also that on the model level only a subset of all possible reachable states is relevant to this situation. It would be of interest to isolate formally and constructively this subset of states.

Considering the SPN model we propose an approach to calculate the part of the state space corresponding to a steady state in the BioRS. To do so, we map the SPN-model onto a Time Petri Net-model (TPN) with the same behaviour as that formerly observed one during the simulation of the SPN attaining the steady state. Finally, the TPN can be analyzed qualitatively and quantitatively. Notably, the time length of the cycle(s) representing the steady state can now be computed. To our knowledge there is no way to achieve this using the SPN-model or the ODE-model.

In addition, this approach helps for validating the correctness of the calculated (and used) rates in the steady state in the BioRS and of the parameters used in the original ODEs, fixed by experiments in the wet labs, both being -a priori- subject to a certain degree of uncertainty.

We thank the PC for inviting us to present our work at this workshop so that we can help to build a bridge between traditionnal MecBIC themes and Petri Nets. All details of this work can be found in http://ceur-ws.org/Vol-724.