Efficient, Correct Simulation of Biological Processes in the Stochastic Pi-calculus

Andrew Phillips and Luca Cardelli

Microsoft Research, 7 JJ Thomson Avenue, CB3 0FB Cambridge UK
{andrew.phillips,luca}@microsoft.com

Abstract. This paper presents a simulation algorithm for the stochastic π-calculus, designed for the efficient simulation of biological systems with large numbers of molecules. The cost of a simulation depends on the number of species, rather than the number of molecules, resulting in a significant gain in efficiency. The algorithm is proved correct with respect to the calculus, and then used as a basis for implementing the latest version of the SPiM stochastic simulator. The algorithm is also suitable for generating graphical animations of simulations, in order to visualise system dynamics.

1 Introduction

In recent years, there has been considerable research on designing programming languages for complex parallel computer systems. Interestingly, some of this research is also applicable to biological systems, which are typically highly complex and massively parallel. In particular, a mathematical programming language known as the stochastic π-calculus has recently been used to model and simulate a range of biological systems [7,12,13]. The calculus allows the components of a biological system to be modelled independently, rather than modelling the individual reactions. This allows large models to be constructed by composition of simple components [2]. The calculus also facilitates mathematical analysis of systems using a range of established techniques, which could eventually shed light on some of the fundamental properties of biological systems. Various stochastic simulators have been developed for the calculus [13,9,1], in order to perform virtual experiments on biological system models. Such *in silico* experiments can be used to formulate testable hypotheses on the behaviour of biological systems, as a guide to future experimentation *in vivo*.

Currently available simulators for the stochastic π-calculus are implemented based on standard theory of chemical kinetics, using an adaptation of the Gillespie algorithm [5]. This algorithm has the distinct advantage of being mathematically exact, enabling accurate simulation of biological models. Unfortunately, the algorithm is also highly computationally intensive, particularly when simulating large models. As a result, there has been considerable research on optimisations for the Gillespie algorithm, resulting in a plethora of alternatives, both exact and approximate [4,6,15]. Like the original algorithm, these alternatives are defined

M. Calder and S. Gilmore (Eds.): CMSB 2007, LNBI 4695, pp. 184–199, 2007.

in terms of systems of chemical reactions, the *de facto* standard for biological modelling. This reaction view of systems differs in many ways from the component view of the stochastic π-calculus. As a result, techniques for efficient simulation of chemical reactions cannot be directly applied to the stochastic π-calculus, but need to be adapted to account for the differences between the two formalisms [10]. Given these differences, and given the importance of efficiency in the stochastic simulation of biological models, research on efficient simulation algorithms for the stochastic π-calculus seems of interest. There has already been substantial research on efficient implementation techniques for variants of the π-calculus, in the context of programming languages for parallel computer systems [16]. However, this research does not take into account the specific properties of biological systems, which differ from most computer systems in fundamental ways. One key difference is that biological systems are often composed of large numbers of processes with identical behaviour, such as thousands of proteins of the same type.

This paper presents a simulation algorithm for the stochastic π-calculus, designed for the efficient simulation of biological systems with large numbers of molecules. The paper is structured as follows. Section 2 illustrates the principle of the simulation algorithm with the help of a biological example. Section 3 presents the full definition of the algorithm, and Sec. 4 outlines a proof of correctness with respect to the stochastic π-calculus. Finally, Sec. 5 shows how the algorithm can be mapped to executable program code, in order to implement a stochastic simulator.

2 Biological Example

This section introduces the simulation algorithm for the stochastic π-calculus, with the help of a biological example. The example describes a system of three genes with negative control that mutually repress each other, as presented in [11]. The system consists of an environment, which contains definitions for a $Gene(a, b)$ and a $Protein(b)$, together with a top-level process, which contains three genes executing in parallel :

$$\begin{aligned}
&\{Gene(a,b) = \tau_t.(Gene(a,b) \mid Protein(b)) + ?a.\tau_u.Gene(a,b),\\
&Protein(b) = !b.Protein(b) + \tau_d\}\\
&\vdash\\
&Gene(a,b) \mid Gene(b,c) \mid Gene(c,a)
\end{aligned}$$

A $Gene(a, b)$ is parameterised by its promoter region a, together with the promoter region b that is recognised by its transcribed proteins. The gene can perform one of two actions, represented as a choice $(+)$. Either it can transcribe a $Protein(b)$ by doing a stochastic delay τ_t, after which the new protein is executed in parallel with the gene, or it can block by doing an input $?a$ on its promoter region a and then unblock by doing a stochastic delay τ_u. A $Protein(b)$ can repeatedly do an output $!b$ on the promoter region b, or it can degrade by doing a stochastic delay τ_d. According to the reduction rules of the calculus, the input

$?b$ of a $Gene(b,c)$ can interact with the output $!b$ of a corresponding protein, becoming blocked as a result. The three genes in the system can mutually repress each other, since $Gene(a,b)$ produces proteins that can block $Gene(b,c)$, which produces proteins that can block $Gene(c,a)$, which produces proteins that can block $Gene(a,b)$, completing the cycle. Stochastic behaviour is incorporated into the system by associating each of the channels a,b,c with corresponding interaction rates given by $\rho(a),\rho(b),\rho(c)$, respectively, and by associating each of the delays τ_t,τ_u,τ_d with corresponding delay rates given by t,u,d. These rates are used to calculate the probabilities of all the reactions in the system, where the probability of a reaction is proportional to its rate.

The above system is simulated by encoding it to a system $E \vdash V$ of the stochastic π-machine, which consists of a machine environment E and a machine term V:

$$
\begin{array}{l}
\{Gene(a,b) = \tau_t.(Gene(a,b) \mid Protein(b)) + ?a.X(a,b), \\
X(a,b) = \tau_u.Gene(a,b), \\
Protein(b) = !b.Protein(b) + \tau_d\} \\
\vdash \\
\emptyset, \{a \mapsto (1{,}0{,}0{,}0), b \mapsto (1{,}0{,}0{,}0), c \mapsto (1{,}0{,}0{,}0), t \mapsto (3{,}3t)\}, \\
\{Gene(a,b) \mapsto 1, \{t \mapsto 1, a \mapsto (1{,}0)\}, \tau_t.(Gene(a,b) \mid Protein(b)) + ?a.X(a,b), \\
Gene(b,c) \mapsto 1, \{t \mapsto 1, b \mapsto (1{,}0)\}, \tau_t.(Gene(b,c) \mid Protein(c)) + ?b.X(b,c), \\
Gene(c,a) \mapsto 1, \{t \mapsto 1, c \mapsto (1{,}0)\}, \tau_t.(Gene(c,a) \mid Protein(a)) + ?c.X(c,a)\}
\end{array}
$$

The machine environment E is similar to a calculus environment, with the additional constraint that each choice of one or more actions must be associated with a corresponding identifier. In order to satisfy this constraint, the encoding creates a new definition $X(a,b)$, which keeps track of the number of genes in a blocked state. Note that the constraint is not enforced at the calculus level, since this would be too much of syntactic burden. Instead, the extra definitions are created by the encoding.

The machine term V consists of a set of channels Z, a store of reactions S and a heap of species H. The set Z denotes the set of all the private channels in the system, which is empty in this example. The store S records the apparent rate of all the delays and channels in the system. The apparent rate of a delay of rate r is given by the number of possible delays τ_r multiplied by r. The apparent rate of a channel x is given by the number of possible interactions on the channel multiplied by $\rho(x)$. This information is recorded in the Store S, where each delay is mapped to the number of delays and the apparent rate of the delay, and each channel is mapped to the number of inputs, outputs, mixed interactions (i.e. the number of pairs of inputs and outputs that cannot interact with each other) and the apparent rate of the channel. In the above example, there are initially three delays of rate t and one input on each channel a,b,c. The heap H records information about each species that is currently being simulated, including the population of the species, the choice of actions that the species can perform and the number of each type of action. Initially there are three species in the system, $Gene(a,b)$, $Gene(b,c)$ and $Gene(c,a)$, where each gene with a given set

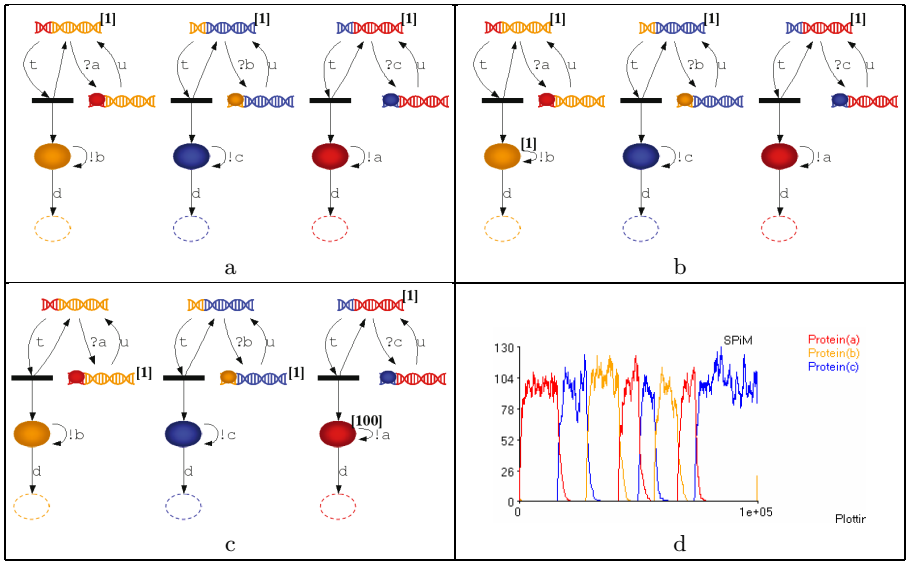

Fig. 1. Graphical representation of a network of three genes with inhibitory control that mutually repress each other. It is assumed that $\rho(a) = \rho(b) = \rho(c)$ and $\rho(a) \gg t \gg d \gg u$. Initially there is one copy of each gene (a), and one of the genes then transcribes a protein (b). After a sequence of reduction steps, two of the genes become blocked, and the third gene produces 100 proteins (c). The mutual repression of genes gives rise to alternate oscillations of protein levels, as shown in the simulation plot (d), where the vertical axis represents the number of proteins and the horizontal axis represents the simulation time. The results were obtained with $\rho(a) = 1.0$, $t = 0.1$ $d = 0.001$ and $u = 0.0001$.

of parameters denotes a separate species. The $Gene(a, b)$ can do a delay at rate t or an input on channel a, and similarly for the remaining genes in the system.

A corresponding graphical representation for this system is shown in Fig. 1(a) based on [11], where a separate graph is drawn for each gene. Each shape in the graph represents a species, and each labelled edge represents an action that the species can perform. Multiple edges from a species correspond to a choice, while multiple edges from a horizontal bar correspond to a parallel composition.

In order to execute this system, the stochastic π-machine chooses one of the possible reactions using an adaptation of the Gillespie algorithm [5], where the probability of a reaction is proportional to its rate. Initially, the machine can do one of three delay reactions with rate t, where the apparent rate of the delay is $3t$. The machine chooses one of these delays with equal probability. Suppose the $Gene(a, b)$ is chosen to perform the delay. An additional $Protein(b)$ is produced, giving rise to the following machine term, which corresponds to Fig. 1(b):

$$\emptyset, \{a \mapsto (1,0,0,0), b \mapsto (1,1,0,\rho(b)), c \mapsto (1,0,0,0), d \mapsto (1,d), t \mapsto (3,3t)\},$$
$$\{Gene(a,b) \mapsto 1, \{t \mapsto 1, a \mapsto (1,0)\}, \tau_t.(Gene(a,b) \mid Protein(b)) + ?a.X(a,b),$$
$$Protein(b) \mapsto 1, \{d \mapsto 1, b \mapsto (0,1)\}, !b.Protein(b) + \tau_d,$$
$$Gene(b,c) \mapsto 1, \{t \mapsto 1, b \mapsto (1,0)\}, \tau_t.(Gene(b,c) \mid Protein(c)) + ?b.X(b,c),$$
$$Gene(c,a) \mapsto 1, \{t \mapsto 1, c \mapsto (1,0)\}, \tau_t.(Gene(c,a) \mid Protein(a)) + ?c.X(c,a)\}$$

The Gillespie algorithm is then used to execute the next reaction. Assuming $\rho(b) \gg t$ there is a high likelihood that a reaction on b will be chosen, which blocks $Gene(b,c)$. Subsequently, a $Protein(a)$ is transcribed, which blocks $Gene(a,b)$. Since both $Gene(b,c)$ and $Gene(a,b)$ are blocked, no more $Protein(c)$ or $Protein(b)$ are produced. Eventually 100 copies of $Protein(a)$ are produced, giving rise to the following machine term, which corresponds to Fig. 1(c):

$$\emptyset, \{a \mapsto (0,100,0,0), b \mapsto (1,0,0,0), c \mapsto (1,0,0,0), t \mapsto (1,t), u \mapsto (2,2u), d \mapsto (100,100d)\},$$
$$\{X(a,b) \mapsto 1, \{u \mapsto 1\}, \tau_u.Gene(a,b),$$
$$X(b,c) \mapsto 1, \{u \mapsto 1\}, \tau_u.Gene(b,c),$$
$$Gene(c,a) \mapsto 1, \{t \mapsto 1, c \mapsto (1,0)\}, \tau_t.(Gene(c,a) \mid Protein(a)) + ?c.\tau_u.X(c,a),$$
$$Protein(a) \mapsto 100, \{d \mapsto 1, a \mapsto (0,1)\}, !a.Protein(a) + \tau_d\}$$

This represents the first oscillation cycle from the simulation results of Fig. 1(d). Note how the graphical representation relies on the ability to count the number of copies of each species, in order to label the corresponding node with its population. The graphs are generated using a translation to DOT syntax [3] based on [11]. The resulting sequence of pictures can also be used to produce a 3D animation of the simulation, as shown in [9]. Note that the simulation algorithm does not require the countable species of a simulation to be known beforehand. Rather, the populations of species are grouped on-the-fly according to the species name and parameters. In the above example, at the start of the simulation there are three definitions for $Gene(a,b)$, $X(a,b)$ and $Protein(a)$ in the environment, and three species $Gene(a,b)$, $Gene(b,c)$ and $Gene(c,a)$ in the heap. As the simulation proceeds, additional species are dynamically created in the heap by instantiating the definitions with different parameters. In the general case, a new species is dynamically created for each new combination of parameters, which is potentially unbounded.

3 Simulation Algorithm

This section presents a formal definition of the stochastic π-machine (SPiM). The syntax of processes and environments in SPiM is given in Definition 1. This is a subset of the syntax of the stochastic π-calculus (Definition 10), with the additional constraint that each choice of one or more actions can only occur at the top level of a definition, as in [11]. Stochastic behaviour is incorporated into the system by associating each channel x with a corresponding interaction rate given by $\rho(x)$, and by associating each delay τ_r with a corresponding delay rate r. Each rate characterises an exponential distribution, such that the probability

$P, Q ::=$	$\mathbf{0}$	Null	$E ::=$	$\emptyset$	Empty
$\mid$	$X(\tilde{n})$	Instance	$\mid$	$E, X(\tilde{m}) = P$	Process
$\mid$	$P \mid Q$	Parallel	$\mid$	$E, X(\tilde{m}) = C$	Choice
$\mid$	$\nu x\, P$	Restriction			
			$\pi ::=$	$?x(\tilde{m})$	Input
$M ::=$	$\mathbf{0}$	Null	$\mid$	$!x(\tilde{n})$	Output
$\mid$	$\pi.P + M$	Action	$\mid$	τ_r	Delay
$C ::=$	$\nu\tilde{n}\, M$	Choice			

Definition 1. *Syntax of processes and environments in* SPiM. For convenience, D is used to denote the body of a definition, which can be a process P or a choice C. For each definition of the form $X(\tilde{m}) = D$ it is assumed that $\mathrm{fn}(D) \subseteq \tilde{m}$.

$S ::=$	$\emptyset$	Empty	$U ::=$	$\emptyset$	Empty
$\mid$	$S, r \mapsto (\mathrm{Delay}_r, a_r)$	Delay	$\mid$	$U, r \mapsto \mathrm{Delay}_r$	Delay
$\mid$	$S, x \mapsto (\mathrm{In}_x, \mathrm{Out}_x, \mathrm{Mix}_x, a_x)$	Channel	$\mid$	$U, x \mapsto (\mathrm{In}_x, \mathrm{Out}_x)$	Channel

Definition 2. *Syntax of stores and substores in* SPiM..

$V ::=$	Z, S, H	Term	$H ::=$	$\emptyset$	Empty
$I ::=$	$X(\tilde{n})$	Instance	$\mid$	$H, I \mapsto (i, U, C)$	Species

Definition 3. *Syntax of terms in* SPiM. For each mapping $I \mapsto (i, U, C)$ it is assumed that $I \equiv C$ according to Definition 12(19) and $U = \mathrm{Sub}(C)$ according to Definition 4.

$$
\begin{aligned}
\mathrm{In}_x(?x(\tilde{m}).P + M) &\triangleq 1 + \mathrm{In}_x(M) \\
\mathrm{In}_x(\pi.P + M) &\triangleq \mathrm{In}_x(M) \textbf{ if } \pi \neq ?x(\tilde{m}) \\
\mathrm{Out}_x(!x(\tilde{n}).P + M) &\triangleq 1 + \mathrm{Out}_x(M) \\
\mathrm{Out}_x(\pi.P + M) &\triangleq \mathrm{Out}_x(M) \textbf{ if } \pi \neq !x(\tilde{n}) \\
\mathrm{Delay}_r(\tau_r.P + M) &\triangleq 1 + \mathrm{Delay}_r(M) \\
\mathrm{Delay}_r(\pi.P + M) &\triangleq \mathrm{Delay}_r(M) \textbf{ if } \pi \neq \tau_r
\end{aligned}
$$

$$
\begin{aligned}
\mathrm{Sub}(\nu\tilde{n}\, M) \triangleq\ & \{x \mapsto (i, o) \mid i = \mathrm{In}_x(M) \wedge o = \mathrm{Out}_x(M) \wedge (i, o) \neq (0, 0) \wedge x \notin \tilde{n}\} \\
& \cup \{r \mapsto d \mid d = \mathrm{Delay}_r(M) \wedge d \neq 0\}
\end{aligned}
$$

Definition 4. *Creating a substore in* SPiM*, where* $\mathrm{In}_x(\mathbf{0}) = \mathrm{Out}_x(\mathbf{0}) = \mathrm{Delay}_r(\mathbf{0}) = 0$.

of a reaction with rate r occurring within time t is given by $F(t) = 1 - e^{-rt}$. The average duration of the reaction is given by the mean $1/r$ of this distribution.

The syntax of machine terms is given in Definitions 2 and 3. A machine term V consists of a set of private channels Z, a store S and a heap H. The store S records the activity and apparent rate of all the unguarded delays and channels in the system. The activity of a delay with rate r is given by Delay_r, which records the total number of delays of rate r. The apparent rate a_r of the delay is equal to $r \times \mathrm{Delay}_r$. The activity of a channel x is given by the triple $\mathrm{In}_x, \mathrm{Out}_x, \mathrm{Mix}_x$, which records the total number of inputs, outputs and mixed interactions on x, respectively, where the number of mixed interactions denotes the number of pairs of inputs and outputs that cannot interact on x. The apparent rate a_x of the channel is equal to $\rho(x) \times (\mathrm{In}_x \times \mathrm{Out}_x - \mathrm{Mix}_x)$. The heap H keeps track of the number of copies of identical species in the system, and consists of zero or more mappings from species I to triples (i, U, C), where the number i records the population of the species, the choice C records the actions that the species can perform, and the substore U records the number of inputs and outputs on each channel in C, together with the number of each type of delay in C. The notation $H(I)$ denotes the values associated to I in the heap H, as usual. A substore U is created from a choice C according to Definition 4.

The system is executed according to the reduction rules of the stochastic π-machine, described in Definition 9. The rules rely on a number of auxiliary functions, given in Definitions 5-8. The expression $S \oplus U$ adds a substore U to a store S, as described in Definition 5. This is used to update the store each time the population of a species changes during a simulation. The number of delays, inputs and outputs in the species are added to the totals in the store, where the number of mixed interactions is calculated incrementally. Subtraction $S \ominus U$ is defined in a similar way.

The expression $(Z, S, H) \oplus \{I \mapsto C\}$ adds a species I with body C to a term (Z, S, H), as described in Definition 6. If a binding (i, U, C) for I is already present in the heap then the population i of the species is incremented (1). Otherwise, a new binding (i, U, C) for I is created, where the substore U denotes the total activity of the species, given by $\mathrm{Sub}(C)$, and the population of the species is set to 1 (2). Note that whenever a new species is added to a term, the substore U of the species needs to be added to the store S. The expression $(Z, S, H) \ominus \{I \mapsto C\}$ removes a species I with body C from a term (Z, S, H) (3).

The expression $V \oplus P$ adds a machine process P to a machine term V, as described in Definition 7. The null process $\mathbf{0}$ is discarded (4). If an instance $X(\tilde{n})$ is defined as a choice $X(\tilde{m}) = C$ then the term is updated with a mapping from $X(\tilde{n})$ to the body of the definition, in which the parameters $\tilde{m}$ are instantiated with the values $\tilde{n}$ (5). If an instance $X(\tilde{n})$ is defined as a process $X(\tilde{m}) = P$ then the body of the definition is added to the term, in which the parameters $\tilde{m}$ are instantiated with the values $\tilde{n}$ (6). A parallel composition $P \mid Q$ is split so that each process is added separately (7). A restriction $\nu x\, P$ is added to a term by replacing x with a fresh channel y and adding this to the set of private channels Z (8).

$$
\begin{aligned}
S \oplus \emptyset &\triangleq S \\
S \oplus (U, r \mapsto d) &\triangleq (S \oplus U), r \mapsto (d, (d \times r)) \text{ if } S(r) = \emptyset \\
(S, r \mapsto (d, a)) \oplus (U, r \mapsto d') &\triangleq (S \oplus U), r \mapsto (d + d', a + d' \times r) \\
S \oplus (U, x \mapsto (i, o)) &\triangleq (S \oplus U), x \mapsto (i, o, i \times o, 0) \text{ if } S(x) = \emptyset \\
(S, x \mapsto (i, o, m, a)) \oplus (U, x \mapsto (i', o')) &\triangleq (S \oplus U), x \mapsto (i + i', o + o', m + i' \times o', a') \\
&\textbf{if } a' = a + (i \times o' + i' \times o) \times \rho(x)
\end{aligned}
$$

Definition 5. *Adding a substore to a store in* SPiM.

$$
\begin{aligned}
(Z, S, H) \oplus \{I \mapsto C\} &\triangleq Z, (S \oplus U), H\{I \mapsto (i{+}1, U, C)\} \textbf{ if } H(I) = (i, U, C) && (1)\\
(Z, S, H) \oplus \{I \mapsto C\} &\triangleq Z, (S \oplus U), H\{I \mapsto (1, U, C)\} \textbf{ if } H(I) = \emptyset, U = \mathrm{Sub}(C) && (2)\\
(Z, S, H) \ominus \{I \mapsto C\} &\triangleq Z, (S \ominus U), H\{I \mapsto (i{-}1, U, C)\} \textbf{ if } H(I) = (i, U, C), i{>}0 && (3)
\end{aligned}
$$

Definition 6. *Adding and removing a species from a term in* SPiM.

$$
\begin{aligned}
V \oplus \mathbf{0} &\triangleq V && (4)\\
V \oplus X(\tilde{n}) &\triangleq V \oplus \{X(\tilde{n}) \mapsto C_{\{\tilde{n}/\tilde{m}\}}\} \textbf{ if } X(\tilde{m}){=}C && (5)\\
V \oplus X(\tilde{n}) &\triangleq V \oplus P_{\{\tilde{n}/\tilde{m}\}} \textbf{ if } X(\tilde{m}){=}P && (6)\\
V \oplus (P \mid Q) &\triangleq V \oplus P \oplus Q && (7)\\
(Z, S, H) \oplus (\nu x\, P) &\triangleq (Z \cup \{y\}, S, H) \oplus P_{\{y/x\}} \textbf{ if } y \text{ fresh} && (8)
\end{aligned}
$$

Definition 7. *Adding a process to a term in* SPiM.

1. Calculate $a_0 = \sum_{i=1}^{N} a_i$ for all the reactions $\theta_1, \ldots, \theta_N$ in the domain of S
2. Generate two random numbers $n_1, n_2 \in [0, 1]$ and calculate t, μ such that:

$$t = (1/a_0) \ln(1/n_1)$$

$$\sum_{i=1}^{\mu-1} a_i < n_2 a_0 \leq \sum_{i=1}^{\mu} a_i$$

3. $Gillespie(Z, S, H) = \theta_\mu, t$

Definition 8. *Choosing the next reaction in* SPiM *using the Gillespie algorithm* [5].

$$\frac{\begin{array}{l} r, t = Gillespie(V) \\ V' = V \ominus \{I \mapsto \nu\tilde{m}\,(\tau_r.P + M)\} \end{array}}{V \xrightarrow{r} V' \oplus (\nu\tilde{m}\,P)} \qquad (9)$$

$$\frac{\begin{array}{l} x, t = Gillespie(V) \\ V' = V \ominus \{I_1 \mapsto \nu\tilde{m}_1\,(!x(\tilde{n}).P_1 + M_1)\} \ominus \{I_2 \mapsto \nu\tilde{m}_2\,(?x(\tilde{m}).P_2 + M_2)\} \end{array}}{V \xrightarrow{\rho(x)} V' \oplus \nu\tilde{m}_1\,\nu\tilde{m}_2\,(P_1 \mid P_{2\,\{\tilde{n}/\tilde{m}\}})} \qquad (10)$$

$x \notin \tilde{m}_1 \cup \tilde{m}_2$
$\tilde{n} \cap \tilde{m}_2 = \emptyset$
$\tilde{m}_1 \cap \tilde{m}_2 = \emptyset$

Definition 9. *Reduction in* SPiM.

The expression $Gillespie(V)$ chooses the next channel or delay on which to perform a reaction and calculates the duration t of the reaction, as described in Definition 8.

Finally, the expression $V \xrightarrow{r} V'$ simulates a single reaction for a machine term V and produces an updated machine term V', as described in Definition 9. The rate of the reaction is given by r, and the simulation time is incremented by the reaction time t. If a delay with rate r has been chosen from a term V by the Gillespie algorithm, and if the term contains a species with a delay $\tau_r.P$, the term can perform a reaction with rate r and then execute the process $\nu\tilde{m}\, P$ (9). If an interaction on channel x has been chosen from a term V by the Gillespie algorithm, and if the term contains a species with an output $!x(\tilde{n}).P_1$, together with a species with a corresponding input $?x(\tilde{m}).P_2$ then the input and output can interact on channel x with rate $\rho(x)$ and evolve to the process $\nu\tilde{m}_1\, \nu\tilde{m}_2\, (P_1 \mid P_{2\,\{\tilde{n}/\tilde{m}\}})$, where the value $\tilde{n}$ is bound to $\tilde{m}$ in P_2 (10).

4 Correctness

This section outlines a proof of correctness of the stochastic π-machine with respect to the stochastic π-calculus. Once the main technical lemmas and definitions have been formulated, the proofs themselves are relatively direct. .

The syntax of the stochastic π-calculus (SPi) is given in Definition 10, and is identical to the syntax described in [11]. The reduction rules of the calculus are given in Definition 11. In the general case, each rule is of the form $E \vdash P \xrightarrow{r} E \vdash P'$, which states that a system $E \vdash P$ can reduce to a system $E \vdash P'$ by doing a reaction with rate r. Since the environment E remains constant over time, the rules can be abbreviated to the form $P \xrightarrow{r} P'$. The reduction rules rely on a structural congruence relation, given in Definition 12, which defines a notion of equality on processes.

In this setting, the probability of performing the reaction $P \xrightarrow{r} P'$ is given by $r/R(P)$, where $R(P)$ denotes the apparent rate of P. This corresponds to the sum of the rates of all the reactions in P, and is defined as

$$R(P) \triangleq \sum_{\theta \in P} R(\theta, P) \tag{38}$$

for all the delays and channels in P, where θ can be a delay r or a channel x. By definition, $R(\theta, P)$ is the apparent rate of θ in process P, as described in Definition 13. The apparent rate of a given channel x is equal to the number of possible combinations of inputs and outputs on x, multiplied by the rate of x (34). The functions $\mathrm{In}_x(P)$ and $\mathrm{Out}_x(P)$ return the number of unguarded inputs and outputs on channel x in P, respectively, while $\mathrm{Mix}_x(P)$ returns the sum of $\mathrm{In}_x(M_i) \times \mathrm{Out}_x(M_i)$ for each choice M_i in P. The definition of apparent rate takes into account the fact that an input and an output in the same choice cannot interact, by subtracting $\mathrm{Mix}_x(P)$ from the product of the number of inputs and outputs on x. The apparent rate of a delay r is equal to the rate of the delay times the number of unguarded delays of rate r in P, written $\mathrm{Delay}_r(P)$ (35).

$P, Q ::=$	M	Choice	$E ::=$	$\emptyset$	Empty
$\mid$	$X(\tilde{n})$	Instance	$\mid$	$E, X(\tilde{m}) = P$	Definition
$\mid$	$P \mid Q$	Parallel			$\text{fn}(P) \subseteq \tilde{m}$
$\mid$	$\nu x\, P$	Restriction			
			$\pi ::=$	$?x(\tilde{m})$	Input
$M ::=$	$\mathbf{0}$	Null	$\mid$	$!x(\tilde{n})$	Output
$\mid$	$\pi.P + M$	Action	$\mid$	τ_r	Delay

Definition 10. *Syntax of* SPi*, as defined in* [11].

$$\tau_r.P + M \xrightarrow{r} P \tag{11}$$

$$!x(\tilde{n}).P + M \mid ?x(\tilde{m}).Q + N \xrightarrow{\rho(x)} P \mid Q_{\{\tilde{n}/\tilde{m}\}} \tag{12}$$

$$P \xrightarrow{r} P' \quad \Rightarrow \quad \nu x\, P \xrightarrow{r} \nu x\, P' \tag{13}$$

$$P \xrightarrow{r} P' \quad \Rightarrow \quad P \mid Q \xrightarrow{r} P' \mid Q \tag{14}$$

$$Q \equiv P \xrightarrow{r} P' \equiv Q' \quad \Rightarrow \quad Q \xrightarrow{r} Q' \tag{15}$$

Definition 11. *Reduction in* SPi.

$$P \mid \mathbf{0} \equiv P \tag{16}$$

$$P \mid Q \equiv Q \mid P \tag{17}$$

$$P \mid (Q \mid R) \equiv (P \mid Q) \mid R \tag{18}$$

$$X(\tilde{n}) \equiv P_{\{\tilde{n}/\tilde{m}\}} \textbf{ if } X(\tilde{m}) = P \tag{19}$$

$$\nu x\, \mathbf{0} \equiv \mathbf{0} \tag{20}$$

$$\nu x\, \nu y\, P \equiv \nu y\, \nu x\, P \tag{21}$$

$$\nu x\, (P \mid Q) \equiv P \mid \nu x\, Q \textbf{ if } x \notin \text{fn}(P) \tag{22}$$

Definition 12. *Structural Congruence Axioms in* SPi. Structural congruence is defined as the least congruence that satisfies these axioms. Processes in SPi are also equal up to renaming of bound names and reordering of terms in a choice, as in [8].

The apparent rate $R(V)$ of a machine term V can be defined in a similar fashion, where $R(\theta, V)$ denotes the apparent rate of θ in term V, as described in Definition 14. The apparent rate of an unrestricted channel x is equal to the apparent rate of x in the heap (36). The apparent rate of a delay r is equal to the apparent rate of r in the heap, plus the apparent rates of all the restricted channels of rate r (37). The apparent rate of a restricted channel is recorded as a delay in order to ensure that the machine preserves the compositionality properties of the calculus. This is useful in cases where multiple machines are executed in parallel, for example in distributed or multi-core systems. In this setting, the restricted channels of a given machine are not be visible outside the scope of the machine, and interactions on these channels appear externally as delays.

The function $(\!|E \vdash P|\!)$ encodes a system $E \vdash P$ in SPi to a corresponding system in SPiM, as described in Definition 17. A corresponding decoding from the stochastic π-machine to the stochastic π-calculus is described in Definition 18.

$$\mathrm{In}_x(\nu x\, P) \triangleq 0 \tag{23}$$
$$\mathrm{In}_x(\nu y\, P) \triangleq \mathrm{In}_x(P) \textbf{ if } x \neq y \tag{24}$$
$$\mathrm{In}_x(P \mid Q) \triangleq \mathrm{In}_x(P) + \mathrm{In}_x(Q) \tag{25}$$
$$\mathrm{In}_x(X(\tilde{n})) \triangleq \mathrm{In}_x(P_{\{\tilde{n}/\tilde{m}\}}) \textbf{ if } X(\tilde{m}) = P \tag{26}$$

$$\mathrm{Delay}_r(\nu x\, P) \triangleq \mathrm{Delay}_r(P) \textbf{ if } \rho(x) \neq r \tag{27}$$
$$\mathrm{Delay}_r(\nu x\, P) \triangleq \mathrm{Delay}_r(P) + \mathrm{Act}_x(P) \textbf{ if } \rho(x) = r \tag{28}$$
$$\mathrm{Delay}_r(P \mid Q) \triangleq \mathrm{Delay}_r(P) + \mathrm{Delay}_r(Q) \tag{29}$$
$$\mathrm{Delay}_r(X(\tilde{n})) \triangleq \mathrm{Delay}_r(P_{\{\tilde{n}/\tilde{m}\}}) \textbf{ if } X(\tilde{m}) = P \tag{30}$$

$$\mathrm{Mix}_x(M) \triangleq \mathrm{In}_x(M) \times \mathrm{Out}_x(M) \tag{31}$$
$$\mathrm{Act}_x(P) \triangleq \mathrm{In}_x(P) \times \mathrm{Out}_x(P) - \mathrm{Mix}_x(P) \tag{32}$$
$$\tag{33}$$
$$R(x, P) \triangleq \rho(x) \times (\mathrm{Act}_x(P)) \tag{34}$$
$$R(r, P) \triangleq r \times \mathrm{Delay}_r(P) \tag{35}$$

Definition 13. *Apparent Rate in* SPi *based on [11].* The definitions of $\mathrm{In}_x(P)$, $\mathrm{Out}_x(P)$ and $\mathrm{Delay}_x(P)$ are given for processes P in SPi and extend the definitions of $\mathrm{In}_x(M)$, $\mathrm{Out}_x(M)$ and $\mathrm{Delay}_r(M)$ from Definition 4. The definitions of $\mathrm{Mix}_x(P)$ and $\mathrm{Out}_x(P)$ are similar to that of $\mathrm{In}_x(P)$ and are omitted.

$$R(x, (Z, H, S)) \triangleq a \textbf{ if } H(x) = (i, o, m, a) \textbf{ and } x \notin Z \tag{36}$$
$$R(r, (Z, H, S)) \triangleq a + \Sigma_i a_i \textbf{ if } H(r) = (d, a) \textbf{ and } x_i \in Z \textbf{ and } \rho(x_i) = r \textbf{ and } H(x_i) = (j_i, o_i, m_i, a_i) \tag{37}$$

Definition 14. *Apparent Rate in* SPiM.

Theorem 1 ensures that the terms of the stochastic π-machine are closed under reduction.

Theorem 1. $\forall E, V \in \mathrm{SPiM}.\ E \vdash V \xrightarrow{r} E \vdash V' \Rightarrow E \vdash V' \in \mathrm{SPiM}$

Proof. By induction on the derivation of reduction in SPiM □

Theorem 2 and Theorem 3 ensure that the stochastic π-calculus and the stochastic π-machine are reduction equivalent.

Theorem 2. $\forall E, V \in \mathrm{SPiM}.\ E \vdash V \xrightarrow{r} E \vdash V' \Rightarrow [\![E \vdash V]\!] \xrightarrow{r} [\![E \vdash V']\!]$

Proof. By induction on the derivation of reduction in SPiM □

Theorem 3. $\forall E, P \in \mathrm{SPi}.\ E \vdash P \xrightarrow{r} E \vdash P' \Rightarrow (\!|E \vdash P|\!) \xrightarrow{r} \equiv (\!|E \vdash P'|\!)$

$$(\!|\emptyset|\!) \triangleq \emptyset \tag{39}$$
$$(\!|E, X(\tilde{m}) = D|\!) \triangleq (\!|E|\!) \cup (\!|X(\tilde{m}) = D|\!) \tag{40}$$
$$(\!|X(\tilde{m}) = \nu\tilde{n}\textstyle\sum_{i=1}^{N}\pi_i.P_i|\!) \triangleq \bigcup_{i=1}^{N} E_i, X(\tilde{m}) = \nu\tilde{n}\textstyle\sum_{i=1}^{N}\pi_i.P_i' \textbf{ if } E_i \vdash P_i' = (\!|P_i|\!) \tag{41}$$
$$(\!|X(\tilde{m}) = P|\!) \triangleq E', X(\tilde{m}) = P' \textbf{ if } E' \vdash P' = (\!|P|\!) \text{ and } P \neq C \tag{42}$$

Definition 15. *Encoding an environment from* SPi *to* SPiM*, based on* [11]. The notation $\sum_{i=1}^{N}\pi_i.P_i$ is an abbreviation for a choice between zero or more actions $\pi_1.P_1 + \ldots + \pi_N.P_N + \mathbf{0}$.

$$(\!|\mathbf{0}|\!) \triangleq \emptyset \vdash \mathbf{0} \tag{43}$$
$$(\!|\nu\tilde{n}\, M|\!) \triangleq (\!|X(\tilde{m}) = \nu\tilde{n}\, M|\!) \vdash X(\tilde{m}) \textbf{ if } \tilde{m} = \mathrm{fn}(\nu\tilde{n}\, M) \textbf{ and } M \neq \mathbf{0} \textbf{ and} X \text{ fresh} \tag{44}$$
$$(\!|X(\tilde{n})|\!) \triangleq \emptyset \vdash X(\tilde{n}) \tag{45}$$
$$(\!|P_1 \mid P_2|\!) \triangleq E_1 \cup E_2 \vdash P_1' \mid P_2' \textbf{ if } E_1 \vdash P_1' = (\!|P_1|\!) \textbf{ and } E_2 \vdash P_2' = (\!|P_2|\!) \tag{46}$$
$$(\!|\nu x\, P|\!) \triangleq E \vdash \nu x\, P' \textbf{ if } E \vdash P' = (\!|P|\!) \textbf{ and } P \neq \nu\tilde{n}\, M \tag{47}$$

Definition 16. *Encoding a process from* SPi *to* SPiM*, based on* [11].

$$(\!|E \vdash P|\!) \triangleq (\!|E|\!) \cup E' \vdash (\emptyset, \emptyset, \emptyset) \oplus P' \textbf{ if } E' \vdash P' = (\!|P|\!) \tag{48}$$

Definition 17. *Encoding a system from* SPi *to* SPiM.

$$[\![E \vdash V]\!] \triangleq E \vdash [\![V]\!] \tag{49}$$
$$[\![Z, S, H]\!] \triangleq \nu Z\, [\![H]\!] \tag{50}$$
$$[\![\emptyset]\!] \triangleq \mathbf{0} \tag{51}$$
$$[\![H, X(\tilde{n}) \mapsto (i, U, C)]\!] \triangleq \underbrace{X(\tilde{n}) \mid \ldots \mid X(\tilde{n})}_{i} \mid [\![H]\!] \tag{52}$$

Definition 18. *Decoding a system from* SPiM *to* SPi. The environment E is unchanged (49), and for each mapping $X(\tilde{n}) \mapsto (i, U, C)$ in the heap, i copies of the instance are executed in parallel (52).

Proof. By induction on the derivation of reduction in SPi, where machine terms are structurally congruent up to renaming of definitions, garbage-collection of unused definitions and structural congruence of processes. These assumptions are necessary since the definitions created in the encoding $(\!|E \vdash P|\!)$ can have different names to those created in $(\!|E \vdash P'|\!)$. Similarly, $(\!|E \vdash P'|\!)$ can have less definitions than $(\!|E \vdash P|\!)$ after the process P has been reduced. □

Finally, Theorem 4 and Theorem 5 ensure that the apparent rate of reactions is preserved by encoding and decoding.

Theorem 4. $\forall P, \theta \in \text{SPi}. R(\theta, P) = R(\theta, (\!|P|\!))$

Proof. By induction on the derivation of encoding in SPi □

Theorem 5. $\forall V, \theta \in \text{SPiM}. R(\theta, V) = R(\theta, [\![V]\!])$

Proof. By induction on the derivation of decoding in SPiM □

5 Implementation

This section shows how the simulation algorithm of Sec. 3 can be mapped to functional program code, in order to implement a stochastic simulator. The mapping is relatively direct, indicating that the algorithm is sufficiently low-level to be readily implemented.

The processes of the stochastic π-machine are implemented as functional datatypes, as shown in Fig. 2. In addition, the environment, the store and the heap are implemented using a standard map library, where StringMap, SpeciesMap and ValueMap are maps indexed by strings X, species I and values θ, respectively, and a value can be a delay r or a channel x. A term is implemented as a triple consisting of a counter, a store and a heap. Each time a fresh channel is created, the counter is incremented and used to generate a fresh name. As a result, the term does not need to explicitly store all the private channels in the system, since the counter keeps track of all the channels that have been created, thereby preventing name clashes.

The implementation of reduction is also described in Fig. 2. The function *reduce* is based on Definition 9 while the function *add* is based on Definition 7. The function *remove* is based on equation (3) of Definition 6, and is implemented so that each delay of a given rate r or interaction on a given channel x has an equal probability of being selected, once a particular delay rate or interaction channel has been chosen by the Gillespie algorithm. The new simulator has been tested on the full range of examples available from [9], in most cases with significant improvement in efficiency. For instance, the example from Sec. 2 with 100 copies of each gene and simulation time 200000 took 8 minutes in the previous version of SPiM, but just 10 seconds in the optimised version (compared with 21 minutes in the BioSPI simulator). As a first step this paper focuses on reducing the algorithmic complexity of the simulation algorithm, rather than optimising the final implementation.

In the previous version of the simulator, a separate process is created for each gene or protein in the system, as described in [10]. In terms of efficiency, this is analogous to defining the heap H of a machine term as a list of choices $C_1 :: \ldots :: C_N$ instead of a mapping $I_1 \mapsto (i_1, U_1, C_1), \ldots, I_M \mapsto (i_M, U_M, C_M)$ to keep track of the number of copies of each choice. In both versions the cost of computing the Gillespie algorithm to choose the next reaction is unchanged. However, in the previous version the cost of finding a choice to execute a reaction is $O(N)$, where N is the number of choices, while the cost of inserting a choice is constant, since choices are inserted at the head of the list. In the new version the cost of finding a choice to execute a reaction is $O(M)$, where M is the number of species, while the cost of insertion is $O(\log\ M)$, assuming that the heap is a balanced tree. In addition, the new version pre-computes the number of inputs, outputs and delays for each species in the substore U, so that the store S can be

```
type action =
    Input of value * pattern
  | Output of value * value
  | Delay of value

type process =
    Null
  | Instance of string * value
  | Parallel of process * process
  | New of value * process

type choice =
  value * ((action * process) list)

type definition =
    Process of process
  | Choice of choice

type env = (pattern * definition) StringMap.t
type record = (int * int * int * int * float)
type store = record ValueMap.t
type heap = (int * store * choice) SpeciesMap.t
type term = int * store * heap

let reduce (e:env) (t:term) = match gillespie t with
    None -> None
  | Some(Rate(r),time) -> ( match remove (Delay(r)) t with
        Some(m,Delay(r),p,t') -> Some(time,add e (New(m,p)) t')
      | _ -> None )
  | Some(Channel(x),time) -> match remove (Input(x,m0)) t with
        Some(m1,Input(x,m),p1,t') -> ( match remove (Output(x,v0)) t' with
              Some(m2,Output(x,n),p2,t') ->
                let p2 = bind (eval n) m p2
                in Some(time,add e (New(m1,New(m2,Parallel(p1,p2)))) t')
            | _ -> None )
      | _ -> None
```

Fig. 2. Implementing SPiM in OCaml

quickly updated whenever there is a change in the species population. Inferring a species name for each choice also allows various additional optimisations to be implemented. For example, the current version of SPiM keeps a lookup table inside U to track which species can input and output on which channels, allowing the cost of finding a species to be further reduced to a lookup $O(\log\ M)$. In situations where each species has a population of 1 there will be little improvement, apart from not having to re-compute the number delays, inputs and outputs for each species. However, in situations where the population of species is large, which is very common in a biological setting, there will be significant improvement. For example, in the system of Fig. 1 there are generally thousands of copies of a given protein.

6 Conclusions

This paper presented a simulation algorithm for the stochastic π-calculus, designed for the efficient simulation of biological systems with large numbers of

molecules. The algorithm was proved correct with respect to the calculus, and then used as the basis for implementing an efficient simulator. To our knowledge, this is the first provably correct simulation algorithm for the stochastic π-calculus to formalise such optimisations.

Previous simulators for the stochastic π-calculus include the BioSPI simulator [1], the StoPi simulator [13], and an earlier version of the SPiM simulator [9]. The main difference with the current work is that these simulators do not formally describe an algorithm for keeping track of identical processes. The simulation algorithm of [10] was proved correct with respect to a variant of the stochastic π-calculus, and then mapped to executable program code in order to implement a stochastic simulator. This paper uses similar techniques, applied to a more efficient algorithm. In more recent work [11], a graphical variant of the stochastic π-calculus was presented, together with a corresponding graphical execution model. The graphical calculus required each choice to be associated with a corresponding identifier, so that it could be traced during execution. This paper uses similar syntactic constraints, allowing a graphical representation to be generated after each reaction as shown in Fig. 1, by adapting the graph generation algorithm of [11].

There are a number of improvement in this paper with respect to the original algorithm in [10]. In addition to the syntactic constraint placed on calculus processes, choices are dynamically grouped into species during execution according to their identifier and associated parameters. The algorithm also introduces store and substore data structures, which keep track of all the possible reactions in the heap and in the individual species, respectively. The corresponding correctness proof takes into account these extensions by checking that the store and substore data structures remain consistent with the heap, and by ensuring that the syntactic constraints do not introduce simulation errors.

There are a number of areas of future work. In the short term, the prototype simulator presented in this paper will form the basis of the next release of the Stochastic Pi Machine, available from [9]. The algorithm presented in this paper is also being extended in order to efficiently handle the dynamic creation of complexes during a simulation. Preliminary results indicate that a suitable extension can be defined with relatively few changes to the existing machine. The algorithm presented in this paper exploits the fact that biological systems typically contain large numbers of processes with identical behaviour, in contrast with most computer systems. In future, more specific optimisations for the algorithm could be investigated, such as the use of more refined data structures like priority queues, in the style of [4]. There also seems to be a close link between models that can be efficiently simulated, and those that are amenable to formal analysis, since the size of the model needs to be reduced in both cases. More generally, the simulation algorithm presented in this paper has a broader scope beyond the stochastic π-calculus, and could in principle be applied to a range of name-passing process calculi for biological modelling such as [14], in order to develop efficient simulators that are provably correct.

References

1. Bloch, A., Haagensen, B., Hoyer, M.K., Knudsen, S.U.: The StoPi-calculus and Simulator, http://www.cs.aau.dk/bh/education.html
2. Blossey, R., Cardelli, L., Phillips, A.: A compositional approach to the stochastic dynamics of gene networks. Transactions in Computational Systems Biology 3939, 99–122 (2006)
3. Gansner, E.R., North, S.C.: An open graph visualization system and its applications to software engineering. Software-Practice and Experience , 1–5 (1999)
4. Gibson, M.A., Bruck, J.: Efficient exact stochastic simulation of chemical systems with many species and many channels. J. Phys. Chem. 104, 1876–1889 (2000)
5. Gillespie, D.T.: Exact stochastic simulation of coupled chemical reactions. J. Phys. Chem. 81(25), 2340–2361 (1977)
6. Gillespie, D.T.: Approximate accelerated stochastic simulation of chemically reacting systems. J. Chem. Phys. 115, 1716–1733 (2001)
7. Lecca, P., Priami, C.: Cell cycle control in eukaryotes: a biospi model. In: BioConcur'03. ENTCS (2003)
8. Milner, R.: Communicating and Mobile Systems: the π-Calculus.
9. Phillips, A.: The Stochastic Pi-Machine (2006), Available from http://research.microsoft.com/~aphillip/spim/
10. Phillips, A., Cardelli, L.: A correct abstract machine for the stochastic pi-calculus. In: Bioconcur'04, ENTCS (August 2004)
11. Phillips, A., Cardelli, L., Castagna, G.: A graphical representation for biological processes in the stochastic pi-calculus. Transactions in Computational Systems Biology 4230, 123–152 (2006)
12. Priami, C., Regev, A., Shapiro, E., Silverman, W.: Application of a stochastic name-passing calculus to representation and simulation of molecular processes. Information Processing Letters 80, 25–31 (2001)
13. Regev, A., Silverman, W., Shapiro, E.: Representation and simulation of biochemical processes using the pi- calculus process algebra. In: Pacific Symposium on Biocomputing, vol. 6, pp. 459–470 (2001)
14. Romanel, A., Dematte, L., Priami, C.: The Beta Workbench. Available from, http://www.cosbi.eu/Rpty_Soft_BetaWB.php
15. Tian, T., Burrage, K.: Binomial leap methods for simulating stochastic chemical kinetics. J. Chem. Phys. 121, 10356–10364 (2004)
16. David, N.: Turner. The Polymorphic Pi-Calculus: Theory and Implementation. PhD thesis, June, CST-126-96 (also published as ECS-LFCS-96-345) (1996)