

A Graphical Representation for Biological Processes in the Stochastic π -calculus

Andrew Phillips¹, Luca Cardelli¹, and Giuseppe Castagna²

¹ Microsoft Research, 7 JJ Thomson Avenue, CB3 0FB Cambridge UK
{andrew.phillips,luca}@microsoft.com

² École Normale Supérieure, 45 rue d'Ulm, 75005 Paris FRANCE
giuseppe.castagna@ens.fr

Abstract. This paper presents a graphical representation for the stochastic π -calculus, which is formalised by defining a corresponding graphical calculus. The graphical calculus is shown to be reduction equivalent to stochastic π , ensuring that the two calculi have the same expressive power. The graphical representation is used to model a couple of example biological systems, namely a bistable gene network and a mapk signalling cascade. One of the benefits of the representation is its ability to highlight the existence of cycles, which are a key feature of biological systems. Another benefit is its ability to animate interactions between system components, in order to visualise system dynamics. The graphical representation can also be used as a front end to a simulator for the stochastic π -calculus, to help make modelling and simulation of biological systems more accessible to non computer scientists.

1 Introduction

The stochastic π -calculus has been used to model and simulate a range of biological systems [9,17,19]. One of the main benefits of the calculus is its ability to model large systems incrementally, by composing simpler models of subsystems in an intuitive way [2]. Various stochastic simulators have been developed for the calculus [19,14], 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*. The calculus also facilitates mathematical analysis of systems using a range of techniques, including types, behavioural equivalences and model checking. In future, such analysis could help provide insight into some of the fundamental properties of biological systems. In spite of these benefits, the mathematical nature of the stochastic π -calculus can sometimes limit its accessibility to a wider audience. In such cases, it can be useful to present an alternative graphical representation for the calculus, to complement its textual notation. From our experience, such a representation would be particularly welcomed by experimental systems biologists.

This paper presents a graphical representation for the stochastic π -calculus, which is formalised by defining a corresponding graphical calculus. The paper

is structured as follows. Section 2 presents a variant of the stochastic π -calculus that supports internal transitions and recursive definitions, based on [18]. Section 3 presents a graphical representation for the stochastic π -calculus, and explains why additional syntax constraints are needed to define a corresponding graphical execution model. Section 4 presents the graphical stochastic π -calculus, which formalises the syntax constraints identified in Sec. 3. The graphical calculus is shown to be reduction equivalent to the stochastic π -calculus of Sec. 2, ensuring that the two calculi have the same expressive power. Section 5 uses the graphical stochastic π -calculus to model a couple of example biological systems, namely a bistable gene network [4] and a mapk signalling cascade [8]. Finally, Section 6 shows how the graphical representation can be used as a front end to a simulator for the stochastic π -calculus.

2 The Stochastic π -calculus

This section presents a variant of the stochastic π -calculus that supports internal transitions and recursive definitions, based on [18]. Recursive definitions have been argued in [19] to be a more practical programming abstraction for biological systems than the basic replication semantics of the π -calculus. This paper also shows how internal transitions labelled with a stochastic delay can provide a useful programming abstraction.

The syntax of the stochastic π -calculus ($S\pi$) used in this paper is summarised in Definition 1. A system $E \vdash P$ of the calculus consists of a process P together with a constant environment E . Each definition $X(m) = P$ in the environment maps a given identifier X to a corresponding process P , parameterised by m . Since the definitions themselves do not change over time, the environment E remains constant during execution. Stochastic behaviour is incorporated into the calculus by associating each channel x with a corresponding interaction rate given by $rate(x)$, and by associating each internal transition τ with a rate r . Each rate characterises an exponential distribution, such that the probability of an interaction with rate r occurring within time t is given by $F(t) = 1 - e^{-rt}$. The average duration of the interaction is given by the mean of this distribution, which is $1/r$. In this paper, it is assumed that all recursive calls to definitions are *guarded* inside an action prefix π . This prevents undesirable definitions such as $X = X$, or $X = Y, Y = (X \mid X)$. More precisely, it is assumed that for every infinite unfolding of definitions there are infinitely many occurrences of actions.

The execution rules for the calculus are summarised in Definition 3. In the general case, each rule is of the form $E \vdash P \xrightarrow{\alpha} E \vdash P'$, which states that a system $E \vdash P$ can reduce to a system $E \vdash P'$ by doing an interaction α . The definition of interaction labels is summarised in Definition 2. Since the environment E remains constant over time, the rules can be abbreviated to the form $P \xrightarrow{\alpha} P'$. Where necessary, additional predicates are used to denote the presence of specific definitions in the environment.

The probability of performing an interaction is determined by basic principles of chemical kinetics, and is proportional to the apparent rate of the interac-

$P, Q ::=$	$\nu x P$	Restriction	$E ::=$	$X(m) = P$	Definition, $\text{fn}(P) \subseteq m$
	$ P Q$	Parallel		$ E_1, E_2$	Union
	$ M$	Choice		$ \emptyset$	Empty
	$ X(n)$	Instance			
			$\pi ::=$	$?x(m)$	Input
$M ::=$	$\pi.P + M$	Action		$!x(n)$	Output
	$ \mathbf{0}$	Null		$ \tau_r$	Delay

Definition 1. *Syntax of $S\pi$, with processes P, Q , actions π , channels x, y and tuples m, n .* In a biological setting, each process typically describes the behaviour of a molecule, such as a gene or protein, and each action describes what a given molecule can do. A delay action τ_r describes a change in the internal structure of a molecule, such as a radioactive decay or a change in shape. Each delay is associated with a rate r that characterises an exponential distribution. In the case of radioactive decay, the rate determines the half-life of the reaction. Two molecules can interact by performing a complementary input $?x(m)$ and output $!x(n)$ on a common channel x . This can represent two proteins with complementary shapes, or two chemicals with complementary electronic configurations. In practice, reactions between more than two molecules are extremely rare, since the probability of three or more molecules interacting simultaneously is very low. Thus, the binary interaction model of the stochastic π -calculus fits well with the biological reality. Values m, n can also be sent and received during a reaction, e.g. to represent the transfer of an electron or a phosphate from one molecule to another. A choice of actions $\pi_1.P_1 + \dots + \pi_N.P_N$ represents the ability of a molecule to react in N different ways, while a parallel composition $P_1 | \dots | P_M$ represents the existence of M molecules in parallel. A definition of the form $X(m) = P$ represents a particular type of molecule X , parameterised by m . The parameters are assumed to contain all of the free names of P , written $\text{fn}(P) \subseteq m$. The definitions are recorded in a constant environment E , which is assumed to contain a single definition for each instance $X(n)$. A process P together with its constant environment E denotes a *system* in the calculus, written $E \vdash P$. Finally, a restriction $\nu x P$ is used to represent the formation of complexes between molecules, where a complex of two processes P and Q is modelled as $\nu x (P | Q)$. The restriction denotes a private channel x on which the two molecules can synchronise to split the complex.

tion [6]. The apparent rate of a delay τ_r is simply the rate r of the delay, while 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 [15]. The function $R(x, P)$ calculates the apparent rate of channel x in process P and is defined by:

$$R(x, P) = \text{rate}(x) \times (\text{In}_x(P) \times \text{Out}_x(P) - \text{Mix}_x(P)) \quad (12)$$

α	Description	$\text{fn}(\alpha)$	$\text{bn}(\alpha)$
$?x(n)$	Receive a value n on channel x	$\{x, n\}$	\emptyset
$!x(n)$	Send a value n on channel x	$\{x, n\}$	\emptyset
$!x(\nu y)$	Send a private channel y on channel x	$\{x\}$	$\{y\}$
x	Interact on channel x	$\{x\}$	\emptyset
r	Perform an action with apparent rate r	\emptyset	\emptyset

Definition 2. *Interaction labels in $S\pi$, where $\text{fn}(\alpha)$ and $\text{bn}(\alpha)$ denote the set of free and bound names in α , respectively.* Each label α denotes an interaction that a given process can perform. The labels for receive $?x(n)$, send $!x(n)$ and private send $!x(\nu y)$ are defined as in [18]. The label x denotes an interaction on channel x , where the rate of interaction depends on the number of inputs and outputs on the channel. The label keeps track of the channel name so that the rate can be re-calculated whenever new inputs or outputs are added in parallel. Finally, the label r denotes an interaction with constant apparent rate r , such as a stochastic delay or an interaction on a private channel.

$$!x(n).P + M \xrightarrow{!x(n)} P \quad (1)$$

$$?x(m).P + M \xrightarrow{?x(n)} P_{\{n/m\}} \quad (2)$$

$$\tau_r.P + M \xrightarrow{r} P \quad (3)$$

$$P \xrightarrow{!x(n)} P' \quad Q \xrightarrow{?x(n)} Q' \Rightarrow P \mid Q \xrightarrow{x} P' \mid Q' \quad (4)$$

$$n \notin \text{fn}(Q) \quad P \xrightarrow{!x(\nu n)} P' \quad Q \xrightarrow{?x(n)} Q' \Rightarrow P \mid Q \xrightarrow{x} \nu n(P' \mid Q') \quad (5)$$

$$P \xrightarrow{x} P' \Rightarrow \nu x P \xrightarrow{R(x,P)} \nu x P' \quad (6)$$

$$x \neq y \quad P \xrightarrow{!x(y)} P' \Rightarrow \nu y P \xrightarrow{!x(\nu y)} P' \quad (7)$$

$$x \notin \text{fn}(\alpha) \cup \text{bn}(\alpha) \quad P \xrightarrow{\alpha} P' \Rightarrow \nu x P \xrightarrow{\alpha} \nu x P' \quad (8)$$

$$M \xrightarrow{\alpha} P' \Rightarrow \pi.P + M \xrightarrow{\alpha} P' \quad (9)$$

$$\text{bn}(\alpha) \cap \text{fn}(Q) = \emptyset \quad P \xrightarrow{\alpha} P' \Rightarrow P \mid Q \xrightarrow{\alpha} P' \mid Q \quad (10)$$

$$X(m) = P \quad P_{\{n/m\}} \xrightarrow{\alpha} P' \Rightarrow X(n) \xrightarrow{\alpha} P' \quad (11)$$

Definition 3. *Reduction in $S\pi$.* An output $!x(n).P$ can send the value n on channel x and then execute process P (1). An input $?x(m).P$ can receive a value n on channel x and then execute process P , in which the received value is assigned to m (2). A delay $\tau_r.P$ can perform an internal action with apparent rate r and then execute the process P (3). If a process P can send a value n on channel x and a process Q can receive a value n on channel x then P and Q can interact on x (4). If n is private then the scope of n is extended over the resulting processes, where $\nu n(P' \mid Q')$ denotes the formation of a complex between P' and Q' (5). If two processes interact on a private channel x then the apparent rate of the interaction is constant, and is given by $R(x, P)$ (6). Rule (7) allows a private channel to be sent. Finally, rules (8), (9), (10) and (11) allow an action to be performed inside a restriction, a choice, a parallel composition and a definition, respectively. For each of the rules (4), (5) and (10) there exists a symmetric rule (not shown) in which $P \mid Q$ and $P' \mid Q'$ are commuted.

where $\text{In}_x(P)$ and $\text{Out}_x(P)$ are the number of unguarded inputs and outputs on channel x in P , respectively, and $\text{Mix}_x(P)$ is the sum of $\text{In}_x(M_i) \times \text{Out}_x(M_i)$ for each choice M_i in P . The definition 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 .

3 Graphical Representation

This section presents a graphical representation for the stochastic π -calculus, and explains why additional syntax constraints are needed to define a corresponding graphical execution model. The principle of the graphical representation is to display each process P as a *node* in a graph and to draw an *edge* from the node to each nested process in P . This allows the syntax tree of a given process to be represented as a tree of nodes. In addition, each definition in the environment assigns a unique *identifier* to a node. The identifiers are used to define additional edges between nodes, as in standard graph notations.

The graphical representation of the stochastic π -calculus is defined in Fig. 1 and Fig. 2, and is based on an abbreviated syntax for the calculus, presented in Definition 4. The abbreviated syntax is equivalent to the syntax of the stochastic π -calculus presented in Definition 1, but uses a more compact notation for restriction, parallel composition, choice and union. As an example, Figure 3 uses the graphical representation to visualise a stochastic π -calculus model of a gene with inhibitory control, as presented in [2].

The graphical representation described so far is essentially a static way of visualising systems of the stochastic π -calculus. The next stage is to define a dynamic representation, in order to visualise system execution. The principle of the dynamic representation is to add a token to each node in the graph that corresponds to a currently executing process. For example, in Fig. 3 a token is added to the *Gene* node to represent the execution of a single gene, and a new token is added to the *Protein* node each time a new protein is created. Similarly, a token needs to be added to the corresponding node whenever the gene becomes blocked after doing an input on a . However, in order for a token to be added, the node needs to be associated with a suitable identifier. This can be achieved by augmenting the model in Fig. 3 with the definition $\text{Blocked}(a, b) = \tau_u.Gene(a, b)$ and by replacing $?a.\tau_u.Gene(a, b)$ with $?a.\text{Blocked}(a, b)$. In the general case, each choice needs to be defined separately in the environment, so that a token can be added to the appropriate node during execution. It turns out that this simple constraint is sufficient to derive a graphical execution model for the stochastic π -calculus, as shown in the next section.

$P ::= \nu z \sum_{i=1}^N \pi_i.P_i$	Choice	$E ::= X(m)=P$	Definition
$\nu z \prod_{i=1}^M P_i$	Parallel	$\bigcup_{i=1}^N E_i$	Union
$\nu z X(n)$	Instance		

Definition 4. *Abbreviated Syntax of $S\pi$, where $N \geq 0$ and $M \geq 2$.* A sequence of zero or more restricted names is abbreviated to a tuple z , which may be empty. A choice between zero or more actions $\pi_1.P_1 + \dots + \pi_N.P_N + \mathbf{0}$ is abbreviated to a sum $\sum_{i=1}^N \pi_i.P_i$. The choice can also be written $\pi_1.P_1 + \dots + \pi_N.P_N$ if $N \geq 1$. A parallel composition of two or more processes $P_1 \mid \dots \mid P_M$ is abbreviated to a product $\prod_{i=1}^M P_i$. Finally, a union of zero or more environments E_1, \dots, E_N is abbreviated to $\bigcup_{i=1}^N E_i$. The abbreviated syntax is used as the basis for the graphical representation.

E	Definition $X(m)=P$	Union E_1, \dots, E_N
E	P	E1 ... EN

Fig. 1. Graphical representation of environments in $S\pi$. Each process in the environment represents a *node* in a graph, and each definition assigns an *identifier* to a given node. The identifiers are used to define *edges* between nodes, as in standard graph notations. A definition $X(m)=P$ is displayed as the process P , where the name X is used as an identifier for P . By convention, any edges leading to X are connected to the node of P . A union of environments E_1, \dots, E_N is displayed by drawing the environments E_1, \dots, E_N next to each other. Edges between nodes in the environments are determined by the node identifiers.

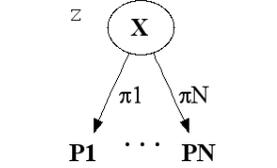
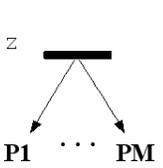
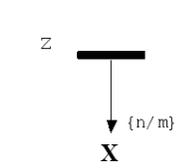
P	Choice $\nu z (\pi_1.P_1 + \dots + \pi_N.P_N)$	Parallel $\nu z (P_1 \mid \dots \mid P_M)$	Instance $X(m)=P \vdash \nu z X(n)$
P			

Fig. 2. Graphical representation of processes in $S\pi$. A choice $\pi_1.P_1 + \dots + \pi_N.P_N$ with restricted names z is displayed as an elliptical node with label z and with edges to processes P_1, \dots, P_N . Each edge to a process P_i is labelled with an action π_i and denotes an alternative execution path in the system. The node can also be annotated with an optional name X . A parallel composition $P_1 \mid \dots \mid P_M$ with restricted names z is displayed as a rectangular node with label z and with edges to processes P_1, \dots, P_M . Each edge to a process P_i denotes a concurrent execution path in the system. An instance $X(n)$ with restricted names z is displayed as a rectangular node with label z and with an edge to the process identified by X . If $X(m)=P$ and $m \neq n$ then the tip of the edge is labelled with the substitution $\{n/m\}$. This represents the passing of parameters from one process to another. If z is empty then edges from a choice or parallel composition can be connected directly to node X by omitting the rectangle.

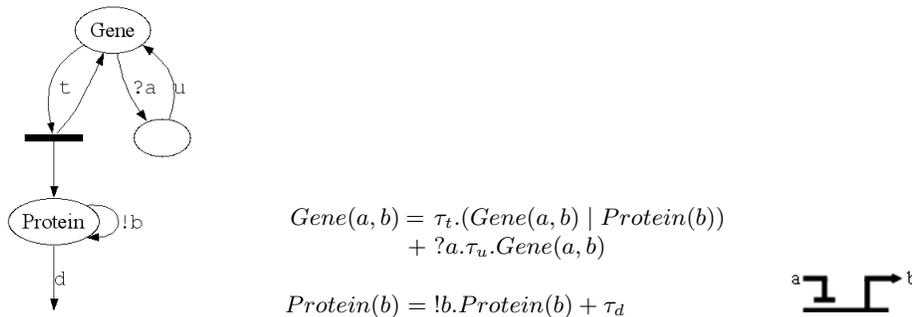


Fig. 3. A stochastic π -calculus model of a gene with inhibitory control, as presented in [2]. The gene can transcribe a protein by first doing a stochastic delay at rate t and then executing a new protein in parallel with the gene. Alternatively, it can block by doing an input on its promoter region a and then unblock by doing a stochastic delay at rate u . The transcribed protein can repeatedly do an output on the promoter region b , or it can decay at rate d . The gene is parameterised by its promoter region a , together with the promoter region b that is recognised by its transcribed proteins. The functional behaviour of the gene can be visualised using a corresponding high-level representation (right), which abstracts away from the internal dynamics. According to the reduction rules of the calculus, the output $!b$ of the transcribed protein can interact with the input $?b$ of a corresponding $Gene(b, c)$, which becomes blocked, as a result. This simple model of a gene can be used to build arbitrarily complex networks, as described in [2]. An example of one such network is presented in Sec. 4.

4 Graphical Calculus

This section presents the graphical stochastic π -calculus, which formalises the syntax constraints identified in Sec. 3. The graphical calculus is shown to be reduction equivalent to the stochastic π -calculus of Sec. 2, ensuring that the two calculi have the same expressive power. The syntax of the graphical stochastic π -calculus (GS π) is presented in Definition 5, and a corresponding abbreviated syntax is presented in Definition 6. The graphical calculus GS π is a subset of the calculus S π , with the additional constraint that each choice is defined separately in the environment. Similarly, the graphical representation of GS π is a subset of the graphical representation of S π , as shown in Fig. 4 and Fig. 5.

The graphical calculus can also be used as the basis for a graphical execution model. In this setting, a system $E \vdash P$ is displayed in two parts, a static environment E which remains constant over time, and a dynamic process P which is updated after each execution step. The environment E is displayed using the static representation of environments and processes defined in Fig. 4 and Fig. 5, whereas the process P is displayed using a dynamic representation, defined in Fig. 6. The principle of the dynamic representation is to display each instance $X(n)$ of a definition $X(m) = P$ by attaching a substitution token $\{n/m\}$ to the node identified by X . In addition, a dotted edge is drawn from each restricted

$P, Q ::= \nu x P$	Restriction	$E ::= X(m) = D$	Definition, $\text{fn}(P) \subseteq m$
$P \mid Q$	Parallel	E_1, E_2	Union
$\mathbf{0}$	Null	\emptyset	Empty
$X(n)$	Instance		
		$D ::= P$	Process
$M ::= \pi.P + M$	Action	M	Choice
$\mathbf{0}$	Null	$\nu x D$	Restriction

Definition 5. *Syntax of $\text{GS}\pi$.* This is a subset of the syntax of $\text{S}\pi$, with the additional constraint that processes in $\text{GS}\pi$ can only contain empty choices, and definitions in $\text{GS}\pi$ can only contain a choice at the top-level. The constraints ensure that each choice is defined separately in the environment.

$P ::= \nu z \mathbf{0}$	Null	$E ::= X(m) = \nu z \sum_{i=1}^N \pi_i.P_i$	Choice
$\nu z \prod_{i=1}^M P_i$	Parallel	$X(m) = P$	Process
$\nu z X(n)$	Instance	$\bigcup_{i=1}^N E_i$	Union

Definition 6. *Abbreviated syntax of $\text{GS}\pi$, where $N \geq 0$ and $M \geq 2$.* This is a subset of the abbreviated syntax of $\text{S}\pi$, and is used as the basis for the graphical representation.

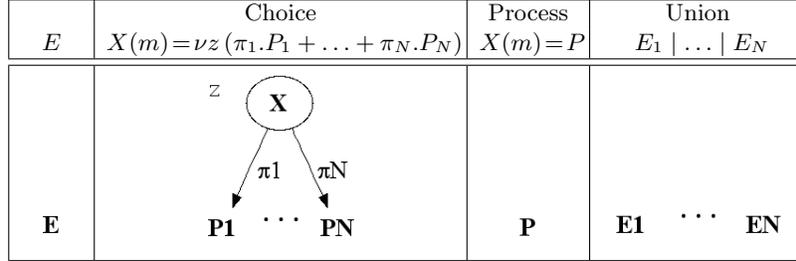


Fig. 4. Graphical representation of environments in $\text{GS}\pi$, which is a subset of the graphical representation of environments in $\text{S}\pi$. For a definition of a choice, the node can also be annotated with the name X of the definition, or with the name and parameters $X(m)$. Note that all node annotations are optional.

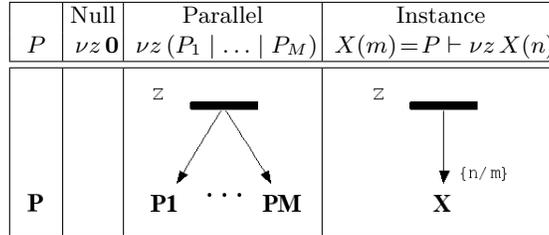


Fig. 5. Graphical representation of processes in $\text{GS}\pi$, which is a subset of the graphical representation of processes in $\text{S}\pi$.

	Null	Parallel	Instance
P	$\nu z \mathbf{0}$	$\nu z (P_1 \mid \dots \mid P_M)$	$X(m) = P \vdash \nu z X(n)$
P			

Fig. 6. Dynamic graphical representation of processes in $\text{GS}\pi$. A null process $\mathbf{0}$ with restricted names z is not displayed. An instance $X(n)$ of a definition $X(m) = P$ is displayed by placing a substitution token $\{n/m\}$ next to the node identified by X . For clarity, the node is highlighted when at least one token is present, and any restricted names z are connected to the token by a dotted edge. A collection of parallel processes $P_1 \mid \dots \mid P_M$ with restricted names z is displayed by drawing a dotted edge from z to each of the processes P_1, \dots, P_M . This represents the formation of a complex between the processes, where the names z can be used to split the complex. If P_i is an instance $X(n)$ then the dotted edge is connected to the corresponding substitution token.

name to all of the tokens that share this name, in order to represent the formation of complexes. For example, a process $\nu x (X_1(n_1) \mid X_2(n_2))$ where $X_1(m_1) = P_1$ and $X_2(m_2) = P_2$ is displayed by placing tokens $\{n_1/m_1\}$ and $\{n_2/m_2\}$ next to the nodes identified by X_1 and X_2 , respectively. A dotted edge is drawn from the name x to both tokens, in order to represent the formation of a complex between X_1 and X_2 . The resulting graph is displayed as: $X_1 \{n_1/m_1\} \cdots x \cdots \{n_2/m_2\} X_2$.

Additional syntactic sugar can be defined for the dynamic representation, in order to improve the display of processes. For example, if N identical substitution tokens are attached to the same node, they can be replaced by a single token preceded by the number N . Furthermore, if the substitution token is empty it can be omitted, leaving just the number N . Similarly, if there are N copies of a restriction $\nu x P$ they can be replaced by a single copy of the restriction, where the name x is preceded by the number N . The scope of a restricted channel can be further clarified by only drawing a dotted edge to a token if it contains a free occurrence of the channel name. For example, a restriction $\nu x_1 \nu x_2 (P_1 \mid P_2 \mid P_3)$ where $x_1 \notin \text{fn}(P_3)$ and $x_2 \notin \text{fn}(P_1)$ can be displayed as: $P_1 \cdots x_1 \cdots P_2 \cdots x_2 \cdots P_3$. The graphical representation is more informative than the corresponding textual syntax, since it clearly shows that P_1 and P_3 do not share any restricted names. In contrast, to verify this property for the textual syntax one needs to check which names occur inside which processes, and whether any of the names are shared. The extra clarity is not a particular property of the calculus, but simply a consequence of the fact that the graphical representation uses two dimensions, whereas the textual syntax uses just one.

More generally, if multiple substitution tokens of different values are attached to the same node X , a separate copy of the graph connected to X can be spawned

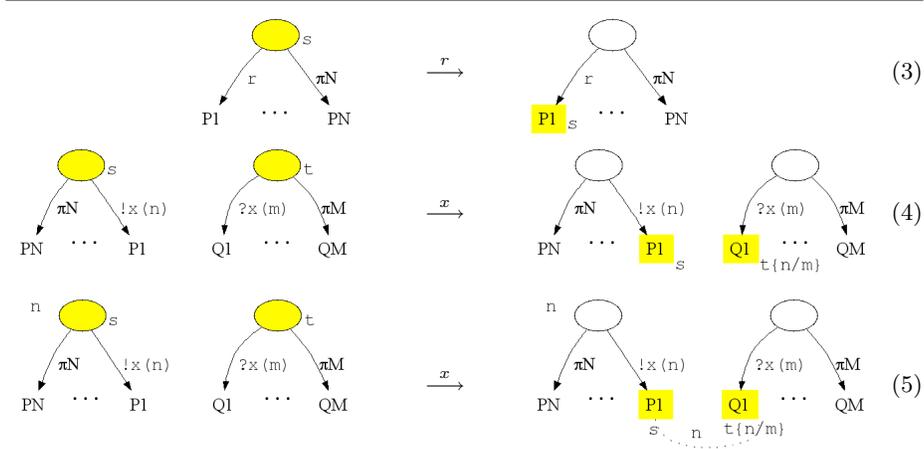
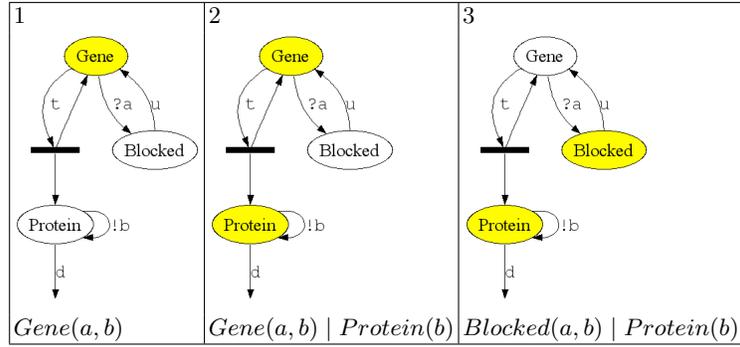


Fig. 7. Example graphical reductions in $GS\pi$, where P_1, \dots, P_N and Q_1, \dots, Q_M are assumed to represent choices. A given process is executed graphically by first applying one of the calculus reduction rules and then displaying the resulting process.

for each token of a given value. This can be used to visualise the execution of different types of components in the system, where a token of a given value corresponds to a particular type of component. At the finest level of granularity, a separate graph can be spawned for each individual token, in order to visualise the execution of individual components in the system. In this setting, only a single token is present in the graph at a given instant. This allows successive nodes in the graph to be highlighted after each reduction step, in the style of state machines.

Since the graphical calculus $GS\pi$ is a subset of the calculus $S\pi$, reduction in $GS\pi$ can be defined using the rules for $S\pi$ presented in Definition 3. The only required change is to replace P with D in rule (11), since definitions in $GS\pi$ are of the form $X(m) = D$. A graphical execution model is obtained by applying these rules to a given process in $GS\pi$, and then displaying the resulting process after each reduction step. Figure 7 illustrates a number of example graphical reductions, based on the reduction rules of Definition 3. Figures 8 and 9 illustrate the execution of a single gene and a network of genes, respectively, based on the model in Fig. 3. During execution, it is also useful to expand instances of definitions that are not choices, so that tokens are only attached to nodes that are waiting to execute. This can be achieved by defining an additional normalisation rule, such that if $X(m) = P$ then the process $X(n)$ is expanded to $P_{\{n/m\}}$, where P does not contain a choice of actions.

So far, the graphical stochastic π -calculus has been used for both the static and dynamic visualisation of calculus processes. The next stage is to prove its equivalence with respect to the stochastic π -calculus, in order to ensure that the



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

Fig. 8. Execution trace for a gene with inhibitory control, based on Fig. 3, where each choice is defined separately in the environment. The sequence of transitions is given by $1 \xrightarrow{t} 2 \xrightarrow{?a} 3 \xrightarrow{u} 2$. All substitution tokens in the graphs are empty, since the arguments of each instance are equal to the formal parameters. Parallel execution is represented by highlighting two different nodes on the same graph at the same time.

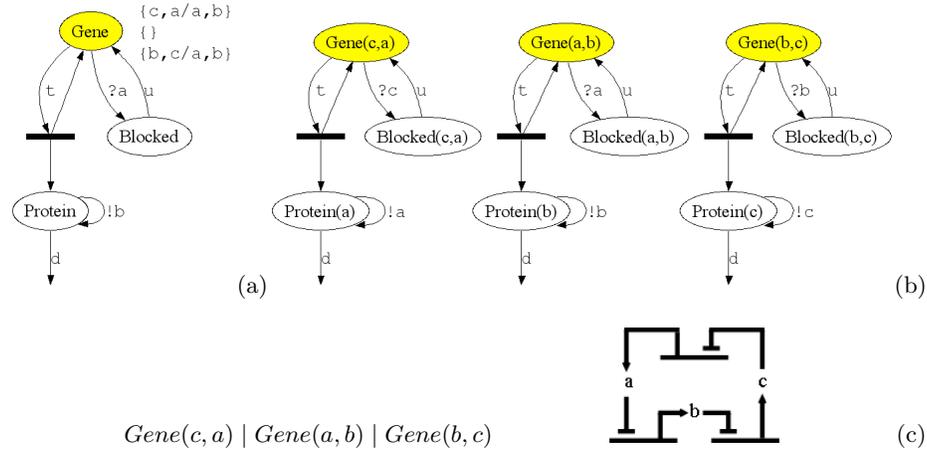


Fig. 9. Constructing a simple network using the gene of Fig. 8. The network consists of three genes that mutually repress each other, and was previously genetically engineered in living bacteria [3]. The network has been dubbed the *repressilator*, since the mutual repression of the three genes gives rise to alternate oscillations in the expression levels of the corresponding proteins. The network is displayed by adding substitution tokens $\{c, a/a, b\}$, $\{a, b/a, b\}$ and $\{b, c/a, b\}$ to the node identified by *Gene*, as shown in (a). By default, the parameters are not shown explicitly on the node label but can be optionally included, as stated in Fig. 4. For clarity, a separate graph can be generated for each token (b), where the names of the parameters are used to distinguish between the different genes. A high-level graphical representation of the network is also given (c).

$$\llbracket E \vdash P \rrbracket \triangleq \llbracket E \rrbracket, E' \vdash P' \quad (13)$$

$$\text{where } E' \vdash P' = \llbracket P \rrbracket$$

$$\llbracket \nu z \mathbf{0} \rrbracket \triangleq \emptyset \vdash \nu z \mathbf{0} \quad (14)$$

$$\llbracket \nu z M \rrbracket \triangleq \llbracket X(n) = \nu z M \rrbracket \vdash X(n) \quad (15)$$

$$\text{where } X \text{ fresh } \quad M \neq \mathbf{0} \quad n = \text{fn}(\nu z M)$$

$$\llbracket \nu z \prod_{i=1}^M P_i \rrbracket \triangleq \bigcup_{i=1}^M E_i \vdash \nu z \prod_{i=1}^M P'_i \quad (16)$$

$$\text{where } E_i \vdash P'_i = \llbracket P_i \rrbracket$$

$$\llbracket \nu z X(n) \rrbracket \triangleq \emptyset \vdash \nu z X(n) \quad (17)$$

$$\llbracket X(m) = \nu z \sum_{i=1}^N \pi_i.P_i \rrbracket \triangleq \bigcup_{i=1}^N E_i, X(m) = \nu z \sum_{i=1}^N \pi_i.P'_i \quad (18)$$

$$\text{where } E_i \vdash P'_i = \llbracket P_i \rrbracket$$

$$\llbracket X(m) = P \rrbracket \triangleq E, X(m) = P' \quad (19)$$

$$\text{where } E \vdash P' = \llbracket P \rrbracket \quad P \neq \nu z M$$

$$\llbracket \bigcup_{i=1}^N E_i \rrbracket \triangleq \bigcup_{i=1}^N \llbracket E_i \rrbracket \quad (20)$$

Definition 7. *Encoding $S\pi$ to $GS\pi$.* The function $\llbracket E \vdash P \rrbracket$ encodes a given system $E \vdash P$ in $S\pi$ to a corresponding system in $GS\pi$ (13). The encoding relies on a function $\llbracket P \rrbracket$, which encodes a process P in $S\pi$ to a process and environment in $GS\pi$ as follows. An empty choice $\nu z \mathbf{0}$ is unchanged. A fresh definition is created for each non-empty choice $\nu z M$, and the choice is replaced by an instance of this definition (15). Each process P_i in a parallel composition $\nu z \prod_{i=1}^M P_i$ is replaced by its encoding, and any new definitions are added to the environment (16). An instance $\nu z X(n)$ is unchanged. The encoding also relies on a function $\llbracket E \rrbracket$, which encodes an environment E in $S\pi$ to an environment in $GS\pi$ as follows. Each process P_i in a choice $\nu z \sum_i \pi_i.P_i$ is replaced by its encoding, and any new definitions are added to the environment (18). A process P that is not a choice is replaced by its encoding, and any new definitions are added to the environment. (19). Finally, each environment E_i in a union $\bigcup_{i=1}^N E_i$ is replaced by its encoding (20).

$$\llbracket \llbracket E \vdash P \rrbracket \rrbracket \triangleq E \vdash P \quad (21)$$

Definition 8. *Decoding $GS\pi$ to $S\pi$.* Since the graphical calculus $GS\pi$ is a subset of the calculus $S\pi$, the decoding is simply the identity function.

two calculi can be used interchangeably. This can be achieved by defining an encoding from the calculus $S\pi$ to the graphical calculus $GS\pi$, as presented in Definition 7, where the function $\llbracket E \vdash P \rrbracket$ encodes a given system $E \vdash P$ in $S\pi$ to a corresponding system in $GS\pi$.

Lemma 1 ensures that the encoding is well-defined. The lemma states that if a system $E \vdash P$ is in the calculus $S\pi$ then its encoding $\llbracket E \vdash P \rrbracket$ is in the graphical calculus $GS\pi$.

Lemma 1. $\forall E, P \in S\pi. \llbracket E \vdash P \rrbracket \in \text{GS}\pi$

Proof. By straightforward induction on the definition of encoding in $S\pi$. \square

Lemma 2 ensures that the graphical calculus $\text{GS}\pi$ is a subset of the calculus $S\pi$. The lemma states that if a system $E \vdash P$ is in the graphical calculus $\text{GS}\pi$, then it is also in the calculus $S\pi$. This allows reduction in $\text{GS}\pi$ to use the same rules as reduction in $S\pi$.

Lemma 2. $\forall E, P \in \text{GS}\pi. E \vdash P \in S\pi$

Proof. By straightforward induction on the syntax of $\text{GS}\pi$. By definition, the calculus $\text{GS}\pi$ requires each choice to be defined separately in the environment, which is nothing more than a syntactic constraint on the calculus $S\pi$. \square

Theorem 1 ensures that the syntax of the graphical calculus is preserved by reduction. The theorem states that if a system $E \vdash P$ in $\text{GS}\pi$ can reduce to $E \vdash P'$ then the resulting system is also in $\text{GS}\pi$. This ensures that a given process can be graphically displayed after each reduction step.

Theorem 1. $\forall E, P \in \text{GS}\pi. E \vdash P \xrightarrow{\alpha} E \vdash P' \Rightarrow E \vdash P' \in \text{GS}\pi$

Proof. By straightforward induction on the definition of reduction in $\text{GS}\pi$. It is clear that if each choice is defined separately in the environment then this property will also hold after a reduction, since the reduction rules do not expand definitions of choices. This is the only additional constraint that needs to be preserved with respect to reduction in $S\pi$. \square

Finally, Theorem 2 and Theorem 3 ensure that the graphical calculus $\text{GS}\pi$ and the calculus $S\pi$ are reduction equivalent. This ensures that the two calculi have the same expressive power, and can therefore be used interchangeably.

Theorem 2. $\forall E, P \in \text{GS}\pi. E \vdash P \xrightarrow{\alpha} E \vdash P' \Rightarrow \llbracket E \vdash P \rrbracket \xrightarrow{\alpha} \llbracket E \vdash P' \rrbracket$

Proof. The proof is immediate, since the graphical calculus $\text{GS}\pi$ is a subset of the calculus $S\pi$, where the decoding $\llbracket E \vdash P \rrbracket$ is given in Definition 8 as the identity function. \square

Theorem 3. $\forall E, P \in S\pi. E \vdash P \xrightarrow{\alpha} E \vdash P' \Rightarrow (E \vdash P) \xrightarrow{\alpha} (E \vdash P')$

Proof. The proof is by straightforward induction on the definition of reduction in $S\pi$. The encoding $\llbracket E \vdash P \rrbracket$ merely creates a separate definition $X(n) = \nu z M$ in the environment E for each choice $\nu z M$ in the system. Moreover, rule (11) ensures that if a given process can perform a reduction, then the same reduction can be performed if the process is defined separately in the environment. Therefore, any reductions that are possible in the system $E \vdash P$ will also be possible in the corresponding encoding. Note that the definitions created in the encoding $\llbracket E \vdash P \rrbracket$ can have different names to those created in $\llbracket E \vdash P' \rrbracket$. Furthermore, the encoding $\llbracket E \vdash P' \rrbracket$ can have less definitions than the encoding $\llbracket E \vdash P \rrbracket$, e.g. if some of the choices in $\llbracket E \vdash P \rrbracket$ are reduced. Therefore, in order to ensure that $\llbracket E \vdash P \rrbracket \xrightarrow{\alpha} \llbracket E \vdash P' \rrbracket$ the proof assumes that systems $E \vdash P$ of the graphical calculus are equal up to renaming of definitions and up to garbage-collection of unused definitions. \square

5 Biological Examples

This section uses the graphical stochastic π -calculus to model a couple of example biological systems, namely a bistable gene network [4] and a mapk signalling cascade [8]. A visual comparison between the stochastic π -calculus models and their corresponding reaction equations is provided in Appendix B.

5.1 Bistable Gene Network

In [4], a number of gene networks were evolved in silico to perform specific functions. At each stage in the evolution, various criteria were used to select the networks that best matched the desired behaviour. One of the networks was evolved to perform the function of a bistable switch, as summarised in Fig. 10. The evolved network was shown to be considerably more stable than the simpler, more intuitive network in which two genes mutually repress each other.

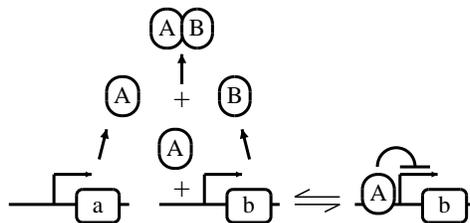
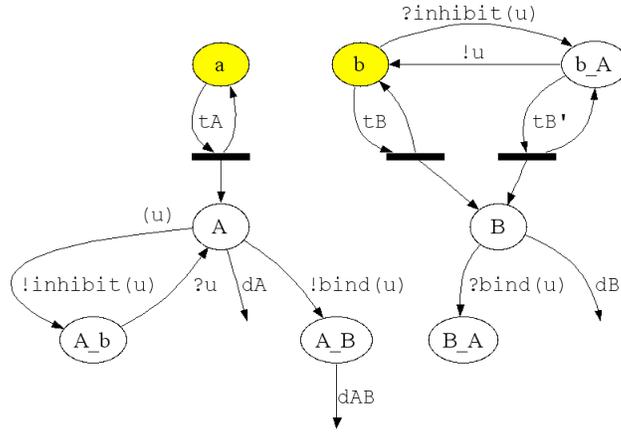


Fig. 10. A bistable gene network obtained by evolution in silico, as presented in [4]. The genes a and b can transcribe proteins A and B respectively, at constitutive transcription rates. Proteins A and B can bind irreversibly to produce the complex AB , which eventually degrades. Protein A can also bind reversibly to gene b , in order to inhibit the transcription of B . Initially, if A binds to b then production of A stabilises at a high level, since B is produced at a much lower rate. Alternatively, if A binds to B then production of B stabilises at a high level, since each subsequent A that is produced immediately binds to B and is degraded.

A graphical stochastic π -calculus model of this system is presented in Fig. 11, and the corresponding code for the model is presented in Fig. 22 of Appendix A. The model is directly executable, in contrast with the informal diagram of Fig. 10. Example execution traces for the model are shown in Fig. 12, which help to clarify the overall system function. Stochastic simulation results for the model are shown in Fig. 13, which match those presented in [4]. The results indicate that the system does indeed behave as a bistable switch.



$$\begin{aligned}
z &= \text{inhibit}, \text{bind} \\
a(z) &= \tau_{tA}.(A(z) \mid a(z)) \\
A(z) &= \nu u \\
&\quad (\tau_{dA} \\
&\quad + !\text{bind}(u).A_B(u) \\
&\quad + !\text{inhibit}(u).A_b(u, \text{inhibit}, \text{bind})) \\
A_b(u, \text{inhibit}, \text{bind}) &= ?u.A(z) \\
A_B(u) &= \tau_{dAB}
\end{aligned}$$

$$\begin{aligned}
b(z) &= \tau_{tB}.(B(z) \mid b(z)) \\
&\quad + ?\text{inhibit}(u).b_A(u) \\
b_A(u) &= \tau_{tB'}.(B(z) \mid b_A(u)) \\
&\quad + !u.b(z) \\
B(z) &= \tau_{dB} \\
&\quad + ?\text{bind}(u).B_A(u) \\
B_A(u) &= \mathbf{0}
\end{aligned}$$

$$a(z) \mid b(z)$$

Fig. 11. A graphical stochastic π -calculus model of the bistable gene network presented in [4]. The corresponding textual representation of the network is also given. Each gene a and b is modelled as a separate process with parameters z . Gene a can transcribe a protein A by doing a stochastic delay at rate tA and then executing a new process A in parallel with the gene. Protein A can either degrade by doing a stochastic delay at rate dA , or bind to gene b by doing an output on channel inhibit , or bind irreversibly to protein B by doing an output on channel bind . When protein A binds to gene b it sends a private channel u and then executes the process A_b , which can unbind from the gene by doing an input on u . When protein A binds irreversibly to protein B it executes the process A_B , which can degrade by doing a stochastic delay at rate dAB . Thus, protein A is *neutralised* by protein B . Conversely, gene b can either transcribe a protein B by doing a stochastic delay at rate tB , or bind to protein A by doing an input on channel inhibit . When gene b binds to protein A it receives a private channel u and then executes the process b_A , which can either unbind from the protein by doing an output on u , or transcribe a protein B at a much slower rate tB' . Thus, gene b is *inhibited* by protein A . Protein B can either degrade by doing a stochastic delay at rate dB , or bind irreversibly to protein A by doing an input on channel bind .

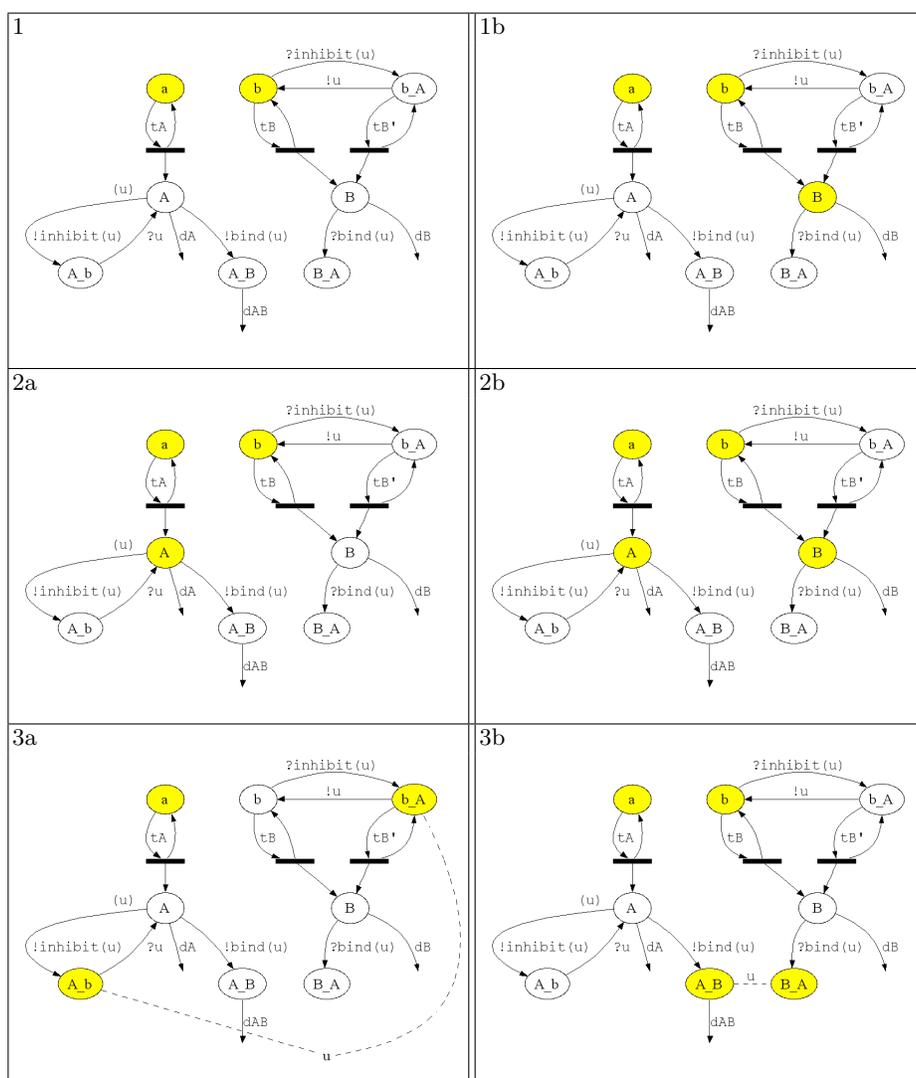


Fig. 12. Execution traces for the bistable gene network of Fig. 11. Initially there are two genes a and b that can transcribe proteins A and B at rates tA and tB , respectively (1). The transcription rate tB is only slightly faster than tA , giving a similar probability for transcribing either protein A or B . If protein A is transcribed first (2a) it can bind to gene b and inhibit the production of protein B (3a). Since protein B is transcribed at a much slower rate tB' , a higher proportion of protein A is produced. Alternatively, if protein B is transcribed first (1b), any subsequently transcribed protein A (2b) can bind irreversibly to protein B and be degraded (3b). Since protein B is transcribed faster than protein A , a higher proportion of protein B is produced.

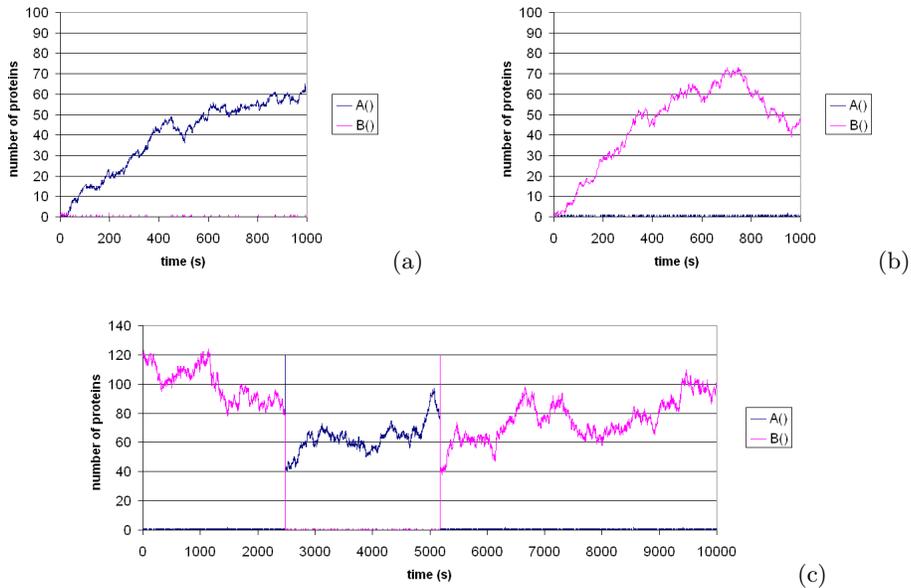


Fig. 13. Simulation results for the bistable gene network of Fig. 11, which show the evolution of the number of proteins over time. The results were obtained by executing the code from Fig. 22 of Appendix A using the SPiM simulator, assuming that the rates are in s^{-1} . Initially, there is a single copy of each gene a and b , with a similar probability of transcribing either protein A or protein B . Depending on the initial transcriptions, the system will either transcribe a high proportion of protein A (a) or a high proportion of protein B (b). When a high proportion of a given protein is transcribed, it suppresses the other protein and the system remains in a stable state. It is possible to toggle between two stable states by injecting a large amount of protein into the system after a given time interval. For example, a system that has a stable production of protein B can be “switched” by artificially injecting a large amount of protein A at time $t \simeq 2500$, and then “switched” again by injecting a large amount of protein B at time $t \simeq 5000$ (c).

5.2 Mapk Cascade

In [8], a model of the mitogen-activated protein kinase (mapk) cascade was presented, and the cascade was shown to perform the function of an ultrasensitive switch. The cascade was studied using a set of reaction equations, which were converted to ordinary differential equations. The equations were solved numerically, and the response curves for the cascade were shown to be steeply sigmoidal. The basic function of the cascade is summarised in Fig. 14.

A graphical stochastic π -calculus model of this system is presented in Fig. 17, and the corresponding code for the model is presented in Fig. 23 of Appendix A.

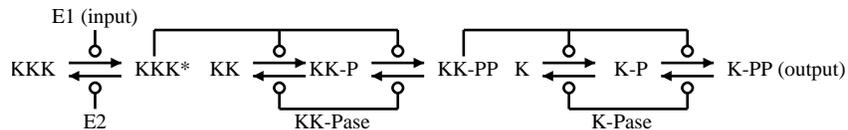
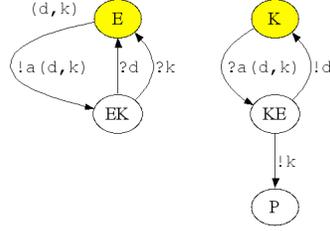


Fig. 14. A model of the mitogen-activated protein kinase (mapk) cascade, as presented in [4]. Initially the cascade contains a large reservoir of substrates KKK, KK and K. When a single enzyme E1 is added, it drives the transformation of KKK to KKK*, which in turn drives the transformation of KK to KK-P to KK-PP, which in turn drives the transformation of K to K-P to K-PP. The effect of these transformations is to produce a rapid increase in the output level of K-PP when an input E1 is added. The transformations can also be driven in the reverse direction by the enzymes E2, KK-Pase and K-Pase, respectively. This allows the output level of K-PP to revert back to zero when the input E1 is removed, so that the cascade can be re-used.

The model represents the reaction between an enzyme E and a substrate K in two stages, as shown in Fig. 15. First, the enzyme binds to the substrate, after which it can either unbind or transform the substrate into a product. An execution trace of a reaction between an enzyme and a substrate is shown in Fig. 16. Stochastic simulation results for the mapk cascade are shown in Fig. 18. The results highlight the increase in signal response as the cascade is traversed from KKK, to KK to K, in accordance with the predictions of [8]. Further simulations across a range of values indicate that the overall function of the system is robust to changes in reactions rates. Even when all of the rates were set to a nominal value of 1.0, the system still behaves as an ultrasensitive switch. Such robustness in system behaviour is perhaps not a coincidence, given that the cascade is used to trigger important processes such as cell division in living organisms.

In previous work, the stochastic π -calculus was used to construct a high-level library of genes, which was used to build networks of varying size and complexity [2]. In principle, a similar approach can also be applied to signalling pathways, such as the mapk cascade in Fig. 17. The cascade is a fairly regular system that consists of proteins with only two types of behaviour, namely enzyme and substrate. The complexity of the system lies in the way multiple combinations of behaviours can be defined for the same protein. The ability to combine behaviours in this way can be modelled more directly by defining a high-level library of enzymes and substrates, as shown in Fig. 19. The library uses simple syntactic sugar, which enables a stochastic π -calculus model for the mapk cascade to be constructed by a combination of calls to the library, as shown in Fig. 20. Taking things a step further, one can also envisage a high-level graphical representation for the library, as illustrated in Fig. 20. In general, one can envision multiple high-level (graphical) libraries for different types of systems, such as gene networks and signalling cascades, all defined in terms of a single underlying (graphical) programming language.



$$\begin{aligned}
 E(a) &= \nu d \nu k !a(d, k).EK(a, d, k) & K(a) &= ?a(d, k).KE(a, d, k) \\
 EK(a, d, k) &= ?d.E(a) + ?k.E(a) & KE(a, d, k) &= !d.K(a) + !k.P() \\
 & & & E(a) \mid K(a)
 \end{aligned}$$

Fig. 15. A stochastic π -calculus model of enzymes and substrates. The reaction between an enzyme E and a substrate K takes place in two stages. First, the enzyme binds to the substrate with a given rate a , after which the enzyme can either unbind with rate d , or transform the substrate to a product P with rate k . This is represented by the reaction equation $E + K \xrightarrow{a} EK \xrightarrow{k} E + P$. A reaction of this form is modelled in the stochastic π -calculus by defining separate processes $E(a)$ and $K(a)$ for the enzyme and substrate, respectively. The enzyme E can bind to the substrate by sending private channels d and k on channel a . The bound enzyme can either unbind by doing an input on d , or react by doing an input on k . Similarly, the substrate K can bind to an enzyme by receiving private channels d and k on channel a . The bound substrate can either unbind by doing an output on d , or react by doing an output on k to produce a product P .

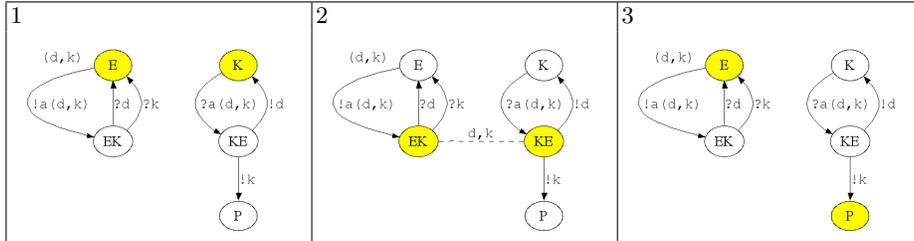


Fig. 16. Execution trace for the enzyme and substrate of Fig. 15. Initially, there is an enzyme E and a substrate K that can interact on channel a (1). The enzyme binds to the substrate by sending private channels d and k on channel a (2). The bound enzyme and substrate can unbind by doing a complementary input and output on channel d , and return to their original state (1). Alternatively, they can react by doing a complementary input and output on channel k . The enzyme returns to its original state, while the substrate is transformed into a product P (3).

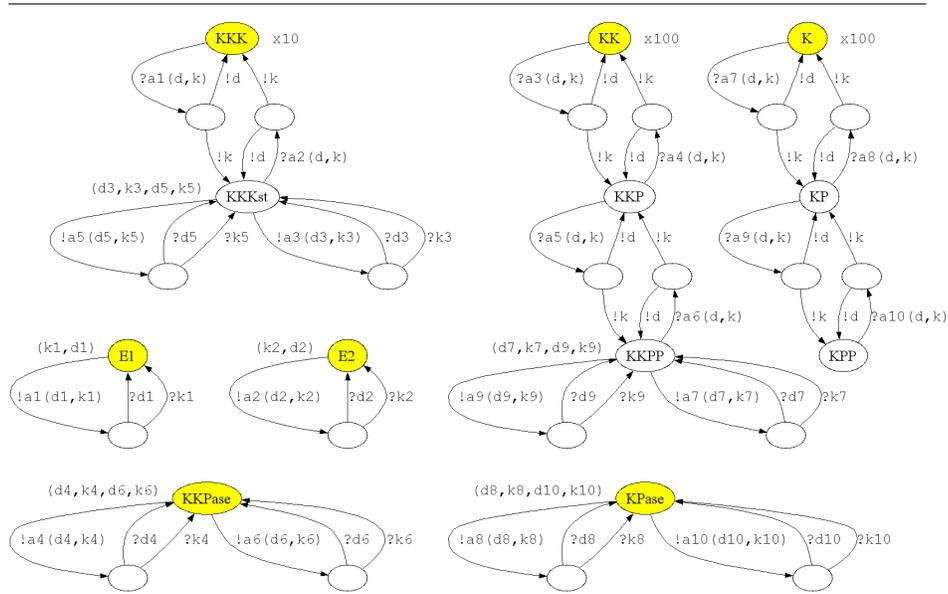


Fig. 17. A graphical stochastic π -calculus model of the mapk cascade presented in [8]. The cascade consists of proteins that can act as enzymes or substrates, as defined in Fig. 15. The process $E1$ can act as an enzyme on a_1 , and the process KKK can act as a substrate on a_1 to produce a product $KKKst$. Conversely, the process $KKKst$ can act as a substrate on a_2 to produce a product KKK . It can also act as an enzyme on both a_3 and a_5 . The remaining enzymes and substrates are defined in a similar fashion.

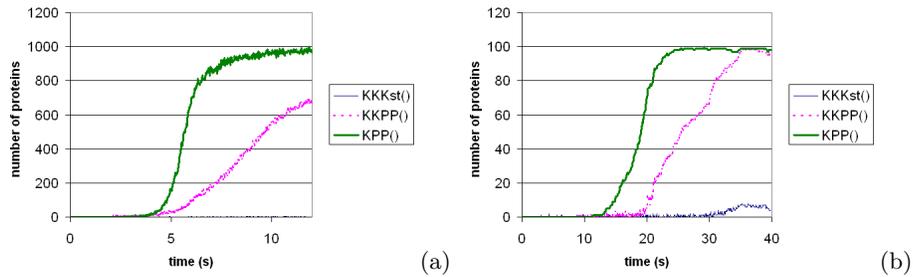
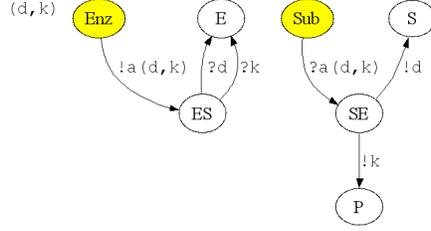


Fig. 18. Simulation results for the mapk cascade of Fig. 17. The results were obtained by executing the code from Fig. 23 of Appendix A, using the SPiM simulator. Simulation (a) was obtained using rates and quantities derived from [8], with $rate(a_i) = 1.0s^{-1}$, $rate(d_i) = rate(k_i) = 150.0s^{-1}$, starting with one of $E1$, $E2$ and $KKPase$, 120 of $KPase$, 3 of KKK and 1200 of KK and K . Simulation (b) was obtained by setting all the rates to a nominal value of 1.0, starting with the quantities in Fig. 17. Both simulations exhibit an increase in signal response as the cascade is traversed from KKK to KK and K . Functionally similar response profiles were observed for the output KPP in both simulations, in spite of the differences in simulation conditions.



$$\begin{aligned}
 \text{Enz}(E, a) &= \nu d \nu k !a(d, k).ES(E, a, d, k) & \text{Sub}(S, P, a) &= ?a(d, k).SE(S, P, a, d, k) \\
 ES(E, a, d, k) &= ?d.E() + ?k.E() & SE(S, P, a, d, k) &= !d.S() + !k.P()
 \end{aligned}$$

Fig. 19. A library of enzymes and substrates, based on the definitions in Fig. 15. The library uses a higher-order variant of the stochastic π -calculus, in which process names can be passed as parameters. The definition of an enzyme Enz is parameterised by the name of the enzyme E , while the definition of a substrate Sub is parameterised by the names of the substrate S and the product P . This allows multiple enzyme and substrate behaviours to be defined for a given protein, by simple combinations of calls to the library. For example, $X() = Sub(X, P, a2) + Enz(X, a3) + Enz(X, a5)$ defines a protein X that can act as a substrate on $a2$ to produce a product P , as an enzyme on $a3$, or as an enzyme on $a5$. The definition relies on additional syntactic sugar for placing an instance inside a choice, where $\nu z'(X(n) + N)$ is short for $\nu z z'(M + N)$, assuming $X(n) = \nu z M$ and $z \cap z' = \emptyset$.

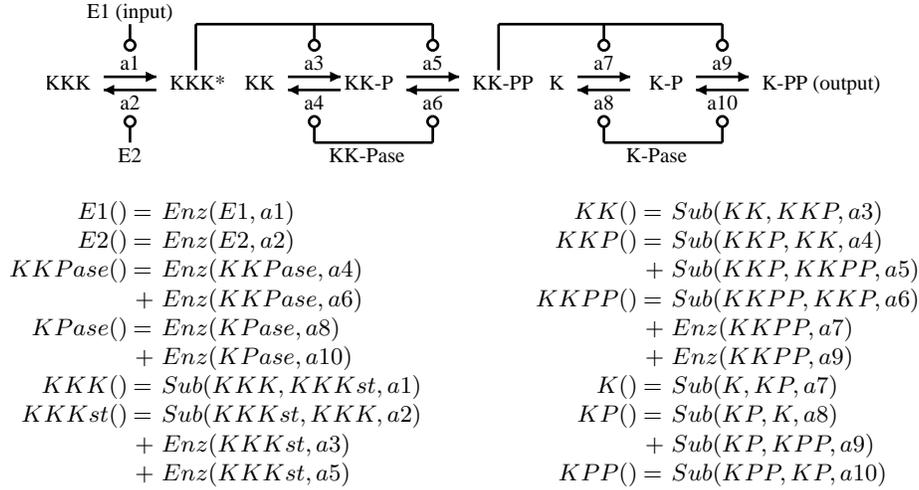


Fig. 20. High-level program code for the mapk cascade of Fig. 17. The code is constructed by calls to the library of enzymes and substrates defined in Fig. 19. The structure of the code gives a clear indication of the function of each protein in the cascade. A corresponding high-level graphical representation for the code is also given. The representation is similar to the biological diagram of Fig. 14, but also contains explicit channel names to denote possible interactions. If the stochastic π -calculus model is *closed* by restricting channels a_1, \dots, a_{10} then the set of possible interactions is fixed, and we obtain exactly the diagram of Fig. 14.

6 Implementation

The variant of the stochastic π -calculus described in this paper has been used to implement the current version of the SPiM programming language, which is used to simulate models of biological systems [14]. The language extends the syntax of the calculus by allowing mutually recursive processes to be defined at arbitrary levels of nesting. This gives rise to a more scalable syntax, which facilitates programming of large systems. A collection of mutually recursive processes is of the form:

$$\text{let } X_1(m_1) = P_1 \text{ and } \dots \text{ and } X_N(m_N) = P_N \text{ in } Q$$

This is encoded into the calculus by expanding the scope of each definition $X_i(m_i) = P_i$ to the top level, adding parameters to each top-level definition to ensure that $\text{fn}(P_i) \subseteq m_i$, and renaming process definitions where necessary to ensure that all top-level definitions are distinct. The transformations are based on standard encodings presented in [21,11,20]. A core syntax of the SPiM programming language is presented in Appendix A.

The implementation can display a process of the graphical stochastic π -calculus by exporting to an open graph syntax such as DOT [5]. DOT is a textual syntax for representing directed graphs, which can be rendered using the Graphviz DOT layout engine. A symbolic core syntax for DOT graphs is described in Definition 9. An encoding $\{E \vdash P\}$ for generating a DOT graph from a system $E \vdash P$ in $\text{GS}\pi$ is presented in Definition 10. The encoding maps each definition of X to a corresponding node with identifier X . This allows mutually recursive definitions to be encoded compositionally, since the DOT layout engine can link edges to nodes based on their identifiers.

Theorem 4. $\forall E, P \in \text{GS}\pi. \{E \vdash P\} \in \text{DOT}$

Proof. By straightforward induction on the definition of encoding in DOT. \square

The way in which nodes, edges and labels are displayed can be customised for a given DOT graph. A node X that corresponds to a choice is displayed as an ellipse with label X , whereas a node that corresponds to a parallel composition is displayed as a solid rectangle. A node X with toplevel z is displayed with label z near the top left of the node. An edge $X \xrightarrow{\pi}_{\sigma} Y$ from a node X to a node Y is displayed as a directed edge from X to Y , with the label π at the midpoint of the edge and the label σ at the head of the edge. A subgraph ${}^z\{G\}$ is displayed by creating a new text node with name z and drawing a dotted edge to each of the nodes in G . If the number of nodes in G is sufficiently large then an alternative representation can be used, in which the nodes are enclosed in a dotted rectangle with label z .

The encoding has been used to implement a graph generating tool, which produces a DOT graph from a given source file written in the SPiM language. First, the SPiM program is encoded to a process of the graphical calculus by adding new definitions according to Definition 7. The resulting process is then encoded to a corresponding DOT graph, according to Definition 10. The graphs

$G ::= X \xrightarrow{\pi}_\sigma Y$ Edge from node X to Y with label π and headlabel σ
 $| X \xrightarrow{\sigma} Y$ Edge from node X to Y with headlabel σ
 $| \quad {}^z X$ Node X with toplevel z
 $| \quad X_\sigma$ Node X with bottomlabel σ
 $| \quad {}^z \{G\}$ Subgraph G with label z
 $| \bigcup_{i=1}^N G_i$ Union of graph declarations $G_1; \dots; G_N$ where $N \geq 0$

Definition 9. *Symbolic Syntax of DOT Graphs.*

$$\{E \vdash P\} \triangleq \{E\}_E; \{P\}_E \quad (22)$$

$$\{\nu z \prod_{i=1}^M P_i\}_E \triangleq {}^z \{\bigcup_{i=1}^M \{P_i\}_E\} \quad (23)$$

$$\{\nu z \mathbf{0}\}_E \triangleq \emptyset \quad (24)$$

$$\{\nu z X(n)\}_E \triangleq {}^z \{X_{\{n/m\}}\} \quad (25)$$

where $X(m) = D \in E$

$$\{\bigcup_{i=1}^N E_i\}_E \triangleq \bigcup_{i=1}^N \{E_i\}_E \quad (26)$$

$$\{X(m) = \nu z \sum_{i=1}^N \pi_i.P_i\}_E \triangleq {}^z X; \bigcup_{i=1}^N X \xrightarrow{\pi_i} [P_i]_E \quad (27)$$

$$\{X(m) = \nu z \prod_{i=1}^M P_i\}_E \triangleq {}^z X; \bigcup_{i=1}^M X \longrightarrow [P_i]_E \quad (28)$$

$$\{X(m) = \nu z Y(n)\}_E \triangleq {}^z X; X \longrightarrow [Y(n)]_E \quad (29)$$

$$[X(n)]_E \triangleq {}_{\{n/m\}} X \quad (30)$$

where $X(m) = D \in E$

$$[P]_E \triangleq Y; \{Y(m) = P\}_E \quad (31)$$

where $P \neq X(n)$ Y fresh $m = \text{fn}(P)$

Definition 10. *Encoding from $\text{GS}\pi$ to DOT.* The function $\{E \vdash P\}$ generates a DOT graph from a given system $E \vdash P$ in $\text{GS}\pi$. The encoding relies on a function $\{E'\}_E$ and a function $\{P\}_E$, which generate a DOT graph from an environment E' and a process P , respectively. Both functions take the initial environment E as a parameter, which is needed for looking up definitions. The encoding also relies on a function $[P]_E$, which ensures that each process P has a corresponding process identifier when drawing an edge to P . This is because the DOT syntax requires each node in the graph to have a unique identifier. If P is of the form $X(n)$ then the identifier X is used. Otherwise, a fresh definition is generated, and the definition name is used as the identifier for P .

in this paper were generated using this tool, and further examples of generated graphs are available from [14]. In practice, some of the elements in the high-level DOT syntax of Definition 9 need to be fine-tuned to improve layout, but the additional modifications are mostly straightforward.

Finally, an abstract machine has been defined for the variant of the stochastic π -calculus presented in this paper, based on the abstract machine presented in [15]. The abstract machine has been used to implement a simulator for the calculus, based on the current implementation available from [14]. In a future version of the simulator, we plan to adapt the encoding of Definition 10 to generate a DOT graph from a machine term after each execution step, in order to render a graphical debugger for visualising the current state of a simulation.

7 Related Work

Pioneering work on Statecharts [7] highlighted the need for a scalable, self-contained graphical representation of concurrent systems. More recent work proposed a synchronous variant to Statecharts, in which concurrent processes can synchronise on shared labels [1]. Our graphical representation uses a similar principle, in contrast with foundational work on graphical representations for the π -calculus [10], which uses more elaborate rules for graph re-writing. In general, graphical representations for process calculi are still an active area of research. For example, [12] describes an automata-based representation for the π -calculus, in which each state of the system is represented as a node in the graph of an automaton. In this paper we adopt a less ambitious but perhaps more scalable approach, which allows new copies of a graph to be generated on demand. From a biological perspective, each new copy represents a new molecule or component, whose internal behaviour is described by a separate graph. Molecules can interact by synchronising on common channels and can also degrade, after which the corresponding graph is deleted. The use of substitution tokens in the graphical calculus is also reminiscent of Petri Nets [?], where each token represents a separate entity in the system.

Preliminary informal ideas on a graphical representation for the stochastic π -calculus were previously presented in [15]. This paper formalises and extends these ideas to produce a novel representation, in which different node types are used to distinguish between stochastic choice and parallel composition. An extended abstract for this paper is presented in [16].

The reduction semantics of [15] relies on a notion of structural congruence for the re-ordering of processes. Although this gives rise to a simplified definition of reduction, it cannot be used in the context of the graphical calculus, since it does not preserve the syntax of processes. In particular, the following structural congruence rule allows $X(n)$ to be instantiated with D , which may contain a choice:

$$X(n) = D \quad \Rightarrow \quad X(n) \equiv D_{\{n/m\}}$$

This violates the syntax of the graphical calculus, since a choice should only occur inside a definition of the environment. In contrast, the transition system

of Definition 3 does not violate the syntax of the graphical calculus, since the corresponding rule (11) allows a reduction to occur without instantiating $X(n)$:

$$X(m) = D \quad D_{\{n/m\}} \xrightarrow{\alpha} P' \quad \Rightarrow \quad X(n) \xrightarrow{\alpha} P'$$

8 Conclusion

This paper presented a graphical representation for the stochastic π -calculus, which was used to model a bistable gene network and a mapk signalling cascade. One of the benefits of the representation is its ability to highlight the existence of cycles, which are a key feature of biological systems. Another benefit is its ability to animate interactions between system components, in order to visualise system dynamics. Such graphical animations are particularly valuable when debugging complex system models.

There are various areas for future work. One issue is to investigate how high-level libraries for different types of biological systems could be built on top of the stochastic π -calculus, as discussed in Sec. 5. It would be interesting to define high-level graphical representations for these libraries, inspired by diagrams such as [13] that are currently being used by biologists.

Another area for future work is to explore ways of minimising the occurrence of substitution labels in the graphical representation of a given process. Such labels are needed whenever a definition is instantiated with arguments that are different from the formal parameters. Interestingly, for the examples considered in this paper, it was always possible to rename the formal parameters in a collection of mutually recursive definitions so that they were same as the applied arguments. For example, in the gene network of Fig. 8 the arguments for $Gene(a, b)$, $Protein(b)$ and $Blocked(a, b)$ were such that the substitution labels in the corresponding graphical representation were all empty. It would be interesting to define algorithms for parameter renaming in the general case, in order to minimise the occurrence of substitution labels.

A somewhat unexpected property of the graphical calculus is that it can potentially be used as the basis for an efficient execution algorithm for the stochastic π -calculus. In particular, the requirement to define each choice separately in the environment is a way of partially mapping out the state space of the system. Thus, instead of generating a new copy of a given process, one can simply keep track of the number of identical copies being executed. In this setting, two processes are considered identical if they instantiate the same process definition with the same parameters. This optimisation would be particularly useful when executing large numbers of identical processes, and could be formally described in terms of the graphical calculus presented in this paper.

In the short term, we plan to use our graph generation tool to implement a graphical debugger for the SPiM simulator. In the longer term, it would be interesting to develop a tool for drawing graphical models, which could automatically generate the corresponding π -calculus code. One can also envisage an interactive visualisation environment, in which disjoint graphs can be displayed

separately or collapsed to a single node by clicking on the graph. Such features are crucial for the scalability of a graphical representation, since they allow a user to visualise parts of the system in a modular fashion, rather than trying to visualise the entire system at once. This ongoing research on graphical interfaces can be used to complement the existing textual interface to the simulator, to help make modelling and simulation of biological systems more accessible to non computer scientists.

References

1. Ch. Andre. Synccharts: A visual representation of reactive behaviors. research report tr95-52, University of Nice, Sophia Antipolis, 1995.
2. Ralf Blossey, Luca Cardelli, and Andrew Phillips. A compositional approach to the stochastic dynamics of gene networks. *Transactions in Computational Systems Biology*, 3939:99–122, January 2006.
3. Michael B. Elowitz and Stanislas Leibler. A synthetic oscillatory network of transcriptional regulators. *Nature*, 403(6767):335–338, January 2000.
4. Paul Francois and Vincent Hakim. Design of genetic networks with specified functions by evolution in silico. *PNAS*, 101:580–585, 2004.
5. Emden R. Gansner and Stephen C. North. An open graph visualization system and its applications to software engineering. *Software-Practice and Experience*, 1-5, 1999.
6. Daniel T. Gillespie. Exact stochastic simulation of coupled chemical reactions. *J. Phys. Chem.*, 81(25):2340–2361, 1977.
7. David Harel. Statecharts : A Visual Formalism for Complex Systems. *Sci. Comput. Prog.*, 8:231–274, 1987.
8. Chi-Ying F. Huang and James E. Ferrel Jr. Ultrasensitivity of the mitogen-activated protein kinase cascade. *PNAS*, 93:10078–10083, 1996.
9. Paola Lecca and Corrado Priami. Cell cycle control in eukaryotes: a biospi model. In *BioConcur'03*. ENTCS, 2003.
10. Robin Milner. Pi-nets: A graphical form of π -calculus. In *ESOP'94*, pages 26–42, 1994.
11. Robin Milner. *Communicating and Mobile Systems: the π -Calculus*. Cambridge University Press, May 1999.
12. Ugo Montanari and Marco Pistore. History-dependent automata: An introduction. *Lecture Notes in Computer Science*, 3465:1–28, 2005.
13. Kanae Oda, Yukiko Matsuoka, Akira Funahashi, and Hiroaki Kitano. A comprehensive pathway map of epidermal growth factor receptor signaling. *Molecular Systems Biology*, 1:E1–E17, May 2005.
14. Andrew Phillips. *The Stochastic Pi-Machine*. Available from <http://www.doc.ic.ac.uk/~anp/spim/>.
15. Andrew Phillips and Luca Cardelli. A correct abstract machine for the stochastic pi-calculus. In *Bioconcur'04*. ENTCS, August 2004.
16. Andrew Phillips and Luca Cardelli. A graphical representation for the stochastic pi-calculus. In *Bioconcur'05*, August 2005.
17. 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.

18. Corrado Priami. Stochastic π -calculus. *The Computer Journal*, 38(6):578–589, 1995. Proceedings of PAPM'95.
19. A. Regev, W. Silverman, and E. Shapiro. Representation and simulation of biochemical processes using the pi- calculus process algebra. In R. B. Altman, A. K. Dunker, L. Hunter, and T. E. Klein, editors, *Pacific Symposium on Biocomputing*, volume 6, pages 459–470, Singapore, 2001. World Scientific Press.
20. Davide Sangiorgi and David Walker. *The π -calculus: a Theory of Mobile Processes*. Cambridge University Press, 2001.
21. David N. Turner. *The Polymorphic Pi-Calculus: Theory and Implementation*. PhD thesis, LFCS, June 1996. CST-126-96 (also published as ECS-LFCS-96-345).

A Program Code

$Dec ::= \mathbf{new} \ x\{\textcircled{r}\} : t$	Channel Declaration
$\mathbf{type} \ n = t$	Type Declaration
$\mathbf{val} \ m = v$	Value Declaration
$\mathbf{run} \ P$	Process Declaration
$\mathbf{let} \ D_1 \ \mathbf{and} \ \dots \ \mathbf{and} \ D_N$	Definitions, $N \geq 1$
$D ::= X(m_1, \dots, m_N) = P$	Definition, $N \geq 0$
$P ::= ()$	Null Process
$(P_1 \mid \dots \mid P_M)$	Parallel, $M \geq 2$
$X(v_1, \dots, v_N)$	Instantiation, $N \geq 0$
$\pi\{; P\}$	Action
$\mathbf{do} \ \pi_1\{; P_1\} \ \mathbf{or} \ \dots \ \mathbf{or} \ \pi_M\{; P_M\}$	Choice, $M \geq 2$
$(Dec_1 \dots Dec_N \ P)$	Declarations, $N \geq 0$
$\pi ::= !x\{(v_1, \dots, v_N)\}$	Output, $N \geq 0$
$?x\{(m_1, \dots, m_N)\}$	Input, $N \geq 0$
$\mathbf{delay}\textcircled{r}$	Delay

Fig. 21. The core SPiM language, where optional elements are enclosed in braces {}.

```

val tA = 0.20 val dA = 0.002
val tB = 0.37 val dB = 0.002
val tB' = 0.027 val dAB = 0.53
new bind@0.72:chan new inhibit@0.19:chan(chan)
let a() = delay@tA; ( A() | a() )
and A() = (
  new u@0.42:chan
  do delay@dA
  or !bind; A_B()
  or !inhibit(u); A_b(u)
)
and A_b(u:chan) = ?u; A()
and A_B() = delay@dAB
let b() =
  do delay@tB; ( B() | b() )
  or ?inhibit(u); b_A(u)
and b_A(u:chan) =
  do !u; b()
  or delay@tB'; B(); b_A(u)
and B() = do delay@dB or ?bind
run (a() | b())

```

Fig. 22. Program code for the bistable gene network of Fig. 11.

```

let E1() = (
  new k1@rk1:chan new d1@rd1:chan
  !a1(d1,k1); do ?d1; E1() or ?k1; E1()
)
let E2() = (
  new k2@rk2:chan new d2@rd2:chan
  !a2(d2,k2); do ?d2; E2() or ?k2; E2()
)
let KKK() = ?a1(d,k); (do !d; KKK() or !k; KKKst())
and KKKst() = (
  new d3@rd3:chan new k3@rk3:chan
  new d5@rd5:chan new k5@rk5:chan
  do ?a2(d,k); (do !d; KKKst() or !k; KKK())
  or !a3(d3,k3); (do ?d3; KKKst() or ?k3; KKKst())
  or !a5(d5,k5); (do ?d5; KKKst() or ?k5; KKKst())
)
let KK() = ?a3(d,k); (do !d; KK() or !k; KKP())
and KKP() =
  do ?a4(d,k); (do !d; KKP() or !k; KK())
  or ?a5(d,k); (do !d; KKP() or !k; KKPP())
and KKPP() = (
  new d7@rd7:chan new k7@rk7:chan
  new d9@rd9:chan new k9@rk9:chan
  do ?a6(d,k); (do !d; KKPP() or !k; KKP())
  or !a7(d7,k7); (do ?d7; KKPP() or ?k7; KKPP())
  or !a9(d9,k9); (do ?d9; KKPP() or ?k9; KKPP())
)
let K() = ?a7(d,k); (do !d; K() or !k; KP())
and KP() =
  do ?a8(d,k); (do !d; KP() or !k; K())
  or ?a9(d,k); (do !d; KP() or !k; KPP())
and KPP() = ?a10(d,k); (do !d; KPP() or !k; KP())
let KKPase() = (
  new d4@rd4:chan new k4@rk4:chan
  new d6@rd6:chan new k6@rk6:chan
  do !a4(d4,k4); (do ?d4; KKPase() or ?k4; KKPase())
  or !a6(d6,k6); (do ?d6; KKPase() or ?k6; KKPase())
)
let KPase() = (
  new d8@rd8:chan new k8@rk8:chan
  new d10@rd10:chan new k10@rk10:chan
  do !a8(d8,k8); (do ?d8; KPase() or ?k8; KPase())
  or !a10(d10,k10); (do ?d10; KPase() or ?k10; KPase())
)
run (10 of KKK() | 100 of KK() | 100 of K())
run ( 1 of E2() | 1 of KKPase() | 1 of KPase() | E1())

```

Fig. 23. Program code for the mapk cascade of Fig. 17.

B The Stochastic π -calculus vs. Reaction Equations

This appendix gives a visual comparison between the stochastic π -calculus and reaction equations, using the example biological systems described in the main text. The stochastic π -calculus allows the description of a biological system to be decomposed into distinct components, where each component is described by a separate connected graph. Each node in the graph represents a state of the component, and each labelled edge represents a potential interaction with another component. The interactions between components are determined by the complementarity of actions on the edges, and do not need to be given explicitly. This allows new components to be added directly, without modifying the existing system. As a result, large and complex systems can be defined incrementally, by direct composition of simpler components. In contrast, reaction equations require the interactions between components to be defined explicitly, resulting in a highly connected graph. If a new component is added to the system, each interaction with an existing component needs to be defined by an additional edge to the component. If each new component can interact with multiple existing components, this leads to a combinatorial explosion in the number of edges.

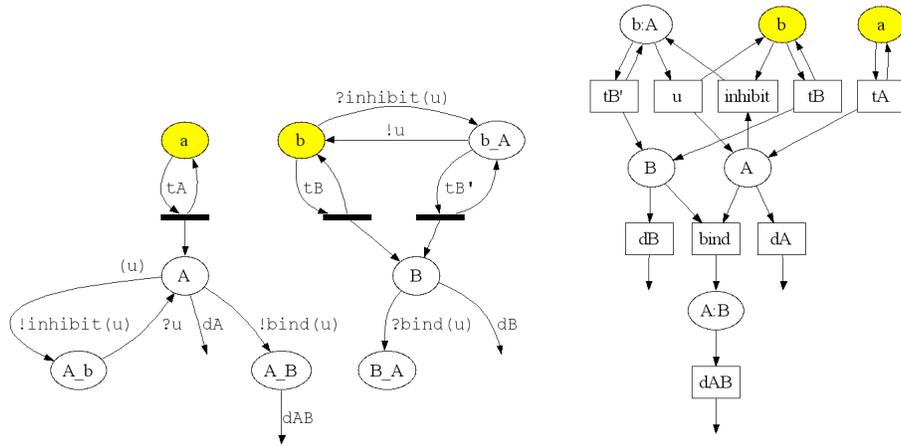


Fig. 24. Visual comparison between stochastic π -calculus processes (left) and reaction equations (right) for the bistable gene network of Fig. 11.

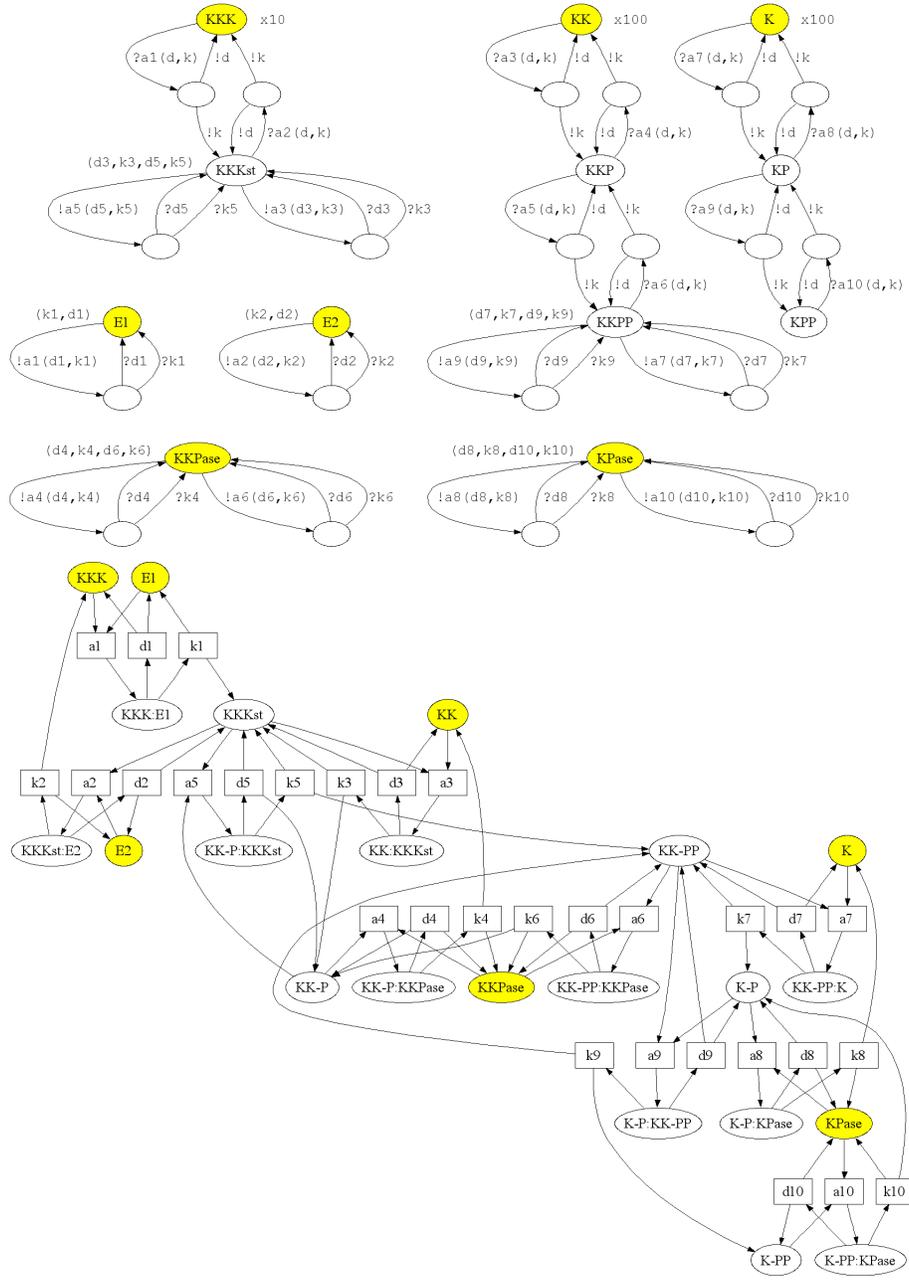


Fig. 25. Visual comparison between stochastic π -calculus processes (top) and reaction equations (bottom) for the mapk cascade of Fig. 17.