Efficient 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 pi-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 compact 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 [8,13,14]. 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 [14,10,1], 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 [4]. 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 variants [3,5,16]. Like the original algorithm, these variants are defined 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 $\pi$-calculus. As a result, techniques for efficient simulation of chemical reactions cannot be directly applied to the stochastic $\pi$-calculus, but need to be adapted to account for the differences between the two formalisms [11]. Given these differences, and given the importance of efficiency in the stochastic simulation of biological models, research on efficient simulation algorithms for the stochastic $\pi$-calculus seems of interest. There has already been substantial research on efficient implementation techniques for variants of the $\pi$-calculus, in the context of programming languages for parallel computer systems [17]. 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. Another difference is that the scope of private interaction channels is often limited to a relatively small number of processes, usually to represent the formation of complexes. Such channels are typically used only once for the dissociation of a complex, and subsequently discarded. These and other differences need to be taken account in order to improve the efficiency of simulations.

This paper presents a simulation algorithm for the stochastic $\pi$-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 Section 4 outlines a proof of correctness with respect to the stochastic $\pi$-calculus. Finally, Section 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 a simulation algorithm for the stochastic $\pi$-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 described in [12]. The system consists of an environment, which contains definitions for a $\text{Gene}(a, b)$ and a $\text{Protein}(b)$, together with a top-level process, which contains three genes executing in parallel:

$$
\{ \text{Gene}(a, b) = \tau_t (\text{Gene}(a, b) \mid \text{Protein}(b)) + \tau_a \text{Gene}(a, b) \\
\text{Protein}(b) = !b.\text{Protein}(b) + \tau_d \}
$$

$$
\vdash \\
\text{Gene}(a, b) \mid \text{Gene}(b, c) \mid \text{Gene}(c, a)
$$

A $\text{Gene}(a, b)$ 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 $\text{Protein}(b)$ by doing a stochastic delay at rate $t$, 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 at rate \( u \). A \( Protein(b) \) can repeatedly do an output \( !b \) on the promoter region \( b \), or it can degrade by doing a stochastic delay at rate \( d \). According to the reduction rules of the calculus, the output \( !b \) of the protein can interact with the input \( ?b \) of a corresponding \( Gene(b, c) \), which becomes blocked as a result. The three genes in the system can mutually repress each other, since \( Gene(a, b) \) produces a protein that can block \( Gene(b, c) \), which produces a protein that can block \( Gene(c, a) \), which completes 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 \( \tau_t, \tau_u, \tau_d \) with corresponding delay rates given by \( t, u, d \). The rates are used to calculate the probability of all the reactions in the system, where a reaction can be either an interaction on a particular channel or a stochastic delay with a particular rate.

The above system is simulated by encoding it to a system of the stochastic \( \pi \)-machine, which consists of a machine environment and a machine term:

\[
\begin{align*}
\{ \text{Gene}(a, b) \} \rightarrow & \tau_t (\text{Gene}(a, b) \mid Protein(b)) + ?a.X(a, b) \\
, X(a, b) = & \tau_u \text{Gene}(a, b) \\
, Protein(b) = & !b.Protein(b) + \tau_d \\
\end{align*}
\]

\[
\vdash
\emptyset, \{ a \rightarrow (1, 0, 0, 0), b \rightarrow (1, 0, 0, 0), c \rightarrow (1, 0, 0, 0), t \rightarrow (3.3t) \},
\{ \text{Gene}(a, b) \} \rightarrow 1, \{ t \rightarrow 1, a \rightarrow (1, 0) \}, \tau_t (\text{Gene}(a, b) \mid Protein(b)) + ?a.X(a, b) \\
, \text{Gene}(b, c) \rightarrow 1, \{ t \rightarrow 1, b \rightarrow (1, 0) \}, \tau_t (\text{Gene}(b, c) \mid Protein(c)) + ?b.X(b, c) \\
, \text{Gene}(c, a) \rightarrow 1, \{ t \rightarrow 1, c \rightarrow (1, 0) \}, \tau_t (\text{Gene}(c, a) \mid Protein(a)) + ?c.X(c, a) \}
\]

The machine environment is similar to the calculus environment, with the additional constraint that each choice of one or more actions can only occur at the top level of a definition. 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. The machine term consists of a set of private channels, which is empty \( \emptyset \), a store of reactions and a heap of species. The store records the apparent rate and the activity of all the reactions in the system. Each channel is mapped to the number of inputs, outputs, mixed interactions and the apparent rate of the channel, while each delay is mapped to the number of delays and the apparent rate of the delay. Initially there is one input on each channel \( a, b, c \) and three delays of rate \( t \). The heap records state information about each species that is currently being simulated, including the population of the species, the choice of reactions that the species can perform and the number of each type of reaction. 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 of parameters represents a different 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), along the lines of [12], where a separate graph is drawn for each gene.
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). One of the genes transcribes a protein (b). After a sequence of reduction steps, two of the genes have become blocked, and the third gene has produced 100 proteins (c). The mutual repression of genes gives rise to alternate oscillation 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 $u = 0.0001$.

The stochastic $\pi$-machine chooses one of the available reactions using an adaptation of the Gillespie algorithm in [4], where the probability of a reaction is proportional to its apparent rate. Initially, the machine can do one of three delay reactions with rate $t$, where the apparent rate of the reaction is $3t$. The machine chooses one of the available delays with equal probability. Suppose the Gene($a, b$) is chosen to perform the delay. An additional Protein($b$) is produced, which gives rise to the following machine term, Fig. 1(b):

$$\emptyset, \{a \mapsto (1, 0, 0, 0), b \mapsto (1, 1, 0, \rho(b)), c \mapsto (1, 0, 0, 0), d \mapsto (1, d), t \mapsto (3, 3t)\},$$

$$\{\text{Gene}(a, b) \mapsto 1, \{t \mapsto 1, a \mapsto (1, 0)\}, \tau_1 \text{(Gene}(a, b) | \text{Protein}(b)) + ?a.X(a, b)\}$$

$$, \text{Protein}(b) \mapsto 1, \{d \mapsto 1, b \mapsto (0, 1)\}, !b.\text{Protein}(b) + \tau_d$$

$$, \text{Gene}(b, c) \mapsto 1, \{t \mapsto 1, b \mapsto (1, 0)\}, \tau_1 \text{(Gene}(b, c) | \text{Protein}(c)) + ?b.X(b, c)\}$$

$$, \text{Gene}(c, a) \mapsto 1, \{t \mapsto 1, c \mapsto (1, 0)\}, \tau_1 \text{(Gene}(c, a) | \text{Protein}(a)) + ?c.X(c, a)\}$$

The Gillespie algorithm is then used to execute the next reduction. 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...
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, Fig. 1(c):

$$\emptyset, \{a \mapsto (0, 100, 0, 0), b \mapsto (1, 0, 0, 0), c \mapsto (1, 0, 0, 0), t \mapsto (1, t), u \mapsto (2, 2u), d \mapsto (100, 100d)\},$$

$$\{X(a, b) \mapsto 1, \{u \mapsto 1\}, \tau_u.Gene(a, b)\}$$

$$, X(b, c) \mapsto 1, \{u \mapsto 1\}, \tau_u.Gene(b, c)\}$$

$$, Gene(c, a) \mapsto 1, \{t \mapsto 1, c \mapsto (1, 0)\}, \tau_t.(Gene(c, a) | Protein(a)) + ?c.\tau_u.X(c, a)\}$$

$$, Protein(a) \mapsto 100, \{d \mapsto 1, a \mapsto (0, 1)\}, !a.\text{Protein}(a) + \tau_d\}$$

This represents the first oscillation cycle in the simulation, as shown in Fig. 1. Note how the corresponding graphical representation is able to keep track of the number of copies of each protein, simply by labelling the corresponding node with its population. The resulting sequence of pictures can be used to produce a graphical animation of the simulation.

3 Simulation Algorithm

This section presents the full definition of the stochastic \(\pi\)-machine. The syntax of the stochastic \(\pi\)-machine (SPiM) is given in Definitions 1, 2 and 3.

The syntax of processes and environments in SPiM is a subset of the syntax of the stochastic \(\pi\)-calculus (SPi) given in Definition 10, with the additional constraint that each choice of actions is defined separately in the environment. 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 \(\tau_r\) with a corresponding rate \(r\). Each rate characterises an exponential distribution, such that the probability 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.

A machine term \(V\) consists of a set of private channels \(Z\), a store \(S\) and a heap \(H\). The heap keeps track of the number of copies of identical species, while the store records the activity of all the reactions in the heap.

The system is executed according to the reduction rules of the stochastic \(\pi\)-machine, described in Definition 8. The rules rely on a construction operator \(V \oplus P\), which adds a machine process \(P\) to a machine term \(V\), described in Definition 7.
### Definition 1. Syntax of processes and environments in SPiM

The syntax is a normal form for the stochastic π-calculus, in which each choice of actions can only occur at the top level of a definition, as in [12]. For convenience, $C$ is used to denote a restricted choice $\nu \tilde{n} M$ and $D$ is used to denote the body of a definition. For each definition of the form $X(\tilde{n}) = D$ it is assumed that $\ln(D) \subseteq \tilde{n}$.

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

The store records the activity and apparent rate of all the reactions in a system, while the substore records the activity of all the reactions in a single choice. The activity of a delay with rate $r$ is given by $\text{Delay}_r$, which records the total number of delays of rate $r$. The apparent rate $a_r$ of the delay is equal to $r \times \text{Delay}_r$. The activity of a channel $x$ is given by the triple $\text{In}_x, \text{Out}_x, \text{Mix}_x$, which records the total number of inputs, outputs and mixed interactions on $x$, respectively. The apparent rate $a_x$ of the channel is equal to $\rho(x) \times (\text{In}_x \times \text{Out}_x - \text{Mix}_x)$.

### Definition 3. Syntax of terms in SPiM

A 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 reactions in the heap. The heap $H$ 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 substore $U$ records the activity of the species and the choice $C$ records the actions that the species can perform. For each mapping $I \mapsto (i, U, C)$ it is assumed that $I \equiv C$ according to (19) and that $U = \text{Sub}(C)$ according to Definition 4.
Creating a substore in SPiM

\[ \text{In}_{x}(\pi \cdot P + M) = 1 + \text{In}_{x}(M) \]
\[ \text{Out}_{x}(\pi \cdot P + M) = \text{Out}_{x}(M) \] if \( \pi \neq ?x(m) \)
\[ \text{Out}_{x}(\pi \cdot P + M) = 1 + \text{Out}_{x}(M) \] if \( \pi \neq !x(n) \)
\[ \text{Delay}_{x}(\tau \cdot P + M) = 1 + \text{Delay}_{x}(M) \]
\[ \text{Delay}_{x}(\tau \cdot P + M) = \text{Delay}_{x}(M) \] if \( \pi \neq \tau \)

\[ \text{Sub}(M) = \{ \langle i, o \rangle \mid i = \text{In}_{x}(M) \land o = \text{Out}_{x}(M) \land (i, o) \neq (0, 0) \} \]
\[ \cup \{ \langle r \rightarrow d \mid d = \text{Delay}_{x}(M) \land d \neq 0 \} \]

**Definition 4.** Creating a substore in SPiM, where \( \text{In}_{x}(0) = \text{Out}_{x}(0) = \text{Delay}_{x}(0) = 0 \). The substore records the number of inputs and outputs on each channel inside a choice, together with the number of each type of delay in the choice.

\[ S \oplus \emptyset = S \]
\[ (S \oplus U, r \mapsto d) \triangleq \langle S \oplus U \rangle, r \mapsto (d, (d \times r)) \] if \( S(r) = \emptyset \)
\[ (S, r \mapsto (d, a)) \oplus (U, r \mapsto d') \triangleq \langle S \oplus U \rangle, r \mapsto (d + a + d' \times r) \]
\[ (S \oplus (U, x \mapsto (i, o)), r \mapsto (i, o, i \times o, 0)) \triangleq \langle S \oplus U \rangle, x \mapsto (i + i', o + o', m + i' \times o', a') \]

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

\[ (Z, S, H) \oplus \{ I \mapsto C \} \triangleq Z, (S \oplus U), H\{ I \mapsto (i + 1, U, C) \} \] if \( H(I) = (i, U, C) \) \hspace{1cm} (1)
\[ (Z, S, H) \oplus \{ I \mapsto C \} \triangleq Z, (S \oplus U), H\{ I \mapsto (1, U, C) \} \] if \( H(I) = 0, U = \text{Sub}(C) \) \hspace{1cm} (2)
\[ (Z, S, H) \oplus \{ I \mapsto C \} \triangleq Z, (S \oplus U), H\{ I \mapsto (i - 1, U, C) \} \] if \( H(I) = (i, U, C), i > 0 \) \hspace{1cm} (3)

**Definition 6.** Adding and removing a species from a term in SPiM. The expression \((Z, S, H) \oplus \{ I \mapsto C \}\) adds a species \( I \) with body \( C \) to a term \((Z, S, H)\). 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 \( \text{Sub}(C) \), 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 \( S \), given by \( S \oplus U \). The expression \((Z, S, H) \oplus \{ I \mapsto C \}\) removes a species \( I \) with body \( C \) from a term \((Z, S, H)\) (3).
Definition 7. Adding a process to a term in SPiM. The null process \( \emptyset \) is discarded (4). If an instance \( X(\hat{n}) \) is defined as a choice \( X(\hat{m}) = C \) then the term is updated with a mapping from \( X(\hat{n}) \) to the body of the definition, in which the parameters \( \hat{m} \) are instantiated with the values \( \hat{n} \) (5). If an instance \( X(\hat{n}) \) is defined as a process \( X(\hat{m}) = P \) then the body of the definition is added to the term, in which the parameters \( \hat{m} \) are instantiated with the values \( \hat{n} \) (6). A parallel composition \( P \parallel Q \) is split so that each process is added separately (7). A restriction \( \nu x P \) is added to a term by adding \( x \) to the set of private channels \( Z \) (8).

\[
\begin{align*}
V \oplus \emptyset & \triangleq V \\
V \oplus X(\hat{n}) & \triangleq V \oplus \{ X(\hat{n}) \rightarrow C(\hat{n}/\hat{m}) \} \text{ if } X(\hat{m}) = C \\
V \oplus X(\hat{n}) & \triangleq V \oplus P(\hat{n}/\hat{m}) \text{ if } X(\hat{m}) = P \\
V \oplus (P \parallel Q) & \triangleq V \oplus P \parallel Q \\
(Z, S, H) \oplus (\nu x P) & \triangleq (Z \cup \{ x \}, S, H) \oplus P \text{ if } x \notin Z
\end{align*}
\]

Definition 8. Reduction in SPiM. 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 \( \tau, P \), the term can perform a reaction with rate \( r \) and then execute the process \( \nu m \ P \) (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 \( !x(\hat{n}).P_1 \), together with a species with a corresponding input \( ?x(\hat{m}).P_2 \) then the input and output can interact on channel \( x \) with rate \( \rho(x) \) and evolve to the process \( \nu m_1 \nu m_2 (P_1 \mid P_2_{(\hat{n}/\hat{m})}) \), where the value \( \hat{n} \) is bound to \( \hat{m} \) in \( P_2 \) (10).

\[
\begin{align*}
r, t = \text{Gillespie}(V) \\
V \rightarrow (\nu m \ P) \oplus V'
\end{align*}
\]

\[
\begin{align*}
x \notin \tilde{m}_1 \cup \tilde{m}_2 \\
\tilde{n} \cap \tilde{m}_2 = \emptyset \\
\tilde{m}_1 \cap \tilde{m}_2 = \emptyset
\end{align*}
\]

\[
\begin{align*}x, t = \text{Gillespie}(V) \\
V = V' \oplus \{ I_1 \rightarrow \nu \tilde{m}_1 (\tau, P + M) \} \\
V \rightarrow (\nu \tilde{m} \ P) \oplus V'
\end{align*}
\]

\[
\begin{align*}
& x \notin \tilde{m}_1 \cup \tilde{m}_2 \\
& \tilde{n} \cap \tilde{m}_2 = \emptyset \\
& \tilde{m}_1 \cap \tilde{m}_2 = \emptyset
\end{align*}
\]

\[
\begin{align*}
& x, t = \text{Gillespie}(V) \\
& V = V' \oplus \{ I_1 \rightarrow \nu \tilde{m}_1 (\tau, P) + M_1 \} \oplus \{ I_2 \rightarrow \nu \tilde{m}_2 (\tau, P) + M_2 \} \\
& V \rightarrow (\nu \tilde{m} \ P) \oplus \nu \tilde{m}_1 \nu \tilde{m}_2 (P_1 \mid P_2_{(\hat{n}/\hat{m})})
\end{align*}
\]

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 \( t, \mu \) 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. \( \text{Gillespie}(Z, S, H) = \theta_\mu, t \)

Definition 9. Choosing the next reaction in SPiM using the Gillespie algorithm [5]. Once \( \theta_\mu, t \) have been calculated, the simulator increments the reaction time by \( t \).
4 Correctness

\[
P, Q ::= M \quad \text{Choice} \quad E ::= \emptyset \quad \text{Empty} \\
P | Q \quad \text{Parallel} \\
\nu x P \quad \text{Restriction} \\
X(\tilde{n}) \quad \text{Instance} \\
\pi . P + M \quad \text{Action} \\
\pi ::= ?x(\tilde{m}) \quad \text{Input} \\
!x(\tilde{n}) \quad \text{Output} \\
\tau \quad \text{Delay}
\]

\[
M ::= 0 \quad \text{Null} \\
\pi . P + M \quad \text{Action}
\]

Definition 10. Syntax of SPi, as defined in [12].

\[
\begin{align*}
\tau_r . P + M & \xrightarrow{r} P \\
!x(\tilde{n}).P + M | ?x(\tilde{m}).Q + N & \xrightarrow{\sigma(\tau)} P | Q_{\{\tilde{n}/\tilde{m}\}} \\
P & \xrightarrow{r} P' \Rightarrow \nu x P \xrightarrow{r} \nu x P' \\
P & \xrightarrow{r} P' \Rightarrow P | Q & \xrightarrow{r} P' | Q \\
Q & \equiv P \xrightarrow{r} P' \equiv Q' \Rightarrow Q & \xrightarrow{r} Q'
\end{align*}
\]

Definition 11. Reduction in SPi.

\[
\begin{align*}
P | 0 & \equiv P \quad (16) \\
\nu x 0 & \equiv 0 \quad (20) \\
P | Q & \equiv Q | P \quad (17) \\
\nu x v y P & \equiv v y \nu x P \quad (21) \\
P | (Q | R) & \equiv (P | Q) | R \quad (18) \\
\nu x (P | Q) & \equiv P | \nu x Q \text{ if } x \notin \text{fn}(P) \quad (22) \\
X(\tilde{n}) & \equiv P_{\{\tilde{n}/\tilde{m}\}} \text{ if } X(\tilde{m}) = P \quad (19)
\end{align*}
\]

Definition 12. Structural Congruence Axioms in SPi. Structural congruence is defined as the least congruence that satisfies these axioms. Processes in SPi are assumed to be equal up to renaming of bound names and reordering of terms in a choice, as in [9].

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

The syntax of the stochastic π-calculus (SPi) is given in Definition 10, and is identical to the syntax described in [12]. The reduction rules of the calculus are given in Definition 11. In the general case, each rule is of the form \( E \vdash P \xrightarrow{r} P' \), 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 \( P \xrightarrow{r} P' \). Where necessary, additional predicates are used to denote the presence of specific definitions in the environment. 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 total apparent rate of \( P \) and is given by

\[
R(P) \triangleq \sum_{\theta \in P} R(\theta, P)
\]

for all reactions \( \theta \) in \( P \), where \( \theta \) can be a channel \( x \) or a delay \( r \). By definition, \( R(\theta, P) \) is the apparent rate of reaction \( \theta \) in process \( P \), as described in Definition 13. Note that \( R(V) \) can be defined in a similar fashion for machine terms, where \( R(\theta, V) \) is the apparent rate of reaction \( \theta \) in term \( V \), as described in Definition 14.

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 \( \pi \)-machine to the stochastic \( \pi \)-calculus is described in Definition 18.

Theorem 1 ensures that the terms of the stochastic \( \pi \)-machine are closed under reduction.

**Theorem 1.** \( \forall E, V \in \text{SPiM}. E \vdash V \xrightarrow{r} E \vdash V' \Rightarrow E \vdash V' \in \text{SPiM} \)

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

Theorem 2 and Theorem 3 ensure that the stochastic \( \pi \)-calculus and the stochastic \( \pi \)-machine are reduction equivalent.

**Theorem 2.** \( \forall E, V \in \text{SPiM}. E \vdash V \xrightarrow{r} E \vdash V' \Rightarrow \{E \vdash V\} \xrightarrow{r} \{E \vdash V'\} \)

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

**Theorem 3.** \( \forall E, P \in \text{SPi}. E \vdash P \xrightarrow{r} E \vdash P' \Rightarrow \{E \vdash P\} \xrightarrow{r} \{E \vdash P'\} \)

**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.

**Proposition 1.** \( \forall P \in \text{SPi}. \forall \theta. P \equiv Q \Rightarrow R(\theta, P) = R(\theta, Q) \)

**Proof.** By induction on the derivation of structural congruence in SPi

**Theorem 4.** \( \forall P \in \text{SPi}. \forall \theta. R(\theta, P) = R(\theta, [P]) \wedge R(P) = R([P]) \)

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

**Theorem 5.** \( \forall E, V \in \text{SPiM}. \forall \theta. R(\theta, V) = R(\theta, [V]) \wedge R(V) = R([V]) \)

**Proof.** By induction on the derivation of decoding in SPiM
\[ \text{In}_x(\nu \cdot P) \triangleq 0 \] (23)
\[ \text{In}_x(\nu y \cdot P) \triangleq \text{In}_x(P) \text{ if } x \neq y \] (24)
\[ \text{In}_x(P \cdot Q) \triangleq \text{In}_x(P) + \text{In}_x(Q) \] (25)
\[ \text{In}_x(X(\tilde{m})) \triangleq \text{In}_x(P\{\tilde{m}/\tilde{n}\}) \text{ if } X(\tilde{m}) = P \] (26)

\[ \text{Delay}_r(\nu x \cdot P) \triangleq \text{Delay}_r(P) \text{ if } \rho(x) \neq r \] (27)
\[ \text{Delay}_r(\nu x \cdot P) \triangleq \text{Delay}_r(P) + \text{Act}_x(P) \text{ if } \rho(x) = r \] (28)
\[ \text{Delay}_r(P \cdot Q) \triangleq \text{Delay}_r(P) + \text{Delay}_r(Q) \] (29)
\[ \text{Delay}_r(X(\tilde{m})) \triangleq \text{Delay}_r(P\{\tilde{m}/\tilde{n}\}) \text{ if } X(\tilde{m}) = P \] (30)

\[ \text{Mix}_x(M) \triangleq \text{In}_x(M) \times \text{Out}_x(M) \] (31)
\[ \text{Act}_x(P) \triangleq \text{In}_x(P) \times \text{Out}_x(P) - \text{Mix}_x(P) \] (32)

\[ R(x, P) \triangleq \rho(x) \times (\text{Act}_x(P)) \] (33)
\[ R(r, P) \triangleq r \times \text{Delay}_r(P) \] (34)

**Definition 13.** Apparent Rate in SPI, based on [12]. The apparent rate of an interaction on a given channel \( x \) is equal to the number of possible combinations of inputs and outputs on \( x \), multiplied by the rate of \( x \) (33). The functions \( \text{In}_x(P) \) and \( \text{Out}_x(P) \) return the number of available inputs and outputs on channel \( x \) in \( P \), respectively, while \( \text{Mix}_x(P) \) returns the sum of \( \text{In}_x(M_i) \times \text{Out}_x(M_i) \) 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 \( \text{Mix}_x(P) \) 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 available delays of rate \( r \) in \( P \), written \( \text{Delay}_r(P) \). The definition of \( \text{Out}_x(P) \) is similar to \( \text{In}_x(P) \).

\[ R(x, (Z, H, S)) \triangleq a \text{ if } H(x) = (i, o, m, a) \text{ and } x \notin Z \] (35)
\[ R(r, (Z, H, S)) \triangleq a + \sum a_i \text{ if } H(r) = (d, a) \text{ and } x_i \in Z \text{ and } \rho(x_i) = r \text{ and } H(x_i) = (j, o_1, m_1, a_i) \] (36)

**Definition 14.** Apparent Rate in SPI-M. The apparent rate of an unrestricted channel \( x \) is equal to the apparent rate of \( x \) in the heap (35). 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 \( r \).
Definition 15. Encoding an environment from $\text{SPi}$ to $\text{SPiM}$, based on [12]. The notation $\sum_{i=1}^{N} \pi_i P_i$ is an abbreviation for a choice between zero or more actions $\pi_1 P_1 + \ldots + \pi_N P_N + 0$.

\[
\begin{align*}
\emptyset & \triangleq \emptyset & (38) \\
[\emptyset, X(\tilde{n}) = D] & \triangleq [\emptyset] \cup [X(\tilde{n}) = D] & (39) \\
[X(\tilde{n}) = \nu\sum_{i=1}^{N} \pi_i P_i] & \triangleq \bigcup_{i=1}^{N} E_i, X(\tilde{n}) = \nu\sum_{i=1}^{N} \pi_i P_i & \text{if } E_i \vdash P'_i = [P] & (40) \\
[X(\tilde{n}) = P] & \triangleq E', X(\tilde{n}) = P' & \text{if } E' \vdash P' = [P] & \text{and } P \neq C & (41)
\end{align*}
\]

Definition 16. Encoding a process from $\text{SPi}$ to $\text{SPiM}$, based on [12].

\[
\begin{align*}
[\emptyset] & \triangleq \emptyset \vdash \emptyset & (42) \\
[M] & \triangleq [X(\tilde{n}) = M] \vdash X(\tilde{n}) & \text{if } \tilde{n} = \text{fn}(M) \text{ and } M \neq 0 \text{ and } X \text{ fresh} & (43) \\
[X(\tilde{n})] & \triangleq \emptyset \vdash X(\tilde{n}) & (44) \\
[P_1 | P_2] & \triangleq E_1 \cup E_2 \vdash P'_1 | P'_2 & \text{if } E_1 \vdash P'_1 = [P_1] \text{ and } E_2 \vdash P'_2 = [P_2] & (45) \\
[\nu x P] & \triangleq E \vdash \nu x P' & \text{if } E \vdash P' = [P] & (46)
\end{align*}
\]

Definition 17. Encoding a system from $\text{SPi}$ to $\text{SPiM}$, based on [12].

\[
\begin{align*}
[E \vdash P] & \triangleq [E] \cup [E'] \vdash (\emptyset, \emptyset, \emptyset) \oplus P' & \text{if } E' \vdash P' = [P] & (47)
\end{align*}
\]

Definition 18. Decoding a system from $\text{SPiM}$ to $\text{SPi}$. The environment $E$ is unchanged (48), and for each mapping $X(\tilde{n}) \mapsto (i, U, C)$ in the heap, $i$ copies of the instance are executed in parallel (51).

\[
\begin{align*}
[E \vdash V] & \triangleq E \vdash V & (48) \\
[Z, S, H] & \triangleq \nu Z[H] & (49) \\
[\emptyset] & \triangleq 0 & (50) \\
[H, X(\tilde{n}) \mapsto (i, U, C)] & \triangleq \underbrace{X(\tilde{n}) | \ldots | X(\tilde{n})}_{i} | [H] & (51)
\end{align*}
\]

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 straightforward, indicating that the algorithm is sufficiently low-level to be readily implemented.
The processes of the stochastic π-machine can be implemented as functional datatypes, as shown in Fig. 2. In addition, the environment, the store and the heap can be implemented using a standard map library, where StringMap, SpeciesMap and ValueMap are maps indexed by strings $X$, species $I$ and values $\theta$, 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 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 described in Fig. 3. The function $\text{reduce}$ is based on Definition 8 while the function $\text{add}$ is based on Definition 7. The function $\text{remove}$ is based on equation (3) of Definition 6. The new simulator has been tested on the range of examples available from [10], in many cases with significant improvement in efficiency. For instance, the example from Sec. 2 with $t = 10.0$ and simulation time 50000 took 32mins in the previous version of SPiM, but just 7 seconds in the optimised version. As the SPiM user community continues to grow, such increases in efficiency will become increasingly important.

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 stochastic π-calculus, and then used as the basis for implementing an efficient simulator. The algorithm was also shown to be suitable for generating compact graphical animations of simulations, in order to visualise system dynamics.

Previous simulators for the stochastic π-calculus include the BioSPI simulator [1], the StoPi simulator [14], and an earlier version of the SPiM simulator [10]. 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 simulation algorithm. In more recent work [12], a graphical variant of the stochastic π-calculus was presented, together with a corresponding graphical execution model. The graphical calculus constrains the syntax of the stochastic π-calculus so that each choice is defined separately in the environment. This paper uses similar syntax constraints as a starting point for the simulation algorithm.

The 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 [10]. Not only will this result in improved efficiency of the simulator, but it will also allow more compact graphical ani-
Fig. 2. Implementing the syntax of SPiM using OCaml datatypes.

```ocaml
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')
  | _ -> reduce t (* try again *)
  | _ -> None
```

Fig. 3. Implementing SPiM reduction in OCaml.

This algorithm presents the potential for simulations to be generated automatically, to help make modelling and simulation of biological systems more accessible to non computer scientists. The algorithm presented in this paper is currently 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, as outlined in Appendix A.

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 struc-
tures like priority queues, in the style of [3]. Another characteristic of biological system models that could be further exploited is the fact that restricted channels are typically shared among few processes and used a small number of times. In future, static analysis of channel scoping and interaction could be used to optimise simulations, e.g. based on existing type systems for linear channels [7].

More generally, there seems to be a close link between models that can be efficiently simulated, and those that are amenable to formal analysis. It would be interesting to investigate whether static typing of systems for the purpose of efficient simulation could also be used improve efficiency of system analysis, e.g. by reducing the state space in the context of stochastic model-checking [6].

References


A Dynamic Creation of Complexes

This appendix gives a brief outline of how the stochastic π-machine can be extended to count the number of complexes created during a simulation. Consider the π-calculus definitions of an enzyme $E$ and substrate $K$, as described in [2]:

\[
E(a) = \nu d \nu k \lambda a(d,k). (\beta d.E(a) + \beta k.E(a))
\]

\[
K(a) = ?a(d,k). (! d.K(a) + ! k.P())
\]

The enzyme $E$ can bind to the substrate $K$ by sending private channels $d$ and $k$ on channel $a$. The bound enzyme and substrate can unbind by doing a complementary input and output on channel $d$, and return to their original state. Alternatively, they can react by doing a complementary input and output on channel $k$. After the reaction, the enzyme returns to its original state, while the substrate is transformed into a product $P$.

Suppose that there are initially 100 copies of enzyme $E(a)$ and substrate $K(a)$. This is encoded by the following machine term:

\[
\{ E(a) = \nu d \nu k \lambda a(d,k). EK(a,d,k) \\
, EK(a,d,k) = \beta d.E(a) + \beta k.E(a) \\
, K(a) = ?a(d,k). KE(a,d,k) \\
, KE(a,d,k) = ! d.K(a) + ! k.P() \}
\]

\[
\emptyset, \{ a \mapsto (100, 100, 0, 1000\rho(a)) \}
\]

\[
\{ E(a) \mapsto 100, \{ a \mapsto (0, 1) \}, \nu d \nu k \lambda a(d,k). EK(a,d,k) \\
, K(a) \mapsto 100, \{ a \mapsto (1, 0) \}, ?a(d,k). KE(a,d,k) \}
\]

If one of the enzymes binds to one of the substrates, a new complex

\[
\nu k \nu d (EK(a,d,k) \mid KE(a,d,k))
\]

is added to the heap. The choice of actions is obtained by using an expansion relation to convert the parallel composition of species to a stochastic choice.

\[
\emptyset, \{ d \mapsto (1, d), k \mapsto (1, k), a \mapsto (99, 99, 0, 9801\rho(a)) \}
\]

\[
\{ E(a) \mapsto 99, \{ a \mapsto (0, 1) \}, \nu d \nu k \lambda a(d,k). EK(a,d,k) \\
, K(a) \mapsto 99, \{ a \mapsto (1, 0) \}, ?a(d,k). KE(a,d,k) \\
, \nu k \nu d (EK(a,d,k) \mid KE(a,d,k)) \mapsto 1, \{ \rho(d) \mapsto 1, \rho(k) \mapsto 1 \}, \tau_{\rho(d)}.(E(a) \mid K(a)) + \tau_{\rho(k)}.(E(a) \mid P()) \}
\]

The fact that private channels $k, d$ are shared between $EK(a,d,k)$ and $KE(a,d,k)$ denotes the formation of a complex between enzyme and substrate. For each subsequent binding that occurs, the population of the complex is incremented in the heap. For example, a term with 50 enzymes, 50 substrates and 50 complexes is represented as:
The stochastic $\pi$-machine can be extended to keep track of the number of dynamically created complexes, as follows. Assume that restrictions are only used to denote the formation of complexes. We can change the addition of a restriction $\nu x P$ to a term $V$ as

$$(Z, S, H) \oplus (\nu x P) \triangleq (Z, S, H) \oplus \{\nu x P \mapsto |\nu x P|\}$$

The rule relies on an expansion $|\nu n P|$, which converts a process to an equivalent choice of actions in the standard way, as described in [15].