A Graphical Representation for the Stochastic Pi-calculus

Andrew Phillips and Luca Cardelli

Microsoft Research,
7 JJ Thomson Avenue
Cambridge, UK
{v-anphil,luca}@microsoft.com

Abstract. This paper presents a graphical representation for the stochastic pi-calculus, which builds on previous formal and informal notations. The graphical representation is used to model a Mapk signalling cascade and an evolved gene network. One of the main benefits of the representation is its ability to clearly highlight the existence of cycles, which are a key aspect of many biological systems. Another advantage is its ability to animate interactions between biological system components, in order to clarify the overall system function. The paper also shows how the graphical representation can be used as a front end to a stochastic simulator for the pi-calculus, in order to allow the direct simulation of graphical models. This complements the existing textual interface of the simulator, with a view to making modelling and simulation of biological systems more accessible to non computer scientists.

1 Introduction

The stochastic pi-calculus has been used to model and simulate a range of biological systems [6,13,14]. One of the main benefits of the calculus is its ability to model independent system components, which can then be composed in order to predict emergent system behaviour. The calculus is mathematically precise, which makes it amenable to a range of analysis techniques, including type systems, behavioural equivalences and model checking. However, the mathematical syntax of the calculus, together with the subtleties of its stochastic reduction semantics, can sometimes limit the accessibility of the model to a wider audience. In such cases, it can be useful to present a graphical view of the calculus to complement its textual notation. From our experience, such an alternative representation would be particularly welcomed by experimental systems biologists.

This paper presents a graphical representation for the stochastic pi-calculus, building on a number of existing formal and informal notations. The paper is structured as follows. Section 2 presents a variant of the stochastic pi-calculus that supports internal transitions and recursive definitions. From our experience, such constructs are useful for modelling biological systems at a more abstract level. Section 3 presents a graphical representation for the stochastic pi-calculus, together with a graphical execution model. The representation is designed to

highlight the existence of cycles, which are a key feature of many biological systems. The execution model can be used to animate the interactions between biological system components, in order to clarify the overall system function. Section 4 describes how the graphical representation can be used to model a Mapk signalling cascade [5] and an evolved gene network [2]. Finally, Section 5 shows how the graphical representation can be used as a front end to a stochastic simulator for the pi-calculus, in order to allow the direct simulation of graphical models. This complements the existing textual interface to the simulator, with a view to making modelling and simulation of biological systems more accessible to non computer scientists.

2 The Stochastic Pi-calculus

$P, Q ::=$	$\nu x\, P$	Restriction	$\Sigma ::=$	$\mathbf{0}$	Null
$\mid$	$P \mid Q$	Parallel	$\mid$	$\pi.P + \Sigma$	Action
$\mid$	Σ	Summation			
$\mid$	$X(n)$	Instance	$\pi ::=$	$?x(m)$	Input
			$\mid$	$!x(n)$	Output
$\Gamma ::=$	$\emptyset$	Empty	$\mid$	τ_r	Delay
$\mid$	$X(m) \triangleq P, \Gamma$	Definition			

Definition 1. *Syntax of* SPi

$$!x(n).P + \Sigma \mid ?x(m).Q + \Sigma' \xrightarrow{rate(x)} P \mid Q_{\{n/m\}} \quad (1)$$

$$\tau_r.P + \Sigma \xrightarrow{r} P \quad (2)$$

$$P \xrightarrow{r} P' \Rightarrow P \mid Q \xrightarrow{r} P' \mid Q \quad (3)$$

$$P \xrightarrow{r} P' \Rightarrow \nu x\, P \xrightarrow{r} \nu x\, P' \quad (4)$$

$$Q \equiv P \xrightarrow{r} P' \equiv Q' \Rightarrow Q \xrightarrow{r} Q' \quad (5)$$

Definition 2. *Reduction in* SPi

$$P \mid \mathbf{0} \equiv P \quad (6)$$

$$P \mid Q \equiv Q \mid P \quad (7)$$

$$P \mid (Q \mid R) \equiv (P \mid Q) \mid R \quad (8)$$

$$\nu x\, \mathbf{0} \equiv \mathbf{0} \quad (9)$$

$$\nu x\, \nu y\, P \equiv \nu y\, \nu x\, P \quad (10)$$

$$\pi.P + \pi'.P' + \Sigma \equiv \pi'.P' + \pi.P + \Sigma \quad (11)$$

$$x \notin \mathrm{fn}(P) \Rightarrow \nu x\, (P \mid Q) \equiv P \mid \nu x\, Q \quad (12)$$

$$X(m) \triangleq P \Rightarrow X(n) \equiv P_{\{n/m\}} \quad (13)$$

$$P \equiv P' \Rightarrow \nu x\, P \equiv \nu x\, P' \quad (14)$$

$$P \equiv P' \Rightarrow P \mid Q \equiv P' \mid Q \quad (15)$$

$$P \equiv P' \Rightarrow \pi.P + \Sigma \equiv \pi.P' + \Sigma \quad (16)$$

$$\Sigma \equiv \Sigma' \Rightarrow \pi.P + \Sigma \equiv \pi.P + \Sigma' \quad (17)$$

Definition 3. *Structural Congruence in* SPi

The variant of the stochastic pi-calculus used in this paper is summarised in Definitions 1, 2 and 3. The calculus supports internal transitions and recursive definitions, as presented in [8,15]. Recursive definitions have been argued in [14] to be a more practical programming abstraction for biological systems than the basic replication semantics of the pi-calculus. In this paper, we also show how internal transitions labelled with a stochastic delay can provide a useful programming abstraction. Stochastic behaviour is incorporated into the calculus by associating each channel x with a corresponding reaction rate given by $rate(x)$ and labelling each reduction with a reaction rate r as in [12]. In addition, each internal transition τ is associated with a reaction rate r. A given process of the calculus is assumed to execute in the presence of a finite environment Γ, which contains a number of top-level definitions. For each process $X(n)$ there is assumed to be a corresponding definition of the form $X(m) \triangleq P$ with $\text{fn}(P) \subseteq m$, as described in [8,15].

The variant of the stochastic pi-calculus presented above has been used as the basis for the current version of the SPiM programming language [11]. The language also allows a process P to be of the form:

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

which can be used to define mutually recursive processes at arbitrary levels of nesting. This gives rise to a more scalable and modular syntax, which is essential for programming large systems. A process of this form can be encoded into the calculus by expanding the scope of each definition to the top level, adding parameters to each top-level definition $X(m) \triangleq P$ to ensure that $\text{fn}(P) \subseteq m$, and renaming process definitions where necessary to ensure that all top-level definitions are distinct. The transformations are based on standard encodings presented in [16,8,15].

3 Graphical Representation

A graphical representation for the stochastic pi-calculus can be defined with the help of a corresponding graphical calculus, as described in Definitions 4 and 5. The syntax of the graphical calculus corresponds to a normal form, in which each summation or guarded parallel composition (prefixed by an action π) is associated with a separate definition. Each definition can also contain a number of restricted channels.

Definition 6 describes how the definitions of the graphical calculus can be displayed. A collection of mutually recursive definitions

$$X_1(m_1) \triangleq C_1, \ldots, X_N(m_N) \triangleq C_N$$

is displayed as a directed graph with nodes $X_1 \ldots X_N$ and with edges between these nodes. Each definition $X(m) \triangleq C$ is displayed as a node X with zero or more edges to subsequent nodes in the graph. If C is of the form

$$X(m) \triangleq \nu x_1\ \ldots \nu x_M\ (\pi_1.X_1(n_1) + \ldots + \pi_N.X_N(n_N))$$

$P, Q ::=$	$\nu x\, P$	Restriction	$\pi ::=$	$?x(m)$	Input
	$\mid P \mid Q$	Parallel	$\mid$	$!x(n)$	Output
	$\mid X(n)$	Instance	$\mid$	τ_r	Delay

Definition 4. *Syntax of Processes in* GSPi

$\Gamma ::=$	$\emptyset$	Empty	$\Sigma ::=$	$\mathbf{0}$	Null
	$\mid X(m) \triangleq C, \Gamma$	Definition		$\mid \pi.X(n) + \Sigma$	Action
$C ::=$	$\nu x\, C$	Restriction	$\Pi ::=$	$X(n)$	Instance
$\mid$	Σ	Summation	$\mid$	$\Pi \mid \Pi$	Parallel
$\mid$	Π	Composition			

Definition 5. *Syntax of Definitions in* GSPi

then each edge from node X to node X_i is labelled with an action π_i and denotes an alternative execution path in the system. The node X is represented as an ellipse with name X, inside a bubble labelled with the restricted names $x_1 \ldots x_M$. Alternatively, if C is of the form

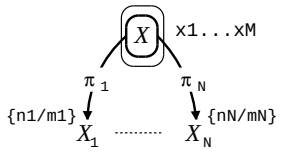
$$X(m) \triangleq \nu x_1 \ldots \nu x_M \, (X_1(n_1) \mid \ldots \mid X_N(n_N))$$

then each edge from node X to node X_i denotes a concurrent execution path in the system. The node X is represented as a solid rectangle, inside a bubble labelled with the restricted names $x_1 \ldots x_M$.

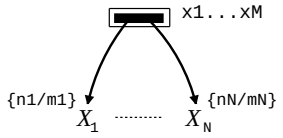
In both cases, the head of each edge leading to a node X_i is labelled with a substitution $_{\{n_i/m_i\}}$, where m_i represents the formal parameters of the definition X_i and n_i represents the parameter instantiations. Mappings in the substitution that correspond to the identity function are not shown, and if $n_i = m_i$ then the substitution is omitted altogether. Interestingly, most of the chemical and biological systems we have encountered can be modelled using mutually recursive definitions for which the parameter instantiations of each definition are equal to the formal parameters. This gives rise to graphical models in which there are no substitution labels on any of the edges. Examples of these chemical and biological system models can be found at [11] together with their corresponding graphical representation. Section 4 highlights two of our more recent models.

In order to obtain a more compact representation of definitions, one can also envisage an interactive navigation 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.

$X(m) \triangleq \nu x_1 \ldots \nu x_M \, (\pi_1.X_1(n_1) + \ldots + \pi_N.X_N(n_N))$



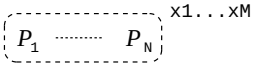
$X(m) \triangleq \nu x_1 \ldots \nu x_M \, (X_1(n_1) \mid \ldots \mid X_N(n_N))$



Definition 6. *Graphical Display of Definitions in* GSPi

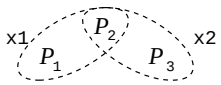
$X(n)$

$\nu x_1 \ldots \nu x_M \, (P_1 \mid \ldots \mid P_N)$



Definition 7. *Graphical Display of Processes in* GSPi

Definition 7 describes how the processes of the graphical calculus can be displayed. An instance $X(n)$ of a definition $X(m) \triangleq C$ is displayed as a highlighted node X, labelled with the substitution $_{\{n/m\}}$. A collection of parallel processes $P_1, \ldots P_N$ with a number of restricted channels $x_1, \ldots, x_M$ is displayed as a bubble around the processes, labelled with the names of the channels. This is reminiscent of previous graphical representations for restriction [8] and is useful for visualising the formation of a complex between two processes. The use of restriction to model complexation was first described in [14], where a complex of two processes P and Q is modelled as a restriction $\nu x \, (P \mid Q)$. The restricted name x represents a private channel on which the two processes can synchronise in order to split the complex. The graphical notation highlights the formation of the complex by grouping P and Q inside a common boundary labelled with the name of the channel x. In addition, the graphical representation can more accurately reflect the scope of restricted channels through the use of overlapping boundaries. For example, a restriction $\nu x_1 \, \nu x_2 \, (P_1 \mid P_2 \mid P_3)$ where $x_1 \notin \mathrm{fn}(P_3)$ and $x_2 \notin \mathrm{fn}(P_1)$ can be displayed as:



An alternative to drawing a boundary around nodes that share a restricted channel is to draw a dotted line between the nodes, labelled with the channel name. A line connecting two nodes represents the formation of a bond between these nodes, resulting in a more compact representation that is still relatively intuitive.

In general, there are a number of complementary ways to display a graphical process. For example, instead of just displaying the node X labelled with a substitution $_{\{n/m\}}$, the entire graph of nodes connected to X can be reproduced. The graph can be updated as subsequent definitions are instantiated, allowing successive nodes to be highlighted in the style of state machines. Note that after each parallel composition a new copy of the graph is spawned. This ensures that only a single node in a given graph can be highlighted at a given instant. Alternatively, the substitution labels can all be placed on the same graph, where each substitution corresponds to a token in the style of Petri Nets [10]. In this representation a new token is produced after each parallel composition, and tokens can move around the graph concurrently. Note that care must also be taken to rename restricted channels in order to avoid name clashes.

By definition, the set of graphical processes GSPi corresponds to a normal form for the set of processes SPi, such that GSPi $\subset$ SPi. As a result, reduction in GSPi can be defined using reduction in SPi. According to Proposition 1, if a process in the set GSPi can reduce then the resulting process is also in the set GSPi, up to structural congruence. The proof is by induction on the definition of reduction in SPi, where each summation or guarded parallel composition is associated with a separate definition.

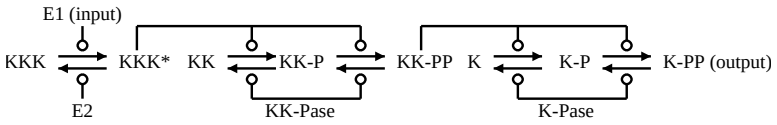
Proposition 1. $\forall P \in \text{GSPi}.P \xrightarrow{r} P' \Rightarrow \exists P'' \in \text{GSPi}.P' \equiv P''$

A dynamic representation of the current state of a process can be obtained by displaying the graphical syntax after each reduction step. This can be used to visualise the interactions between biological system components, in order to clarify the overall system function or to allow for graphical system debugging. Similarly, the rules for structural congruence can be used to re-arrange the processes of the graphical calculus, in order to re-arrange the corresponding graph. The graphical representation has a number of advantages over its textual counterpart. Firstly, it clearly illustrates the connectivity between process definitions through the use of labelled arcs. In particular, these arcs highlight the existence of cycles, which are a key feature of many biological systems. The representation also resembles standard representations of state machines, in which the current active state is highlighted. By definition, each highlighted node can perform a transition to a subsequent node in the graph, which will in turn be highlighted. This successive highlighting of nodes can be used to visualise the execution trace of a model, for both tutorial and debugging purposes.

4 Biological Examples

4.1 Mapk Cascade

The graphical stochastic pi-calculus can be used to model the system described in [5], which studies the ultrasensitivity of the mitogen-activated protein kinase (Mapk) cascade. The system was described using a set of reaction equations, which were then converted to ordinary differential equations. These equations were solved numerically and the response curves for the MAPK cascade were shown to be steeply sigmoidal. The overall function of the system is illustrated as follows:



The input E1 acts as an enzyme to drive 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, completing the cascade.

Based on the reaction equations described in the paper, we developed the corresponding pi-calculus model illustrated in Fig. 1. The model represents the enzymatic reaction between an enzyme E and a substrate K 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 can be written using the reaction equation:

$$E + K \; {}_{d}{\rightleftarrows}^{a} \; E{\cdot}K \rightarrow^{k} E + P$$

A reaction of this form is modelled in the pi-calculus by defining separate processes for the enzyme and the substrate. The enzyme E is modelled as a process $E(a)$ with private channels d and k. The enzyme can first bind with a substrate by performing an output on a, sending the channels d and k. The bound enzyme can then either unbind by performing an input on d, or react by performing an input on k:

$$E(a) \triangleq \nu d\, \nu k\, !a(d,k).(?d.E(a) \; + \; ?k.E(a))$$

The substrate K is modelled as a process $K(a)$. The substrate can first bind with an enzyme by performing an input on a, receiving the channels d and k. The bound substrate can then either unbind by performing an output on d, or react by performing an output on k to produce a product $P()$:

$$K(a) \triangleq ?a(d,k).(!d.K(a) \; + \; !k.P())$$

For example, the process $E1$ in Fig. 1 is an enzyme that can interact on a_1 and the process KKK is a substrate that can interact on a_1 to produce a product

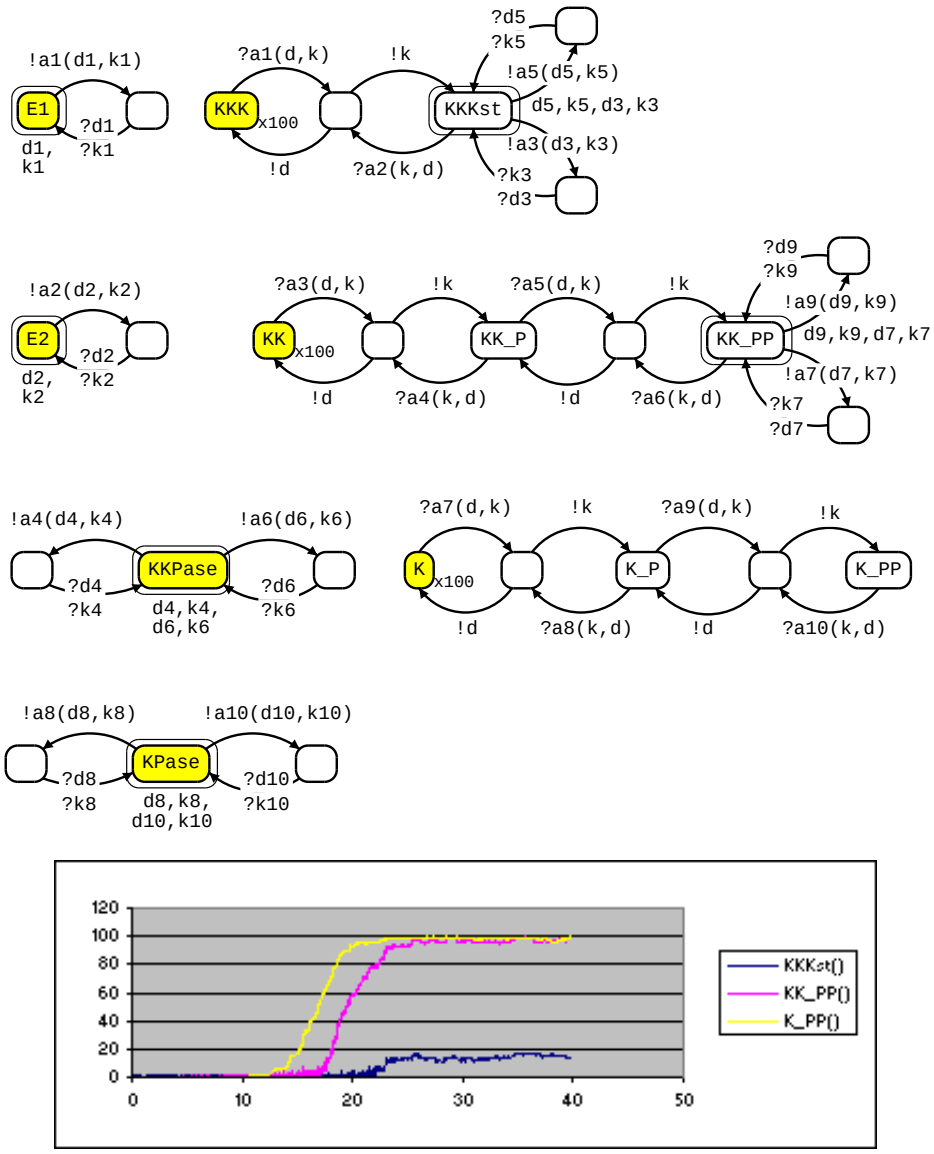


Fig. 1. Graphical model and simulation results for the Mapk Cascade [5]. The results were obtained with all rates set to 1.0, starting with 1 of each enzyme and 100 of each substrate.

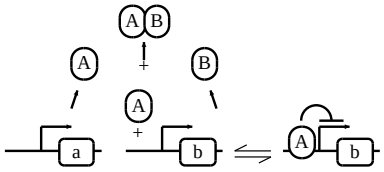
$KKKst$. Note that $KKKst$ is itself an enzyme that can interact on both a_3 and a_5. The remaining enzymes and substrates in Fig. 1 are defined in a similar fashion.

The model was simulated in SPiM, where the corresponding code is given in Appendix A. The code is given for reference only, since the graphical model in Fig. 1 is a complete description of the system. The graphical representation highlights the cyclic nature of the cascade, and shows the relationship between the different states of each substrate. A graphical debugging trace is available from [11], which shows a cartoon sequence of reactions for the model. This helps clarify the behaviour of the model, in order to check that it does indeed reflect the behaviour of the system. Such graphical traces are particularly valuable for developing and debugging complex models in the stochastic pi-calculus.

The simulation results for the model are presented in Fig. 1. 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 [5]. From our simulations we have been able to demonstrate that the overall system response is highly robust to changes in reactions rates. Even when all of the rates were set to a nominal value of 1.0, the system response profile was preserved. More detailed simulation results are available from [11].

4.2 Evolved Gene Network

The graphical stochastic pi-calculus can also be used to model one of the systems described in [2], which shows how gene networks can be evolved in silico to perform specific functions. Unlike the Mapk example, in this system the number of components changes dynamically, since new proteins can be produced and degraded over time. The system was evolved to perform the basic function of a bistable switch, which resulted in the following gene network:



In this network, the genes a and b can produce proteins A and B respectively, with a constitutive transcription rate. Proteins A and B can bind irreversibly to produce the complex AB, which can eventually degrade. Protein A can also bind reversibly to gene b, in order to inhibit the transcription of B. The corresponding pi-calculus model of this system is described in Fig. 2, and the model code is given in Appendix A. The stochastic simulation results illustrate that the system does indeed behave as a bistable switch. 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. Further simulation results of the system are also available from [11],

including various responses to input pulses, which match the results presented in [2]. A cartoon execution trace of the system is also available.

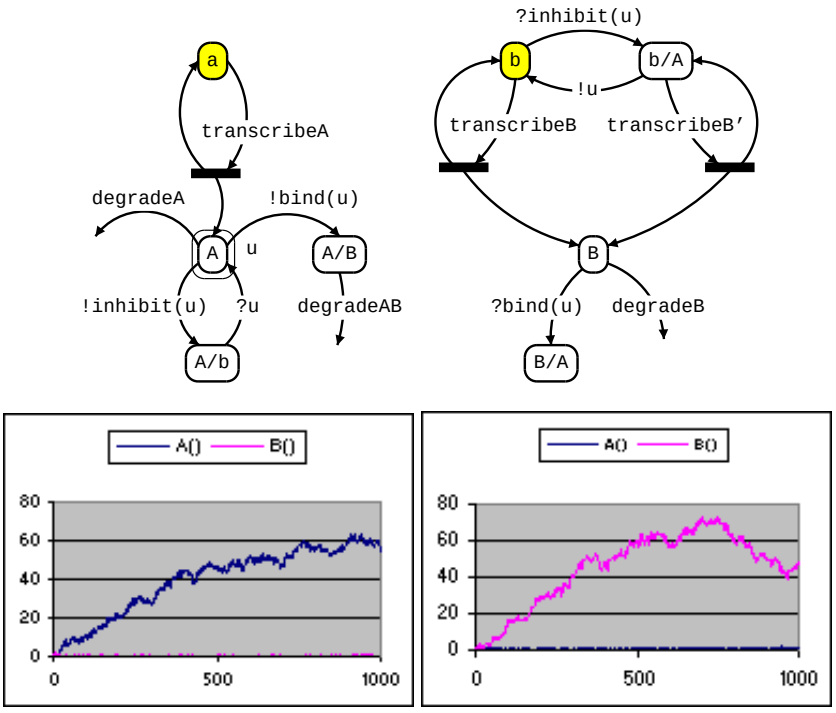


Fig. 2. Graphical model and simulation results for an Evolved Gene Network [2]. The results were obtained with the rates given in Appendix A, starting with a single gene for a and b.

5 Graph Generation

A process of the graphical stochastic pi-calculus can be visualised by exporting to an open graph syntax known as DOT [3]. 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 8, where labels are optional and represent arbitrary strings. An encoding $[\![\Gamma]\!]_I$ for generating a DOT graph from a process environment Γ is presented in Definition 9, where I represents a node identifier. The encoding maps each definition X to a corresponding node with identifier X. This allows mutually recursive definitions to be encoded compositionally, since the layout engine can

$$
\begin{array}{rcll}
G ::= & I[label] & \text{Labelled node with id } I \\
| & I \xrightarrow{label}_{label} J & \text{Labelled edge from node } I \text{ to node } J \\
| & G; G' & \text{Sequence of graph declarations}
\end{array}
$$

Definition 8. *Syntax of DOT Graphs*

$$
\begin{array}{rcl}
(\!|\emptyset|\!)_I & \triangleq & \emptyset \\
(\!|X(m) \triangleq C, \Gamma|\!)_I & \triangleq & (\!|C|\!)_X; (\!|\Gamma|\!)_I \\
(\!|\nu x_1 \ldots \nu x_N\, C|\!)_I & \triangleq & I[x_1, \ldots, x_N]; (\!|C|\!)_I \\
X(m) \triangleq C \quad \Rightarrow \quad (\!|X(n)|\!)_I & \triangleq & I \longrightarrow_{\{n/m\}} X \\
X(m) \triangleq C \quad \Rightarrow \quad (\!|X(n) \mid \Pi|\!)_I & \triangleq & I \longrightarrow_{\{n/m\}} X; (\!|\Pi|\!)_I \\
(\!|\mathbf{0}|\!)_I & \triangleq & \emptyset \\
X(m) \triangleq C \quad \Rightarrow \quad (\!|\pi.X(n) + \Sigma|\!)_I & \triangleq & I \xrightarrow{\pi}_{\{n/m\}} X; (\!|\Sigma|\!)_I
\end{array}
$$

Definition 9. *Encoding a Graphical Process Environment into a DOT Graph*

link edges to nodes based on their identifiers. The encoding is proved sound with respect to the DOT syntax, as stated in Proposition 2, where GSPi_Γ is the set of process environments of the graphical calculus. (the proof is straightforward).

Proposition 2. $\forall \Gamma. \Gamma \in \text{GSPi}_\Gamma \Rightarrow (\!|\Gamma|\!)_I \in \text{DOT}$

The way in which nodes, edges and labels are displayed can be customised for a given DOT graph. A node I that corresponds to a summation is displayed as an ellipse around the identifier I, whereas a node that corresponds to a parallel composition is displayed as a solid rectangle. A labelled node $I[l]$ is displayed as a bubble around the node I, labelled with l. An edge $I \xrightarrow{l}_h J$ from a node I to a node J is displayed as a directed arc from I to J, with the label l at the midpoint of the arc and the label h at the head of the arc.

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. For space reasons, the graphs in this paper were produced manually, but they closely resemble the generated graphs produced by the tool. Examples of generated graphs are available from [11]. A corresponding encoding can also be defined for generating a graph from an arbitrary process of the graphical calculus. The encoding adds environment labels to nodes and spawns new copies of graphs where appropriate. The encoding also connects nodes that share restricted channels by a dotted line, labelled with the names of the channels.

Finally, an abstract machine has been defined for the stochastic pi-calculus with recursion and internal transitions, based on the abstract machine presented in [12]. The abstract machine can be used to implement a simulator for the

calculus, based on the current implementation available from [11]. The encoding from graphical processes to DOT graphs can also be adapted to generate a graph from a given machine term. In a future version of the simulator we plan to generate a DOT graph after each execution step, in order to render a graphical debugger for visualising the current state of a simulation.

6 Related Work and Conclusion

This paper presents a graphical representation for the stochastic pi-calculus, which is used to model a Mapk signalling cascade and an evolved gene network. One of the main benefits of the representation is its ability to clearly highlight the existence of cycles, which are a key aspect of many biological systems. Another advantage is its ability to animate interactions between biological system components, in order to clarify the overall system function and to debug changes in the system.

Pioneering work on Statecharts [4] 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 pi-calculus that rely on more elaborate rules for graph re-writing [7]. Graphical representations are still very much an issue in modern-day computing. In particular, [9] describes an automata-based representation for the pi-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. Preliminary informal ideas on a graphical representation for the stochastic pi-calculus were previously presented in [12]. 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.

There are numerous areas for future work. The graphical representation exploits the fact that, within a collection of mutually recursive definitions, the arguments applied to a definition are often the same as the formal parameters. This observation was based on experience in modelling a range of biological systems, but was not obvious a priori. In general, it would be interesting to investigate which design patterns occur most frequently in various types of biological systems, in order to improve modelling and visualisation techniques. In the short term, we plan to use our existing graph generation tool to implement a graphical debugger for the SPiM simulator. In addition, it would be interesting to develop a tool for drawing graphical models, which could automatically generate the corresponding pi-calculus code. This ongoing research on graphical

interfaces can be used to complement the existing textual interface to the simulator, with a view to making modelling and simulation of biological systems more accessible to non computer scientists.[1]

References

1. Ch. Andre. Syncharts: A visual representation of reactive behaviors. research report tr95-52, University of Nice, Sophia Antipolis, 1995.
2. Paul Francois and Vincent Hakim. Design of genetic networks with specified functions by evolution in silico. volume 101, pages 580–585, 2004.
3. 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.
4. David Harel. Statecharts : A Visual Formalism for Complex Systems. *Sci. Comput. Prog.*, 8:231–274, 1987.
5. Chi-Ying F. Huang and Jr. James E. Ferrel. Ultrasensitivity of the mitogen-activated protein cascade. volume 93, pages 10078–10083, 1996.
6. Paola Lecca and Corrado Priami. Cell cycle control in eukaryotes: a biospi model. In *BioConcur'03*. ENTCS, 2003.
7. Robin Milner. Pi-nets: A graphical form of π-calculus. In *ESOP'94*, pages 26–42.
8. Robin Milner. *Communicating and Mobile Systems: the π-Calculus*. May 1999.
9. Ugo Montanari and Marco Pistore. History-dependent automata: An introduction. volume 3465, 2005.
10. Carl Adam Petri. *Kommunikation mit Automaten.* PhD thesis, Bonn: Institut für Instrumentelle Mathematik, Schriften des IIM Nr. 2, 1962. Second Edition:, New York: Griffiss Air Force Base, Technical Report RADC-TR-65–377, Vol.1, 1966, Pages: Suppl. 1, English translation.
11. Andrew Phillips. *The Stochastic Pi-Machine*. Available from http://www.doc.ic.ac.uk/~anp/spim/.
12. Andrew Phillips and Luca Cardelli. A correct abstract machine for the stochastic pi-calculus. In *Bioconcur'04*. ENTCS, August 2004.
13. 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*, 2001. In press.
14. 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.
15. Davide Sangiorgi and David Walker. *The π-calculus: a Theory of Mobile Processes*. Cambridge University Press, 2001.
16. David N. Turner. *The Polymorphic Pi-Calculus: Theory and Implementation*. PhD thesis, June 1996. CST-126-96 (also published as ECS-LFCS-96-345).

[1] Many thanks to Giuseppe Castagna, James Margetson and Leonard Goldstein for their useful suggestions and comments on this work.

A Program Code

Dec	::= **new** x{@r}:t	Channel Declaration
	| **type** n = t	Type Declaration
	| **val** m = v	Value Declaration
	| **run** P	Process Declaration
	| **let** D_1 **and** ... **and** D_N	Definitions, $N \geq 1$
D	::= X(m_1, ..., m_N) = P	Definition, $N \geq 0$
P	::= ()	Null Process
	| (P_1 | ... | P_M)	Parallel, $M \geq 2$
	| X(v_1, ..., v_N)	Instantiation, $N \geq 0$
	| π{; P}	Action
	| **do** π_1{; P_1} or ... or π_M{; P_M}	Choice, $M \geq 2$
	| (Dec_1 ... Dec_N P)	Declarations, $N \geq 0$
π	::= !x {(v_1, ..., v_N)}	Output, $N \geq 0$
	| ?x {(m_1, ..., m_N)}	Input, $N \geq 0$
	| delay@r	Delay

Fig. 3. The Stochastic Pi Language

```
val transcribeA = 0.20 val degradeA = 0.002
val transcribeB = 0.37 val degradeB = 0.002
val transcribeB' = 0.027 val degradeAB = 0.53
new bind@0.72:chan new inhibit@0.19:chan(chan)
let a() = delay@transcribeA; ( A() | a() )
and A() = (
  new u@0.42:chan
  do delay@degradeA
  or !bind; A_B()
  or !inhibit(u); A_b(u)
)
and A_b(u:chan) = ?u; A()
and A_B() = delay@degradeAB
let b() =
  do delay@transcribeB; ( B() | b() )
  or ?inhibit(u); b_A(u)
and b_A(u:chan) =
  do !u; b()
  or delay@transcribeB'; B(); b_A(u)
and B() =  do delay@degradeB  or ?bind
run (a() | b())
```

Fig. 4. Program Code for Evolved Gene Network

```
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); KKK_E(d,k)
and KKK_E(d:chan,k:chan) = 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(k,d); KKK_E(d,k)
  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); KK_E(d,k)
and KK_E(d:chan,k:chan) = do !d; KK() or !k; KK_P()
and KK_P() = do ?a4(k,d); KK_E(d,k) or ?a5(d,k); KK_P_E(d,k)
and KK_P_E(d:chan,k:chan) = do !d; KK_P() or !k; KK_PP()
and KK_PP() = (
  new d7@rd7:chan new k7@rk7:chan
  new d9@rd9:chan new k9@rk9:chan
  do ?a6(k,d); KK_P_E(d,k)
  or !a7(d7,k7); (do ?d7; KK_PP() or ?k7; KK_PP())
  or !a9(d9,k9); (do ?d9; KK_PP() or ?k9; KK_PP())
)
let K() = ?a7(d,k); K_E(d,k)
and K_E(d:chan,k:chan) = do !d; K() or !k; K_P()
and K_P() = do ?a8(k,d); K_E(d,k)  or ?a9(d,k); K_P_E(d,k)
and K_P_E(d:chan,k:chan) = do !d; K_P() or !k; K_PP()
and K_PP() = ?a10(k,d); K_P_E(d,k)

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 100 of (KKK() | KK() | K())
run ( E2() | KKPase() | KPase() | E1())
```

Fig. 5. Program Code for Mapk Cascade