6th Workshop on Membrane Computing and Biologically Inspired Process Calculi

MeCBIC 2012

Newcastle, UK
8th September 2012

Editors: Bogdan Aman
Gabriel Ciobanu

Table of Contents

MeCBIC 2012: Biologically Inspired Formalisms

Gabriel Ciobanu
Romanian Academy, Institute of Computer Science
Iaşi, Romania
gabriel@info.uaic.ro

This volume contains the papers presented at the 6th MeCBIC (Membrane Computing and Biologically Inspired Process Calculi), a satellite workshop of the 23rd International Conference on Concurrency Theory (CONCUR 2012) held on 8th September 2012 in Newcastle upon Tyne.

The modelling and the analysis of biological systems has attracted the interest of several research communities. The notion of compartments appears in rule-based formalisms as membrane computing, and in several process calculi (bio-ambients, brane calculi, etc.). Multiset rewriting appears both in membrane computing and Petri nets. A cross fertilization of various research areas leads to deeper investigations of the relations between these related formalisms, trying also to understand their similarities and differences. MeCBIC started as an workshop devoted to membrane computing and biologically inspired process calculi. In the last years, it also attracted papers dealing with (bio-inspired) Petri nets, emphasizing the links between Petri nets and membrane systems. Membrane computing deals with the computational properties, making use of automata, formal languages, and complexity results. Petri nets are used to model and analyse several biological systems by using advanced software tools. Certain process calculi, such as mobile ambients and brane calculi, describe the compartments and their interactions, emphasizing on behaviour equivalences and stochastic aspects.

The main aim of the workshop is to bring together researchers working in these biologically inspired formalisms (membrane systems, Petri nets, ambient and brane calculi, various stochastic approaches) in order to present their recent results and to discuss new ideas concerning these formalisms, their properties and relationships. Topics presented at MeCBIC include (but are not limited to):

- Biologically inspired models and calculi;
- Biologically inspired systems and their applications;
- Analysis of properties of biologically inspired models and languages;
- Theoretical links and comparison between different models/systems.

The submitted papers describe biologically inspired models and languages, as well as various properties and links between different models. They include various stochastic approaches, spatial interactions, modular verification of pathways, behaviour-based language in synthetic biology, and RNA-induced transcriptional aspects. The papers were reviewed by at least three referees. We thank very much to the members of the Programme Committee:

Bogdan Aman	Roberto Barbuti	Luca Cardelli	Gabriel Ciobanu (chair)
Erik de Vink	Jean-Louis Giavitto	Jane Hillston	Jetty Kleijn
Jean Krivine	Emanuela Merelli	Paolo Milazzo	Gethin Norman
G. Michele Pinna	Franck Pommereau	Jason Steggles	Angelo Troina.

Pre-Proceedings of 6th Workshop on Membrane Computing and Biologically Inspired Process Calculi
MeCBIC 2012, Pages 1–2.

We express our gratitude to the invited speakers Maciej Koutny and Vincent Danos for their interesting talks. Maciej Koutny presents the automated synthesis from behavioural specifications for a number of Petri net models relevant from the point of view of membrane systems. Vincent Danos defines a class of local stochastic rewrite rules on directed site trees, giving a compact presentation of coarse-grained differential systems describing the dynamics of these rules.

Many thanks to Bogdan Aman for his work in preparing this volume, and to Jason Steggles, Emilio Tuosto and local organizers for their help.

Invited Talks

Coarse-graining the Dynamics of Ideal Branched Polymers

Vincent Danos
University of Edinburgh
vincent.danos@gmail.com

We define a class of local stochastic rewrite rules on directed site trees. We give a compact presentation of (often countably infinite) coarse-grained differential systems describing the dynamics of these rules in the deterministic limit, and study in a simple case finite approximations based on truncations to a certain size. We show an application to the modelling of the dynamics of sugar polymers.

Pre-Proceedings of 6th Workshop on Membrane Computing and Biologically Inspired Process Calculi
MeCBIC 2012, Page 5.

Petri Net Synthesis and Membrane Systems

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

Maciej Koutny
School of Computing Science
Newcastle University
Newcastle upon Tyne, UK
maciej.koutny@ncl.ac.uk

Marta Pietkiewicz-Koutny
School of Computing Science
Newcastle University
Newcastle upon Tyne, UK
marta.koutny@ncl.ac.uk

Grzegorz Rozenberg
LIACS, Leiden University
Leiden, The Netherlands
and
Department of Computer Science
University of Colorado at Boulder
Boulder, Colorado, USA
rozenber@liacs.nl

Membrane systems are a computational model inspired by the functioning of living cells and their architecture, in particular the way chemical reactions take place in cells divided by membranes into compartments. Petri nets are a well-established general model for distributed computation with an extensive range of tools and methods for construction, analysis, and verification of concurrent systems. There are intrinsic similarities between Petri nets and membrane systems. In particular, there is a canonical way of translating membrane systems into Petri nets. This translation is faithful in the sense that it relates computation steps at the lowest level and induces in a natural way (sometimes new) extensions and interpretations of Petri net structure and behaviour (e.g., inhibitor arcs, localities, maximal concurrency). This strong semantical link between the two models invites to extend where necessary existing Petri net techniques and bring them to the domain of membrane systems. An example is the process semantics of Petri nets that, on the one hand, could help to understand the dynamics and causality in the biological evolutions represented by membrane systems and, on the other hand, poses new challenges for Petri nets with localities. In this talk we focus on the synthesis problem, that is, the problem of automated construction of a system from a specification of its (observed or desired) behaviour.

Automated synthesis from behavioural specifications is an attractive and powerful way of constructing correct concurrent systems. We start from the problem of synthesising a Petri net from a behavioural specification given in the form of a transition system which specifies the desired state space of the Petri net to be constructed. We will discuss solutions to this problem based on the notion of region of a transition system. Intuitively, a region captures a place of the net through essential behavioural characteristics as encoded in the transition system, including marking information and connectivity with all the transitions. Since places represent objects in compartments, transitions chemical reactions, and markings the configurations of a membrane system, such synthesis procedure paves the way for automated synthesis of membrane systems from state spaces.

We will outline the essence of the general region-based solution to the net synthesis problem in the setting of so-called τ-nets and corresponding τ-regions. Here the parameter τ is a general and convenient way of capturing different types of connections (arcs and their combinations) between places and transitions, removing the need to re-state and re-prove the key results every time a new kind of transitions or arcs is introduced. Next we review existing solutions to the synthesis problem for a number of Petri

Pre-Proceedings of 6th Workshop on Membrane Computing and Biologically Inspired Process Calculi
MeCBIC 2012, Pages 7–8.

net models relevant from the point of view of membrane systems, including place/transition nets, nets with inhibitor and activator arcs (modelling inhibitors and promoters in reaction systems), and nets with localities (compartments in membrane systems). To deal with nets with localities, we extend the synthesis to include also firing policies (such as maximal concurrency). Finally, we will discuss two recently introduced Petri net classes, viz. SET-nets (with qualitative rather than quantitative resource management, motivated by reaction systems) and nets with a/sync connections (which could provide a novel way of interpreting the interplay between different, simultaneously occurring reactions). There will also be some brief comments on automated synthesis from behavioural specifications other than state spaces, such as sets of sample executions.

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

References

[1] E.Badouel, P.Darondeau. Theory of Regions. In [16], 529-586, 1998.

[2] P.Darondeau, M.Koutny, M.Pietkiewicz-Koutny, A.Yakovlev. Synthesis of Nets with Step Firing Policies. Fundamenta Informaticae 94, 275-303, 2009.

[3] J.Desel, W.Reisig. The Synthesis Problem of Petri Nets. Acta Informatica 33, 297-315, 1996.

[4] J.Desel, A.Yakovlev (eds.). Applications of Region Theory 2011. Proc. of the Workshop Applications of Region Theory 2011 (ART-2011), CEUR-WS 725, 2011. `http://ceur-ws.org/Vol-725/`

[5] A.Ehrenfeucht, G.Rozenberg. Partial 2-structures; Part I: Basic Notions and the Representation Problem, and Part II: State Spaces of Concurrent Systems. Acta Informatica 27, 315-368, 1990.

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

[7] J.Kleijn, M.Koutny. Petri Nets and Membrane Computing. In [15], 389-412, 2009.

[8] J.Kleijn, M.Koutny. Membrane Systems with Qualitative Evolution Rules. Fundamenta Informaticae, 110, 217-230, 2011.

[9] J.Kleijn, M.Koutny, M.Pietkiewicz-Koutny. Regions of Petri Nets with a/sync Connections. Theoretical Computer Science 454, 189-198, 2012.

[10] J.Kleijn, M.Koutny, M.Pietkiewicz-Koutny, G.Rozenberg. Classifying Boolean Nets for Region-based Synthesis. In [4], 5-21, 2011.

[11] J.Kleijn, M.Koutny, M.Pietkiewicz-Koutny, G.Rozenberg. Step Semantics of Boolean Nets Acta Informatica (to appear).

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

[13] J.Kleijn, M.Koutny, G.Rozenberg. Petri Nets for Biologically Motivated Computing. Scientific Annals of Computer Science 21, 199-225, 2011.

[14] M.Koutny, M.Pietkiewicz-Koutny. Synthesis of Petri Nets with Localities. Scientific Annals of Computer Science 19, 1-23, 2009.

[15] G.Păun, G.Rozenberg, A.Salomaa. The Oxford Handbook of Membrane Computing. Oxford University Press, 2009.

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

Regular Papers

A Process Calculus for Spatially-explicit Ecological Models

Margarita Antonaki Anna Philippou
Department of Computer Science
University of Cyprus

cs05ma@cs.ucy.ac.cy annap@cs.ucy.ac.cy

We propose PALPS, a Process Algebra with Locations for Population Systems. PALPS allows us to produce spatially-explicit, individual-based models and to reason about their behavior. Our calculus has two levels: at the first level we may define the behavior of an individual of a population while, at the second level, we may specify a system as the collection of individuals of various species located in space, moving through their life cycle while changing their location, if they so wish, and interacting with each other in various ways such as preying on each other. Furthermore, we propose a probabilistic temporal logic for reasoning about the behavior of PALPS processes. We illustrate our framework via models of dispersal in metapopulations.

1 Introduction

During the last decade we have witnessed an increasing trend towards the use of formal frameworks for reasoning about biological as well as ecological systems including process algebras [29, 28, 9, 23, 13], Membrane Systems [27, 11] and cellular automata [16]. Process algebras, first proposed in [24, 19] to aid the understanding and reasoning about communication and concurrency, provide a number of features that make them suitable for capturing biological processes. In particular, process algebras are especially suited towards the so-called "individual-based" approach of modeling populations, as they enable one to describe the evolution of each individual of the population as a process and, subsequently, to compose a set of individuals (as well as their environment) into a complete ecological system. Features such as time, probability and stochastic behavior, which have been extensively studied within the context of process algebras, can be exploited to provide more accurate models, while associated analysis tools can be used to analyze and predict their behavior.

In this work, our aim is to introduce a process-algebraic framework to enable spatially-explicit modeling of ecological systems. Such modeling [14, 3] has been of special interest to conservation scientists and practitioners who have employed it in order to predict how species will respond to specific management schemes and guide the selection of reservation sites and reintroduction efforts, e.g. [18, 26]. The use of spatially-explicit, individual-based modeling requires the description of the environment as well as each individual residing in it, including a description of each individual's interaction with other individuals as well as the environment. As far as the environment is concerned, these models typically involve the use of *patches* or a *lattice* to represent the habitat. Individuals are then placed on specific locations of the modeled landscape and their behavior, including events such as birth, mortality, and dispersal, is simulated at the individual or population level and analyzed.

In order to capture this type of behavior our process algebra, PALPS, associates processes with information about their location and their species. The habitat is defined as a graph consisting of a set of locations and a neighborhood relation. Movement of located processes is then modeled as the change in the location of a process, with the restriction that the originating and the destination locations are

Pre-Proceedings of 6th Workshop on Membrane
Computing and Biologically Inspired Process Calculi
MeCBIC 2012, Pages 11–25.

neighboring locations. In addition to moving between locations, located processes may communicate with each other by exchanging messages upon channels. Communication may take place only between processes which reside at the same location while special channels allow processes to engage in preying and reproduction. Furthermore, PALPS may model probabilistic events, with the aid of a probabilistic choice operator, and uses a discrete treatment of time. Finally, in PALPS, each location may be associated with a set of attributes capturing relevant information such as the capacity or the quality of the location. These attributes form the basis of a set of expressions that refer to the state of the environment and are employed within models to enable the enunciation of location-dependent behavior.

The operational semantics of our calculus is given in terms of a labeled transition system on which we may check properties expressed in an instantiation of the PCTL temporal logic. We illustrate the expressiveness of PALPS by constructing spatially-explicit individual-based models for metapopulation dispersal.

There exists a variety of previous proposals which introduces locations or compartments into formal frameworks, e.g. [2, 10, 12, 2, 21, 4, 7], while work has been carried out to employ these frameworks for modeling and analyzing population systems [5]. PALPS is to a large extent influenced by such works. However, it departs from these works in that it is the first process-algebraic framework developed specifically for reasoning about ecological models as well as in its treatment of a state and its capability of expressing state-dependent behavior. In particular, it can be considered as an extension of WSCCS of [29] with locations and location attributes, while it shares a similar treatment of locations with process algebras developed for reasoning about mobile ad hoc networks, e.g. [22, 17]. As such, PALPS considers a two-dimensional space where locations and their interconnections are modeled as a graph upon which individuals may move as computation proceeds. The main feature that distinguishes PALPS from existing formal frameworks is the fact that it associates locations with a set of attributes that model special characteristics of locations that may be of interest when modeling a system and the ability to express behavior of individuals that is conditional on the values of these attributes. Examples of attributes that can be observed by individuals is the number of individuals a location can support as well the current number of individuals present at a location.

In the remainder of the paper we present the syntax and the semantics of PALPS in Section 2, while in Section 3 we provide models of metapopulation dispersal. In Section 4 we conclude with a discussion on future work.

2 The Process Calculus

In our calculus, PALPS (Process Algebra with Locations for Population Systems), we consider a system as a set of individuals operating in space, each possessing a species and a location identifier. Movement in the calculus is modeled via a specialized action whose effect is to change the location of an individual, with the restriction that the originating and the destination locations are neighboring locations. The notion of neighborhood is implemented via a relation **Nb** where $(\ell, \ell') \in \mathbf{Nb}$ exactly when locations ℓ and ℓ' are neighbors. We also use **Nb** as a function and write $\mathbf{Nb}(\ell)$ for the set of all neighbors of ℓ.

2.1 The Syntax

We continue to formalize the syntax of PALPS. We begin by describing the basic entities of the calculus.

- We assume a set of special labels **S** corresponding to the species under consideration, ranged over by $\mathbf{s}$, $\mathbf{s}'$.

- Furthermore, we assume a set of channels **Ch**, ranged over by lower-case strings. This set contains the special channels $rep_{\mathbf{s}}$ and $prey_{\mathbf{s}}$, $\mathbf{s} \in \mathbf{S}$, which are channels used to model reproduction of species **s** and preying on species **s**.
- Finally, we assume a set of locations **Loc** ranged over by ℓ, ℓ'. Locations can be associated with a set of attributes that model special characteristics of locations of interest within a system. We write ψ for attributes and ψ_ℓ for the value of attribute ψ at location ℓ.

Our calculus also employs two sets of expressions: logical expressions ranged over by e and arithmetic expressions, ranged over by w. One of our main aims being to facilitate reasoning about spatially-dependent behavior, these expressions are intended to capture environmental (location-relevant) situations which may affect the behavior of individuals. Expressions e and w, are constructed as follows:

$$\begin{array}{rcl} e & ::= & true \mid \neg e \mid e_1 \wedge e_2 \mid w \bowtie c \\ w & ::= & c \mid \psi@\ell^\star \mid \mathbf{s}@\ell^\star \mid @\ell^\star \mid \mathbf{op}_1(w) \mid \mathbf{op}_2(w_1, w_2) \end{array}$$

where c is a natural number, $\bowtie \in \{=, \leq, \geq\}$ and $\ell^\star \in \mathbf{Loc} \cup \{\mathsf{myloc}\}$. Let us informally consider the introduced expressions. To begin with logical expressions are built using the propositional calculus connectives as well as comparisons between an arithmetic expression w and a constant c, i,e. $w \bowtie c$. Moving on to arithmetic expressions, these include three special expressions interpreted as follows: Expression $\psi@\ell^\star$ is equal to the value of attribute ψ at location $\ell^\star$. Expression $(\mathbf{s}@\ell^\star)$ is equal to the number of individuals of species **s** at location $\ell^\star$ and expression $@\ell^\star$ denotes the total number of individuals of all species at location $\ell^\star$. As specified above, $\ell^\star$ can be an arbitrary location or the special location myloc. This label is employed to bestow individuals the ability to express conditions on the status of their current location no matter where that might be as computation proceeds. Specifically, myloc refers to the actual location of the individual in which the expression appears and it is instantiated to this location when the condition needs to be evaluated (see rule (Cond) in Table 3).

Thus, arithmetic expressions are the set of all expressions formed by arbitrary constants c, quantities $\psi@\ell^\star$, $\mathbf{s}@\ell^\star$, $@\ell^\star$and the usual unary and binary arithmetic operations ($\mathbf{op}_1$ and $\mathbf{op}_2$) on the real numbers. Logical expressions and arithmetic expressions are evaluated within a system environment. The precise definition of the evaluation function is postponed to Tables 1 and 2.

We may now move on to the syntax of PALPS which is given at three levels: (1) the individual level (ranged over by P), (2) the species level (ranged over by R) and (3) the system level (ranged over by S). Their syntax is defined via the following BNF's:

$$\begin{array}{rcl} P & ::= & \mathbf{0} \mid \eta.P \mid \sum_{i \in I} w_i : P_i \mid \mathsf{cond}\,(e_1 \rhd P_1, \ldots, e_n \rhd P_n) \mid C \\ R & ::= & !rep.P \\ S & ::= & \mathbf{0} \mid P{:}[\![\mathbf{s}, \ell]\!] \mid R{:}[\![\mathbf{s}]\!] \mid S_1 \,|\, S_2 \mid S \backslash L \end{array}$$

where $a \in \mathbf{Ch}$, $L \subseteq \mathbf{Ch}$, C ranges over a set of process constants $\mathscr{C}$, each with an associated definition of the form $C \stackrel{\text{def}}{=} P$, where the node P may contain occurrences of C, as well as other constants, and

$$\eta ::= a \mid \overline{a} \mid go\ \ell \mid \surd.$$

Beginning with the *individual* level P, process $\mathbf{0}$ represents the inactive individual, that is, an individual who has ceased to exist. $\eta.P$ describes the individual who first engages in activity η and then behaves as P. Activity η can be an input action on a channel a, written simply as a, a complementary

output action on a channel a, written as $\overline{a}$, a movement action with destination ℓ, $go\,\ell$, or the time-passing action, $\surd$. Actions of the form a, and $\overline{a}$, $a \in \mathbf{Ch}$, are used to model arbitrary activities performed by an individual e.g. eating, preying, observing the environment as well as reproduction. Thus, for example, the actions $prey_{\mathbf{s}}$ and $\overline{prey_{\mathbf{s}}}$ are executed by a prey of population $\mathbf{s}$ and a predator who is preying on individuals of population $\mathbf{s}$. The tick action $\surd$ measures a tick on a global clock and is used to separate the phases/rounds of an individual's behavior. Essentially, the intention is that in any given time unit all individuals perform their available actions possibly synchronizing as necessary until they synchronize on their next $\surd$ action and proceed to their next round.

$\sum_{i\in I} w_i : P_i$ represents the probabilistic choice between processes P_i, $i \in I$. Each alternative is associated with a probability of appearance, which is the value to which the expression w_i evaluates. The conditional process $\mathsf{cond}\,(e_1 \rhd P_1, \ldots, e_n \rhd P_n)$ presents the conditional choice between a set of processes: it behaves as P_i, where i is the smallest integer for which e_i evaluates to true. Finally, process constants provide a mechanism for including recursion in the calculus.

Moving on to the notion of reproduction, to capture the creation of new individuals, we employ the special *species* processes R. R, defined as $!rep.P$, are replicated processes which may continuously receive input through channel rep and creating new instances of process P, where P is a new individual of species R. Such inputs will be provided by individuals in the phase of reproduction via the complementary action $\overline{rep}$.

Finally, population systems are built by composing in parallel located individuals, $P{:}[\![\mathbf{s},\ell]\!]$, where $\mathbf{s}$ and ℓ are the species and the location of the individual, and species $R{:}[\![\mathbf{s}]\!]$, where $\mathbf{s}$ is the name of the species. Finally, $S\backslash L$ models the restriction of the use of channels in set L within S.

As an example, we consider the model described in [8] where a set of individuals live on an $n \times n$ lattice of resource sites and go through phases of reproduction and dispersal. Specifically, the studied model considers a population where individuals disperse in space while competing for a location site during their reproduction phase. They produce an offspring only if they have exclusive use of a location. After reproduction the offspring disperse and continue indefinitely with the same behavior. In PALPS, we may model the described species $\mathbf{s}$ as $R \stackrel{\text{def}}{=} !rep.P$, where

$$P \stackrel{\text{def}}{=} \sum_{\ell \in Neigh(\mathsf{myloc})} \frac{1}{4} : go\,\ell.\surd.\mathsf{cond}\,(\mathbf{s}@\mathsf{myloc} = 1 \rhd P_1;\ \mathsf{true} \rhd \surd.P)$$

$$P_1 \stackrel{\text{def}}{=} p : \overline{rep}.\surd.P_1 + (1-p) : \overline{rep}.\overline{rep}.\surd.P_1$$

We point out that the conditional construct allows us to determine the exclusive use of a location by an individual. The special label myloc is used to illustrate that the location of interest is the actual location of an individual once the individual is placed in a context within a system definition. Furthermore, note that P_1 models the probabilistic production of one or two offspring of the species. During the dispersal phase, an individual moves to a neighboring location which is chosen probabilistically among the four neighboring locations on the lattice of the individual. Then a system containing of two individuals at a location ℓ and one in location ℓ' can be modeled as

$$System \stackrel{\text{def}}{=} (P{:}[\![\ell,\mathbf{s}]\!] | P{:}[\![\ell,\mathbf{s}]\!] | P{:}[\![\ell',\mathbf{s}]\!] | (!rep.P){:}[\![\mathbf{s}]\!]) \backslash \{rep\}.$$

To model a competing species $\mathbf{s}'$ which preys on $\mathbf{s}$, we may define the process $R' \stackrel{\text{def}}{=} !rep'.Q$, where

$$
\begin{aligned}
Q &\stackrel{\text{def}}{=} \mathsf{cond}\ (\mathbf{s}@\mathsf{myloc} > 1 \rhd \mathit{prey}_s.\surd.Q_1, \mathsf{true} \rhd \surd.Q_2) \\
Q_1 &\stackrel{\text{def}}{=} \overline{\mathit{rep}'}.\surd.Q \\
Q_2 &\stackrel{\text{def}}{=} \mathsf{cond}\ (\mathbf{s}@\mathsf{myloc} > 1 \rhd \mathit{prey}_s.\surd.Q_1, \mathsf{true} \rhd \surd.\mathbf{0})
\end{aligned}
$$

An individual of this species looks for a prey. If it succeeds in locating one, then it produces an offspring. If it fails for two consecutive time units it dies.

2.2 The Semantics

The semantics of PALPS is defined in terms of a structural operational semantics given at the level of configurations of the form (E,S), where E is an *environment* and S is a population system. The environment E is an entity which captures how the various locations of the system are populated. More precisely, $E \subset \mathbf{Loc} \times \mathbf{S} \times \mathbb{N}$, where each pair ℓ and $\mathbf{s}$ is represented in E at most once and where $(\ell, s, m) \in E$ denotes the existence of m individuals of species s at location ℓ. The environment E plays a central role in defining the semantics of the calculus and, in particular, for evaluating expressions. The satisfaction relation for logical expression $\models$ is defined inductively on the structure of a logical expression as shown in Table 2.

Table 1: **The satisfaction relation for logical expressions**

$E \models \mathsf{true}$	always	
$E \models \neg e$	if and only if	$\neg(E \models e)$
$E \models e_1 \wedge e_2$	if and only if	$E \models e_1 \wedge E \models e_2$
$E \models w \bowtie e$	if and only if	$\mathsf{val}\ (E,w) \bowtie e$

The relation $\models$ is straightforward and depends on the evaluation function for arithmetic expressions $\mathsf{val}\ (E,w)$ defined in Table 2.

Table 2: **The evaluation relation for arithmetic expressions**

$\mathsf{val}\ (E,c)$	$=$	c
$\mathsf{val}\ (E,\psi@\ell)$	$=$	ψ_ℓ
$\mathsf{val}\ (E,\mathbf{s}@\ell)$	$=$	$\mathsf{num}(E,\ell,\mathbf{s})$
$\mathsf{val}\ (E,@\ell)$	$=$	$\mathsf{num}'(E,\ell)$
$\mathsf{val}\ (E,\mathbf{op}_1(w))$	$=$	$\mathbf{op}_1(\mathsf{val}\ (E,w))$
$\mathsf{val}\ (E,\mathbf{op}_2(w_1,w_2))$	$=$	$\mathbf{op}_2(\mathsf{val}\ (E,w_1), \mathsf{val}\ (E,w_2))$

The auxiliary functions $\mathsf{num}(E,\ell,\mathbf{s})$ and $\mathsf{num}'(E,\ell)$ compute the number of individuals at location ℓ in environment E of a specific species $\mathbf{s}$ ($\mathsf{num}(E,\ell,\mathbf{s})$) or for all species ($\mathsf{num}'(E,\ell)$) and are defined by $\mathsf{num}(E,\ell,\mathbf{s}) = n$ where $(\ell,\mathbf{s},n) \in E$ and $\mathsf{num}'(E,\ell) = \sum_{s\in S} \mathsf{num}(E,s,\ell)$.

Before we proceed to the semantics we define some additional operations on environments that we will use in the sequel:

Definition 1. *Consider environment E location ℓ and species* **s**.

- $E \oplus (\mathbf{s}, \ell)$ *increases the count of individuals of species* **s** *at location* ℓ *in environment* E *by* 1*:*

$$E \oplus (\mathbf{s}, \ell) = \begin{cases} E' \cup \{(\ell, \mathbf{s}, m+1)\} & \text{if } E = E' \cup \{(\ell, \mathbf{s}, m)\} \text{ for some } m \\ E \cup \{(\ell, \mathbf{s}, 1)\} & \text{otherwise} \end{cases}$$

- $E \ominus (\mathbf{s}, \ell)$ *decreases the count of individuals of species* **s** *at location* ℓ *in environment* E *by* 1*:*

$$E \ominus (\mathbf{s}, \ell) = \begin{cases} E' \cup \{(\ell, \mathbf{s}, m-1)\} & \text{if } E = E' \cup \{(\ell, \mathbf{s}, m)\}, m > 1 \\ E' & \text{if } E = E' \cup \{(\ell, \mathbf{s}, 1)\} \\ \bot & \text{otherwise} \end{cases}$$

We may now define the semantics of PALPS, presented in Tables 3 and 4, and given in terms of two transition relations, the nondeterministic relation $\longrightarrow_n$ and the probabilistic relation $\longrightarrow_p$. A transition of the form $(E,S) \xrightarrow{\mu}_n (E',S')$ signifies that configuration (E,S) may execute action μ and become (E',S') whereas a transition of the form $(E,S) \xrightarrow{w}_p (E',S')$ signifies that configuration (E,S) may evolve into configuration (E',S') with probability w. Whenever the type of the transition is irrelevant to the context we write $(E,S) \xrightarrow{\alpha} (E',S')$ to denote that either $(E,S) \xrightarrow{\mu}_n (E',S')$ or $(E,S) \xrightarrow{w}_p (E',S')$. Action μ appearing in the nondeterministic relation may have one of the following forms:

- $a@\ell$ and $\overline{a}@\ell$ denote the execution of actions a and $\overline{a}$ respectively at location ℓ.
- τ denotes the internal action. This may arise when two complementary actions take place at the same location or when a move or a prey action take place. We are not interested in the precise location of internal actions, thus, this information is omitted.
- $\surd$ denotes the time passing action.

The rules of Table 3 prescribe the semantics of located individuals in isolation. The first four axioms define nondeterministic transitions, the fifth axiom defines a probabilistic transition, and the last two rules refer to both the nondeterministic and the probabilistic case. All rules are concerned with the evolution of the individual in question and the effect of this evolution to the system's environment. A key issue in the enunciation of the rules is to preserve the compatibility of P and E as transitions are executed. We consider each of the rules separately. Axiom (Tick) specifies that a $\surd$-prefixed process will execute the time consuming action $\surd$ and then proceed as P. The state of the new environment depends on the state of P: if $P = \mathbf{0}$ then the individual has terminated its computation and, therefore, it is removed from E (see the definition of E^P) whereas, if $P \neq \mathbf{0}$ then, obviously, E remains unchanged. Axiom (Act) specifies that $\eta.P$ executes action $\eta@\ell$ and evolves to P. Note that the action is decorated by the location of the individual executing the transition to enable synchronization of the action with complementary actions taking place at the same location (see rule (Par2), Table 4). This axiom excludes the case of $\eta = go\,\ell$ which is treated separately in the next axiom. Specifically, according to Axiom (Go), an individual may change its location. This gives rise to action τ and has the expected effect on the environment E. Moving on to Axiom (Prey), this describes that any individual can become the victim of a preying action. This may happen at any point during the lifetime of the individual giving rise to the action $prey_{\mathbf{s}}@\ell$ and causing the individual to terminate with the appropriate changes to the state of the environment. Rule (PSum) expresses the semantics of probabilistic choice: once the probability expressions are evaluated within the environment, the probabilistic action is taken leading to the appropriate continuation: if the resulting state of the individual, namely P_i, is equal to $\mathbf{0}$, then the

Table 3: **Transition rules for individuals**

(Tick)	$(E, \surd.P{:}[\![s,\ell]\!]) \xrightarrow{\surd}_n (E^P, P{:}[\![s,\ell]\!])$	
(Act)	$(E, \eta.P{:}[\![s,\ell]\!] \xrightarrow{\eta@\ell}_n (E^P, P{:}[\![s,\ell]\!])$	$\eta \neq go\,\ell'$
(Go)	$(E, go\,\ell'.P{:}[\![s,\ell]\!]) \xrightarrow{\tau}_n ((E \ominus (s,\ell)) \oplus (s,\ell'), P{:}[\![s,\ell']\!])$	$(\ell,\ell') \in \mathbf{Nb}$
(Prey)	$(E, P{:}[\![s,\ell]\!]) \xrightarrow{prey_s@\ell}_n (E \ominus (s,\ell), \mathbf{0}{:}[\![s,\ell]\!])$	
(PSum)	$(E, \sum_{i\in I} w_i : P_i{:}[\![s,\ell]\!]) \xrightarrow{\mathsf{val}\,(E,w_i@\ell)}_p (E^{P_i}, P_i{:}[\![s,\ell]\!])$	
(Const)	$\dfrac{(E, P{:}[\![s,\ell]\!]) \xrightarrow{\alpha} (E', P'{:}[\![s,\ell]\!])}{(E, C{:}[\![s,\ell]\!]) \xrightarrow{\alpha} (E', P'{:}[\![s,\ell]\!])}$	$C \stackrel{\text{def}}{=} P{:}[\![s,\ell]\!]$
(Cond)	$\dfrac{(E, P_i{:}[\![s,\ell]\!]) \xrightarrow{\alpha} (E', P_i'{:}[\![s,\ell']\!]), E \models e_i@\ell, E \not\models e_j@\ell, j<i}{(E, \mathsf{cond}\,(e_1 \rhd P_1, \ldots, e_n \rhd P_n)) \xrightarrow{\alpha} (E', P_i'{:}[\![s,\ell']\!])}$	

$$\text{where } E^P = \begin{cases} E \ominus (\mathbf{s},\ell) & \text{if } P = \mathbf{0} \\ E & \text{otherwise} \end{cases}$$

individual is removed from the environment E. Note that we write $w@\ell$ for the expression w with all occurrences of myloc substituted by location ℓ: $w@\ell = w[\ell/\mathsf{myloc}]$. Next (Const) express the semantics of process constants in the expected way. Finally, rule (Cond) stipulates that a conditional process may perform an action of continuation P_i assuming that $e_i@\ell$ evaluates to true and all $e_j@\ell$, $j<i$ evaluate to false. Similarly to $w@\ell$, $e@\ell$ is the expression e with all occurrences of myloc substituted by location ℓ.

We may now move on to Table 4 which defines the semantics of system-level operators. The first rule defines the semantics for the replication operator, the next five rules define the semantics of the parallel composition operator, and the last rule deals with the restriction relation.

Thus, according to axiom (Rep), a species process may execute action $rep_{\mathbf{s}}@\ell$ for any location ℓ and create a new individual P of species $\mathbf{s}$ at location ℓ. Next, rules (Par1) - (Par4) specify how the actions of the components of a parallel composition may be combined. Note that the symmetric versions of these rules are omitted. According to (Par1), if a component may execute a nondeterministic transition and no probabilistic transition is enabled by the other component (denoted by $(E,S_2) \not\xrightarrow{}_p$), then the transition may take place. If the parallel components may execute complementary actions, then they may synchronize with each other producing action τ (rule (Par2)). If both components may execute probabilistic transitions then they may proceed together with probability the product of the two distinct probabilities (rule (Par3)) and, finally, if exactly one of them enables a probabilistic transition then this transition takes precedence over any nondeterministic transitions of the other component (rule (Par4)). Note that in case that the components proceed simultaneously then the environment of the resulting configuration should take into account the changes applied in both of the constituent transitions (rules

Table 4: **Transition rules for systems**

(Rep)	$\dfrac{R = !rep_{\mathbf{s}}.P{:}[\![\mathbf{s}]\!],\ \ell \in \mathbf{Loc}}{(E,R) \xrightarrow{rep_{\mathbf{s}}@\ell}_n (E \oplus (\mathbf{s},\ell), P{:}[\![\mathbf{s},\ell]\!] \mid R)}$
(Par1)	$\dfrac{(E,S_1) \xrightarrow{\mu}_n (E',S_1'),\ (E,S_2) \not\xrightarrow{}_p}{(E,S_1 \mid S_2) \xrightarrow{\mu}_n (E',S_1' \mid S_2)}$
(Par2)	$\dfrac{(E,S_1) \xrightarrow{a@\ell}_n (E_1,S_1'),\ (E,S_2) \xrightarrow{\overline{a}@\ell}_n (E_2,S_2')}{(E,S_1 \mid S_2) \xrightarrow{\tau}_n (E \otimes (E_1,E_2), S_1' \mid S_2')}$
(Par3)	$\dfrac{(E,S_1) \xrightarrow{w_1}_p (E_1,S_1'),\ (E,S_2) \xrightarrow{w_2}_p (E_2,S_2')}{(E,S_1 \mid S_2) \xrightarrow{w_1 \cdot w_2}_p (E \otimes (E_1,E_2), S_1' \mid S_2')}$
(Par4)	$\dfrac{(E,S_1) \xrightarrow{w}_p (E,S_1'),\ (E,S_2) \not\xrightarrow{}_p}{(E,S_1 \mid S_2) \xrightarrow{w}_p (E,S_1' \mid S_2)}$
(Time)	$\dfrac{(E,S_1) \xrightarrow{\surd}_n (E,S_1'),\ (E,S_2) \xrightarrow{\surd}_n (E,S_2')}{(E,S_1 \mid S_2) \xrightarrow{\surd}_n (E,S_1' \mid S_2')}$
(Res)	$\dfrac{(E,S) \xrightarrow{\alpha} (E',S'),\ \alpha \notin \{a@\ell, \overline{a}@\ell \mid a \in L\}}{(E,S\backslash L) \xrightarrow{\alpha} (E',S')\backslash L}$

(Par2) and (Par4). This is implemented by $E \otimes (E_1,E_2)$ as follows:

$$\begin{aligned}
E \otimes (E_1,E_2) \;=\; & \{(\ell,\mathbf{s},m) \mid (\ell,\mathbf{s},m) \in E \cap E_1 \cap E_2\} \\
\cup\; & \{(\ell,\mathbf{s},m) \mid (\ell,\mathbf{s},m) \in E, (\ell,\mathbf{s},m-1) \in E_1, (\ell,\mathbf{s},m+1) \in E_2\} \\
\cup\; & \{(\ell,\mathbf{s},m-1) \mid (\ell,\mathbf{s},m) \in E \cap E_i, (\ell,\mathbf{s},m-1) \in E_{3-i}, i \in \{1,2\}\} \\
\cup\; & \{(\ell,\mathbf{s},m-2) \mid (\ell,\mathbf{s},m) \in E (\ell,\mathbf{s},m-1) \in E_1 \cap E_2\} \\
\cup\; & \{(\ell,\mathbf{s},m+1) \mid (\ell,\mathbf{s},m) \in E \cap E_i, (\ell,\mathbf{s},m+1) \in E_{3-i}, i \in \{1,2\}\}
\end{aligned}$$

Next, rule (Time) defines that parallel processes must synchronize on $\surd$ actions, thus allowing one tick of time to pass and all processes to proceed to their next round. Finally, rule (Res) defines the semantics of the restriction operator in the usual way.

As a final note, we observe that given a system S, the semantical rules are applied to the initial configuration (E,S) where $(\ell,\mathbf{s},m) \in E$ if and only if S contains exactly m individuals of species $\mathbf{s}$ located at ℓ. In general, we say that E is *compatible* with S whenever $(\ell,\mathbf{s},m) \in E$ if any only if S contains exactly m individuals of species $\mathbf{s}$ located at ℓ. It is possible to prove the following lemma by structural induction on S [1].

Lemma 1. *Whenever $(E,S) \xrightarrow{\alpha} (E',S')$ and E is compatible with S, then E' is also compatible with S'.*

2.3 Model Checking PALPS

Model-checking of PALPS processes may be implemented via an instantiation of the PCTL logic [6]. The instantiation involves the adoption of PALPS logical expressions as the atomic propositions of the

logic. Specifically, the syntax of the PCTL instantiation that we consider, is given by the following grammar where Φ and ϕ range over PCTL state and path formulas, respectively, $p \in [0,1]$ and $k \in \mathbb{N}$.

$$\begin{array}{rcl} \Phi & := & \mathit{true} \mid e \mid \neg\Phi \mid \Phi \wedge \Phi' \mid \mathsf{P}_{\bowtie p}[\phi] \\ \phi & := & \mathsf{X}\Phi \mid \Phi \mathsf{U}^k \Phi \mid \Phi_1 \mathsf{U} \Phi \end{array}$$

In the syntax above, we distinguish between state formulas Φ and path formulas ϕ, which are evaluated over states and paths, respectively. A state formula is built over PALPS logical expressions and the construct $\mathsf{P}_{\bowtie p}[\phi]$. Intuitively, a configuration s satisfies property $\mathsf{P}_{\bowtie p}[\phi]$ if for any possible execution beginning at the configuration, the probability of taking a path that satisfies the path formula ϕ satisfies the condition $\bowtie p$. Path formulas include the X (next), U^k (bounded until) and U (until) operators, which are standard in temporal logics. Intuitively, $\mathsf{X}\Phi$ is satisfied in a path if the next state satisfies path formula Φ, $\Phi_1 \mathsf{U}^k \Phi_2$ is satisfied in a path if Φ_1 is satisfied continuously on the path until Φ_2 becomes true within k time units (where time units are measured by $\surd$ events in PALPS) and $\Phi_1 \mathsf{U} \Phi_2$ is satisfied if Φ_2 is satisfied at some point in the future and Φ_1 holds up until then.

The semantics of PCTL are defined over Markov Decision Processes (MDPs), a type of transition systems that combine probabilistic and nondeterministic behavior. It is not difficult to see that the operational semantics of PALPS gives rise to transition systems that can easily be translated to MDPs [1]. For the details of the semantics and the model checking algorithm we refer the reader to [15].

As a final note we observe that in order to check the satisfaction of PCTL properties by PALPS processes it is sufficient to restrict our attention to the E component of each configuration (E,S). This is due to the fact that E is the only information required in order decide the satisfaction of logical expressions by configurations (see Tables 1 and 2).

3 Examples

During the last few decades, the theory of metapopulations has been an active field of research in Ecology and it has been extensively studied by conservation scientists and landscape ecologists to analyze the behavior of interacting populations and to determine how the topology of fragmented habitats may influence various aspects of these systems such as local and global population persistence and species evolution. The notion of a metapopulation refers to a group of distinct populations of the same species residing on a fragmented habitat or, a so-called set of patches, and cycle in relative independence through their life cycle while interacting with other populations and colonizing previously unoccupied locations through dispersal. It has been observed that while populations of a metapopulation may go extinct as a consequence of demographic stochasticity, the metapopulation as a whole is often stable because immigrants from another population are likely to re-colonize habitat which has been left open by the extinction of another population or because immigration to a small population may rescue that population from extinction. Indeed the process of dispersal is of vital importance in metapopulations. It affects the long-term persistence of populations, the coexistence of species and genetic differentiation between subpopulations and understanding this process is essential for obtaining a good understanding of the behavior of metapopulations. The evolution of dispersal has received much attention by scientists and it has been studied in connection to various parameters such as the connectivity of the habitat on which a metapopulation exists, patch quality and local dynamics.

In this section, we describe two examples relating to metapopulation dispersal through which we illustrate how our calculus can be used to construct models of this phenomenon.

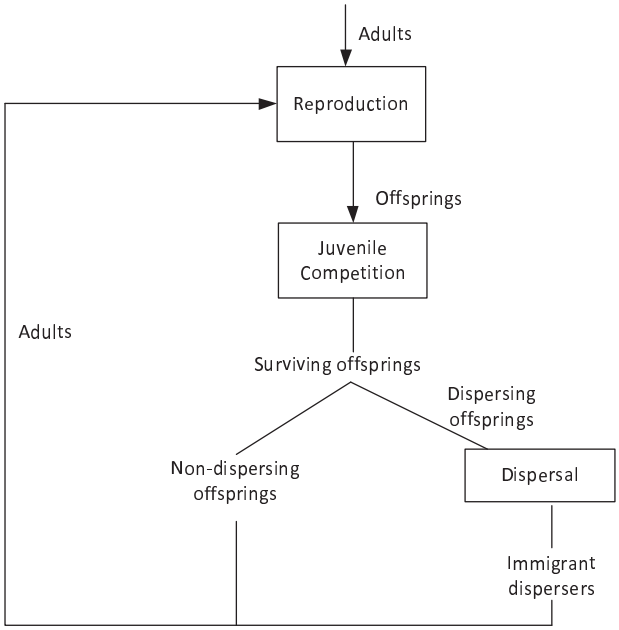


Figure 1: The sequence of events in the lifetime of a dispersing species

Example 1. The first example we consider is motivated by the spatially-explicit, individual-based model of [30]. In this work the authors construct a fairly simple model of metapopulation dispersal which departs from previous works in that, unlike previous models of metapopulation dispersal which tended to be deterministic and at the level of population densities, the model constructed is both stochastic and individual-based.

According to this study, a set of genotypes co-exist within a habitat which differ only in their propensity to disperse. The metapopulation is composed of $n \times n$ subpopulations inhabiting a set of patches arranged on a square lattice with cyclic boundaries, so that individuals leaving the "top" or "right-side" of the world reappear on the "bottom" or "leftside" respectively and vice versa. Each patch is associated with a so-called patch quality related to the capacity of the patch. The behavior of an individual of the genotypes under study is illustrated diagrammatically in Figure 1. According to this model, an adult individual initially produces λ offspring. Subsequently, a phase of competition takes place between the juveniles of the population of which a fraction survives. Each surviving offspring may disperse according to a probability of dispersal distinct to its genotype. In case it disperses, the neighboring patch it moves to is selected with equal probability among all neighbors. This sequence of events in the behavior of an individual is presented diagrammatically in Figure 1. We point out that the percentage of offspring surviving juvenile competition at patch ℓ is given by $\gamma_\ell = (1+\alpha_\ell \cdot N_\ell)^\beta$, where α_ℓ is the measure of the patch quality, N_ℓ is the number of individuals residing at patch ℓ and β is a constant that relates to the degree of competition.

This metapopulation can be modeled in PALPS as follows. We consider the set of of locations (i, j), $1 \leq i, j \leq n$, where two locations (i, j) and (k, l) are neighbors if they are adjacent on the grid. Finally, let us consider the location attribute α_ℓ as a measure of the quality of the patch at ℓ. Then, genotype i

with probability of dispersal p_i and $\lambda = 3$ can be defined as the species process $R_i = !rep_i.J_i$, where

$$\begin{array}{lcll} A_i & \stackrel{\text{def}}{=} & \overline{rep_i}.\overline{rep_i}.\overline{rep_i}.0 & \text{Adult Individual} \\ J_i & \stackrel{\text{def}}{=} & q_i : S_i + (1-q_i) : \mathbf{0} & \text{Juvenile} \\ S_i & \stackrel{\text{def}}{=} & p_i : D_i + (1-p_i) : \surd.A_i & \text{Surviving Juvenile} \\ D_i & \stackrel{\text{def}}{=} & \sum_{\ell \in Neigh(\mathsf{myloc})} \frac{1}{4} : go\,\ell.\surd.A_i & \text{Dispersing Juvenile} \end{array}$$

and q_i the probability of survival of juvenile competition is given by $q_i = (1 + \alpha_\ell \cdot @\ell)^\beta$. Then a system can be modeled as the composition of the various genotypes as well as the individuals of the initial population under study:

$$System \stackrel{\text{def}}{=} [(R_1{:}[\![1]\!] \mid \ldots R_k{:}[\![k]\!] \mid \prod_{1 \le i \le m_1} A_1{:}[\![\ell_1, 1]\!] \mid \ldots) \backslash \{rep_1, \ldots rep_k\}.$$

Analysis in this model may focus on the effect that the dispersal rates, the degree of competition and/or patch quality may have on the degree of population dispersals.

Example 2. As another more complex example, let us consider a model of wood thrush dispersal, initially proposed in [31] and expanded upon in [25]. This model considers three types of birds: adult breeders, adult floaters, and juveniles which are birds in their first year of life. According to this model, adult breeders produce an offspring at a rate dictated by various system parameters such as clutch size, nest predation and paratisism rates which we denote as r_b. Following reproduction, each individual has a probability of dying before the next time step which is higher in juveniles and adult floaters in comparison to adult breeders. We write q_b, q_j and q_f for the mortality rates of breeders, juveniles and floaters, respectively. If following mortality a habitat patch has more birds than its capacity allows, then dispersal will occur according to a probability determined by the size of the patch and the distance between neighboring patches. This probability is higher in floaters and juveniles in comparison to adult breeders who exhibit a high site fidelity. We write p_b, p_j and p_f for the dispersion rates of breeders, juveniles and floaters, respectively. If a bird reaches a patch with available capacity then it will settle. If not, then it will either attempt to disperse to another patch or it will become a floater depending on whether it has reached its maximum number of dispersal events. Once dispersal has occurred, the juveniles become adults and the model begins another cycle. This sequence of events in the behavior of the populations is presented diagrammatically in Figure 2.

This metapopulation can be modeled in PALPS as follows. We consider the set of of locations and an associated predefined neighbor function as well as a distance function that may be instantiated according to the modeler's preference to capture Euclidean distance or some other function of interest [25]. We also assume the existence of a set of probabilities $\{p_{i,j}\}_{i,j \in \mathbf{Loc}}$ where $p_{i,j}$ represents the probability of dispersal from patch i to patch j. Finally, we introduce the location attribute c_ℓ as a measure of the capacity of patch ℓ. Then, wood thrush species can be modeled by the process $R = !rep.Juv$, where

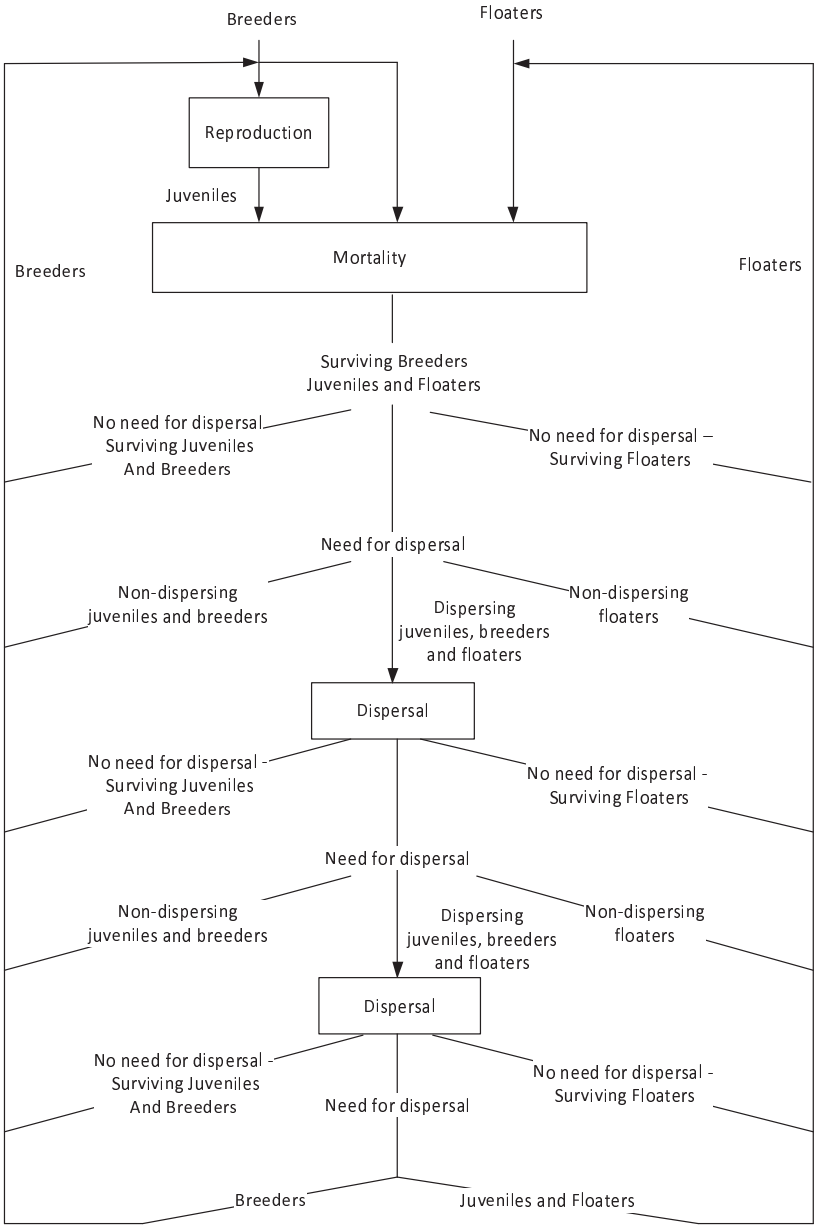


Figure 2: A cycle in the lifetime of the metapopulation

$$
\begin{array}{rcll}
Juv & \stackrel{\text{def}}{=} & q_j : JC_0 + (1-q_j) : \mathbf{0} & \text{Juvenile survival} \\
JC_0 & \stackrel{\text{def}}{=} & \mathsf{cond}\,(@\mathsf{myloc} > c_{\mathsf{myloc}} \rhd JD_0, \mathsf{true} \rhd \surd.AB) & \text{Check patch capacity} \\
JD_0 & \stackrel{\text{def}}{=} & p_j : JA_1 + (1-p_j) : \surd.AB & \text{Decide whether to disperse} \\
JA_1 & \stackrel{\text{def}}{=} & \sum_{\ell \in Neigh(\mathsf{myloc})} p_{\mathsf{myloc},\ell} : go\,\ell.JC_1 & \text{Dispersal attempt 1} \\
JC_1 & \stackrel{\text{def}}{=} & \mathsf{cond}\,(@\mathsf{myloc} > c_{\mathsf{myloc}} \rhd JD_1, \mathsf{true} \rhd \surd.AB) & \text{Check patch capacity} \\
JD_1 & \stackrel{\text{def}}{=} & p_j : JA_2 + (1-p_j) : \surd.AB & \text{Decide whether to disperse} \\
JA_2 & \stackrel{\text{def}}{=} & \sum_{\ell \in Neigh(\mathsf{myloc})} p_{\mathsf{myloc},\ell} : go\,\ell.JC_2 & \text{Dispersal attempt 2} \\
JC_2 & \stackrel{\text{def}}{=} & \mathsf{cond}\,(@\mathsf{myloc} > c_{\mathsf{myloc}} \rhd \surd.Fl, \mathsf{true} \rhd \surd.AB) & \text{Become floater or adult} \\
 & & & \\
AB & \stackrel{\text{def}}{=} & r_b : \overline{r_{bi}}.BS + (1-rb).BS & \text{Breeder reproduction} \\
BS & \stackrel{\text{def}}{=} & q_b : BC_0 + (1-q_b) : \mathbf{0} & \text{Breeder survival} \\
BC_0 & \stackrel{\text{def}}{=} & \mathsf{cond}\,(@\mathsf{myloc} > c_{\mathsf{myloc}} \rhd BD_0, \mathsf{true} \rhd \surd.AB) & \text{Check patch capacity} \\
BD_0 & \stackrel{\text{def}}{=} & p_b : BA_1 + (1-p_b) : \surd.AB & \text{Decide whether to disperse} \\
BA_1 & \stackrel{\text{def}}{=} & \sum_{\ell \in Neigh(\mathsf{myloc})} p_{\mathsf{myloc},\ell} : go\,\ell.BC_1 & \text{Dispersal attempt 1} \\
BC_1 & \stackrel{\text{def}}{=} & \mathsf{cond}\,(@\mathsf{myloc} > c_{\mathsf{myloc}} \rhd BD_1, \mathsf{true} \rhd \surd.AB) & \text{Check patch capacity} \\
BD_1 & \stackrel{\text{def}}{=} & p_b : BA_2 + (1-p_b) : \surd.AB & \text{Decide whether to disperse} \\
BA_2 & \stackrel{\text{def}}{=} & \sum_{\ell \in Neigh(\mathsf{myloc})} p_{\mathsf{myloc},\ell} : go\,\ell.BC_2 & \text{Dispersal attempt 2} \\
BC_2 & \stackrel{\text{def}}{=} & \mathsf{cond}\,(@\mathsf{myloc} > c_{\mathsf{myloc}} \rhd \surd.Fl, \mathsf{true} \rhd \surd.AB) & \text{Floater or adult} \\
 & & & \\
Fl & \stackrel{\text{def}}{=} & q_f : FC_0 + (1-q_f) : \mathbf{0} & \text{Floater survival} \\
FC_0 & \stackrel{\text{def}}{=} & \mathsf{cond}\,(@\mathsf{myloc} > c_{\mathsf{myloc}} \rhd FD_0, \mathsf{true} \rhd \surd.Fl) & \text{Check patch capacity} \\
FD_0 & \stackrel{\text{def}}{=} & p_f : FA_1 + (1-p_f) : \surd.Fl & \text{Decide whether to disperse} \\
FA_1 & \stackrel{\text{def}}{=} & \sum_{\ell \in Neigh(\mathsf{myloc})} p_{\mathsf{myloc},\ell} : go\,\ell.FC_1 & \text{Dispersal attempt 1} \\
FC_1 & \stackrel{\text{def}}{=} & \mathsf{cond}\,(@\mathsf{myloc} > c_{\mathsf{myloc}} \rhd FD_1, \mathsf{true} \rhd \surd.Fl) & \text{Check patch capacity} \\
FD_1 & \stackrel{\text{def}}{=} & p_f : FA_2 + (1-p_f) : \surd.Fl & \text{Decide whether to disperse} \\
FA_2 & \stackrel{\text{def}}{=} & \sum_{\ell \in Neigh(\mathsf{myloc})} p_{\mathsf{myloc},\ell} : go\,\ell.\surd.Fl & \text{Dispersal attempt 2}
\end{array}
$$

As before, the system can be modeled as the composition of the species as well as the various individuals that form the study:

$$
System \stackrel{\text{def}}{=} [(R{:}[\![1]\!] \mid \prod_{1 \le i \le n_b^1} AB{:}[\![\ell_1, 1]\!] \parallel \prod_{1 \le i \le n_j^1} Juv{:}[\![\ell_1, 1]\!] \mid \prod_{1 \le i \le n_f^1} Fl{:}[\![\ell_1, 1]\!] \ldots) \backslash \{rep_1, \ldots rep_k\}.
$$

Varying the model parameters, e.g. the habitat topology, patch quality and dispersal distance, may allow an analysis of the effects of the parameters on patch and metapopulation persistence.

4 Concluding remarks

This paper reports on work in progress towards the development of a process-calculus framework for the spatially-explicit and individual-based modeling of ecological systems. In related work [1] we have also implemented a prototype tool and conducted simulations for the spatially-explicit model of [8]. In future

work we intend to provide optimizations for our tool via an implementation of a spatial extension of the Gillespie simulation algorithm [20] and by taking advantage of concepts developed in process-algebraic frameworks for state-space reduction such as confluence and minimization according to equivalence relations. At the same time it is our intention to enhance the syntax of PALPS to enable a more succinct presentation of systems especially in terms of the multiplicity of individuals. Other possible directions for future work include the adoption of continuous time within the framework as well as the use of dynamic attributes to allow exploring the system while, e.g. patch quality degrades, temperatures increase, etc.

References

[1] M. Antonaki (2012): *A Probabilistic Process Algebra and a Simulator for Modeling Population Systems*. Master's thesis, University of Cyprus.

[2] R. Barbuti, A. Maggiolo-Schettini, P. Milazzo & G. Pardini (2009): *Spatial Calculus of Looping Sequences*. *Electronic Notes in Theoretical Computer Science* 229(1), pp. 21–39.

[3] L. Berec (2002): *Techniques of Spatially Explicit Individual-based Models: Construction, Simulation, and Mean-field Analysis*. *Ecological Modeling* 150, pp. 55–81.

[4] D. Besozzi, P. Cazzaniga, D. Pescini & G. Mauri (2008): *Modelling Metapopulations with Stochastic Membrane Systems*. *BioSystems* 91(3), pp. 499–514.

[5] D. Besozzi, P. Cazzaniga, D. Pescini & G. Mauri (2010): *An Analysis on the Influence of Network Topologies on Local and Global Dynamics of Metapopulation Systems*. *EPTCS* 33, pp. 1–17.

[6] A. Bianco & L. de Alfaro (1995): *Model Checking of Probabilistic and Nondeterministic Systems*. In: *Proceedings of FSTTCS'95*, LNCS 1026, Springer, pp. 499–513.

[7] L. Bioglio, C. Calcagno, M. Coppo, F. Damiani, E. Sciacca, S. Spinella & A. Troina (2011): *A Spatial Calculus of Wrapped Compartments*. *CoRR* abs/1108.3426. Available at `http://arxiv.org/abs/1108.3426`.

[8] A. Brännström & D. J. T. Sumpter (2005): *Coupled Map Lattice Approximations for Spatially Explicit Individual-based Models of Ecology*. *Bulletin of Mathematical Biology* 67(4), pp. 663–682.

[9] L. Cardelli (2005): *Brane Calculi - Interactions of Biological Membranes*. In: *Proceedings of CMSB'04*, LNCS 3082, Springer, pp. 257–278.

[10] L. Cardelli & P. Gardner (2010): *Processes in Space*. In: *Proceedings of CiE'10*, LNCS 6158, Springer, pp. 78–87.

[11] M. Cardona, M. Colomer, A. Margalida, I. Pérez-Hurtado, M. J. Pérez-Jiménez & D. Sanuy (2010): *A P System Based Model of an Ecosystem of the Scavenger Birds*. In: *Proceedings of WMC'09*, LNCS 5957, Springer, pp. 182–195.

[12] F. Ciocchetta & M. L. Guerriero (2009): *Modelling Biological Compartments in Bio-PEPA*. *Electronic Notes in Theoretical Computer Science* 227, pp. 77–95.

[13] F. Ciocchetta & J. Hillston (2009): *Bio-PEPA: a Framework for the Modelling and Analysis of Biochemical Networks*. *Theoretical Computer Science* 410(33-34), pp. 3065–3084.

[14] J. B. Dunning, D. J. Stewart, B. J. Danielson, B. R. Noon, T. L. Root, R. H. Lamberson & E. E. Stevens (1995): *Spatially Explicit Population Models: Current Forms and Future Uses*. *Ecological Applications* 5, pp. 3–11.

[15] V. Forejt, M. Kwiatkowska, G. Norman & D. Parker (2009): *Automated Verification Techniques for Probabilistic Systems*. In: *Proceedings of SFM'11*, LNCS 6659, Springer, pp. 53–113.

[16] S. C. Fu & G. Milne (2004): *A Flexible Automata Model for Disease Simulation*. In: *Proceedings of ACRI'04*, LNCS 3305, Springer, pp. 642–649.

[17] V. Galpin (2009): *Modelling Network Performance with a Spatial Stochastic Process Algebra*. In: *Proceedings of AINA'09*, IEEE Computer Society, pp. 41–49.

[18] L. R. Gerber & G. R. VanBlaricom (2001): *Implications of Three Viability Models for the Conservation Status of the Western Population of Steller Sea Lions (Eumetopias Jubatus)*. Biological Conservation 102, pp. 261–269.

[19] C. A. R. Hoare (1985): *Communicating Sequential Processes*. Prentice-Hall.

[20] M. Jeschke, R. Ewald & A. Uhrmacher (2011): *Exploring the Performance of Spatial Stochastic Simulation Algorithms*. Journal of Computational Physics 230(7), pp. 2562–2574.

[21] M. John, R. Ewald & A. M. Uhrmacher (2008): *A Spatial Extension to the π-Calculus*. Electronic Notes in Theoretical Computer Science 194, pp. 133–148.

[22] D. Kouzapas & A. Philippou (2011): *A Process Calculus for Dynamic Networks*. In: Proceedings of FMOODS/FORTE'11, LNCS 6722, Springer, pp. 213–227.

[23] C. McCaig, R. Norman & C. Shankland (2008): *Process Algebra Models of Population Dynamics*. In: Proceedings of AB'08, LNCS 5147, Springer, pp. 139–155.

[24] R. Milner (1980): *A Calculus of Communicating Systems*. Springer.

[25] E. S. Minor, R. I. McDonald, E. A. Treml & D. L. Urban (2008): *Uncertainty in Spatially Explicit Population Models*. Biological Conservation 141(4), pp. 956–970.

[26] R. G. Pearson & T. P. Dawson (2005): *Long-distance Plant Dispersal and Habitat Fragmentation: Identifying Conservation Targets for Spatial Landscape Planning Under Climate Change*. Biological Conservation 123, pp. 389–401.

[27] G. Păun (2002): *Membrane Computing: An Introduction*. Springer-Verlag.

[28] A. Regev, E.M. Panina, W. Silverman, L. Cardelli & E. Shapiro (2004): *BioAmbients: an Abstraction for Biological Compartments*. Theoretical Computer Science 325(1), pp. 141–167.

[29] C. Tofts (1994): *Processes with Probabilities, Priority and Time*. Formal Aspects of Computing 6, pp. 536–564.

[30] J. M. J. Travis & C. Dytham (1998): *The Evolution of Disperal in a Metapopulation: a Spatially Explicit, Individual-based Model*. Proceedings: Biological Sciences 265(1390), pp. 17–23.

[31] D. L. Urban & H. H. Shugart (1986): *Avian Demography in Mosaic Landscapes: Modeling Paradigm and Preliminary Results*. Wildlife 2000: Modeling Habitat Relationships of Terrestrial Vertebrates, pp. 273–279.

GUBS, a Behavior-based Language for Open System Dedicated to Synthetic Biology

Adrien Basso-Blandin
IBISC lab.
Evry University
abasso@ibisc.univ-evry.fr

Franck Delaplace
IBISC lab.
Evry University
franck.delaplace@ibisc.univ-evry.fr

In this article, we propose a domain specific language, GUBS (Genomic Unified Behavior Specification), dedicated to the behavioural specification of synthetic biological devices, viewed as discrete open dynamical systems. GUBS is a rule-based declarative language. By contrast to a closed system, a program is always a partial description of the behaviour of the system. The semantics of the language accounts the existence of some hidden non-specified actions that possibly alter the behaviour of the programmed devices. The compilation framework follows a scheme similar to automatic theorem proving, aiming at improving synthetic biological design safety.

1 Introduction

Synthetic biology is an emerging scientific field combining the investigative nature of biology with the constructive nature of engineering [22] to design synthetic biological systems. The issue is to devise new functionality/behaviour that does not exist in nature. Then, the field of synthetic biology is looking forward principles and tools to make the biological devices inter-operable and programmable [19]. Synthetic biology projects were first focusing on the design and the improvement of small genetic devices comparable to logical gates for electronic circuits [23, 11]. Recently, projects have attempted to develop large bio-systems integrating different devices with as a long-term goal, the design of *de-novo* synthetic genome [16]. In this endeavour, the computer-aided-design (CAD) environments play a central role by providing the required features to engineer systems: specification, analysis, and tuning [4, 20, 25, 12]. Pioneer applications show the valuable potential of such environments in IGEM competition.

Currently, the design specifies the structural assembly of DNA sequences (biobrick) as in GENOCAD [7]. Although this description is indispensable to provide a finalized specification of devices, the abstraction level seems inappropriate for tackling large bio-systems. The required size of programs for sequence description likely makes the task error-prone and un-come-at-able. In the same way as large softwares cannot be programmed in binary, large biological systems cannot be described as aDNA sequence assembly. Then, scaling up the complexity of the synthetic biological systems needs to complete the structural description by an additional abstract programming layout based on a high-level programming language and harness the automatic conversion of the design specification into a DNA sequence, like compilers. High level programming language for synthetic biology is announced as a key milestone for the second wave of synthetic biology to overcome the complexity of large synthetic system design [22]. Nonetheless, in this domain, language technology is still in its infancy and transforming this vision into a concrete reality remains a daunting challenge.

Pre-Proceedings of 6th Workshop on Membrane Computing and Biologically Inspired Process Calculi
MeCBIC 2012, Pages 27–47.

Such high-level language should describe the devices in term of functionalities, offering the ability to program the behaviour directly instead of the structure supporting this behaviour. Indeed, behaviour specification contributes to accurately document the device by adding its behavioural description, to assess its functionality automatically and formally, notably by generating test-benches from this specification, and to get a relative independence to technology because different biological structures can carry out the same functionality. In this framework, the components are selected and organized automatically or semi-automatically to generate a structural description of the device at compile phase whose behaviour complies with the specified function. A such approach has been already achieved in hardware by using languages as VHDL [1] or VERILOG [24] to overcome the growing complexity of electronic circuits. However, the major difference in synthetic biology relates to the openness of biological system. Thereby, the issue is to propose a behavioural language for open systems. More precisely, GUBS is a rule-based declarative language dedicated to the behavioural specification of *discrete open dynamical systems* for synthetic biology interacting with its environment. GUBS symbolically defines the behaviours to provide a relative independence from structures by postponing the biological component selection at compile phase. Within this framework, the compiler translates the behavioural specification to a structural description of a device whose behaviour carries the functional features defined by a program. The proposed compilation method is inspired by automated theorem proving.

After introducing the main features of GUBS language (Section 2), we define the semantics of GUBS based on hybrid logic. Then, we detail the proof-based principles governing the compilation (Section 3) illustrated with a complete example (Section 4). After a survey of the related works, Section 5, we conclude (Section 6).

2 GUBS language

In this section, we describe the main features of GUBS.

Constant and variables. In GUBS, two kinds of objects are distinguished: the *constants* and the *variables*. The constants designate the pre-defined objects in a corpus of knowledge. In biology, the constants may refer to proteins or genes of interest. For example, the agent *LacZ* refers to LacZ protein or gene. By convention, their name starts with a capital letter. The variables refer to an abstraction of these pre-defined objects and can be potentially replaced (substituted) by any constant. By convention, the variable names start with a minuscule letter.

Agents, attributes and states. The *agents* represent the biological objects. Their different observable *states* characterize their different *behaviours*. The behaviours actually define the different capacities for actions on the state of the other agents. They are characterized symbolically by a set of *attributes* categorizing these different capacities. The real significance of the attributes is a matter of convention depending on the targeted realization (*e.g.*, protein pathways, gene network) and

will be addressed through examples. For instance, the regulatory activity of a gene is observationally related to thresholds of RNA transcripts concentration. At a given threshold, a gene regulates a given set of genes whereas at another one the regulation applies to another set of genes (See Figure 1). The different thresholds define the levels of gene activities leading to different regulatory activities. For a gene G, if we identify three different kinds of regulatory activities, the state of this gene will be defined by three different attributes $\{Low, Mid, High\}$ that characterize symbolically three possible behaviours. For example, $G(Low)$ expresses the fact that agent G is in state Low and then ready for the action corresponding to this attribute. In some cases, a single state is sufficient to qualify the capacity for the action of the agent. Hence, the agent is identified to its capacity. Then, G means that agent G is available.

By contrast, $G(\overline{Low})$ signifies that the state of the agent differs from Low ($\overline{G}$ when an agent has a single capacity). It is worth to point out that, not being in a state defined by an attribute, does not necessarily means that the agent state is in another attribute. Indeed, for open systems the state of the agents could be of any sort that does not necessarily belong to the pre-defined attributes.

Two kinds of relations on attributes are defined: an order, $\prec$, meaning "less capacity than" and an inequality, $\not\approx$, meaning "different capacity than". Then $Low \prec Mid$ implies that the capacity for the action of Mid includes the capacity related to Low. Usually, in gene regulatory model [14], the set of genes regulated at a given level will also be regulated at a higher level. By contrast, in signalling pathways, the phosphorylation of a protein induces a conformational change of the structure leading to a specific signalling potentiality not occurring in the unphosphosrylated conformation. Assuming that $Phos$ and $UnPhos$ respectively represents the phosphorylated and the unphosphorylated conformations of protein P, we have $Phos \not\approx UnPhos$. Then, $P(Phos)$ implies $P(\overline{UnPhos})$ implicitly. The attributes and the relation between attributes will be declared as follows: $G :: \{Low \prec Mid, Mid \prec High\}, P :: \{Phos \not\approx UnPhos\}$. A simple set of attributes replaces the relations if unknown and no specific relation is set between attributes.

Finally, the description of the agent state is extended to a collection of agent states as follows: $g_1 + \ldots + g_n$, meaning that all the agent states, g_i, are observed concomitantly.

Trace, event, and history. A GUBS program describes a behaviour, its interpretation is based on the observations of designed systems. Then, the issue is to formalize the notion of behaviour observation. To this end, we focus on the notion of *trace* that symbolically represents the evolution of some quantities related to the agents of interest by the evolution of these agent states. A trace can be obtained from experiments by establishing a correspondence between measurements of some quantities (*e.g.*, RNA transcript concentration) and attributes of agents. Formally, a trace, $(T_t)_{1 \leq t \leq m}$, is a finite sequence of agent state sets where each set contains the agent states at a given instant. For instance, the evolution of a concentration evolving from Low to $High$ for G may be described by the following trace of 6 instants: $(\{G(Low)\}_1, \{G(Low)\}_2, \{G(Mid)\}_3, \{G(Mid)\}_4, \{G(Mid)\}_5, \{G(High)\}_6)_7$. However, all the events in a trace are not necessarily relevant with regards to the behaviour description. For example, if we focus on the evolution from Low to $High$ for G , only three events are relevant for the behaviour description: $G(Low), G(Mid), G(High)$; without accounting the inter-

mediary evolution stages occurring between. Then, the behaviour recognition always emphasizes the key events in a trace entailing its contraction to show their succession. Such a contracted series is called a *consistent history* of the expected behaviour. Generally speaking, an history is related to a *chronological division* of a trace into periods where the events of a period represent all the agent states occurring at each instant. Then, an history is a sequence of these event sets. Given a trace $(T_t)_{0\leq t\leq m}$, and a chronological division, $(d_i)_{1\leq i\leq n}$, corresponding to a sequence of the starting dates for each period, the history is a sequence of agent states occurring in each period, $(H_i)_{1\leq i<n}$, such that each $H_i = \bigcup_{d_i\leq t<d_{i+1}} T_t$. Hence, a consistent history is purposely made to point the characteristic event steps of a behaviour description out.

In the previous example, a chronological division[1] of the trace leading to an history consistent with the expected evolution from *Low* to *High* for G is $(1,3,6,7)$ which corresponds to following discrete time-intervals $([1,2],[3,5],[6,6])$. The resulting history is: $(\{G(Low)\},\{G(Mid)\},\{G(High)\})$. Notice that $(1,2,4,7)$ also fits. However, the chronological division $(1,3,7)$ leads to an inconsistent history because the level *Mid* is not seen as an intermediary event in the history (See also Figure 1 depicting the trace and consistent history of the dependences). The formal definition of the consistency in the scope of the semantics will be given in Section 2.1.

Behavioral dependence and observation spot. A behavioural dependence identifies a relation between behaviours as a causal relation on events. Basically, the dependences should define the control of agents on another. However, the definition of the causality also needs to tackle the openness of a system by adapting it to this context. A seminal definition of the causality, proposed by Hume [17], is formulated in terms of regularity on events: "[we may define] a cause to be an object, followed by another, and where all the objects similar to the first are followed by objects similar to the second". Although this definition appropriately characterizes the notion of control, the openness of the system implies to account the environment actions that possibly alter the causal dependence chain. For example, a programmed activation $G_1 \xrightarrow{+} G_2$ may be contradicted by an existing inhibition $G_3 \xrightarrow{-} G_2$ addressing the same target gene G_2. Hence, while G_1 is active, it may appear that G_2 will not be active because the regulatory strength of G_3 is greater than the regulatory strength of G_1, contradicting the expected activation by a hidden inhibition. Hence, pushed to the limit, this consideration prevents the ability to describe any behaviour causally because any programmed action can be unexpectedly preempted by an external one.

However, by assuming that the design always describes a new functionality which is not observed naturally, the effect becomes the event indicating the effectiveness of a causal relation. As no cause external to the description can trigger the effect, the over-determination by unknown causes is prevented, then insuring that the program is the sole device entailing the expected effect in the biological system. Hence, the definition of the causal dependence will be governed by the effect leading to the following definition of the dependence: *"if effect e would occur then c occurs"*. Moreover, the scope of future (resp. past) is narrowed to a *closest future (resp. past) period*, representing the fact that a response is always expected in a given delay. Notice that, the proposed

[1] Step 7 is inserted as an extra step to comply with the definition of the chronological division.

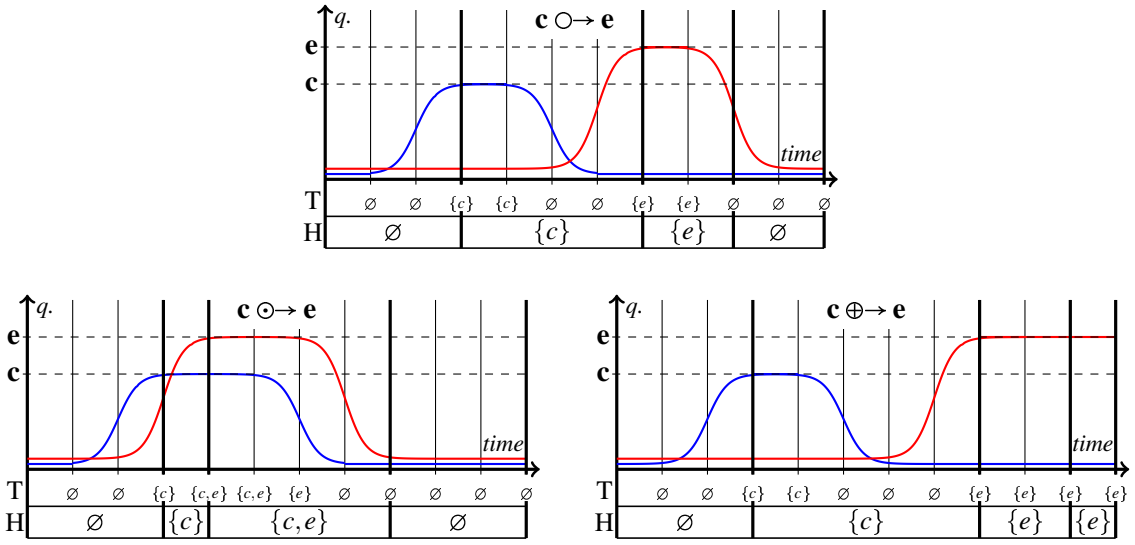


Figure 1: The curves represent the typical behaviours of the causal dependences based on the time evolution of a quantity (q) related to agents c and e (*e.g.*, RNA transcript for gene regulation). The symbolic agent states c and e are here both associated to the maximal threshold of the quantity. The symbolic trace (T) is issued from a periodic sampling of the evolution by identifying whether c or e occur. A consistent history (H) complying to a causal dependence definition is represented below the trace. The first graphic illustrates the normal causality: $c \bigcirc\!\!\rightarrow e$, the second the persistent: $c \odot\!\!\rightarrow e$ and the third the remanent one: $c \oplus\!\!\rightarrow e$.

definition circumvents the afore mentioned problem illustrated by the hidden inhibition because if the effect does not occur the question of the existence of a cause is meaningless. This definition is somehow equivalent to the causal claims proposed by Lewis [18] in terms of counter-factual conditionals, *i.e.*, "If c had not occurred, e would not have occurred".

Three behavioural dependences are defined in GUBS: the *normal* denoted by $\bigcirc\!\!\rightarrow$, *persistent* by $\odot\!\!\rightarrow$, and *remanent* by $\oplus\!\!\rightarrow$. Informally, for normal dependence the cause precedes the effect providing the effect is observed; for persistent dependence the cause still precedes the effect but it is maintained while the effect is observed; and for remanent dependence, the effect is maintained despite the cause has disappeared. These dependences symbolize common biological interactions. For instance, in genetic engineering, the recombination enables the emergence of a regulated gene or an hereditary trait permanently. A such mechanism typifies the remanent dependence in biology. The relations between gene expression at steady state are symbolized by persistent dependence. The behavioural dependences are defined as follows (see Section 2.1 for their formalization):

- $c \bigcirc\!\!\rightarrow e$: if e occurs then c occurs in the closest past.
- $c \odot\!\!\rightarrow e$: if e occurs then c occurs in the closest past and also currently.

- $c \oplus\!\!\rightarrow e$: if e occurs then, either e occurs in the closest past or the dependence complies to the property of the normal dependence.

Figure 1 exemplifies the correspondence between experimental traces, symbolic traces and the history for the causal dependences. All the dependences are extended to a set of causes and a set of consequences, *i.e.*, $c_1 + \ldots + c_n \bigcirc\!\!\rightarrow e_1 + \ldots + e_m$. For example, let us define the activation and the inhibition as follows: $g_1 \xrightarrow{+} g_2 \equiv g_1 \odot\!\!\rightarrow g_2, \overline{g}_1 \bigcirc\!\!\rightarrow \overline{g}_2$ and $g_1 \xrightarrow{-} g_2 \equiv \overline{g}_1 \odot\!\!\rightarrow g_2, g_1 \bigcirc\!\!\rightarrow \overline{g}_2$, the program depicting a negative regulatory circuit with two genes, *i.e.*, $g_1 \xrightarrow{+} g_2, g_2 \xrightarrow{-} g_1$, is: $\{g_1 \odot\!\!\rightarrow g_2, \overline{g}_1 \bigcirc\!\!\rightarrow \overline{g}_2, \overline{g}_2 \odot\!\!\rightarrow g_1, g_2 \bigcirc\!\!\rightarrow \overline{g}_1\}$.

The *observation spots* describe the set of observations expected in a trace. For instance, observing that gene G is at level high is written $Obs{::}G(High)$. As the activation of a dependence lies on the observation of the effect, the observation spot is used to determine which effects must be necessarily observed. For example, in the negative regulatory circuit, the characteristic observation spots are: $obs_1{::}g_1 + \overline{g}_2, obs_2{::}\overline{g}_1 + g_2$.

Compartment & Context. A compartment encloses a set of dependences making them local to the compartment. For instance, $C\{g_1 \bigcirc\!\!\rightarrow g_2\}$ describes a normal dependence occurring in compartment C. The compartments are hierarchically organized and all the compartments are included in another except for the outermost one. Although the compartments directly refer to the compartmentalized cellular organization (*e.g.*, nucleus, mitochondria), they are also used to emphasize the isolation of some interactions by syntactically enclosing the dependences into a compartment. $C.s$ refers to an agent state in compartment C.

A context refers to a stimulus acting on the system, as environmental conditions or external signalling. The application of a context c to a set of dependences b is written $[c]b$ where c is either a variable or a constant. This means that dependences of b are triggered when the context c is present. For instance, recently Ye et al. [26] explore the opto-genetics signalling to control the expression of target transgenes. The blue-light induces the expression of transgene (tg) via a signalling cascade leading to the binding of NFAT transcription factor to a specific promoter (PNFAT). The following program using a context summarizes the process: $[\text{BlueLight}]\{\text{NFAT} \odot\!\!\rightarrow tg\}$. A context can be decomposed to several contexts, $[k_1, \ldots, k_n]b$, meaning that all the conditions must be met to trigger the dependences of b. The interpretation is equivalent to a context cascading, $[k_1][k_2]\ldots[k_n]b$. Moreover, the observation spots and the attribute definition are context insensitive.

2.1 Semantics of GUBS

The interpretation of GUBS is a formula such that the set of all the models validating it defines all the possible histories complying to the programmed behaviour. The interpretation is based on multi-modal hybrid logic with the "Always" operator, $\mathcal{H}(\mathbf{A}, @)$.

Hybrid logic. In what follows, we recall the formal syntax and semantics of hybrid logic. The hybdrid logic [5, 6] offers the possibility to denominate worlds by new symbols called *nominals*.

They will be used in satisfaction modal operators $@_a$; the formula $@_a\phi$ asserts that ϕ is satisfied at the unique point named by the nominal a identifying a particular truth values of a formula at this point. Given a set of propositional symbol, PROP, a set of relational symbol REL, and a set of nominal NOM disjoint to PROP, a set of well formed formula in the signature of $\langle \mathsf{PROP}, \mathsf{NOM}, \mathsf{REL} \rangle$ is defined as follows:

$$\phi ::= \top \mid p \mid a \mid \neg\phi \mid \phi \wedge \phi \mid @_a\phi \mid \langle k \rangle \phi \mid \langle k \rangle^- \phi \mid \mathbf{A}\phi.$$

with $p \in \mathsf{PROP}, a \in \mathsf{NOM}$ and $k \in \mathsf{REL}$. Moreover, the syntax is extended to other logical operators classically [2]: $\bot, \vee, \rightarrow, [k], \mathbf{E}$.

The interpretation is carried out using the Kripke model satisfaction definition (Table 2.1). $\mathcal{M}, w \Vdash \phi$ is interpreted as the satisfaction of a formula ϕ by a model $\mathcal{M}$ at world w where $\Vdash$ stands for the realizability relation (*i.e.*, "is a model of"). A model *validates* a formula, denoted by $\mathcal{M} \Vdash \phi$, if and only if it is satisfied for all the worlds of the model (*i.e.*, $\forall w \in \mathrm{Dom}\, \mathcal{M} : \mathcal{M}, w \Vdash \phi$).

Definition 1 (Kripke model)**.** *A Kripke model is a structure* $\mathcal{M} = \langle W, (R_k)_{k\in\tau}, V \rangle$ *where* $W = Dom\, \mathcal{M}$ *is a non-empty set of* worlds*,* $\tau \subseteq REL$ *a subset of relational symbols denoting the* modalities*,* $R_k \subseteq W \times W, k \in \tau$ *a relation of accessibility,* $V : (PROP \cup NOM) \rightarrow \mathbf{2}^W$ *an interpretation attributing to each nominal and propositional variable a set of worlds such that any nominal addresses one world at most (i.e.,* $\forall a \in NOM : |V(a)| \leq 1$*).*

By convention, R stands for the union of the accessibility relation, $R = (\bigcup_{k\in\tau} R_k)$.

A *modal theory* of a model $\mathcal{M}$ regarding to a set of formulas F, $\mathrm{TH}_F(\mathcal{M})$, is the set of formulas of F validated by $\mathcal{M}$, *i.e.*, $\mathrm{TH}_F(\mathcal{M}) = \{\phi \in F \mid \mathcal{M} \Vdash \phi\}$. $\mathsf{KS}(\phi)$ denotes the set of all models validating ϕ, *i.e.*, $\mathsf{KS}(\phi) = \{\mathcal{M} \mid \mathcal{M} \Vdash \phi\}$.

$\mathcal{M}, w \Vdash \top$	iff *true*	$\mathcal{M}, w \Vdash @_a\phi$	iff $\exists w' \in W : \mathcal{M}, w' \Vdash \phi$ and $\{w'\} = V(a)$
$\mathcal{M}, w \Vdash a$	iff $w \in V(a), a \in \mathsf{NOM} \cup \mathsf{PROP}$	$\mathcal{M}, w \Vdash \langle k \rangle \phi$	iff $\exists w' \in W : \mathcal{M}, w' \Vdash \phi$ and wR_kw'
$\mathcal{M}, w \Vdash \neg\phi$	iff $\mathcal{M}, w \nVdash \phi$	$\mathcal{M}, w \Vdash \langle k \rangle^- \phi$	iff $\exists w' \in W : \mathcal{M}, w' \Vdash \phi$ and $w'R_kw$
$\mathcal{M}, w \Vdash \phi_1 \wedge \phi_2$	iff $\mathcal{M}, w \Vdash \phi_1$ and $\mathcal{M}, w \Vdash \phi_2$	$\mathcal{M}, w \Vdash \mathbf{A}\phi$	iff $\forall w' \in W : \mathcal{M}, w' \Vdash \phi$

Table 1: Hybrid logic interpretation.

Semantics. A GUBS program is interpreted by a hybrid logic formula where the modal operators characterize here the temporal observations on an history: $[\ ]$ means *"observed in all the closest futures"* and $\langle\ \rangle$ means *"observed in a possible closest future at least"* (resp. $\langle\ \rangle^-, [\ ]^-$ for the closest past). Moreover, we assume that the accessibility relations, $(R_k)_{k\in\tau}$, are indexed by the non empty parts of the set of all the contexts of a program P, denoted by K_P (*i.e.*, $\tau = \mathbf{2}^{K_P} \setminus \{\varnothing\}$). Then, a non-empty set of contexts ,$\varnothing \subset K \subseteq K_P$, is a modality, *i.e.*, $\langle K \rangle, [K]$ with $\langle\ \rangle = \langle \varnothing \rangle$ by convention.

[2] $\bot = \neg\top, \psi \vee \phi = \neg(\neg\psi \wedge \neg\phi), \psi \rightarrow \phi = \neg(\psi \wedge \neg\phi), [k]\phi = \neg\langle k \rangle \neg\phi, \mathbf{E}\phi = \neg\mathbf{A}\neg\phi.$

Let $\langle \mathsf{W}, \bullet, \Lambda \rangle$ be the set of words W with the concatenation operation and the neutral element, the empty word Λ and $\mathsf{F}_{\mathcal{H}}$ the set of well-formed formulas of $\mathcal{H}(\mathbf{A}, @)$, the semantics is defined by four functions: $[\![.]\!] : \mathsf{P} \to \mathsf{F}_{\mathcal{H}}, [\![.]\!]_P : \mathsf{P} \to \mathsf{W} \to \mathbf{2}^{\mathsf{W}} \to \mathsf{F}_{\mathcal{H}}, [\![.]\!]_B : \mathsf{B} \to \mathsf{W} \to \mathsf{F}_{\mathcal{H}}, [\![.]\!]_R : \mathsf{R} \to \mathsf{W} \to \mathsf{F}_{\mathcal{H}}$, where $\mathsf{P}, \mathsf{B}, \mathsf{R}$ respectively stand for the set of GUBS programs, the set of agent state set and the set of relations on attributes. $[\![.]\!]$ initiates the interpretation. Table 2.1 defines these functions. For

$$
\begin{array}{ll}
[\![\{b\}]\!] & = \mathbf{A}\left([\![b]\!]_P(\Lambda)(\varnothing)\right) \\
 & \\
[\![\varepsilon]\!]_P(C)(K) & = \top \\
[\![b_1, b_2]\!]_P(C)(K) & = [\![b_1]\!]_P(C)(K) \wedge [\![b_2]\!]_P(C)(K) \\
[\![s_1 \circ\!\!\rightarrow s_2]\!]_P(C)(K) & = [\![s_2]\!]_B(C) \to \langle K \rangle^{-}\left([\![s_1]\!]_B(C)\right) \\
[\![s_1 \odot\!\!\rightarrow s_2]\!]_P(C)(K) & = [\![s_2]\!]_B(C) \to \left([\![s_1]\!]_B(C) \wedge \langle K \rangle^{-}\left([\![s_1]\!]_B(C)\right)\right) \\
[\![s_1 \oplus\!\!\rightarrow s_2]\!]_P(C)(K) & = [\![s_2]\!]_B(C) \to \left((\langle\,\rangle^{-}[\![s_2]\!]_B(C)) \vee (\langle K \rangle^{-}[\![s_1]\!]_B(C))\right) \\
[\![g_1, \cdots, g_n : \{r_1, \cdots, r_m\}]\!]_P(C)(K) & = \bigwedge_{i=1}^{n} \bigwedge_{j=1}^{m} [\![r_j]\!]_R(C.g_i) \\
[\![l :: s]\!]_P(C)(K) & = @_l [\![s]\!]_B(C) \\
[\![C'\{b\}]\!]_P(C)(K) & = [\![b]\!]_P(C.C')(K) \\
[\![[K]\{b\}]\!]_P(C)(K') & = [\![b]\!]_P(C)(K \cup K') \\
 & \\
[\![s_1 + \ldots + s_n]\!]_B(C) & = \bigwedge_{i=1}^{n} [\![s_i]\!]_B(C) \\
[\![C'.s]\!]_B(C) & = [\![s]\!]_B(C.C') \\
[\![g(a)]\!]_B(C) & = C.g_a \\
[\![g(\overline{a})]\!]_B(C) & = \neg C.g_a \\
[\![g]\!]_B(C) & = C.g \\
[\![\overline{g}]\!]_B(C) & = \neg C.g \\
 & \\
[\![a_1 \prec a_2]\!]_R(g) & = g_{a_2} \to g_{a_1} \\
[\![a_1 \not\approx a_2]\!]_R(g) & = g_{a_1} \to \neg g_{a_2} \wedge g_{a_2} \to \neg g_{a_1} \\
[\![a]\!]_R(g) & = \top
\end{array}
$$

Table 2: Semantics of GUBS. In the definition, a represents an attribute, b a behaviour, g an agent, s a set of agent states or an agent state, r a relation on attributes, C a compartment, K a set of contexts and b a set of behaviours (*i.e.*, contexts, compartments, dependences, attributes, observation spots).

instance, the program of the negative regulatory network, $\{g_1 \odot\!\!\rightarrow g_2, \overline{g}_1 \circ\!\!\rightarrow \overline{g}_2, \overline{g}_1 \odot\!\!\rightarrow g_2, g_1 \circ\!\!\rightarrow \overline{g}_2, obs_1 :: g_1 + \overline{g}_2, obs_2 :: \overline{g}_1 + g_2\}$, is translated into the following formula:

$$
\begin{array}{l}
\mathbf{A}\big(\ g_2 \to ((\langle\,\rangle^{-} g_1) \wedge g_1) \wedge \neg g_2 \to (\langle\,\rangle^{-} \neg g_1) \wedge g_2 \to (((\langle\,\rangle^{-} \neg g_1) \wedge \neg g_1) \wedge \neg g_2 \to (\langle\,\rangle^{-} g_1) \wedge \\
\quad @_{obs_1}(g_1 \wedge \neg g_2) \wedge @_{obs_2}(\neg g_1 \wedge g_2)
\end{array}
$$

Consistent history. Now, we formally define the consistency of the history with regards to models. An history is assimilated to a path in a model ending by a world labelled with an observation spot label. The set of Kripke-models validating the interpretation of a program P, $\mathsf{KS}([\![P]\!])$, not only contains all the consistent histories, but also the possible histories corresponding to behavioural alterations due to external perturbations. Thus, the compilation generates a device such that all the models validating its interpretation integrate all the observations related to the program, including the consistent and the inconsistent ones.

More precisely, the consistency lies on the identification of the largest number of "relevant" events characterizing a complete causal chain described in a program. As an history is also a model, a consistent history should validate the interpretation of the complete causal chain. The dependence formula set F_P of a program P corresponds to a set of formulas where each formula is the interpretation of a dependence taken separately with the attributes related to the involved agents. By definition of the semantics, any model validating the interpretation of a program also validates each formula of this set. The consistency of an history is then based on the validated formulas of this set by this history. *An history $\mathcal{M}_H$ is consistent for P if and only if no other modal theory of histories based on F_P (i.e., $TH_{F_P}(\mathcal{M})$ with $\mathcal{M}$ as an history), ending with the same labelled world includes the modal theory of this history (i.e., $TH_{F_P}(\mathcal{M}_H) \nsubseteq TH_{F_P}(\mathcal{M})$).*

3 Compilation

At compile phase, a program is transformed to a structure (*e.g.*, a DNA sequence) while inserted in a vector cell, should behave according to the programmed specification. The structure will result to an assembly of several devices stored in a library of components (*e.g.*, parts registry). As the design relates here to a behavioural/functional description, we need to bridge the gap between structural and functional description. This stage is called the *functional synthesis*. The issue is to select a set of components whose assembly preserves the behaviour of the program. To achieve this goal, a GUBS program is associated to each component to describe its behaviour. Thereby, the component assembly corresponds to a program assembly preserving the behaviour of the compiled program. Preserving a behaviour is laid on a property called the *behavioural inclusion* formalizing the fact that the characteristic observational traits of the specified function must be recognized in traces related to the device experiments. In other words, we can exhibit histories consistent with the programmed behaviour from histories consistent with the device behaviour description. The behavioural inclusion is defined from the interpretation of the programs, as a logical consequence (Definition 2).

Definition 2 (Behavioral inclusion)**.** *A program Q* behaviourally includes *another program P, if and only if the interpretation of the latter is a logical consequence of the interpretation of the former:*

$$P \Subset Q \triangleq \forall \mathcal{M} : \mathcal{M} \Vdash [\![Q]\!] \Longrightarrow \mathcal{M} \Vdash [\![P]\!] .$$

The behavioural inclusion is a pre-order[3] such that the empty program, denoted by ε, is a minimum, meaning that a program with no behaviour can be observed in all traces. And a program whose interpretation equals $\bot$, is a maximum. Figure 2 illustrates the behavioural inclusion on a particular model.

Observability. It may arise that no history will be consistent with a programmed behaviour. For example, the program $\{Obs :: g, \overline{g} \odot\!\!\rightarrow g\}$ is not observable in a trace. Indeed, its interpretation yields

[3] A reflexive and transitive relation.

P	Q
$\{g_1 \circ\!\!\rightarrow g_3,$ $[k_3]\{g_3 \circ\!\!\rightarrow g_4\},$ $[k_4]\{g_3 \circ\!\!\rightarrow g_5\},$ $[k_6]\{g_8 \circ\!\!\rightarrow g_9\},$ $[k_7]\{g_8 \circ\!\!\rightarrow g_{10}\},$ $g_9 \circ\!\!\rightarrow g_{11},$ $g_{10} \circ\!\!\rightarrow g_{11},$ $a :: g_4\}$	$\{[k_1]\{g_0 \circ\!\!\rightarrow g_1\},$ $[k_2]\{g_0 \circ\!\!\rightarrow g_2\},$ $g_1 \circ\!\!\rightarrow g_3,$ $g_2 \circ\!\!\rightarrow g_5,$ $[k_3]\{g_3 \circ\!\!\rightarrow g_4\},$ $[k_4]\{g_3 \circ\!\!\rightarrow g_5\},$ $g_6 \circ\!\!\rightarrow g_8,$ $[k_5]\{g_6 \circ\!\!\rightarrow g_7\},$ $[k_6]\{g_8 \circ\!\!\rightarrow g_9\},$ $[k_7]\{g_8 \circ\!\!\rightarrow g_{10}\},$ $g_9 \circ\!\!\rightarrow g_{11},$ $g_{10} \circ\!\!\rightarrow g_{11},$ $a :: g_4, b :: g_5, c : g_{11}\}$

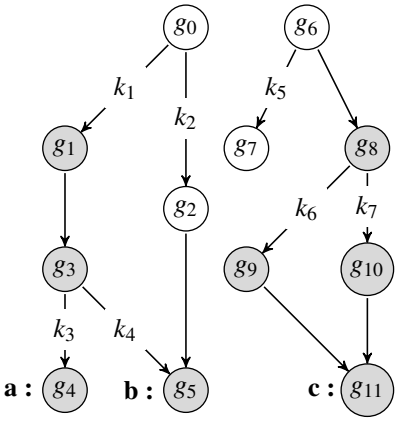


Figure 2: Behavioral inclusion example. Consistent histories of P necessary contains worlds coloured in gray.

to the following formula: $\mathbf{A}((@_{Obs}g) \wedge (g \rightarrow ((\langle\,\rangle^{-}\neg g) \wedge \neg g)))$, false in all models because world *Obs* must both satisfies g and $\neg g$ by definition of the persistent dependence. A GUBS program is said *observable* if and only if the formula resulting from its interpretation is validated by one model at least. Hence, the interpretation of an unobservable program is an antilogy. An unobservable program can be assimilated to a programming error. The detection of such errors can be carried out at compile-phase by using tableaux method [9] that automatically determines whether a formula is satisfiable in a model. Indeed GUBS uses fragment of *HL(@)* logic which is decidable. Notice that an observable program always behaviourally includes an observable program (Proposition 1).

Proposition 1. *A program behaviourally included in an observable program is observable:* $\forall P, Q \in P : \textbf{\textit{obs}}\, Q \wedge P \Subset Q \implies \textbf{\textit{obs}}\, P$.

3.1 Functional synthesis

The functional synthesis is the operation whereby biological components of a library are selected and assembled to generate a device behaviourally including the designed function. The behaviour of each component is described by a GUBS program. At its simplest, the functional synthesis could be considered as a proper substitution of variables by constants. For example, in the following activation $\{G_1 \xrightarrow{+} g_2\}$, g_2 will be substituted by gene G_2, providing that component Q describes the activation $\{G_1 \xrightarrow{+} G_2\}$. However, more complex situations may arise during component selection. For example, if the activation $G_1 \xrightarrow{+} G_2$ occurs with another regulation only *i.e.*, $Q = \{G_1 \xrightarrow{+} G_2, G_3 \xrightarrow{+} G_4\}$ then the selection of Q adds a supplementary regulation.

Formally, a finite substitution is a set of mappings, $\sigma = \{v_i/b_i\}_i$, on variables and constants such that a variable can be substituted by a variable or a constant, and a constant can only substituted

by itself[4]. For instance, we have: $\{Obs{::}G(l)+b_2, b_1 \circ\!\!\rightarrow G(l)\}[\{b_1 \mapsto B_1, b_2 \mapsto B_2, l \mapsto Low\}] = \{Obs{::}G(Low)+B_2, B_1 \circ\!\!\rightarrow G(Low)\}$.

Functional synthesis rules. The functional synthesis is defined by rules (Table 3) governing the component assembly. Only the dependences and the attributes will be functionally synthesize. The observation spots are considered as annotations used for the compilation process. To insure the correctness, each transform must preserved the seminal behaviour. Hence, each program resulting from the application of a rule must behaviourally includes the previous one. Formally, the functional synthesis is modelled by a relation on programs denoted by $\leftarrow$, *i.e.*, $Q \leftarrow_\sigma P$ where P is the initial program and Q the transformed one, such that each rule insures that: $Q \leftarrow_\sigma P$ *is correct with regards to a substitution* σ*, that is* $P[\sigma] \Subset Q[\sigma]$ *and* $Q[\sigma]$ *is observable.* Also notice that the behavioural inclusion is preserved by substitution (Proposition 2).

Proposition 2. *For all substitutions* σ*, we have:* $P \Subset Q \implies P[\sigma] \Subset Q[\sigma]$.

Table 3 describes the functional synthesis rules[5]. Γ is a set of components representing the library. $P \subseteq_{\text{Asm}} Q$ denotes the fact that program Q corresponds to an assembly including P *i.e.*, $Q = (Q_1, P, Q_2)$ where Q_1 or Q_2 may be an empty program. Rule (Inst.) describes the fact that an

- INSTANTIATION -

$$\frac{Q[\sigma] \subseteq_{\text{Asm}} P[\sigma] \qquad \mathbf{obs}(Q[\sigma]) \qquad Q \in \Gamma}{Q \leftarrow_\sigma P} \text{ (Inst.)}$$

- COMMUTATIVITY, CONTRACTION -

$$\frac{Q \leftarrow_\sigma P, P'}{Q \leftarrow_\sigma P', P} \text{ (Com.)} \qquad \frac{Q \leftarrow_\sigma P}{Q \leftarrow_\sigma P, P} \text{ (Cont.)}$$

- ASSEMBLY -

$$\frac{Q \leftarrow_\sigma P \qquad Q' \leftarrow_{\sigma'} P' \qquad \sigma|_{\text{VA}(P)\cap\text{VA}(P')} = \sigma'|_{\text{VA}(P)\cap\text{VA}(P')} \qquad \mathbf{obs}(Q[\sigma], Q'[\sigma'])}{Q, Q' \leftarrow_{\sigma\cup\sigma'} P, P'} \text{ (Asm.)}$$

Table 3: Functional synthesis rules

observable instance of a part of a component in the library is functionally synthesized. Rule (Com.) expresses the commutativity of the assembly. Rule (Cont.) contracts the redundant formulation of programs. Finally, Rule (Asm.) details the conditions for an assembly of two components, each representing a functional synthesis of a part of the designed function. A detailed example of their use on a real case is given in Section 4.

Theorem 1. *The functional synthesis rules (Table 3) are correct.*

[4] $P\sigma$ or $P[\sigma]$ represents its application on program P and identity substitutions are omitted.

[5] Rules are of the form: $\frac{\text{hypothesis}}{\text{conclusion}}$.

- DEPENDENCES -

$$\frac{Q \leftarrow_\sigma S_1 \odot\!\!\rightarrow S_2, S_2 \odot\!\!\rightarrow S_3, \Delta}{Q \leftarrow_\sigma S_1 \odot\!\!\rightarrow S_3, \Delta}\ \text{(Trans.)} \qquad \frac{Q \leftarrow_\sigma S_1 \odot\!\!\rightarrow S_2, \Delta}{Q \leftarrow_\sigma S_1 \bigcirc\!\!\rightarrow S_2, \Delta}\ \text{(N2P.)} \qquad \frac{Q \leftarrow_\sigma S_1 \bigcirc\!\!\rightarrow S_2, \Delta}{Q \leftarrow_\sigma S_1 \oplus\!\!\rightarrow S_2, \Delta}\ \text{(R2N.)}$$

- AGENT STATES -

$$\frac{S_1 + S_2}{S_2 + S_1}\ \text{(SCom.)} \qquad \frac{S + s}{S + s + s}\ \text{(SCont.)} \qquad \frac{S + s}{S}\ \text{(Incl.)}$$

Table 4: Rules for the dependences and the agent states. S_i stands for a collection, $s_1 + \ldots + s_n$, of agent states, including negation, and Δ stands for the rest of the program.

Another set of rules, more specifically devoted to dependences (Table 4), defines the alternate possibilities to express similar behaviours. The table also includes the rules for agent sets. Rule (Trans.) expands the chain of the persistent dependences by adding intermediary dependence to refine a pathway. Rule (N2P.) transforms a normal dependence to a persistent one since the latter is a normal dependence with an additional property. And Rule (R2N.) transforms a remanent dependence to a normal dependence, since normal dependence is also remanent dependence with a repetition of the effect restricted to one step. According to these rules, all the dependence chains can be implemented with persistent dependences.

A possible algorithm for the assembly could be based on a combinatorial application of the rules. However, such algorithm may reveal inefficient in practice. The conditions for an efficient algorithm of compilation should be based on an internal representation of a program, as a set of contextualized dependences with attributes, $\{\{A, [K]S_1 \circledast\!\!\rightarrow S_2\}\}$, such that A, K, S_1, S_2 are respectively: a set of attributes specification related to the agent involved in the dependency, a set of contexts and sets of agent states. Any program can be encoded under this representation from a normal form of the program (not detailed here). Accordingly, the problem solved by the compilation algorithm can be defined as follows (Definition 3):

Definition 3 (Functional Synthesis Problem)**.** *Let* $\Gamma = \{Q_i\}_{1 \leq i \leq n}$ *be set where each* Q_i *is a set of contextualized dependences with attributes and* P *a set of contextualized dependences with attribute, can we find the smallest observable subset of components* $C \subseteq \Gamma$*, such that there exists a substitution* σ *so that its application on the components of* C *form a cover of* $P[\sigma]$*,i.e.,* $\exists \sigma : P[\sigma] \subseteq \bigcup_{Q_j \in C} Q_j[\sigma] \wedge \boldsymbol{obs}\, C$.

As the set cover problem is reducible to this problem, the problem is NP-complete. Then, the resolution is oriented towards a heuristic algorithm.

4 Example

The compilation process is here exemplified in a real case by the design of the Band Detector proposed in [2]. This example explains how from a simple abstract definition of the functionality a complex design can be synthesized. Accordingly, GUBS may be used to describe a behaviour with a high-level of programming well as a low-level, detailing the components involved in the process.

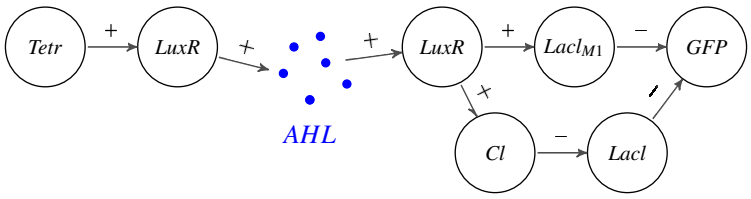


Figure 3: The band detector regulatory circuit.

Although, the functional synthesis is not yet performed automatically, it is worth to point out that the different transforms of the high-level program to obtain the final design complies to rules of Tables 3, 4, insuring its correctness and so, its functional safety in the context of open system.

The design aims at forming patterns of different colours in a population of bacteria exploiting the quorum sensing phenomenon by staining with fluorescent protein (GFP). The amount of molecules of interest that receives a cell depends on its relative position to the cell diffusing the molecule of interest controlled by an external event: more the cell is far from the source, the fewer is the amount of molecules received. The activation or inhibition of the fluorescent protein due to the concentration will distinguish the bands surrounding the source. In the original design, the protein does not fluoresce in an intermediary band.

From a computing standpoint, we can assimilate the design to a message transmission coupled to a sensor/actuator responsible for fluorescence, then leading to a concise GUBS program presented below: the diffusive molecule is *AHL* which production is controlled by a context and the observation is applied on *GFP*. Two categories of cells are defined: the *Sender* and the *Receiver*. Therefore, two GUBS programs identify the two cell types.

$$\begin{aligned}
&\mathit{Sender} \;=\{\; \mathsf{AHL}{:}\{\mathit{low} \Subset \mathit{mid} \Subset \mathit{high}\}, [\mathit{Light}]\{\mathit{detect} \bigcirc\!\!\!\rightarrow \mathsf{AHL}(\mathit{low}), \mathit{detect} \bigcirc\!\!\!\rightarrow \mathsf{AHL}(\mathit{mid}), \mathit{detect} \bigcirc\!\!\!\rightarrow \mathsf{AHL}(\mathit{high})\}\}\\
&\mathit{Receiver}{=}\{\; \mathsf{AHL}(\mathit{low}) \bigcirc\!\!\!\rightarrow \overline{\mathsf{GFP}}, \mathsf{AHL}(\mathit{mid}) \bigcirc\!\!\!\rightarrow \mathsf{GFP}, \mathsf{AHL}(\mathit{high}) \bigcirc\!\!\!\rightarrow \overline{\mathsf{GFP}}, \mathit{obs}_1{::}\mathsf{GFP}, \mathit{obs}_2{::}\overline{\mathsf{GFP}}\}
\end{aligned}$$

Figure 3 describes the original genetic circuit used in the article. The diffusible molecule is the constant *AHL*. The gene *LuxR* has three activation thresholds: at Level 2, it activates both *LaclM1* and *Cl*, at level 1, the amount of *AHL* only allows activation of *Cl*, and finally, at level 0, none are activated. We show that from the sender-receiver program, we obtain the original design by applying the afore mentioned rules with an appropriate selection of components. The regulations of Figure 3 are described in GUBS program (Table 5) translating in term of dependences and relations on their attributes their regulatory action. We focus here on some illustrative steps of the sender program compilation. The complete functional synthesis is given in Appendix. The compilation consists in finding the appropriate components whose assembly behaviourally includes the sender-receiver program, with the particularity that the diffusive molecule must be the same in both programs. To ease compilation follow-up, we label each dependency of the sender-receiver program (Table 6). Let us consider P_{11} whose compilation is closed to P_{12} and P_{13}. Notice that P_{11} cannot be directly instantiated with any component because, in the one hand, the component Q_1 contains a context like P_{11} but applied on gene *Tetr* instead of *AHL*, and on the other hand Q_3 has

Q_1={ [Light]{*detect* ○→ Tetr}}
Q_2={ Tetr $\xrightarrow{+}$ LuxI}
Q_3={ AHL:{*low* ⋕ *mid* ⋕ *high*}, LuxI $\xrightarrow{+}$ AHL(*low*), LuxI $\xrightarrow{+}$ AHL(*mid*), LuxI $\xrightarrow{+}$ AHL(*high*)}
Q_4={ AHL:{*low* ⋕ *mid* ⋕ *high*}, LuxR:{*low* ⋕ {*mid* ≺ *high*}}, AHL(*mid*) ○→ LuxR(*mid*), AHL(*high*) ○→ LuxR(*high*)}
Q_5={ LuxR:{*low* ⋕ {*mid* ≺ *high*}}, LuxR(*mid*) $\xrightarrow{+}$ CI, LuxR(*high*) $\xrightarrow{+}$ CI + LacIM1}
Q_6={ CI $\xrightarrow{-}$ LacI}
Q_7={ LacIM1 $\xrightarrow{-}$ GFP}
Q_8={ LacI $\xrightarrow{-}$ GFP}

Table 5: Part of the database dedicated to the Band Detector.

Sender	*Receiver*
P_{11} = {[*Light*]{*detect* ○→ AHL(*low*)}}	P_{21} = {AHL(*low*) ○→ $\overline{\text{GFP}}$}
P_{12} = {[*Light*]{*detect* ○→ AHL(*mid*)}}	P_{22} = {AHL(*mid*) ○→ GFP}
P_{13} = {[*Light*]{*detect* ○→ AHL(*high*)}}	P_{23} = {AHL(*high*) ○→ $\overline{\text{GFP}}$}

with {AHL:{*low* ⋕ *mid* ⋕ *high*}} as attributes of *AHL*.

Table 6: Separation of the dependences.

the *AHL* molecule but no context is defined. So, to fit P_{11} with the components Q_1, Q_2 and Q_3, first, the normal dependence is converted to persistent one (Rule (N2P.)).

$$\frac{Q_1, Q_2, Q_3 \leftarrow_\sigma \{[light]\{detect \odot\!\!\rightarrow AHL(low)\}\}}{Q_1, Q_2, Q_3 \leftarrow_\sigma P_{11}} \text{ (N2P.)}$$

Thereby, the resulting dependence can be separated to match the assembly Q_1, Q_2, Q_3 by applying (Trans.) rule twice. v_1 and v_2 are fresh variables.

$$\frac{\dfrac{Q_1, Q_2, Q_3 \leftarrow_\sigma P'_{11} = \{[light]\{detect \odot\!\!\rightarrow v_2, v_2 \odot\!\!\rightarrow v_1, v_1 \odot\!\!\rightarrow AHL(low)\}}{Q_1, Q_2, Q_3 \leftarrow_\sigma [light]\{detect \odot\!\!\rightarrow v_1, v_1 \odot\!\!\rightarrow AHL(low)\}} \text{ (Trans.)}}{Q_1, Q_2, Q_3 \leftarrow_\sigma [light]\{detect \odot\!\!\rightarrow AHL(low)\}} \text{ (Trans.)}$$

Finally, we obtain a new program program P'_{11} compatible with Q_1, Q_2, Q_3, and each variable is substituted by a constant (biological element) with the application of Rule (Inst.). For P'_{11} we have:

$$\frac{Q_1, Q_2, Q_3[\sigma = \{light/Light, v_1/Tetr, v_2/Luxl\}] \subseteq_{Asm} P'_{11}[\sigma] \qquad obs(Q_1, Q_2, Q_3[\sigma])}{Q_1, Q_2, Q_3 \leftarrow_\sigma [light]\{detect \odot\!\!\rightarrow v_1, v_1 \odot\!\!\rightarrow v_2, v_2 \odot\!\!\rightarrow AHL(low)\}} \text{ (Inst.)}$$

By following this scheme for P_{12} and P_{13}, we respectively obtain P'_{12} and P'_{13}. The final assembly corresponds to the functional synthesis of *Sender* program.

$$\frac{\begin{array}{ccc} Q_1, Q_2, Q_3 \leftarrow_\sigma P'_{11} & Q_1, Q_2, Q_3 \leftarrow_\sigma P'_{12} & Q_1, Q_2, Q_3 \leftarrow_\sigma P'_{13} \\ \vdots & \vdots & \vdots \\ Q_1, Q_2, Q_3 \leftarrow_\sigma P_{11} & Q_1, Q_2, Q_3 \leftarrow_{\sigma'} P_{12} & Q_1, Q_2, Q_3 \leftarrow_{\sigma''} P_{13} \end{array}}{Q_1, Q_2, Q_3 \leftarrow_{\sigma \cup \sigma' \cup \sigma''} P_{11}, P_{12}, P_{13}} \text{ (Asm.)}$$

In conclusion, the functional synthesis generates the original genetic circuit (Figure 3) from the sender program. A similar approach can be also applied to obtain the receiver program (see the complete proof in Appendix 6).

$$\begin{aligned} \mathit{Sender} \quad = \quad & \{\mathsf{AHL}{:}\{\mathit{low} \not\approx \mathit{mid} \not\approx \mathit{high}\}, [\mathrm{Light}]\{\mathit{detect} \circ\!\!\rightarrow \mathsf{Tetr}\}, \\ & \mathsf{Tetr} \xrightarrow{+} \mathsf{LuxI}, \mathsf{LuxI} \xrightarrow{+} \mathsf{AHL}(\mathit{low}), \mathsf{LuxI} \xrightarrow{+} \mathsf{AHL}(\mathit{mid}), \mathsf{LuxI} \xrightarrow{+} \mathsf{AHL}(\mathit{high})\} \end{aligned}$$

5 Related works

Several domain specific languages have been developped to model and simulate biological systems. Based on process-calculus, seminally used to model process concurrency, several rule-based languages model protein interactions [21, 13, 10]. Another approach is based on logic, such as BIOCHAM [8] that formalizes the temporal properties of a biological system. As these languages are dedicated to simulation, the objective is to close the systems because the simulations need to integrate all the characteristics of the analysed systems. By comparison, the purpose of GUBS is different since the issue is to represent the behaviour of a synthetic device in an organism, leading to translate the notion of the openness of biological systems by the semantics of the language.

In synthetic biology, the structural description languages [12, 20, 4] allow to specify well-formed genome sequences by grammars modularly and hierarchically. Although the sequence description is necessary, the programmer must previously anticipate the behaviour of the device to conceive. Besides, the behavioural design is not included in the program while it initially motivates it. In GUBS, the design is driven by a behaviour description and sequence selection is postponed at compile phase. Moreover, the size of the structural description is also subject to a combinatorial explosion when the complexity of programmed systems increases.

Amorphous programming language has been also investigated to specify the biological devices at the scale of cell colony, here considered as a possible computing medium for amorphous program. J. Beal [3] demonstrates the proof of concept of this approach in PROTO, showing the feasibility of an automatic compile chain. In GUBS, the compile chain is based on rewriting rules whose correctness have been formally proved with regards to a semantics describing the constraints of an open system.

Developing a language for biological systems actually involves to consider several unknown due to their openness: lack of knowledge on all the interactions in biological circuits and imprecise definition of initial conditions. We only know the result of a chain of effects. Then, the major constraint for programming open system seems to be: how to provide an expressive language to describe the dynamics of such systems, but simple enough to capture the essence of the biological questions in a small program in order to allow programming of large biological systems with a program humanly achievable.

In the future, the design in synthetic biology will certainly require different programming layouts based on different paradigms addressing the integration levels of biological systems. In a tower of languages, starting from a language with collective operations on cell colony, using an amorphous programming language as Proto [3] or a language for dynamical systems with dynamical structures as MGS [15], and ending by a structural description programmed in a grammar based

language, GUBS language occupies the intermediary level dedicated to cell entity behavioural programming.

6 Conclusion

In GUBS language, we propose to characterize a programming paradigm abstracting the molecular interactions in the context of open system, that differs to an approach dedicated to biological system modeling. Accordingly, the interactions are symbolized by causal dependences whose interpretation is driven by effect. We have demonstrated the proof-of-concept of the compilation based on rewriting rules, and illustrated it on a realistic example. The perspective of this work is to find an efficient compilation algorithm. Identifying the biological parameters guiding the component selection should be a key issue in this undertaking.

Acknowledgements. The funding for most of this work is granted by the ANR SYNBIOTIC (ANR BLAN 0307 01) and we would like to thank the colleagues of this project for their fruitful discussions.

References

[1] PJ Ashenden (2008): *The Designer's Guide to VHDL*. Morgan Kaufmann Publishers.

[2] S. Basu, Y. Gerchman, C. H Collins, F. H Arnold & R. Weiss (2005): *A Synthetic Multicellular System for Programmed Pattern Formation.* Nature 434(7037), pp. 1130–4.

[3] J. Beal, T. Lu & R. Weiss (2011): *Automatic Compilation from High-Level Biologically-Oriented Programming Language to Genetic Regulatory Networks*. PLoS ONE 6(8), p. e22490.

[4] L. Bilitchenko, A. Liu, S. Cheung, E. Weeding, B. Xia, M. Leguia, J C. Anderson & D. Densmore (2011): *Eugene–A Domain Specific Language for Specifying and Constraining Synthetic Biological Parts, Devices, and Systems.* PloS one 6(4), p. e18882.

[5] P. Blackburn, J. F. A. K. van Benthem & F. Wolter (2006): *Handbook of Modal Logic, Volume 3 (Studies in Logic and Practical Reasoning)*. Elsevier Science Inc.

[6] T. Braüner (2010): *Hybrid Logic and Its Proof-Theory*. Springer.

[7] Y. Cai, M. W Lux, L. Adam & J. Peccoud (2009): *Modeling Structure-function Relationships in Synthetic DNA Sequences Using Attribute Grammars.* PLoS Computational Biology 5(10).

[8] L. Calzone, F. Fages & S. Soliman (2006): *BIOCHAM: An Environment for Modeling Biological Systems and Formalizing Experimental Knowledge.* Bioinformatics (Oxford, England) 22(14), pp. 1805–7.

[9] S. Cerrito & M. C. Mayer (2011): *A Tableaux Based Decision Procedure for a Broad Class of Hybrid Formulae with Binders*. In: Proceedings of the 20th International Conference on Automated Reasoning with Analytic Tableaux and Related Methods, TABLEAUX'11, Springer-Verlag, pp. 104–118.

[10] F. Ciocchetta & J. Hillston (2009): *Bio-PEPA: A Framework for the Modelling and Analysis of Biological Systems*. Theoretical Computer Science 410(33-34), pp. 3065–3084.

[11] K. Clancy & C. A Voigt (2010): *Programming Cells: Towards an Automated Genetic Compiler*. Current Opinion in Biotechnology 21(4), pp. 581–572.

[12] M. J Czar, Y. Cai & J. Peccoud (2009): *Writing DNA with GenoCAD*. Nucleic Acids Research 37(Web Server issue), pp. W40–7.

[13] V. Danos, J. Feret, W. Fontana, R. Harmer & J. Krivine (2007): *Rule-Based Modelling of Cellular Signalling*. In: CONCUR, pp. 17–41.

[14] F. Delaplace, H. Klaudel & A. Cartier-Michaud (2010): *Discrete Causal ModelView of Biological Networks*. In: Proceedings of the 8th International Conference on Computational Methods in Systems Biology - CMSB '10, ACM Press, New York, New York, USA, pp. 4–13.

[15] J.L. Giavitto, O. Michel, J. Cohen & A. Spicher (2005): *Computations in Space and Space in Computations*. In: Unconventional Programming Paradigms, Lecture Notes in Computer Science 3566, Springer Berlin / Heidelberg, pp. 97–97.

[16] D.G. Gibson, J.I. Glass, C. Lartigue, V.N. Noskov, R.Y. Chuang, M.A. Algire, G.A. Benders, M.G. Montague, L. Ma, M.M. Moodie & Others (2010): *Creation of a Bacterial Cell Controlled by a Chemically Synthesized Genome*. Science 329(5987), p. 52.

[17] D. Hume (1739): *A Treatise of Human Nature, Being an Attempt to Introduce the Experimental Method of Reasoning into Moral Subjects*. unknow.

[18] D. Lewis (2000): *Causation as Influence*. The Journal of Philosophy 97(4), pp. 182–197.

[19] T. K Lu, A. S Khalil & J. J Collins (2009): *Next-generation Synthetic Gene Networks*. Nature Biotechnology 27(12), pp. 1139—-1150.

[20] M. P. Pedersen (2009): *Towards Programming Languages for Genetic Engineering of Living Cells*. Journal of the Royal Society, Interface 6 Suppl 4, pp. S437–450.

[21] C. Priami, A. Regev, E. Shapiro & W. Silverman (2001): *Application of a Stochastic Name-passing Calculus to Representation and Simulation of Molecular Processes*. Information Processing Letters 80(1), pp. 25–31.

[22] P. E M Purnick & R. Weiss (2009): *The Second Wave of Synthetic Biology: From Modules to Systems*. Nature Reviews. Molecular Cell Biology 10(6), pp. 410–22.

[23] S. Regot, J. Macia, N. Conde, K. Furukawa, J. Kjellén, T. Peeters, S. Hohmann, E. de Nadal, F. Posas & R. Solé (2010): *Distributed Biological Computation with Multicellular Engineered Networks*. Nature, pp. 2–6.

[24] D. E. Thomas & P. Moorby (1998): *The Verilog Hardware Description Language*. Kluwer Academic Publishers.

[25] P. Umesh, F. Naveen, C.U.M. Rao & S.A. Nair (2010): *Programming Languages for Synthetic Biology*. Systems and Synthetic Biology 4(4), pp. 265–269.

[26] H. Ye, M. Daoud-El Baba, R-W. Peng & M. Fussenegger (2011): *A Synthetic Optogenetic Transcription Device Enhances Blood-glucose Homeostasis in Mice*. Science (New York, N.Y.) 332(6037), pp. 1565–8.

program	::= {behaviour}
behaviour	::= behaviour, behaviour | behaviour
behaviour	::= compartment | dependence | context | observation | defattributes
compartment	::= varconstant {behaviour}
observation	::= varconstant**::**worlds
context	::= [varconstants] {behaviour}
dependence	::= worlds $\circ\!\!\rightarrow$ worlds | worlds $\odot\!\!\rightarrow$ worlds | worlds $\oplus\!\!\rightarrow$ worlds
world	::= attribute | varconstant(attribute) | varconstant.world
worlds	::= worlds + world | world
attribute	::= varconstant | $\overline{\text{varconstant}}$
defattribute	::= varconstants : attspec
attspec	::= defspec{varconstants} | {attrels}
defspec	::= exclusion | inclusion
attrels	::= attrels, attrel | attrel
attrel	::= varconstant $\prec$ varconstant | varconstant $\not\approx$ varconstant | varconstant
varconstant	::= *word* | *Word*
varconstants	::= varconstants, varconstant | varconstant

Table 7: Syntax of GUBS program

Appendix

Proofs

Proposition 1. By contradiction, assume that P is unobservable, then there does not exist a model satisfying the formula. As Q is observable, we deduce that there exists models satisfying Q, but no restricted model must satisfy P, that contradicts the definition of the behavioural consequence. □

Proposition 3. *Let $\psi \in F_{\mathcal{H}}$ be a formula, let $\sigma : (NOM \cup PROP \cup REL) \to (NOM \cup PROP \cup REL)$ be a substitution on nominals, variables and relational symbols, let $\mathcal{M} = \langle W, (R_k)_{k\in\tau}, V\rangle$ be a model, we define the model $\tilde{\mathcal{M}} = \langle W, (\tilde{R}_k)_{k\in\tilde{\tau}}, \tilde{V}\rangle$ from $\mathcal{M}$ as follows:*

1. $\forall a \in NOM \cup PROP, \forall w \in W : w \in V(a\sigma) \iff w \in \tilde{V}(a)$
2. $\forall k \in \tilde{\tau} : wR_{k\sigma}w' \iff w\tilde{R}_k w'$;

we have: $\mathcal{M}, w \Vdash \psi\sigma \iff \tilde{\mathcal{M}}, w \Vdash \psi$.

Proof. The proof is defined by induction on the formula:

without loss of generality, we assume that ψ is in Negation Normal Form where negation occurs only immediately before variables only. Recall that every formula can be set in Negation Normal Form.

- $\mathcal{M}, w \Vdash a \iff \tilde{\mathcal{M}}, w \Vdash a, a \in \mathsf{PROP} \cup \mathsf{NOM}$. By (1), we have $w \in V(a\sigma) \iff w \in \tilde{V}(a)$ leading to the equivalence.

- $\mathcal{M}, w \Vdash \neg a \iff \tilde{\mathcal{M}}, w \Vdash \neg a$. By definition of the realizability relation, this is equivalent to: $\tilde{\mathcal{M}}, w \nVdash a \iff \tilde{\mathcal{M}}, w \nVdash a$. By (1), this equivalence holds.
- $\mathcal{M}, w \Vdash (\psi_1 \wedge \psi_2)\sigma \iff \tilde{\mathcal{M}}, w \Vdash (\psi_1 \wedge \psi_2)$. By definition of the substitution, we have to prove: $\mathcal{M}, w \Vdash (\psi_1\sigma) \wedge (\psi_2\sigma) \iff \tilde{\mathcal{M}}, w \Vdash (\psi_1 \wedge \psi_2)$. By definition of the realizability relation we can formulate the property equivalently as follows:

  $$\mathcal{M}, w \Vdash (\psi_1\sigma) \wedge \tilde{\mathcal{M}}, w \Vdash (\psi_2\sigma) \iff \tilde{\mathcal{M}}, w \Vdash \psi_1 \wedge \tilde{\mathcal{M}}, w \Vdash \psi_2.$$

  By induction hypothesis, we have: $\tilde{\mathcal{M}}, w \Vdash (\psi_1\sigma) \iff \tilde{\mathcal{M}}, w \Vdash \psi_1$ and $\tilde{\mathcal{M}}, w \Vdash (\psi_2\sigma) \iff \tilde{\mathcal{M}}, w \Vdash \psi_2$, implying the previous condition.
- $\mathcal{M}, w \Vdash (\psi_1 \vee \psi_2)\sigma \iff \tilde{\mathcal{M}}, w \Vdash (\psi_1 \vee \psi_2)$. The proof is similar to the proof of the previous item ($\wedge$).
- $\mathcal{M}, w \Vdash (@_a\psi)\sigma \iff \tilde{\mathcal{M}}, w \Vdash @_a\psi$. By definition of the substitution we have to prove that: $\mathcal{M}, w \Vdash (@_{a\sigma}\psi\sigma) \iff \tilde{\mathcal{M}}, w \Vdash @_a\psi$ By definition of the realizability relation, this is equivalent to:

  $$\exists w' \in W : w \in V(a\sigma) \wedge \mathcal{M}, w' \Vdash \psi\sigma \iff \exists w'' \in W : w'' \in \tilde{V}(a)\sigma \wedge \tilde{\mathcal{M}}, w'' \Vdash \psi.$$

  By setting $w' = w''$, from (1) we have: $w' \in V(a\sigma) \iff w' \in V(a)$. By induction hypothesis, we have: $\mathcal{M}, w' \Vdash \psi\sigma \iff \tilde{\mathcal{M}}, w' \Vdash \psi$. The both last properties imply that:

  $$\exists w' \in W : w \in V(a\sigma) \wedge \mathcal{M}, w' \Vdash \psi\sigma \iff \exists w' \in W : w' \in \tilde{V}(a)\sigma \wedge \tilde{\mathcal{M}}, w'' \Vdash \psi,$$

  which implies the initial property.
- $\mathcal{M}, w \Vdash (\langle k \rangle \psi)\sigma \iff \tilde{\mathcal{M}}, w \Vdash \langle k \rangle \psi$. By definition of the substitution we prove that: $\mathcal{M}, w \Vdash \langle k\sigma \rangle \psi\sigma \iff \tilde{\mathcal{M}}, w \Vdash \langle k \rangle \psi$.

  By definition of the realizability relation the condition is equivalent to:

  $$\exists w' \in W : \mathcal{M}, w' \Vdash \psi\sigma \wedge w R_{k\sigma} w' \iff \exists w'' \in W : \tilde{\mathcal{M}}, w'' \Vdash \psi \wedge w \tilde{R}_k w''.$$

  By setting $w' = w''$, the following equivalence holds from (2): $w R_{k\sigma} w' \iff w \tilde{R}_k w'$. By induction hypothesis, we have: $\mathcal{M}, w' \Vdash \psi\sigma \iff \tilde{\mathcal{M}}, w' \Vdash \psi$. The both last properties imply that:

  $$\exists w' \in W : \mathcal{M}, w' \Vdash \psi\sigma \wedge w R_{k\sigma} w' \iff \tilde{\mathcal{M}}, w' \Vdash \psi \wedge w \tilde{R}_k w'$$

  which implies the initial property.
- $\mathcal{M}, w \Vdash ([k]\psi)\sigma \iff \tilde{\mathcal{M}}, w \Vdash [k]\psi$. The proof is similar to the previous item.
- $\mathcal{M} \Vdash (\mathbf{E}\psi)\sigma \iff \tilde{\mathcal{M}} \Vdash \mathbf{E}\psi$. By definition of the substitution we prove that: $\mathcal{M}, w \Vdash \mathbf{E}(\psi\sigma) \iff \tilde{\mathcal{M}}, w \Vdash \mathbf{E}\psi$.

  By definition of the realizability relation, we have:

  $$\exists w \in W : \mathcal{M}, w \Vdash (\psi\sigma) \iff \tilde{\mathcal{M}}, w \Vdash \psi,$$

  which is directly verified by induction hypothesis.

- $\mathcal{M} \Vdash (\mathbf{A}\psi)\sigma \iff \tilde{\mathcal{M}} \Vdash \mathbf{A}\psi$. The proof is similar to the previous item.

□

Proposition 2. First, let us remark that when $P \notin Q$, the property is trivially verified. Besides, under the assumption $P \Subset Q$, if $Q[\sigma]$ is not observable the property is also verified because an unobservable program includes all programs behaviourally (Definition 2).

In the rest of the proof, we assume that P is behaviourally included in Q and $Q[\sigma]$ is observable (*i.e.*, $P \Subset Q$ and $\mathbf{obs}\,Q[\sigma]$). Hence, by definition of the observability there exists a model $\mathcal{M}$ such that $\mathcal{M} \Vdash [\![Q[\sigma]]\!]$. By proposition 3, we deduce that there exists a model $\tilde{\mathcal{M}}$ such that: $\tilde{\mathcal{M}} \Vdash [\![Q]\!]$. Moreover, as $P \Subset Q$ by hypothesis, there exists $\tilde{S} \subseteq \mathrm{Dom}\ \tilde{\mathcal{M}}$ such that: $\tilde{\mathcal{M}}_{\tilde{S}} \Vdash [\![P]\!]$. By construction of $\tilde{\mathcal{M}}$ we deduce that there exists a sub model of $\mathcal{M}$, denoted by $\mathcal{M}'$, complying to the properties, (1) and (2) of Proposition 3 which corresponds to $\tilde{\mathcal{M}}_{\tilde{S}}$. Moreover, we have $\mathcal{M}' \Vdash P[\sigma]$ by Proposition 3. Then we conclude that: $P[\sigma] \Subset Q[\sigma]$. □

Theorem 1. First, let us remark that $P \Subset Q$ is true whenever $\mathcal{M} \nVdash Q$ by definition of the behavioural inclusion (Definition 2). Hence, the proof doesn't consider the trivial verified case but rather the case where $\mathcal{M} \Vdash Q$.

Inst. Directly from the definition of the behavioural inclusion (Definition 2).

Com. By definition of the semantics $[\![P,P']\!] = \mathbf{A}(\phi \wedge \phi') = \mathbf{A}(\phi' \wedge \phi) = [\![P',P]\!]$ with $[\![P]\!]_P = \phi$ and $[\![P']\!]_P = \phi'$. Thus, for all $\mathcal{M}$ we have: $\mathcal{M} \Vdash [\![P,P']\!] \iff \mathcal{M} \Vdash [\![P',P]\!]$. Hence, if $Q \Subset P,P'$ we conclude that: $Q \Subset P',P$.

Cont. Similar to the proof of (Com.).

Asm. First let us remark that $\sigma|_{\mathrm{VA}(P)\cap\mathrm{VA}(P')} = \sigma'|_{\mathrm{VA}(P)\cap\mathrm{VA}(P')}$ means that the substitution of the common variables are the same for σ and σ', leading to, $Q[\sigma \cup \sigma'] = Q[\sigma]$ and $Q'[\sigma \cup \sigma'] = Q'[\sigma']$. Let $\sigma'' = \sigma \cup \sigma'$. Then, we have the following property by definition of the semantics (Table 2.1) and σ''.

$$\forall \mathcal{M} \in \mathsf{KS}([\![(Q,Q')[\sigma'']]\!]) : \mathcal{M} \Vdash [\![Q[\sigma]]\!] \wedge \mathcal{M} \Vdash [\![Q'[\sigma']]\!].$$

Notice that the set of models, $\mathsf{KS}([\![(Q,Q')[\sigma'']]\!])$, is not empty since, by hypothesis, $\mathbf{obs}\,(Q[\sigma],Q'[\sigma'])$ holds. As $Q \leftarrow_{\sigma} P$ and $Q' \leftarrow_{\sigma'} P'$, any model validating Q (resp. Q') also validates P, (resp. P') by definition of the functional synthesis. Then, we deduce that:

$$\forall \mathcal{M} \in \mathsf{KS}([\![(Q,Q')[\sigma'']]\!]) : \mathcal{M} \Vdash [\![P[\sigma]]\!] \wedge \mathcal{M} \Vdash [\![P'[\sigma']]\!].$$

Then, we conclude that:

$$\forall \mathcal{M} \in \mathsf{KS}([\![(Q,Q')[\sigma'']]\!]) : \mathcal{M} \Vdash [\![(P,P')[\sigma'']]\!].$$

□

Complete compilation of the Band Detector

- SENDER -

$$\frac{Q_1,Q_2,Q_3[\sigma=\{detect/Detect,light/Light,v_1/Tetr,v_2/Luxl\}]\subseteq_{Asm} P'_{11}[\sigma] \qquad obs(Q_1,Q_2,Q_3[\sigma])}{Q_1,Q_2,Q_3 \hookleftarrow_{\sigma} P'_{11}\}}\ \text{(Inst.)}$$

$$\frac{Q_1,Q_2,Q_3[\sigma'=\{detect/Detect,light/Light,v_3/Tetr,v_4/Luxl\}]\subseteq_{Asm} P'_{12}[\sigma'] \qquad obs(Q_1,Q_2,Q_3[\sigma'])}{Q_1,Q_2,Q_3 \hookleftarrow_{\sigma'} P'_{12}}\ \text{(Inst.)}$$

$$\frac{Q_1,Q_2,Q_3[\sigma''=\{detect/Detect,light/Light,v_5/Tetr,v_6/Luxl\}]\subseteq_{Asm} P'_{13}[\sigma''] \qquad obs(Q_1,Q_2,Q_3[\sigma''])}{Q_1,Q_2,Q_3 \hookleftarrow_{\sigma''} P'_{13}}\ \text{(Inst.)}$$

$$\frac{\dfrac{\dfrac{P'_{11}=[light]\{detect \odot\!\!\rightarrow v_1, v_1 \odot\!\!\rightarrow v_2, v_2 \odot\!\!\rightarrow AHL(low)\}}{[light]\{detect \odot\!\!\rightarrow v_1, v_1 \odot\!\!\rightarrow AHL(low)\}}\ \text{(Trans.)}}{[light]\{detect \odot\!\!\rightarrow AHL(low)\}}\ \text{(Trans.)}}{P_{11}}\ \text{(N2P.)} \qquad \frac{\dfrac{\dfrac{P'_{12}=[light]\{detect \odot\!\!\rightarrow v_3, v_3 \odot\!\!\rightarrow v_4, v_4 \odot\!\!\rightarrow AHL(mid)\}}{[light]\{detect \odot\!\!\rightarrow v_3, v_3 \odot\!\!\rightarrow AHL(mid)\}}\ \text{(Trans.)}}{[light]\{detect \odot\!\!\rightarrow AHL(mid)\}}\ \text{(Trans.)}}{P_{12}}\ \text{(N2P.)} \qquad \frac{\dfrac{\dfrac{P'_{13}=[light]\{detect \odot\!\!\rightarrow v_5, v_5 \odot\!\!\rightarrow v_6, v_6 \odot\!\!\rightarrow AHL(high)\}}{[light]\{detect \odot\!\!\rightarrow v_5, v_5 \odot\!\!\rightarrow AHL(high)\}}\ \text{(Trans.)}}{[light]\{detect \odot\!\!\rightarrow AHL(high)\}}\ \text{(Trans.)}}{P_{13}}\ \text{(N2P.)}$$

$$\frac{Q_1,Q_2,Q_3 \hookleftarrow_{\sigma} P_{11} \qquad Q_1,Q_2,Q_3 \hookleftarrow_{\sigma'} P_{12} \qquad Q_1,Q_2,Q_3 \hookleftarrow_{\sigma''} P_{13}}{Q_1,Q_2,Q_3 \hookleftarrow_{\sigma\cup\sigma'\cup\sigma''} P_{11},P_{12},P_{13}}\ \text{(Asm.)}$$

- RECEIVER -

$$\frac{Q_4,Q_5,Q_6,Q_8[\sigma=\{v_1/LuxR,v_2/Cl,v_3/Lacl\}]\subseteq_{Asm} P'_{21}[\sigma] \qquad obs(Q_4,Q_5,Q_6,Q_8[\sigma])}{Q_4,Q_5,Q_6,Q_8 \hookleftarrow_{\sigma} P'_{21}}\ \text{(Inst.)}$$

$$\frac{Q_4,Q_5,Q_6,Q_8[\sigma'=\{v_4/LuxR,v_5/Cl,v_6/Lacl\}]\subseteq_{Asm} P'_{22}[\sigma'] \qquad obs(Q_4,Q_5,Q_6,Q_8[\sigma'])}{Q_4,Q_5,Q_6,Q_8 \hookleftarrow'_{\sigma} P'_{22}}\ \text{(Inst.)}$$

$$\frac{Q_4,Q_5,Q_7[\sigma''=\{v_7/LuxR,v_8/LacM1\}]\subseteq_{Asm} P'_{23}[\sigma''] \qquad obs(Q_4,Q_5,Q_7[\sigma''])}{Q_4,Q_5,Q_7 \hookleftarrow''_{\sigma} P'_{23}}\ \text{(Inst.)}$$

$$\frac{\dfrac{\dfrac{P'_{21}=AHL(low) \odot\!\!\rightarrow v_1, v_1 \odot\!\!\rightarrow v_2, v_2 \odot\!\!\rightarrow v_3, v_3 \odot\!\!\rightarrow \overline{GFP}}{AHL(low) \odot\!\!\rightarrow v_1, v_1 \odot\!\!\rightarrow v_2, v_2 \odot\!\!\rightarrow \overline{GFP}}\ \text{(Trans.)}}{AHL(low) \odot\!\!\rightarrow v_1, v_1 \odot\!\!\rightarrow \overline{GFP}}\ \text{(Trans.)}}{AHL(low) \odot\!\!\rightarrow \overline{GFP}}\ \text{(Trans.)}}{P_{21}}\ \text{(N2P.)} \qquad \frac{\dfrac{\dfrac{P'_{22}=AHL(mid) \odot\!\!\rightarrow v_4, v_4 \odot\!\!\rightarrow v_5, v_5 \odot\!\!\rightarrow v_6, v_6 \odot\!\!\rightarrow GFP}{AHL(mid) \odot\!\!\rightarrow v_4, v_4 \odot\!\!\rightarrow v_5, v_5 \odot\!\!\rightarrow GFP}\ \text{(Trans.)}}{AHL(mid) \odot\!\!\rightarrow v_4, v_4 \odot\!\!\rightarrow GFP}\ \text{(Trans.)}}{AHL(mid) \odot\!\!\rightarrow GFP}\ \text{(Trans.)}}{P_{22}}\ \text{(N2P.)} \qquad \frac{\dfrac{\dfrac{P'_{23}=AHL(high) \odot\!\!\rightarrow v_7, v_7 \odot\!\!\rightarrow v_8, v_8 \odot\!\!\rightarrow \overline{GFP}}{AHL(high) \odot\!\!\rightarrow v_7, v_7 \odot\!\!\rightarrow \overline{GFP}}\ \text{(Trans.)}}{AHL(high) \odot\!\!\rightarrow \overline{GFP}}\ \text{(Trans.)}}{P_{23}}\ \text{(N2P.)}$$

$$\frac{Q_4,Q_5,Q_6,Q_8 \hookleftarrow_{\sigma} P_{21} \qquad Q_4,Q_5,Q_6,Q_8 \hookleftarrow_{\sigma'} P_{22} \qquad Q_4,Q_5,Q_7 \hookleftarrow_{\sigma''} P_{23}}{Q_4,Q_5,Q_6,Q_7,Q_8 \hookleftarrow_{\sigma\cup\sigma'\cup\sigma''} P_{21},P_{22},P_{23}}\ \text{(Asm.)}$$

- FINAL DESIGN -

Sender	*Receiver*
$\{\mathsf{AHL}{:}\{low \circledast mid \circledast high\}$, $[\mathsf{Light}]\{detect \bigcirc\!\!\rightarrow \mathsf{Tetr}\}$,	$\{\mathsf{AHL}{:}\{low \circledast mid \circledast high\}$, $\mathsf{LuxR}{:}\{low \circledast \{mid \prec high\}\}$,
$\mathsf{Tetr} \xrightarrow{+} \mathsf{LuxL}$, $\mathsf{LuxI} \xrightarrow{+} \mathsf{AHL}(low)$,	$\mathsf{AHL}(mid) \bigcirc\!\!\rightarrow \mathsf{LuxR}(mid)$, $\mathsf{AHL}(high) \bigcirc\!\!\rightarrow \mathsf{LuxR}(high)$,
$\mathsf{LuxI} \xrightarrow{+} \mathsf{AHL}(mid)$, $\mathsf{LuxI} \xrightarrow{+} \mathsf{AHL}(high)\}$	$\mathsf{LuxR}(mid) \xrightarrow{+} \mathsf{Cl}$, $\mathsf{LuxR}(high) \xrightarrow{+} \mathsf{LaclM1}$,
	$\mathsf{Cl} \xrightarrow{-} \mathsf{Lacl}$, $\mathsf{LaclM1} \xrightarrow{-} \mathsf{GFP}$, $\mathsf{Lacl} \xrightarrow{-} \mathsf{GFP}\}$

Table 8: Complete band detector compilation.

Combining Insertion and Deletion in RNA-editing Preserves Regularity

E.P. de Vink*
Department of Mathematics and Computer Science, Technische Universiteit Eindhoven
Centrum Wiskunde en Informatica, Amsterdam

H. Zantema
Department of Mathematics and Computer Science, Technische Universiteit Eindhoven
Institute for Computing and Information Sciences, Radboud University Nijmegen

D. Bošnački
Department of Biomedical Engineering, Technische Universiteit Eindhoven

Abstract Inspired by RNA-editing as occurs in transcriptional processes in the living cell, we introduce an abstract notion of string adjustment, called guided rewriting. This formalism allows simultaneously inserting and deleting elements. We prove that guided rewriting preserves regularity: for every regular language its closure under guided rewriting is regular too. This contrasts an earlier abstraction of RNA-editing separating insertion and deletion for which it was proved that regularity is not preserved. The particular automaton construction here relies on an auxiliary notion of slice sequence which enables to sweep from left to right through a completed rewrite sequence.

1 Introduction

We study an elementary biologically inspired formalism of string replacement referred to as guided rewriting. Given a fixed and finite set G of strings, also called guides, a rewriting step amounts to adapting a substring towards a guide. We consider two versions of guided rewriting: *guided insertion/deletion*, which is close to an editing mechanism as encountered in the living cell, and general guided rewriting based on an *adjustment relation*, which is mathematically more amenable. For guided insertion/deletion the guide and the part of the string that is rewritten do not need to be of the same length. They are required to be equal up to occurrences of a distinguished dummy symbol. For general guided rewriting the correspondence of the guide and the substring that is rewritten is element-wise. The guide and substring are equivalent symbol-by-symbol according to a fixed equivalence relation called adjustment.

In both cases, for a finite set of guides G, only a finite set of strings can be obtained by repeatedly rewriting a given string. Starting from a language L, we may consider the extension $L_{i/d}$ of the language with all the rewrites obtained by guided insertion/deletion and the extension L_G of the language obtained by adding all the adjustment-based guided rewrites. We address the question if regularity of L implies regularity of $L_{i/d}$ and of L_G. The results of the paper state that in the case of guided insertion/deletion regularity is preserved if the strings of dummy symbols involved are bounded and that guided rewriting based on adjustment always preserves regularity.

The motivation for this work stems from transcriptional biology. RNA can be seen as strings over the alphabet $\{C,G,A,U\}$. Replication of the encoded information is one of the most essential mechanisms in

*Corresponding author, evink@win.tue.nl

Pre-Proceedings of 6th Workshop on Membrane Computing and Biologically Inspired Process Calculi
MeCBIC 2012, Pages 49–63.

life: strands of RNA are faithfully copied by the well-known processes of RNA-transcription. However, typical for eukaryotic cells, the synthesis of RNA does not yield an exact copy of part of the DNA, but a modification obtained by post-processing. The class of the underlying adjustment mechanisms is collectively called *RNA-editing*.

Abstracting away from biological details, the computational power of insertion-deletion systems for RNA-editing is studied in [14]: an insertion step is the replacement of a string uv by the string $u\alpha v$ taken from a particular finite set of triples u, α, v. Similarly, a deletion step replaces $u\alpha v$ by uv for another finite set of triples u, α, v. In [10] the restriction is considered where u and v are both empty. The approach claims full computational power, that is, they generate all recursively enumerable languages.

The notion of splicing, inspired by DNA recombination, has been proposed by Head in [5]. A so-called splicing rule is a tuple $r = (u_1, v_1; u_2, v_2)$. Given two words $w_1 = x_1u_1v_1y_1$ and $w_2 = x_2u_2v_2y_2$ the rule r produces the word $w = x_1u_1v_2y_2$. So, the word w_1 is split in between u_1 and v_1, the word w_2 in between u_2 and v_2 and the resulting subwords x_1u_1 and v_2y_2 are recombined into the word w. For splicing a closure result, reminiscent to the one for guided rewriting considered in this paper, has been established. Casted in our terminology, if L is a regular language and S is a finite set of splicing rules, then L_S is regular too, cf. [8, 11]. Here, L_S is the least language containing L and closed under the splicing rules of S.

In the RNA-editing mechanisms occurring in nature, however, only very limited instances of these formats apply. Often only the symbol U is inserted and deleted, instead of arbitrary strings α, see e.g. [1]. Therefore, following [15], we investigate guided insertion/deletion focusing on the special role of the distinguished symbol 0, the counterpart of the RNA-base U. However, in order to prove that under this scheme regularity is preserved we extend our investigations to guided rewriting based on an abstraction adjustment relation. In fact, we prove the theorem for guided insertion/deletion by appealing to the result for guided rewriting based on adjustment.

The proof of the latter result relies on reorganizing sequences of guided rewrites into sequences of so-called slices. The point is that, since guides may overlap, each guided rewrite step adds a 'layer' on top of the previous string. In this sense guided rewriting is vertically oriented. E.g., Figure 2 in Section 5 shows six rewrite steps of the string *ebcfa* yielding the string *fbcfb* involving eight layers in total. However, in reasoning about recognition by a finite automaton a horizontal orientation is more natural. One would like to sweep from left to right, so to speak. Again referring to Figure 2, five slices can be distinguished, viz. a slice for each symbol of the string *ebcfa*. The technical machinery developed in this paper allows for a transition between the two orientations.

The organization of this paper is as follows. The biological background of RNA-editing is provided in Section 2. Section 3 presents the theorem on preservation of regularity for guided insertion-deletion. The notion of guided rewriting based on an adjustment relation is introduced in Section 4 and a corresponding theorem on preservation of regularity is presented. To pave the way for its proof, Section 5 introduces the notions of a rewrite sequence and of a slice sequence and establishes their relationship. Rewrite sequences record the subsequent guided rewrites that take place, slice sequences represent the cumulative effect of all rewrites at a particular position of the string being adjusted. In Section 6 we provide, given a finite automaton accepting a language L, the construction of an automaton for the extended language L_G with respect to a set of guides G. Section 7 wraps up with related work and concluding remarks.

2 Biological Motivation

This section provides a description of RNA editing from a biological perspective. In this paper we focus on the insertion and deletion of uracil in messenger RNA (mRNA) and provide abstractions of the underlying mechanism in the sequel. However, in the living cell there are different kinds of RNA editing that vary in the type of RNA that is edited and the type of editing operations. Uracil is represented by the letter U. The three other types of nucleotides for RNA, viz. adenine, guanine and cytosine are represented by the letters A, G and C, respectively.

U-insertion/deletion editing is widely studied in the mitochondrial genes of kinetoplastid protozoa [13]. Kinetoplastids are single cell organisms that include parasites like *Trypanosoma brucei* and *Crithidia fasciculata*, that can cause serious diseases in humans and/or animals. Modifications of kinetoplastid mRNA are usually made within the coding regions. These are the parts that are translated into proteins, which are the building blocks of the cells. This way coded information of the original gene can be altered and therefore expressed, i.e. translated into proteins, in a varying number of ways, depending on the environment in the cell. This provides additional flexibility as well as potential specialization of different parts of the organisms for particular functions.

Here we describe a somewhat simplified version of the mechanism for the insertion and deletion of U. More details can be found, for instance, in [13, 1, 3, 12]. For simplicity we assume that only identical letters match with one another. In reality, the matching is based on complementarity, usually assuming the so-called Crick-Watson pairs: A matches with U and G matches with C.

In general, a single step in the editing of mRNA involves two strands of RNA, a strand of messenger RNA and a strand of guide RNA, the latter typically referred to as the guide. To explain the mechanism for the insertion of uracil, let us consider an example. See Figure 1. Assume that we start of with an mRNA fragment: $u = N_1N_2N_3N_4N_5$ and the guide $g = N_2N_3UUUN_4$, where N_i can be an arbitrary nucleotide A, G or C, but not U. Obviously, there is some match between u and g involving the letters N_2, N_3, and N_4, which is partially 'spoiled' by the UUU sequence. By pairing of letters we have that g attaches to u; the matching substrings N_2N_3 and N_4 serve as anchors.

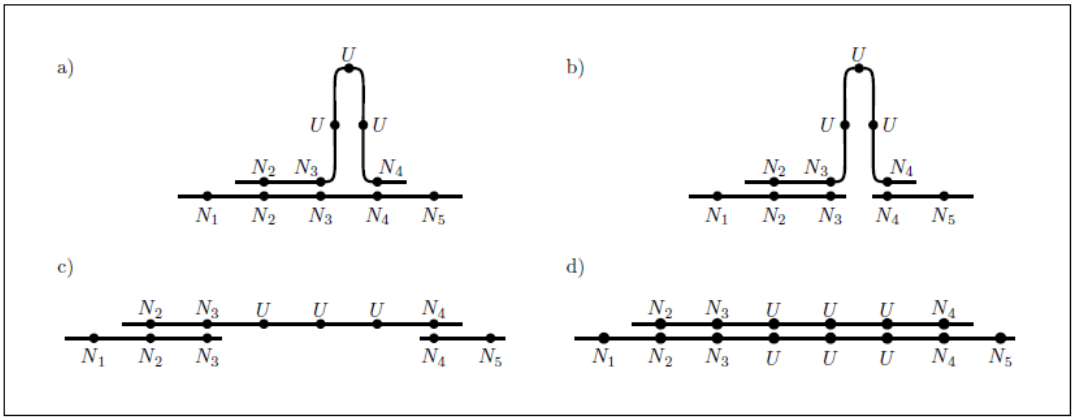


Figure 1: Various stages of guided U-insertion

By chemical reactions involving special enzymes u is split open between N_3 and N_4. The gap between the anchors is then filled by the enzyme mechanism using the guide as a template. For each letter U in the guide a U is added also in the gap. As a result the mRNA string u is transformed into $N_1N_2N_3UUUN_4N_5$.

In general, one can have more than two anchors (involving only non-U letters) in which the guide and the mRNA strand match. In that case mRNA is opened between each pair of anchors and all gaps between these anchors are filled with U such that the number of Us in the guide is matched.

A similar biochemical mechanism implements the deletion of Us from a strand of mRNA. We illustrate the deletion process on the following example. Let us assume that we have the mRNA strand $u = N_1N_2N_3UUN_4N_5$ and the guide $g = N_2N_3N_4$. Like in the insertion case, g initiates the editing by attaching itself to u at the matching positions N_2,N_3, and N_4. Only now the enzymatic complex removes the mismatching UU substring between N_3 and N_4 to ensure the perfect match between the substring and the guide. As a result the edited string $N_1N_2N_3N_4N_5$ is obtained. In general, we can have several anchoring positions on the same string. In that case, all Us between each two matching positions are removed from the mRNA.

A guide can also induce both insertions and deletions of U simultaneously. For instance the guide $N_2N_3UUUN_4$ can induce editing in parallel of the string $N_1UN_2UN_3UN_4UN_5UN_6$ which results in the string $N_1UN_2N_3UUUN_4UN_5UN_6$, where the U between N_2 and N_3 has been deleted and two U's between N_3 and N_4 have been inserted. This is done by the same biochemical mechanisms that are involved in separate insertions and deletions. Analogously as above, we can have multiple insertions and deletions induced by the same guide on the original pre-edited sequence.

The net effect of all three cases considered above is that a strand $u = xyz$, such that y equals g up to occurrences of U, is modified by the insertion and deletion mechanism and becomes a string $v = xgz$. It is noteworthy that the rewriting system that we describe in the sequel also applies to another case with the same effect. For example, consider a guide $g = N_2N_3UUUN_4$ and a pre-edited mRNA $u = N_1N_2N_3UUN_4N_5N_6$. Now, to obtain the match of the guide g and a substring y of u, a U is inserted in u, resulting in the string $v = N_1N_2N_3UUUN_4N_5N_6$. If the U subsequence in y was longer though, like in the case for $u' = N_1N_2N_3UUUN_4N_5N_6$ and $g' = N_2N_3UUN_4$, then we have that the extra U in u' is removed resulting in $v' = N_1N_2N_3UUN_4N_5N_6$.

To summarize, the mRNA editing mechanism underlying U-insertion/deletion can be interpreted as symbolic manipulations of strings. In the sequel symbol U will be denoted by 0 and obviously plays a special role. The crucial point is that in a single step some substring y is replaced by a guide g for which y and g coincide except for the symbol 0.

3 Guided insertion / deletion

Inspired by the biological scheme of editing of mRNA as discussed in the previous section, we study more abstract notion of guided insertion and deletion and guided rewriting based on an adjustment relation in the remainder of this paper. In this section we address guided insertion and deletion, turning to guided rewriting in Section 4.

More precisely, fix an alphabet Σ_0 and distinguish $0 \notin \Sigma_0$. Put $\Sigma = \Sigma_0 \cup \{0\}$. Choose a finite set $G \subseteq \Sigma^*$, with elements g also referred to as guides. Reflecting the biological mechanism, we assume that each $g \in G$ is not equal to the empty string ε and that the first and last letter of each $g \in G$ is not equal to 0. Hence, $G \subseteq \Sigma_0 \cdot \Sigma^* \cdot \Sigma_0$. Now a guided insertion/deletion step $\Rightarrow_{i/d}$ with respect to G is given by

$$u \Rightarrow_{i/d} v \iff u = xyz \wedge v = xgz \wedge g \in G \wedge \pi(y) = \pi(g)$$

where $y \in \Sigma_0 \cdot \Sigma^* \cdot \Sigma_0$, and $\pi(y)$ and $\pi(g)$ are obtained from y and g, respectively, by removing their 0s. Thus, $\pi : \Sigma^* \to \Sigma_0^*$ is the homomorphism such that $\pi(\varepsilon) = \varepsilon$, $\pi(0) = \varepsilon$ and $\pi(a) = a$ for $a \in \Sigma_0$. So,

intuitively, g is anchored on the substring y of u and sequences of 0s are adjusted as prescribed by the guide g, in effect replacing the substring y by the guide g while maintaining the prefix x and suffix z.

As a simple example of a single guided insertion/deletion step, for $G = \{g\}$ with $g = bcb000ab0c$ and $u = a00bc00babcc00a00b$, we have $u \Rightarrow_{i/d} v$ for $v = a00bcb000ab0cc00a00b$. Here we have $u = a00 \cdot bc00babc \cdot c00a00b$, $\pi(bc00babc) = bcbabc = \pi(bcb000ab0c)$ and $v = a00 \cdot bcb000ab0c \cdot c00a00b$. Note, for the string v, being the result of a rewrite with guide g itself with only one possible anchoring, only trivial steps can be taken further. So, the operation of guided insertion/deletion with the same guide g at the same position in a string is idempotent. However, anchoring may overlap. Consider the set of guides $G = \{\, aa0a, a0aa \,\}$, for example. Then the string aaa yields an infinite rewrite sequence

$$aaa \Rightarrow_{i/d} aa0a \Rightarrow_{i/d} a0aa \Rightarrow_{i/d} aa0a \Rightarrow_{i/d} a0aa \cdots$$

Still, from aaa only finitely many different rewrites can be obtained by insertion/deletion steps guided by this G, viz. $\{\, aaa, aa0a, a0aa \,\}$.

The restrictions put on G exclude arbitrary deletions (possible if ε would be allowed as guide) and infinite pumping (if guides need not be delimited by symbols from Σ_0). As an illustration of the latter case, starting from the string abc and 'guide' $0ab$, the infinite sequence $abc \Rightarrow_{i/d} 0abc \Rightarrow_{i/d} 00abc \Rightarrow_{i/d} 000abc \ldots$ would be obtained. The restriction on the substring y prevents to make changes outside the scope of the guide g and forbids $a0b000c \Rightarrow_{i/d} ab0c$ by way of the guide ab.

As a first observation we show that the set $L^u_{i/d} = \{\, v \in \Sigma^* \mid u \Rightarrow^*_{i/d} v \,\}$, for any finite set of guides G and any string u, is finite. Write $u = a_0 0^{i_1} a_1 \ldots a_{n_1} 0^{i_n} a_n$ where $a_i \in \Sigma_0$, $i_k \geqslant 0$, for some $n \geqslant 0$. In effect, a guided insertion/deletion step only modifies the substrings 0^{i_k} or leaves them as is. Therefore, after one or more guided insertion/deletion steps the substrings 0^{i_k} are strings taken from the set

$$Z^u_{i/d} = \{\, 0^{i_k} \mid 1 \leqslant k \leqslant n \,\} \cup \{\, 0^\ell \mid xa \cdot 0^\ell bz \in G,\ a,b \in \Sigma_0,\ \ell \geqslant 0 \,\}$$

Thus, if $u \Rightarrow^*_{i/d} v$ then $v \in \hat{L}^u_{i/d} = \{\, a_0 z_1 a_1 \ldots a_{n_1} z_n a_n \mid z_k \in Z^u_{i/d},\ 1 \leqslant k \leqslant n \,\}$, i.e. $L^u_{i/d} \subseteq \hat{L}^u_{i/d}$. Since the set of guides G is finite, it follows that $Z^u_{i/d}$ is finite, that $\hat{L}^u_{i/d}$ is finite and that $L^u_{i/d}$ is finite as well.

More generally, given a set of guides G, we define the extension by insertion/deletion $L_{i/d}$ of a language L over Σ by putting $L_{i/d} = \{\, v \in \Sigma^* \mid \exists u \in L\colon\ u \Rightarrow^*_{i/d} v \,\}$. Casted to the biological setting of Section 2, L are the strands of messenger RNA, G are strands of guide RNA. Next, we consider the question whether regularity of the language L is inherited by the induced language $L_{i/d}$. Note, despite the finiteness of the insertion/deletion scheme for a single string, it is not obvious that such would hold.

For example, consider the language corresponding to the regular expression $(ab)^*$ together with the operation *sort* which maps a string w over the alphabet $\{a,b\}$ to the string $a^n b^m$ where $n = \#_a(w)$, $m = \#_b(w)$. Thus $sort(w)$ is a sorted version of w with the a's preceding the b's. Note, for $w \in (ab)^*$ there is only one string $sort(w)$, as sorting is a deterministic, hence finitary operation. However, despite $\mathscr{L}(\,(ab)^*\,)$, the language associated by the regular expression, is regular, the language

$$sort(\,(ab)^*\,) = \{\, sort(w) \mid w \in (ab)^* \,\} = \{\, a^n b^n \mid n \geqslant 0 \,\}$$

is *not* regular. Also, if we define the rewrite operation $ba \to_R ab$, then $\{\, v \in \{a,b\}^* \mid u \to^*_R v \,\}$ contains shuffles of the string u, i.e. all strings over $\{a,b\}$ having the same number of a's and b's but are smaller lexicographically. Thus, the set $\{\, v \in \{a,b\}^* \mid u \to^*_R v \,\}$ is finite for each string u. However, the language $\hat{L} = \{\, v \in \{a,b\}^* \mid \exists u \in L\colon\ u \to^*_R v \,\}$ cannot be regular: intersection with the language of a^*b^* does not yield a regular language. More specifically, $\hat{L} \cap \mathscr{L}(\,a^*b^*\,) = \{\, a^n b^n \mid n \geqslant 0 \,\}$. We conclude that the question of $L_{i/d}$ being regular, given regularity of the language L, is not straightforward.

With machinery of rewrite sequences and slice sequences developed in the sequel of the paper, we will be able to prove the following for guided insertion/deletion.

Theorem 1. *In the setting above, if L is a regular language and for some number $k \geqslant 0$ it holds that no string of L or G contains k (or more) consecutive 0's, then the language $L_{i/d}$ is regular too.*

We will prove Theorem 1 by applying a more general result on guided rewriting, viz. Theorem 3 formulated in the next section and ultimately proven in Section 6. As in the notion of guided rewriting as developed in the sequel, symbols are only replaced by single symbols by which lengths of strings are always preserved, a transformation is required to be able to apply Theorem 3.

Before doing so we relate our results to those of [15]. There a relation similar to $\Rightarrow_{i/d}$ was introduced, with the only difference that in a single step either 0's are deleted or inserted, but not simultaneously. A main conclusion of [15] is that in that setting regularity is *not* preserved, so the opposite of the main result in the present setting.

4 Guided rewriting

The idea of guided rewriting is that symbols are replaced by equivalent symbols with respect to some *adjustment relation* $\sim$. The one-one correspondence of the symbols of the string u and its guided rewrite v, enjoyed by this notion of reduction, will turn out technically convenient in the sequel.

Let Σ be a finite alphabet and $\sim$ an equivalence relation on Σ, called the *adjustment relation*. If $a \sim b$ we say that a can be adjusted to b. For a string $u \in \Sigma^*$ we write $\#u$ for its length, use $u[i]$ to denote its i-th element, $i = 1, \ldots, \#u$, and let $u[p,q]$ stand for the substring $u[p]\,u[p{+}1] \cdots u[q]$. The relation $\sim$ is lifted to Σ^* by putting

$$u \sim v \quad \textit{iff} \quad \#u = \#v \wedge \forall i = 1, \ldots, \#u \colon u[i] \sim v[i]$$

Next we define a notion of guided rewriting that involves an adjustment relation.

Definition 2. *We fix a finite subset $G \subseteq \Sigma^*$, called the set of guides.*

(a) For $u, v \in \Sigma^*$, $g \in G$, $p \geqslant 0$, we define $u \Rightarrow_{g,p} v$, stating that v is the rewrite of u with guide g at position p, by

$$u \Rightarrow_{g,p} v \quad \textit{iff} \quad \exists x, y, z \in \Sigma^* \colon u = xyz \wedge \#x = p \wedge y \sim g \wedge v = xgz$$

(b) We write $u \Rightarrow v$ if $u \Rightarrow_{g,p} v$ for some $g \in G$ and $p \geqslant 0$. We use $\Rightarrow^*$ to denote the reflexive transitive closure of $\Rightarrow$. A sequence $u_1 \Rightarrow u_2 \Rightarrow \cdots \Rightarrow u_n$ is called a reduction.

(c) For a language L over Σ and a set of guides G we write

$$L_G = \{\, v \in \Sigma^* \mid \exists u \in L \colon u \Rightarrow^* v \,\}$$

So, a $\Rightarrow$-step adjusts a substring to a guide in G element-wise, and L_G consists of all strings that can be obtained from a string from L by any number of such adjustments. For example, if $\Sigma = \{a, b, c\}$, $G = \{bb\}$ and $a \sim b$ but not $a \sim c$, then by a $\Rightarrow$-step two consecutive symbols not equal to c are replaced by two consecutive b's. In particular, $aaacaa \rightarrow_{bb,1} abbcaa$ and $abbcaa \rightarrow_{bb,0} bbbcaa$. We have

$$\{aaacaa\}_G = \{\, aaacaa, bbacaa, abbcaa, aaacbb, bbbcaa, abbcbb, bbacbb, bbbcbb \,\}$$

Next, we state the main result of this paper regarding guided rewriting.

Theorem 3. *Given an equivalence relation $\sim$ on Σ, let G be a finite set of guides. Suppose L is a regular language. Then L_G is regular too.*

Before going to the proof, we first show that both finiteness of G and the requirement of $\sim$ being an equivalence relation are essential. Below, for a regular expression r we write $\mathscr{L}(r)$ for its corresponding language.

To see that finiteness of G is essential for Theorem 3 to hold, let $G = \{\, c\,a^k\,c\,b^k\,c \mid k \geqslant 0 \,\}$ and $L = \mathscr{L}(c\,a^*\,c\,a^*\,c)$. Let $\sim$ satisfy $a \sim b$ but not $a \sim c$. Then all elements of L on which an adjustment is applicable are of the shape ca^kca^kc, where the result of the adjustment is ca^kcb^kc, which can not be changed by any further adjustment. So

$$L_G \cap \mathscr{L}(c\,a^*\,c\,b^*\,c) \;=\; \{\, c\,a^k\,c\,b^k\,c \mid k \geqslant 0 \,\}$$

is not regular. Since regularity is closed under intersection we conclude that L_G cannot be regular itself.

Also equivalence properties of $\sim$ are essential for Theorem 3. For $G = \{\, ab \,\}$ and $\sim\, = \{\, (a,b), (b,a) \,\}$ the only possible $\Rightarrow$-steps are replacing the pattern ba by ab. Note that here $\sim$ is neither reflexive nor transitive. Since ba may be replaced by ab, bubble sort on a's and b's can be mimicked by $\Rightarrow^*$, while on the other hand $\Rightarrow^*$ preserves both the number of a's and the number of b's. Hence

$$\mathscr{L}((ab)^*)_G \cap \mathscr{L}(a^*b^*) \;=\; \{\, a^kb^k \mid k \geqslant 0 \,\}$$

which proves that $\mathscr{L}((ab)^*)_G$ is not regular, again since regularity is closed under intersection.

5 Rewrite sequences and slice sequences

Fix an alphabet Σ, an adjustment relation $\sim$, and a set of guides G.

Definition 4. *A sequence $\rho = (g_k, p_k)_{k=1}^r$ of guide-position pairs is called a guided rewrite sequence for a string $u \in \Sigma^*$ if it holds that (i) $g_k \in G$, (ii) $0 \leqslant p_k \leqslant \#u - \#g_k$, and (iii) $u[p_k{+}1, p_k{+}\#g_k] \sim g_k$, for all $k = 1, \ldots, r$.*

A guide-position pair (g, p) indicates a redex for a guided rewrite with g of the string u. The position p is relative to u. For the rewrite to fit we must have $p + \#g \leqslant \#u$. The first p symbols of u, i.e. the substring $u[1, p]$, are not affected by the rewrite, as are the last $\#u - p + \#g$ symbols of u, i.e. the substring $u[p{+}\#g{+}1, \#u]$.

The sequence ρ induces a sequence of strings $(u_k)_{k=0}^r$ by putting $u_0 = u$ and u_k such that $u_{k-1} \Rightarrow_{g_k, p_k} u_k$ for $k = 1, \ldots, r$. To conclude that $u_{k-1} \Rightarrow_{g_k, p_k} u_k$ is indeed a proper guided rewrite step, in particular that we have $u_{k-1}[p_k{+}1, p_k{+}\#g_k]$, we use the assumption $u[p_k{+}1, p_k{+}\#g_k] \sim g_k$ and the fact that if $u \Rightarrow_{g,p} v$ then $u[p+1, p+\#g] \sim v[p+1, p+\#g]$. So we obtain $u \Rightarrow^* u_r$ by construction. The string u_r is referred to as the yield of ρ for u, notation $\mathit{yield}(\rho)$. Conversely, every specific reduction from u to v gives rise to a corresponding guided rewrite sequence for u.

Definition 5. *Let $a \in \Sigma$. A sequence $s\ell = (g_i, q_i)_{i \in I}$ of guide-offset pairs, for $I \subseteq \mathbb{N}$ a finite index set, is called a slice for a and G if it holds that (i) $g_i \in G$, (ii) $1 \leqslant q_i \leqslant \#g_i$, and (iii) $a \sim g_i[q_i]$, for all $i \in I$. The slice $s\ell$ is called a slice for a string $u \in \Sigma^*$ at position n, $1 \leqslant n \leqslant \#u$, if it is a slice of $u[n]$.*

Note that in a guide-offset pair (g, q) of a slice sequence, the offset q is relative to the guide q. Since we require $1 \leqslant q \leqslant \#g$ for such a pair, the symbol $g[q]$ is well-defined. We will reserve the use of q for offsets, indices within a guide, and the use of p for positions after which a rewrite may take place, i.e. for lengths of proper substrings of a given string.

The goal of the notion of slice is to summarize the effect of a number of guided rewrites local to a specific position within a string. The symbol generated by the last rewrite that affected the position, i.e. the particular symbol of the last element of the slice sequence, is part of the overall outcome of the total rewrite. This symbol is called the *yield* of the slice. More precisely, if $I \neq \emptyset$, the yield of a slice $s\ell$ for a symbol a is defined as $yield(s\ell) = g_{i_{\max}}[q_{i_{\max}}]$ where $i_{\max} = \max(I)$. In case $I = \emptyset$, we put $yield(s\ell) = a$. Occasionally we write $a \sim s\ell$, as for a slice $s\ell$ for a symbol a it always holds that $a \sim yield(s\ell)$.

A slice $s\ell$ is said to be repetition-free if $g_i = g_j \wedge q_i = q_j$ implies $i = j$. If we have $I = \emptyset$, the slice $s\ell$ is called the empty slice.

Next we consider sequences of slices, and investigate the relationship between slices on two consecutive positions in a guided rewrite sequence.

Definition 6. *A sequence $\sigma = (s\ell_n)_{n=1}^{\#u}$ is called a slice sequence for a string u if the following holds:*

- *$s\ell_n$ is a slice for u at position n, for $n = 1, \ldots, \#u$;*
- *for $n = 1, \ldots, \#u-1$, putting $s\ell_n = (g_i, q_i)_{i\in I}$ and $s\ell_{n+1} = (g'_i, q'_i)_{i\in J}$, there exists a monotone partial injection $\gamma_n : I \to J$ such that, for all $i \in I$ and $j \in J$,*
  - *$i \notin dom(\gamma_n) \implies q_i = \#g_i$*
  - *$\gamma_n(i) = j \iff g_i = g'_j \wedge q_i + 1 = q'_j$*
  - *$j \notin rng(\gamma_n) \implies q'_j = 1$*
- *the slices $s\ell_1$ and $s\ell_{\#u}$, say $s\ell_1 = (g_i, q_i)_{i\in I}$ and $s\ell_{\#u} = (g'_j, q'_j)_{j\in J}$, satisfy $q_i = 1$, for all $i \in I$, and $q'_j = \#g'_j$, for all $j \in J$, respectively.*

For the slices $s\ell_n$ and $s\ell_{n+1}$ the mapping $\gamma_n : I \to J$ is called the cut for $s\ell_n$ and $s\ell_{n+1}$. It witnesses that $s\ell_n$ and $s\ell_{n+1}$ match in the sense that a rewrite may end at position n, may continue for its next offset at position $n+1$, and may start at position $n+1$. Since a cut γ is an order-preserving bijection from $dom(\gamma)$ to $rng(\gamma)$, and $dom(\gamma)$ and $rng(\gamma)$ are finite, it follows that for two slices $s\ell, s\ell'$ the cut $s\ell \to s\ell'$ is unique. We write $s\ell \rightsquigarrow s\ell'$. A slice $s\ell = (g_i, q_i)_{i\in I}$ is called a start slice if $q_i = 1$ for all $i \in I$. Similarly, $s\ell$ is called an end slice if $q_i = \#g_i$ for all $i \in I$. A start slice is generally associated with the first position of the string that is rewritten, an end slice with the last position. Note, a start slice as well as an end slice are allowed to be empty. The yield of the slice sequence σ is the sequence of the yield of its slices, i.e. we define $yield(\sigma) = yield(s\ell_1) \cdots yield(s\ell_{\#u})$.

Example 7. *Let $\sim$ be the adjustment relation with equivalence classes $\{a,b\}, \{c,d\}, \{e,f\}$ and let the set of guides G be given by $G = \{ g_1, g_2, g_3 \}$ where $g_1 = fb$, $g_2 = ace$ and $g_3 = d$. For the string $u = ebcfa$ we consider the guided rewrite sequence $\rho = ((g_3,2), (g_1,0), (g_2,1), (g_1,0), (g_1,3), (g_1,3))$. The associated reduction looks like*

$$ebcfa \Rightarrow_{g_3,2} ebdfa \Rightarrow_{g_1,0} fbdfa \Rightarrow_{g_2,1} facea \Rightarrow_{g_1,0} fbcea \Rightarrow_{g_1,3} fbcfb \Rightarrow_{g_1,3} fbcfb \qquad (1)$$

Recording what happens at all of the five positions of the string u yields, for this example, the slice sequence $\sigma = (s\ell_n)_{n=1}^{5}$ given in the table at the left-hand side of Figure 2, where the slice sequence is visualized too.

For the choice of $I_1, \ldots, I_5$, the monotone partial injection γ_n, $n = 1 \ldots 4$, maps every number to itself. It is easily checked that all requirements of a slice sequence hold. The ovals covering guide-offset pairs reflect the cuts as mappings between to adjacent slices. However, they also comprise, in this situation derived from a guided rewrite sequence, complete guides. Note, $s\ell_1$ is a start slice, $s\ell_5$ is an end slice. We have for the slice sequence $\sigma = (s\ell_n)_{i=1}^{5}$ that $yield(\sigma) = yield(s\ell_1) \cdot \cdots \cdot yield(s\ell_5) = fbcfb$. Indeed, this coincides with the yield of the guided rewrite sequence ρ of (1).

	I_n	$(g_i,q_i)_{i\in I_n}$
$s\ell_1$	$2,4$	$2\mapsto(g_1,1),\ 4\mapsto(g_1,1)$
$s\ell_2$	$2,3,4$	$2\mapsto(g_1,2),\ 3\mapsto(g_2,1),\ 4\mapsto(g_1,2)$
$s\ell_3$	$1,3$	$1\mapsto(g_3,1),\ 3\mapsto(g_2,2)$
$s\ell_4$	$3,5,6$	$3\mapsto(g_2,3),\ 5\mapsto(g_1,1),\ 6\mapsto(g_1,1)$
$s\ell_5$	$5,6$	$5\mapsto(g_1,2),\ 6\mapsto(g_1,2)$

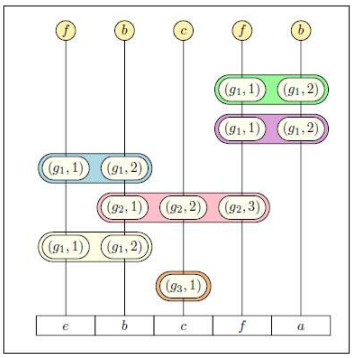


Figure 2: An example slice sequence

The rest of this section is devoted to proving that the above holds in general: Given a string and a set of guides, for every guided rewrite sequence there exists a slice sequence and for every slice sequence there exists a guided rewrite sequence. Moreover, the yield of the guided rewrite sequence and slice sequence are the same.

Theorem 8. *Let $\rho=(g_k,p_k)_{k=1}^{r}$ be a guided rewrite sequence for a string u. Then there exists a slice sequence $\sigma=(s\ell_n)_{n=1}^{\#u}$ for u such that $yield(\sigma)=yield(\rho)$.*

Proof sketch. Induction on r. If ρ is the empty rewrite sequence, we take for σ the slice sequence of n empty slices. Suppose ρ is non-empty. Let $(u_k)_{k=0}^{r}$ be the sequence of strings induced by ρ. By induction hypothesis there exists a slice sequence σ' for the first $r-1$ steps of ρ. Suppose $u_{r-1}\Rightarrow_{g_r,p_r}u_r$. The slice sequence σ is obtained by extending the slices of σ' from position p_r+1 to $p_r+\#g_r$ with the pairs $(g_r,n-p_r)$. Then,

$$\begin{aligned} yield(\sigma) &= yield(\sigma'[1,p_r])\cdot g_r[1,\#g_r]\cdot yield(\sigma'[p_r+\#g_r+1,\#u]) \\ &= u_{r-1}[1,p_r]\cdot g_r\cdot u_{r-1}[p_r+\#g_r+1,\#u_{r-1}] = u_r = yield(\rho) \end{aligned}$$

Verification of σ being a slice sequence for u requires transitivity of $\sim$. □

In order to show the reverse of Theorem 8 we proceed in a number of stages. First we need to relate individual guide-offset pairs in neighboring slices. For this purpose we introduce the ordering $\preccurlyeq$ on so-called chunks.

Definition 9. *Let $\sigma=(s\ell_n)_{n=1}^{\#u}$ be a slice sequence for u. Assume we have $s\ell_n=(g_{n,i},q_{n,i})_{i\in I_n}$, for $n=1,\ldots,\#u$. Let $\gamma_n\colon I_n\to I_{n+1}$ be the cut for $s\ell_n$ and $s\ell_{n+1}$, $1\leqslant n<\#u$. Let $\mathscr{X}=\{(g_{n,i},q_{n,i},i,n)\mid 1\leqslant n\leqslant \#u,\ i\in I_n\}$ be the set of chunks of σ and define the ordering $\preccurlyeq$ on $\mathscr{X}$ by putting $(g,q,i,n)\preccurlyeq(g',q',i',n')$ iff*

- *either $n'\geqslant n$ and there exist indexes $\ell_0,h_0,\ldots,\ell_{n'-n},h_{n'-n}$ such that*
  - *$\ell_k,h_k\in I_{n+k}$ and $\ell_k\leqslant h_k$, $0\leqslant k\leqslant n'-n$*
  - *$h_k\in dom(\gamma_{n+k})$ and $\gamma_{n+k}(h_k)=\ell_{k+1}$, $0\leqslant k<n'-n$*
  - *$\ell_0=i$ and $h_{n'-n}=i'$*
- *or $n'\leqslant n$ and there exist indexes $\ell_0,h_0,\ldots,\ell_{n-n'},h_{n-n'}$ such that*
  - *$\ell_k,h_k\in I_{n'+k}$ and $\ell_k\leqslant h_k$, $0\leqslant k\leqslant n-n'$*

– $\ell_k \in dom(\gamma_{n'+k})$ *and* $\gamma_{n'+k}(\ell_k) = h_{k+1}$, $0 \leqslant k < n-n'$
– $h_0 = i'$ *and* $\ell_{n-n'} = i$

In the above setting with $n' \geqslant n$, we say that the sequence $\ell_0, h_0, \ell_1, h_1, \ldots, \ell_{n'-n}, h_{n'-n}$ is leading from $i \in I_n$ up to $i' \in I_{n'}$. Likewise for the case where $n' \leqslant n$.

For example, for the slice sequence $(s\ell_i)_{i=1}^{r}$ of Figure 2, to identify the guide belonging to the guide-offset pair $(g_2,1)$ of slice $s\ell_2$, the pair is more precisely represented by the chunk $(g_2,1,3,2)$, for the pair is associated with index $3 \in I_2$ of slice $s\ell_2$. Since for the cuts $\gamma_2 : I_2 \to I_3$ and $\gamma_3 : I_3 \to I_4$ we have $\gamma_2(3)$ and $\gamma_3(3) = 3$, we have $(g_2,1,3,2) \preccurlyeq (g_2,2,3,3) \preccurlyeq (g_2,3,3,4)$ via the sequence $3,3,3,3$ connects $(g_2,1)$ and $(g_2,2)$, and $3,3,3,3$ connecting $(g_2,2)$ and $(g_2,3)$. (Hence the combination of sequences $3,3,3,3,3,3$ connecting $(g_2,1)$ and $(g_2,3)$ directly.) As no jumps from a low index ℓ to a high index h needs to be taken, we also have $(g_2,1,3,2) \succcurlyeq (g_2,2,3,3) \succcurlyeq (g_2,3,3,4)$. Thus $(g_2,1,3,2) \equiv (g_2,2,3,3) \equiv (g_2,3,3,4)$ }. In fact, $\{\,(g_2,1,3,2), (g_2,2,3,3), (g_2,3,3,4)\,\}$ is an equivalence class for $\mathscr{X}$ corresponding to the guide g_2 (cf. Lemma 10). Differently, we have $(g_2,1,3,2) \preccurlyeq (g_1,2,6,5)$ relating g_2 to the fourth occurrence of g_1 via the sequence $3,3,3,3,3,5,5,5$, for example. Since there is a jump here from $\ell_2 = 3$ to $h_2 = 5$, we do not have $(g_2,1,3,2) \succcurlyeq (g_1,2,6,5)$. This reflects that apparently the rewrite with this occurrence of g_1 is on top of part of the rewrite using g_2 as guide.

Given a slice sequence σ, the ordering $\preccurlyeq$ on the chunks of σ in $\mathscr{X}$ gives rise to a partial ordering on the set $\mathscr{X}/\equiv$ of equivalence classes of chunks. As we will argue, the equivalence classes correspond to guides and their ordering corresponds to the relative order in which the guides occur in a rewrite sequence ρ having the same yield as the slice sequence σ.

Lemma 10. *(a) The relation $\preccurlyeq$ on $\mathscr{X}$ is reflexive and transitive.*

(b) The relation $\equiv$ on $\mathscr{X}$ such that $x \equiv y \iff x \preccurlyeq y \wedge y \preccurlyeq x$ is an equivalence relation.

(c) The ordering $\preccurlyeq$ on $\mathscr{X}/\equiv$ induced by $\preccurlyeq$ on X by $[x] \preccurlyeq [y] \iff \exists x' \in [x]\, \exists y' \in [y] \colon x' \preccurlyeq y'$, makes $\mathscr{X}/\equiv$ a partial order. □

The next lemma describes the form of the equivalence class holding a chunk $x = (g,q,i,n)$. Using the cuts, equivalent chunks can be found backwards up to position $n-q+1$ and forward up to position $n-q+\#g$. These chunks together, $(g,1,i_{n-q+1},n-q+1)$, …, (g,q,i_n,n), …, $(g,\#g,i_{n-q+\#g},n-q+\#g)$ span the guide g that is to be applied, in the rewrite sequence to be constructed.

Lemma 11. *Let $\sigma = (s\ell_n)_{n=1}^{\#u}$ be a slice sequence for a string u. Let $\mathscr{X} = \{\,(g_{n,i},q_{n,i},i,n) \mid 1 \leqslant n \leqslant \#u,\ i \in I_n\,\}$ be the set of chunks and choose $x \in \mathscr{X}$, say $x = (g,q,i,n)$. Put $p = n-q$. Then there exist $j_1 \in I_{p+1}, \ldots, j_{\#g} \in I_{p+\#g}$ such that $[x] = \{\,(g,s,j_s,p+s) \mid 1 \leqslant s \leqslant \#g\,\}$.* □

We are now in a position to prove the reverse of Theorem 8.

Theorem 12. *Let σ be a slice sequence for a string u. Then there exists a guided rewrite sequence ρ for u such that $yield(\rho) = yield(\sigma)$.*

Proof. Suppose $\sigma = (s\ell_n)_{n=1}^{\#u}$, $s\ell_n = (g_{i,n},q_{i,n})_{i\in I_n}$, for $n = 1,\ldots,\#u$, and let $\mathscr{X} = \{\,(g_{n,i},q_{n,i},i,n) \mid 1 \leqslant n \leqslant \#u,\ i \in I_n\,\}$ be the corresponding set of chunks. We proceed by induction on $\#\mathscr{X}$. Basis, $\#\mathscr{X} = 0$: In this case every slice is empty and $yield(\sigma) = yield(s\ell_1)\cdots yield(s\ell_{\#u}) = u[1]\cdots u[\#u] = u$ and the empty guided rewrite sequence for u has also yield u.

Induction step, $\#\mathscr{X} > 0$: Clearly, $\mathscr{X}/\equiv$ is finite and therefore we can choose, by Lemma 10, $x \in \mathscr{X}$ such that $[x]$ is maximal in $\mathscr{X}/\equiv$. By Lemma 11 we can assume $[x] = \{\,(g,s,i_s,p+s) \mid 1 \leqslant s \leqslant \#g\,\}$ for

suitable p and indexes $i_s \in I_{p+s}$, for $s = 1,\ldots,\#g$. Note, by maximality of $[x]$, the indexes i_s must be the maximum of I_{p+s}. In particular, $yield(\sigma)[p+s] = yield(s\ell_{p+s}) = g[s]$, for $s = 1,\ldots,\#g$.

Now, consider the slice sequence $\sigma' = (s\ell'_n)_{n=1}^{\#u}$ where

$$s\ell'_n = \begin{cases} s\ell_n & \text{for } n = 1,\ldots,p \text{ and } n = p{+}\#g{+}1,\ldots,\#u \\ (g_{i,n}, q_{i,n})_{i \in I_n \setminus \{i_{n-p}\}} & \text{for } n = p{+}1,\ldots,p{+}\#g \end{cases}$$

So, the slice sequence σ' is obtained from the slice sequence σ by leaving out the guide-offset pairs related to the particular occurrence of g.

Let $\mathscr{X}'$ be the set of chunks of σ'. Then $\#\mathscr{X}' < \#\mathscr{X}$. By induction hypothesis we can find a guided rewrite sequence $\rho' = (g'_k, p'_k)_{k=1}^{r}$ for u such that $yield(\rho') = yield(\sigma')$. Define the guided rewrite sequence $\rho = (g_k, p_k)_{k=1}^{r+1}$ by $g_k = g'_k$, $p_k = p'_k$ for $k = 1,\ldots,r$ and $g_{r+1} = g$, $p_{r+1} = p$. We have $0 \leqslant p \leqslant \#u{-}\#g$ and $u[p{+}1, p{+}\#g] \sim g$ since $s\ell_{p+1},\ldots,s\ell_{p+\#g}$ are slices for $u[p{+}1],\ldots,u[p{+}\#g]$, respectively. So, ρ is a well-defined guided rewrite sequence for u.

It holds that $yield(\rho') \Rightarrow_{g,p} yield(\rho)$ as ρ extends ρ' with the pair (g,p). Therefore,

$$yield(\rho)[n] = \begin{cases} yield(\rho')[n] & \text{for } n = 1,\ldots,p \text{ and } n = p{+}\#g{+}1,\ldots,p{+}\#g \\ g[n{-}p] & \text{for } n = p{+}1,\ldots,p{+}\#g \end{cases}$$

From this it follows, for any index n, $1 \leqslant n \leqslant p$ or $p{+}\#g{+}1 \leqslant n \leqslant \#u$, that $yield(\rho)[n] = yield(\rho')[n] = yield(\sigma')[n] = yield(\sigma)[n]$, and for any index n, $p{+}1 \leqslant n \leqslant p{+}\#g$, that $yield(\rho)[n] = g[n{-}p] = yield(\sigma)[n]$. As $\#yield(\rho) = \#yield(\sigma) = \#u$, we obtain $yield(\rho) = yield(\sigma)$, as was to be shown. □

For the slice sequence $(s\ell_i)_{i=1}^{5}$ of Figure 2 we have the following equivalence classes of chunks:

$$\begin{array}{lcllcl} G_3 & = & \{(g_3,1,1,3)\} & G_2 & = & \{(g_2,1,3,2),(g_2,2,3,3),(g_2,3,3,4)\} \\ G_1^1 & = & \{(g_1,1,2,1),(g_1,2,2,2)\} & G_1^3 & = & \{(g_1,1,5,4),(g_1,2,5,5)\} \\ G_1^2 & = & \{(g_1,1,4,1),(g_1,2,4,2)\} & G_1^4 & = & \{(g_1,1,6,4),(g_1,2,6,5)\} \end{array}$$

Moreover, $G_3 \preccurlyeq G_1^1 \preccurlyeq G_2$, $G_2 \preccurlyeq G_1^2$ and $G_2 \preccurlyeq G_1^3 \preccurlyeq G_1^4$. A possible linearization is $G_3 \preccurlyeq G_1^1 \preccurlyeq G_2 \preccurlyeq G_1^3 \preccurlyeq G_1^4 \preccurlyeq G_1^2$. This corresponds to the rewrite sequence

$$ebcfa \Rightarrow_{g_3,2} ebdfa \Rightarrow_{g_1,0} fbdfa \Rightarrow_{g_2,1} facea \Rightarrow_{g_1,3} facfb \Rightarrow_{g_1,3} facfb \Rightarrow_{g_1,0} fbcfb$$

Note that the yield *fbcfb* of this rewrite sequence is the same as the yield of the sequence (1) of Example 7. However, here the second rewrite with $g)1$ of (1) has been moved to the end. This does not effect the end result as the particular rewrites do not overlap.

6 Guided rewriting preserves regularity

Given a language L and a set of guides G, the language L_G is given as the set $\{ v \in \Sigma^* \mid \exists u \in L\colon u \Rightarrow^* v \}$. One of the main results of this paper, Theorem 3 formulated in Section 4, states that if L is regular than L_G is regular too. We will prove the theorem by constructing a non-deterministic finite automaton accepting L_G from a deterministic finite automaton accepting L. The proof exploits the correspondence of rewrite sequences and slice sequences, Theorem 8 and Theorem 12. First we need an auxiliary result to assure finiteness of the automaton for L_G.

Lemma 13. *Let G be a finite set of guides. Let $Z = \{\, s\ell \mid s\ell$ repetition-free slice for a and G, $a \in \Sigma \,\}$. Then Z is finite. Moreover, for every string u and every rewrite sequence ρ for u, there exists a slice sequence σ for u consisting of slices from Z only such that $yield(\sigma) = yield(\rho)$.*

Proof sketch. Finiteness of Z is immediate: there are finitely many guide-offset pairs (g,q), hence finitely many repetition-free finite sequences of them. Thus, there are only finitely many repetition-free slices.

Now, let ρ be a rewrite sequence for a string u. By Theorem 8 we can choose a slice sequence σ' such that $yield(\sigma') = yield(\rho)$. Suppose $\sigma' = (s\ell_n)_{n=1}^{\#u}$ and $s\ell_n = (g_{i,n}, q_{i,n})_{i \in I_n}$ for $n = 1, \ldots, \#u$. By Lemma 11 it follows that given a repeated guide-offset pair (g,q), say $(g,q) = (g_{i,n}, q_{i,n})$ and $(g,q) = (g_{j,n}, q_{j,n})$ for indexes $i < j$ in I_n, we can delete the complete equivalence class of (g_i, q_i, i, n) from slices $s\ell_{n-q+1}$ to $s\ell_{n-q+\#g}$, while retaining a slice sequence. In fact, we are removing the 'lower' occurrence of the guide g. Moreover, the resulting slice sequence has the same yield as for all slices the topmost guide-offset pair remains untouched. The existence of a repetition-free slice sequence σ such that $yield(\sigma) = yield(\sigma')$, hence $yield(\sigma) = yield(\rho)$, then follows by induction on the number of repetitions. □

As a corollary we obtain that every rewrite sequence has a repetition-free equivalent, an intuitive result which require some technicalities though to obtain directly.

We are now prepared to prove that guided rewriting preserves regularity.

Proof of Theorem 3. Without loss of generality $\varepsilon \notin L$. Let $M = (\Sigma, Q, \to, q_o, F)$ be a DFA accepting L. We define the NFA $M' = (\Sigma, Q', \to', q_0, F')$ as follows: Let q_F be a fresh state. Put $Q' = Q \cup (Q \times Z) \cup \{q_F\}$ with Z as given by Lemma 13, $F' = \{q_F\}$ and

$$\begin{array}{rl} q_0 \xrightarrow{\varepsilon}{}' q_0 \times \zeta & \text{if } \zeta \text{ is a start slice} \\ q \times \zeta \xrightarrow{b}{}' q' \times \zeta' & \text{if } q \xrightarrow{a} q',\ a \sim \zeta,\ yield(\zeta) = b,\ \zeta \rightsquigarrow \zeta' \\ q \times \zeta \xrightarrow{b}{}' q_F & \text{if } \exists q' \colon q \xrightarrow{a} q' \in F,\ a \sim \zeta,\ yield(\zeta) = b,\ \zeta \text{ is an end slice} \end{array}$$

Note, by Lemma 13, Q' is a finite set of states. The automaton M' has only one final state, viz. q_F.

Suppose $v \in L_G$. Then there exist $u = a_1 \cdots a_s \in L$, a rewrite sequence $\rho = (g_k, p_k)_{k=1}^{r}$ and strings $u_0, u_1, \ldots, u_r$ such that $u = u_0$, $u_{k-1} \Rightarrow_{g_k,p_k} u_k$ for $k = 1, \ldots, r$, and $v = u_r$. Let, by Theorem 8 and Lemma 13, σ be a slice sequence for u of repetition-free slices with $yield(\sigma) = yield(\rho)$. Say $\sigma = (s\ell_n)_{n=1}^{\#u}$ and $s\ell_n = (g_{i,n}, q_{i,n})_{i \in I_n}$ for $n = 1, \ldots, \#u$. Let $q_0 \xrightarrow{a_1} q_1 \cdots \xrightarrow{a_s} q_s \in F$ be an accepting computation of M for u. Then $q_0 \xrightarrow{\varepsilon}{}' q_0 \times s\ell_1 \xrightarrow{b_1}{}' \cdots q_{s-1} \times s\ell_s \xrightarrow{b_s}{}' q_F$ is an accepting computation of M'. Since we have $b_1 \cdots b_s = yield(s\ell_1) \cdots yield(s\ell_s) = yield(\sigma) = v$, it follows that $v \in \mathscr{L}(M')$. So, $L_G \subseteq \mathscr{L}(M')$.

Let $v = b_1 \cdots b_s$ be a string in $\mathscr{L}(M')$. Given the definition of the transition relation on M', we can find states $q_0, q_1, \ldots, q_{s-1}$, repetition-free slices $s\ell_1, \ldots s\ell_s$ such that $s\ell_n \rightsquigarrow s\ell_{n+1}$ for $n = 1, \ldots, s-1$, and a computation $q_0 \xrightarrow{\varepsilon}{}' q_0 \times s\ell_1 \xrightarrow{b_1}{}' \cdots q_{s-1} \times s\ell_s \xrightarrow{b_s}{}' q_F$. Thus, there exist a final state q_s and a computation $q_0 \xrightarrow{a_1} q_1 \cdots q_{s-1} \xrightarrow{a_s} q_s \in F$ such that $a_n \sim s\ell_s$ for $n = 1, \ldots, s$, i.e. $s\ell_n$ is a slice for a_n. Put $u = a_1 \cdots a_s$. Then $u \in L$, $(s\ell_n)_{n=1}^{\#u}$ is a slice sequence for u and $yield(\sigma) = v$. By Theorem 12 we can find a rewrite sequence ρ for u such that $yield(\rho) = yield(\sigma) = v$. It follows that $u \Rightarrow^* v$ and $v \in L_G$. Thus, $\mathscr{L}(M') \subseteq L_G$. We conclude that $L_G = \mathscr{L}(M')$ and regularity of L_G follows. □

Since $L \subseteq L_G$ the automaton M' should accept any word $a_1 \ldots a_s \in L$, $s > 0$. This can be verified as follows. Let ζ_i be the empty slice for a_i, $i = 1 \ldots s$. Then $a_i \sim \zeta_i$, i.e. $a_i = yield(\zeta_i)$, which holds by definition. Moreover, ζ_1 is a start slice, $\zeta_i \rightsquigarrow \zeta_{i+1}$ for $i = 1 \ldots s-1$, and ζ_s is an end slice. It follows that we can turn an accepting computation of M, say $q_0 \xrightarrow{a_1} q_1 \xrightarrow{a_2} \cdots \xrightarrow{a_s} q_s \in F$ into an accepting computation of M': $q_0 \xrightarrow{\varepsilon}{}' q_0 \times \zeta_1 \xrightarrow{a_1}{}' q_1 \times \zeta_2 \xrightarrow{a_2}{}' \cdots \xrightarrow{a_{s-1}}{}' q_{s-1} \times \zeta_s \xrightarrow{a_s}{}' q_F$.

We now return to a proof of Theorem 1 formulated in Section 3 for which we want to apply Theorem 3. For the latter theorem to apply we need a preparatory transformation. The point is, in the setting of guided insertion/deletion strings are allowed to grow or shrink while guided insertions and deletions are being applied, whereas in the setting of guided rewriting the strings do not change length.

The key idea of the transformation is that every group of 0's is compressed to a single symbol. Let a language L over Σ and a number k be given by Theorem 1. So, L does not contain strings with k or more 0s. We introduce k fresh symbols $0_0, 0_1, \ldots, 0_{k-1}$. Put $\Theta = \{\, 0_0, 0_1, \ldots, 0_{k-1} \,\}$. For any string u over Σ not containing the substring 0^k, i.e. not containing k or more zeros, we define the string $\bar{u}$ over the alphabet $\overline{\Sigma} = (\Sigma \setminus \{0\}) \cup \{0_0, 0_1, \ldots, 0_{k-1}\}$ that is obtained from u by replacing every maximal pattern 0^i by the single symbol 0_i. Note, between two consecutive non-zero letters ab the symbol 0_0 is interspersed. For instance, for $k \geqslant 3$, $\overline{10023} = 10_2 20_0 3$. Also note, that the compression scheme constitutes a 1–1 correspondence of $\Sigma^* \cap \{\, w \mid w \text{ has no substring } 0^k \,\}$ and $\big(\Theta \cdot \Sigma_0\big)^* \cdot \Theta$.

Next, we show that the above operation of compressing groups of 0s preserves regularity using basic closure properties of the class of regular languages, cf. [7, Section 3].

Lemma 14. *Let L be a language without strings containing 0^k and let $\overline{L} = \{\bar{u} \mid u \in L\}$. Then $\overline{L}$ is regular if and only if L is regular.*

Proof. The language L is the homomorphic image of $\overline{L}$ for $h\colon \overline{\Sigma}^* \to \Sigma^*$ with $h(0_i) = 0^i$ and $h(a) = a$ otherwise. So, if $\overline{L}$ is regular, so is L. Reversely, $\overline{L} = (\Theta \cdot \Sigma)^* \cdot \Theta \cap h^{-1}(L)$. Hence, if L is regular, so is $\overline{L}$. □

With the above lemma in place we can give a proof of the preservation of regularity by guided insertion/deletion.

Proof of Theorem 1. Let k be as given by the statement of the theorem. Obtain $\overline{L}$ by applying the compression of strings 0^i, for $i < k$, changing from the alphabet Σ to $\overline{\Sigma}$, as introduced above. By Lemma 14 we then have that $\overline{L}$ is regular. Let $\overline{G}$ be obtained from G, again by compression of strings 0^i, for $i < k$. Then $\overline{G}$ is a finite set of guides with respect to $\overline{\Sigma}$. Now let the adjustment relation $\sim$ be the equivalence relation on $\overline{\Sigma}$ generated by $0_i \sim 0_j$, $0 \leqslant i, j < k$. By Theorem 3 we obtain that $\overline{L}_{\overline{G}}$ is regular.

Next we note that if $u \Rightarrow_{i/d} v$ with respect to Σ, then $\bar{u} \Rightarrow \bar{v}$ with respect to $\overline{\Sigma}$. Vice versa, if $\bar{u} \Rightarrow \bar{v}$ and there exist (unique) u and v such that u, v map to $\bar{u}, \bar{v}$ under compression, then $u \Rightarrow_{i/d} v$. It follows that $\overline{L}_{\overline{G}}$ and $\overline{L_{i/d}}$ coincide. Finally, by another application of Lemma 14, we conclude that $L_{i/d}$ is regular. □

7 Related work and concluding remarks

In this paper we discussed abstract concepts of guided rewriting: a more flexible notion focusing on insertions and deletions of a dummy symbol, another more strict notion based on an equivalence relation. Given a language L we considered the extended languages $L_{i/d}$ and L_G comprising the closure of L for the two types of guided rewriting with guides from a finite set G. In particular, as our main result we proved that these closures preserve regularity. For doing so we investigated the local effect of guided rewriting on two consecutive string positions, leading to a novel notion of a slice sequence. Finally, the theorem for adjustment-based rewriting was proved by an automaton construction exploiting a slice sequence characterization of guided rewriting. Via a compression scheme for strings of dummy symbols, the theorem for guide insertion/deletion followed.

Preservation of regularity by closing a language with respect to a given notion of rewriting arises as a natural question. In Section 3 we observed that by closing the regular language $\mathscr{L}((ab)^*)$ under rewriting with respect to the single rewrite rule $ba \to ab$ the resulting language is not regular. So, by arbitrary string rewriting regularity is not necessarily preserved. A couple of specific rewrite formats have been proposed in the literature. In [6] it was proved that regularity is preserved by deleting string rewriting, where a string rewriting system is called deleting if there exists a partial ordering on its alphabet such that each letter in the right-hand side of a rule is less than some letter in the corresponding left-hand side. In [9] it was proved that regularity is preserved by so-called period expanding or period reducing string rewriting. When translated to the setting of [15], as also touched upon in Section 3, our present notion of guided insertions and deletions allows for simultaneous insertion and deletion of the dummy symbol. A phenomenon also supported by biological findings. Remarkably, the more liberal guided insertion/deletion approach preserves regularity, whereas in the more restricted mechanism of [15], not mixing insertions and deletions per rewrite step, regularity is not preserved.

The computational power of a variant of insertion-deletion systems was studied in [14]. There deletion means that a string $u\alpha v$ is replaced by uv for a predefined finite set of triples u, α, v, while by insertion a string uv is replaced by $u\alpha v$ for another predefined finite set of triples u, α, v. This notion of insertion-deletion is quite different from ours, and seems less related to biological RNA editing. In the same vein are the guided insertion/deletion systems of [2]. There a hierarchy of classes of insertion/deletion systems and related closure properties are studied. Additionally, a non-mixing insertion/deletion system that models part of the RNA-editing for kinetoplastids is given. A rather different application of term rewriting in the setting of RNA is reported in [4], where the rewrite engine of Maude is exploited to predict the occurrence of specific patterns in the spatial formation of RNA, with competitive precision compared to techniques that are more frequently used in bioinformatics.

Possible future work includes investigation of preservation of context-freedom and of lifting the bound on the number of consecutive 0's in Theorem 1. More specifically, for a context-free language L, does it hold, for a finite set of guides G, that L_G is context-free too? Considering the set of guides, a generalization to regular sets G is worthwhile studying. Note that the counter-example given in Section 4 involves a non-regular set of guides. So, if L is regular and G is regular, do we have that L_G is regular? Similarly for L context-free. We also plan to consider guided rewriting based on other types of adjustment relations. In particular, rather than comparing strings symbol-by-symbol, one can consider two strings compatible if they map to the same string for a chosen string homomorphism. A prime example would be the erasing of the dummy 0 in the context of Section 3 for which we conjecture a variant of Theorem 3 to hold.

Acknowledgment We acknowledge fruitful feedback from Peter van der Gulik and detailed comment from the reviewers of the MeCBIC 2012 workshop.

References

[1] J.D. Alfonzo, O. Thiemann & L. Simpson (1997): *The Mechanism of Insertion/Deletion RNA Editing in Kinetoplastid Mitochondria*. Nucleic Acids Research 25(19), pp. 3751–3759.

[2] F. Biegler, M.J. Burrell & M. Daley (2007): *Regulated RNA Rewriting: Modelling RNA Editing with Guided Insertion*. Theoretical Computer Science 387(2), pp. 103–112.

[3] B. Blum, N. Bakalara & L. Simpson (1990): *A Model for RNA Editing in Kinetoplastid Mitochondria: RNA Molecules Transcribed From Maxicircle DNA Provide the Edited Information*. Cell 60, pp. 189–198, doi:10.1016/0092-8674(90)90735-W.

[4] X.Z. Fu, H. Wang, W. Harrison & R. Harrison (2005): *RNA Pseudoknot Prediction using Term Rewriting*. In: *Proc. BIBE'05, Minneapolis*, IEEE Computer Society, pp. 169–176.

[5] T. Head (1987): *Formal Language Theory and DNA: An Analysis of the Generative Capacity of Specific Recombinant Behaviors*. *Bulletin of Mathematical Biology* 49(6), pp. 737–759.

[6] D. Hofbauer & J. Waldmann (2004): *Deleting String Rewriting Systems Preserve Regularity*. *Theoretical Computer Science* 327, pp. 301–317.

[7] J.E. Hopcroft & J.D. Ullman (1979): *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley.

[8] K. Cullik II & T. Harju (1991): *Splicing Semigroups and Dominoes and DNA*. *Discrete Applied Mathematics* 31(3), pp. 261–271.

[9] P. Leupold (2008): *On Regularity-Preservation by String-Rewriting Systems*. In C. Martín-Vide, F. Otto & H. Fernau, editors: *Proc. LATA 2008*, LNCS 5196, pp. 345–356.

[10] M. Margenstern, G. Paun, Y. Rogozhin & S. Verlan (2005): *Context-free Insertion-deletion Systems*. *Theoretical Computer Science* 330, pp. 339–348.

[11] D. Pixton (1996): *Regularity of Splicing Languages*. *Discrete Applied Mathematics* 70(1), pp. 57–79.

[12] H. van der Spek, G.J. Arts, R.R. Zwaal, J. van den Burg, P. Sloof & R. Benne (1991): *Conserved Genes Encode Guide RNAs in Mitochondria of* Crithidia Fasciculata. *EMBO* 10(5), pp. 1217–1224.

[13] K. Stuart, T.E. Allen, S. Heidmann & S.D. Seiwert (1997): *RNA editing in kinteoplastid protozoa*. *Micorbiology and Molecular Biology Reviews* 61(1), pp. 105–120.

[14] A. Takahara & T. Yokomori (2003): *On the Computational Power of Insertion-Deletion Systems*. *Natural Computing* 2(4), pp. 321–336.

[15] H. Zantema (2010): *Complexity of Guided Insertion-Deletion in RNA-Editing*. In A.-H. Dediu, H. Fernau & C. Martín-Vide, editors: *Proc. LATA 2010*, LNCS 6031, pp. 608–619.

Towards Modular Verification of Pathways: Fairness and Assumptions

Peter Drábik
Istituto di Informatica e Telematica
Consiglio Nazionale delle Ricerche
Pisa, Italy
peter.drabik@iit.cnr.it

Andrea Maggiolo-Schettini
Dipartimento di Informatica
Università di Pisa
Pisa, Italy
maggiolo@di.unipi.it

Paolo Milazzo
Dipartimento di Informatica
Università di Pisa
Pisa, Italy
milazzo@di.unipi.it

Modular verification is a technique used to face the state explosion problem often encountered in the verification of properties of complex systems such as concurrent interactive systems. The modular approach is based on the observation that properties of interest often concern a rather small portion of the system. As a consequence, reduced models can be constructed which approximate the overall system behaviour thus allowing more efficient verification.

Biochemical pathways can be seen as complex concurrent interactive systems. Consequently, verification of their properties is often computationally very expensive and could take advantage of the modular approach.

In this paper we report preliminary results on the development of a modular verification framework for biochemical pathways. We view biochemical pathways as concurrent systems of reactions competing for molecular resources. A modular verification technique could be based on reduced models containing only reactions involving molecular resources of interest.

For a proper description of the system behaviour we argue that it is essential to consider a suitable notion of fairness, which is a well-established notion in concurrency theory but novel in the field of pathway modelling. We propose a modelling approach that includes fairness and we identify the assumptions under which verification of properties can be done in a modular way.

We prove the correctness of the approach and demonstrate it on the model of the EGF receptor-induced MAP kinase cascade by Schoeberl et al.

1 Introduction

A big challenge of current biology is understanding the principles and functioning of complex biological systems. Despite the great effort of molecular biologists investigating the functioning of cellular components and networks, we still cannot provide a detailed answer to the question "how a cell works?".

In the last decades, scientists have gathered an enormous amount of molecular level information. To uncover the principles of functioning of a biological system, just collecting data does not suffice. Actually, it is necessary to understand the functioning of parts and the way these interact in complex systems. The aim of *systems biology* is to build, on top of the data, the science that deals with principles of operation of biological systems. The comprehension of these principles is done by modelling and analysis exploiting mathematical means.

A typical scenario of modelling a biological system is as follows. To build a model that explains the behaviour of a real biological system, first a formalism needs to be chosen. Then a model of the system is created, simulation is performed, and the behaviour is observed. The model is validated by comparing the results with the real experiments. The advantage of simulation is not only validation of laboratory experiments, but also prediction of behaviour under new conditions and automation of the whole process.

Pre-Proceedings of 6th Workshop on Membrane Computing and Biologically Inspired Process Calculi
MeCBIC 2012, Pages 65–82.

Simulation can give either the average system behaviour or a number of possible system behaviours. This may be insufficient when one is interested in analysing all the behaviours of a system.

Model checking may be of help. This technique permits the verification of properties (expressed as logical formulae) by exploring all the possible behaviours of a system. This analysis technique typically relies on a state space representation whose size, unfortunately, makes the analysis often intractable for realistic models. This is true in particular for systems of interest in systems biology (such as metabolic pathways, signalling pathways, and gene regulatory networks), which often consist of a huge number of components interacting in different ways, thus exhibiting very complex behaviours.

Many formalisms originally developed by computer scientists to model systems of interacting components have been applied to biology, also with extensions to allow more precise descriptions of the biological behaviours [2, 4, 6, 9, 18, 19]. Examples of well-established formal frameworks that can be used to model, simulate and model check descriptions of biological systems are [6, 14, 15].

Model checking techniques have traditionally suffered from the state explosion problem. Standard approaches to the solution of this problem are based on abstractions or similar model reduction techniques (e.g. [7]). Moreover, the use of Binary Decision Diagrams (BDDs) [8] to represent the state space (symbolic model checking) often allows significantly larger model to be treated [3].

A method for trying to avoid the state space explosion problem is to consider a decomposition of the system, and to apply a modular verification technique allowing global properties to be inferred from properties of the system components. This approach can be particularly efficient when the modelled systems consist of a high number of components, whereas properties of interest deal only with rather small subset of them. This is often the case for properties of biological systems. Hence, for each property it would be useful to be able to isolate a minimal fragment of the model that is necessary for verifying such a property. If such a fragment can be obtained by working only on the syntax of the model, the application of a standard verification technique on the semantics of the fragment avoids the state explosion.

In previous work we developed a modular verification technique in which the system of interest is described by means of a general automata-based formalism suitable for qualitative description of a large class of biological systems, called sync-programs, which supports modular construction [11, 12]. Sync-programs include a notion of synchronization that enables the modelling of biological systems. The modular verification technique is based on property preservation and allows the verification of properties expressed in the temporal logic ACTL^- to be verified on fragments of models. In order to handle modelling and verification of more realistic biological scenarios, we have proposed a dynamic version of our formalism along with an extension of the modular verification framework [10].

The long-term aim of our research is the development of an efficient modular verification framework specifically designed for biochemical pathways, and of a pathway analysis tool based on such a framework. Presently, we are at the first stages of the development of the modular verification framework. However, we already faced some problems whose solution required the definition of concepts related to the formal modelling of biochemical pathways and that we believe could be interesting not only in the context of modular verification. In particular, we defined a notion of *fairness* for biochemical pathways and a notion of *molecular component* of a pathway. The former is a well-known concept in concurrency theory that could be useful to describe more accurately the dynamics of a pathway (in a qualitative framework). The latter is a notion relating species involved in the same pathway such that two species are considered to be part of the same molecular component if they can be seen as different states of the same molecule. As far as we know, the adoption of a notion of fairness in the context of biology is new. On the other hand, the notion of molecular component has been often implicitly used (for instance in the modelling of biological systems by means of automata), but now we can provide new insight on this

notion.

In this paper we report preliminary results obtained during the development of the modular verification framework. Modular verification requires either adopting a modular notation for pathway modelling or finding a way to decompose a pathway, simply expressed as a set of biochemical reactions, into a number of modules. The approach that we choose to follow is in between these two alternatives. Actually, we assume the pathway to be expressed as a set of reactions satisfying some modularisation requirements, and then we define a modularisation procedure that allows modules to be inferred from reactions. Modules will be molecular components, hence our modularisation procedure will allow us to consider a pathway not only as a set of reactions, but also as a set of entities interacting with each other (through reactions) and consequently changing state.

Once the molecular components of a pathway are identified, we can use them to decompose the verification of a global pathway property into the verification of a number of sub-properties related with groups of components. To this aim we define a *projection operation* that allows a model fragment describing the behaviour of a group of components to be obtained from a model describing the whole pathway. Such a projection operation is actually an abstraction function, since the behaviour of the group of components will be over-approximated (i.e. the model will include behaviours that are not present in the model of the whole pathway). By considering a suitable temporal logic for the specification of properties (namely ACTL$^-$, a fragment of the CTL logic consisting only of universally quantified formulae) we can prove that properties holding in model fragments obtained by projection also hold in the complete model of the pathway. Nothing can be said, instead, of properties that do not hold in the the model fragment.

In order to verify properties of complete pathway models or of model fragments it is possible to translate them into the input language of an existing model checking tool. Specifically, we use the NuSMV model checker [5], which is a well-established and efficient instrument.

We demonstrate the modular verification approach on the model of the EGF receptor-induced MAP kinase cascade by Schoeberl et al. [20] and we discuss how we plan to continue the development of the approach to improve its efficiency.

2 Modelling Biochemical Pathways with a Notion of Fairness

In biochemistry, metabolic pathways are networks of biochemical reactions occurring within a cell. The reactions are connected by their intermediates: products of one reaction are substrates for subsequent reactions. Reactions are influenced by catalysts and inhibitors, which are molecules (proteins) which can stimulate and block the occurrence of reactions, respectively. For the sake of simplicity we do not consider inhibitors in this paper, although they could be easily dealt with.

2.1 Syntax and semantics of the modelling notation

Given an infinite set of species S, let us assume biochemical reactions constituting a pathway to have the following form:

$$r_1,\ldots,r_n \;\rightarrow\; p_1,\ldots,p_{n'} \;\{\, c_1,\ldots,c_m \,\}$$

where r_j, p_j and c_j, for suitable values of j, are all in S. We have that r_js are reactants, p_js are products and c_js are catalysts of the considered reaction. Given a reaction R we define $re(R) = \{r_1,\ldots,r_n\}$, $pro(R) = \{p_1,\ldots,p_{n'}\}$, and $cat(R) = \{c_1,\ldots,c_m\}$. We denote the set of species involved in reaction R as $species(R) = re(R) \cup pro(R) \cup cat(R)$.

A pathway P is simply a set of reactions, $P = \{R_1, \ldots, R_N\}$. Given a pathway P, we can infer the set of species involved in it as $species(P) = \bigcup_{R \in P} species(R)$.

The dynamics of a pathway can be described at several different levels of abstraction. The most precise level consists of a quantitative description in which quantities (or concentrations) of species are taken into account, as well as reaction rates in either a deterministic or a stochastic framework. At a more abstract level reaction rates can be ignored. Ultimately, also quantities of species can be ignored by considering only their presence (or absence) in the considered biochemical solution. The less abstract description level is obviously the most precise, but also the most difficult to treat with formal analysis techniques. The more abstract levels are more suitable for the application of formal analysis techniques and are often precise enough to provide some information on the role of the species and of the reactions involved in the pathway. We choose to adopt the most abstract description level, and hence we define a qualitative formal semantics of pathways in which species can only be either present or absent.

The dynamics of a pathway starts from an initial state representing a biochemical solution and is determined by the reactions. A reaction essentially causes the appearance of some new species in the biochemical solution. Actually, we choose to interpret the effect of a reaction depending on whether it is catalysed or not. In our interpretation a reaction without catalysts creates the products but does not consume the reactants. We choose this interpretation since non-catalysed reactions usually reach a steady-state of dynamic equilibrium in which both reactants and products are present in the biochemical solution. On the other hand, a reaction favoured by catalysts usually tends to be performed as long as there are reactants. Therefore, in our interpretation a reaction with catalysts creates the products and consumes the reactants. This choice implies that a reversible reaction in which both directions are catalysed, which frequently occurs in biological pathways, oscillates between two states. This is realistic in some cases (oscillatory behaviours) but not always. We leave a more detailed treatment of this aspect as future work.

Lastly, we assume that all of the catalysts are required to be present in order for the reaction to occur. Alternative combinations of catalysts that may enable the reaction should be modelled as different reactions having the same reactants and products.

Formally, given a pathway P and a set $\mathbf{s_0} \subseteq species(P)$ representing species present in the initial state of the system, the *semantics* of P is given by the labelled transition system $(\mathscr{P}(species(P)), \mathbf{s_0}, \rightarrow_R)$, where $\mathscr{P}(species(P))$ is the powerset of the set of species of P, meaning that each state of the LTS is a configuration of the pathway indicating which species are present. We use the boldface notation, e.g. $\mathbf{s}$ to denote states in the semantics, while a simple s denotes a species which means either a reactant, a product or a catalyst. Furthermore, $\rightarrow_R$: $\mathscr{P}(species(P)) \times P \times \mathscr{P}(species(P))$ is the least transition relation satisfying the following inference rules

$$\frac{re(R) \subseteq \mathbf{s},\ pro(R) \not\subseteq \mathbf{s},\ \emptyset \neq cat(R) \subseteq \mathbf{s}}{\mathbf{s} \xrightarrow{R} (\mathbf{s} \setminus re(R)) \cup pro(R)}(\text{cat}) \qquad \frac{re(R) \subseteq \mathbf{s},\ pro(R) \not\subseteq \mathbf{s},\ cat(R) = \emptyset}{\mathbf{s} \xrightarrow{R} \mathbf{s} \cup pro(R)}(\text{no-cat}).$$

Rules (cat) and (no-cat) formalise the dynamics of reactions in the presence and absence of catalysts, respectively. Both rules contain an assumption which states that the reaction does not occur if its products already exist. Note that thanks to this optimisation transitions that do not change the state of the system are excluded, which is convenient for the verification as the size of the transition system is smaller but the set of properties that hold stays the same. We denote the semantic function as *LTS*, i.e. $LTS : P \mapsto (\mathscr{P}(species(P)), \mathbf{s_0}, \rightarrow_R)$.

A path in $LTS(P)$ can be either a finite sequence $\mathbf{s_0}, R_0, \mathbf{s_1}, R_1, \ldots, \mathbf{s_n}$ or an infinite sequence $\mathbf{s_0}, R_0, \mathbf{s_1}, R_1, \ldots$ where for all i, $\mathbf{s_i}$ is a state and R_i is a reaction and $\mathbf{s_i} \xrightarrow{R_i} \mathbf{s_{i+1}}$ is a transition in $LTS(P)$. The

path consisting only of the initial state $\mathbf{s_0}$ is denoted ε. In this paper we consider only maximal paths, corresponding to behaviours of the pathway in which as long as some reactions can occur, the pathway activity does not halt. It is worth noting that maximal paths are not necessarily infinite, as a state where no reactions can occur has no successor and a path leading to such a state is finite.

2.2 Fairness

In order to describe the behaviour of a pathway more accurately we consider a notion of fairness. We motivate it by considering a quantitative system consisting of four reactions $A \xrightarrow{k_1} B \ \{ D \}$, $B \xrightarrow{k_2} A \ \{ D \}$, $A \xrightarrow{k_3} C \ \{ D \}$ and $C \xrightarrow{k_4} A \ \{ D \}$, where k_1, k_2, k_3 and k_4 are the reaction rates. By performing the qualitative abstraction, we get a pathway containing reactions $R_1 = A \rightarrow B \ \{ D \}$ and $R_2 = B \rightarrow A \ \{ D \}$, $R_3 = A \rightarrow C \ \{ D \}$ and $R_4 = C \rightarrow A \ \{ D \}$, whose semantics as defined above includes behaviours such as the one where R_3 never occurs. Such a behaviour is a qualitative abstraction which is not correct, since the standard quantitative dynamics ruled by the law of mass action would imply that both R_1 and R_3 occur with a frequency proportional to their kinetic rates. Actually, in a stochastic setting both R_1 and R_3 would infinitely occur with probability 1. A correct qualitative abstraction of our system should therefore only include maximal paths in which both R_1 and R_3 occur infinitely many times.

A concept from concurrency theory that allows to specify the correct behaviour is fairness, which stipulates that reactions should compete in a fair manner. We consider the well-known notion of strong fairness [13], also called compassion, which requires that if a reaction is enabled (ready to occur) infinitely many times, then it will occur infinitely many times.

Technically, fairness is specified by a linear temporal logic (LTL) formula. LTL [17] is built up from formulae over a finite set of atomic propositions S, therefore a $s \in S$ is a LTL formula and if f and g are LTL formulae, then so are $\neg f$, $f \vee g$, $X\ g$ and $f\ U\ g$ where X is read as next and U as until. Additional logical operators can be defined, $true = s \vee \neg s$, $false = \neg true$, $f \wedge g = \neg(\neg f \vee \neg g)$ and $f \rightarrow g = \neg f \vee g$, additional temporal operators eventually $F\ g = true\ U\ g$ and globally $G\ g = \neg F\ \neg g$. A LTL formula can be satisfied by a maximal path π in $LTS(P)$ as described by the satisfaction relation $\models_{LTL}$: $\pi \models_{LTL} s$ if $\pi = \mathbf{s}, R, \pi'$, $\pi \models_{LTL} \neg g$ if not $\pi \models_{LTL} g$, $\pi \models_{LTL} f \vee g$ if $\pi \models_{LTL} f$ or $\pi \models_{LTL} g$, $\pi \models_{LTL} X\ g$ if $\pi = \mathbf{s}, R, \pi'$ and $\pi' \models_{LTL} g$, and finally $\pi \models_{LTL} f\ U\ g$ if there is an $i \geq 0$ such that $\pi = \mathbf{s_0}, R_0, \mathbf{s_1}, R_1, \ldots$ and $\mathbf{s_i}, R_i, \pi_i \models_{LTL} f$ and forall $0 \leq k < i$, $s_k, R_k, \pi_k \models_{LTL} g$.

Fairness is expressed by formula Φ, and a maximal path π is fair iff it satisfies Φ, i.e. $\pi \models_{LTL} \Phi$. We have

$$\Phi \iff \bigwedge_{R \in P} (GF\ enabled(R) \rightarrow GF\ occurred(R))$$

where $enabled(R) \iff ((\bigwedge_{r \in re(R)} r) \wedge (\bigvee_{p \in pro(R)} \neg p) \wedge (\bigwedge_{c \in cat(R)} c))$ and the satisfaction of proposition $occurred(R)$ is defined as $\pi' \models_{LTL} occurred(R)$ iff there is a path π such that $\pi = \mathbf{s}, R, \pi'$.

It should be noted that our fairness neither requires all reactions to occur infinitely nor requires fair paths to be infinite.

2.3 Modelling the EGF receptor-induced MAP kinase cascade

We apply our modular verification approach to a well-established computational model of the EGF signalling pathway. We consider the model of the MAP kinase cascade activated by surface and internalised EGF receptors, proposed by Schoeberl et al. in [20]. This model includes a detailed description of the reactions that involve active EGF receptors and several effectors named GAP, ShC, SOS, Grb2, RasGDP/GTP and Raf. Moreover, the model describes the activity of internalised receptors, namely

receptors that are no longer located on the cell membrane, but on a vesicle obtained by endocytosis and floating in the cytoplasm. Such internalised receptors continue to interact with effectors and to contribute to the pathway functioning, but actually the pathway can be seen as composed by two almost identical branches: the first consisting of the reactions stimulated by receptors on the cell membrane and the second consisting of reactions stimulated by internalised receptors.

A diagram representing all of the reactions of the pathway considered in the model is shown in Figure 1. In the figure, species are identified by a short name, but also by a number (in black) in the interval $[1-60]$. Arrows represent reactions, which are also associated with an identifier (in grey) in the interval $[v0-v101]$. Note that the two branches of the pathway are partially combined in the figure. In particular, the representation of most of the species is combined with the representation of its internalised counterpart. In such cases, the number between brackets denotes the number identifying the internalised species. The same holds for reactions: in many cases an arrow denotes both a reaction stimulated by receptors in the cell membrane and the corresponding reaction stimulated by internalised receptors.

The set of reactions constituting the pathway can be trivially reconstructed from the diagram in Figure 1. The only non-trivial aspect is related with the presence in the diagram of some reactions in which one reactants is actually acting as a catalyst. For instance, this happens in the case of the reactions involving Raf* and MEK, in which Raf* initially binds MEK and then releases it phosphorylated. We describe these two reactions in the diagram with the following single catalysed reaction:

$$\text{MEK} \rightarrow \text{MEK-P} \ \{ \ \text{Raf}^* \ \}$$

Other species acting as catalysts are MEK-PP, Phosphatase1, Phosphatase2 and Phosphatase3. By applying the same transformation also to the reactions they are involved in we obtain a pathway constituted by 80 reactions. We call this pathway P_{EGF}.

We recall that fairness requires that a reaction that is infinitely often enabled is also infinitely often performed. This prevents starvation situations to happen among reactions. In the case of P_{EGF} the two branches of the pathway include reactions that could be involved in infinite loops (e.g. the reactions involving MEK and ERK). This means that the semantics of the pathway includes behaviours in which only one branch executes forever even if the other is constantly enabled. Such unrealistic behaviours are excluded by the adoption of fairness.

3 Identification of Molecular Components

In this section we argue, that under conditions often found in practice, a pathway can be decomposed into components, which, as it will be shown in the following sections, can be used for modular verification.

3.1 Assumptions

Intuitively, a species can be seen as a part of a "state" or "configuration" of a more general system component, and a reaction can be seen as a synchronised state change of a set of such system components. In order to view a pathway through this optics, it is convenient to assume that the pathway has equal number of reactants and products (which is not the case in general). Moreover, we assume a positional correspondence between the reactants and the products, in particular we assume that product p_j is the result of the transformation of reactant r_j by the reaction. In our experience, it is usually possible to translate a reaction of a pathway into such a "normal form". Reactions of cellular pathways very often represent bindings (and unbindings) of well-defined macromolecules, such as proteins and genes, to

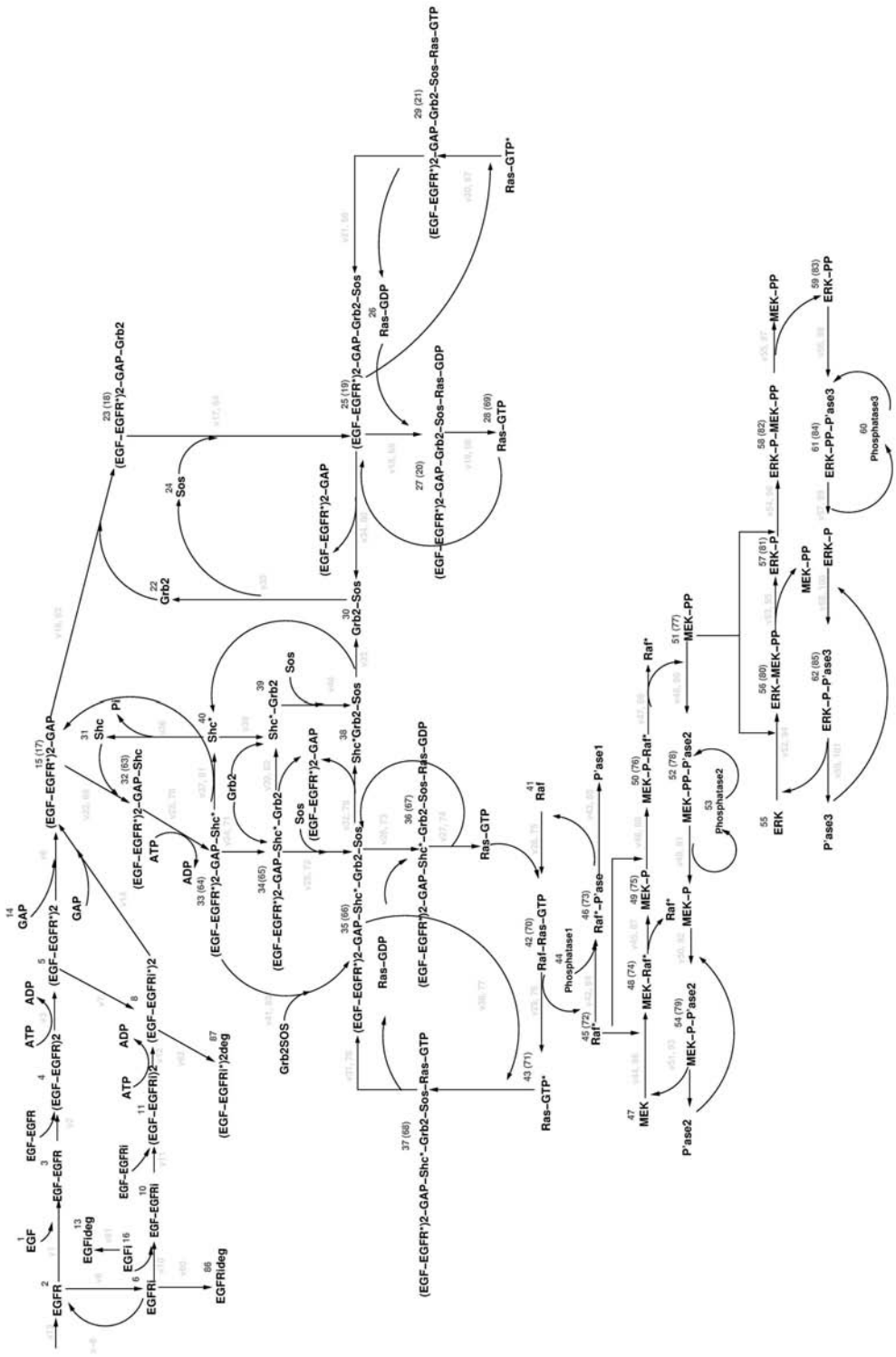


Figure 1: Scheme of the EGF receptor-induced MAP kinase cascade [20]

form (or to break) complexes either with other macromolecules or with small molecules such as ions and nutrients. Also conformational changes are common, in which a protein (or a complex constituted by a few proteins) changes its own "state". If we consider a complex not as a single entity, but as a combination of macromolecules we have that all of the mentioned kinds of reaction do not change the number of (macro)molecules in the system. Hence, it should be possible to model them with the reactions in the form we assume here. For the moment we leave the translation of reaction into the assumed form to the modeller.

In Section 2.1 we have introduced the syntax of the modelling formalism of biochemical pathways, in which a reaction is allowed to have a different number of reactants and products.

3.2 Components identification

Let us, thus, assume that the pathway P consists of reactions in the following form:

$$r_1, \ldots, r_n \ \rightarrow \ p_1, \ldots, p_n \ \{ \ c_1, \ldots, c_m \ \}.$$

Such a form enables us to identify a set of components I that constitute the pathway. Now we present an algorithm that given a pathway P returns the set of components I along with the partition of the set of species belonging to respective components.

We illustrate the intuitive idea on an example. Each reaction can be seen as a synchronisation of components. For example reaction $r_1, r_2 \ \rightarrow \ p_1, p_2 \ \{ \ c \ \}$ can be interpreted as a synchronisation of three components: one that changes its state from a state where r_1 holds into a state where p_1 is present and r_1 is not, another component that changes its state from a state where r_2 holds to a state where p_2 is present and r_2 is not, and a component which participates passively and stays in a state where c is present. Since we suppose that only one reaction takes place at a time in the whole system, the states of all the components do not change other than those involved in the reaction in the way we described. From the example we can see that species r_1 and p_1 belong to the same component. Similarly r_2 belongs to the component that contains p_2, while c is from a separate one.

The algorithm follows. We start by assuming that each species belongs to a different component and we refine this assumption by iterating over the reactions constituting P. The result of the algorithm is a mapping *map* assigning each species to its component.

Algorithm 1 Algorithm to partition species into different components

Let $map : S \mapsto I$ be an injective mapping
for all R in P **do**
  for all r_j in $re(R)$ **do**

$$map := \begin{cases} p \mapsto map(r_j) & \forall p \in \{s \in S \mid map(s) = map(p_j)\} \\ s \mapsto map(s) & \text{otherwise} \end{cases}$$

  end for
end for
return *map*

The algorithm updates the mapping by unifying the elements assigned to reactants and products in the same position in a reaction, and this is done for all reactions in the pathway.

The set of components $comp(P) = I$ of pathway P is the image of mapping *map*. Components of a reaction R denoted, using the same notation, as $comp(R)$ are defined as $comp(\{R\})$.

3.3 Initial state

We adopt a semi-automatic heuristic procedure to find an initial state of the pathway. The idea is the following: for each species s in $species(P)$, if there is no reaction creating it (i.e. if $s \notin \bigcup_{R \in P} pro(R)$) then in the initial state s is present. This means that species that cannot be produced are assumed to be present in the initial state. Otherwise their presence in the model would not be meaningful. Subsequently, we resort again to the partitioning of species according to components to find other species to be inserted. In particular, we find those components containing no species present in the previous phase. These components must contain loops, hence we choose manually some of their species to insert. All other species are assumed absent.

3.4 Visualisation of component interaction

A component interaction graph can be drawn which visualises the components of a pathway and their interactions. It is a directed graph in which vertices are system components (elements of I) and edges connect components that are involved together in a reaction. If two components are both involved as reactants (and consequently products), the edge connecting them will not be oriented (displayed as bidirectional). If one of the two is involved as reactant and the other as catalyst, then the edge will start from the vertex representing the latter to the vertex representing the former. There is no edge between vertices representing components involved in the same reactions only as catalysts.

3.5 The model

The model P_{EGF} is made up of 143 species and 80 reactions. It is in the correct form assumed in Section 3.1 and no preprocessing is needed. After performing the components identification procedure, 14 components are identified. On Figure 2 we can see the component interaction graph of P_{EGF}. Each node of the graph is labelled by the intuitive name of the component that we have chosen.

Visually, we can do some simple observations on the component interaction graph. We can identify enzymes like Phosphatase1, Phosphatase2 and Phosphatase3. We can see the first part of the pathway corresponding to the EGF receptor and its interaction with effectors, and its connection to the MAP kinase cascade through the component RasGDP.

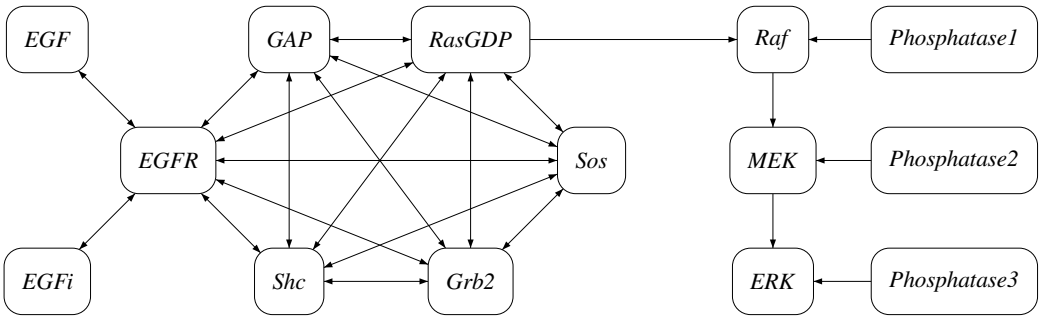


Figure 2: Component interaction graph of P_{EGF}

4 Modular Verification

In this section we define a modular verification technique for pathway models. We proceed by defining the projection of a pathway with help of the identified components. Such a projection can be seen as an abstraction, giving rise to abstract pathways. We prove that a successful verification of a property in the abstraction implies its truth in the original model.

4.1 Abstract pathways: syntax, semantics and fairness

We are interested in analysing only a portion of the entire pathway, in particular a portion induced by only a subset of all components. Let $I = comp(P)$ and $J \subseteq I$, we define the projection of a pathway P onto J as an abstract pathway $P{\restriction}J$.

We will need an extension of function *species*, abusing the notation, which operates on a component set: $species(J) = \{s \in species(R) \mid R \text{ s.t. } comp(R) \cap J \neq \emptyset\}$.

Definition 1. *An abstract pathway $P{\restriction}J$ is a pair (PR, AR), where*

- $PR = \{R \in P \mid comp(R) \subseteq J\}$
- $AR = \bigcup_{\substack{R \in P,\ comp(R) \cap J \neq \emptyset \\ comp(R) \cap (I \setminus J) \neq \emptyset}} \{re(R){\restriction}J \ \rightarrow\ pro(R){\restriction}J \ \{\ cat(R){\restriction}J\ \},\ re(R){\restriction}J \ \rightarrow\ re(R){\restriction}J \ \{\ cat(R){\restriction}J\ \}\}$

where the projection of a set of species $u \subseteq S$ is defined as $u{\restriction}J = \{s \in u \mid s \in species(J)\}$.

An abstract pathway consists of two sets of of reactions: PR contains reactions which influence only components inside J, and AR contains projections of reactions that influence components both inside and outside J. Reactions in PR are exactly as in P, since all of the species involved in such reactions are considered in the abstract pathway. On the other hand, reactions in AR are obtained from the reactions in P which involve species both from some components in J and from some components not in J. For each of such reactions in P we have two reactions in AP: one describing the situation in which species not in $species(J)$ are assumed to be configured such that the reaction can occur, and the other describing the opposite situation. In the first case the reaction in AR produces some products; in the second case the reaction in AR performs a self-loop (i.e. it does not change the state).

The abstract pathway semantics is defined as $LTS_\alpha : P{\restriction}J \mapsto LTS(PR \cup AR)$, that is by using the standard semantics *LTS* on the projected reactions in both PR and AR.

What changes with respect to the pathway model is the definition of fairness. In fact, in this case fairness constraints can be applied only to reactions in PR since reactions in AR consist of pairs of reactions always applicable at the same time and in which it is reasonable to assume that one of the two is always preferred (describing the situation in which the corresponding reaction in R is always enabled or disabled). We define the notion of abstract fairness as

$$\Phi_\alpha \iff \bigwedge_{R \in PR} (GF\ enabled(R) \rightarrow GF\ occurred(R)).$$

Here the compassion is only required for reactions from PR.

Note that a pathway is a special case of abstract pathway, since the semantics of P is equivalent (isomophic) to that of $P{\restriction}I$ and in this case also $\Phi \equiv \Phi_\alpha$ holds.

4.2 Logic for specifying properties

Properties of pathways are specified in temporal logic with species as atomic propositions.

The logic we consider is a fragment of the Computation Tree Logic CTL. Following Attie and Emerson [1], we assume the logic ACTL$^-$ for specification of properties. ACTL is the "universal fragment" of CTL which results from CTL by restricting negation to propositions and eliminating the existential path quantifier and ACTL$^-$ is ACTL without the AX modality.

Definition 2. *The* syntax of ACTL$^-$ *is defined inductively as follows:*

- *The constants true and false are formulae. s and $\neg s$ are formulae for any atomic proposition s, where the set of atomic propositions AP are the set of all species S.*
- *If f, g are formulae, then so are $f \wedge g$ and $f \vee g$.*
- *If f, g are formulae, then so are $A[f\ U\ g]$ and $A[f\ U_w\ g]$.*

We define the logic ACTL^-_J to be ACTL$^-$ where the atomic propositions are drawn from $AP_J = species(J)$. Abbreviations in ACTL$^-$: $AFf \equiv A[true\ U\ f]$ and $AGf \equiv A[f\ U_w\ false]$.

Properties expressible by ACTL$^-$ formulae represent a significant class of properties investigated in the systems biology literature as identified in [16], such as properties concerning exclusion (*"It is not possible for a state* $\mathbf{s}$ *to occur"*), necessary consequence (*"If a state* $\mathbf{s_1}$ *occurs, then it is necessarily followed by a state* $\mathbf{s_2}$"), and necessary persistence (*"A state* $\mathbf{s}$ *must persist indefinitely"*).

On the other hand, properties as occurrence, possible consequence, sequence and possible persistence are of inherently existential nature, and are not expressible in ACTL$^-$.

Definition of the semantics of ACTL$^-$ formulae on labelled transition system $LTS(P)$ follows. Note that only fair maximal paths are considered.

Definition 3. Semantics of ACTL$^-$. *We define $LTS(P), \mathbf{s} \vDash_\Phi f$ (resp. $LTS(P), \pi \vDash_\Phi f$) meaning that f is true in structure $LTS(P)$ at state $\mathbf{s}$ (resp. fair maximal path π). We define $\vDash_\Phi$ inductively:*

- *$LTS(P), \mathbf{s} \vDash_\Phi true$. $LTS(P), \mathbf{s} \nvDash false$. $LTS(P), \mathbf{s} \vDash_\Phi s$ iff $\mathbf{s}(s) = tt$. $LTS(P), \mathbf{s} \vDash_\Phi \neg s$ iff $\mathbf{s}(s) = ff$.*
- *$LTS(P), \mathbf{s} \vDash_\Phi f \wedge g$ iff $LTS(P), \mathbf{s} \vDash_\Phi f$ and $LTS(P), \mathbf{s} \vDash_\Phi g$.*
  $LTS(P), \mathbf{s} \vDash_\Phi f \vee g$ iff $LTS(P), \mathbf{s} \vDash_\Phi f$ or $LTS(P), \mathbf{s} \vDash_\Phi g$.
- *$LTS(P), \mathbf{s} \vDash_\Phi Af$ iff for every fair maximal path $\pi = (\mathbf{s}, R, \ldots)$ in $LTS(P) : LTS(P), \pi \vDash_\Phi f$.*
- *$LTS(P), \pi \vDash_\Phi f$ iff $LTS(P), \mathbf{s} \vDash_\Phi f$, where $\mathbf{s}$ is the first state of π*
- *$LTS(P), \pi \vDash_\Phi f \wedge g$ iff $LTS(P), \pi \vDash_\Phi f$ and $LTS(P), \pi \vDash_\Phi g$.*
  $LTS(P), \pi \vDash_\Phi f \vee g$ iff $LTS(P), \pi \vDash_\Phi f$ or $LTS(P), \pi \vDash_\Phi g$.
- *$LTS(P), \pi \vDash_\Phi f\ U\ g$ iff $\pi = (\mathbf{s_0}, R_0, \mathbf{s_1}, R_1, \ldots)$ and there is $m \in \mathbb{N}$ such that $LTS(P), \mathbf{s_m} \vDash_\Phi g$ and for all $m' < m : LTS(P), \mathbf{s_{m'}} \vDash_\Phi f$.*
- *$LTS(P), \pi \vDash_\Phi f\ U_w\ g$ iff $\pi = (\mathbf{s_0}, R_0, \mathbf{s_1}, R_1, \ldots)$ and for all $m \in \mathbb{N}$, if $LTS(P), \mathbf{s_{m'}} \nvDash g$ for all $m' < m$ then $LTS(P), \mathbf{s_m} \vDash_\Phi f$.*

We assume $\vDash_{\Phi_\alpha}$ to be defined as $\vDash_\Phi$, but with abstract fairness Φ_α replacing Φ.

4.3 Modular verification theorems

Now we prove that in order to verify an ACTL^-_J property for a pathway P, it is enough to verify the same property in the abstract semantics of the abstract pathway $P\restriction J$. The principle behind property preservation is that each path in the semantics of the modelled pathway must have a corresponding abstract path in the abstract semantics of a model obtained by projection. This, combined with the fact that ACTL^- properties are universally quantified (namely describe properties that have to be satisfied by all paths) ensure that if an ACTL^- property holds in the abstract semantics of the projection, then it will also hold in the semantics of the orginal model. In fact, for the components considered in a projection the semantics of the original model will contain essentially a subset of the paths of the projected model.

ACTL^- properties are universally quantified, namely they they deal with all paths starting form a given initial state, and the fact that all original paths (more precisely their projections) are included amongst the paths of the projection. Thus if one proves that the property holds in the projection for all paths it will hold for all paths also in the original system.

First we define the path projection, which from a path in semantics of a pathway with the set of components I removes transitions made by components outside of portion $J \subseteq I$ and restricts the rest of transitions onto J.

Definition 4.

$$\pi\lceil J = \begin{cases} \varepsilon & \text{if } \pi = \varepsilon \\ \pi'\lceil J & \text{if } \pi = \mathbf{s}, R, \pi' \text{ and } comp(R) \cap J = \emptyset \\ \mathbf{s}\lceil J, R, \pi'\lceil J & \text{if } \pi = \mathbf{s}, R, \pi' \text{ and } comp(R) \cap J \neq \emptyset \end{cases}$$

Follows the infinite path projection which ensures that the resulting traces are infinite. In case of a finite original trace it adds an infinite looping in the final state.

Definition 5. *Given $\pi\lceil J$ with initial state $\mathbf{s_0}$ we define $\pi\lceil^\infty J = \pi\lceil J$ if $\pi\lceil J$ is infinite, otherwise if $\pi\lceil J = \mathbf{s_0}, R_0, \ldots, \mathbf{s_{n-1}}, R_{n-1}, \mathbf{s_n}$ we define $\pi\lceil^\infty J = \pi\lceil J, (*, \mathbf{s_n})^\infty$, where $(R, \mathbf{s})^\infty = R, \mathbf{s}, (R, \mathbf{s})^\infty$. We denote $\varepsilon\lceil^\infty J = \mathbf{s_0}, (*, \mathbf{s_0})^\infty$ as ε^∞.*

Now we are in the position to present the crucial result, which states that a fair maximal path in the semantics of a pathway is either projected or infinitely projected into an abstractly fair maximal path in the abstract semantics of an abstract pathway. It is split in two lemmas, where the first one states that at least one of the projections is present as a maximal path in the abstract semantics. The second lemma proves the abstract fairness of the projections.

Lemma 1. *$\pi \in LTS(P)$ implies ($\pi\lceil J \in LTS_\alpha(P\restriction J)$ or $\pi\lceil^\infty J \in LTS_\alpha(P\restriction J)$).*

Proof. We prove a stronger statement, namely $\pi \in LTS(P)$ with initial state $\mathbf{s}$ implies ($\pi\lceil J \in LTS_\alpha(P\restriction J)$ with initial state $\mathbf{s}\lceil J$ or $\pi\lceil^\infty J \in LTS_\alpha(P\restriction J)$ with initial state $\mathbf{s}\lceil J$). We proceed by induction on path π.

Case $\pi = \varepsilon$. We have that $\pi\lceil J = \varepsilon$ and $\pi\lceil^\infty J = \varepsilon^\infty$. Since $\pi = \varepsilon$, the initial state $\mathbf{s_0}$ is such that no reaction is enabled in $\mathbf{s_0}$. By the definition of abstract pathway semantics we know that the initial state of $LTS_\alpha(P\restriction J)$ is $\mathbf{s_0}\lceil J$. Let R be any reaction from P, we consider three cases:

- $comp(R) \cap J = \emptyset$: R is not in $P\restriction J$, thus $\varepsilon = \pi\lceil J \in LTS_\alpha(P\restriction J)$.
- $comp(R) \subseteq J$: R is in $P\restriction J$ but by contradiction we have that R is not enabled in $\mathbf{s_0}\lceil J$, thus $\varepsilon = \pi\lceil J \in LTS_\alpha(P\restriction J)$.
- $comp(R) \cap J \neq \emptyset$ and $comp(R) \cap (I \setminus J) \neq \emptyset$: in $P\restriction J$ are the following two reactions

$$R_1 = re(R)\restriction J \ \rightarrow \ pro(R)\restriction J \ \{ \ cat(R)\restriction J \ \}$$
$$R_2 = re(R)\restriction J \ \rightarrow \ re(R)\restriction J \ \{ \ cat(R)\restriction J \ \}$$

Cases:

- $\mathbf{s_0}\lceil J \not\supseteq re(R)\restriction J \cup cat(R)\restriction J$: none of R_1 and R_2 are enabled in $\mathbf{s_0}\lceil J$, thus $\varepsilon = \pi\lceil J \in LTS_\alpha(P\restriction J)$.
- $\mathbf{s_0}\lceil J \supseteq re(R)\restriction J \cup cat(R)\restriction J$ and $\mathbf{s_0}\lceil J \not\supseteq pro(R)\restriction J$: both R_1 and R_2 are enabled in $\mathbf{s_0}\lceil J$, from the latter we have $\varepsilon^\infty = \pi\lceil^\infty J \in LTS_\alpha(P\restriction J)$.
- $\mathbf{s_0}\lceil J \supseteq re(R)\restriction J \cup cat(R)\restriction J \cup pro(R)\restriction J$: R_1 is not enabled in $\mathbf{s_0}\lceil J$, but is R_2 is and since it does not change the state it will be enabled forever, thus $\varepsilon^\infty = \pi\lceil^\infty J \in LTS_\alpha(P\restriction J)$.

Case $\pi = \mathbf{s}, R, \pi'$. We have that $\pi\lceil J = \mathbf{s}\lceil J, R, \pi'\lceil J$ and $\pi\lceil^\infty J = \mathbf{s}\lceil J, R, \pi'\lceil^\infty J$. We distinguish two cases:

- $comp(R) \cap J = \emptyset$: $\mathbf{s}'$ is the initial state of π', so by induction hypothesis $\mathbf{s}'\lceil J$ is the initial state of either $\pi'\lceil J$ or $\pi'\lceil^\infty J$. Moreover, state $\mathbf{s}\lceil J = \mathbf{s}'\lceil J$ then $\mathbf{s}\lceil J$ is the initial state of either $\pi'\lceil J$ or $\pi'\lceil^\infty J$, which means that either $\pi'\lceil J = \pi\lceil J \in LTS_\alpha(P\restriction J)$ or $\pi'\lceil^\infty J = \pi\lceil^\infty J \in LTS_\alpha(P\restriction J)$.
- $comp(R) \cap J \neq \emptyset$: $\mathbf{s}'$ is the initial state of π', so by induction hypothesis $\mathbf{s}'\lceil J$ is the initial state of either $\pi'\lceil J$ or $\pi'\lceil^\infty J$. Moreover, in $P\restriction J$ there are R_1 and R_2 as above. R_1 is enabled in $\mathbf{s}\lceil J$. Therefore either $\mathbf{s}\lceil J, R, \pi'\lceil J = \pi\lceil J \in LTS_\alpha(P\restriction J)$ of $\mathbf{s}\lceil J, R, \pi'\lceil^\infty J = \pi\lceil^\infty J \in LTS_\alpha(P\restriction J)$.

In all the cases we have that $\pi\lceil J \in LTS_\alpha(P\restriction J)$ or $\pi\lceil^\infty J \in LTS_\alpha(P\restriction J)$, which concludes the proof of the lemma. □

Now we state and prove the second lemma.

Lemma 2. *$\pi \vDash_{LTL} \Phi$ implies $\pi\lceil J \vDash_{LTL} \Phi_\alpha$ and $\pi\lceil^\infty J \vDash_{LTL} \Phi_\alpha$.*

Proof. Suppose that $\pi \vDash_{LTL} \Phi$, i.e. $\pi \vDash_{LTL} \bigwedge_{R\in P}(GF\ enabled(R) \rightarrow GF\ occurred(R))$. Let $J \subseteq I$ and $P\restriction J = (AR, PR)$. We want to prove that $\pi\lceil J \vDash_{LTL} \Phi_\alpha$, that is $\pi\lceil J \vDash_{LTL} \bigwedge_{R\in PR}(GF\ enabled(R) \rightarrow GF\ occurred(R))$ This holds because of two facts (1) and (2) that can be easily checked: for any reaction R from PR

- $\mathbf{s} \vDash_{LTL} enabled(R)$ implies $\mathbf{s}\lceil J \vDash_{LTL} enabled(R)$ (1)
- $\mathbf{s} \vDash_{LTL} occurred(R)$ implies $\mathbf{s}\lceil J \vDash_{LTL} occurred(R)$ (2)

Analogously $\pi\lceil^\infty J \vDash_{LTL} \bigwedge_{R\in PR}(GF\ enabled(R) \rightarrow GF\ occurred(R))$. □

Finally, the property preservation theorem states that a successful verification of a property in the abstraction implies its truth in the original model.

Theorem 3. *For a pathway P and a $J \subseteq I$ where I is the component set of P and f an ACTL$^-$ formula we have $LTS_\alpha(P\restriction J) \vDash_{\Phi_\alpha} f$ implies $LTS(P) \vDash_\Phi f$.*

Proof. By induction on the structure of f (for all $\mathbf{s}$).

$f = s$. By definition of state projection and the fact that AP_Rs are pairwise disjoint, for all atomic propositions s from AP_J we get that $LTS_\alpha(P\restriction J), \mathbf{s}\lceil J \vDash_{\Phi_\alpha} s$ iff $LTS(P), \mathbf{s} \vDash_\Phi s$. Analogously for $f = \neg s$.

$f = g \wedge h$. From the assumption $LTS_\alpha(P\restriction J), \mathbf{s}\lceil J \vDash_{\Phi_\alpha} g \wedge h$ by CTL semantics, $LTS_\alpha(P\restriction J), \mathbf{s}\lceil J \vDash_{\Phi_\alpha} g$ and $LTS_\alpha(P\restriction J), \mathbf{s}\lceil J \vDash_{\Phi_\alpha} h$. By induction hypothesis $LTS(P), \mathbf{s} \vDash_\Phi g$ and $LTS(P), \mathbf{s} \vDash_\Phi h$. Hence, $LTS(P), \mathbf{s} \vDash_\Phi g \wedge h$. Case $f = g \vee h$ is proved analogously.

$f = A[g\ U_w\ h]$. Let π be an arbitrary fair maximal path starting in $\mathbf{s}$. We establish $LTS(P), \pi \vDash_\Phi [g\ U_w\ h]$. By Lemma 1 at least one of $\pi\lceil J$ or $\pi\lceil^\infty J$ is a path in $LTS_\alpha(P\restriction J)$, and by Lemma 2 both are abstractly fair.

Let us suppose first that $\pi\lceil J$ is the abstractly fair maximal path in $LTS_\alpha(P\restriction J)$. Hence by the assumption $LTS_\alpha(P\restriction J), \pi\lceil J \vDash_{\Phi_\alpha} [g\ U_w\ h]$. There are two cases:

1. $LTS_\alpha(P\restriction J), \pi\lceil J \vDash_{\Phi_\alpha} G\ g$. Let t be any state along π. By CTL semantics $LTS_\alpha(P\restriction J), t\lceil J \vDash_{\Phi_\alpha} g$. By induction hypothesis we have $LTS(P), t \vDash_\Phi g$. Since t was an arbitrary state of π, we get $LTS(P), \pi \vDash_\Phi G\ g$ and thus $LTS(P), \pi \vDash_\Phi g\ U_w\ h$.
2. $LTS_\alpha(P\restriction J), \pi\lceil J \vDash_{\Phi_\alpha} [g\ U\ h]$. Let $\mathbf{s}_J^{m''}$ be the first state along $\pi\lceil J$ that satisfies h. Then there is at least one state $\mathbf{s}^{m''}$ along π such that $\mathbf{s}^{m''}\lceil J = \mathbf{s}_J^{m''}$. Let $\mathbf{s}^{m'}$ be first such state. By induction hypothesis $LTS(P), \mathbf{s}^{m'} \vDash_\Phi h$. From the definition of path projection any $\mathbf{s}^m$ with $m < m'$ projects to $\mathbf{s}^m\lceil J$ that is before $\mathbf{s}_J^{m'}$ in $\pi\lceil J$. By the assumption $LTS_\alpha(P\restriction J), \mathbf{s}^m\lceil J \vDash_{\Phi_\alpha} g$, hence by induction hypothesis $LTS(P), \mathbf{s}^m \vDash_\Phi g$. By CTL semantics we get $LTS(P), \pi \vDash_\Phi g\ U\ h$.

In both cases we showed $LTS(P), \pi \vDash_\Phi g\ U_w\ h$. Since π was arbitrary fair maximal path starting in $\mathbf{s}$, we conclude $LTS(P), \mathbf{s} \vDash_\Phi A[g\ U_w\ h]$.

The reasoning for the case in which the abstractly fair maximal path in $LTS_\alpha(P\restriction J)$ is $\pi\lceil^\infty J$ is analogous to the considered case.

$f = A[g\ U\ h]$. Let π be an arbitrary fair maximal path starting in $\mathbf{s}$. By Lemmas 1 and 2 we have that $\pi\lceil J$ or $\pi\lceil^\infty J$ is a fair maximal path in $LTS_\alpha(P\restriction J)$ and by the assumption $LTS_\alpha(P\restriction J), \pi\lceil J \vDash_{\Phi_\alpha} [g\ U\ h]$ or $LTS_\alpha(P\restriction J), \pi\lceil^\infty J \vDash_{\Phi_\alpha} [g\ U\ h]$. By the above case we get $LTS(P), \mathbf{s} \vDash_\Phi A[g\ U\ h]$. □

5 Experiments

In this section we exploit the NuSMV model checker to perform some experiments on the model of the EGF pathway. To carry out the projection and encode the resulting abstract pathway in the NuSMV format we have developed a tool (available upon request).

The first experiment is aimed at showing how modular verification could be applied to verify a global property of the pathway, namely that the final product of the pathway is always produced. This can be done in a modular way by proving sub-properties in three different model fragments obtained by projection.

Subsequently, a number of experiments are performed with the aim of showing how the molecular components we identified in the pathway can be used to better understand the pathway dynamics. In particular, we check whether there are some molecular components that are not really necessary to obtain the final product of the pathway. This will be done by applying model checking on models in which molecular components are selectively disabled by setting their initial states to false. Also in this case the modular verification approach is adopted.

In this case study modular verification allows properties to be verified faster than on the complete model. However, modular verification is still not significantly more efficient than verification on the complete model. This is due to the projection operation we are considering at the moment, which is rather rough. In Section 6 we discuss why this modular verification is a promising approach for the analysis of pathways, and how we plan to improve the approach to make it substantially more efficient.

To run the experiment we used NuSMV 2.5.4 on a workstation equipped with an Intel i5 CPU 2.80 Ghz, with 8GB RAM and running Ubuntu GNU/Linux. In order to make verification faster NuSMV was executed in batch mode by enabling dynamic reordering of BDD variables and by disabling the generation of counterexamples.

5.1 Modular verification of a global property

The final product of the MAP kinase cascade activated by surface and internalised EGF receptors is species ERK-PP. Since surface and internalised receptors activate two different branches of the pathway,

we denote by ERK-PP the product of the branch activated by the surface receptors and by ERK-PPi the product of the branch activated by the internalised receptors.

The property to be verified is

$$AF(\text{ERK-PP} \vee \text{ERK-PPi}) \tag{1}$$

The property holds in the complete model and its verification required 260 seconds. By looking at the diagram in Figure 1 we noticed that the pathway could be partitioned in three parts, with two species acting as "gates". These two species are (EGF-EGFR*)2-GAP and Raf*. Hence, we decided to try to apply modular verification by splitting property 1 into the following three sub-properties:

$$AF((\text{EGF-EGFR}^*)2\text{-GAP}) \tag{2}$$

$$AG((\text{EGF-EGFR}^*)2\text{-GAP} \rightarrow (AF\ \text{Raf}^*)) \tag{3}$$

$$AG(\text{Raf}^* \rightarrow AF(\text{ERK-PP} \vee \text{ERK-PPi})) \tag{4}$$

Property (2) states that in all paths of the system a state in which species (EGF-EGFR*)2-GAP is present is eventually reached. Property (3) states that whenever a state is reached in which species (EGF-EGFR*)2-GAP is present, then a state in which Raf* is present is eventually reached. Finally, property (4) states that whenever a state is reached in which in which species Raf* is present, then a state in which either ERK-PP or ERK-PPi is present is eventually reached. It is easy to see that the conjunction of (2), (3) and (4) implies (1).

We considered three projections of the complete model to be used to verify properties (2), (3) and (4), respectively. In particular, from the component interaction graph of the model (shown in Figure 2) we extracted the following subsets to be used for projections:

- in order to verify (2) we considered the subset J_1 consisting of components *EGF*, *EGFi*, *EGFR* and *GAP*;
- in order to verify (3) we considered the subset J_2 consisting of components *EGFR*, *GAP*, *Shc*, *RasGDP*, *Grb2* and *Sos*;
- in order to verify (4) we considered the subset J_3 consisting of components *RasGDP*, *Raf*, *MEK*, *ERK*, *Phosphatase1*, *Phosphatase2* and *Phosphatase3*.

We obtained that (2), (3) and (4) hold in the abstract semantics of the abstract pathways $P{\restriction}J_1$, $P{\restriction}J_2$ and $P{\restriction}J_3$, respectively. Moreover, model checking required less than three seconds for (2), 213 seconds for (3) and less than one second for (4). Overall, modular verification required 217 seconds, that is 43 seconds less than verification on the complete model.

5.2 Reasoning on molecular components

As it can be seen in the component interaction graph and in the diagram in Figure 1, some molecular components are involved in complex interactions. This is true in particular for components *EGFR*, *GAP*, *RasGDP*, *Sos*, *Shc* and *Grb2* which form a clique in the component interaction graph. We are interested in understanding whether all of these components are really necessary in order to obtain the final products of the pathway. The idea is to test whether the final species are produced when the components of interest are assumed one by one as disabled. Molecular components *EGFR* and *RasGDP* are for sure necessary since they connect the clique with the other molecular components of the pathway. Consequently, we focus our analysis on *GAP*, *Sos*, *Shc* and *Grb2*.

In order to disable a molecular component we consider as absent all of its species in the initial state of the systems. Hence, we consider a set of four (complete) models, each with one of the four components

	Verification complete model			Modular Verification			
Disabled component	**Property**	**Result**	**Time**	**Property**	**Result**	**Time**	**Total time**
none	(1)	true	260s	(2) (3) (4)	true true true	3s 213s 1s	217s
GAP	(1)	false	252s	(2),(5)	false,true	2s	2s
Sos	(1)	false	253s	(2) (3),(6)	true false,true	3s 207s	210s
Shc	(1)	true	252s	(2) (3) (4)	true true true	3s 208s 1s	212s
Grb2	(1)	false	253s	(2) (3),(6)	true false,true	3s 208s	211s

Table 1: Model checking results and comparison of verification times

under study disabled. On each model we try to verify property (1): if the property does not hold, then the component that is disabled in such a model is necessary for the pathway; on the other hand, if the property holds, then the component turns out to be not necessary since the products of the pathway can be obtained even without it. The same tests can be also done in a modular way by decomposing the pathway and the property as in Section 5.1.

In Table 1 we summarise the property verification results and compare verification times obtained by model checking the complete models and by following the modular approach. The first row of data in the table reports verification results in which no component is disabled (as in Section 5.1). The other results show that *Shc* is not a necessary component, whereas all of the other three are. As previously, the time required by modular verification is smaller than the one required by model checking the complete model. This is true in particular in the case in which *GAP* is disabled since property (2), the verification of which is very fast, turns out to be false.

Note that in the case of modular verification of the models in which *GAP*, *Sos* and *Grb2* were disabled we needed to verify some additional properties. In particular, in the case of *GAP* we have that property (2) does not hold in the abstract semantics of $P{\uparrow}J_1$, and in the cases of *Sos* and *Grb2* property (3) does not hold in the abstract semantics of $P{\uparrow}J_2$. We remark that our modular verification approach guarantees only that properties proved to hold in a model fragment also hold in the complete model. Nothing can be said, instead, of properties that does not hold in the model fragments. In order to avoid applying model checking on the complete model to check whether these properties hold there, we consider some new properties whose satisfaction in suitable model fragments implies that properties (2) and (3) actually do not hold. In order to prove that (2) is actually false when *GAP* is disabled we consider the following property:

$$AG(\neg(\text{EGF-EGFR}^*)\text{2-GAP}) \tag{5}$$

In order to prove that (3) is actually false when either *Sos* or *Grb2* is disabled we consider the following property:

$$AG(\neg\text{Raf}^*) \tag{6}$$

Note that it is convenient to verify properties (5) and (6) together with (2) and (3), respectively. This avoids spending twice the time needed by the model checker to construct the data structure necessary

to perform the verification. In the case of our experiments the construction of such data structures takes usually the 98%-99% of the verification time. Times reported in Table 1 are based on this optimisation.

6 Discussion and conclusions

In this paper we presented preliminary results in the development of a modular verification framework for biochemical pathways. We defined a modelling notation for pathways associated with a formal semantics and a notion of fairness that allows the dynamics to be accurately described by avoiding starvation situations among reactions. Moreover, we investigated a notion of molecular component of a pathway and we provided a methodology to infer molecular components from pathways the reactions of which satisfy some assumptions. Molecular components were then used by a projection operation that allows abstract pathways modelling an over-approximation of the behaviour of a group of components to be obtained from a pathway model. The fact that a property expressed by means of the ACTL$^-$ logic holds in an abstract pathway was shown to imply that they hold also in the complete pathway model. This preservation is at the basis of the modular verification approach which was demonstrated on a well-established model of the EGF pathway.

The results of experiments given in Section 5 show that our modular verification approach allows properties to be verified in a shorter time than in the case of verification of the complete pathway model. However, in most of the cases the time saved was relatively small ($\sim 15\%$). We believe that the cause of this limited gain in efficiency is due to the projection operation we are considering at the moment, which is still somewhat rough. Our plan to improve efficiency is to define a projection operation that combines the current one (that essentially removes some molecular components from the model) with another that somehow minimises the description of components not removed by the model, but whose role in the property to be verified is marginal. In the case of the considered case study this would allow, for example, to reduce the size of the model of the components constituting the clique in the component interaction graph in Figure 2 by focusing on components *EGFR* and *RasGDP*, and by minimising the description of components *GAP*, *Shc*, *Sos* and *Grb2*. This would allow for a significant improvement in modular verification efficiency.

References

[1] Paul C. Attie & E. Allen Emerson (1998): *Synthesis of Concurrent Systems with Many Similar Processes*. ACM Transactions on Programming Languages and Systems 20(1), pp. 51–115.

[2] Roberto Barbuti, Andrea Maggiolo-Schettini, Paolo Milazzo & Angelo Troina (2006): *A Calculus of Looping Sequences for Modelling Microbiological Systems*. Fundamenta Informaticae 72(1-3), pp. 21–35.

[3] Jerry R. Burch, Edmund M. Clarke, Kenneth L. McMillan, David L. Dill & L. J. Hwang (1992): *Symbolic Model Checking: 10^{20} States and Beyond*. Information and Computation 98(2), pp. 142–170.

[4] Luca Cardelli (2005): *Brane Calculi*. Computational Methods in Systems Biology, pp. 257–278.

[5] Alessandro Cimatti, Edmund Clarke, Enrico Giunchiglia, Fausto Giunchiglia, Marco Pistore, Marco Roveri, Roberto Sebastiani & Armando Tacchella (2002): *NuSMV Version 2: An OpenSource Tool for Symbolic Model Checking*. In: Proc. International Conference on Computer-Aided Verification (CAV 2002), LNCS 2404, Springer, Copenhagen, Denmark.

[6] Federica Ciocchetta & Jane Hillston (2009): *Bio-PEPA: A Framework for the Modelling and Analysis of Biological Systems*. Theoretical Computer Science 410(33-34), pp. 3065–3084.

[7] Edmund M. Clarke, Orna Grumberg & David E. Long (1994): *Model Checking and Abstraction*. ACM Transactions on Programming Languages and Systems 16(5), pp. 1512–1542.

[8] Edmund M. Clarke, Orna Grumberg & Doron Peled (1999): *Model Checking*. MIT Press.

[9] Vincent Danos & Cosimo Laneve (2004): *Formal Molecular Biology*. Theoretical Computer Science 325(1), pp. 69–110.

[10] Peter Drábik, Andrea Maggiolo-Schettini & Paolo Milazzo (2010): *Dynamic Sync-programs for Modular Verification of Biological Systems*. In: 2nd Int. Workshop on Non-Classical Models of Automata and applications (NCMA'10), 263, Austrian Computer Society, Jena, Germany.

[11] Peter Drábik, Andrea Maggiolo-Schettini & Paolo Milazzo (2010): *Modular Verification of Interactive Systems with an Application to Biology*. Electronic Notes in Theoretical Computer Science 268, pp. 61–75.

[12] Peter Drábik, Andrea Maggiolo-Schettini & Paolo Milazzo (2011): *Modular Verification of Interactive Systems with an Application to Biology*. Scientific Annals of Computer Science 21, pp. 39–72.

[13] E. Allen Emerson & Chin-Laung Lei (1987): *Modalities for Model Checking: Branching Time Logic Strikes Back*. Science of Computer Programming 8, pp. 275–306.

[14] François Fages, Sylvain Soliman & Nathalie Chabrier-Rivier (2004): *Modelling and Querying Interaction Networks in the Biochemical Abstract Machine Biocham*. Journal of Biological Physics and Chemistry 4, pp. 64–73.

[15] John Heath, Marta Kwiatkowska, Gethin Norman, David Parker & Oksana Tymchyshyn (2008): *Probabilistic Model Checking of Complex Biological Pathways*. Theoretical Computer Science 391(3), pp. 239–257.

[16] Pedro T. Monteiro, Delphine Ropers, Radu Mateescu, Ana T. Freitas & Hidde de Jong (2008): *Temporal Logic Patterns for Querying Dynamic Models of Cellular Interaction Networks*. Bioinformatics 24(16), pp. i227–233.

[17] Amir Pnueli (1981): *A Temporal Logic of Concurrent Programs*. Theoretical Computer Science 13, pp. 45 – 60.

[18] Corrado Priami, Aviv Regev, Ehud Shapiro & William Silverman (2001): *Application of a Stochastic Name-passing Calculus to Representation and Simulation of Molecular Processes*. Information Processing Letters 80(1), pp. 25–31.

[19] Aviv Regev, Ekaterina M. Panina, William Silverman, Luca Cardelli & Ehud Shapiro (2004): *BioAmbients: An Abstraction for Biological Compartments*. Theoretical Computer Science 325(1), pp. 141–167.

[20] Birgit Schoeberl, Claudia Eichler-Jonsson, Ernst Dieter Gilles & Gertraud Muller (2002): *Computational Modeling of the Dynamics of the MAP Kinase Cascade Activated by Surface and Internalized EGF Receptors*. Nature Biotechnology 20(4), pp. 370–375.

Implementing the Stochastics Brane Calculus in a Generic Stochastic Abstract Machine

Marino Miculan Ilaria Sambarino
Department of Mathematics and Computer Science, University of Udine, Italy
marino.miculan@uniud.it ilaria.sambarino@gmail.com

In this paper, we deal with the problem of implementing an abstract machine for a stochastic version of the Brane Calculus. Instead of defining an *ad hoc* abstract machine, we consider the generic stochastic abstract machine introduced by Lakin, Paulevé and Phillips. The nested structure of membranes is flattened into a set of species where the hierarchical structure is represented by means of *names*. In order to reduce the overhead introduced by this encoding, we modify the machine by adding a *copy-on-write* optimization strategy. We prove that this implementation is adequate with respect to the stochastic structural operational semantics recently given for the Brane Calculus. These techniques can be ported also to other stochastic calculi dealing with nested structures.

1 Introduction

A fundamental issue in Systems Biology is modelling the membrane interaction machinery. Several models have been proposed in the literature [11, 14, 3]; among them, the Brane Calculus [4] has been arisen as a good model focusing on abstract membrane interactions, still being sound with respect to biological constraints (e.g. bitonality). In this calculus, a process represents a system of nested compartments, where active components are *on* membranes, not inside them. This reflects the biological evidence that functional molecules (proteins) are embedded in membranes, with consistent orientation.

In the original definition of the Brane Calculus [4] (which we will recall in Section 2) membranes interact according to three basic reaction rules corresponding to *phagocytosis*, *endo/exocytosis*, and *pinocytosis*. However, this semantics does not take into account quantitative aspects, like stochastic distributions, which are important for, e.g., implementing stochastic simulations.

A stochastic semantics for the Brane Calculus has been provided in [2], following an approach pioneered in [5] (but see also [8, 12] for Markov processes). Instead of giving a stochastic version $P \xrightarrow{a,r} Q$ of the reaction relation, in this semantics each process is given a *measure* of the stochastic distribution of the possible outcomes. More precisely, we define a relation $P \to \mu$ associating to a process P an action-indexed family of measures μ: for an action a, the measure μ_a specifies for each measurable set S of processes, the rate $\mu_a(S) \in \mathbb{R}^+$ of a-transitions from P to (elements of) S. An advantage of this approach is that we can apply results from measure theory for solving otherwise difficult issues, like instance-counting problems; moreover, process measures are defined *compositionally*, and in fact the relation $P \to \mu$ can be characterized by means of a set of rules in a GSOS-like format. We will recall this stochastic semantics and its main properties in Section 3.

In this paper, we use this new semantics for defining a *stochastic abstract machine* for the Brane Calculus, so that it can be effectively used for *in silico* simulations of membrane systems. Defining an *ad hoc* abstract machine for the Brane Calculus would be a complex task; instead, we take advantage of the *generic abstract machine for stochastic process calculi* (GSAM for short) introduced in [13, 10] as a general tool for simulating a broad range of calculi. This machine can be instantiated to a particular

Pre-Proceedings of 6th Workshop on Membrane Computing and Biologically Inspired Process Calculi
MeCBIC 2012, Pages 83–101.

calculus by defining a function for transforming a process of the calculus to a set of *species*, and another for computing the set of possible reactions between species.

An important aspect is that this abstract machine does not have a native notion of compartment, which is central in the Brane Calculus (as in any other model of membranes). To overcome this problem, we adopt a "flat" representation of membrane systems, used also in [10], where the hierarchical structure is represented by means of *names*: each name represents a compartment, and each species is labelled with the name of the compartment where it is located, and the name of its inner compartment (if any). So names and species are the nodes and the arcs of the tree, respectively. This technique can be used for representing any system with a tree-like structure of compartments.

However, this approach does not scale well, as the population of species may grow enormously: for instance, a population of n identical cells would lead to n different single species, differing only for the name of its inner compartment, instead of a single specie with multiplicity n. For circumventing this problem, in Section 4 we introduce a variant of the GSAM with a *copy-on-write* optimization strategy—hence called COWGSAM. The idea is to keep a single copy of each species, with its multiplicity; when a reaction has to be applied, fresh copies of the compartments involved are generated on-the-fly, and reactions and rates are updated accordingly. In this way, the hierarchical structure is unfolded only if and when needed.

In Section 5 we show how the Brane Calculus can be represented in the COWGSAM, and we will prove that the abstract machine obtained in this way is adequate with respect to the stochastic semantics of the Brane Calculus; in this proof, we take advantage of the compositional definition of this semantics.

Conclusions and final remarks are in Section 6.

2 Brane Calculus

In this section we recall Cardelli's Brane Calculus [4] focusing on its basic version (without communication primitives, complexes and replication).

First, let us fix the notation we will use hereafter. Let S be a set of *sorts* (or "types"), ranged over by s,t, and T a set of S-sorted terms; for $t \in S$, $T_t \subseteq T$ denotes the set of terms of sort t. For A a set of symbols, A^* denotes the set of finite words (or lists) over A, and $\langle a_1,\ldots,a_n\rangle$ denotes a word in A^*. For a word $\langle t_1,\ldots,t_n\rangle$ in S^*, we define $T_{\langle t_1,\ldots,t_n\rangle} \triangleq T_{t_1} \times \cdots \times T_{t_n}$.

Syntax The sorts and the set $\mathbb{B}$ of terms of Brane Calculus are the following:

Sorts :: S	$t ::= \mathsf{sys} \mid \mathsf{mem}$
Membranes :: $\mathbb{B}_{\mathsf{mem}}$	$\sigma,\tau ::= \mathbf{0} \mid \sigma\vert\tau \mid \mathbf{J}_n.\sigma \mid \mathbf{J}_n^{\mathsf{I}}(\tau).\sigma \mid \mathbf{K}_n.\sigma \mid \mathbf{K}_n^{\mathsf{I}}.\sigma \mid \mathbf{G}_n(\tau).\sigma$
Systems :: $\mathbb{B}_{\mathsf{sys}}$	$P,Q ::= \mathbf{k} \mid P \,\mathtt{m}\, Q \mid \sigma\mathbf{h}P\mathbf{i}$

The subscripted names n are taken from a countable set Λ. By convention we shall use M, N, ... to denote generic Brane Calculus terms in $\mathbb{B}$.

A *membrane* can be either the empty membrane $\mathbf{0}$, or the parallel composition of two membranes $\sigma|\tau$, or the action-prefixed membrane $\varepsilon.\sigma$. Actions are: *phagocytosis* $\mathbf{J}$, *exocytosis* $\mathbf{K}$, and *pinocytosis* $\mathbf{G}$. Each action but pinocytosis comes with a matching co-action, indicated by the superscript $^{\mathsf{I}}$.

A *system* can be either the empty system $\mathbf{k}$, or the parallel composition $P \,\mathtt{m}\, Q$, or the system nested within a membrane $\sigma\mathbf{h}P\mathbf{i}$. Notice that, differently from [4], pino actions are indexed by names in Λ. In [4], names are meant only to pair up an action with its corresponding co-action, hence a pino action

$$\frac{}{\mathbf{J}_n^!(\rho).\tau|\tau_0\mathbf{h}Q\mathbf{i}\,\mathbf{m}\,\mathbf{J}_n.\sigma|\sigma_0\mathbf{h}P\mathbf{i}\ \}\ \tau|\tau_0\mathbf{h}\rho\mathbf{h}\sigma|\sigma_0\mathbf{h}P\mathbf{i}\mathbf{i}\,\mathbf{m}\,Q\mathbf{i}}\ \text{(red-phago)}$$

$$\frac{}{\mathbf{K}_n^!.\tau|\tau_0\mathbf{h}\mathbf{K}_n.\sigma|\sigma_0\mathbf{h}P\mathbf{i}\,\mathbf{m}\,Q\mathbf{i}\ \}\ \sigma|\sigma_0|\tau|\tau_0\mathbf{h}Q\mathbf{i}\,\mathbf{m}\,P}\ \text{(red-exo)}$$

$$\frac{}{\mathbf{G}(\rho).\sigma|\sigma_0\mathbf{h}P\mathbf{i}\ \}\ \sigma|\sigma_0\mathbf{h}\rho\mathbf{h}\mathbf{k}\mathbf{i}\,\mathbf{m}\,P\mathbf{i}}\ \text{(red-pino)} \qquad \frac{P\ \}\ Q}{\sigma\mathbf{h}P\mathbf{i}\ \}\ \sigma\mathbf{h}Q\mathbf{i}}\ \text{(red-loc)}$$

$$\frac{P\ \}\ Q}{P\,\mathbf{m}\,R\ \}\ Q\,\mathbf{m}\,R}\ \text{(red-comp)} \qquad \frac{P\equiv P' \quad P'\ \}\ Q' \quad Q'\equiv Q}{P\ \}\ Q}\ \text{(red-equiv)}$$

Table 1: Reduction semantics for the Brane Calculus.

does not need to be indexed by any name. Actually, names can be thought of as an abstract representation of particular protein conformational shapes; hence, each name can correspond to a different biological behaviour. Therefore, if we want to observe also kinetic properties of processes, it is important to keep track of names in pino actions.

Terms can be rearranged according to a structural congruence relation; the intended meaning is that two congruent terms actually denote the same system. Structural congruence $\equiv$ is the smallest equivalence relation over $\mathbb{B}$ which satisfies the axioms and rules listed below.

$$P\,\mathbf{m}\,Q\equiv Q\,\mathbf{m}\,P \qquad P\,\mathbf{m}\,(Q\,\mathbf{m}\,R)\equiv(P\,\mathbf{m}\,Q)\,\mathbf{m}\,R \qquad P\,\mathbf{m}\,\mathbf{k}\equiv P$$

$$\sigma|\tau\equiv\tau|\sigma \qquad \sigma|(\tau|\rho)\equiv(\sigma|\tau)|\rho \qquad \sigma|\mathbf{0}\equiv\sigma$$

$$\mathbf{0}\mathbf{h}\mathbf{k}\mathbf{i}\equiv\mathbf{k} \qquad \frac{P\equiv Q}{P\,\mathbf{m}\,R\equiv Q\,\mathbf{m}\,R} \qquad \frac{\sigma\equiv\tau}{\sigma|\rho\equiv\tau|\rho} \qquad \frac{P\equiv Q \quad \sigma\equiv\tau}{\sigma\mathbf{h}P\mathbf{i}\equiv\tau\mathbf{h}Q\mathbf{i}}$$

$$\frac{\alpha\in\{\mathbf{J}_n,\mathbf{K}_n,\mathbf{K}_n^!\}_{n\in\Lambda} \quad \sigma\equiv\tau}{\alpha.\sigma\equiv\alpha.\tau} \qquad \frac{\beta\in\{\mathbf{J}_n^!,\mathbf{G}_n\}_{n\in\Lambda} \quad \rho\equiv\nu \quad \sigma\equiv\tau}{\beta(\rho).\sigma\equiv\beta(\nu).\tau}$$

Differently from [4], we allow to rearrange also the sub-membranes contained in co-phago and pino actions (by means of the last inference rule above).

Reduction Semantics The dynamic behaviour of Brane Calculus is specified by means of a reduction semantics, defined over a reduction relation ("reaction") $\}\ \subseteq \mathbb{B}_{\mathsf{sys}}\times\mathbb{B}_{\mathsf{sys}}$, whose rules are listed in Table 1. Notice that the presence of (red-phago/exo/pino) and (red-equiv) makes this not a *structural* presentation, since these rules are not primitive recursive in the syntax (i.e., structural recursive) as required by the SOS format.

3 Stochastic Structural Operational Semantics for the Brane Calculus

In this section we recall the stochastic structural operational semantics for the Brane Calculus, as defined in [2]. Following the pattern of [5], we replace the classic "pointwise" rules of the form $P\xrightarrow{a,r}P'$ with rules of the form $P\to\mu$, where μ is an indexed class of *measures* on the *measurable space* of processes. We assume the reader to be familiar with basic notions from measure theory; for a brief summary, see Appendix A.

$$\frac{}{\mathbf{0} \to_{\mathsf{mem}} \omega_{\mathsf{mem}}}\ (\text{zero}) \qquad \frac{\alpha \in \{\mathbf{J}_n, \mathbf{K}_n, \mathbf{K}_n^! \mid n \in \Lambda\}}{\alpha.\sigma \to_{\mathsf{mem}} [\alpha]_\sigma}\ (\text{pref})$$

$$\frac{\beta \in \{\mathbf{J}_n^!, \mathbf{G}_n \mid n \in \Lambda\}}{\beta(\tau).\sigma \to_{\mathsf{mem}} [\beta]_\sigma^\tau}\ (\text{pref-arg}) \qquad \frac{\sigma \to_{\mathsf{mem}} \mu' \qquad \tau \to_{\mathsf{mem}} \mu''}{\sigma|\tau \to_{\mathsf{mem}} \mu' {}_\sigma\oplus_\tau \mu''}\ (\text{par})$$

$$\frac{}{\mathbf{k} \to_{\mathsf{sys}} \omega_{\mathsf{sys}}}\ (\text{void}) \qquad \frac{\sigma \to_{\mathsf{mem}} \nu \qquad P \to_{\mathsf{sys}} \mu}{\sigma \mathbf{h} P \mathbf{i} \to_{\mathsf{sys}} \mu @_P^\sigma \nu}\ (\text{loc}) \qquad \frac{P \to_{\mathsf{sys}} \mu' \qquad Q \to_{\mathsf{sys}} \mu''}{P \,\mathtt{m}\, Q \to_{\mathsf{sys}} \mu' {}_P\otimes_Q \mu''}\ (\text{comp})$$

Table 2: Stochastic structural operational semantics for Brane Calculus

The set of action labels for the Brane Calculus will be denoted by $\mathbb{A}$ and can be partitioned with respect to the source sort (i.e., either systems or membranes), as follows:

$$\begin{aligned}
\mathbb{A}_{\mathsf{sys}} \triangleq\ & \{id : \mathsf{sys} \to \langle \mathsf{sys} \rangle\} \cup \{\mathrm{ph}_n : \mathsf{sys} \to \langle \mathsf{sys}, \mathsf{sys} \rangle \mid n \in \Lambda\} \cup \\
& \{\mathrm{ph}_n^\perp : \mathsf{sys} \to \langle \mathsf{mem}, \mathsf{mem}, \mathsf{sys}, \mathsf{sys} \rangle \mid n \in \Lambda\} \cup \\
& \{\mathrm{ex}_n : \mathsf{sys} \to \langle \mathsf{mem}, \mathsf{sys}, \mathsf{sys} \rangle \mid n \in \Lambda\} \\
\mathbb{A}_{\mathsf{mem}} \triangleq\ & \{\mathbf{J}_n, \mathbf{K}_n, \mathbf{K}_n^! : \mathsf{mem} \to \langle \mathsf{mem} \rangle \mid n \in \Lambda\} \cup \\
& \{\mathbf{J}_n^!, \mathbf{G}_n : \mathsf{mem} \to \langle \mathsf{mem}, \mathsf{mem} \rangle \mid n \in \Lambda\}
\end{aligned}$$

Let a range over $\mathbb{A}$, and $ar(a)$ denote its arity. To ease the reading in the following we will use the notation $\Delta_a(T, \Sigma)$ to denote the set of measures $\Delta(T_{\langle t_1,\ldots,t_n \rangle}, \bigotimes_{i=1}^n \Sigma_{t_i})$, for $ar(a) = t \to \langle t_1, \ldots, t_n \rangle$.

Let $\mathbb{B}/_\equiv$ be the set of $\equiv$-equivalence classes on $\mathbb{B}$. For $M \in \mathbb{B}$, we denote by $[M]_\equiv$ the $\equiv$-equivalence class of M.

Definition 3.1 (Measurable space of terms). *The* measurable space of terms $(\mathbb{B}, \Pi)$ *is given by the measurable space over* $\mathbb{B}$ *where* Π *is the* σ*-algebra generated by* $\mathbb{B}/_\equiv$.

Notice that $\mathbb{B}/_\equiv$ is a denumerable partition of $\mathbb{B}$, hence it is a base (a generator such that all its elements are disjoint) for Π. Any element of Π can be obtained by a countable union of elements of the base, i.e., for all $\mathcal{M} \in \Pi$ there exist $\{M_i\}_{i \in I}$, for some countable I, such that $\mathcal{M} = \bigcup_{i \in I} [M_i]_\equiv$. As a consequence, in order to generate the whole Π we can simply compute all these unions, without the need of any closure by complement.

A similar argument holds for the product space $(\mathbb{B}_{\langle t_1,\ldots,t_n \rangle}, \bigotimes_{i=1}^n \Pi_{t_i})$, where $t_i \in \{\mathsf{mem}, \mathsf{sys}\}$ $(1 \leq i \leq n)$; indeed $\bigotimes_{i=1}^n \Pi_{t_i}$ can be generated from the base $\mathbb{B}_{\langle t_1,\ldots,t_n \rangle}/_{\equiv_{\langle t_1,\ldots,t_n \rangle}}$, where $\equiv_{\langle t_1,\ldots,t_n \rangle} \subseteq \mathbb{B}_{\langle t_1,\ldots,t_n \rangle} \times \mathbb{B}_{\langle t_1,\ldots,t_n \rangle}$ is defined by

$$\langle M_1, \ldots, M_n \rangle \equiv_{\langle t_1,\ldots,t_n \rangle} \langle N_1, \ldots, N_n \rangle \qquad \text{iff} \qquad M_i \equiv N_i, \text{ for all } 1 \leq i \leq n,$$

which can be easily checked to be an equivalence relation. $\equiv_{\langle t_1,\ldots,t_n \rangle}$-equivalence classes are rectangles, i.e. $[\langle M_1, \ldots, M_n \rangle]_{\equiv_{\langle t_1,\ldots,t_n \rangle}} = [M_1]_\equiv \times \cdots \times [M_n]_\equiv$, therefore the product measure $\bigotimes_{i=1}^n \Pi_{t_i}$ is well defined. For sake of simplicity in the following we write $[\langle M_1, \ldots, M_n \rangle]_\equiv$ in place of $[\langle M_1, \ldots, M_n \rangle]_{\equiv_{\langle t_1,\ldots,t_n \rangle}}$, and $\mathbb{B}_{\langle t_1,\ldots,t_n \rangle}/_\equiv$ in place of $\mathbb{B}_{\langle t_1,\ldots,t_n \rangle}/_{\equiv_{\langle t_1,\ldots,t_n \rangle}}$.

The operational semantics associates with each membrane a family of measures in $\Delta^{\mathbb{A}_{\mathsf{mem}}}(\mathbb{B}, \Pi)$, and with each system a family of measures in $\Delta^{\mathbb{A}_{\mathsf{sys}}}(\mathbb{B}, \Pi)$. This can be represented by two relations

$\to_{\mathsf{mem}}$: $T_{\mathsf{mem}} \to \Delta^{\mathbb{A}_{\mathsf{mem}}}(\mathbb{B},\Pi)$, $\to_{\mathsf{sys}}$: $T_{\mathsf{sys}} \to \Delta^{\mathbb{A}_{\mathsf{sys}}}(\mathbb{B},\Pi)$, defined by the SOS rules listed in Table 2. (In the following, for sake of readability, we will drop the indexes $_{\mathsf{mem},\mathsf{sys}}$). In these rules we use some constants and operations over indexed families of measures, that we define next. For a set (of labels) A, let us denote by $\Delta^A(\mathbb{B},\Pi)$ the set $\prod_{a\in A}\Delta_a(\mathbb{B},\Pi)$ of A-indexed families of measures over $(\mathbb{B},\Pi)$. Given a family of measures $\mu \in \Delta^A(\mathbb{B},\Pi)$ and $a \in A$, the a-component of μ will be denoted as $\mu_a \in \Delta_a(\mathbb{B},\Pi)$.

Null: Let $\omega_{\mathsf{mem}} \in \Delta^{\mathbb{A}_{\mathsf{mem}}}(\mathbb{B},\Pi)$ be the constantly zero measure, i.e., for all $a{\in}\mathbb{A}_{\mathsf{mem}}$ such that $ar(a) = t \to \langle t_1,\ldots,t_n\rangle$ and $\mathcal{M} \in \bigotimes_{i=1}^n \Pi_{t_i}$: $(\omega_{\mathsf{mem}})_a(\mathcal{M}){=}0$.

Prefix: For arbitrary $n \in \Lambda$, $\alpha \in \{\mathbf{J},\mathbf{K},\mathbf{K}^!\}$, and $\beta \in \{\mathbf{J}^!,\mathbf{G}\}$, let the constants $[\alpha_n]_\sigma, [\beta_n]^\tau_\sigma \in \Delta^{\mathbb{A}_{\mathsf{mem}}}(\mathbb{B},\Pi)$ be defined, for arbitrary $X,Y \in \mathbb{B}_{\mathsf{mem}}/_\equiv$, as follows:

$$([\alpha_n]_\sigma)_{\alpha_m}(X) = \begin{cases} \iota(n) & \text{if } n=m \text{ and } \sigma \in X \\ 0 & \text{otherwise} \end{cases}$$
$$([\alpha_n]_\sigma)_{\beta_m}(X\times Y) = 0$$
$$([\beta_n]_\sigma)_{\alpha_m}(X) = 0$$
$$([\beta_n]^\tau_\sigma)_{\beta_m}(X\times Y) = \begin{cases} \iota(n) & \text{if } n=m \text{ and } \sigma\in X,\ \tau\in Y \\ 0 & \text{otherwise} \end{cases}$$

Parallel: For $\mu,\mu' \in \Delta^{\mathbb{A}_{\mathsf{mem}}}(\mathbb{B},\Pi)$, let $\mu_\sigma \mathbin{\mathbb{O}}_\tau \mu' \in \Delta^{\mathbb{A}_{\mathsf{mem}}}(\mathbb{B},\Pi)$ be defined, for $n\in\Lambda$, $\alpha \in \{\mathbf{J},\mathbf{K},\mathbf{K}^!\}$, $\beta \in \{\mathbf{J}^!,\mathbf{G}\}$, and $X,Y \in \mathbb{B}_{\mathsf{mem}}/_\equiv$, as follows (where for all X,τ: $X_{|\tau} \triangleq \bigcup\{[\sigma]_\equiv \mid \sigma|\tau \in X\}$):

$$(\mu_\sigma \mathbin{\mathbb{O}}_\tau \mu')_{\alpha_n}(X) = \mu_{\alpha_n}(X_{|\tau}) + \mu'_{\alpha_n}(X_{|\sigma})$$
$$(\mu_\sigma \mathbin{\mathbb{O}}_\tau \mu')_{\beta_n}(X\times Y) = (\mu)_{\beta_n}(X_{|\tau}\times Y) + (\mu')_{\beta_n}(X_{|\sigma}\times Y)$$

Void: Let $\omega_{\mathsf{sys}} \in \Delta^{\mathbb{A}_{\mathsf{sys}}}(\mathbb{B},\Pi)$ be defined by $(\omega_{\mathsf{sys}})_a(\mathcal{M}) = 0$ for any $a \in \mathbb{A}_{\mathsf{sys}}$, such that $ar(a) = t \to \langle t_1,\ldots,t_n\rangle$, and $\mathcal{M} \in \bigotimes_{i=1}^n \Pi_{t_i}$.

Nesting: For $\nu \in \Delta^{\mathbb{A}_{\mathsf{mem}}}(\mathbb{B},\Pi)$ and $\mu \in \Delta^{\mathbb{A}_{\mathsf{sys}}}(\mathbb{B},\Pi)$, let $\mu @^\sigma_P \nu \in \Delta^{\mathbb{A}_{\mathsf{sys}}}(\mathbb{B},\Pi)$ be defined, for $X,Y \in \mathbb{B}_{\mathsf{mem}}/_\equiv$ and $Z,W \in \mathbb{B}_{\mathsf{sys}}/_\equiv$, as follows:

$$(\mu @^\sigma_P \nu)_{\mathrm{ph}_n}(Z\times W) = \begin{cases} \nu_{\mathbf{J}_n}([\sigma]_\equiv) & \text{if } \sigma\mathbf{h}P\mathbf{i} \in Z \text{ and } \mathbf{k}\in W \\ 0 & \text{otherwise} \end{cases}$$
$$(\mu @^\sigma_P \nu)_{\mathrm{ph}^\perp_n}(X\times Y\times Z\times W) = \begin{cases} \nu_{\mathbf{J}^!_n}(X\times Y) & \text{if } P\in Z \text{ and } \mathbf{k}\in W \\ 0 & \text{otherwise} \end{cases}$$
$$(\mu @^\sigma_P \nu)_{\mathrm{ex}_n}(X\times Z\times W) = \begin{cases} \nu_{\mathbf{K}_n}(X) & \text{if } P\in Z \text{ and } \mathbf{k}\in W \\ 0 & \text{otherwise} \end{cases}$$

$$(\mu @^\sigma_P \nu)_{id}(X) = \mu_{id}(X_{\sigma\mathbf{h}\mathbf{i}}) + \sum_{X'\mathbf{h}X''\mathbf{h}[\mathbf{k}]_\equiv \mathbf{i}\,\mathbb{m}[P]_\equiv \mathbf{i}=X}^{n\in\Lambda} \nu_{\mathbf{G}_n}(X'\times X'') + \sum_{X'|X''\mathbf{h}Y''\,\mathbf{i}\,\mathbb{m}Y'=X}^{n\in\Lambda} \frac{\mu_{\mathrm{ex}_n}(X'\times Y'\times Y'')\cdot \nu_{\mathbf{K}^!_n}(X'')}{\iota(n)}$$

Composition: For $\mu, \mu' \in \Delta^{\mathbb{A}_{\mathsf{sys}}}(\mathbb{B}, \Pi)$, let $\mu_P \otimes_Q \mu' \in \Delta^{\mathbb{A}_{\mathsf{sys}}}(\mathbb{B}, \Pi)$ be defined, for $X, Y \in \mathbb{B}_{\mathsf{mem}}/_{\equiv}$ and $Z, W \in \mathbb{B}_{\mathsf{sys}}/_{\equiv}$, as follows (where for all W, Q, $W_{\mathtt{m}Q} \triangleq \bigcup\{[P]_{\equiv} \mid P\,\mathtt{m}\,Q \in W\}$):

$$(\mu_P \otimes_Q \mu')_{\mathrm{ph}_n}(Z \times W) = \mu_{\mathrm{ph}_n}(Z \times W_{\mathtt{m}Q}) + \mu'_{\mathrm{ph}_n}(Z \times W_{\mathtt{m}P})$$

$$(\mu_P \otimes_Q \mu')_{\mathrm{ph}_n^{\perp}}(X \times Y \times Z \times W) = \mu_{\mathrm{ph}_n^{\perp}}(X \times Y \times Z \times W_{\mathtt{m}Q}) + \mu'_{\mathrm{ph}_n^{\perp}}(X \times Y \times Z \times W_{\mathtt{m}P})$$

$$(\mu_P \otimes_Q \mu')_{\mathrm{ex}_n}(X \times Z \times W) = \mu_{\mathrm{ex}_n}(X \times Z \times W_{\mathtt{m}Q}) + \mu'_{\mathrm{ex}_n}(X \times Z \times W_{\mathtt{m}P})$$

$$(\mu_P \otimes_Q \mu')_{id}(X) = \mu_{id}(X_{\mathtt{m}Q}) + \mu'_{id}(X_{\mathtt{m}P}) + \sum_{X_1 \mathtt{h} X_2 \mathtt{h} Y_1 \,\mathtt{i}\, \mathtt{m} Z_1 \,\mathtt{i}\, \mathtt{m} Y_2 \mathtt{m} Z_2 = X}^{n \in \Lambda} \frac{\mu_{\mathrm{ph}_n}(Y_1 \times Y_2) \cdot \mu'_{\mathrm{ph}_n^{\perp}}(X_1 \times X_2 \times Z_1 \times Z_2)}{\iota(n)} + \sum_{X_1 \mathtt{h} X_2 \mathtt{h} Z_1 \,\mathtt{i}\, \mathtt{m} Y_1 \,\mathtt{i}\, \mathtt{m} Z_2 \mathtt{m} Y_2 = X}^{n \in \Lambda} \frac{\mu_{\mathrm{ph}_n^{\perp}}(X_1 \times X_2 \times Y_1 \times Y_2) \cdot \mu'_{\mathrm{ph}_n}(Z_1 \times Z_2)}{\iota(n)}$$

These operators have nice algebraic properties (e.g., $\mu'_{\sigma} \oplus_{\tau} \mu'' = \mu''_{\tau} \oplus_{\sigma} \mu'$, $(\mu'_{\sigma} \oplus_{\tau} \mu'')_{\sigma|\tau} \oplus_{\rho} \mu''' = \mu'_{\sigma} \oplus_{\tau|\rho} (\mu''_{\tau} \oplus_{\rho} \mu''')$, ...), and respect the structural congruence (e.g., if $P \equiv P'$ and $Q \equiv Q'$ then $\mu'_P \otimes_Q \mu'' = \mu'_{P'} \otimes_{Q'} \mu''$). We refer to [2] for further details about these properties, very useful in calculations.

The next lemmata state that the stochastic transition relation $\rightarrow$ (and hence the operational semantics) is well-defined and consistent, that is, for each process we have exactly one family of measures of its continuations, and this family respects structural congruence.

Lemma 3.2 (Uniqueness). *For $a \in \mathbb{A}$ such that $ar(a) = t \rightarrow \langle t_1, \ldots, t_n \rangle$, and $M \in \mathbb{B}_t$, there exists a unique $\mu \in \Delta^{\mathbb{A}_t}(\mathbb{B}, \Pi)$ such that $M \rightarrow \mu$.*

Lemma 3.3. *If $M \equiv N$ and $M \rightarrow \mu$, then $N \rightarrow \mu$.*

This operational semantics can be used to define the " traditional" pointwise semantics:

$$M \xrightarrow{a,r} \langle M_1, \ldots, M_n \rangle \iff^{\triangle} M \rightarrow \mu \text{ and } \mu_a([\langle M_1, \ldots, M_n \rangle]_{\equiv}) = r$$

and it is conservative with respect to the non-stochastic reduction semantics.

Proposition 3.4. *For all systems $P, Q \in \mathbb{B}_{\mathsf{mem}}$, if $P \rightarrow \mu$ and $\mu_{id}([Q]) > 0$ then $P \;\}\; Q$.*

4 The COW Generic Stochastic Abstract Machine

In this section we present a variant of the generic stochastic abstract machine (GSAM), oriented to systems with nested compartments.

The GSAM has been introduced in [13, 10] for simulating a broad range of process calculi with an arbitrary reaction-based simulation algorithm. Although it does not have a native notion of "compartment", nested systems can represented by "flattening" all compartments and species into a single multiset, where each species is tagged with names representing their position in the hierarchy, as shown in [10]. The idea is to represent each compartment (i.e., each cell) as a different species, keeping track of the relative position in the hierarchy by means of *(node) names*. These names are ranged over by $x, y, z, \ldots$, and are different from names in actions. As an example, a system $\sigma \mathtt{h} \tau \mathtt{h} \mathtt{i}\,\mathtt{i}$ is represented as the multiset $\{\sigma \mathtt{h}^x_y \mathtt{i} \mapsto 1, \tau \mathtt{h}^y_z \mathtt{i} \mapsto 1\}$, where the species $\sigma \mathtt{h}^x_y \mathtt{i}$ should be read "a cell with membrane σ, located in the compartment x and whose compartment is y". Analogously, $\tau \mathtt{h}^y_z \mathtt{i}\}$ is positioned in y and its

compartment is z. Reactions can happen only if the names tagging the involved species match according to the required nesting structure.

However, this approach does not scale well, as the population of species may grow enormously. Let us consider a system composed of n copies of the same cell, e.g., $n \cdot (\sigma \mathbf{h} \mathbf{i})$ (where n can be easily in the order of 10^3–10^4). According to the original GSAM idea, this should be represented in the machine as a single species with multiplicity n, and a single instance of possible reactions whose propensity takes into account of this multiplicity. Instead, the "flat encoding" above yields n different species $\sigma \mathbf{h}^x_{y_1} \mathbf{i}, \ldots, \sigma \mathbf{h}^x_{y_1} \mathbf{i}$, each with multiplicity 1; also the set of reactions explode correspondingly.

For circumventing this problem we introduce a variant of the GSAM with a *copy-on-write* strategy—hence it is called COWGSAM. The idea is to keep a single copy of each species, with its multiplicity; the same applies to reactions. When a reaction has to be applied, the compartments involved are "unfolded", i.e., fresh copies of the compartments are generated and the reaction set is modified accordingly; then, the reaction can be applied. In this way, the hierarchical structure is unfolded only if and when needed.

In order to implement this idea, we have to modify the notion of machine term, reaction and reaction rule. The COWGSAM (with the Next Reaction method) is shown in Figure 1.

First, for generating fresh names, we have to keep track of those already allocated. To this end we introduce *environments*, which are finite sets of names. Then, the machine state is represented by a *machine term* T, i.e. a quadruple $E \vdash (t, S, R)$ where E is an environment; t is the current time; S is a finite function mapping each species I to its population $S(I)$ (if not null); and R maps each reaction O to its activity A, which is used to compute the next reaction. (Notice that the syntax of species I is left unspecified, as it depends on the specific process calculus one has to implement.) We say that a machine term $E \vdash (t, S, R)$ is *well-formed* if for all $x_i, x_j \in E : x_i = x_j \Rightarrow i = j$, and the free names in S, R appear in E. In the following, we assume that machine terms are well formed.

Each reaction is a quadruple (S_1, r, f, S_2), where

- S_1 and S_2 denote the *reactant* population and *product* population respectively;
- r denotes the rate (in s^{-1}) of the reaction, i.e., reactions are assumed to be of the form $S_1 \xrightarrow{r} S_2$;
- f is a function mapping machine terms to machine terms; this functions implements any global update of the machine term after the reaction (if needed).

The *transitions* of the abstract machine are represented by a relation $T \xrightarrow{a,O} T'$ between machine terms, indexed by the propensity a and reaction rules. This should be read as "T goes to T' with rate a, using the rule O". This relation is defined by (Reaction rule) in Figure 1, where the function $next(T)$ selects the next reaction, i.e. it returns a pair (O, t') where $O = (S_1, r, f, S_2)$ is the reaction to happen first among all possible reactions in T, and t' is the new time of the system. Once the reaction has been selected, we have first to create the separate (private) copies of the compartments involved, and to update the reaction set accordingly. This is done by the functions $cow(\_)$ and $duplicate(\_)$, which implement a deep copy-on-write: $cow(E \vdash (t, S, R), S_1)$ is a machine term $E' \vdash (t, S', R')$ representing the same state as $E \vdash (t, S, R)$, but in S' the species indicated in S_1 are unfolded; E' contains all names which have been generated in the process, and R' is the new set of reactions. (Actually, the freshly generated copies represent the instances which are *not* involved by the reaction; this simplifies the reaction application.)

At this point, the reaction is executed, by removing the reactants S_1 from the machine term (via the operation $\ominus$), adding the products S_2 (via the operation $\oplus$) and updating the current time of the machine. The function f performs any global "clean-up" and restructuring to the machine term that may be required by the reaction (e.g., garbage collection, elimination of names not used anymore,...). Moreover, since a reaction can rearrange the hierarchy structure, possibly creating new compartments and deleting others, we have to add to the environment any fresh name introduced in the products.

$$
\begin{aligned}
T &::= E \vdash (t,S,R) && \text{(Machine term)}\\
E &::= x_1,\ldots,x_n && \text{(Environments)}\\
S &::= \{I_1 \mapsto i_1,\ldots,I_N \mapsto i_N\} && \text{(Populations)}\\
R &::= \{O_1 \mapsto A_1,\ldots,O_N \mapsto A_N\} && \text{(Reactions)}\\
O &::= (S_1,r,f,S_2) && \text{(Reaction)}
\end{aligned}
$$

$$
\frac{((S_1,r,f,S_2),a,t') = next(t,S,R) \quad E' \vdash (t,S',R') = cow(E \vdash (t,S,R),S_1)}{E \vdash (t,S,R) \xrightarrow{a,(S_1,r,f,S_2)} norm(E' \cup fn(S_2) \vdash (f(S_2 \oplus ((t',S',R') \ominus S_1))))} \quad \text{(Reaction rule)}
$$

$$
next(t,S,R) \triangleq (O,a,t') \quad \text{if } R(O) = (t',a) \text{ and } t' = \min\{t \mid R(O) = (t,a)\}
$$

$$
cow(E \vdash (t,S,R),\emptyset) \triangleq E \vdash (t,S,R)
$$

$$
\begin{aligned}
cow(E \vdash (t,S,R),\{\rho\mathbf{h}_z^y\,\mathbf{i} \mapsto j\} \cup S_1) \triangleq\ & cow(E \cup fn(S'') \vdash (t,S' \cup S'',R' \cup init(L,(t,S',R)) \cup R''),S_1)\\
& \text{where if } S(\sigma\mathbf{h}_y^x\,\mathbf{i}) > 0 \text{ for some } \sigma\mathbf{h}_y^x\,\mathbf{i}:\\
& \quad S' = S\{\sigma\mathbf{h}_y^x\,\mathbf{i} \mapsto 1, \sigma\mathbf{h}_w^x\,\mathbf{i} \mapsto i-1\}, \text{for } i = S(\sigma\mathbf{h}_y^x\,\mathbf{i}) \text{ and } w \notin E,\\
& \quad L = reactions(\sigma\mathbf{h}_w^x\,\mathbf{i} \mapsto i-1,S'),\\
& \quad R' = \{O \mapsto A \in R \mid O \notin L\},\\
& \quad (S'',R'') = duplicate(E \vdash (t,S,R),y,w)\\
& \text{otherwise}\\
& \quad S' = S\{\rho\mathbf{h}_z^y\,\mathbf{i} \mapsto j, \rho\mathbf{h}_w^y\,\mathbf{i} \mapsto i-j\}, \text{for } i = S(\rho\mathbf{h}_z^y\,\mathbf{i}) \text{ and } w \notin E,\\
& \quad L = reactions(\rho\mathbf{h}_w^y\,\mathbf{i} \mapsto i-j,S'),\\
& \quad (S'',R'') = duplicate(E \vdash (t,S,R),z,w)
\end{aligned}
$$

$$
\begin{aligned}
duplicate(E \vdash (t,S,R),y,y') \triangleq\ & (S' \cup S'',R' \cup init(L,(t,S',R)) \cup R'')\\
& \text{where } S' = S \cup \{\rho\mathbf{h}_{w'}^{y'}\,\mathbf{i} \mapsto i \mid S(\sigma\mathbf{h}_y^z\,\mathbf{i}) > 0, S(\rho\mathbf{h}_w^y\,\mathbf{i}) = i\}, w' \notin E,\\
& \quad L = reactions(\rho\mathbf{h}_{w'}^{y'}\,\mathbf{i} \mapsto i,S'),\\
& \quad R' = \{O \mapsto A \in R \mid O \notin L\},\\
& \quad (S'',R'') = duplicate(E \cup \{w'\} \vdash (t,S,R),w,w')
\end{aligned}
$$

Figure 1: The COW Generic Stochastic Abstract Machine, with the Next Reaction method.

Finally, the term can be "normalized" by collapsing equivalent copies of the same subtree into a single copy (with the right multiplicity), by the function $norm(\_)$. In its simplest form, $norm(\_)$ can be the identity, i.e., no normalization at all. Although this would be correct, it can lead to unnecessary copies of the same subtrees. We could define $norm(\_)$ such that

$$
norm(\{I_1 \mapsto i_1, I_2 \mapsto i_2\} \cup S,R) = norm(\{I_1 \mapsto i_1 + i_2\} \cup S,R[I_2/I_1]) \quad \text{if } I_1 \equiv I_2
$$

where the equivalence between species can be implemented by comparing the subtrees starting from I_1, I_2, e.g., by calculating a suitable hash value. We leave this refinement as future development.

$$\{I_1 \mapsto i_1, \ldots, I_N \mapsto i_N\} \oplus (t,S,R) \triangleq I_1 \mapsto i_1 \oplus \ldots \oplus I_N \mapsto i_N \oplus (t,S,R)$$

$$I \mapsto i \oplus (t,S,R) \triangleq \begin{cases} (t,S',R \cup updates(I,(t,S',R))) & \text{if } S(I) = i' \text{ and } S' = S\{I \mapsto i'+i\} \\ (t,S',R \cup init(L,(t,S',R))) & \text{if } I \notin dom(S), S' = S\{I \mapsto i\} \\ & \text{and } L = reactions(I \mapsto i, S) \end{cases}$$

$$(t,S,R) \ominus \{I_1 \mapsto i_1, \ldots, I_N \mapsto i_N\} \triangleq (t,S,R) \ominus I_1 \mapsto i_1 \ominus \ldots \ominus I_N \mapsto i_N$$

$$(t,S,R) \ominus I \mapsto i \triangleq (t,S',R \cup updates(I,(t,S',R))) \quad \text{if } S(I) = i', i' \geq i \text{ and } S' = S\{I \mapsto i'-i\}$$

$$init(L,(t,S,R)) \triangleq \{O \mapsto (t',a) \mid O \in L \text{ and } a = propensity(O,S) \text{ and } O = (S_1,r,f,S_2) \text{ and } t' = t + delay(r,a)\}$$

$$updates(I,(t,S,R)) \triangleq \{O \mapsto (t',a') \mid R(O) = (t'',a) \text{ and } O = (S_1,r,f,S_2) \text{ and } S_1(I) > 0 \text{ and}$$
$$\text{if } t'' > t \text{ then } a' = propensity(O,S) \text{ and } t' = t + (a/a')(t''-t)$$
$$\text{if } t'' = t \text{ then } a' = propensity(O,S) \text{ and } t' = t + delay(r,a)\}$$

$$propensity((S_1,r,f,S_2),S) \triangleq r \cdot \begin{pmatrix} S^*(I_1) \\ j_1 \end{pmatrix} \cdot \ldots \cdot \begin{pmatrix} S^*(I_n) \\ j_n \end{pmatrix} \text{ if } S_1 = \{I_1 \mapsto j_1, \ldots, I_n \mapsto j_n\}$$

$$S^*(\sigma \mathbf{h}^x_y \mathbf{i}) \triangleq \begin{cases} S(\sigma \mathbf{h}^x_y \mathbf{i}) & \text{if } x = root \\ S(\sigma \mathbf{h}^x_y \mathbf{i}) \cdot S^*(\rho \mathbf{h}^z_x \mathbf{i}) & \text{if } x \neq root \text{ and } S(\rho \mathbf{h}^z_x \mathbf{i}) > 0 \end{cases}$$

Figure 1: The COW Generic Stochastic Abstract Machine (cont.).

In order to implement the Next Reaction method, each reaction O is associated with a pair $R(O) = (a,t)$, where a is the reaction *propensity* and t is the time at which the reaction is supposed to occur. The function $delay(r,a)$ computes a time interval from a random variable with rate r and propensity a.

This general structure can be instantiated with a given process calculus, by providing the definition for the missing parts. Given a set *Proc* of processes, we have to define:

1. the syntax of the species I (which may be different from that of processes);
2. a function $species_{E,x}(P)$ mapping a process $P \in Proc$ to a species set located in x;
3. a function $reactions(I \mapsto i, S)$ for computing the multiset of reactions between a (new) species I with multiplicity i and a population of (existing) species S.

The function *species* is used to initialise the abstract machine at the beginning of a simulation. If we aim to simulate the execution of a process $P \in Proc$, the corresponding initial state (rooted in x) is

$$(\!|P|\!)_x \triangleq fn(J) \vdash J \oplus (0,\emptyset,\emptyset) \quad \text{where } J = species_{\emptyset,x}(P).$$

The *reactions* function is used to adjust the set of possible reactions dynamically.

5 Implementing the Stochastic Brane Calculus in COWGSAM

In this section, we show how the COW Generic Stochastic Abstract Machine can be used to implement the Stochastic Brane Calculus, following the protocol described in the previous section.

5.1 Encoding of the Stochastic Brane Calculus

Syntax of species We can define the species for the brane calculus, which in turn lead us to introduce *complexes* and *actions*.

$$I ::= C\mathbf{h}^x_y\mathbf{i} \qquad \text{(Species)}$$
$$C ::= A_1,\ldots,A_n \qquad \text{(Complexes)}$$
$$A ::= \mathbf{J}_n.\sigma \mid \mathbf{J}^!_n(\tau).\sigma \mid \mathbf{K}_n.\sigma \mid \mathbf{K}^!_n.\sigma \mid \mathbf{G}_n(\tau).\sigma \qquad \text{(Actions)}$$

Notice that (despite the deceiving syntax) species are not systems and complexes are not membranes; nevertheless, actions are a subset of membranes.

Node names can appear in the species in S and in reactions R.

The *species* function We can now provide the definition of the translation of a process $P \in \mathbb{B}_{\mathsf{sys}}$ into a species set. Basically, each compartment is assigned a different, fresh node name; therefore, the function $species_{E,x}(P)$ is parametric in the set E of allocated node names, and the name $x \in E$ to be used as the location of the system P. The name x changes as we descend the compartment hierarchy.

In order to capture correctly the multiplicity of each species, we assume that systems are in *normal form*. Basically, this form is a shorthand for products where n copies of the same system, i.e., $P \circ \cdots \circ P$, are represented as $n \cdot P$.

$$\text{Normal Systems} :: \mathbb{B}^n_{\mathsf{sys}} \qquad Q ::= n_1 \cdot \sigma_1\mathbf{h}Q_1\,\mathbf{i}\,\mathtt{m}\ldots\mathtt{m}\,n_k \cdot \sigma_k\mathbf{h}Q_k\,\mathbf{i}$$
$$\text{where } k \geq 0 \text{ and for } i \neq j : \sigma_i\mathbf{h}Q_i\,\mathbf{i} \not\equiv \sigma_j\mathbf{h}Q_j\,\mathbf{i}$$

A system in normal form can be translated into a system just by unfolding the products. For $Q \in \mathbb{B}^n_{\mathsf{sys}}$ a system in normal form, let $\lceil Q \rceil \in \mathbb{B}_{\mathsf{sys}}$ defined as follows:

$$\lceil \mathbf{k} \rceil = \mathbf{k} \qquad \lceil n \cdot \sigma\mathbf{h}Q'\,\mathbf{i} \circ Q'' \rceil = \overbrace{\sigma\mathbf{h}\lceil Q' \rceil\,\mathbf{i} \circ \cdots \circ \sigma\mathbf{h}\lceil Q' \rceil\,\mathbf{i}}^{n \text{ times}} \circ \lceil Q'' \rceil$$

Proposition 5.1. *For all $P \in \mathbb{B}_{\mathsf{sys}}$, there exists a system in normal form $Q \in \mathbb{B}^n_{\mathsf{sys}}$ such that $\lceil Q \rceil \equiv P$.*

As a consequence, we can give the definition of *species* on systems in normal form, as follows:

$$species_{E,x}(\mathbf{k}) \triangleq \emptyset$$
$$species_{E,x}(n \cdot \sigma\mathbf{h}Q_1\,\mathbf{i} \circ Q_2) \triangleq \{s(\sigma)\mathbf{h}^x_y\mathbf{i} \mapsto n\} \cup species_{E \uplus \{y\},y}(Q_1) \cup species_{E,x}(Q_2)$$
$$\text{with } y \notin E \text{ and } fn(species_{E \uplus \{y\},y}(Q_1)) \cap fn(species_{E,x}(Q_2)) \subseteq \{x\}$$

The condition in the second case ensures that two different compartments are never given the same name—any name clash has to be resolved by an α-conversion. The function $s(\_)$ converts a membrane into a set of complexes:

$$s(\mathbf{0}) \triangleq \emptyset \qquad s(\sigma|\sigma') \triangleq s(\sigma) \cup s(\sigma') \qquad s(\pi.\sigma) \triangleq \{\pi.\sigma\}$$

The *reactions* function The next step is to define the function $reactions(I \mapsto i, S)$, for I a species with multiplicity i and S a population.

$$
\begin{aligned}
reactions_E(I_1 \mapsto i, S) &\triangleq unary_E(I_1) \cup binary_E(I_1, S) \\
unary_E(I_1) &\triangleq \{(\{I_1 \mapsto 1\}, r_n, id, \{(s(\sigma) \cup U_1')\mathbf{h}_y^x\,\mathbf{i} \mapsto 1, s(\rho)\mathbf{h}_w^y\,\mathbf{i} \mapsto 1\}) \mid \\
&\qquad I_1 = U_1\mathbf{h}_y^x\,\mathbf{i},\ U_1 = \{\mathbf{G}_n(\rho).\sigma\} \cup U_1', w \notin E\} \\
binary_E(I_1, S) &\triangleq \big\{(\{I_1 \mapsto 1, I_2 \mapsto 1\}, r_n, f, \{(s(\tau) \cup U_1' \cup s(\sigma) \cup U_2')\mathbf{h}_y^x\,\mathbf{i} \mapsto 1\}) \mid I_2 \in dom(S), \\
&\qquad (I_1 = U_1\mathbf{h}_y^x\,\mathbf{i}, I_2 = U_2\mathbf{h}_w^y\,\mathbf{i}) \vee (I_2 = U_1\mathbf{h}_y^x\,\mathbf{i}, I_1 = U_2\mathbf{h}_w^y\,\mathbf{i}), \\
&\qquad U_1 = \{\mathbf{K}_n^!.\tau\} \cup U_1', U_2 = \{\mathbf{K}_n.\sigma\} \cup U_2', f = \lambda T.T[w := x]\big\} \ \uplus \\
&\quad \big\{(\{I_1 \mapsto 1, I_2 \mapsto 1\}, r_n, id, \\
&\qquad \{(s(\tau) \cup U_1')\mathbf{h}_y^x\,\mathbf{i} \mapsto 1, s(\rho)\mathbf{h}_w^y\,\mathbf{i} \mapsto 1, (s(\sigma) \cup U_2')\mathbf{h}_z^w\,\mathbf{i} \mapsto 1\}) \mid I_2 \in dom(S), \\
&\qquad (I_1 = U_1\mathbf{h}_y^x\,\mathbf{i}, I_2 = U_2\mathbf{h}_z^x\,\mathbf{i}) \vee (I_2 = U_1\mathbf{h}_y^x\,\mathbf{i}, I_1 = U_2\mathbf{h}_z^x\,\mathbf{i}), \\
&\qquad U_1 = \{\mathbf{J}_n^!(\rho).\tau\} \cup U_1', U_2 = \{\mathbf{J}_n.\sigma\} \cup U_2',\ w \notin E\big\}
\end{aligned}
$$

In the case of Brane Calculus, the unary reactions are only those arising from pinocytosis, while binary reactions arise from exocytosis and phagocytosis. In both cases, the multiplicity of each reactant is 1, so the multiplicity of I_1 is not relevant. Exocytosis merges two compartments; this is reflected by the fact that the "rearranging" function f substitutes every occurrence of the name w in T with x. On the other hand, pinocytosis and phagocytosis create new compartments; to represent the new structure, we choose a fresh name w representing the new intermediate nesting level, and reconnect the various compartments accordingly. Therefore, for any reaction $(S_1, r, f, S_2) \in reactions_E(I_1 \mapsto i, S)$, $fn(S_2) \setminus E$ is either $\emptyset$ (in the case of exocytosis) or $\{w\}$.

5.2 Adequacy results

Before proving the correctness of our implementation, we have to define how to translate a species set back to a system of the brane calculus.

Let S be a non empty species set. A *root name* of S, denoted by $root(S)$, is a name x such that $S(C\mathbf{h}_y^x\,\mathbf{i}) > 0$ for some C, y, and for all $z, C' : C'\mathbf{h}_x^z\,\mathbf{i} \notin dom(S)$. The next result states that $root(\_)$ is well defined on the species sets we encounter during a simulation.

Lemma 5.2. *For all $P \in \mathbb{B}_{\mathsf{sys}}$:*

1. *if $P \neq \mathbf{k}$: $root((\!|P|\!)_x) = x$.*
2. *if $(\!|P|\!)_x \xrightarrow{a,O} E \vdash (t, S, R)$ and $S \neq \emptyset$, then $root(S) = x$.*

Proof. (1.) is trivial by definition. (2.) It is enough to check that the reaction rules introduced by $reactions(\_)$ do not change the name of the root, nor introduce new ones. □

We can now define a function $[\![\_]\!]$ which maps complexes to membranes, and a function $[\![\_]\!]_x$ mapping species sets to systems; the latter is parametric in the name x of the root of the system.

$$
[\![A_1, \ldots, A_n]\!] \triangleq A_1 | \ldots | A_n \qquad [\![S]\!]_x \triangleq \prod_{C\mathbf{h}_y^x\,\mathbf{i} \in dom(S)} S(C\mathbf{h}_y^x\,\mathbf{i}) \cdot ([\![C]\!]\mathbf{h}[\![S]\!]_y\,\mathbf{i})
$$

where the notation $n \cdot P$ is a shorthand for $P\,\mathtt{m}\ldots\mathtt{m}\,P$, n times.

Lemma 5.3. *For all $P \in \mathbb{B}_{\mathsf{sys}}$:*

1. $[\![(\!|P|\!)_x]\!]_x \equiv P$.
2. *if* $(\!|P|\!)_x \xrightarrow{a,O} E \vdash (t,S,R)$ *then* $[\![S]\!]_x$ *is well defined.*

Proof. (1.) is easy. (2.) It is enough to check that the reaction rules introduced by $reactions(\_)$ do not introduce loops (i.e., the order among names is well founded). □

We can now state and prove the main results of this section.

Proposition 5.4 (Soundness). *For all $P \in \mathbb{B}_{\mathsf{sys}}$, if $(\!|P|\!)_x \xrightarrow{a,O} E \vdash (t,S,R)$ then there exists μ such that $P \to \mu$ and $\mu_{id}([[\![S]\!]_x]) = a$.*

Proof. The proof is by cases on which reaction rule O is selected by the function *next*. By additivity of measures, we can restrict ourselves to when the whole process P is the redex of the reduction. Let us see here the case when P is the redex of a (red-pino) (another is in Appendix B).

Let $P = \mathbf{G}_n(\rho).\sigma|\sigma_0\mathbf{h}P'\,\mathbf{i}$ and let us assume that σ_0 does not exhibit a $\mathbf{G}_n$ action. Then, the translation of P is $(\!|P|\!)_x = E \vdash species_{\emptyset,x}(P) \oplus (0,\emptyset,\emptyset)$, where

$$\begin{aligned} species_{\emptyset,x}(P) &= \{s(\mathbf{G}_n(\rho).\sigma|\sigma_0)\mathbf{h}^x_y\,\mathbf{i} \mapsto 1\} \cup species_{\{y\},x}(P') \\ &= \{(s(\mathbf{G}_n(\rho).\sigma) \cup s(\sigma_0))\mathbf{h}^x_y\,\mathbf{i} \mapsto 1\} \cup species_{\{y\},x}(P') \\ &= \{(\{\mathbf{G}_n(\rho).\sigma\} \cup s(\sigma_0))\mathbf{h}^x_y\,\mathbf{i} \mapsto 1\} \cup species_{\{y\},x}(P') \end{aligned}$$

Let $I_1 = (\{\mathbf{G}_n(\rho).\sigma\} \cup s(\sigma_0))\mathbf{h}^x_y\,\mathbf{i}$ and $J_{P'} = species_{\{y\},x}(P')$; then $(\!|P|\!)_x = I_1 \mapsto 1 \oplus J_{P'} \oplus (0,\emptyset,\emptyset) = J_{P'} \oplus (0,S',R')$ where

$$\begin{aligned} S' &= \{I_1 \mapsto 1\} \\ R' &= init(L,(0,S',\emptyset)) = \{O_L \mapsto (t_1,a_1)\} \\ O_L &= (\{I_1 \mapsto 1\}, r_n, id, \{(s(\sigma) \cup s(\sigma_0))\mathbf{h}^x_y\,\mathbf{i} \mapsto 1, s(\rho)\mathbf{h}^y_w\,\mathbf{i} \mapsto 1\}) \\ L &= reactions(I_1 \mapsto 1,\emptyset) = unary(I_1) = \{(\{I_1 \mapsto 1\}, r_n, id, \{(s(\sigma) \cup s(\sigma_0))\mathbf{h}^x_y\,\mathbf{i} \mapsto 1, s(\rho)\mathbf{h}^y_w\,\mathbf{i} \mapsto 1\})\} \end{aligned}$$

Now, the reaction O in $(\!|P|\!)_x \xrightarrow{a,O} T$ is O_L (otherwise it would involve P', not the pino of the whole P). This means that $(\!|P|\!)_x \xrightarrow{a,O} T$ is derived by means of an application of the (Reaction rule) as follows, where $S_1 = \{I_1 \mapsto 1\}$ and $S_2 = \{(s(\sigma) \cup s(\sigma_0))\mathbf{h}^x_y\,\mathbf{i} \mapsto 1, s(\rho)\mathbf{h}^y_w\,\mathbf{i} \mapsto 1\}$:

$$\frac{((S_1,r_n,id,S_2),a_1,t_1) = next(0,S',R') \quad (E' \vdash (0,S'',R'')) = cow(E \vdash (0,S',R'),S_1)}{E \vdash \{J_{P'}\} \oplus (0,S',R') \xrightarrow{a_1,(S_1,r_n,id,S_2)} norm(E \cup fn(S_2) \vdash (\{J_{P'}\} \oplus S_2 \oplus ((t_1,S'',R'') \ominus S_1)))}$$

where $S'' = S'$ and $R'' = R'$.

$$\begin{aligned} \{J_{P'}\} \oplus S_2 \oplus ((t_1,S'',R'') \ominus S_1) = \\ &= J_{P'} \oplus \{(s(\sigma) \cup s(\sigma_0))\mathbf{h}^x_y\,\mathbf{i} \mapsto 1, s(\rho)\mathbf{h}^y_w\,\mathbf{i} \mapsto 1\} \oplus ((t_1,S',R') \ominus \{I_1 \mapsto 1\}) \\ &= J_{P'} \oplus \{(s(\sigma) \cup s(\sigma_0))\mathbf{h}^x_y\,\mathbf{i} \mapsto 1, s(\rho)\mathbf{h}^y_w\,\mathbf{i} \mapsto 1\} \oplus (t_1,\{I_1 \mapsto 0\},\{O_L \mapsto (t_2,a_2)\}) \\ &= J_{P'} \oplus J' \oplus (t_1,\{I_1 \mapsto 0\},\{O_L \mapsto (t_2,a_2)\}) \end{aligned}$$

Now let us define Q as $Q = \sigma|\sigma_0\mathbf{h}\rho\mathbf{hk}\,\mathbf{i}\;\mathbf{m}\,P'\,\mathbf{i}$, then $(\!|Q|\!)_x = species_{\emptyset,x}(Q) \oplus (0,\emptyset,\emptyset)$ where

$$\begin{aligned}
species_{\emptyset,x}(Q) &= species_{\emptyset,x}(\sigma|\sigma_0\mathbf{h}\rho\mathbf{hk}\,\mathbf{i}\;\mathbf{m}\,P'\,\mathbf{i}) \\
&= \{s(\sigma|\sigma_0)\mathbf{h}^x_y\,\mathbf{i} \mapsto 1\} \cup species_{\{x\},y}(\rho\mathbf{hk}\,\mathbf{i}\;\mathbf{m}\,P') \\
&= \{(s(\sigma)\cup s(\sigma_0))\mathbf{h}^x_y\,\mathbf{i} \mapsto 1\} \cup species_{\{x\},y}(\rho\mathbf{hk}\,\mathbf{i}) \cup species_{\{x\},y}(P') \\
&= \{(s(\sigma)\cup s(\sigma_0))\mathbf{h}^x_y\,\mathbf{i} \mapsto 1\} \cup \{s(\rho)\mathbf{h}^y_w\,\mathbf{i} \mapsto 1\} \cup \emptyset \cup J_{P'} = J' \cup J_{P'}
\end{aligned}$$

and hence $Q \equiv [\![J_{P'} \oplus J' \oplus (t_1, \{I_1 \mapsto 0\}, \{O_L \mapsto (t_2, a_2)\})]\!]_x$. It remains to prove that $r_n = \mu_{id}([Q])$. Let us notice that the derivation of $P \to \mu$ is actually as follows:

$$\dfrac{\dfrac{\mathbf{G}_n(\rho).\sigma \to [\mathbf{G}_n]^\rho_\sigma \qquad \sigma_0 \to \mu''}{\mathbf{G}_n(\rho).\sigma|\sigma_0 \to [\mathbf{G}_n]^\rho_{\sigma\mathbf{G}_n(\rho).\sigma \uplus \sigma_0}\,\mu''} \qquad P' \to \mu'}{\mathbf{G}_n(\rho).\sigma|\sigma_0\mathbf{h}P'\,\mathbf{i} \to \mu}\;\begin{matrix}(\text{par})\\(\text{loc})\end{matrix}$$

where $\mu = \mu' @^{\mathbf{G}_n(\rho).\sigma|\sigma_0}_{P'}([\mathbf{G}_n]^\rho_{\sigma\mathbf{G}_n(\rho).\sigma\uplus\sigma_0}\,\mu'')$. Then:

$$\begin{aligned}
\mu_{id}([\sigma|\sigma_0\mathbf{h}\rho\mathbf{hk}\,\mathbf{i}\;\mathbf{m}\,P'\,\mathbf{i}]) &= (\mu' @^{\mathbf{G}_n(\rho).\sigma|\sigma_0}_{P'}([\mathbf{G}_n]^\rho_{\sigma\ \mathbf{G}_n(\rho).\sigma\uplus\sigma_0}\,\mu''))_{id}([\sigma|\sigma_0\mathbf{h}\rho\mathbf{hk}\,\mathbf{i}\;\mathbf{m}\,P'\,\mathbf{i}]) \\
&= \mu'_{id}([\sigma|\sigma_0\mathbf{h}\rho\mathbf{hk}\,\mathbf{i}\;\mathbf{m}\,P'\,\mathbf{i}]) + ([\mathbf{G}_n]^\rho_{\sigma\mathbf{G}_n(\rho).\sigma\uplus\sigma_0}\,\mu'')_{\mathbf{G}_n}([\sigma|\sigma_0]\times[\rho]) \\
&= ([\mathbf{G}_n]^\rho_{\sigma\mathbf{G}_n(\rho).\sigma\uplus\sigma_0}\,\mu'')_{\mathbf{G}_n}([\sigma|\sigma_0]\times[\rho]) \\
&= ([\mathbf{G}_n]^\rho_\sigma)_{\mathbf{G}_n}([\sigma|\sigma_0]\times[\rho]) + \mu''_{\mathbf{G}_n}([\sigma|\sigma_0]\times[\rho]) \\
&= r_n + \mu''_{\mathbf{G}_n}([\sigma|\sigma_0]\times[\rho]) = r_n
\end{aligned}$$

where the last equivalence holds because $\mu''_{\mathbf{G}_n}([\sigma|\sigma_0]\times[\rho]) = 0$ because we assumed that the reaction does not involve σ_0. □

Proposition 5.5 (Progress). *For all processes P, Q, if P } Q then there exists a reaction O and a term T such that $(\!|P|\!)_x \xrightarrow{a,O} T$ and $Q \equiv [\![T]\!]_x$.*

Proof. By induction on the derivation of P } Q. Let us see the case of (red-pin), the others being similar. Let $P = \mathbf{G}(\rho).\sigma|\sigma_0\mathbf{h}P'\,\mathbf{i}$ and $Q = \sigma|\sigma_0\mathbf{h}\rho\mathbf{hk}\,\mathbf{i}\;\mathbf{m}\,P'\,\mathbf{i}$. Then,

$$(\!|P|\!)_x = species_{\{y\},x}(P') \oplus (0, \{(\{\mathbf{G}_n(\rho).\sigma\} \cup s(\sigma_0))\mathbf{h}^x_y\,\mathbf{i} \mapsto 1\}, \{O_L \mapsto (t_1, a_1)\})$$

where $O_L = (\{(\{\mathbf{G}_n(\rho).\sigma\} \cup s(\sigma_0))\mathbf{h}^x_y\,\mathbf{i} \mapsto 1\}, r_n, id, \{(s(\sigma)\cup s(\sigma_0))\mathbf{h}^x_y\,\mathbf{i} \mapsto 1, s(\rho)\mathbf{h}^y_w\,\mathbf{i} \mapsto 1\})$. Then, by the (Reaction rule) we can take $T = species_{\{x\},y}(P') \oplus (t_1, \{(s(\sigma)\cup s(\sigma_0))\mathbf{h}^x_y\,\mathbf{i} \mapsto 1, s(\rho)\mathbf{h}^y_w\,\mathbf{i} \mapsto 1\}, \emptyset)$. It is easy to check that $Q \equiv [\![T]\!]_x$. □

Proposition 5.6 (Completeness). *For all processes P, Q, if $P \to \mu$ and $\mu_{id}([Q]) > 0$ then for all node name x, there exists a reaction O and a term T such that $(\!|P|\!)_x \xrightarrow{a,O} T$, $Q \equiv [\![T]\!]_x$ and $a = \mu_{id}([Q])$.*

Proof. If $P \to \mu$ and $\mu_{id}([Q]) > 0$ then P } Q by Prop. 3.4. By Prop. 5.5, we have that for some a, O, T, $(\!|P|\!)_x \xrightarrow{a,O} T$ and $Q \equiv [\![T]\!]_x$. But then $a = \mu_{id}([Q])$ by soundness (Prop. 5.4). □

5.3 Example

We conclude this section with an example. Let $P = 10000 \cdot \mathbf{J}_n.\mathbf{K}_m\mathbf{h}\mathbf{J}_k\mathbf{h}\mathbf{i}\,\mathbf{i}\,\mathbf{m}\,100 \cdot ((\mathbf{J}_n^{\mathsf{I}}(\mathbf{K}_m^{\mathsf{I}}) \mid \mathbf{K}^{\mathsf{I}})\mathbf{h}\mathbf{J}_k\mathbf{h}\mathbf{i}\,\mathbf{i})$. Then, its reductions in the Brane Calculus are as follows:

$$
\begin{aligned}
& 10000 \cdot \mathbf{J}_n.\mathbf{K}_m\mathbf{h}\mathbf{J}_k\mathbf{h}\mathbf{i}\,\mathbf{i}\,\mathbf{m}\,100 \cdot ((\mathbf{J}_n^{\mathsf{I}}(\mathbf{K}_m^{\mathsf{I}}) \mid \mathbf{K}^{\mathsf{I}})\mathbf{h}\mathbf{J}_k\mathbf{h}\mathbf{i}\,\mathbf{i}) \\
\} \;& 9999 \cdot \mathbf{J}_n.\mathbf{K}_m\mathbf{h}\mathbf{J}_k\mathbf{h}\mathbf{i}\,\mathbf{i}\,\mathbf{m}\,99 \cdot ((\mathbf{J}_n^{\mathsf{I}}(\mathbf{K}_m^{\mathsf{I}}) \mid \mathbf{K}^{\mathsf{I}})\mathbf{h}\mathbf{J}_k\mathbf{h}\mathbf{i}\,\mathbf{i})\,\mathbf{m}\mathbf{K}^{\mathsf{I}}\mathbf{h}\mathbf{K}_m^{\mathsf{I}}\mathbf{h}\mathbf{K}_m\mathbf{h}\mathbf{J}_k\mathbf{h}\mathbf{i}\,\mathbf{i}\,\mathbf{i}\,\mathbf{m}\,\mathbf{J}_k\mathbf{h}\mathbf{i}\,\mathbf{i} \\
\} \;& 9999 \cdot \mathbf{J}_n.\mathbf{K}_m\mathbf{h}\mathbf{J}_k\mathbf{h}\mathbf{i}\,\mathbf{i}\,\mathbf{m}\,99 \cdot ((\mathbf{J}_n^{\mathsf{I}}(\mathbf{K}_m^{\mathsf{I}}) \mid \mathbf{K}^{\mathsf{I}})\mathbf{h}\mathbf{J}_k^{\mathsf{I}}()\mathbf{h}\mathbf{i}\,\mathbf{i})\,\mathbf{m}\mathbf{K}^{\mathsf{I}}\mathbf{h}2 \cdot \mathbf{J}_k\mathbf{h}\mathbf{i}\,\mathbf{m}\,\mathbf{k}\mathbf{h}\mathbf{k}\mathbf{i}\,\mathbf{i}
\end{aligned}
$$

The translation of P is $(\!|P|\!)_x = E \vdash species_{\emptyset,x}(P) \oplus (0,\emptyset,\emptyset)$, where

$$
\begin{aligned}
species_{\emptyset,x}(P) &= species_{\{x\}}(10000 \cdot \mathbf{J}_n.\mathbf{K}_m\mathbf{h}\mathbf{J}_k\mathbf{h}\mathbf{i}\,\mathbf{i} \circ 100 \cdot (\mathbf{J}_n^{\mathsf{I}}(\mathbf{K}_m^{\mathsf{I}}) \mid \mathbf{K}^{\mathsf{I}})\mathbf{h}\mathbf{J}_k\mathbf{h}\mathbf{i}\,\mathbf{i}) \\
&= \{s(\mathbf{J}_n.\mathbf{K}_m)\mathbf{h}_y^x\mathbf{i} \mapsto 10000\} \cup species_{\{x,y\},y}(\mathbf{J}_k\mathbf{h}\mathbf{i}) \cup \{s(\mathbf{J}_n^{\mathsf{I}}(\mathbf{K}_m^{\mathsf{I}}) \mid \mathbf{K}^{\mathsf{I}})\mathbf{h}_z^x\mathbf{i} \mapsto 100\} \\
&\quad \cup species_{\{x,z\},z}(\mathbf{J}_k\mathbf{h}\mathbf{i}) \\
&= \{\{\mathbf{J}_n.\mathbf{K}_m\}\mathbf{h}_y^x\mathbf{i} \mapsto 10000, \{\mathbf{J}_n^{\mathsf{I}}(\mathbf{K}_m^{\mathsf{I}}), \mathbf{K}^{\mathsf{I}}\}\mathbf{h}_z^x\mathbf{i} \mapsto 100, \{\mathbf{J}_k\}\mathbf{h}_w^y\mathbf{i} \mapsto 1, \\
&\quad \{\mathbf{J}_k\}\mathbf{h}_v^z\mathbf{i} \mapsto 1\}
\end{aligned}
$$

Let $I_1 = \{\mathbf{J}_n.\mathbf{K}_m\}\mathbf{h}_y^x\mathbf{i}$, $I_2 = \{\mathbf{J}_n^{\mathsf{I}}(\mathbf{K}_m^{\mathsf{I}}), \mathbf{K}^{\mathsf{I}}\}\mathbf{h}_z^x\mathbf{i}$, $I_3 = \{\mathbf{J}_k\}\mathbf{h}_w^y\mathbf{i}$, $I_4 = \{\mathbf{J}_k\}\mathbf{h}_v^z\mathbf{i}$, $r_n = 10s^{-1}$, $r_k = 5s^{-1}$ and $r_m = 5s^{-1}$; then

$$
\begin{aligned}
(\!|P|\!)_x &= I_1 \mapsto 10000 \oplus I_2 \mapsto 100 \oplus I_3 \mapsto 1 \oplus I_4 \mapsto 1 \oplus (0,\emptyset,\emptyset) \\
&= I_2 \mapsto 100 \oplus I_3 \mapsto 1 \oplus I_4 \mapsto 1 \oplus (0,S_1,R_1) \\
&= I_3 \mapsto 1 \oplus I_4 \mapsto 1 \oplus (0,S_2,R_2) \\
&= I_4 \mapsto 1 \oplus (0,S_3,R_3) = (0,S_4,R_4)
\end{aligned}
$$

where

$$
\begin{aligned}
L_1 &= reactions(I_1 \mapsto 10000, \emptyset) = \emptyset \\
S_1 &= S\{I_1 \mapsto 10000\} \\
R_1 &= R \cup init(L_1, (0,S_1,\emptyset)) = \emptyset \\
L_2 &= reactions(I_2 \mapsto 100, S_1) \\
&= (\{I_2 \mapsto 1, I_1 \mapsto 1\}, 10, id, \{\{\mathbf{K}^{\mathsf{I}}\}\mathbf{h}_z^x\mathbf{i} \mapsto 1, \{\mathbf{K}_m^{\mathsf{I}})\}\mathbf{h}_w^z\mathbf{i} \mapsto 1, \{\mathbf{K}_m\}\mathbf{h}_y^w\mathbf{i} \mapsto 1\}) \\
S_2 &= S_1\{I_2 \mapsto 100\} = \{I_1 \mapsto 10000, I_2 \mapsto 100\} \\
R_2 &= R_1 \cup init(L_2, (0,S_2,R_1)) = \{O_{L_2} \mapsto (t_1,a_1)\} \\
L_3 &= reactions(I_3 \mapsto 1, S_2) = \emptyset \\
S_3 &= S_2\{I_3 \mapsto 1\} = \{I_1 \mapsto 10000, I_2 \mapsto 100, I_3 \mapsto 1\} \\
R_3 &= R_2 \cup init(L_3, (0,S_3,R_2)) = R_2 \\
L_4 &= reactions(I_4 \mapsto 1, S_3) = \emptyset \\
S_4 &= S_3\{I_4 \mapsto 1\} = \{I_1 \mapsto 10000, I_2 \mapsto 100, I_3 \mapsto 1, I_4 \mapsto 1\} \\
R_4 &= R_3 \cup init(L_4, (0,S_4,R_3)) = R_3
\end{aligned}
$$

$$\begin{array}{rl}\text{with} \quad O_{L_2} & = (\{I_2 \mapsto 1, I_1 \mapsto 1\}, 10, id, \{\{\mathbf{K}^{\mathsf{I}}\}\mathbf{h}_z^x\,\mathbf{i} \mapsto 1, \{\mathbf{K}_m^{\mathsf{I}}\}\mathbf{h}_w^z\,\mathbf{i} \mapsto 1, \{\mathbf{K}_m\}\mathbf{h}_y^w\,\mathbf{i} \mapsto 1\}) \\ a_1 & = propensity(O_{L_2}, S_2) = 10000000 \\ t_1 & = 0 + delay(10, 10000000).\end{array}$$

Now, the reaction O in $(\!|P|\!)_x \xrightarrow{a,O} T$ is O_L. This means that $(\!|P|\!)_x \xrightarrow{a,O} T$ is derived by means of an application of the (Reaction rule) as follows, where $E = x,y,z,w$, $S_1 = \{I_2 \mapsto 1, I_1 \mapsto 1\}$, and $S_2 = \{\{\mathbf{K}^{\mathsf{I}}\}\mathbf{h}_z^x\,\mathbf{i} \mapsto 1, \{\mathbf{K}_m^{\mathsf{I}}\}\mathbf{h}_w^z\,\mathbf{i} \mapsto 1, \{\mathbf{K}_m\}\mathbf{h}_y^w\,\mathbf{i} \mapsto 1\}$:

$$\frac{((S_1, 10, id, S_2), 10, t_1) = next(0, S_4, R_4) \quad (E' \vdash (0, S_5, R_5)) = cow(E \vdash (0, S_4, R_4), S_1)}{E \vdash (0, S_4, R_4) \xrightarrow{10,(S_1,10,id,S_2)} norm(E' \cup fn(S_2) \vdash (S_2 \oplus ((t_1, S_5, R_5) \ominus S_1)))}$$

where

$$\begin{array}{l}S_5 = \{I_1 \mapsto 1, \{\mathbf{J}_n.\mathbf{K}_m\}\mathbf{h}_{y'}^x\,\mathbf{i} \mapsto 9999, I_2 \mapsto 1, \{\mathbf{J}_n^{\mathsf{I}}(\mathbf{K}_m^{\mathsf{I}}), \mathbf{K}^{\mathsf{I}}\}\mathbf{h}_{z'}^x\,\mathbf{i} \mapsto 99, I_3 \mapsto 1, \\ \qquad I_4 \mapsto 1, \{\mathbf{J}_k\}\mathbf{h}_{w'}^{y'}\,\mathbf{i} \mapsto 1, \{\mathbf{J}_k\}\mathbf{h}_{v'}^{z'}\,\mathbf{i} \mapsto 1\} \\ R_5 = \{O_{L_2} \mapsto (t_1, a_1), O_1 \mapsto (t_2, a_2), O_2 \mapsto (t_3, a_3), O_3 \mapsto (t_4, a_4)\}\end{array}$$

with

$$\begin{array}{l}O_1 = (\{\{\mathbf{J}_n.\mathbf{K}_m\}\mathbf{h}_{y'}^x\,\mathbf{i} \mapsto 1, I_2 \mapsto 1\}, 10, id, \{\{\mathbf{K}^{\mathsf{I}}\}\mathbf{h}_z^x\,\mathbf{i} \mapsto 1, \{\mathbf{K}_m^{\mathsf{I}}\}\mathbf{h}_{w''}^z\,\mathbf{i} \mapsto 1, \{\mathbf{K}_m\}\mathbf{h}_y^{w''}\,\mathbf{i} \mapsto 1\}) \\ O_2 = (\{\mathbf{J}_n^{\mathsf{I}}(\mathbf{K}_m^{\mathsf{I}}), \mathbf{K}^{\mathsf{I}}\}\mathbf{h}_{z'}^x\,\mathbf{i} \mapsto 1, I_1 \mapsto 1\}, 10, id, \{\{\mathbf{K}^{\mathsf{I}}\}\mathbf{h}_{z'}^x\,\mathbf{i} \mapsto 1, \{\mathbf{K}_m^{\mathsf{I}}\}\mathbf{h}_{w'''}^{z'}\,\mathbf{i} \mapsto 1, \{\mathbf{K}_m\}\mathbf{h}_y^{w'''}\,\mathbf{i} \mapsto 1\}) \\ O_3 = (\{\{\mathbf{J}_n.\mathbf{K}_m\}\mathbf{h}_{y'}^x\,\mathbf{i} \mapsto 1, \{\mathbf{J}_n^{\mathsf{I}}(\mathbf{K}_m^{\mathsf{I}}), \mathbf{K}^{\mathsf{I}}\}\mathbf{h}_{z'}^x\,\mathbf{i} \mapsto 1\}, 10, id, \{\{\mathbf{J}^{\mathsf{I}}\}\mathbf{h}_{z'}^x\,\mathbf{i} \mapsto 1, \{\mathbf{K}_m^{\mathsf{I}}\}\mathbf{h}_{z''}^{z'}\,\mathbf{i} \mapsto 1, \{\mathbf{K}_m\}\mathbf{h}_{y'}^{z''}\,\mathbf{i}\}) \\ a_2 = propensity(O_1, S_5) = 99990 \\ a_3 = propensity(O_2, S_5) = 990 \\ a_4 = propensity(O_3, S_5) = 989901 \\ t_2 = 0 + delay(10, 99990) \\ t_3 = 0 + delay(10, 990) \\ t_4 = 0 + delay(10, 989901)\end{array}$$

We can now compute the multiset of the new machine state:

$$\begin{array}{rl}S_2 \oplus ((t_1, S_5, R_5) \ominus S_1) & = S_2 \oplus ((t_1, S_5, R_5) \ominus \{I_2 \mapsto 1, I_1 \mapsto 1\}) \\ & = S_2 \oplus ((t_1, S_6, R_6\}) \ominus \{I_1 \mapsto 1\}) \\ & = \{\{\mathbf{K}^{\mathsf{I}}\}\mathbf{h}_z^x\,\mathbf{i} \mapsto 1, \{\mathbf{K}_m^{\mathsf{I}}\}\mathbf{h}_w^z\,\mathbf{i} \mapsto 1, \{\mathbf{K}_m\}\mathbf{h}_y^w\,\mathbf{i} \mapsto 1\} \oplus (t_1, S_7, R_7) \\ & = (t_1, S_8, R_8)\end{array}$$

with

$$\begin{array}{ll}a_2' = propensity(O_1, S_6) = 0 & S_8 = S_6 \cup \{\{\mathbf{K}^{\mathsf{I}}\}\mathbf{h}_z^x\,\mathbf{i} \mapsto 1, \{\mathbf{K}_m^{\mathsf{I}}\}\mathbf{h}_w^z\,\mathbf{i} \mapsto 1, \{\mathbf{K}_m\}\mathbf{h}_y^w\,\mathbf{i} \mapsto 1\} \\ t_2' = t_1 + (a_2/a_2')(t_2 - t_1) & R_6 = \{O_1 \mapsto (t_2', a_2')\} \\ a_3' = propensity(O_2, S_7) = 0 & R_7 = R_6 \cup \{O_2 \mapsto (t_3', a_3')\} \\ t_3' = t_1 + (a_3/a_5)(t_3 - t_1) & R_8 = R_7 \cup \{\{\{\mathbf{K}_m^{\mathsf{I}}\}\mathbf{h}_w^z\,\mathbf{i} \mapsto 1, \{\mathbf{K}_m\}\mathbf{h}_y^w\,\mathbf{i} \mapsto 1\}, 5, f, \{\emptyset\mathbf{h}_w^z\,\mathbf{i}\} \mapsto 1\} \\ S_6 = S_5 \backslash \{I_2 \mapsto 1\} & \\ S_7 = S_6 \backslash \{I_1 \mapsto 1\} & f = \lambda T.T[y := z]\end{array}$$

6 Conclusions

In this paper, we have presented an abstract machine for the Stochastic Brane Calculus. Instead of defining an *ad hoc* machine, we have adopted the generic abstract machine for stochastic calculi (GSAM) recently introduced by Lakin, Paulevé and Phillips. According to the encoding technique we have adopted, membranes are flattened into a set of species, where the hierarchical structure is represented by means of *names*. In order to keep track of these names, and for dealing efficiently with multiple copies of the same species, we have introduced a new generic abstract machine, called COWGSAM, which extends the GSAM with a *name environment* and a *copy-on-write* optimization strategy. We have proved that the implementation of the Stochastic Brane Calculus in COWGSAM is adequate with respect to the stochastic structural operational semantics of the calculus given in [2].

We think that COWGSAM can be used for implementing other stochastic calculi dealing with nested structures, also beyond the models for systems biology. In particular, it is interesting to apply this approach to Stochastic Bigraphs [9], a general meta-model well-suited for representing a range of stochastic systems with compartments; in this way we would obtain a *General Stochastic Bigraphical Machine*, which could be instantiated to any given stochastic bigraphic reactive system. However, such a machine would not scale well, as in general the COW strategy may be not very useful; thus, we can restrict our attentions to smaller subsets of BRSs, specifically designed to some application domain. For biological applications, the bigraphic reactive systems considered in [1, 7] might be a more reasonable target.

Further work include comparison with other stochastic simulation tools dealing with compartments, like BioPEPA [6].

Acknowledgment Work funded by MIUR PRIN project "SisteR", prot. 20088HXMYN.

References

[1] G. Bacci, D. Grohmann, and M. Miculan. Bigraphical Models for Protein and Membrane Interactions. In G. Ciobanu, editor, *Proc. MeCBIC'09*, volume 11 of *EPTCS*, 2009.

[2] G. Bacci and M. Miculan. Measurable Stochastics for Brane Calculus. *Theoretical Comput. Sci.*, 431:117–136, 2012. doi:10.1016/j.tcs.2011.12.055.

[3] R. Barbuti, A. Maggiolo-Schettini, P. Milazzo, and A. Troina. The Calculus of Looping Sequences for Modelling Biological Membranes. In G. Eleftherakis, P. Kefalas, G. Paun, G. Rozenberg, and A. Salomaa, editors, *Workshop on Membrane Computing*, volume 4860 of *Lecture Notes in Computer Science*, pages 54–76. Springer, 2007.

[4] L. Cardelli. Brane Calculi. In V. Danos and V. Schächter, editors, *Proc. CMSB*, volume 3082 of *Lecture Notes in Computer Science*, pages 257–278. Springer, 2004.

[5] L. Cardelli and R. Mardare. The Measurable Space of Stochastic Processes. In *Proc. QEST*, pages 171–180. IEEE Computer Society, 2010.

[6] F. Ciocchetta and M. L. Guerriero. Modelling Biological Compartments in Bio-PEPA. *Electronic Notes in Theoretical Computer Science*, 227:77–95, 2009.

[7] T. C. Damgaard, E. Højsgaard, and J. Krivine. Formal Cellular Machinery. *Electronic Notes in Theoretical Computer Science*, 284:55–74, 2012.

[8] H. Hermanns. *Interactive Markov Chains: The Quest for Quantified Quality*, volume 2428 of *Lecture Notes in Computer Science*. Springer, 2002.

[9] J. Krivine, R. Milner, and A. Troina. Stochastic Bigraphs. In *Proc. 24th MFPS*, volume 218 of *ENTCS*, pages 73–96, 2008.

[10] M. R. Lakin, L. Paulevé, and A. Phillips. Stochastic Simulation of Multiple Process Calculi for Biology. *Theoretical Comput. Sci.*, 431:181–206, 2012.

[11] C. Laneve and F. Tarissan. A Simple Calculus for Proteins and Cells. *Theor. Comput. Sci.*, 404(1-2):127–141, 2008.

[12] P. Panangaden. *Labelled Markov Processes*. Imperial College Press, London, U.K., 2009.

[13] L. Paulevé, S. Youssef, M. R. Lakin, and A. Phillips. A Generic Abstract Machine for Stochastic Process Calculi. In P. Quaglia, editor, *Proc. CMSB*, pages 43–54. ACM, 2010.

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

A Some measure theory

Given a set M, a family Σ of subsets of M is called a σ-*algebra* if it contains M and is closed under complements and (infinite) countable unions:

1. $M \in \Sigma$;
2. $A \in \Sigma$ implies $A^c \in \Sigma$, where $A^c = M \setminus A$;
3. $\{A_i\}_{i\in\mathbb{N}} \subset \Sigma$ implies $\bigcup_{i\in\mathbb{N}} A_i \in \Sigma$.

Since $M \in \Sigma$ and $M^c = \emptyset$, $\emptyset \in \Sigma$, hence Σ is nonempty by definition. A σ-algebra is closed under countable set-theoretic operations: is closed under finite unions ($A, B \in \Sigma$ implies $A \cup B = A \cup B \cup \emptyset \cup \emptyset \cup \cdots \in \Sigma$), countable intersections (by DeMorgan's law $A \cap B = (A^c \cup B^c)^c$ in its finite and inifite version), and countable subtractions ($A, B \in \Sigma$ implies $A \setminus B = A \cap B^c \in \Sigma$).

Definition A.1 (Measurable Space). *Given a set M and a σ-algebra on M, the tuple (M,Σ) is called a* measurable space*, the elements of Σ* measurable sets*, and M the* support-set.

A set $\Omega \subseteq 2^M$ is a *generator for the σ-algebra* Σ on M if Σ is the closure of Ω under complement and countable union; we write $\sigma(\Omega) = \Sigma$ and say that Σ is generated by Ω. Note that the σ-algebra generated by a Ω is also the smallest σ-algebra containing Ω, that is, the intersection of all σ-algebras that contain Ω. In particular it holds that a completely arbitrary intersection of σ-algebras is a σ-algebra. A σ-algebra generated by Ω, denoted by $\sigma(\Omega)$, is minimal in the sense that if $\Omega \subset \Sigma$ and Σ is a σ-algebra, then $\sigma(\Omega) \subset \Sigma$. If Ω is a σ-algebra then obviously $\sigma(\Omega) = \Omega$; if Ω is empty or $\Omega = \{\emptyset\}$, or $\Omega = \{M\}$, then $\sigma(\Omega) = \{\emptyset, M\}$; if $\Omega \subset \Sigma$ and Σ is a σ-algebra, then $\sigma(\Omega) \subset \Sigma$. A generator Ω for Σ is a *base for* Σ if it has disjoin elements. Note that if Ω is a base for Σ, all measurable sets in Σ can be decomposed into countable unions of elements in Ω.

A *measure* on a measurable space (M,Σ) is a function $\mu : \Sigma \to \mathbb{R}^+_\infty$, where $\mathbb{R}^+_\infty$ denotes the extended positive real line, such that

1. $\mu(\emptyset) = 0$;
2. for any disjoint sequence $\{N_i\}_{i\in I} \subseteq \Sigma$ with $I \subseteq \mathbb{N}$, it holds

$$\mu(\textstyle\bigcup_{i\in I} N_i) = \sum_{i\in I} \mu(N_i)\,.$$

The triple (M,Σ,μ) is called a *measure space*. A measure space (M,Σ,μ) is called *finite* if $\mu(M)$ is a finite real number; it is called σ-*finite* if M can be decomposed into a countable union of measurable sets of finite measure, that is, $M = \bigcup_{i\in I} N_i$, for some $I \subseteq \mathbb{N}$ and $\mu(N_i) \in \mathbb{R}^+$ for each $i \in I$. A set in a measure space has σ-*finite measure* if it is a countable union of sets with finite measure. Specifying a measure includes specifying its domain. If μ is a measure on a measurable space (M,Σ) and Σ' is a σ-algebra

contained in Σ, then the restriction μ' of μ to Σ' is also a measure, and in particular a measure on (M',Σ'), for some $M' \subseteq M$ such that Σ' is a σ-algebra on M'.

Given two measurable spaces and measures on them, one can obtain the product measurable space and the product measure on that space. Let (M_1,Σ_1) and (M_2,Σ_2) be measurable spaces, and μ_1 and μ_2 be measures on these spaces. Denote by $\Sigma_1 \otimes \Sigma_2$ the σ-algebra on the cartesian product $M_1 \times M_2$ generated by subsets of the form $B_1 \times B_2$, said rectangles, where $B_1 \in \Sigma_1$ and $B_2 \in \Sigma_2$. The *product measure* $\mu_1 \otimes \mu_2$ is defined to be the unique measure on the measurable space $(M_1 \times M_2, \Sigma_1 \otimes \Sigma_2)$ such that, for all $B_1 \in \Sigma_1$ and $B_2 \in \Sigma_2$

$$(\mu_1 \otimes \mu_2)(B_1 \times B_2) = \mu_1(B_1) \cdot \mu_2(B_2)$$

The existence of this measure is guaranteed by the Hahn-Kolmogorov theorem. The uniqueness of the product measure is guaranteed only in the case that both (M_1,Σ_1,μ_1) and (M_2,Σ_2,μ_2) are σ-finite.

Let $\Delta(M,\Sigma)$ be the family of measures on (M,Σ). It can be organized as a measurable space by considering the σ-algebra generated by the sets $\{\mu \in \Delta(M,\Sigma) : \mu(S) \geq r\}$, for arbitrary $S \in \Sigma$ and $r > 0$.

Given two measurable spaces (M,Σ) and (N,Θ) a mapping $f\colon M \to N$ is *measurable* if for any $T \in \Theta$, $f^{-1}(T) \in \Sigma$. Measurable functions are closed under composition: given $f\colon M \to N$ and $g\colon N \to O$ measurable functions then $g \circ f\colon M \to O$ is also measurable.

B Proof of Prop. 5.4

Let $P = \mathbf{K}^!_n.\tau|\tau_0 \mathbf{h} \mathbf{K}_n.\sigma|\sigma_0 \mathbf{h} P' \,\mathbf{i}\, \mathbb{m}\, P'' \,\mathbf{i}$ then, the translation of P is $(\!|P|\!)_x = E \vdash species_{\emptyset,x}(P) \oplus (0,\emptyset,\emptyset)$, where

$$\begin{aligned}
&species_{\emptyset,x}(P) = \{s(\mathbf{K}^!_n.\tau|\tau_0)\mathbf{h}^x_y \,\mathbf{i}\} \cup species_{\{x\},y}(\mathbf{K}_n.\sigma|\sigma_0 \mathbf{h} P' \,\mathbf{i}\, \mathbb{m}\, P'') \\
&= \{(\{\mathbf{K}^!_n.\tau\} \cup s(\tau_0))\mathbf{h}^x_y \,\mathbf{i} \mapsto 1\} \cup species_{\{x\},y}(\mathbf{K}_n.\sigma|\sigma_0 \mathbf{h} P' \,\mathbf{i}) \cup species_{\{x\},y}(P'') \\
&= \{(\{\mathbf{K}^!_n.\tau\} \cup s(\tau_0))\mathbf{h}^x_y \,\mathbf{i} \mapsto 1\} \cup \{s(\mathbf{K}_n.\sigma|\sigma_0)\mathbf{h}^y_w \,\mathbf{i} \mapsto 1\} \cup species_{\{x,y\},w}(P') \cup species_{\{x\},y}(P'') \\
&= \{(\{\mathbf{K}^!_n.\tau\} \cup s(\tau_0))\mathbf{h}^x_y \,\mathbf{i} \mapsto 1\} \cup \{(s(\mathbf{K}_n.\sigma) \cup s(\sigma_0))\mathbf{h}^y_w \,\mathbf{i} \mapsto 1\} \cup species_{\{x,y\},w}(P') \cup species_{\{x\},y}(P'') \\
&= \{(\{\mathbf{K}^!_n.\tau\} \cup s(\tau_0))\mathbf{h}^x_y \,\mathbf{i} \mapsto 1\} \cup \{(\{\mathbf{K}_n.\sigma\} \cup s(\sigma_0))\mathbf{h}^y_w \,\mathbf{i} \mapsto 1\} \cup species_{\{x,y\},w}(P') \cup species_{\{x\},y}(P'')
\end{aligned}$$

Let $I_1 = (\{\mathbf{K}^!_n.\tau\} \cup s(\tau_0))\mathbf{h}^x_y \,\mathbf{i}$, $I_2 = (\{\mathbf{K}_n.\sigma\} \cup s(\sigma_0))\mathbf{h}^y_w \,\mathbf{i}$, $J_{P'} = species_{\{x,y\},w}(P')$ and $J_{P''} = species_{\{x\},y}(P'')$; then $(\!|P|\!)_x = I_1 \mapsto 1 \oplus I_2 \mapsto 1 \oplus J_{P'} \oplus J_{P''} \oplus (0,\emptyset,\emptyset) = I_2 \mapsto 1 \oplus J_{P'} \oplus J_{P''} \oplus (0,S',R') = J_{P'} \oplus J_{P''} \oplus (0,S'',R'')$ where

$$\begin{aligned}
L' &= reactions(I_1 \mapsto 1,\emptyset) = \emptyset \\
S' &= S\{I_1 \mapsto 1\} \\
R' &= init(L',(0,S',\emptyset)) = \emptyset \\
L'' &= reactions(I_2 \mapsto 1,S') \\
&= (\{I_2 \mapsto 1, I_1 \mapsto 1\}, r_n, f, \{(s(\tau) \cup s(\tau_0) \cup s(\sigma) \cup s(\sigma_0))\mathbf{h}^x_y \,\mathbf{i} \mapsto 1\}) \\
S'' &= S'\{I_2 \mapsto 1\} = \{I_1 \mapsto 1, I_2 \mapsto 1\} \\
R'' &= init(L'',(0,S'',R')) = \{O_L \mapsto (t_1,a_1)\}
\end{aligned}$$

with $O_L = (\{I_1 \mapsto 1, I_2 \mapsto 1\}, r_n, f, \{(s(\tau) \cup s(\tau_0) \cup s(\sigma) \cup s(\sigma_0))\mathbf{h}^x_y \,\mathbf{i} \mapsto 1\})$. Now, the reaction O in $(\!|P|\!)_x \xrightarrow{F,O} T$ is O_L. This means that $(\!|P|\!)_x \xrightarrow{F,O} T$ is derived by means of an application of the (Reaction rule) as follows, where $S_1 = \{I_1 \mapsto 1, I_2 \mapsto 1\}$, $S_2 = \{(s(\tau) \cup s(\tau_0) \cup s(\sigma) \cup s(\sigma_0))\mathbf{h}^x_y \,\mathbf{i} \mapsto 1\}$ and $f = T[w := x]$:

$$\frac{((S_1,r_n,f,S_2),a_1,t_1)=next(0,S'',R'') \quad (E'\vdash(0,S''',R'''))=cow(E\vdash(0,S'',R''),S_1)}{E\vdash(0,S'',R'')\xrightarrow{a_1,(S_1,r_n,f,S_2)} norm(E'\cup fn(S_2)\vdash(f(S_2\oplus((t_1,S''',R''')\ominus S_1))))}$$

where $S''' = S''$ and $R''' = R''$.

$$\begin{aligned}
&(f(S_2\oplus((t_1,S''',R''')\ominus S_1))) = \\
&\quad = f((s(\tau)\cup s(\tau_0)\cup s(\sigma)\cup s(\sigma_0)\mathbf{h}_y^x\,\mathbf{i})\oplus((t_1,S''',R''')\ominus\{I_1\mapsto 1,I_2\mapsto 1\}) \\
&\quad = f((s(\tau)\cup s(\tau_0)\cup s(\sigma)\cup s(\sigma_0)\mathbf{h}_y^x\,\mathbf{i})\oplus((t_1,\{I_1\mapsto 0,I_2\mapsto 1\},\{O_L\mapsto(t_2,a_2)\})\ominus\{I_2\mapsto 1\}) \\
&\quad = f((s(\tau)\cup s(\tau_0)\cup s(\sigma)\cup s(\sigma_0)\mathbf{h}_y^x\,\mathbf{i})\oplus(t_1,\{I_1\mapsto 0,I_2\mapsto 0\},\{O_L\mapsto(t_3,a_3)\}) \\
&\quad = (s(\tau)\cup s(\tau_0)\cup s(\sigma)\cup s(\sigma_0)\mathbf{h}_y^x\,\mathbf{i})\oplus(t_1,\{I_1\mapsto 0,I_2\mapsto 0\},\{O_L\mapsto(t_3,a_3)\}) \\
&\quad = J'\oplus(t_1,\{I_1\mapsto 0,I_2\mapsto 0\},\{O_L\mapsto(t_3,a_3)\})
\end{aligned}$$

Now let us define Q as $Q=\sigma|\sigma_0|\tau|\tau_0\mathbf{h}P''\,\mathbf{i}\,\mathbb{m}\,P'$, then $(\!|Q|\!)_x = species_{\emptyset,x}(Q)\oplus(0,\emptyset,\emptyset)$ where

$$\begin{aligned}
species_{\emptyset,x}(Q) &= species_{\emptyset,x}(\sigma|\sigma_0|\tau|\tau_0\mathbf{h}P''\,\mathbf{i}\,\mathbb{m}\,P') \\
&= species_{\emptyset,x}(\sigma|\sigma_0|\tau|\tau_0\mathbf{h}P''\,\mathbf{i})\cup species_{\emptyset,x}(P') \\
&= \{s(\sigma|\sigma_0|\tau|\tau_0)\mathbf{h}_y^x\,\mathbf{i})\mapsto 1\}\cup species_{\{x\},y}(P'')\cup species_{\emptyset,x}(P') \\
&= \{s(\tau)\cup s(\tau_0)\cup s(\sigma)\cup s(\sigma_0)\mathbf{h}_y^x\,\mathbf{i}\mapsto 1\}\cup J_{P''}\cup J_{P'} \\
&= J'\cup J_{P''}\cup J_{P'}
\end{aligned}$$

And hence $Q\equiv[\![J_{P'}\oplus J_{P''}\oplus J'\oplus(t_1,\{I_1\mapsto 0,I_2\mapsto 0\},\{O_L\mapsto(t_3,a_3)\})]\!]_x$. It remains to prove that $r_n=\mu_{id}([Q])$. Let us notice that the derivation of $P\to\mu$ is actually as follows:

$$\dfrac{\dfrac{\dfrac{\mathbf{K}_n.\sigma\to[\mathbf{K}_n]_\sigma \qquad \sigma_0\to\mu''}{\mathbf{K}_n.\sigma|\sigma_0\to[\mathbf{K}_n]_{\sigma\mathbf{K}_n.\sigma\oplus_{\sigma_0}\mu''}}\,(\text{par}) \qquad P'\to\mu'}{\mathbf{K}.\sigma|\sigma_0\mathbf{h}P'\,\mathbf{i}\to\mu'@_{P'}^{\mathbf{K}_n.\sigma|\sigma_0}([\mathbf{K}_n]_{\sigma\mathbf{K}_n.\sigma\oplus_{\sigma_0}\mu''})}\,(\text{loc}) \qquad P''\to\mu'''}{\mathbf{K}_n.\sigma|\sigma_0\mathbf{h}P'\,\mathbf{i}\,\mathbb{m}\,P''\to(\mu'@_{P'}^{\mathbf{K}_n.\sigma|\sigma_0}([\mathbf{K}_n]_{\sigma\mathbf{K}_n.\sigma\oplus_{\sigma_0}\mu''}))_{\mathbf{K}_n.\sigma|\sigma_0\mathbf{h}P'\,\mathbf{i}}\otimes_{P''}\mu'''}\,(\text{comp})$$

$$\dfrac{\dfrac{\mathbf{K}_n^!.\tau\to[\mathbf{K}_n^!]_\tau \qquad \tau_0\to\mu}{\mathbf{K}_n^!.\tau|\tau_0\to[\mathbf{K}_n^!]_{\tau\mathbf{K}_n^!.\tau\oplus_{\tau_0}\mu}}\,(\text{par}) \qquad \mathbf{K}_n.\sigma|\sigma_0\mathbf{h}P'\,\mathbf{i}\,\mathbb{m}\,P''\to(\mu'@_{P'}^{\mathbf{K}_n.\sigma|\sigma_0}([\mathbf{K}_n]_{\sigma\mathbf{K}_n.\sigma\oplus_{\sigma_0}\mu''}))_{\mathbf{K}_n.\sigma|\sigma_0\mathbf{h}P'\,\mathbf{i}}\otimes_{P''}\mu'''}{\mathbf{K}_n^!.\tau|\tau_0\mathbf{h}\mathbf{K}_n.\sigma|\sigma_0\mathbf{h}P'\,\mathbf{i}\,\mathbb{m}\,P''\,\mathbf{i}\to\nu}\,(\text{loc})$$

where $\nu=((\mu'@_{P'}^{\mathbf{K}_n.\sigma|\sigma_0}([\mathbf{K}_n]_{\sigma\mathbf{K}_n.\sigma\oplus_{\sigma_0}\mu''}))_{\mathbf{K}_n.\sigma|\sigma_0\mathbf{h}P'\,\mathbf{i}}\otimes_S\mu''')@_{\mathbf{K}_n.\sigma|\sigma_0\mathbf{h}P'\,\mathbf{i}\mathbb{m}P''}^{\mathbf{K}_n^!.\tau|\tau_0}([\mathbf{K}_n^!]_{\tau\mathbf{K}_n^!.\tau\oplus_{\tau_0}\mu})$,
$\mu_1=(\mu'@_{P'}^{\mathbf{K}_n.\sigma|\sigma_0}([\mathbf{K}_n]_{\sigma\mathbf{K}_n.\sigma\oplus_{\sigma_0}\mu''}))_{\mathbf{K}_n.\sigma|\sigma_0\mathbf{h}P'\,\mathbf{i}}\otimes_{P''}\mu'''$ and $\mu_2=[\mathbf{K}_n^!]_{\tau\mathbf{K}_n^!.\tau\oplus_{\tau_0}\mu}$. Then:

$$\begin{aligned}
&\nu_{id}([\sigma|\sigma_0|\tau|\tau_0\mathbf{h}P''\,\mathbf{i}\,\mathbb{m}\,P'])=(\mu_1\ @_{\mathbf{K}_n.\sigma|\sigma_0\mathbf{h}P'\,\mathbf{i}\mathbb{m}P''}^{\mathbf{K}_n^!.\tau|\tau_0}\ \mu_2)_{id}([\sigma|\sigma_0|\tau|\tau_0\mathbf{h}P''\,\mathbf{i}\,\mathbb{m}\,P']) \\
&=\mu_{1\,id}([\sigma|\sigma_0|\tau|\tau_0\mathbf{h}P''\,\mathbf{i}\,\mathbb{m}\,P'])+\mu_{1\,ex_n}([\sigma|\sigma_0]\times[P']\times[P''])\cdot\mu_{2\mathbf{K}_n^!}([\tau|\tau_0])/r_n \\
&=\mu_{1\,ex_n}([\sigma|\sigma_0]\times[P']\times[P''])\cdot\mu_{2\mathbf{K}_n^!}([\tau|\tau_0])/r_n \\
&=((\mu'@_{P'}^{\mathbf{K}_n.\sigma|\sigma_0}([\mathbf{K}_n]_{\sigma\mathbf{K}_n.\sigma\oplus_{\sigma_0}\mu''}))_{ex_n}([\sigma|\sigma_0]\times[P']\times[P''])+\mu'''_{ex_n}([\sigma|\sigma_0]\times[P']\times[P'']))\cdot\mu_{2\mathbf{K}_n^!}([\tau|\tau_0])/r_n \\
&=(([\mathbf{K}_n]_{\sigma\ \mathbf{K}_n.\sigma\oplus_{\sigma_0}\ \mu''})_{\mathbf{K}_n}([\sigma|\sigma_0])+\mu'''_{ex_n}([\sigma|\sigma_0]\times[P']\times[P'']))\cdot([\mathbf{K}_n^!]_{\tau\mathbf{K}_n^!.\tau\oplus_{\tau_0}\mu})_{\mathbf{K}_n^!}([\tau|\tau_0])/r_n \\
&=(([\mathbf{K}_n]_\sigma)_{\mathbf{K}_n}([\sigma|\sigma_0])+\mu''_{\mathbf{K}_n}([\sigma|\sigma_0])+\mu'''_{ex_n}([\sigma|\sigma_0]\times[P']\times[P'']))\cdot((([\mathbf{K}_n^!]_\tau)_{\mathbf{K}_n^!}([\tau|\tau_0])+\mu_{\mathbf{K}_n^!}([\tau|\tau_0]))/r_n \\
&=(r_n+\mu''_{\mathbf{K}_n}([\sigma|\sigma_0])+\mu'''_{ex_n}([\sigma|\sigma_0]\times[P']\times[P'']))\cdot(r_n+\mu_{\mathbf{K}_n^!}([\tau|\tau_0]))/r_n=r_n
\end{aligned}$$

where the last equivalence holds because $\mu''_{\mathbf{K}_n}([\sigma|\sigma_0])=\mu'''_{ex_n}([\sigma|\sigma_0]\times[P']\times[P''])=\mu_{\mathbf{K}_n^!}([\tau|\tau_0])=0$ because we assumed that the reaction does not involve either σ_0 nor τ_0.

Work-in-Progress

Parallel BioScape: A Stochastic and Parallel Language for Mobile and Spatial Interactions

Adriana Compagnoni
Department of Computer Science
Stevens Institute of Technology

Mariangiola Dezani–Ciancaglini
Dipartimento di Informatica
Università di Torino

Paola Giannini
Dipartimento di Informatica
Università del Piemonte Orientale

Karin Sauer
Department of Biological Sciences
Binghamton University

Vishakha Sharma
Department of Computer Science
Stevens Institute of Technology

Angelo Troina
Dipartimento di Informatica
Università di Torino

BioScape is a concurrent language developed for the stochastic simulation of biological processes in a reactive environment in 3D space. It is based on the stochastic π-Calculus process algebra, and it is motivated by the need for individual-based, continuous motion, and continuous space simulation. In this paper we extend BioScape with a fully parallel semantics. Keeping a general point of view, in a fully parallel fashion, a set of movement reductions, a set of discrete interactions and a set of timed reductions are combined to execute a step of the system.

1 Introduction

Process algebras have been successfully used in the modelling of biological systems, see [29, 10, 2], where they are particularly attractive, because of their ability to accommodate new objects and new behavioural attributes as the complex biological system becomes better understood.

However, most of the modelling languages lack adequate support for the design of systems in which to study complex interactions involving both spatial properties, movements in three-dimensional space, and stochastic interactions. Recently, new spatial modelling languages allowing explicit description of temporal spatial dynamics of biochemical processes have been proposed (SpacePi [21], DCA [34], $L\Pi$ [33], Stochsim [24]). Other agent-based platforms [23] include C-Immsim [31, 12] and PathSim visualizer [28]. However, few of them support individual based, continuous motion, and continuous space stochastic simulation [8], which are important features for modelling temporal spatial dynamics of biochemical processes accurately. To address this problem in previous work we introduced BioScape, a language incorporating both stochasticity and 3D spatial attributes.

BioScape is a concurrent language motivated by the biological landscapes found at the interface of biology and biomaterials [15]. It has been motivated by the need to model antibacterial surfaces, biofilm formation, and the effect of DNAse in treating and preventing biofilm infections. To accomplish this modelling a strict interaction between biologists and computer scientists is needed, and therefore, also a language with high level abstractions that may be shared by both. As its predecessor, SPiM [26], a sequential semantics based on Gillespie's algorithm [20] was defined for BioScape, and its implementation does not scale beyond 1000 agents. However, in order to model larger and more realistic systems a semantics that may take advantage of the new multi-core and GPU architectures is needed. This moti-

Pre-Proceedings of 6th Workshop on Membrane Computing and Biologically Inspired Process Calculi
MeCBIC 2012, Pages 105–120.

vates the introduction of parallel semantics, which is the contribution of this paper: Parallel BioScape, an extension with fully parallel semantics.

Gillespie's algorithm produces two outputs in each iteration: 1) the next reaction R to be executed and 2) a slice of time t to advance the simulation clock. Since many reactions, including many instances of the same reaction, may be available, the slice of time t does not correspond to the time that R would take, but an amount of time proportional to the time it would take to execute all available reactions. In contrast, the parallel semantics will execute all available reactions, not just one instance of one reaction R, and the first challenge is then how to calculate simulated time. Reaction times can vary substantially, for example, some prokaryotic cell mitosis takes ten minutes, some plant cell mitosis takes about half an hour, while some animal cell mitosis takes about three hours. So, if we trigger all reactions, how do we advance the simulation clock? The solution we propose here consists of annotating each product of a reaction with a timer indicating how long that reaction will take.

For example, if $Cell \rightarrow_{30} Cell \mid Cell$ means that a $Cell$ takes 30 minutes to split into two daughter cells, then we will annotate the two daughter cells as $\{\{Cell\}\}^{30}$ and $\{\{Cell\}\}^{30}$. As time lapses the timer will be reduced, and when reaching $\{\{Cell\}\}^{0}$, both cells will be available for new reactions.

Furthermore, not all reactions that start might succeed, bacterial cells in the process of splitting might be lysed by an antibacterial agent. To account for those reactions that might fail, we consider their stochastic rate.

1.1 Related Work

In the field of biological modelling, tools such as SPiM [25, 27] and Dizzy [30] have been used to capture first order approximations to system dynamics using a combination of stochastic simulation and differential equation approximation. SPiM has long been the standard tool for simulating stochastic π calculus models. Still in the context of process algebra and biological modelling, we mention Bio-PEPA [13], a timed process algebra designed for the description of biological phenomena and their analysis through quantitative methods such as stochastic simulation and probabilistic model-checking.

As regards fully parallel reductions and stochasticity, Besozzi et al. [6] consider dynamical probabilistic P systems where a universal clock is assumed to exist. At each step, all rules in all regions are simultaneously applied to all objects which can be the subject of an evolution rule. The scenario is more complex than in our case, since elements can move in and out membranes. In that paper, however, space is not explicitly considered. Simulation results comparing a GPU-based parallel algorithm and CPU-based Gillespie's algorithm for BioScape can be found in [22].

In contrast to the now deep and multidimensional understanding of how tissue cells interact with biomaterials surfaces, comparatively little is known about how surface properties influence interactions with bacteria. Spatial information and stochasticity is crucial for the modelling of dynamic processes in a living cell. To this extent in [9] is presented a survey on simulation techniques using Gillespie's stochastic method in a spatial context. The simulation algorithms and tools presented, however, do not permit to express the complex behaviour that is possible to specify through process algebras. This expressiveness is needed in the modelling of the domain we are interested in, that is the interaction between surface properties and bacteria.

Cardelli and Gardner [11] introduce a geometric process algebra, that combines the interaction primitives of the π-calculus with geometric transformations. In their calculus, 3π, the programmer has to explicitly manipulate geometric primitives, while movement in BioScape is a higher level operation that does not expect the programmer to handle collision or calculate distances. Also, as explicitly said in the introduction, no space or timing issues are considered. The spatial calculus of looping sequences [3]

describes in an accurate way the space occupied by elements, without taking time into account. Collisions may arise as results of transitions. This problem is solved by performing a "rearrangement" of the elements in the system.

Interaction between entities spatially located are considered in some multi-agent systems platforms, e.g., ByoDin, see [1]. However, in ByoDin, the behaviour of entities is driven by a set of predefined attributes, and so it cannot be used to model our complex stochastic interaction. The Shape Calculus [4, 5] is the calculus that most closely resembles BioScape: it is a CCS-like timed calculus, and its simulating tool, BioShape, is a multi-agent system. The Shape Calculus is spatial - with a geometric notion of a 3D space - and it is shape-based, i.e. entities have geometric shapes. Shapes have velocities and their movement takes time, while reactions are instantaneous, so modelling different phenomena than those we can represent in our calculus. There are some other relevant differences. BioScape is stochastic, but the Shape Calculus is not. Unlike BioScape, the Shape Calculus does not allow dynamic creation of channels, moreover, agents in BioScape can be modified by reshaping transformations useful in modelling phenomena such as cell growth, but the Shape Calculus only allows movement specified with a velocity vector. The Shape Calculus has a time primitive for describing a delay, while BioScape has a stochastic delay. The specification of an agent in BioScape describes an area where it is allowed to be, but the Shape Calculus does not. This area is instrumental in describing biomaterials such as antibacterial surfaces and preventing bacteria from penetrating the surface while allowing antibacterial molecules to do so. On the other hand, such behaviour would have to be programmed in the description of the agent in the Shape Calculus.

1.2 Summary

In Section 2 we introduce the syntax of the language, its operational semantics is presented in Section 3, where we also discuss the main novelty of the paper which is the introduction of a fully parallel stochastic reduction strategy. An example of modelling the interaction between bacteria, enzymes, and antimicrobial agents is presented along with the results of some simulations and 3D rendering, in Section 4, we draw some conclusions.

2 Syntax

In Fig. 1 we define the syntax of the calculus, which slightly simplifies the syntax of [15] in order to avoid decorating semantic processes with shapes, as defined in Section 3. We assume a set of channel names, denoted by a and b, and a set of variables, denoted by x, y, with subscripts or superscripts, if needed. As usual, $\overline{a}$ is $a_1,\ldots,a_n$, and similar for $\overline{x}$. The empty process is 0. By $X(\overline{u})$ we denote an instance of the entity defined by X. The actual parameters of the instance may be either channel names or variables, in case the instance occurs in a definition. The process $P \mid Q$ is the parallel composition of processes P and Q. By $(\nu a@\mathtt{r},\mathtt{rad}).P$ we define the channel name a with two parameters $\mathtt{r}$ and $\mathtt{rad}\in\mathbb{R}_{\geq 0}$ within process P; the parameter $\mathtt{r}$ is the stochastic rate for communications through channel a and $\mathtt{rad}$ is the communication radius. The radius is the maximum distance between processes in order to communicate through channel a, and the reaction rate determines how long it takes for two processes to react given that they are close enough to communicate.

The *heterogeneous* choice is denoted by M, where $\pi.P\,[+\,M]$ means $\pi.P \mid \pi.P + M$. Choices may have reaction branches and movement branches. The reaction branches are probabilistic (stochastic), since reactions are subject to kinetic reaction rates, while the movement branches are non-deterministic,

$$
\begin{array}{rll}
P,Q ::= & 0 & \text{Empty Process} \\
\mid & X(\overline{u}) & \text{Entity Instance} \\
\mid & P \mid Q & \text{Parallel Composition} \\
\mid & (\nu a@\mathtt{r},\mathtt{rad}).P & \text{Restriction} \\
M ::= & \pi.P\,[+\,M] & \text{Choice of Prefixed Process} \\
\pi ::= & \mathtt{delay@r} & \text{Delay} \\
\mid & !u(v) & \text{Output} \\
\mid & ?u(x) & \text{Input} \\
\mid & \mathtt{mov} & \text{Move} \\
D ::= & \emptyset & \text{Empty} \\
\mid & D, X(\overline{x}) = M^{\xi,\omega,\sigma} \quad \mathtt{FV}(M) \subseteq \overline{x} & \text{Entity Definition} \\
u,v ::= & a \mid b \mid \cdots \mid x \mid y \mid \cdots & \text{Identifier} \\
E ::= & \emptyset & \text{Empty} \\
\mid & E, a@\mathtt{r},\mathtt{rad} & \text{Channel Declaration}
\end{array}
$$

Figure 1: Syntax

since movement of instances of entities is always enabled. The prefix π denotes the action that the process $\pi.P$ can perform. The prefix `delay@r` is a spontaneous and unilateral reaction of a single process, where `r` is the stochastic rate. The prefix $!u$ denotes output and the prefix $?u$ denotes input. The prefix `mov` moves processes in space according to their diffusion rate (ω) (see below). We use standard syntactic abbreviations such as π for $\pi.0$.

We denote by D a global list of definitions. The equality $X(\overline{x}) = M^{\xi,\omega,\sigma}$ defines process X with formal parameters $\overline{x}$, to be the choice M with geometry ξ,ω,σ, specifying a movement space ξ, a step ω, and a shape σ. M describes the behaviour of X with a choice of prefixed processes. The selection of one of the choices depends not only on the available interactions with other processes, but also on the available space. The movement space ξ is a set of point coordinates in the global coordinate system defining a volume. Intuitively, X can move within ξ. The step $\omega \in \mathbb{R}_{\geq 0}$, is the distance that X can stir in a movement, and it corresponds to the diffusion rate of X; σ is the three-dimensional shape (sphere, cube, etc.) of X. The movement space for the empty process 0 is everywhere, the global space, and its movement step is 0 by default. X can be defined at most once in D, and the free variables of P, denoted by $\mathtt{FV}(P)$, must be a subset of the variables $\overline{x}$. We also write $X(\overline{x}) = (\pi.\pi'.P)^{\xi,\omega,\sigma}$ as short for $X(\overline{x}) = (\pi.Y(\overline{x}))^{\xi,\omega,\sigma}$ and $Y(\overline{x}) = (\pi'.P)^{\xi,\omega,\sigma}$.

We use E to range over environments of channel name declarations. By $a@\mathtt{r},\mathtt{rad}$ we declare channel name a with reaction rate `r` and reaction radius `rad`. A channel name a appears at most once in E.

Consider the following simple example of a bacterium `Bac`, that can either move or divide into two daughter cells. `Bac` is defined with movement space `movB`, movement step `stepB`, and shape `shapeB`. Intuitively, bacteria can move within `movB`, with non-deterministic steps of length `stepB`, and the shape `shapeB` is at all times contained within `movB`. The prefix `mov` represents a non-deterministic movement of length `stepB`, whereas `delay@1.0.(Bac() | Bac())` represents mitosis, the division of a

$$\text{S.LOC}\ \frac{P \equiv Q}{\{P\}_\mu \equiv \{Q\}_\mu} \qquad \text{S.LOC.NU}\ \frac{}{(\nu a@\mathtt{r},\mathtt{rad}).\{P\}_\mu \equiv \{(\nu a@\mathtt{r},\mathtt{rad}).P\}_\mu} \qquad \text{S.LOC.PAR}\ \frac{\mu_1(\mathtt{shape}(P)) \cup \mu_2(\mathtt{shape}(Q)) = \mu(\mathtt{shape}(P \mid Q))}{\{P\}_{\mu_1} \mid \{Q\}_{\mu_2} \equiv \{P \mid Q\}_\mu}$$

$$\text{S.NU.COM}\ \frac{}{(\nu a@\mathtt{r},\mathtt{rad}).(\nu b@\mathtt{r}',\mathtt{rad}').A \equiv (\nu b@\mathtt{r}',\mathtt{rad}').(\nu a@\mathtt{r},\mathtt{rad}).A} \qquad \text{S.NU.PAR}\ \frac{a \notin \mathtt{fn}(B)}{((\nu a@\mathtt{r},\mathtt{rad}).A) \mid B \equiv (\nu a@\mathtt{r},\mathtt{rad}).(A \mid B)}$$

Figure 2: Structural Equivalence of Spatial Configurations

bacterium into two daughter cells: `Bac() | Bac()`, and the `delay@1.0` prefix is used to model the fact that division is not an instantaneous reaction.

$$\mathtt{Bac}() = (\mathtt{mov}.\mathtt{Bac}() + \mathtt{delay@1.0}.(\mathtt{Bac}()|\mathtt{Bac}()))^{\mathtt{movB},\mathtt{stepB},\mathtt{shapeB}}$$

3 Operational Semantics

A run-time system is represented by a parallel composition of entity instances (without free variables) each with its shape, and located in some positions of a global frame. We define the shape of processes inductively as follows:

$$\mathtt{shape}(0) = \emptyset \qquad \mathtt{shape}(X(\overline{a})) = \sigma \ \text{ if } X(\overline{x}) = M^{\xi,\omega,\sigma} \in D$$
$$\mathtt{shape}((\nu a@\mathtt{r},\mathtt{rad}).P) = \mathtt{shape}(P) \qquad \mathtt{shape}(P \mid Q) = \mathtt{shape}(P) \Cup \mathtt{shape}(Q)$$

where $\Cup$ gives a shape obtained by composing two shapes trough juxtaposition. For different applications we can choose suitable functions to realise $\Cup$, we only require $\Cup$ to be a commutative and associative operator, i.e. $\sigma_1 \Cup \sigma_2 = \sigma_2 \Cup \sigma_1$ and $(\sigma_1 \Cup \sigma_2) \Cup \sigma_3 = \sigma_1 \Cup (\sigma_2 \Cup \sigma_3)$.

We use μ to denote a map which applied to a shape locates it in the global space, by putting its barycentre at a fixed point, orienting the shape, and possibly modifying it. So $\mu(\mathtt{shape}(P))$ computes the space occupied by a process P in the global coordinate system. Processes may also share channels for communication. *Spatial configurations*, denoted by A, B, ... are defined as follows:

$$A, B ::= \{P\}_\mu \ \mid \ A \mid B \ \mid \ (\nu a@\mathtt{r},\mathtt{rad}).$$

Structural equivalence on configurations is defined in Fig. 2, omitting the rules for associativity and commutativity of $|$ and $+$. Rule S.LOC uses the standard structural equivalence of Pi-calculus processes. We assume that structural equivalence on spatial configurations is also such that parallel composition is commutative, associative, and has neutral element $\{0\}_\mu$ for any μ. The premise of rule S.LOC.PAR assures that the two equivalent processes occupy exactly the same space. In rule S.NU.PAR, `fn` is a function that returns the set of free channel names of a configuration.

The (parallel) operational semantics of BioScape is based on two *auxiliary* reduction relations: a stochastic relation, $E \vdash A \xrightarrow{\mathtt{r}} B$, for reactions such as synchronisation and delay, defined in Fig. 3, and a non-deterministic (non-stochastic) relation, $A \rightarrow B$, for geometric transformations, in our case movements, defined in Fig. 4. Notice that reduction axioms (SR.DELAY, SR.COM, NR.MOVE) only involve entities ($X(\overline{a})$), and entities evolve according to one of the choices in their definition.

Note that the evolution of systems in parallel BioScape follows the parallel reduction rules of Fig. 6, where space and time are checked.

Figure 3 defines the stochastic reduction relation of BioScape, $E \vdash A \xrightarrow{\mathtt{r}} B$, where $\mathtt{r}$ is the rate of the channel used for synchronization or delay. We write $\mathtt{dis}(\mu,\mu')$ for the distance between the origin of μ and the origin of μ'. In rule SR.COM the condition $\mathtt{dis}(\mu,\mu') \leq \mathtt{rad}$ ensures that located processes $\{P\}_\mu$ and $\{Q\}_{\mu'}$ are close enough to communicate through channel a.

The non-stochastic reduction relation of BioScape, $A \rightarrow B$, is defined in Fig. 4. By $\mathtt{translate}(\omega,\mu)$ we denote the function that randomly generates a new map μ', using the movement step ω and the old map μ. The condition $\mu'(\sigma) \subseteq \xi$ of rule NR.MOVE ensures the new located process $\{P[\overline{a}/\overline{x}]\}_{\mu'}$ is within the movement space ξ of X.

For stochastic reductions we compute the duration of the reduction, based on the exponential distribution associated with the propensity of the reduction. Since reductions may have different durations, we introduce *timed configurations*, $\{\{A\}\}^n$, meaning that, after a time n, this configuration will be A. With the metavariables F, and G we denote either spatial configurations or timed configurations, i.e.,

$$F,G ::= A \quad | \quad \{\{A\}\}^n \quad | \quad F \mid G \quad | \quad (\nu a@\mathtt{r},\mathtt{rad}).F \qquad (n \geq 0)$$

We extend structural equivalence to timed configurations in Fig. 5.

We define a reduction strategy that given the whole configuration, first moves all the processes that can be moved, and then executes all the stochastic reductions that can be executed, omitting only reductions which would lead some entities to clash. Both non-stochastic and stochastic reductions are applied in parallel. For this purpose, we define multi-hole contexts C by the following grammar:

$$C ::= F \quad | \quad [\,] \quad | \quad C \mid C \quad | \quad (\nu x@r,\mathtt{rad}).C$$

Congruence on multi-hole contexts is naturally induced by the congruence on configuration, associativity and commutativity of the parallel operator, and standard rules for ν restrictions similar to S.NU.COM and S.NU.PAR. Given this congruence any multi-hole context, C, may be written in a *canonical form*. That is, there is C', $C \equiv C'$ such that

$$C' = \nu_1 \ldots \nu_n.F_1 \mid \cdots \mid F_m \mid [\,] \mid \cdots \mid [\,] \tag{1}$$

where ν_i, $1 \leq i \leq n$, is an abbreviation for $\nu a_i@\mathtt{r}_i,\mathtt{rad}_i$, and for all j, $1 \leq j \leq m$, $F_j = \{\{A\}\}^n$ for some A, and n, or $F_j = \{P\}_\mu$ for some P, and μ. We say that $a_1@\mathtt{r}_1,\mathtt{rad}_1,\ldots,a_1@\mathtt{r}_1,\mathtt{rad}_i$ is $\mathtt{restr}(C)$. In the following we assume that multi-hole contexts are always in canonical form.

$$\text{SR.DELAY}\ \frac{X(\overline{x}) = (delay@\mathtt{r}.P\ [+\ M])^{\xi,\omega,\sigma} \in D}{E \vdash \{X(\overline{a})\}_\mu \xrightarrow{\mathtt{r}} \{P[\overline{a}/\overline{x}]\}_\mu}$$

$$\text{SR.COM}\ \frac{X(\overline{x}) = (!a(b).P\ [+\ M])^{\xi,\omega,\sigma} \in D \quad Y(\overline{y}) = (?a(z).Q\ [+\ N])^{\xi',\omega',\sigma'} \in D \quad \mathtt{dis}(\mu,\mu') \leq \mathtt{rad}}{E, a@\mathtt{r},\mathtt{rad} \vdash \{X(\overline{c})\}_\mu \mid \{Y(\overline{d})\}_{\mu'} \xrightarrow{\mathtt{r}} \{P[\overline{c}/\overline{x}]\}_\mu \mid \{Q[\overline{d}/\overline{y}][b/z]\}_{\mu'}}$$

$$\text{SR.STR}\ \frac{A \equiv A' \qquad E \vdash A' \xrightarrow{\mathtt{r}} B' \qquad B' \equiv B}{E \vdash A \xrightarrow{\mathtt{r}} B}$$

Figure 3: Stochastic Reduction Relation

$$\text{NR.MOVE}\quad \frac{\mu' = \texttt{translate}(\omega,\mu) \quad \mu'(\sigma) \subseteq \xi \quad X(\overline{x}) = (\texttt{mov}.P\,[+\,M])^{\xi,\omega,\sigma} \in D}{\{X(\overline{a})\}_{\mu} \rightarrow \{P[\overline{a}/\overline{x}]\}_{\mu'}} \qquad \text{NR.STR}\quad \frac{A \equiv A' \qquad A' \rightarrow B' \qquad B' \equiv B}{A \rightarrow B}$$

Figure 4: Non-stochastic Reduction Relation

Our reduction strategy avoids spatial clashes. In particular for moving reductions we have to ensure that moves and reshaping are compatible with the available space, that is after moving no entity overlaps another entity. For stochastic reductions we have to assure that the created entities have their space. To this aim we define the space of a configuration, and a predicate that says whether a configuration does not have any overlapping entities.

Let $\texttt{space}(F)$ be a function on configuration F that returns the space occupied by its processes located in the global frame defined as follows.

$$\begin{array}{lll} \texttt{space}(\{P\}_{\mu}) & = \mu(\texttt{shape}(P)) & \qquad \texttt{space}(\{\!\{A\}\!\}^{n}) = \texttt{space}(A) \\ \texttt{space}(F \mid G) & = \texttt{space}(F) \cup \texttt{space}(G) & \qquad \texttt{space}((\nu a@\texttt{r},\texttt{rad}).F) = \texttt{space}(F) \end{array}$$

We say that a configuration F is *OK* if the various entities in F do not overlap, that is:

$$\{P\}_{\mu}\ \text{OK} \qquad A\ \text{OK} \Rightarrow \{\!\{A\}\!\}^{n}\ \text{OK} \qquad F\ \text{OK} \Rightarrow (\nu x@\texttt{r},\texttt{rad}).F\ \text{OK}$$
$$F\ \text{OK} \wedge G\ \text{OK} \wedge \texttt{space}(F) \cap \texttt{space}(G) = \emptyset \Rightarrow F \mid G\ \text{OK}$$

With the notion of OK configuration we define two notions of well-formedness of configurations. The first to be used for parallel move reductions and the second for parallel stochastic reductions. We say that a configuration obtained by a set of parallel moves is OK_{mv} if it is OK and any "extra" movement would produce a clash, i.e. a configuration where some entities occupy the same space. Similarly we say that a configuration obtained by a set of stochastic transformations is OK_{st} if it is OK and any "extra" transformation would produce a clash. In order to formalise this we first need to single out the sets $\mathfrak{R}_{mv}$ and $\mathfrak{R}_{st}$ of movement and stochastic redexes, i.e. we define:

- $\mathfrak{R}_{mv} = \{\{X(\overline{a})\}_{\mu} \mid X(\overline{x}) = (\texttt{mov}.P + M)^{\xi,\omega,\sigma} \in D\}$,
- $\mathfrak{R}_{st} = \{\{X(\overline{a})\}_{\mu} \mid X(\overline{x}) = (delay@\texttt{r}.P + M)^{\xi,\omega,\sigma} \in D\} \cup$
  $\{\{X(\overline{c})\}_{\mu} \mid \{Y(\overline{d})\}_{\mu'} \mid X(\overline{x}) = (!a(b).P + M)^{\xi,\omega,\sigma} \in D \;\&\; Y(\overline{y}) = (?a(z).Q + N)^{\xi',\omega',\sigma'} \in D\}$

It is also handy to extend the syntax of configurations by allowing underlined spatial configurations. We can then define:

Definition 3.1 *(i) An extended configuration F is* OK_{mv} *if F is* OK *and $F \equiv C[A]$ with A not underlined and $A \in \mathfrak{R}_{mv}$ and $A \rightarrow B$ imply $C[B]$ not* OK.

(ii) An extended configuration F is OK_{st} *if F is* OK *and $F \equiv C[A]$ with A not underlined and $A \in \mathfrak{R}_{st}$ and $A \xrightarrow{r} B$ imply $C[B]$ not* OK.

$$\text{S.TI}\quad \frac{A \equiv B}{\{\!\{A\}\!\}^{n} \equiv \{\!\{B\}\!\}^{n}} \qquad \text{S.TI.ZERO}\quad \frac{}{\{\!\{A\}\!\}^{0} \equiv A}$$

Figure 5: Structural Equivalence of Timed Configurations

$$\text{PR.MOVE}\quad \frac{F_i \to G_i \quad (1 \le i \le p) \qquad C[\underline{G_1}]\cdots[\underline{G_p}]\ \mathrm{OK}_{mv}}{C[F_1]\cdots[F_p] \rightarrowtail C[G_1]\cdots[G_p]}$$

$$\text{PR.STOC}\quad \frac{\mathtt{restr}(C) \vdash A_i \xrightarrow{\mathtt{r}_i} B_i \qquad n_i = \tau(\mathtt{r}_i, C_i[A_i]) \quad (1 \le i \le p) \qquad C[C_1[\underline{B_1}]]\cdots[C_p[\underline{B_p}]]\ \mathrm{OK}_{st}}{C[C_1[A_1]]\cdots[C_p[A_p]] \dashrightarrow C[C_1[\{\!\{B_1\}\!\}^{n_1}]]\cdots[C_p[\{\!\{B_p\}\!\}^{n_p}]]}$$

$$\text{PR.TIMED}\quad \frac{n = min\{n_i \mid 1 \le i \le p\} \quad C \text{ is timed}}{C[\{\!\{A_1\}\!\}^{n_1}]\cdots[\{\!\{A_p\}\!\}^{n_p}] \rightsquigarrow C[\{\!\{A_1\}\!\}^{n_1-n}]\cdots[\{\!\{A_p\}\!\}^{n_p-n}]} \qquad \text{PR.CONF}\quad \frac{F \rightarrowtail F_1 \dashrightarrow F_2 \rightsquigarrow F'}{F \to F'}$$

Figure 6: Parallel Reduction Relation

As a last notion, we say that a context C is *timed* if it does not contain timed configurations.

We are now able to explain our parallel reduction strategy, whose rules are given in Fig. 6. The first three rules deal respectively with parallel movements, stochastic reductions and timed reductions, while the fourth rule maps configurations into configurations by applying first the parallel movements, then the stochastic interactions, and finally by advancing the time of the minimum required to complete one or more interaction. In this way at the next iteration there would be new entities to be moved and/or stochastically reduced.

The condition of obtaining an OK_{mv} extended configuration in rule PR.MOVE assures that all possible moves in $C[F_1]\cdots[F_p]$ which do not cause clash have been done in the reduction. Similar effect is produced by the conditions that the extended configuration is OK_{st} and that the context is timed in the following two rules, respectively. Rule PR.STOC prescribes that the time of a stochastic reaction depends (through the function τ) on the rate of the reduction and on the number of available reactants. The context C_i of the redex A_i is a single hole context capturing the surrounding environment t hat influences the speed of the reduction. In the following section we discuss its use.

3.1 Stochastic Fully Parallel Reductions

Combining stochastic reductions in a fully parallel way could give rise to subtle situations in which the evolution of a system may lead towards several different possible routes. In this paper, we defined the calculus in the more general possible way and left all the different possible executions as a system nondeterministic choice. Note that other solutions could be used according to the particular scenario in which the calculus is applied. As an example, all the possible fully parallel reductions could be weighted and normalised, and the one to be executed could be chosen in a probabilistic fashion.

An example. Consider a molecule `e` which can react with molecules `p1` and `p2` resulting, respectively, in the molecules `p1'` and `p2'`. In BioScape we could model such reactions through two communications on channels `ch1` and `ch2`:

```
e() = !ch1() + !ch2()      p1() = ?ch1().p1'()      p2() = ?ch2().p2'()
```

The stochasticity of each reaction is given by the declared channel communication rate, we suppose them to be `r1` (for `ch1`) and `r2` (for `ch2`). In this particular situation, the channel communication rate could be

used to express the kinetic constant of the described chemical reaction. If we take an initial process built as the parallel composition of the processes

$$\texttt{e()} \mid \texttt{e()} \mid \texttt{p1()} \mid \texttt{p1()} \mid \texttt{p2()} \mid \texttt{p2()}$$

three different parallel executions could be expressed.[1] In particular, the following parallel reductions might arise (leading to tree different system evolutions):

- $\{\{\texttt{p1'()}\}\}^{n_1} \mid \{\{\texttt{p1'()}\}\}^{n_2} \mid \texttt{p2()} \mid \texttt{p2()}$
- $\{\{\texttt{p1'()}\}\}^{n} \mid \{\{\texttt{p2'()}\}\}^{m} \mid \texttt{p1()} \mid \texttt{p2()}$
- $\{\{\texttt{p2'()}\}\}^{m_1} \mid \{\{\texttt{p2'()}\}\}^{m_2} \mid \texttt{p1()} \mid \texttt{p1()}$

where n, n_1 and n_2 are computed from an exponential probability distribution with rate `r1` and m, m_1 and m_2 are computed according to rate `r2`.

Stochastic fully parallel reductions and Gillespie's SSA. Consider the simpler case in which a single reaction rule might happen, for example focusing on the two processes

$$\texttt{e()=!ch()} \qquad\qquad \texttt{p()=?ch().p'()}$$

with kinetic constant `r` as the rate for `ch`. In an initial system consisting of the parallel composition of 100 processes of definition `e()` and 1000 processes of definition `p()` (all of them, assumed to be within communication range), the only possible fully parallel reduction (up to the probability distribution return values n_i) is:

- $\{\{\texttt{p'()}\}\}^{n_1} \mid \{\{\texttt{p'()}\}\}^{n_2} \mid \ldots \mid \{\{\texttt{p'()}\}\}^{n_{100}} \mid \underbrace{\texttt{p()} \ldots \texttt{p()}}_{900}$

In the above example, all n_i values are computed initially with rate `r`, in Gillespie's SSA single reactions are applied sequentially with propensities varying from the initial value of $100 * 1000 * \texttt{r}$ (number of `e()` molecules times the number of `p()` molecules times the kinetic constant), to the final value $1 * 901 * \texttt{r}$ (when there is only one `e()` molecule left that might react with 901 different `p()`).

To approximate Gillespie algorithm with the fully parallel approach we should incorporate in the τ function a counting mechanism keeping track of the available reactants in the communication range (in a way similar to what is done, e.g., in [19, 7] or [18, 16, 17]). In order to approximate Gillespie's algorithm when different reactions could take place, the set of fully parallel possible outputs needs to be weighted according to a probability distribution taking into account the propensities of the different reactions applied. The resulting method shall give an approximation of Gillespie's SSA in the lines of the tau–leaping technique [32].

4 Concurrent Modeling of Bacteria–DNase–Antimicrobial Interactions

Infections are now recognised as a leading cause of failure in implanted biomedical devices. They occur when bacteria colonise a device's biomaterial surface, develop into biofilms (Figure 7), and infect the surrounding tissue. Bacteria in the biofilm state are orders of magnitude more resistant to antibiotics than bacteria in the planktonic state. Thus, the current practice of systemic delivery of broad spectrum

[1] For simplicity, we omit here the spatial configurations and assume that all processes are in the range of communication, thus mimicking a well–stirred mixture of elements.

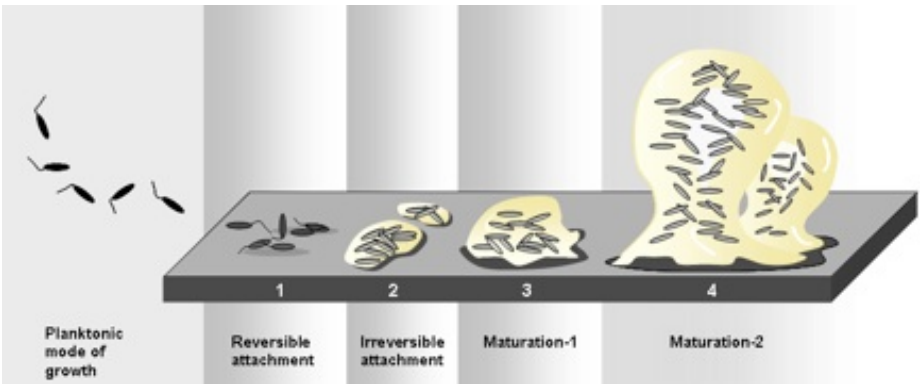


Figure 7: Biofilm developmental stages

antibiotics ultimately exacerbate the problem by selecting for drug-resistant strains rather than curing the infection. One current clinical practice includes localised rather than systemic delivery of antibacterial agents (AmAs) from inserted delivery reservoirs in the implant area. Yet such treatment is not particularly efficient in preventing post-surgery implant infection, because of the high doses required, the toxicity issues, and the eventual depletion of the drug from the reservoir. Therefore standard clinical practice to deal with infected implanted medical devices is to remove the implant altogether and, when possible, perform a revision surgery with a second implant when the infection has cleared. This, however, has tremendous impact on patient well-being and increases the total cost by an order of magnitude or more. The infection-related failure rate ranges from 0.5% to 4% for hip and knee implants, and it can be as high as 40% for fixation devices used in the treatment of traumatic orthopedic injury. The total cost of this problem to the American health-care system is over $6 billion annually. Therefore, engineering biomaterials whose surfaces prevent bacterial attachment or deliver antibacterial agents in targeted and efficient ways is an important yet unsolved problem, from both the fundamental and the clinical perspectives.

Biofilms are complex communities of microorganisms attached to surfaces and embedded in a self–produced extracellular matrix. The extracellular matrix (ECM) can constitute up to 90% of the biofilm biomass and provides a hydrated scaffolding to stabilise and reinforce the biofilm structure. The importance of extracellular genomic DNA (eDNA) as a structural component of biofilms was first demonstrated in *P. aeruginosa*. Whitchurch et al. demonstrated that *P. aeruginosa* PAO1 biofilm formation was attenuated under static growth conditions and significantly reduced under flowing growth conditions by the presence of DNase in the growth medium (Figure 8).

4.1 BioScape Modeling

We now build a prototype model of bacteria–DNase–antimicrobial interactions in solution. The simulation starts with a number of bacteria irreversibly attached to the surface (bacBinit) interacting with a set of antimicrobial agents (AmA) and DNAse enzymes in solution.

Notably the example is written according to both the syntax of [15] and the current one, so that we can compare simulations with both semantics. These simulations will be part of the final version of this paper, in the present version we only put in the appendix results relative to a different example.

```
directive sample 12.0 200
directive plot bacBinit();bacB(); bacDead(); AmA(); bacF(); DNAse(); eDNA(); bioMass()
```

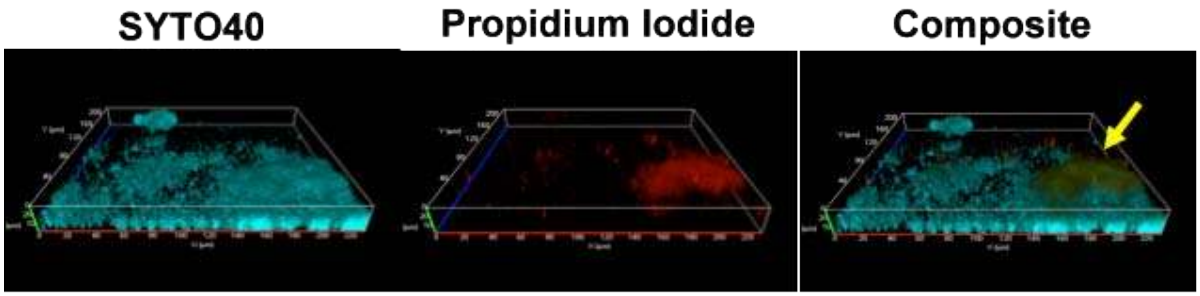


Figure 8: Visualisation of extracellular DNA in *P. aeruginosa* biofilms. The arrow indicates propidium iodide (PI) stainable eDNA. Intracellular DNA was stained with SYTO40. The composite image depicts the overlay of PI and SYTO40 stained confocal image.

```
val top_right = <1.0, 2.0, 3.0>
val bacterialGrowthLayer:space = cuboid(100.0, 100.0, 100.0) @ top_right
val initialBacBLayer:space = cuboid(100.0,100.0,100.0)@ top_right

val step = 5.0
val radius = 1000.0
val radius1 = 1000.0
val radius2 = 1000.0

(* ------------------------------------------------------------------------ *)
(* Channels      *)
(* ------------------------------------------------------------------------ *)

new kill@4.0, radius:chan()          (* comm. between AmA and bacDisp      *)
new lysis@3.0, radius1:chan()          (* comm. between DNAse and eDNA     *)
new disperse@2.0, radius2:chan()       (* comm. between eDNA and bacB      *)

(* ------------------------------------------------------------------------ *)
(* Species      *)
(* ------------------------------------------------------------------------ *)

let bacBinit()@initialBacBLayer, 0 , sphere(1.0) = delay@0.5; bacB()

and bacB()@bacterialGrowthLayer , step , sphere(1.0) =
    do delay@0.5; (bacB()|bacB())
    or delay@0.5; bioMass()
    or delay@0.5; eDNA()
    or ?disperse; bacDisp()

and bioMass()@bacterialGrowthLayer , step , sphere(1.0) = delay@0.5; bioMass()

and eDNA()@bacterialGrowthLayer , step , sphere(1.0) = ?lysis;()

and bacDisp()@bacterialGrowthLayer , step , sphere(1.0)  =
     do delay@4.0; (bacDisp()|bacDisp())
```

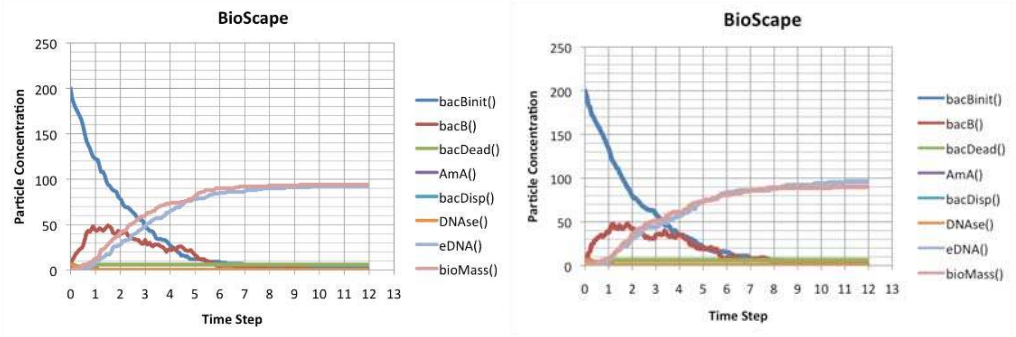


Figure 9: BioScape simulation results for two different executions with identical initial concentrations of the model of bacteria–DNase–antimicrobial interactions in solution

```
     or ?kill; bacDead()
     or mov; bacDisp()

and bacDead()@bacterialGrowthLayer , step ,sphere(1.0) =  delay@1.0;bacDead()

and DNAse()@bacterialGrowthLayer , step , sphere(1.0) = do !lysis;!disperse;()
                                                         or mov; DNAse()

and AmA()@bacterialGrowthLayer , step , sphere(1.0) = do !kill;()
                                                       or mov; AmA()
```

After a few interactions, several runs of the calculus give rise to clusters containing extracellular DNA (eDNA), biomass, and bound bacteria (bacB) representing biofilms.

In this prototype model we assume the same diffusion rate for all species (step), and we use reaction rates and radii that have not yet been validated with wet lab experimental data. We start with bacteria bound to the substrate (bacBinit) that become bound bacteria (bacB). The need for the different species arises because while bound bacteria can grow forming layers beyond the surface (bacterialGrowthLayer), the initial conditions only account for bound bacteria on the substrate (initialBacBLayer), a smaller area than that where a biofilm can develop.

Bound bacteria can secrete biomass and eDNA as they metabolize and decay, and they can be dispersed by DNAse. Biomass is part of the biofilm. Dispersed bacteria (bacDisp) can multiply, similar to bacB, can be killed by AmA, becoming bacDead, and they can also move. DNAse lyses eDNA producing a gap in the extra cellular matrix (ECM) and consequently dispersing bacteria, and they can also move.

```
DNAse()@bacterialGrowthLayer , step , sphere(1.0) = do !lysis; !disperse;()
                                                     or mov; DNAse()
```

is a shorthand for

```
DNAse()@bacterialGrowthLayer , step , sphere(1.0) = do !lysis; aux()
                                                     or mov; DNAse()
 and  aux()@bacterialGrowthLayer , step , sphere(1.0) =  !disperse;()
```

Simulation results for this model can be seen in Figures 9 and 10.

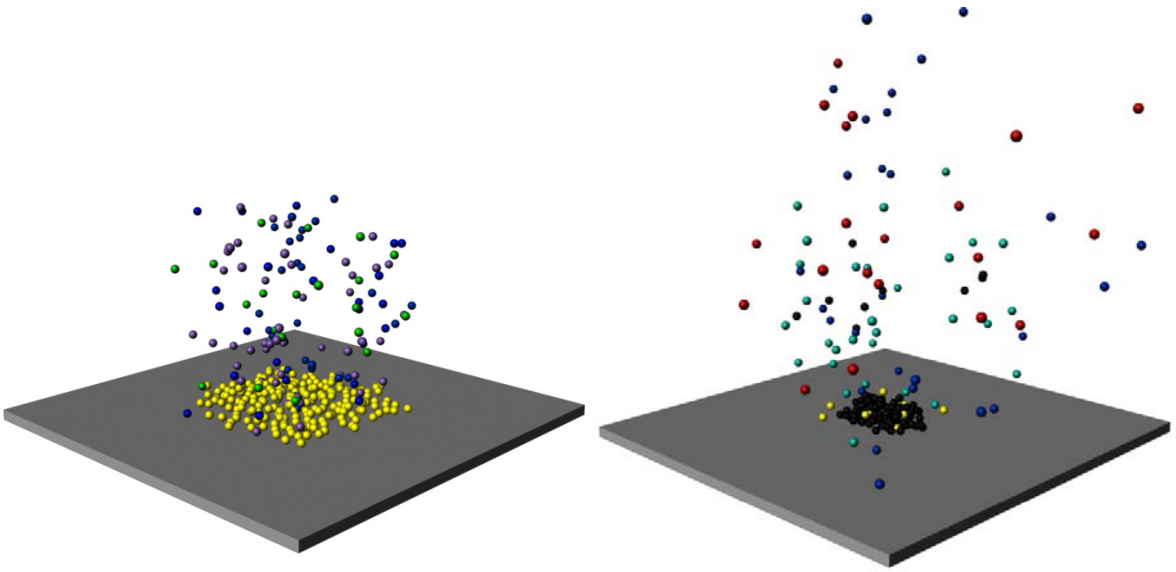


Figure 10: Simulation results rendering using Autodesk Maya animation tool. Color Key: Bacteria bacDisp() green; Drug Molecule AmA() dark blue; DNAse DNAse() purple; eDNA eDNA() turquoise/light blue; BioMass bioMass() Black; Dispersing Bacteria bacB() yellow; Killed Bacteria bacDead() red

5 Conclusions

In this paper, we have presented a fully parallel extension of the BioScape process algebra. Also, we have added some spatial well–formedness conditions to avoid reductions that could give rise to spatial overlapping between the shapes of different processes. The calculus now offers a wide variety of features, ranging from geometrical descriptions to stochastic evolution and fully parallel reductions. BioScape has been extensively used in recent years to model antibacterial surfaces, biofilm formation, and the effect of DNAse in treating and preventing biofilm infections [15, 2, 14].

While BioScape processes are free to move (when they do not clash or overlap) the shape of a BioScape process is constant during the evolution. A possible extension of the calculus consists of a semantics enriched with features to deal with dynamic shapes, thus allowing a processes to change its shape in an explicit way. An alternative way of achieving this in some cases is using affine transformations in rule NR.MOVE.

Also, note that processes within a timed configuration are not allowed to interact until the time of their preceding reduction has elapsed but it might be the case that other reactions could occur during the given time window. An extension of the model could consider a wider set of interactions allowing some kind of reactivity for timed configurations.

While Gillespie's algorithm sequentializes an inherently parallel system, and therefore requires a mathematical model to justify the relation between chemical reactions in a homogeneous volume and a sequential algorithm, our semantics closely captures the world we attempt to model. While in the real world movement and reaction happen simultaneously, our implementation splits the process into two phases: motion and reaction. Entities move in an instantaneous way. The reaction step creates timed entities that will become available when the time it takes to complete the reaction finishes, exactly as

it happens in real phenomena. A simulation clock ticks in the same way reaction time lapses in a the real world. Therefore, we believe that a comparison with Gillespies algorithm could be an interesting intellectual exercise, but, justifying an algorithm that closely matches the real world, may result in an unnecessarily complicated artefact.

Acknowledgment We are grateful to Emily Routenberg for the rendering in Fig. 10. This research is founded by the BioBITs Project (*Converging Technologies* 2007, area: Biotechnology–ICT), Regione Piemonte.

References

[1] P. Ballet. SimBioDyn: Multiagent System for Dynamic Biological Processes. `http://pagesperso.univ-brest.fr/~ballet/pages/8.html`, 2012.

[2] Y. Bao, A. B. Compagnoni, J. Glavy, and T. White. Computational Modeling for the Activation Cycle of G-proteins by G-protein-coupled Receptors. In *Proceedings Fourth Workshop on Membrane Computing and Biologically Inspired Process Calculi 2010. (MeCBIC)*, number 40 in Electronic Proceedings in Theoretical Computer Science, pages 39–53, 2010.

[3] R. Barbuti, A. Maggiolo-Schettini, P. Milazzo, and G. Pardini. Spatial Calculus of Looping Sequences. *Theoretical Computer Science*, 412(43):5976–6001, 2011.

[4] E. Bartocci, D. R. Cacciagrano, M. R. D. Berardini, E. Merelli, and L. Tesei. Timed Operational Semantics and Well-formedness of Shape Calculus. *Sci. Ann. Comp. Sci.*, 20:32–52, 2010.

[5] E. Bartocci, F. Corradini, M. R. D. Berardini, E. Merelli, and L. Tesei. Shape Calculus. A Spatial Mobile Calculus for 3d Shapes. *Sci. Ann. Comp. Sci.*, 20:1–31, 2010.

[6] D. Besozzi, P. Cazzaniga, D. Pescini, and G. Mauri. A Multivolume Approach to Stochastic Modelling with Membrane Systems. *Algorithmic Bioprocesses*, pages 519–542, 2009.

[7] L. Bioglio, M. Dezani-Ciancaglini, P. Giannini, and A. Troina. Typed Stochastic Semantics for the Calculus of Looping Sequences. *Theoretical Computer Science*, 431:165–180, 2012.

[8] A. T. Bittig and A. M. Uhrmacher. Spatial Modeling in Cell Biology at Multiple Levels. In *Winter Simulation Conference*, pages 608–619, 2010.

[9] K. Burrage, P. Burrage, A. Leier, T. Marquez-Lago, and J. Nicolau, DanV. Stochastic Simulation for Spatial Modelling of Dynamic Processes in a Living Cell. In H. Koeppl, G. Setti, M. di Bernardo, and D. Densmore, editors, *Design and Analysis of Biomolecular Circuits*, pages 43–62. Springer New York, 2011.

[10] L. Cardelli, E. Caron, P. Gardner, O. Kahramanogullari, and A. Phillips. A Process Model of rho gtp-binding Proteins. *Theor. Comput. Sci.*, 410(33-34):3166–3185, 2009.

[11] L. Cardelli and P. Gardner. Processes in Space. *Theoretical Computer Science*, 431(0):40–55, 2012.

[12] F. Castiglione and M. Bernaschi. C-immsim: Playing with the Immune Response. In *Proceedings of the Sixteenth International Symposium on Mathematical Theory of Networks and Systems (MTNS2004)*, 2004.

[13] F. Ciocchetta and J. Hillston. Bio-PEPA: An Extension of the Process Algebra PEPA for Biochemical Networks. In *Proc. of 1st Workshop "From Biology To Concurrency and back (FBTC), Lisbon, Portugal*, volume 194 of *Electr. Notes Theor. Comput. Sci.*, pages 103–117. Elsevier, 2008.

[14] A. Compagnoni. Bioscape website. `http://www.cs.stevens.edu/~abc/ComputationalSystemsBiology/Computational_Systems_Biology.html`.

[15] A. Compagnoni, V. Sharma, Y. Bao, P. Bidinger, L. Bioglio, E. Bonelli, M. Libera, and S. Sukhishvili. Bioscape: A Modeling and Simulation Language for Bacteria-materials Interactions. In P. Giannini and E. de Vink, editors, *CS2Bio'12. 3rd International Workshop on Interactions between Computer Science and Biology 16th of June 2012- Stockholm, Sweden*, ENTCS, 2012. To appear.

[16] M. Coppo, F. Damiani, M. Drocco, E. Grassi, E. Sciacca, S. Spinella, and A. Troina. Hybrid Calculus of Wrapped Compartments. In *Proc. of 4th Workshop on Membrane Computing and Biologically Inspired Process Calculi (MeCBIC), Jena, Germany*, volume 40, pages 102–120. EPTCS, 2010.

[17] M. Coppo, F. Damiani, M. Drocco, E. Grassi, E. Sciacca, S. Spinella, and A. Troina. Simulation Techniques for the Calculus of Wrapped Compartments. *Theoretical Computer Science*, 431:75–95, 2012.

[18] M. Coppo, F. Damiani, M. Drocco, E. Grassi, and A. Troina. Stochastic Calculus of Wrapped Compartments. In *Proceedings Eighth Workshop on Quantitative Aspects of Programming Languages (QAPL), Paphos, Cyprus*, volume 28, pages 82–98. EPTCS, 2010.

[19] M. Dezani-Ciancaglini, P. Giannini, and A. Troina. A Type System for a Stochastic CLS. In *Proc. of 4th Workshop on Membrane Computing and Biologically Inspired Process Calculi (MeCBIC), Bologna, Italy*, volume 11, pages 91–105. EPTCS, 2009.

[20] D. T. Gillespie. Exact Stochastic Simulation of Coupled Chemical Reactions. *The Journal of Physical Chemistry*, 81(25):2340–2361, Dec. 1977.

[21] M. John, R. Ewald, and A. M. Uhrmacher. A Spatial Extension to the Pi Calculus. *Electron. Notes Theor. Comput. Sci.*, 194:133–148, 2008.

[22] J. Li, V. Sharma, N. Ganesan, and A. Compagnoni. Simulation and Study of Large-scale Bacteria-materials Interactions Via Bioscape Enabled Gpus. In *Proceedings of ACM-BCB 2012*, 2012. To appear.

[23] C. Macal and M. North. Tutorial on Agent-based Modelling and Simulation. *Journal of Simulation*, 4(3):151–162, 2010.

[24] N. L. Novère and T. S. Shimuzu. Stochsim: Modelling of Stochastic Biomolecular Processes. *BioInformatics*, 17(6):575–576, 2001.

[25] A. Phillips and L. Cardelli. A Correct Abstract Machine for the Stochastic Pi-calculus. In *Proc. of BIOCONCUR, London, England*, Electr. Notes Theor. Comput. Sci., 2004.

[26] A. Phillips and L. Cardelli. Efficient, Correct Simulation of Biological Processes in the Stochastic Pi-calculus. In *Computational Methods in Systems Biology*, volume 4695 of *LNCS*, pages 184–199. Springer, September 2007.

[27] A. Phillips and L. Cardelli. Efficient, Correct Simulation of Biological Processes in the Stochastic Pi-calculus. In *In Proc. of Intl. Conference on Computational Methods in Systems Biology (CMSB), Edinburgh, Scotland*, volume 4695 of *LNCS*, pages 184–199. Springer, 2007.

[28] N. F. Polys, D. A. Bowman, C. North, R. Laubenbacher, and K. Duca. Pathsim Visualizer: An Information-rich Virtual Environment Framework for Systems Biology. In *Web3D 04: Proceedings of the ninth international conference on 3D Web technology*, pages 7–14, New York, NY, USA, 2004. ACM, ACM.

[29] C. Priami, A. Regev, E. Y. Shapiro, and W. Silverman. Application of a Stochastic Name-passing Calculus to Representation and Simulation of Molecular Processes. *Inf. Process. Lett.*, 80(1):25–31, 2001.

[30] S. Ramsey, D. Orrell, and H. Bolouri. Dizzy: Stochastic Simulation of Large-scale Genetic Regulatory Networks (Supplementary Material). *J. Bioinformatics and Computational Biology*, 3(2):437–454, 2005.

[31] N. Rapin, O. Lund, and F. Castiglione. Immune System Simulation Online. *Bioinformatics*, 2011.

[32] M. Rathinam, L. R. Petzold, Y. Cao, and D. T. Gillespie. Stiffness in Stochastic Chemically Reacting Systems: The Implicit Tau-leaping Method. *The Journal of Chemical Physics*, 119(24):12784–12794, 2003.

[33] A. Stefanek, M. Vigliotti, and J. T. Bradley. Spatial Extension of Stochastic Pi Calculus. In *8th Workshop on Process Algebra and Stochastically Timed Activities*, pages 109–117, 2009.

[34] D. S. Wishart, R. Yang, D. Arndt, P. Tang, and J. Cruz. Dynamic Cellular Automata: An Alternative Approach to Cellular Simulation. *In Silico Biology*, 5(2):139–161, 2005.

A CPU and GPU Simulation Results

To substantiate our scalability claims, in this section we include simulation results comparing a CPU-based and a GPU-based implementation of a model of an antibacterial surface embedded with antibacterial agents (AmAs), where pH variations (Hydronium ion concentration) caused by the metabolic activity of bacteria triggers the release of AmAs. We anticipate similar scalability results for the example in Section 4.

The simulation results shown in Figure 11 are from our work "*Simulation And Study Of Large-Scale Bacteria-Materials Interactions Via BioScape Enabled By GPUs*" to appear in the proceedings of ACM-BCB 2012 [22]. In Figure 11(Left b), the GPU simulation results show that initial concentration of surface bound AmAs of 10,000 particles are sufficient to kill 1000 bacteria in 1000 time steps, while (Left c) and (d) plots exhibit insufficient initial concentrations to control infection.

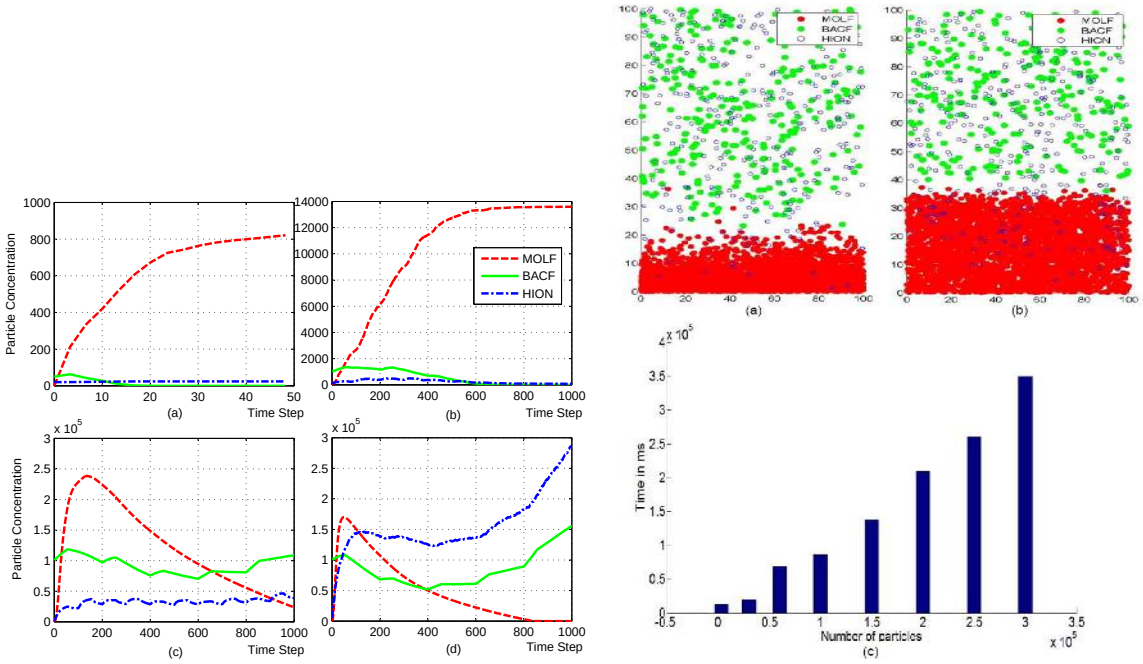


Figure 11: (Left) Results of CPU and GPU simulation. (a) CPU simulation results for 50 BacF/1,000 MolB. (b,c,d) GPU simulation results for different initial concentrations: (b) 1,000 BacF/10,000 MolB, (c) 100,000 BacF/30,000 MolB, (d) 100,000 BacF/200,000 MolB. (Right) (a,b) Particle distribution at time 10 and 100. (c) Performance of GPU simulation in real execution time w.r.t total number of particles.

RNA-interference and Register Machines

Masahiro Hamano
PRESTO , Japan Science and Technology Agency (JST)
4-1-8 Honcho Kawaguchi, Saitama 332-0012, JAPAN.
hamano@is.s.u-tokyo.ac.jp

RNA interference (RNAi) is a mechanism where small RNAs (siRNAs) directly control gene expression without the bypass of proteins. This mechanism consists of interactions among RNAs and small RNAs both of which are either of single or double stranded forms. The *target* of the mechanism is mRNA to be degraded or aberrated and the *initiator* of it is double stranded RNA (dsRNA) to be cleaved into siRNAs. Observing RNAi's digital nature, we represent RNAi as a Minsky register machine so that (i) The two registers hold single and double stranded RNAs respectively, and (ii) Machine's instructions are interpreted by interactions of enzyme (Dicer), siRNA (with RISC complex) and polymerization (RdRp) to the appropriate registers. Interpreting RNAi as a computational structure, we investigate computational meaning of RNAi, especially its complexity. First, the machine is interpreted by Chemical Ground Form (CGF), which interpretation though has an error to have wrong jumps. Second, recursive RNAi is modelled in order to inhibit the wrong jumps, where siRNA targets not only mRNA but also Dicer and RISC of the machine's instructions of our interpretations. Probabilistic termination is investigated in the machine interpreting recursive RNAi. Third, we show that CGF with delayed inputs provides a precise encoding of RNAi in order to gain Turing completeness.

1 Introduction

RNA interference (RNAi), also known as RNA silencing, is a mechanism where a small interfering RNA (siRNA) originating from double stranded RNA (dsRNA) directly controls gene expressions of a target mRNA [1, 7]. The two key steps of RNAi are as follows:
(i) A dsRNA is cleaved into small siRNA's fragments by the enzyme called Dicer.
(ii) A single strand of one small siRNA is recruited by the argonaute protein to form a complex called RISC. Then RISC, using the siRNA as a template, identifies matching sequences in a target mRNA, and leads the mRNA to degradation or aberration (see the right semicircle of Figure 1).
Given the two steps, we can say that the initiator of RNAi is dsRNA (for supplying siRNAs) and its target is mRNA (to be degraded or aberrated by a siRNA in a Watson-Crick complementary manner).

Moreover RNAi has a third step to provide a circular pathway from the target to the initiator [2, 11]:
(iii) By using an aberrant mRNA resulting from (ii) as a template, a dsRNA is produced by polymerization of RNA-dependent RNA polymerase (RdRp) (see the left semicircle of Figure 1).

With each step being digital and their combination maintaining the circularity, RNAi resembles a kind of (digital) computation. This observation is a starting point of the paper: Can RNAi be viewed as a digital computation ? This leads to a natural question what is a computational meaning of RNAi and how computationally complex RNAi is. The purpose of this paper is to answer this question.

In this paper, first, we observe that RNAi can be modelled as a Minsky register machine. The Minsky register machine is a Turing complete model of computation, that (instead of an infinite tape for Turing

Pre-Proceedings of 6th Workshop on Membrane Computing and Biologically Inspired Process Calculi
MeCBIC 2012, Pages 121–132.

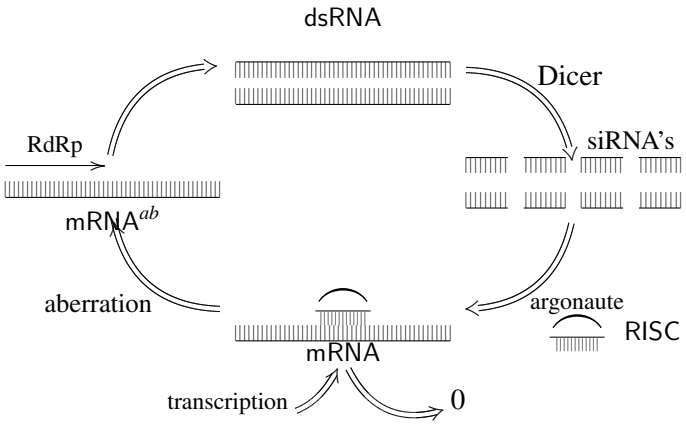


Figure 1: RNA interference

machine) comes equipped with two registers (holding numbers) and a finite number of instructions (increment and decrement/jump) on a certain register [18]. While most biological computational models so far known are based on the Turing machine model via common analogy of DNA as a single tape [10], our Minsky machine interpretation presented in this paper is intrinsic to the RNAi mechanism such that RNAs have both formations of single and double strands. This makes the interference captured by interactions of multi processes, which single and double stranded RNAs form in various length. We first present a naive machine model $\mathbf{RM}_{\mathsf{RNAi}}$ of RNAi so that the two registers are realized respectively by the initiator (dsRNA) and the target (mRNA) of RNAi. Increment/Decrement instructions on the registers are realized by chemical reactions mediated by enzymes and proteins (e.g., RdRp and transcription/Dicer and RISC). Second, however the above naive model lacks any rigorous computational language hence needs giving a syntactical analysis. By virtue of the interactive nature of RNAi and of its machine $\mathbf{RM}_{\mathsf{RNAi}}$, we investigate process calculus interpretations, especially those based on CCS and the π-calculus. Milner's π-calculus is a mathematical model of concurrent computation, which is powerful even in comparison to the lambda calculus (a functional model for computing) and Turing machines [16, 22]. Capturing RNAi as a computational structure, our interpretation aims to extract the computational meaning of RNAi, especially its complexity.

Motivated by Zavattaro-Cardelli [26], we describe our machine $\mathbf{RM}_{\mathsf{RNAi}}$ in the calculus of Chemical Ground Form (CGF), which is a minimal fragment of CCS equipped with interaction rates for each channel, hence a subset of the stochastic π-calculus [20]. CGF is introduced by Cardelli [3] and in spite of its simpleness, CGF sufficiently describes chemical kinetics compositionally by giving correspondence to a stochastic semantics of continuous time Markov chains. However as computational language without errors, the primitive description of CGF lacks any direct representation of the zero-tests for the registers, which causes certain errors of instructions of encoded $\mathbf{RM}_{\mathsf{RNAi}}$ to allow wrong jumps. To avoid the wrong probabilistic jump, a kind of inhibitor is necessary for machine instructions. To realize this biologically we consider *recursive* RNAi (recRNAi), which is known to be an extension of RNAi [14, 21, 25], where siRNA produced during RNAi inhibits not only mRNA but also RISC and Dicer. Biologically the extension is obtained by adding a feedback linkage to RNAi. The recRNAi is directly represented by a register machine $\mathbf{RM}_{\mathsf{recRNAi}}$, where the interactions of siRNAs are naturally interpreted as the inhibitors to instructions. We describe the machine in terms of CGF with fixed points. A probabilistic termination is investigated in the encoded system of recRNAi and Turing completeness up to any degree of precision

is shown.

Finally, we investigate a minimal extension over CGF to gain Turing completeness. For this we show that CGF with delayed inputs interprets the register machine by finite and precise encoding without errors. The delayed input is investigated for π-calculus in [15, 17, 24]. This kind of input enables self communication to interact with the prefix itself, which communication does not exist in ordinary process calculus. Answering to the same question how to complement CGF to gain Turing completeness, our approach differs from Cardelli-Zavattaro [4]'s Biochemical Ground Form (BGF) to augment interactions for association and dissociation over CGF. BGF makes possible representation of each register of machine by a linear polymer whose initial monomer represents the zero. The similar coding of register is also found in [12] for higher order π-calculus. Rather than straightforwardly representing the zero flags for the registers, whose biological substance is not necessarily clear, we propose a certain process algebra language to represent whether chemical reactions of Dicer and of RISC have targets or not for collisions.

Our motivation is to understand the known biological mechanism of RNAi in terms of register machines and to investigate how computational languages, basing on the primitive fragment CGF of the stochastic process calculus, are enhanced correspondingly to certain known biological mechanism over RNAi. Concretely, starting from naive interpretation $\mathbf{RM}_{\mathsf{RNAi}}$ of RNAi in Section 2, we show the following: (i) In Section 3, RNAi is represented modulo certain errors in CGF. (ii) In Section 4, in order to reduce the errors of (i), recRNAi is presented so that its register machine interpretation $\mathbf{RM}_{\mathsf{recRNAi}}$ is represented by CGF with fixed points. In the representation, inhibitors of the error jumps have certain biological meaning in recRNAi and probabilistic termination is shown. (iii) In Section 5, in order to remove the machines' errors inherent in (i) and (ii), CGF is augmented with delayed inputs so that the extended system precisely represents register machines.

2 A Naive Interpretation of RNAi in Minsky Register Machine

In this section, we show that RNAi is naively interpreted as Minsky register machine.

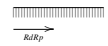
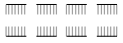
Definition 2.1 (Minsky Register Machine [18]) Minsky register machine consists of two registers r_1 and r_2, and a finite set of indexed instructions $I_1,\ldots,I_n$: Registers r_j ($j \in \{1,2\}$) hold non negative integer numbers and instructions I_i on a certain register r_j are of two kinds:

- (increment on r_j) $I_i = \mathsf{Inc}(r_j)$ increments 1 to the register r_j then proceeds to the next I_{i+1}.
- (decrement on r_j) $I_i = \mathsf{DecJump}(r_j,s)$ first tries to execute $\mathsf{Dec}(r_j)$, then proceeds to $\mathsf{Jump}_i(s)$: $\mathsf{Dec}(r_j)$ decrements the register r_j by 1 if r_j does not hold 0, and $\mathsf{Jump}_i(s)$ is I_{i+1} (respectively,I_s) if $\mathsf{Dec}(r_j)$ has been executed (respectively, otherwise).

A configuration of a Minsky register machine is determined by a triple $(I_i, r_1 = p, r_2 = q)$ where $p,q \in N$ are the states of the machine's counters and i is a label of a instruction. The machine stops when i is not defined.

Definition 2.2 (Register machine $\mathbf{RM}_{\mathsf{RNAi}}$ interpreting RNAi (cf. Figure 2)) RNAi is interpreted in the Minsky register machine $\mathbf{RM}_{\mathsf{RNAi}}$ consisting of the following data: Registers r_1 and r_2 hold species dsRNA and mRNA respectively so that the increment on r_1 (res. r_2) produces one dsRNA (res. one mRNA) and the decrement on r_1 (res. r_2) removes one dsRNA (res.one mRNA). The producing and removing for each molecules are realised by biological reactions as follows: The increment on the register r_1 is realized by *polymerization* RdRp with making aberrant mRNA template and that on the r_2 is

realised by *transcription*. The decrement on r_1 is realized by the *enzyme* Dicer which cleaves dsRNA into siRNAs and that on the r_2 is realized by RISC's complementary *degradation* of mRNA.

register	values	increment/decrement
r_1	$\overbrace{\mathsf{dsRNA} \mid \cdots \mid \mathsf{dsRNA}}^{m_1}$	$\mathsf{Inc}(r_1) := \mathit{RdRp}$ 
		$\mathsf{Dec}(r_1) := \mathit{Dicer}$ 

(a) register r_1

register	values	increment/decrement
r_2	$\overbrace{\mathsf{mRNA} \mid \cdots \mid \mathsf{mRNA}}^{m_2}$	$\mathsf{Inc}(r_2) :=$ transcription
		$\mathsf{Dec}(r_2) := \mathsf{RISC}$ 

(b) register r_2

Figure 2: Register Machine $\mathbf{RM}_{\mathsf{RNAi}}$

Remark 2.3 (Restriction to primer-independent synthesis of dsRNA**)** In order to realize the register machine interpretation of Definition 2.2, this paper assumes that the two species of dsRNA and mRNA have no connections so that the decrement and increment of the both species are independently done without affecting each other. The assumption corresponds to a biological restriction such that the synthesis of dsRNA considered in this paper is only primer-*independent* so that dsRNA is directly duplicated without any primer, as defined for the increment of the register r_1 of dsRNA. The another biological known synthesis of primer-*dependent* one violates the disconnection of the two species, in which synthesis siRNA triggers polymerization, hence enables RdRp to copy a normal mRNA on the register r_2. That is, the register r_2 may decrease in order to increase the register r_1. Our assumption is biologically appropriate since the two syntheses of dsRNA are investigated among experimental biologists from the standpoint that the two syntheses could explain difference of RNAi between plants and animals. See [2, 9] for the two syntheses of dsRNA.

3 RNAi as Chemical Reaction and Register Machines

In this section, we describe the register machine $\mathbf{RM}_{\mathsf{RNAi}}$ in Section 2 in terms of a rigorous computational language of stochastic process algebra. Chemical Ground Form (CGF) is introduced by Cardelli [3] as a subset of π-calculus and of CCS [16] enriched with transition rates to channels. CGF models collision between molecules by complementary synchronous interactions (input ? and output !) by with channels with stochastic rates. The formal definition of CGF is as follows, where the notation $\vdots$ separates syntactical lists.

Definition 3.1 (Chemical Ground Form [3])
(Interaction Prefix)

$$\pi := \tau_{(r)} \vdots ?a_{(r)} \vdots !a_{(r)}$$

where τ for molecular decay and complementary $?a$ and $!a$ for molecular inaction. The parenthesized subscript (r) denotes reaction rate of the channel.
(parallel composition)

$$\mid$$

(choice)

$$\oplus$$

Parrnell composition $\mid$ models concurrent activities of events and choice $\oplus$ models race between events.

Then a CGF is a pair (E,P) of a set E of *reagents* and a initial *solution* P. A reagent $X_i = M_i$ for naming a chemical specie and *molecules* M_i for describing the interaction capabilities of the corresponding species. Solution is a multiset of variables, which is released by interactions:

$$\text{(Reagents) } E := 0 \,\vdots\, X = M, E \quad \text{(Molecule) } M := 0 \,\vdots\, \pi.P \oplus M \quad \text{(Solution) } P := 0 \,\vdots\, X \mid P$$

Computation of CGF is given in terms of Labelled Transition Graph of Definition 3.2. This is determined by reductions of the two kinds:

(decay of molecule)	$\cdots \oplus \tau_{(r)}.Q \oplus \cdots$	$\longrightarrow$	Q
(collision of molecules)	$\cdots \oplus ?a_{(r)}.Q \oplus \cdots \mid \cdots \oplus ?a_{(r)}.R \oplus \cdots$	$\longrightarrow$	$Q \mid R$

Definition 3.2 (Labelled Transition Graph of CGF (Definition 3.2.1 of [3])) Given a CGF (E,P), $Next(E,P)$ is defined to consist of the following labelled transition, where $P^\dagger$ denotes the normalized form of a process P where the variables are sorted in lexicographical order.

- $(\{m.X.i\} : P^\dagger \to^r S^\dagger)$ such that $P^\dagger.m = X$ and $E.X.i = \tau_{(r)}.Q$ and $S = P^\dagger \backslash m \mid Q$.
- $(\{m.X.i, n.Y.j\} : P^\dagger \to^r S^\dagger)$ such that $P^\dagger.m = X$ and $P^\dagger.n = Y$ and $m \neq n$ and $E.X.i = ?a_{(r)}.Q$ and $E.Y.j = !a_{(r)}.R$ and $S = P^\dagger \backslash m, n \mid Q \mid R$.

The labelled transition graph $LTG(E,P)$ of (E,P) is defined as follows:
$LTG(E,P) = \cup_n \Psi_n$ with $\Psi_0 = Next(E,P)$ and $\Psi_{n+1} = \cup\{Next(E,Q) \mid \ Q \text{ is a state of } \Psi_n\}$

The increment operators on r_1 and on r_2 are realized by the following chemical reactions, where mRNA^{ab} denotes an aberrant mRNA.
(Polymerization)

$$\mathsf{RdRp} + \mathsf{mRNA}^{ab} \quad \longrightarrow \mathsf{dsRNA} \quad \text{for } r_1$$

(Transcription)

$$\longrightarrow \mathsf{mRNA} \quad \text{for } r_2$$

After the reactions, we go to the next instruction I_{i+1}. Hence every increment instruction $I_i = \mathsf{Inc}(r_j)$ is formalized directly for $j \in \{1,2\}$ as follows:
(Increment $I_i = \mathsf{Inc}(r_j)$)

$$\begin{aligned} I_i = &\quad \mathsf{RdRp} \mid \tau.I_{i+1} \quad j = 1 \\ I_i = &\quad \mathsf{mRNA} \mid \tau.I_{i+1} \quad j = 2 \end{aligned}$$

The more subtle part is decrement operators. The decrement operators on r_1 and on r_2 are realized by the following chemical reactions:
(cleavage)

$$\mathsf{dsRNA} + \mathsf{Dicer} \quad \longrightarrow \mathsf{siRNA}'s \quad \text{for } r_1 \tag{1}$$

(degradation)

$$\mathsf{mRNA} + \mathsf{RISC} \quad \longrightarrow \widetilde{\mathsf{mRNA}} + \mathsf{RISC} \quad \text{for } r_2 \tag{2}$$

where in (2) $\widetilde{\mathsf{mRNA}}$ denotes either 0 or mRNA^{ab}.

The above chemical reactions guarantee that Dicer and RISC interact to the respective registers of dsRNA and mRNA for the sake to eliminate each one of them. Although the two are the key agents for the decrement of each registers, Dicer and RISC have a different nature observed in the above chemical reactions: RISC is reused for degradation so that it retains its occurrence at right-hand-side of (2), while Dicer catalyzes (1) by disappearing throughout the reaction (1).

In order for the registers to be decremented, we interpret registers as follows:
(Registers)
Register r_1

$$\mathsf{dsRNA} \quad := ?a_1.(\mathsf{siRNA} \mid \cdots \mid \mathsf{siRNA})$$

Register r_2

$$\mathsf{mRNA} \quad := ?a_2.(\tau.0 \oplus \tau.\mathsf{mRNA}^{ab})$$

They represent that dsRNA and mRNA disappear respectively by transforming into siRNAs and by degradation or aberration.

If the chemical reaction happens in the presence of dsRNA (res. mRNA), then we proceed to the instruction I_{i+1}. Otherwise (i.e., if the reaction does not take place due to the absence of dsRNA (res. mRNA)), then we jump to the instruction I_s. Thus in a primitive description of CGF, every decrement instruction $I_i = \mathsf{DecJump}(r_j, s)$ may be represented as follows:
(decrement instruction $I_i = \mathsf{DecJump}(r_j, s)$)
$j = 1$

$$I_i \quad = !a_1.(0 \mid I_{i+1}) \oplus \tau.I_s \quad \text{with Dicer} = !a_1.(0 \mid I_{i+1})$$

$j = 2$

$$I_i \quad = !a_2.(\mathsf{RISC} | I_{i+1}) \oplus \tau.I_s \quad \text{with RISC} = !a_2.(\mathsf{RISC} \mid I_{i+1})$$

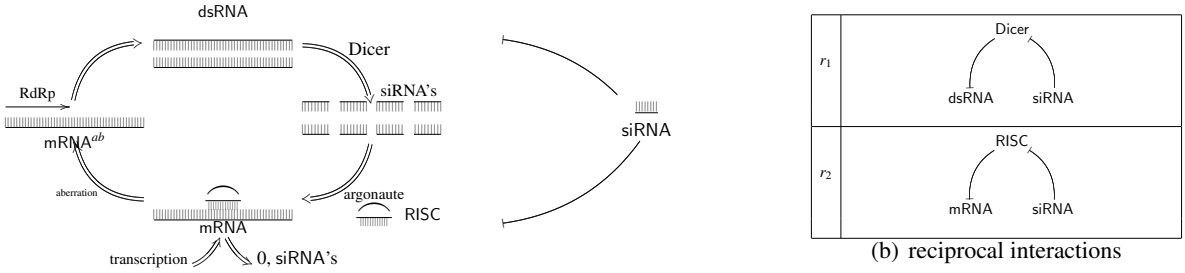
The above recursive definition of RISC for $j = 2$ corresponds to the recycle of RISC described in the degradation (2).

The above definition of decrement instructions has a certain error that the jump to I_s accidentally happens even if the register is non zero (i.e., in spite of the presence of the channel $?a_j$). The error is due to the absence of zero-test of the registers in the above interpretations of decrement instructions, which test though is impossible to be directly represented in terms of CGF. The absence is also discussed in Soloveichik, et al.[23] for investigating stochastic chemical reaction networks. The lacking of zero-test is a main origin of Turing incompleteness of CGF shown in [26] and in order to recover this, Cardelli-Zavattaro presents Biochemical Ground Form [4] as minimalistic extension of CGF.

4 Recursive RNAi and Probabilistic Termination

In this section, we model recursive RNAi in order to improve the defect of Section 3 that the machine interpretation $\mathbf{RM}_{\mathsf{RNAi}}$ has wrong jumps in terms of CGF. We extend RNAi mechanism into recursive RNAi (recRNAi) so that its register machine interpretation $\mathbf{RM}_{\mathsf{recRNAi}}$ in terms of CGF + fixed points guarantees a probabilistic termination of the machine. In this extended mechanism, siRNAs produced during the interference targets not only mRNA but also Dicer and RISC themselves. See Figure 3(a),

whose left hand is the usual RNAi but siRNAs are produced not only by Dicer but also by RISC's degrading mRNA. The right hand of Figure 3(a) describes inhibition arrows from the right most siRNA to Dicer and to RISC in the circle. The mechanism is called recursive because RISC complex containing siRNA is not only for degrading but also for being degraded. With the recursiveness of RNAi, the defect of the decrement operators of Section 3 to have wrong jumps is ameliorated so that siRNAs growing during RNAi work as inhibitors of the decrement operators.



(a) Recursive RNAi

(b) reciprocal interactions

Figure 3: Recursive RNAi

In recRNAi, the chemical reactions involved in Dicer and in RISC are not only (1) and (2) respectively but also the following (3) and (4), respectively.

- (degradation of Dicer)

$$\mathsf{siRNA} + \mathsf{Dicer} \quad \longrightarrow \quad 0 \tag{3}$$

- (degradation of RISC)

$$\mathsf{siRNA} + \mathsf{RISC} \quad \longrightarrow \quad 0 \tag{4}$$

(1) and (3) provide reciprocal interactions on Dicer such that Dicer either makes dsRNA disappear by cleavage or Dicer is degraded by siRNA. Similar reciprocal interactions (2) and (4) for RISC. See Figure 3(b), for these reciprocal interactions for Dicer and RISC, each of which concerns a register r_j with $j \in \{1,2\}$ and the arrows are for inhibitions.

We interpret recRNAi into a register machine $\mathbf{RM}_{\mathsf{recRNAi}}$ in terms of CGF with fixed points.

Definition 4.1 ($\mathbf{RM}_{\mathsf{recRNAi}}$ in CGF with fixed points)
$\mathbf{RM}_{\mathsf{recRNAi}}$ is interpreted in CGF as follows:

- Registers and $I_i = \mathsf{Inc}(r_j)$ are the same as Section 3: That is
  - (Register r_1)
$$\mathsf{dsRNA} := ?a_1.(\mathsf{siRNA} \,\big|\, \cdots \,\big|\, \mathsf{siRNA})$$
  - (Register r_2)
$$\mathsf{mRNA} := ?a_2.(\tau.0 \oplus \tau.\mathsf{mRNA}^{ab})$$
  - (Increment $I_i = \mathsf{Inc}(r_j)$)
$$\begin{aligned} I_i = &\quad \mathsf{RdRp} \,\big|\, \tau.I_{i+1} \quad j = 1 \\ I_i = &\quad \mathsf{mRNA} \,\big|\, \tau.I_{i+1} \quad j = 2 \end{aligned}$$

The decrement instruction is represented as follows together with interpretation of siRNA.

- (Decrement $I_i = \mathsf{DecJump}(r_j, I_s)$)

$$\begin{array}{rcl} I_i & = & !a_j.(0 \mid I_{i+1}) \oplus \tau.(!s.I_i \oplus \tau.I_s) \\ \mathsf{siRNA} & = & ?s.\mathsf{siRNA} \end{array} \tag{5}$$

In the above definition (5) of I_i, when $j = 1$ (res. $j = 2$), the left term $!a_j.(0 \mid I_{i+1})$ corresponds to Dicer (res. RISC) as an agent cleaving dsRNA (res. degrading mRNA) and the right term $\tau.(!s.I_i \oplus I_s)$ corresponds to Dicer (res. RISC) as an agent being degraded by siRNA. Hence our definition of I_i is intrinsic to the reciprocal interactions of Dicer and RISC, intrinsic to recursive RNAi in the presence of siRNA.

The iterative definition of (5) is a fixed point definition of I_i in CGF by using agent variable X so that

$$I_i \quad = \quad \mathsf{fix}_X.[a.(0 \mid I_{i+1}) \oplus \tau.(!s.X \oplus \tau.I_s)]$$

The fixed point definition stems from Zavattaro-Cardelli [26], but we point out that the definition has a certain biological counter part as discussed above. We follow their line to obtain the main theorem of this section by slightly modifying the corresponding result of [26].

Given a state $(I_i, r_1 = l_1, r_2 = l_2)$ of register machine and a natural number h, the solution in $\mathbf{RM}_{\mathsf{recRNAi}}$ is defined by

$$\left[\!\left[(I_i, r_1 = l_1, r_2 = l_2)\right]\!\right]_h \quad := \quad I_i \mid \prod_{l_1} \mathsf{dsRNA} \mid \prod_{l_2} \mathsf{mRNA} \mid \prod_{h} \mathsf{siRNA}$$

where I_i on the right hand is that of Definition 4.1.
That is, $\left[\!\left[(I_i, r_1 = l_1, r_2 = l_2)\right]\!\right]_h$ is the encoding of the state in $\mathbf{RM}_{\mathsf{recRNAi}}$ together with h iterative siRNAs.

Proposition 4.2 (correspondence of computations between machine and $\mathbf{RM}_{\mathsf{recRNAi}}$) *Suppose a one step computation of register machine is given by the following:*

$$(I_i, r_1 = l_1, r_2 = l_2) \quad \longmapsto \quad (I_j, r_1 = l_1^{'}, r_2 = l_2^{'})$$

We have the following for the solutions of the two states of the computation:

- *If $l_j = 0$ and $I_i = \mathsf{Inc}(r_j)$ or $I_i = \mathsf{DecJump}(r_j, s)$, then the solution $\left[\!\left[(I_i, r_1 = l_1, r_2 = l_2)\right]\!\right]_h$ can reach to the solution $\left[\!\left[(I_j, r_1 = l_1^{'}, r_2 = l_2^{'})\right]\!\right]_h^{\dagger}$ with the probability* 1.
- *If $l_j > 0$ and $I_i = \mathsf{DecJump}(r_j, s)$, the solution $\left[\!\left[(I_i, r_1 = l_1, r_2 = l_2)\right]\!\right]_h$ can reach to a solution $\left[\!\left[(I_j, r_1 = l_1^{'}, r_2 = l_2^{'})\right]\!\right]_k^{\dagger}$ for some natural number $k \geq h+1$ with the probability $> 1 - \frac{1}{h}$.*

Proof.
- The case where $l_j = 0$. Since the assertion is direct for $I_i = \mathsf{Inc}(r_j)$, we demonstrate for $I_i = \mathsf{DecJump}(r_j, s)$. The probability for computations in $\mathbf{RM}_{\mathsf{recRNAi}}$ pass through the solution of R.H.S. is

$$\sum_{i=0}^{\infty} (\frac{h}{h+1})^i \times \frac{1}{(h+1)} \quad = 1$$

The calculation corresponds to the following diagram:

$$l_j = 0 \qquad I_i \underset{h}{\overset{1}{\rightleftarrows}} \bullet \xrightarrow{1} I_s = I_j$$

- The case where $l_j \neq 0$.
The probability for computations in $\mathbf{RM}_{\mathsf{recRNAi}}$ pass through the solution of R.H.S. is

$$\sum_{i=0}^{\infty}(\frac{1}{l_j+1} \times \frac{h}{h+1})^i \times \frac{l_j}{l_j+1} \quad > \quad 1-\frac{1}{h}$$

See the following diagram for the calculation:

$$l_j \neq 0 \qquad I_i \underset{h}{\overset{1}{\rightleftarrows}} \bullet \xrightarrow{1} I_s \qquad \bullet \xrightarrow{l_j} I_{i+1} = I_j$$

□

We have the main theorem of this section.

Theorem 4.3 (probabilistic termination) *The following are equivalent:*

- *A Minsky register machine starting from a state $(I_j, r_1 = l_1, r_2 = l_2)$ terminates.*
- *A CGF $(\mathbf{RM}_{\mathsf{recRNAi}}, [\![(I_j, r_1 = l_1, r_2 = l_2)]\!]_h)$ probabilistically terminates with probability*

$$> 1 - \sum_{k=h}^{\infty} \frac{1}{k}.$$

Proof. Note first that after the execution of a decrement instruction, the number of siRNA increases at lease one. This is because at least one siRNA is produced by Dicer cleavage or by RISC's degrading mRNA. By Proposition 4.2 a computation of register machine containing d decrement instructions is faithfully reproduced with probability greater than the following:

$$(1-\frac{1}{h})(1-\frac{1}{h+k_1})\cdots(1-\frac{1}{h+k_1+\cdots+k_d}) \quad \geq \prod_{k=h}^{h+d}(1-\tfrac{1}{k}) \quad > 1-\sum_{k=h}^{h+d}\frac{1}{k}$$

where $k_i \geq 1$ is the number of siRNAs produced by the corresponding decrement instruction. □

5 Precise Embedding of Register Machine into CGF with Delayed Inputs

In this section, we investigate a minimal extension over CGF to gain Turing completeness. For this, we extend CGF by allowing a new kind of interaction, *delayed inputs* among entangled channels of CGF. Our extended system is naturally induced by delayed inputs system of π-calculus (Section 10.4 of [22]), given that CGF is a subsystem of π-calculus with reaction rates. The related systems are investigated in Milner's synchronous version of π-calculus [17] and π_ε-calculus (enabled pi) of [24]. We augment two features, (*self communication*) and (*guarding*), which are not present in CGF.

We first extend the definition of molecules of CGF to allow nesting inputs and outputs:

$$\text{(Molecule)} \quad M := 0 \,\vdots\, \pi.P \oplus M \,\vdots\, \pi.M$$

The first feature enables a process to communicate with itself. This arises by entangling prefixes of inputs and outputs. In its simple form, self communication amounts to

(Self Communication)

$$!a.(?a.P \oplus M) \quad \longrightarrow \quad P$$

The second feature is for regulating the first feature so that self communication is inert while there is a complementary channel outside. In its simple form, guarding amounts to

(Guarding)

The following self communication is enabled only when $b \neq a$.

$$?b.R \oplus N \,\big|\, !a.(?a.P \oplus M) \quad \longrightarrow \quad ?b.R \oplus N \,\big|\, P$$

That is, when $b = a$, the self communication is guarded.

The two additional features are formally specified in terms of LTG for the extended system:

Definition 5.1 (Labelled Transition Graph for CGF with delayed inputs) Given a CGF (E,P) with delayed inputs, $Next(E,P)$ is defined to consist of the labelled transitions defined in Definition 3.2 together with the following labelled transition:

- $(\{m.X.ij\} : P^\dagger \rightarrow^r S^\dagger)$ such that $P^\dagger.m = X$ and $E.X.i = !a_{(r)}.Q$ and $Q_j = ?a_{(r)}.\bar{Q}$ and $S = P^\dagger \backslash m \,\big|\, \bar{Q}$. This transition is subject to the following guarding condition:

$$\forall n \neq m \; \forall i \text{ no prefix of } E.Y.i \text{ is } ?a_{(r)} \text{ with } Y = P^\dagger n$$

CGF with delayed inputs provides a precise encoding of Minsky Register Machine:

Definition 5.2 (encoding of register machine in CGF with delayed inputs)

- (Register r_j holding a number l_j)

$$r_j = \overbrace{?a_j.?b.0 \,\big|\, \cdots \,\big|\, ?a_j.?b.0}^{l_j}$$

- (Instructions I_i of two kinds)

$$\begin{aligned} I_i = \mathsf{Inc}(r_j) &= \tau.(!a_j.0 \,\big|\, I_{i+1}) \\ I_i = \mathsf{DecJump}(r_j,s) &= !a_j.(?a_j.I_s \oplus !b.I_{i+1}) \end{aligned} \tag{6}$$

The preciseness of the encoding of Definition 5.2 is shown for the decrement instruction as follows:

- (The case where $l_j = 0$)
  In the absence of $?a_j.b.0$, the decrement instruction (6) communicates by itself through the outermost $!a_j$

$$!a_j.(?a_j.I_s \oplus ?b.I_{i+1}) \quad \longrightarrow \quad I_s$$

- (The case where $l_j \neq 0$)
  In the presence of $?a_j.b.0$, the decrement instruction (6) interacts to the register r_j we have

$$?a_j.?b.0 \mid \mathsf{DecJump}(r_j, s)$$

  which is reduced to the following:

$$\begin{array}{rcl} ?a_j.?b.0 \mid !a_j.(?a_j.I_s \oplus !b.I_{i+1}) & \longrightarrow & ?b.0 \mid ?a_j.I_s \oplus !b.I_{i+1} \\ & \longrightarrow & 0 \mid I_{i+1} \end{array}$$

Formally, a state $(I_i, r_1 = l_1, r_2 = l_2)$ of Minsky Register Machine is encoded in CGF with delayed inputs by

$$[\![(I_i, r_1 = l_1, r_2 = l_2)]\!] \quad := \quad I_i \mid \prod_{l_1} ?a_1.?b.0 \mid \prod_{l_2} ?a_2.?b.0$$

where I_i on the right hand is that of Definition 5.2.

Then the preciseness of the encoding shown above proves the following theorem.

Theorem 5.3 (correctness) *Suppose a one step computation of register machine is given*

$$(I_i, r_1 = l_1, r_2 = l_2) \quad \longmapsto \quad (I_j, r_1 = l_1', r_2 = l_2').$$

Then the solution $[\![(I_i, r_1 = l_1, r_2 = l_2)]\!]$ *reaches deterministically to the solution* $[\![(I_j, r_1 = l_1', r_2 = l_2')]\!]^{\dagger}$.

Corollary 5.4 *CGF with delayed inputs is Turing complete.*

6 Conclusion and Future Works

In this paper, RNAi is represented by register machine $\mathbf{RM}_{\mathsf{RNAi}}$ by virtue of multi-strand formations to which RNAs take (Section 2). A stochastic process algebra CGF provides a primitive description of $\mathbf{RM}_{\mathsf{RNAi}}$ (Section 3). In order to prevent errors caused by the description, recursive RNAi is shown to realize a biological counterpart of inhibitors of the machine so that siRNAs growing during RNAi targets to the decrement instructions (Section 4). A probabilistic termination is obtained in the recursive RNAi (Theorem 4.3). As a completion of CGF to gain Turing completeness, CGF with delayed inputs is presented (Section 5).

As a future work, we are interested in investigating RNAi with another biological pathway of *primer dependent* synthesis of dsRNA [19]. The pathway is also for maintaining the circularity depicted in Introduction. While polymerization considered in this paper is primer *in*dependent so that dsRNA is directly duplicated without any primer, in the primer dependent one, polymerization for producing dsRNA is triggered by siRNA, which enables RdRp to copy non-aberrant mRNA. Computational meaning of RNAi with primer dep. synthesis is much more involved because siRNA does not only work for inhibitor but also for trigger to produce the initiator of dsRNA. Comparison of the two pathways is an important topic for experimental biology in order to explain difference of RNAi between plants and animal [1, 2, 7, 19].

From an aspect on computational language, it is an interesting future work how the κ-calculus [5, 6] captures recRNAi. The κ-calculus is known to be a Turing complete fragment of the stochastic π-calculus, and the author of this paper in [9] shows that the rule based modelling of κ-calculus provides a compact description to discriminate the two syntheses for dsRNA.

References

[1] David Baulcombe, *RNA Silencing in Plants*, Nature. **431**, 356-63, (2004)

[2] Peter Brodersen and Olivier Voinnet, *The Diversity of RNA Silencing Pathways in Plants*, TRENDS in Genetics 22(5), 268280 (2006)

[3] Luca Cardelli, *On Process Rate Semantics*, Theor. Comput. Sci. **391**(3): 190-215 (2008)

[4] Luca Cardelli and Gianluigi Zavattaro, *Turing Universality of the Biochemical Ground Form*. Mathematical Structures in Computer Science **20**(1): 45-73 (2010)

[5] Vincent Danos, Jérôme Feret, Walter Fontana, Russell Harmer and Jean Krivine, *Rule-based Modelling of Cellular Signalling*, (CONCUR 2007), LNCS **4730**, Springer-Verlag. 17-41 (2007)

[6] Vincent Danos and Cosimo Laneve, *Formal Molecular Biology*, Theor. Comput. Sci., **325** (1), 69 - 110 (2004)

[7] A. Fire, S. Xu, M. Montgomery, S. Kostas, S.Driver and C. Mello, *Potent and Specific Genetic Interference by Double-stranded RNA in Caenorhabditis Elegans*, Nature **391** (6669): 806811 (1998)

[8] M.A.C. Groenenboom, A.F.M. Marée and P. Hogeweg, *The RNA Silencing Pathway: The Bits and Pieces That Matter*, PLoS Comput. Biol. 1(2), 155165 (2005)

[9] Masahiro Hamano, "Sustainability of RNA Interference in Rule Based Modelling", accepted by the third International Workshop on Static Analysis and Systems Biology (SASB 2012), 12pages.

[10] Zoya Ignatova, Karl-Heinz Zimmermann, Israel Martínez-Pérez, "DNA Computing Models", Springer (2008)

[11] Richard A. Jorgensen, *RNA Traffics Information Systemically in Plants*, Proc. Natl. Acad. Sci. USA, **99**(18) 11561-11563 (2002)

[12] Ivan Laney's, Jorge A. Pérez, Davide Sangiorgi and Alan Schmitt, *On the Expressiveness and Decidability of Higher-orderb Process Calculi*. Inf. Comput. **209**(2): 198-226 (2011)

[13] E. Levine, Z. Zhang, T. Kuhlman, and T. Hwa, *Quantitative Characteristics of Gene Regulation by Small RNA*, PLoS Biol **5**, e229 (2007)

[14] W. F. Marshall, *Modeling Recursive RNA Interference*, PLoS Computational Biology, **4** (9) e1000183. (2008)

[15] M. Merro and D. Sangiorgi, *On Asynchrony in Name-passing Calculi*, In Proc. of (ICALP'98) LNCS, **1443**, pp. 856-867, (1998)

[16] Robin Milner, "Communicating and Mobile Systems: the π-calculus", Cambridge University Press (1999).

[17] Robin Milner, *Action Structure for Synchronous pi-Calculus*, FCT (1993) LNCS **710**, Springer-Verlag. 87-105 (1993)

[18] Marvin Minsky, "Computation Finite and Infinite Machines (1st ed.)." Englewood Cliffs, N. J.: Prentice-Hall (1967)

[19] Julia Pak and Andrew Fire, *Distinct Populations of Primary and Secondary Effectors During RNAi in C. elegans*, Science. **315**, 241-244 (2007)

[20] C. Priami, A. Regev, E. Shapiro and W. Silverman, *Application of a Stochastic Name-passing Calculus to Representation and Simulation of Molecular Processes*, Information Processing Letters **80**, 25-31. (2001)

[21] MW. Rhoades, BJ. Reinhart, LP. Lim, CB. Burge, B. Bartel, et al. *Prediction of Plant microRNA Targets*. Cell **110** 51320. (2002)

[22] Davide Sangiorgi and David Walker, "The π-calculus: a Theory of Mobile Processes", Cambridge Universtity Press (2001)

[23] David Soloveichik, Matt Cook, Erik Winfree and Shuki Bruck, *Computation with Finite Stochastic Chemical Reaction Networks*. Natural Computing, **7** (4), 615-633. (2008)

[24] Franck van Breugel, *A Labelled Transition System for π_ε-Calculus*, Proceedings of (TAPSOFT), volume **1214** LNCS, Springer-Verlag. 312-336. (1997)

[25] Z. Xie, KD. Kasschau and JC. Carrington, *Negative Feedback Regulation of Dicer-like1 in Arabidopsis by microRNA-guided mRNA Degradation*. Curr Biol **13**: 784789. (2003)

[26] Gianluigi Zavattaro and Luca Cardelli, *Termination Problems in Chemical Kinetics*. Proc. CONCUR 2008, LNCS **5201**, Springer-Verlag. 477-491(2008)