A Visual Process Calculus for Biology

Andrew Phillips

Microsoft Research
7 JJ Thomson Avenue
CB3 0FB Cambridge UK

Abstract. This chapter presents a visual process calculus for designing and simulating computer models of biological systems. The calculus is based on a graphical variant of stochastic pi-calculus, extended with mobile compartments, and the simulation algorithm is based on standard kinetic theory of physical chemistry. The calculus forms the basis of a formal visual programming language for biology. The basic primitives of the calculus are first introduced by a series of examples involving genes and proteins. More complex features of the calculus are then illustrated by examples involving gene networks, cell differentiation, and immune system response. The main benefit of the calculus is its ability to model large systems incrementally, by directly composing simpler models of subsystems. The formal nature of the calculus also facilitates mathematical analysis of models, which in future could help provide insight into some of the underlying properties of biological systems.

1 Introduction

In many respects, biological systems are like massively parallel, highly complex, error-prone computer systems. The genetic code of an entire organism is stored in digital form as DNA, but instead of a binary code there are four letters, G,A,T,C. A specialised protein travels along the DNA to read the code, like reading a sequence of instructions in a program. The code itself is divided into genes with different functions, in the same way that a computer program is divided into functional modules. The genes produce RNA which is more accessible, like loading a module into memory. The RNA produces proteins, which fold into different shapes to perform the essential functions of a living organism. Some of these proteins can interact with the DNA to switch on different genes at different times, in the same way that some modules of a computer program can trigger the execution of other modules. In general, the genetic code can be viewed not as a single program, but rather as a *library* of programs that are executed on demand in response to environmental signals [22].

The Human Genome Project had the ambitious goal of mapping out the complete genetic code in humans. Scientists had hoped that by listing this code they would be able to unravel the mysteries of how the human body functions. Instead, the code raised many more questions than answers. It was then that the field of Systems Biology rose to the occasion, with the ultimate goal of being able to understand and predict the behaviour of biological organisms. Systems Biology can be broadly divided into two complementary approaches. On the one hand scientists are doing experiments in the lab and studying the results, in order to infer key properties of biological systems. On the other hand they are using this knowledge to build detailed models of systems and then testing these models in the lab. Such models are a powerful tool, allowing scientists to simulate a range of experiments on a computer before testing the most promising ones, saving resources and enabling more effective research. They also help to clarify the mechanisms of how a biological system functions, and are beginning to play a role in understanding disease.

In recent years we have witnessed a rise in the publication of biological models, but as these models grow in size they are becoming increasingly difficult to understand, maintain and extend. Many published models consist of hundreds of reactions, but as our knowledge of biological systems continues to increase, models consisting of tens of thousands of reactions will soon be commonplace. Clearly, writing a model of this size as a single list of reactions is not a scalable solution, for the same reasons that we would not write a large software program as a single list of thousands of instructions. Rather, we need a way of decomposing a large biological model into a collection of smaller, more manageable building blocks. Since the early days of computer programming, many high-level languages have successfully been developed to improve program modularity. But biological programs differ from traditional computer programs in two fundamental ways: first, they are massively parallel and second, each instruction has a certain probability of executing, so we are never quite sure what

will happen next. These fundamental differences suggest a need for specialised programming languages for biology.

Over the years, there has been considerable research on programming languages for complex parallel computer systems. Some of this research is also applicable to biological systems, which are typically highly complex and massively parallel. One example of such a language is the pi-calculus [18,33], originally proposed by Milner, Parrow and Walker [19] to model communicating computer networks that can dynamically reconfigure themselves over time. A stochastic extension was later introduced by Priami [28], to model the performance of computer networks in the presence of communication delays. More recently, the stochastic pi-calculus has been used to model and simulate a range of biological systems [17,28], an approach pioneered by Regev, Silverman and Shapiro [30]. In addition, a graphical variant of the calculus has been defined [26], to help make the calculus more broadly accessible. The stochastic pi-calculus allows the components of a biological system to be modelled independently, rather than modelling the individual reactions, enabling large models to be constructed by direct composition of simple components [1]. The calculus also facilitates mathematical analysis of systems using a range of established techniques, which in future could help provide insight into some of the underlying properties of biological systems. Various stochastic simulators have been developed for the calculus, such as [28,25], which are used to perform virtual experiments on biological 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*.

This chapter presents the SPiM calculus, a visual process calculus for designing and simulating computer models of biological systems. The calculus is based on a graphical variant of the stochastic pi-calculus, originally presented in [26], extended with mobile compartments. The basic primitives of the calculus are first introduced by a series of examples involving genes and proteins. More complex features of the calculus are then illustrated by examples involving gene networks, cell differentiation, and immune system response. All of the biological models in this chapter were simulated using the SPiM simulator [25] and are available for download online, together with the simulator itself[1].

1.1 SPiM Processes and Chemical Reactions

The Gillespie algorithm [13] was introduced in the 1970s for the exact stochastic simulation of coupled chemical reactions, and since then a vast number of chemical and biological models have been simulated using this approach. Even today, many biological models continue to be published as collections of chemical reactions, and the format is supported by most modelling standards such as the Systems Biology Markup Language [15].

Essentially, the SPiM calculus allows a system of chemical reactions to be rewritten as a collection of modular processes, which interact with each other by message-passing. Fig. 1 presents a comparison between chemical reactions and

[1] http://research.microsoft.com/spim

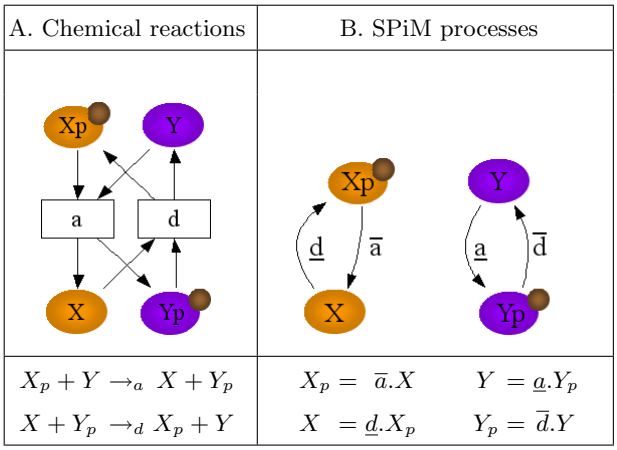


Fig. 1. Comparison between chemical reactions (A) and SPiM processes (B) for a model of protein activation.

SPiM processes for a simple model of protein activation. For both models, the graphical representation at the top is equivalent to the textual representation at the bottom. The chemical reaction model of Fig. 1A is constructed by listing the individual *reactions* in the system. An active protein X_p can interact with a protein Y through an a reaction, to produce a protein X and an active protein Y_p. The reverse reaction d can also occur. For the graphical representation, each shape represents a protein in a particular state and each box represents a reaction, with inbound edges (arcs) from reactants and outbound edges to products. In contrast, the SPiM model of Fig. 1B is constructed by describing the behaviour of the individual *components* in the system. This is achieved by splitting the reaction a into two complementary actions, a send $\overline{a}$ and a receive $\underline{a}$, and similarly for the reaction d. Thus, an active protein X_p can send on $\overline{a}$ and evolve to a protein X, which can receive on $\underline{d}$ and evolve to X_p. Similarly, a protein Y can receive on $\underline{a}$ and evolve to an active protein Y_p, which can send on $\overline{d}$ and evolve to Y. For the graphical representation, each shape represents a protein in a particular state and each connected graph represents the set of possible states of a given protein, where a labelled edge represents an action that the protein can perform in order to change from one state to another. Since X_p can send on $\overline{a}$ and Y can receive on $\underline{a}$, the two proteins can interact with each other and evolve to a new state simultaneously. The SPiM model explicitly represents the fact that the X_p protein evolves to X and the Y protein evolves to Y_p after the interaction takes place. This contrasts with the chemical reaction model, which does not explicitly state which product comes from which reactant. Although we can guess by looking at the reactant and product names, we could equally well interpret the reactions to mean that X_p becomes Y_p and Y becomes X. Another feature of the SPiM model is that the description of the X protein can

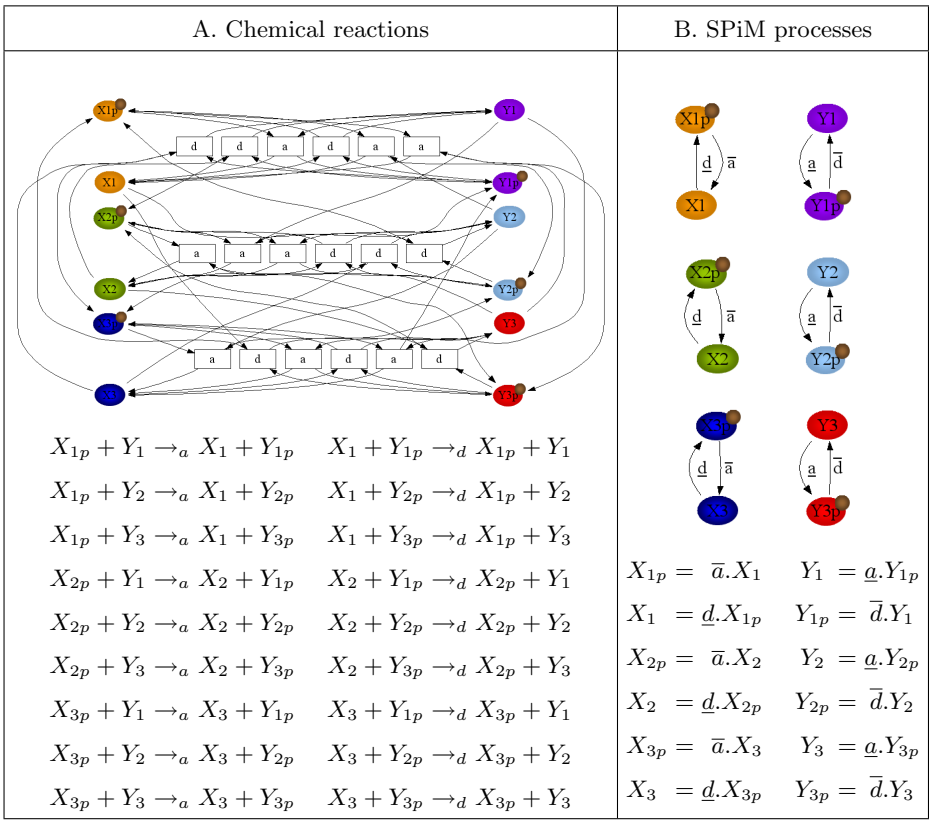


Fig. 2. Comparison between chemical reactions and SPiM processes for a combinatorial model of protein activation. (A) For the chemical reaction model there are 18 reactions, 9 forward reactions between X_{ip} and Y_j and 9 reverse reactions between X_i and Y_{jp} with $i, j \in \{1, \ldots, 3\}$. In the general case of $i \in \{1, \ldots, N\}$ and $j \in \{1, \ldots, M\}$ there will be $N \cdot M$ forward reactions and $N \cdot M$ reverse reactions. (B) For the SPiM model there are 12 actions, 6 forward actions for X_{ip}, Y_j and 6 reverse actions for X_i, Y_{jp} with $i, j \in \{1, \ldots, 3\}$. In the general case of $i \in \{1, \ldots, N\}$ and $j \in \{1, \ldots, M\}$ there will be $N + M$ forward actions and $N + M$ reverse actions.

be given independently of that of the Y protein, allowing the behaviours of the two proteins to be modelled separately. A notion of complementary send $\overline{a}$ and receive $\underline{a}$ is needed in order to decompose the system into individual modules in this way. If we simply wrote $X_p = a.X$ and $Y = a.Y_p$ and allowed all the a actions to interact, there would be nothing to prevent two X_p proteins from interacting with each other, which should not be possible. The use of complementary actions in the model is also consistent with the physical characteristics of the system, since the protein X_p actually sends its phosphate group p to a receiver protein Y, which then becomes phosphorylated as Y_p. Conversely, the

phosphorylated protein Y_p sends its phosphate group p back to protein X. More generally, complementary actions can be used to represent a range of biological interactions, such as two proteins with complementary binding sites, two RNA strands with complementary sequences, two atoms with complementary electron configurations, etc.

In addition to improving modularity, the SPiM calculus can also help to reduce the size of a biological model. Consider the models in Fig. 2, which describe a collection of proteins that activate each other. For both models, the graphical representation at the top is equivalent to the textual representation at the bottom. Each of the proteins $X_{1p}, \ldots, X_{3p}$ can activate each of the proteins $Y_1, \ldots, Y_3$, and similarly for the proteins $Y_{1p}, \ldots, Y_{3p}$ and $X_1, \ldots, X_3$. For the chemical reaction model in Fig. 2A, each interaction needs to be described explicitly, resulting in a combinatorial explosion in the number of reactions. For the SPiM model in Fig. 2B, instead of explicitly stating which proteins interact with which other proteins, it is only necessary to describe the channels on which each protein can send and receive. Implicitly, all the senders on a given channel can interact with all the receivers on the same channel, resulting in a more compact model whose size varies linearly with the number of proteins. As the system grows in size, the advantages of a modular representation become more noticeable, since the behaviours of individual proteins can be modified without changing the rest of the model.

2 Basic Examples

This section presents the basic constructs of the SPiM calculus, through a collection of simple examples. The examples are presented at the level of interacting genes and proteins, but the constructs of the calculus can equally well be used at the level of interacting cells or organisms. The simulation results for the examples were obtained using the SPiM simulator, which executes a SPiM model according to an exact simulation algorithm [25] based on standard principles of physical chemistry [13].

2.1 Protein Production

The first example models the production of protein P from a gene G, as shown in Fig. 3. In the SPiM model of Fig. 3A, the graphical representation at the top is equivalent to the textual representation at the bottom. The gene G can do a *produce* action, after which a new protein P is executed in parallel with the gene, written $(G \mid P)$. Graphically, the parallel execution is represented as a box with outbound edges to G and P. The protein P can then do a *degrade* action, after which the empty process is executed, written $\mathbf{0}$. Graphically, the empty process is represented as a node with a dotted outline. In order to obtain the simulation results of Fig. 3B, the *produce* and *degrade* actions are associated with corresponding rates given by 0.1 and 0.001, respectively. Each rate characterises an exponential distribution, such that the probability of an action

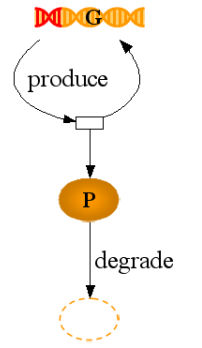
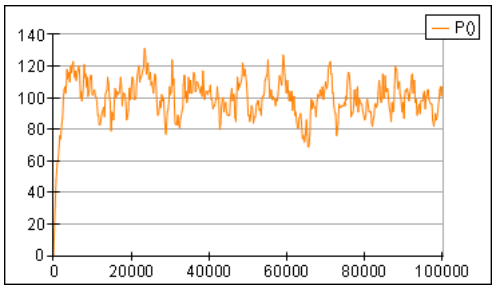
A. SPiM model	B. Simulation results
 $G = produce.(G \mid P)$ $P = degrade.\mathbf{0}$	 Simulating G with $produce = 0.1$, $degrade = 0.001$

Fig. 3. A SPiM model of protein production (A), with associated simulation results (B).

with rate r occurring within time t is given by $F(t) = 1 - e^{-rt}$. The average duration of the action is given by the mean $1/r$ of this distribution. Thus, the *produce* action will take on average 10 time units, while the degrade action will take on average 1000 time units. In this example the time units are assumed to be seconds, but in general they can be any unit that is appropriate for the system under consideration. The simulation results show the population of protein P over time. The population of the gene G is always 1 and is not shown. Initially there is one gene G and no protein P in the system. The number of proteins increases over time until it reaches an equilibrium of about 100, and then continues to fluctuate around this equilibrium.

The simulation of the model in Fig. 3 is described in more detail in Fig. 4 and 5. The execution steps in Fig. 4 describe the evolution of the system over time, where the graphical representation at the top is equivalent to the textual representation at the bottom. Each frame represents a state of the system, where the populations of the gene G and protein P are indicated next to the corresponding nodes in the graph, and the next actions to be executed are highlighted in grey. The calculations in Fig. 5 show the *propensities* of the reactions at each step of the simulation. For the system of Fig. 3 there are two possible reactions, *produce* and *degrade*. The propensity $\mathcal{R}(produce)$ of the *produce* reaction is given by the number of genes G that can perform this action, written $[G]$, multiplied by the rate 0.1 of the action. Similarly, the propensity $\mathcal{R}(degrade)$ of the *degrade* reaction is given by the number of proteins P that can perform this action, written $[P]$, multiplied by the rate 0.001 of the action. In accordance with [13], the prob-

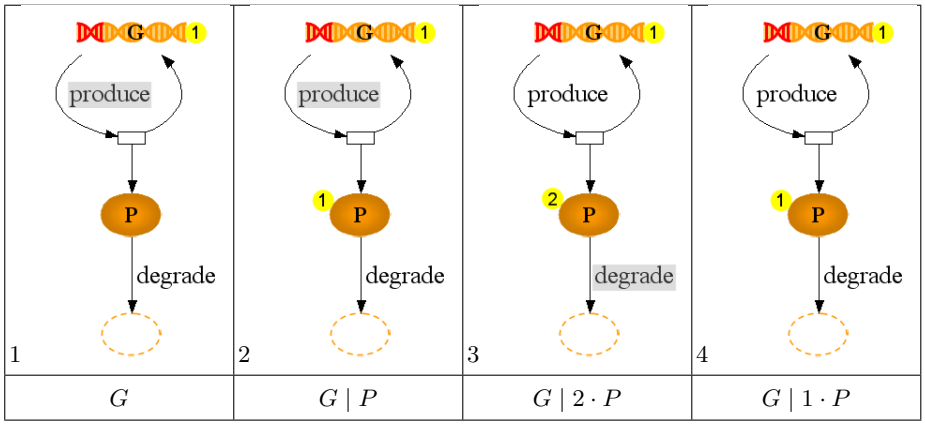


Fig. 4. Execution steps for the protein production model of Fig. 3. (1) Initially there is one gene G and no protein P. The gene then does a *produce* action. (2) A new protein is created in parallel with the gene, which then does a second *produce* action. (3) A second protein is created, and one of the proteins then does a *degrade* action. (4) One of the proteins is degraded.

#	$\mathcal{R}(produce) = [G] \cdot produce$			$\mathcal{R}(degrade) = [P] \cdot degrade$		
1	0.1	=	1·0.1	0	=	0·0.001
2	0.1	=	1·0.1	0.001	=	1·0.001
3	0.1	=	1·0.1	0.002	=	2·0.001
4	0.1	=	1·0.1	0.001	=	1·0.001
...	0.1	=	1·0.1	0.1	=	100·0.001

Fig. 5. Propensity calculations for the execution steps of Fig. 4, where the probability of a reaction is proportional to its propensity. (1) The system starts with a single gene G, which can produce a protein P with propensity 0.1. (2) A second protein can be produced with propensity 0.1. (3) One of the proteins can be degraded with propensity 0.002. Although the propensity of the *degrade* reaction is much lower than that of the *produce* reaction, there is still a small chance that a protein can degrade before another is produced. (...) Eventually an equilibrium is reached at around 100 proteins, where the propensity $[G] \cdot 0.1$ of the *produce* reaction is equal to the propensity $[P] \cdot 0.001$ of the *degrade* reaction.

ability of executing a given reaction is proportional to its propensity. In order to calculate the probability of each reaction at a given instant, the simulation algorithm adds up the propensities of all the possible reactions in the system and divides the propensity of each reaction by this total. The system reaches an equi-

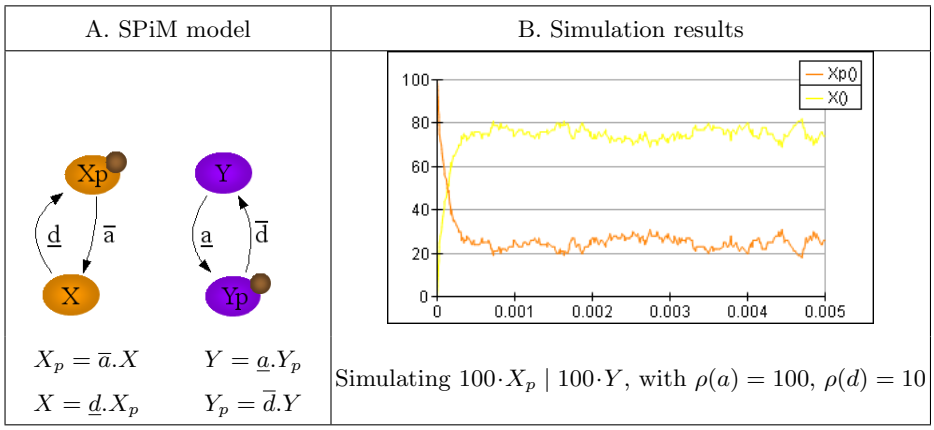


Fig. 6. A SPiM model of protein interaction (A), with associated simulation results (B).

librium when the propensity of the *produce* reaction is equal to the propensity of the *degrade* reaction.

2.2 Protein Interaction

The second example models the interaction between proteins, as shown in Fig. 6, where the SPiM model of Fig. 6A is the same as in Fig. 1A. An active protein X_p can send on $\overline{a}$ and evolve to a protein X, which can receive on $\underline{d}$ and evolve to X_p. Similarly, a protein Y can receive on $\underline{a}$ and evolve to an active protein Y_p, which can send on $\overline{d}$ and evolve to Y. In order to obtain the simulation results of Fig. 6B, the channels a and d are associated with corresponding rates given by $\rho(a) = 100$ and $\rho(d) = 10$, respectively, where the rate of the channel denotes the rate of a single interaction on this channel. As in the previous example, each rate r characterises an exponential distribution with mean $1/r$. Thus, an interaction on channel a will take on average 0.01 time units, while an interaction on channel d will take on average 0.1 time units. The simulation results show the populations of proteins X_p and X over time. The populations of Y and Y_p are not shown since they are equal to those of X_p and X, respectively. Initially there are $100{\cdot}X_p$ and $100 \cdot Y$ proteins. The number of X_p proteins decreases and the number of X proteins increases over time until the system reaches an equilibrium of about $24{\cdot}X_p$ and $76{\cdot}X$ proteins, and the populations then continue to fluctuate around this equilibrium.

The simulation of the model in Fig. 6 is described in more detail in Fig. 7 and 8. The execution steps in Fig. 7 describe the evolution of the system over time, and the calculations in Fig. 8 show the propensities of each reaction at each step of the simulation. There are two possible reactions in the system, a and d. The propensity $\mathcal{R}(a)$ of the a reaction is given by the number of possible interactions

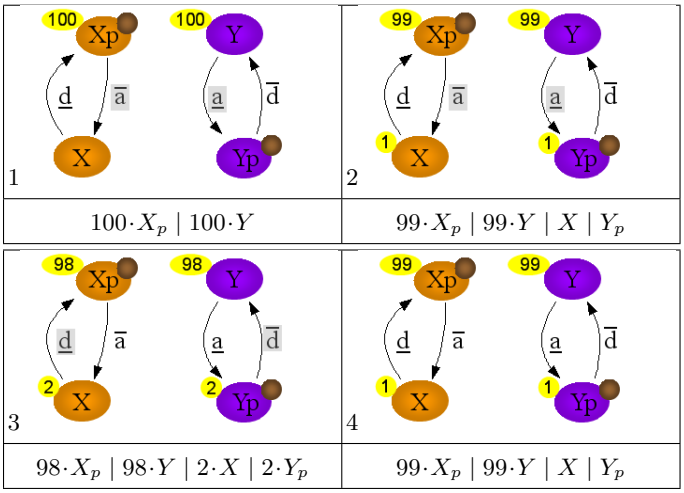


Fig. 7. A sequence of execution steps for the protein interaction model of Fig. 6. (1) Initially there are $100 \cdot X_p$ and $100 \cdot Y$ proteins. An X_p and a Y protein interact on channel a. (2) The two interacting proteins are converted to X and Y_p, respectively. An additional X_p and Y protein interact on channel a. (3) An X and Y_p protein interact on channel d. (4) The two interacting proteins are converted back to X_p and Y, respectively.

#	$\mathcal{R}(a) = [X_p]\cdot[Y]\cdot\rho(a)$	$\mathcal{R}(d) = [X]\cdot[Y_p]\cdot\rho(d)$
1	$1000000 = 100\cdot100\cdot100$	$0 = 0\cdot0\cdot10$
2	$980100 = 99\cdot99\cdot100$	$10 = 1\cdot1\cdot10$
3	$960400 = 98\cdot98\cdot100$	$40 = 2\cdot2\cdot10$
4	$980100 = 99\cdot99\cdot100$	$10 = 1\cdot1\cdot10$
...	$57600 = 24\cdot24\cdot100$	$57760 = 76\cdot76\cdot10$

Fig. 8. Propensity calculations for the execution steps of Fig. 7, where the probability of a reaction is proportional to its propensity. (1) The system starts with $[X_p] = [Y] = 100$. An X_p and a Y protein can interact on channel a with propensity 1000000. (2) An additional X_p and Y protein can interact with propensity 980100. (3) An X and Y_p protein can then interact on channel d with propensity 40. (...) Eventually an equilibrium is reached at around $[X_p] = [Y] = 24$ and $[X] = [Y_p] = 76$, where the propensity $[X_p]\cdot[Y]\cdot100$ of the a reaction is roughly equal to the propensity $[X]\cdot[Y_p]\cdot10$ of the d reaction.

on channel a times the rate of a single interaction. Since each protein X_p can interact with each protein Y on channel a, the number of possible interactions on a is given by the population of X_p, written $[X_p]$, times the population of Y,

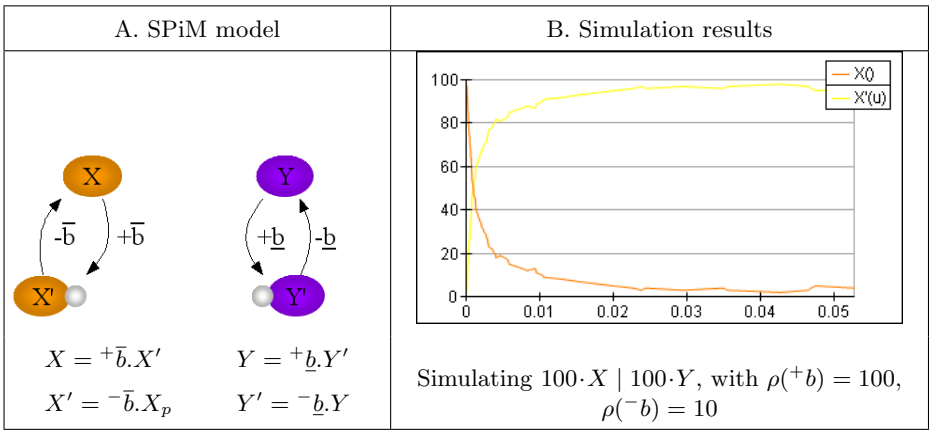


Fig. 9. A SPiM model of protein binding (A), with associated simulation results (B).

written $[Y]$. The rate of a single interaction is given by the rate of the channel a, written $\rho(a)$. Therefore, the propensity $\mathcal{R}(a)$ of the a reaction is given by $[X_p] \cdot [Y] \cdot \rho(a)$. Similarly, the propensity $\mathcal{R}(d)$ of the d reaction is given by $[X] \cdot [Y_p] \cdot \rho(d)$. The system reaches an equilibrium when the propensity of the a reaction is equal to the propensity of the d reaction.

2.3 Protein Binding

The third example models the binding and unbinding of proteins, as shown in Fig. 9. In the SPiM model of Fig. 9A, the graphical representation at the top is equivalent to the textual representation at the bottom. In order to model binding and unbinding reactions, we associate a given channel b with corresponding binding and unbinding channels, written ${}^{+}b$ and ${}^{-}b$, respectively. A protein X can send and bind on ${}^{+}\overline{b}$ and evolve to a protein X', which can send and unbind on ${}^{-}\overline{b}$ and evolve to X. Similarly, a protein Y can receive and bind on ${}^{+}\underline{b}$ and evolve to a protein Y', which can receive and unbind on ${}^{-}\underline{b}$ and evolve to Y. The send and receive actions on binding channel ${}^{+}b$ represent the fact that the interacting proteins become bound to each other after the interaction takes place. The bound proteins can then unbind from each other by interacting on unbinding channel ${}^{-}b$. In order to obtain the simulation results of Fig. 9B, the channels ${}^{+}b$ and ${}^{-}b$ are associated with corresponding rates given by $\rho({}^{+}b) = 100$ and $\rho({}^{-}b) = 10$, respectively. The simulation results show the populations of proteins X and X' over time. The populations of Y and Y' are not shown since they are equal to those of X and X', respectively. Initially there are $100 \cdot X$ and $100 \cdot Y$ proteins. The number of X proteins decreases and the number of X' proteins increases over time until the system reaches an equilibrium of about $3 \cdot X$ proteins and $97 \cdot X'$ proteins.

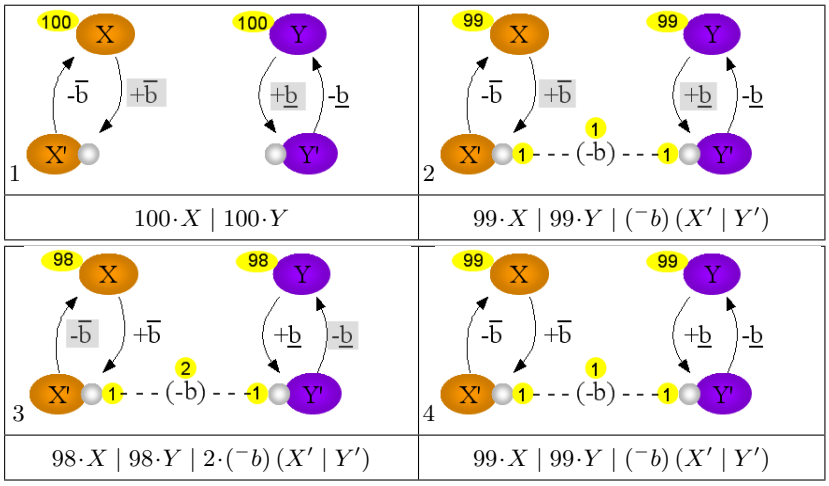


Fig. 10. A sequence of execution steps for the protein binding model of Fig. 9. (1) Initially there are $100 \cdot X$ and $100 \cdot Y$ proteins. An X and a Y protein bind to each other by interacting on channel ${}^{+}b$. (2) The interacting proteins are converted to X' and Y', respectively, and are bound to form a complex $({}^{-}b)\,(X' \mid Y')$. An additional X' and Y' protein bind to each other by interacting on channel ${}^{+}b$, resulting in 2 complexes $2{\cdot}({}^{-}b)\,(X' \mid Y')$. (3) An X' and a Y' protein in one of the complexes unbind from each other by interacting on the local channel ${}^{-}b$. (4) The two interacting proteins are converted back to X and Y, respectively.

#	$\mathcal{R}({}^{+}b) = [X]{\cdot}[Y]{\cdot}\rho({}^{+}b)$	$\mathcal{R}({}^{-}b) = [({}^{-}b)\,(X' \mid Y')]{\cdot}\rho({}^{-}b)$
1	$1000000 = 100{\cdot}100{\cdot}100$	$0 = 0{\cdot}10$
2	$980100 = 99{\cdot}99{\cdot}100$	$10 = 1{\cdot}10$
3	$960400 = 98{\cdot}98{\cdot}100$	$20 = 2{\cdot}10$
4	$980100 = 99{\cdot}99{\cdot}100$	$10 = 1{\cdot}10$
...	$900 = 3{\cdot}3{\cdot}100$	$970 = 97{\cdot}10$

Fig. 11. Propensity calculations for the execution steps of Fig. 10, where the probability of a reaction is proportional to its propensity. (1) The system starts with $[X] = [Y] = 100$. An X and a Y protein can bind by interacting on channel ${}^{+}b$ with propensity 1000000. (2) An additional X and Y protein can interact on ${}^{+}b$ with propensity 980100. (3) A pair of bound X' and Y' proteins can unbind by interacting on ${}^{-}b$ with propensity 20. Eventually an equilibrium is reached at around $[X] = [Y] = 3$ and $[({}^{-}b)\,(X' \mid Y')] = 97$, where the propensity $[X] \cdot [Y] \cdot 100$ of the ${}^{+}b$ reaction is roughly equal to the propensity $[({}^{-}b)\,(X' \mid Y')] \cdot 10$ of the ${}^{-}b$ reaction.

The simulation of the model in Fig. 9 is described in more detail in Fig. 10 and 11. There are two possible reactions in the system, ${}^{+}b$ and ${}^{-}b$. As with

the previous example, the propensity $\mathcal{R}(^{+}b)$ of the ^{+}b reaction is given by the number of possible interactions on binding channel ^{+}b times the rate of a single interaction, written $[X] \cdot [Y] \cdot \rho(^{+}b)$. After the interaction takes place, each X' protein is bound to a single Y' protein on a local channel ^{-}b, and the pair of bound proteins is written $(^{-}b)\,(X' \mid Y')$. The number of possible interactions on unbinding channel ^{-}b is given by the number of pairs of bound X' and Y' proteins, written $[(^{-}b)\,(X' \mid Y')]$. Therefore, the propensity $\mathcal{R}(^{-}b)$ of the ^{-}b reaction is given by $[(^{-}b)\,(X' \mid Y')]\cdot\rho(^{-}b)$. If we were instead to use a non-binding interaction on channel b, then every X' protein would be able to interact with every other Y' protein, and the equilibrium would be 24 X' and Y' proteins, similar to the previous example. But since each X' protein can only interact with a single Y' protein, a reaction between X' and Y' is much less likely to occur, and the equilibrium is 97 X' and Y' proteins. This illustrates how the use of binding interactions can significantly alter the equilibrium of a system.

2.4 Summary

In this section we introduced the basic primitives of the SPiM calculus, using examples of protein production, protein interaction and protein binding. The examples were presented in both graphical and textual forms, which could be used interchangeably.

The first example showed how a gene G could *produce* a new protein P in parallel with itself, written $G = \mathit{produce}.(G \mid P)$, and how a protein P could *degrade* to nothing, written $P = \mathit{degrade}.\mathbf{0}$. Both of these definitions gave the *behaviours* of the gene G and protein P. In order to run a simulation, initial *populations* were specified for processes G and P, and stochastic *rates* were associated to the *produce* and *degrade* actions, where the average duration of an action was defined as one over its rate. The *propensity* of each action was defined as the number of processes that can perform this action, times the rate of the action. At each step of the simulation, a given action was chosen with probability proportional to its propensity. An equilibrium was reached when the propensities of both actions were roughly the same.

The second example showed how a protein X_p could send on $\overline{a}$ and evolve to a protein X, written $X_p = \overline{a}.X$, and how a protein Y could receive on $\underline{a}$ and evolve to a protein Y_p , written $Y = \underline{a}.Y_p$. Since these two proteins could send and receive on the same channel, they could interact on this channel and evolve simultaneously. The reverse interaction could also occur on channel d, where $X = \underline{d}.X_p$ and $Y_p = \overline{d}.Y$. Initial populations were specified for processes X and Y_p, and stochastic rates were associated to the channels a and d, where the average duration of a channel interaction was defined as one over the channel rate. The propensity of each channel interaction was defined as the number of sends on the channel, times the number of receives on the channel, times the rate of the channel. At each step of the simulation, a given channel interaction was chosen with probability proportional to its propensity. An equilibrium was reached when the propensities of both channel interactions were roughly the same.

The third example introduced a notion of binding and unbinding channels, written ${}^{+}b$ and ${}^{-}b$, respectively. The example showed how a protein X could send and bind on ${}^{+}\overline{b}$ and evolve to a protein X', written $X = {}^{+}\overline{b}.X'$, and how a protein Y could receive and bind on ${}^{+}\underline{b}$ and evolve to a protein Y', written $Y = {}^{+}\underline{b}.Y'$. After the binding interaction on channel ${}^{+}b$ the two proteins became bound to each other, written $({}^{-}b)\,(X' \mid Y')$, and the bound proteins could then unbind by interacting on channel ${}^{-}b$, where $X' = {}^{-}\overline{b}.X$ and $Y' = {}^{-}\underline{b}.Y$. Initial populations were specified for processes X and Y, and stochastic rates were associated to channels ${}^{+}b$ and ${}^{-}b$. The key difference with the previous example was that each X' protein became bound to a single Y' protein on channel ${}^{-}b$. The propensity of an unbinding channel interaction was defined as the number of pairs bound by the channel, times the rate of the channel. The example illustrated how the use of binding interactions could alter the equilibrium of a system.

The above three examples illustrate some of the basic constructs of the SPiM calculus. More complex features of the calculus are illustrated in the following sections, and the full definition of the calculus is given in Appendix 7.

3 Gene Networks

3.1 Repressilator Model

One of the ultimate challenges of Systems Biology is to undertand in detail how living systems function. Once this has been achieved, in principle we should be able to engineer living systems to behave in predictable ways. Such engineering of biological systems could also pave the way for new medical therapies and treatments. In recent years, scientists have been investigating how to genetically engineer simple organisms such as bacteria, by inserting combinations of genes that interact with each other in predefined ways. For example, [11] presents the results of a well-known experiment, in which *E. coli* were genetically engineered to repeatedly glow on and off. A network of three genes was inserted into the bacteria, consisting of the genes *tet*, *lambda* and *lac*, which mutually repressed each other. More precisely, the *tet* gene produced proteins to block the *lambda* gene, which produced proteins to block the *lac* gene, which produced proteins to block the *tet* gene, completing the cycle. An additional *gfp* gene was inserted to produce a green fluorescent protein (GFP), which made the bacteria glow. The network was constructed so that the *tet* gene produced proteins that also blocked the *gfp* gene, thereby preventing the production of GFP. When engineered in live bacteria, the gene network caused the bacteria to repeatedly glow on and off. The resulting network was called the *Repressilator*, since the mutual repression of genes gave rise to oscillations in GFP levels. The gene network was also reproduced in the descendants of the bacteria, which themselves continued to glow on and off.

The Repressilator network can be simulated in SPiM by first constructing a parameterised model of a gene with negative control as shown in Fig. 12, based on previous work presented in [1] and [2]. The model is similar to the protein production model of Fig. 3, except that the gene and protein are parameterised

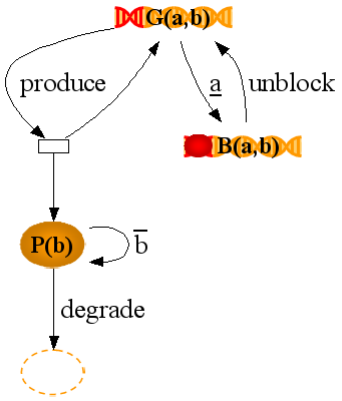
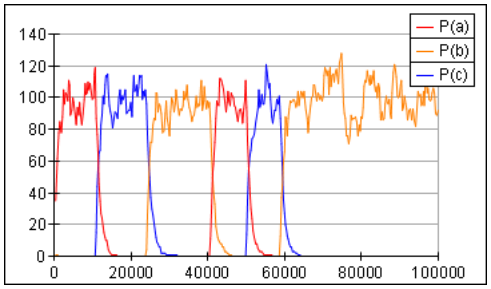
A. SPiM model	B. Simulation results
	
$G(a,b) = produce.(G(a,b) \mid P(b))$ $+ \underline{a}.B(a,b)$ $B(a,b) = unblock.G(a,b)$ $P(b) = degrade.\mathbf{0} + \overline{b}.P(b)$	Simulating $G(a,b) \mid G(b,c) \mid G(c,a)$, with $produce = 0.1$, $degrade = 0.001$, $unblock = 0.0001$ and $\rho(a) = \rho(b) = \rho(c) = 1$.

Fig. 12. A SPiM model of a parameterised gene with negative control (A), with associated simulation results (B).

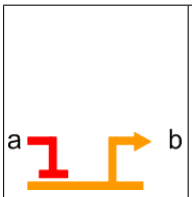
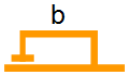
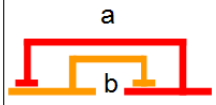
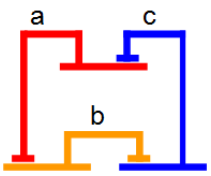
Gate	Auto-inhibitor	Bistable Switch	Oscillator
			
$G(a,b)$	$G(b,b)$	$G(a,b) \mid G(b,a)$	$G(a,b) \mid G(b,c) \mid G(c,a)$

Fig. 13. Modular construction of gene networks using the parameterised gene of Fig. 12.

by the channels on which they can interact. The gene G is parameterised by channels a, b, while the protein P is parameterised by channel b. The parameterised gene $G(a,b)$ can perform one of two actions: either it can *produce*, after which a new protein $P(b)$ is executed in parallel with the gene, or it can receive on $\underline{a}$ and become blocked to $B(a,b)$. The choice between production and blocking is written using a $(+)$ symbol. Graphically, this is represented by two outbound

edges from node $G(a,b)$, labelled with the *produce* and $\underline{a}$ actions, respectively. If the *produce* action is chosen then a new protein is produced in parallel with the gene. If the $\underline{a}$ action is chosen then the gene becomes blocked to $B(a,b)$ and is no longer able to produce. The blocked gene can then *unblock* in order to become active again. The parameterised protein $P(b)$ can also perform one of two actions: either it can *degrade*, after which the empty process is executed, or it can send on $\overline{b}$ and continue executing. Therefore, $G(a,b)$ denotes a gene that is blocked by receiving on $\underline{a}$, and that produces proteins which send on $\overline{b}$.

The use of complementary actions in the model is consistent with the physical characteristics of the system, since $G(a,b)$ denotes a gene with promoter region $\underline{a}$, which produces proteins $P(b)$ with binding site $\overline{b}$. When the promoter region is free, the gene is able to repeatedly produce proteins. When the promoter region $\underline{a}$ is blocked by a protein with a complementary binding site $\overline{a}$, the gene is no longer able to produce. The parameterised gene can be used to construct networks of arbitrary complexity, as illustrated in Fig. 13. An auto-inhibitory gene can be defined as $G(\underline{b},b)$, i.e. a gene with promoter region $\underline{b}$ that produces proteins with binding site $\overline{b}$. Thus, the proteins produced by the gene will in turn cause the gene to block. In this setting, the parameters (a,b) of the gene are replaced with actual channels (b,b), resulting in a gene that blocks itself. A bistable network can be defined as $G(a,b) \mid G(b,a)$, i.e. two genes that block each other. The first gene uses channels (a,b) as usual, while the second gene uses channels (b,a). Similarly, an oscillator network can be defined as $G(a,b) \mid G(b,c) \mid G(c,a)$.

The simulation results for the oscillator network of genes are shown in Fig. 12B, where the *produce*, *degrade* and *unblock* actions are associated with corresponding rates given by 0.1, 0.001 and 0.0001, respectively, and the channels a, b, c are associated with rate 1. The results show the populations of proteins $P(a)$, $P(b)$, $P(c)$ over time. Initially, about 100 $P(a)$ proteins are produced while the other two proteins are absent. Subsequently, about 100 $P(b)$ proteins are produced, which repress the production of $P(a)$. Eventually, about 100 $P(c)$ proteins are produced, which repress the production of $P(b)$, completing the cycle. The alternate production of proteins continues in this order until the end of the simulation. From the simulation results it is not immediately clear how the oscillations are initialised, or why they proceed in a specific order. To understand the mechanism by which the gene network is able to produce these oscillations, it is necessary to closely examine the interactions between genes and proteins over time.

3.2 Model Execution

The simulation of the model in Fig. 12 is described in more detail in Fig. 14 and 15. Fig. 14 shows the first few execution steps for the simulation results of Fig. 12, where the definitions of $G(a,b)$, $G(b,c)$ and $G(c,a)$ are expanded so that the behaviour of each gene is represented as a separate graph. The graph for each gene is simply a copy of the model in Fig. 12, but with parameters (a,b) being replaced by (a,b), (b,c) and (c,a), respectively. Fig. 15 shows the propensities of the different reactions for each of the execution steps of Fig. 14.

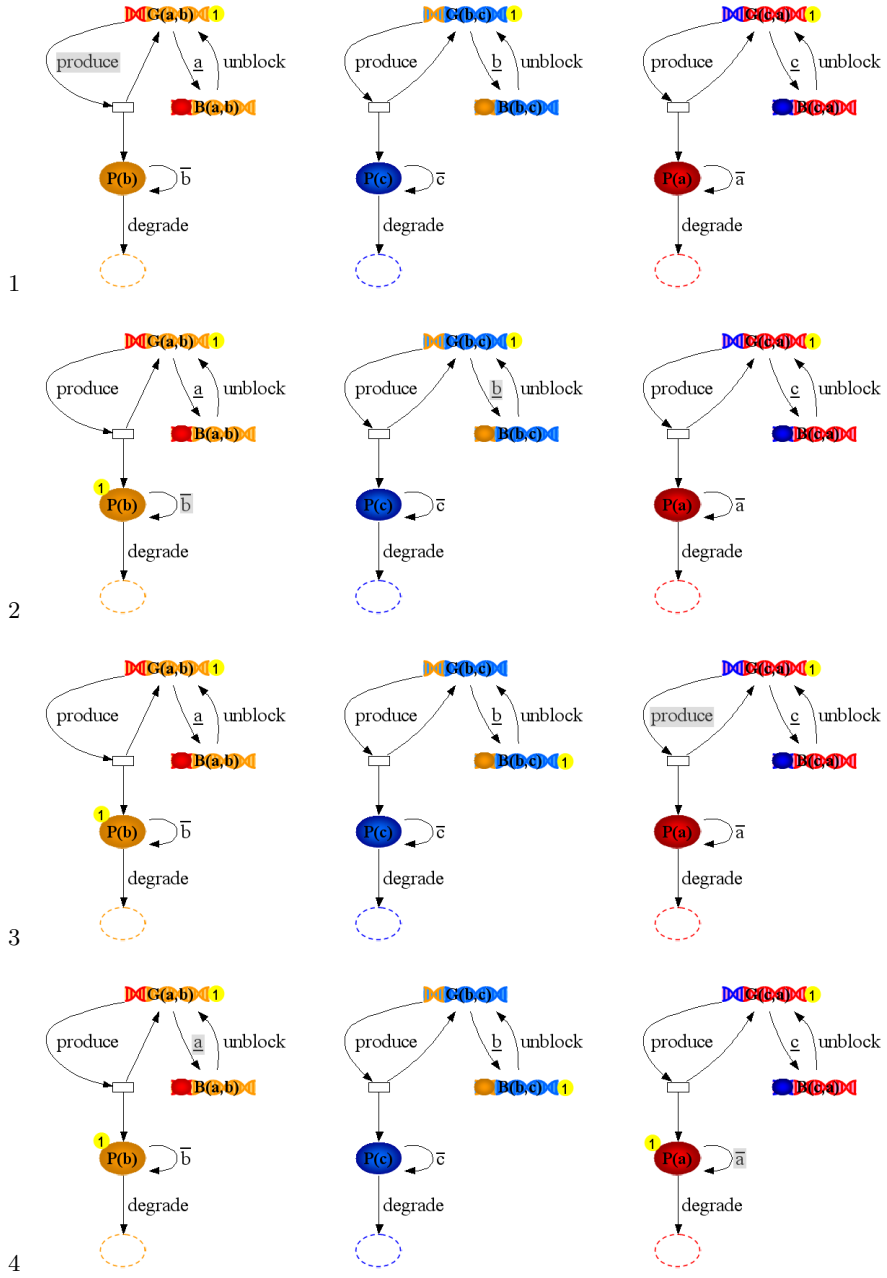


Fig. 14. A sequence of execution steps for the repressilator model of Fig. 12. (1) Initially there is one copy of each gene. The gene $G(a, b)$ produces a protein $P(b)$. (2) Protein $P(b)$ sends on $\overline{b}$ and causes $G(b, c)$ to block. (3) Gene $G(c, a)$ produces a protein $P(a)$. (4) Protein $P(a)$ sends on $\overline{a}$ and causes $G(a, b)$ to block.

#	$\mathcal{R}(produce)$ $= [G(a,b)]\cdot produce$ $+[G(b,c)]\cdot produce$ $+[G(c,a)]\cdot produce$	$\mathcal{R}(unblock)$ $= [B(a,b)]\cdot unblock$ $+[B(b,c)]\cdot unblock$ $+[B(c,a)]\cdot unblock$	$\mathcal{R}(degrade)$ $= [P(b)]\cdot degrade$ $+[P(c)]\cdot degrade$ $+[P(a)]\cdot degrade$	$\mathcal{R}(a)$	$\mathcal{R}(b)$	$\mathcal{R}(c)$
1	$0.1+0.1+0.1$	0	0	0	0	0
2	$0.1+0.1+0.1$	0	$0.001+0+0$	0	$1{\cdot}1{\cdot}1$	0
3	$0.1+0+0.1$	$0+0.0001+0$	$0.001+0+0$	0	0	0
4	$0.1+0+0.1$	$0+0.0001+0$	$0.001+0+0.001$	$1{\cdot}1{\cdot}1$	0	0
5	$0+0+0.1$	$0.0001+0.0001+0$	$0.001+0+0.001$	0	0	0
...	$0+0+0.1$	$0.0001+0.0001+0$	$0+0+100{\cdot}0.001$	0	0	0

Fig. 15. Propensity calculations for the execution steps of Fig. 14, where the propensity $\mathcal{R}(a)$ of the a reaction is defined as $[G(a,b)]\cdot[P(a)]\cdot\rho(a)$. The propensities $\mathcal{R}(b)$ and $\mathcal{R}(c)$ are defined in a similar fashion. (1) The simulation starts with one copy of each gene, which can each produce a protein with propensity 0.1, such that the total propensity of the *produce* reaction is 0.3. (2) A protein $P(b)$ is produced, which can block gene $G(b,c)$ with propensity 1. (3) The gene $G(b,c)$ becomes blocked, preventing the production of protein $P(c)$. But genes $G(c,a)$ and $G(a,b)$ can still each produce with propensity 0.1 (4) A protein $P(a)$ is produced, which can block gene $G(a,b)$ with propensity 1. (5) Now both $G(a,b)$ and $G(b,c)$ are blocked, with a very low propensity to unblock, leaving $G(c,a)$ to produce freely. (...) Eventually an equilibrium is reached between production and degradation of protein $P(a)$, where $\mathcal{R}(produce) = \mathcal{R}(degrade) = 0.1$. This represents the first protein cycle of the simulation.

The propensities demonstrate how the first protein cycle is produced by a sequence of high-probability reactions. There are three possible delay reactions in the system: *produce*, *degrade* and *unblock*. The propensity of the *produce* reaction is given by the sum of the production propensities of each of the genes. The production propensity of gene $G(a,b)$ is given by the population of the gene times the rate of production, written $[G(a,b)]\cdot produce$, and similarly for genes $G(b,c)$ and $G(c,a)$. Thus, the propensity of the *produce* reaction is given by $[G(a,b)]\cdot produce + [G(b,c)]\cdot produce + [G(c,a)]\cdot produce$. The choice of a particular gene to produce happens in two stages: first, the *produce* reaction is chosen and second, one of the active genes is chosen to produce with equal probability. The propensities of the *unblock* and *degrade* reactions are defined in a similar fashion. There are three possible interactions in the system, on channels a,b and c. The propensity of an interaction on channel a is given by the number of proteins $P(a)$ times the number of genes $G(a,b)$, times the rate of the channel, written $[P(a)]\cdot[G(a,b)]\cdot\rho(a)$. This essentially determines the propensity of gene $G(a,b)$ to be blocked by protein $P(a)$. The greater the population of repressor

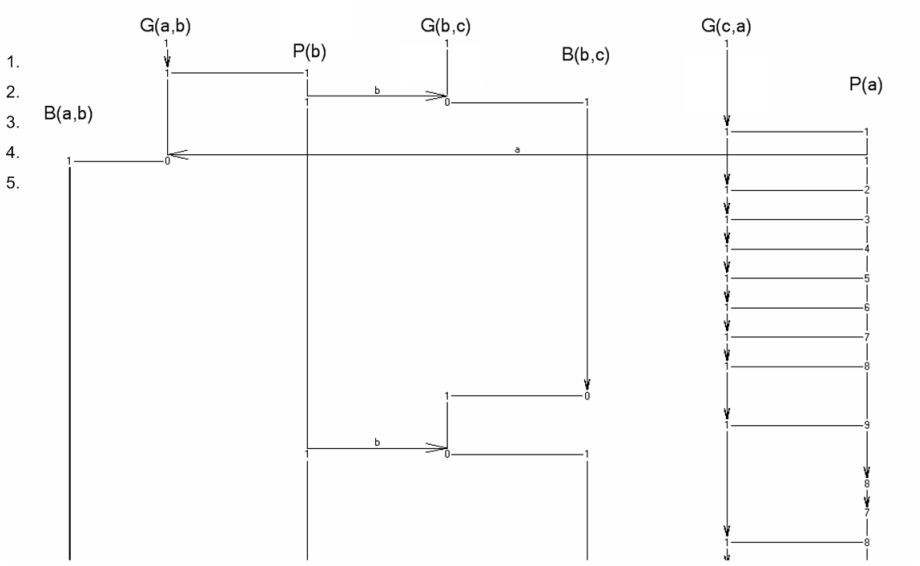


Fig. 16. A sequence chart of interactions between genes and proteins for the execution steps of Fig. 14. (1) Initially there is one copy of each of the genes $G(a, b)$, $G(b, c)$, $G(c, a)$. Gene $G(a, b)$ produces a protein $P(b)$ in parallel with itself. After the production, the populations of $G(a, b)$ and $P(b)$ are both 1. (2) Protein $P(b)$ sends on channel b and interacts with gene $G(b, c)$, which becomes blocked to $B(b, c)$. After the interaction, the populations of $P(b)$, $G(b, c)$ and $B(b, c)$ are 1, 0 and 1, respectively. (3) Gene $G(c, a)$ produces a protein $P(a)$ in parallel with itself. (4) Protein $P(a)$ sends on channel a and interacts with $G(a, b)$, which becomes blocked to $B(a, b)$. (5) Subsequently, $G(c, a)$ produces multiple proteins $P(a)$, some of which are degraded. In addition, $B(b, c)$ unblocks to $G(b, c)$, which is immediately blocked again by $P(b)$.

proteins, the greater the blocking propensity. The propensities of the interactions on channels b and c are defined in a similar fashion.

When examining the simulation results of Fig. 12 or the cartoon strip of Fig. 14, it is difficult to understand the causal chain of events leading up to the first protein cycle. In order to clarify the interactions that take place, together with the consequences of these interactions, Fig. 16 represents the execution steps of Fig. 14 as a *sequence chart*, based on an approach presented in [3]. In general, a chart represents a sequence of reactions from a single simulation run. Each vertical line in the chart represents a separate species, where time proceeds downwards. The vertical line is labelled with the current population of the species, which can evolve as the simulation progresses. Each horizontal line from one species to another represents a parallel composition of species. The line connects the populations of two or more species, and represents the fact that these populations are all executing simultaneously. Arrows are used to denote reactions that take place as the simulation progresses. Vertical arrows

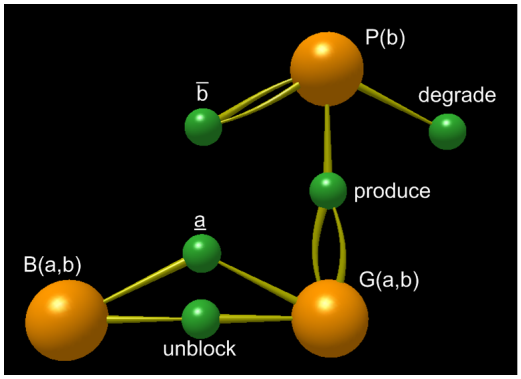


Fig. 17. A 3D representation of the gene with negative control from Fig. 12.

represent delay reactions, while horizontal arrows represent interactions between two species, where the arrow is labelled with the channel on which the interaction takes place. The chart shows how the different genes and proteins interact with each other to initiate the first protein cycle.

3.3 3D Visualisation

This section describes how the repressilator model of Fig. 12 can be simulated in three dimensions, in order to visualise the interactions between genes and proteins over time. The three-dimensional models are displayed using Network 3D[2]. Fig. 17 shows a 3D representation of a gene with negative control, based on the model in Fig. 12. The main difference with the 2D representation is that each action is represented as a node instead of as an edge. As a result, there are now two types of nodes in the model: *action nodes* and *species nodes*. Action nodes are shown in green and are of fixed size, while species nodes are of variable colour and size, where the volume of the node is proportional to the population of the species. The node labels are hidden by default, but can be revealed by clicking on individual nodes. There are also two types of edges: *transition edges* and *interaction edges*. Transition edges denote a transition from one species to another, and go from a species node to an action node, and then from an action node to zero or more species nodes. For example, the $\underline{a}$ action has one inbound edge from $G(a, b)$ and one outbound edge to $B(a, b)$, since the gene can transition from an active state to a blocked state by receiving on $\underline{a}$. The *produce* action has one inbound edge from $G(a, b)$ and two outbound edges to $G(a, b)$ and $P(b)$, since the gene can produce a new protein in parallel with itself. The *degrade* action has only one inbound edge from $P(b)$ and no outbound edges, since the protein disappears after the *degrade* action is executed. Interaction edges denote an interaction between two species, and go from an action node

[2] Network 3D is developed by Rich Williams of Microsoft Research.

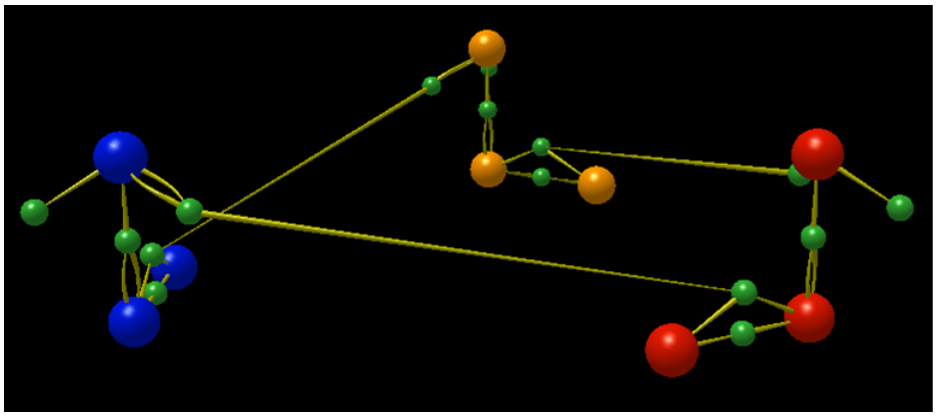


Fig. 18. A 3D representation of the repressilator model of Fig. 12, where the behaviour of each gene is represented as a separate subgraph.

that is sending on a given channel to a corresponding action node that is receiving on the same channel. There are no interaction edges in Fig. 17, since there are no corresponding send and receive actions on the same channel.

Fig. 18 shows a 3D representation of the repressilator model of Fig. 12, where the definitions of $G(a, b)$, $G(b, c)$ and $G(c, a)$ are expanded so that the behaviour of each gene is represented as a separate subgraph, similar to Fig. 14. There are three subgraphs in the figure, one for each gene, represented in orange, blue and red, respectively. The species nodes at the bottom of each subgraph represent active and blocked genes, which have a population of 0 or 1, while the species nodes at the top represent proteins, which have variable populations. The subgraphs are connected to each other via interaction edges, which go from the send action of a protein in one subgraph to the receive action of a corresponding gene in another subgraph. For example, there is an edge from the send action $\overline{c}$ of the blue protein to the receive action $\underline{c}$ of the red gene, representing the fact that the blue protein $P(c)$ can switch off the red gene $G(c, a)$.

Fig. 19 shows a sequence of 3D pictures from a simulation of the repressilator model of Fig. 12. The population of a species is visualised by dynamically altering the size of the corresponding species node. When the population is empty, the species node is obscured. The interactions between species are visualised by drawing an edge from a send action to a corresponding receive action on the same channel. When there are no senders or receivers the interaction edge is obscured. The 3D layout allows complex interactions to be visualised, while avoiding edge crossings that would have been unavoidable in 2D. The software can be used to rotate or zoom into the model, in order to focus on a subset of interactions and species. The 3D simulation is complementary to the 2D plots of species populations over time, since it allows the interactions between species to be visualised as the simulation progresses.

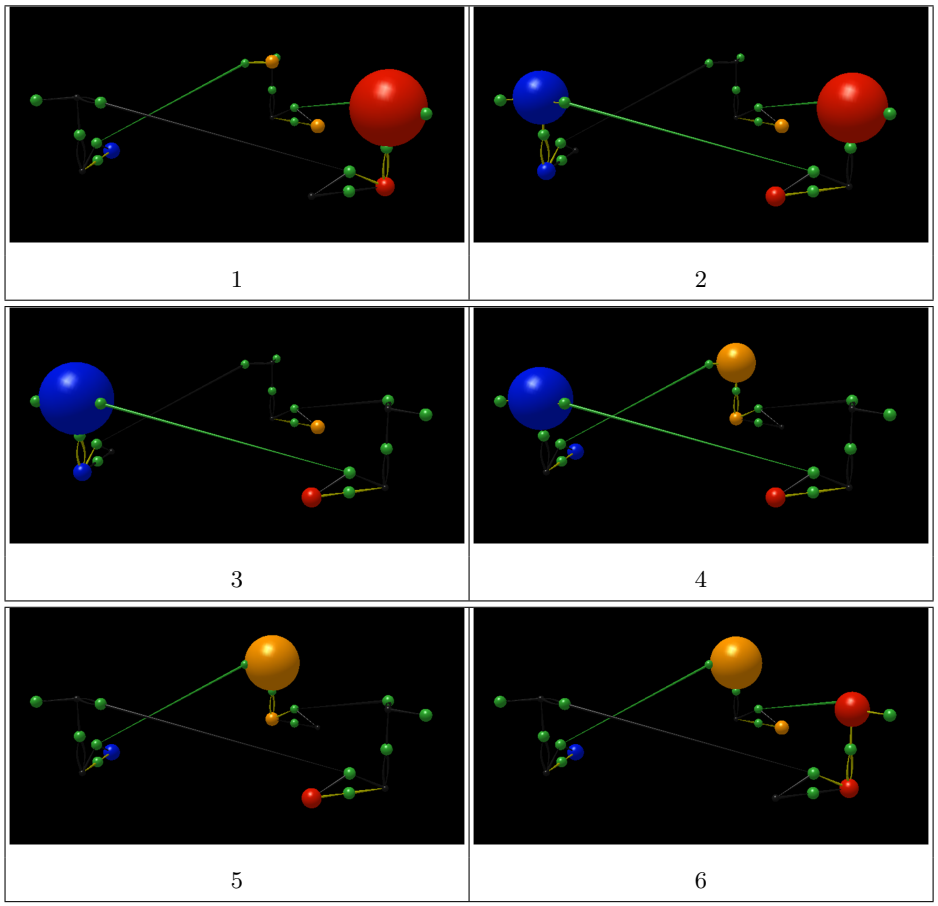


Fig. 19. A sequence of 3D pictures from a simulation of the repressilator model of Fig. 12. (1) Initially there is a large population of red proteins, which switch off the orange gene. (2) The blue gene then switches on and starts producing proteins, which switch off the red gene. (3) Since no more red proteins are produced, the population of red proteins slowly decreases over time, until all of the red proteins are degraded. (4) The orange gene then switches on and starts producing proteins, which switch off the blue gene. (5) Since no more blue proteins are produced, the population of blue proteins slowly decreases over time, until all of the blue proteins are degraded. (6) The red gene then switches on and starts producing proteins, which switch off the orange gene, completing the cycle.

3.4 Model Refinement

Once we understand how the repressilator model functions, we can further refine the model in order to improve the regularity of oscillations, as discussed in [2].

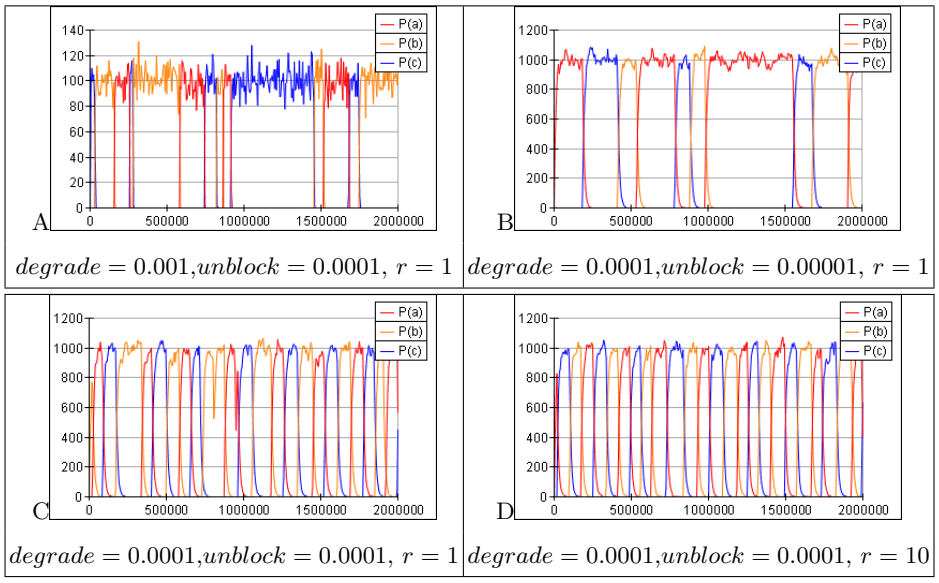
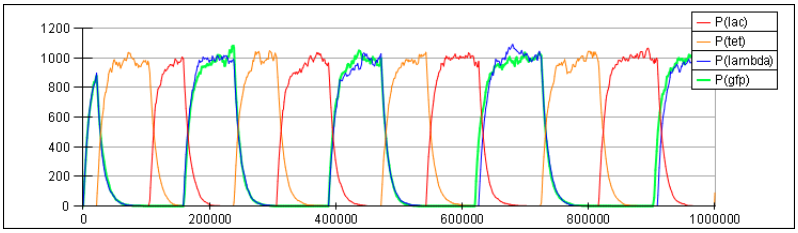


Fig. 20. Parameter variation for the repressilator model of Fig. 12, with $produce = 0.1$ and $\rho(a) = \rho(b) = \rho(c) = r$.

We can start by adjusting the rates of the model, as shown in Fig. 20. We observe that each protein cycle is characterised by a dominant protein whose population stabilises at an equilibrium between production and degradation, given by $\frac{produce}{degrade}$. In Fig. 20A the average population for a given protein cycle is about 100, but this fluctuates significantly due to stochastic noise. We can limit the relative size of the fluctuations by decreasing the degradation rate to 0.0001, resulting in a dominant population of about 1000, as shown in Fig. 20B. We observe that when one protein is dominant the other two proteins are absent and their corresponding genes are blocked, where one of the blocked genes is actively repressed. For example, when the blue protein is dominant both the red and orange proteins are absent, where the red gene is actively repressed and the orange gene is waiting to unblock. This is illustrated in step 3 of Fig. 19. The duration of protein cycles is highly irregular, since it depends mainly on the rate of unblocking of the unrepressed gene, which is relatively slow and is characterised by an exponential distribution. We can reduce this variability by increasing the rate of unblocking to $unblock = 0.0001$, as shown in Fig. 20C. In this setting, the gene unblocks more rapidly, but is immediately blocked again by any remaining repressors. The rapid unblocking means that once the last repressor has been degraded, the gene can become active soon after. Since the repressor degradation curve is fairly regular, we observe an increased regularity in the oscillations. Unfortunately, since the repressed gene is repeatedly trying to unblock, this also increases the likelihood that it will produce a protein before



$$G(lac, tet) \mid G(tet, lambda) \mid G(lambda, lac) \mid G(tet, gfp)$$

Fig. 21. Simulation results for the repressilator model of Fig. 12, with an additional GFP reporter, where $produce = 0.1$, $degrade = unblock = 0.0001$ and $r = 10$.

all of its repressors are degraded, which can result in large fluctuations in protein levels, as observed in Fig. 20C. We can compensate for this by increasing the rate of gene repression to $r = 10.0$, as shown in Fig. 20D. In this setting, even if there are only a few repressor proteins remaining, they will still have a high probability of blocking the corresponding gene, meaning that a gene will only become active once all of its repressors are degraded. This property, together with the regular degradation curve of repressors, gives rise to protein cycles of regular duration and amplitude.

If we were to implement a new version of the repressilator inside a bacterium, we might try to use genes and proteins whose reaction rates correspond roughly to those of Fig. 20D. The simulation results for such a repressilator are shown in Fig. 21, with genes for *tet*, *lambda* and *lac*, together with a reporter gene for *gfp*. Note that GFP is produced in absence of the *tet* protein, where the intermittent expression of GFP corresponds to the bacterium repeatedly glowing on and off.

In addition to adjusting the rates of the model in Fig. 12A, we can also refine the model to include transcription and translation steps explicitly, together with repressor binding and unbinding, as shown in Fig. 22A. The gene $G(a, b)$ can *transcribe*, after which a new mRNA strand $M(b)$ is produced in parallel. Alternatively, the gene can receive and bind on ${}^{+}\underline{a}$ and become blocked by a repressor, after which it can receive and unbind on ${}^{-}\underline{a}$ and release the repressor, becoming active once more. The mRNA strand $M(b)$ can *translate* and produce a new protein $P(b)$ in parallel. Alternatively, the strand can *degrade*. The protein $P(b)$ can send and bind on ${}^{+}\overline{b}$ and become bound to a gene, after which it can send and unbind on ${}^{-}\overline{b}$. Alternatively, the protein can *degrade*. Simulation results for the detailed model are shown in Fig. 22B. Note that the protein levels fluctuate much more widely than in Fig. 12B, since they amplify any small fluctuations that occur in the levels of mRNA. Although the detailed model is much more complex than the simplified model of Fig. 12, similar analysis can still be applied in order to understand how the model functions. However, further refinements to the model are required in order to reduce the fluctuations in mRNA levels. One way of reducing fluctuations is to introduce some form

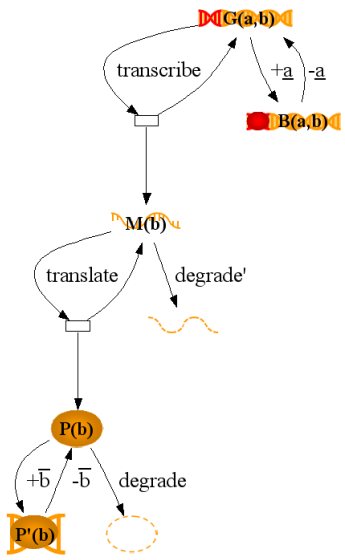
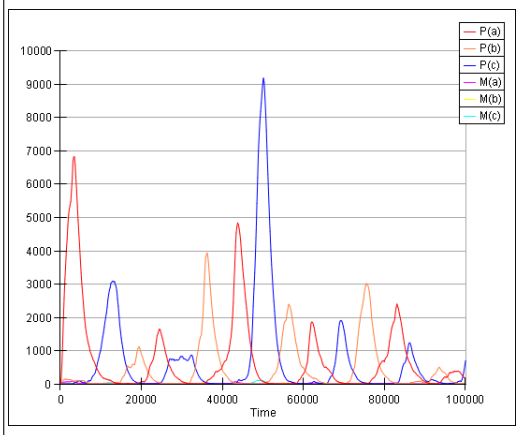
A. SPiM model	B. Simulation results
	
$G(a,b) = transcribe.(G(a,b) \mid M(b))$ $\quad + {}^{+}\underline{a}.B(a,b)$ $B(a,b) = {}^{-}\underline{a}.G(a,b)$ $M(b) = translate.(M(b) \mid P(b))$ $\quad + degrade'.\mathbf{0}$ $P(b) = {}^{+}\overline{b}.P'(b) + degrade.\mathbf{0}$ $P'(b) = {}^{-}\overline{b}.P(b)$	Simulating $G(a,b) \mid G(b,c) \mid G(c,a)$, with $transcribe = translate = 0.1$, $degrade = degrade' = 0.001$, $\rho({}^{+}a) = \rho({}^{+}b) = \rho({}^{+}c) = 1$ and $\rho({}^{-}a) = \rho({}^{-}b) = \rho({}^{-}c) = 0.5$.

Fig. 22. Refining the gene with negative control of Fig. 12 to include transcription, translation and repressor binding.

of cooperativity in the model, for example by allowing proteins to form dimers or even tetramers before repressing a gene. We omit the extensions here, but additional details can be found in [2]. The main point to note is that we can continually refine our parameterised model of a single gene without changing the structure of the overall gene network, which remains fixed at $G(a,b) \mid G(b,c) \mid G(c,a)$. This illustrates the modularity of our approach, where module definitions can be continually refined without changing the main program.

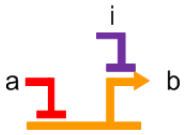
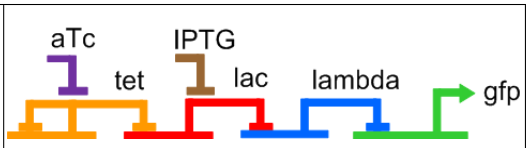
A. Gate with Inducer	B. D038 Logic Gate
	
$Gi(a, b, i)$	$Gi(tet, tet, aTc) \mid Gi(tet, lac, IPTG)$ $\mid G(lac, lambda) \mid G(lambda, gfp)$

Fig. 23. Modular construction of gene networks from two parameterised models of a gene. (A) A parameterised gene whose proteins are inhibited by an inducer i. (B) A gene network that behaves like a boolean logic gate. The proteins aTc and IPTG act as boolean inputs, while GFP acts as a boolean output. Depending on the presence or absence of inhibitors, the GFP levels are either high or low. When implemented in live bacteria, the presence or absence of inducers determines whether or not the bacteria will glow.

3.5 Genetic Logic Gates

A substantial extension to the original repressilator experiment is described in [14], in which a collection of logic gates were implemented in live bacteria. Rather than producing oscillations in GFP levels, the experiment aimed to construct gene networks that were able to function as boolean logic gates. The extension was achieved by introducing inducers aTc and IPTG to inhibit the *tet* and *lac* proteins, respectively. The presence or absence of the inducers represented the logical inputs to the network, while the presence or absence of GFP represented its logical output. In total, 125 different networks of 3 promoter-gene units were constructed, using a combination of 3 different genes and 5 different promoters. For each of these networks, the level of GFP was measured in presence or absence of aTc, IPTG, or both.

We can model these networks in the SPiM calculus by defining a parameterised gene in which the produced proteins are switched off by an inducer, as shown in Fig. 23A. The process $Gi(a, b, i)$ represents a gene that is blocked by receiving on $\underline{a}$, and that produces proteins which send on $\overline{b}$, where the produced proteins are inhibited by receiving on $\underline{i}$. We can use the gene $Gi(a, b, i)$ together with the gene $G(a, b)$ of Fig. 12 to construct the networks described in [14]. One of these networks is illustrated in Fig. 23B. The first gene in the network is inhibited by *tet* and produces *tet*, the second gene is inhibited by *tet* and produces *lac*, while the third gene is inhibited by *lac* and produces *lambda*. Finally, the network is fixed so that the *lambda* gene inhibits *gfp*, meaning that the bacteria glows in absence of *lambda*. The two inputs to the system are aTc and IPTG, where aTc inhibits the *tet* proteins and IPTG inhibits the *lac* proteins.

Fig. 24A shows the definition of the parameterised gene with inducers. The definition is similar to that of Fig. 12, except that the gene is parameterised by

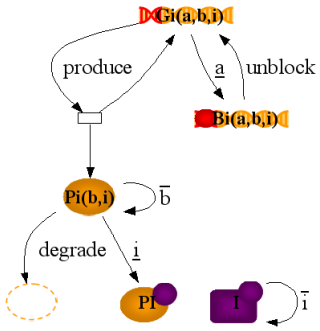
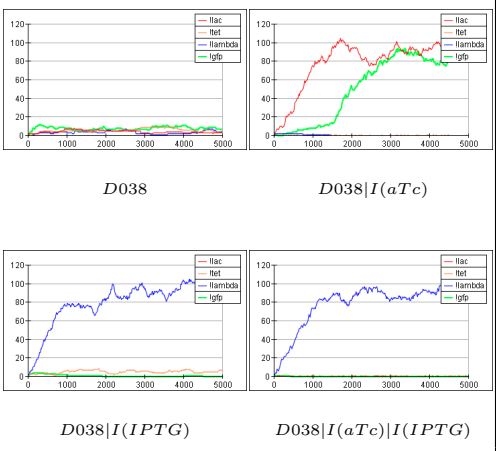
A. SPiM model	B. Simulation results
	 $D038$ $D038\|I(aTc)$ $D038\|I(IPTG)$ $D038\|I(aTc)\|I(IPTG)$
$Gi(a,b,i) = produce.(Gi(a,b,i) \mid Pi(b,i)) + \underline{a}.Bi(a,b,i)$ $Bi(a,b,i) = unblock.Gi(a,b,i)$ $Pi(b,i) = degrade.\mathbf{0} + \overline{b}.Pi(b,i) + \underline{i}.PI$ $I(i) = \overline{i}.I(i)$	Simulating the $D038$ network of Fig. 23, with $produce = 0.1$, $degrade = 0.001$, $unblock = 0.25$ and all channel rates set to 1. $D038$ is defined as $Gi(tet, tet, aTc) \mid Gi(tet, lac, IPTG) \mid G(lac, lambda) \mid G(lambda, gfp)$.

Fig. 24. A SPiM model of a parameterised gene with negative control and protein inhibition (A), with associated simulation results (B). The model is similar to that of Fig. 12, except that the gene is parameterised by an inducer i, which inhibits the produced proteins.

an inducer i, which inhibits the produced proteins. The protein $Pi(b, i)$ can block a gene by sending on $\overline{b}$, or it can be inhibited by receiving on $\underline{i}$. Alternatively, the protein can *degrade*. The behaviour of an inducer $I(i)$ is straightforward, in that it can inhibit a protein by sending on $\overline{i}$. Fig. 24B shows the simulation results for the gene network of Fig. 23B, where the green line represents the levels of GFP. Each plot represents a simulation of the network with different combinations of inputs. We observe that GFP is high in presence of $I(aTc)$ and in absence of $I(IPTG)$, which gives a logical characterisation of the network. The outcome of the simulation is not immediately obvious by looking at the network construction of Fig. 23, not least because of the presence of a *head feedback* loop, where one of the genes is repressed by the protein it produces. A more detailed explanation of the simulation results is given in [1]. The main point to note here

is that we can model a wide range of genetic logic gates using a small number of simple genetic building blocks.

4 C. elegans Development

The biological examples presented so far have all been modelled at the level of interacting genes and proteins. In this section we show how the SPiM calculus can also be used to model systems at the level of interacting cells. We consider an example from developmental biology, relating to the development of the egg-laying system of the *C. elegans* nematode. *C. elegans* has been used to study a number of fundamental cellular processes, and many breakthroughs relating to human physiology were first discovered in *C. elegans*. Examples include *genetic regulation of organ development and programmed cell death*, together with *RNA interference - gene silencing by double-stranded RNA*, both of which were awarded the Nobel prize in Physiology or Medicine in 2002 and 2006, respectively. More recently, the Nobel Prize in Chemistry was awarded *for the discovery and development of the green fluorescent protein, GFP*, for which a significant part of the research was carried out in *C. elegans*.

In this section we consider the phase of development in *C. elegans* when six Vulval Precursor Cells (VPCs) decide to adopt a primary, secondary or tertiary fate depending on their positions with respect to an Anchor Cell (AC). The decision of a precursor cell to adopt a particular fate is an important part of the development of the organism, which will ultimately determine whether or not the adult worm can successfully lay eggs. More details about the system can be found in [36,37,12] and a recent review is presented in [35]. Various models of VPC differentiation have been published recently, such as [9,12,21] to cite a few, in which the differentiation process has been highly simplified. Here we present an even simpler model, which illustrate some of the main factors involved during development. While many biological details are missing, our approach allows further model refinements to be made so that relevant details can be included incrementally.

Fig. 25 illustrates some of the main events that take place during normal differentiation of VPC cells in *C. elegans*. Once the cells have adopted their respective fates, they undergo further stages of proliferation and differentiation to develop into the egg-laying system of the adult worm.

4.1 Individual Cell Models

We start by constructing a simplified model of an anchor cell, together with a generic model of an undifferentiated precursor cell that can adopt one of three fates: primary, secondary or tertiary. The cell models are given in Fig. 26, where the graphical representation at the top is equivalent to the textual representation at the bottom. The anchor cell AC can repeatedly send on $\overline{ac}$, which represents a persistent inductive signal. A precursor cell can have four possible states, representing the undifferentiated cell V, the primary cell fate $V1$ and the secondary

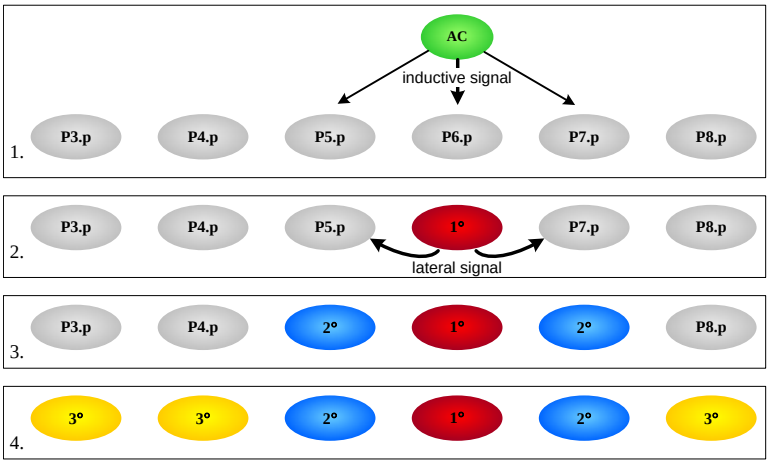


Fig. 25. Some of the main events that take place during normal differentiation of VPC cells in *C. elegans*. (1) Initially the six cells P3.p - P8.p are undifferentiated. The anchor cell sends an inductive signal, which is strongest for the cell P6.p directly in front of it. (2) The strong inductive signal causes cell P6.p to adopt a primary fate, after which it sends a lateral signal to its immediate neighbours. (3) The lateral signal causes cells P5.p and P7.p to adopt a secondary fate. (4) Eventually, the remaining cells adopt a tertiary fate.

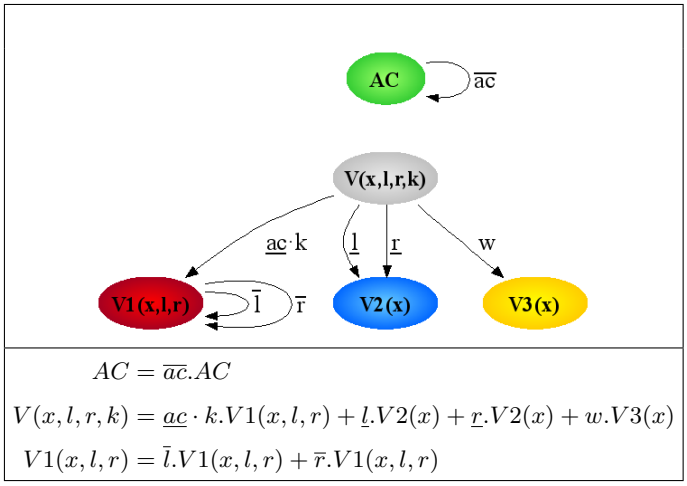


Fig. 26. A SPiM model of an Anchor Cell and a Vulval Precursor Cell, which are instrumental in the development of the egg-laying system of the *C. elegans* nematode. The undifferentiated precursor cell V can adopt one of three fates: a primary fate $V1$, a secondary fate $V2$ or a tertiary fate $V3$.

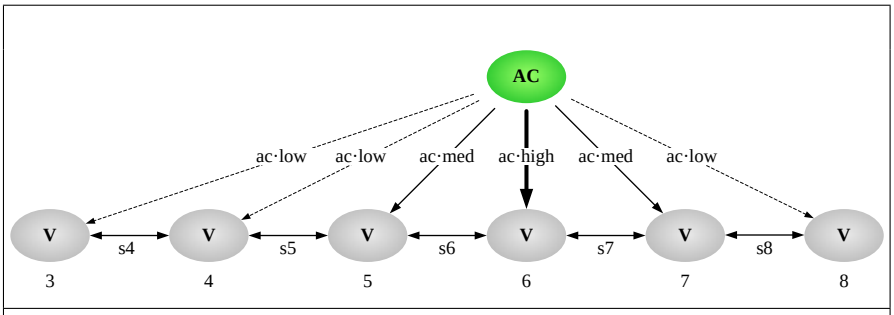


Fig. 27. System model of cell-cell interactions during VPC differentiation in *C. elegans*. The model is obtained by placing an anchor cell in parallel with six different VPC cells, using the generic model of a VPC cell described in Fig. 26.

and tertiary cell fates $V2$ and $V3$, respectively. The undifferentiated cell V is parameterised by its position x inside the worm, which is represented by a number between 3 and 8. Cells with successive numbers are considered to be adjacent to each other, and the cell at position 6 is considered to be directly opposite the anchor cell. The undifferentiated cell is also parameterised by a rate k, which represents the strength of its interaction with the anchor cell. Cells with a low k are considered far away and only interact weakly with the anchor cell, while cells with a high k are considered nearby and interact strongly with the anchor cell. For simplicity we assume only three levels for k: *low*, *medium* and *high*. Each undifferentiated cell is also characterised by its left and right neighbours, given by the channels l and r, which correspond to the lateral signals that a cell can use to communicate with its neighbours. The intuition is that each cell will interact with its left neighbour on channel l and with its right neighbour on channel r. We assume that once the precursor cells have adopted a particular fate, they no longer interact with the anchor cell. Initially, a precursor cell starts out undifferentiated as V. One of three things can then happen:

1. It can receive a signal on $\underline{ac}$ at rate k from the anchor cell, and adopt a primary fate $V1$. The primary cell will then send a persistent signal to its left and right neighbours on $\overline{l}$ and $\overline{r}$, respectively.
2. It can receive a signal from its left or right neighbour on $\underline{l}$ or $\underline{r}$, respectively, and adopt a secondary fate $V2$.
3. In absence of stimulus it can eventually adopt a tertiary fate $V3$, at rate w.

4.2 System Model

A model of the full developmental system is shown in Fig. 27, which consists of the anchor cell and the six VPC cells, where the graphical representation

at the top is equivalent to the textual representation at the bottom. Each of the VPC cells is an instance of the generic model of a single cell described in Fig. 26. The generic model is instantiated with different parameters depending on the position of the cell, the strength of its interaction with the anchor cell, and the neighbouring cells with which it can communicate. Cells are assumed to communicate with their left and right neighbours on dedicated channels, where the left channel of a given cell is equal to the right channel of its left neighbour. For example, the cell $V(6, s6, s7, high)$ at position 6 can communicate with its left neighbour on channel $s6$ and with its right neighbour on channel $s7$. It can also communicate with the anchor cell at a *high* rate, since it is assumed to be in close proximity to the anchor cell. Note that channels $s3$ and $s9$ in Fig. 27 are unused, since the cells to the left of position 3 and to the right of position 8 are not represented. The model can be further extended to represent these cells if necessary.

4.3 Simulation Results

In order to simulate the process of VPC differentiation, we assign approximate rates to the generic VPC model of Fig. 26 and to the overall system model of Fig. 27. The rates of *low*, *medium* and *high* inductive signalling are given by 0.0, 1.0 and 10.0, respectively, while the rate ls of lateral signalling is given by 1.0. We assume that the rate of lateral signalling is the same for all precursor cells, with the rates of channels $s4, \ldots, s8$ being equal to ls. The rate w for a cell to adopt a tertiary fate in the absence of external signals is given by 0.0001. Although these rates are entirely arbitrary, they give an initial approximation for the relative speeds of the various interactions that take place during development, and can be adjusted accordingly as new insight is gained from simulation and experimentation.

The simulation results of Fig. 28 show the total population of undifferentiated cells V and of differentiated cells $V1, V2, V3$ over time. Initially there are 6 undifferentiated cells, with one of the cells adopting a primary fate almost immediately. Two of the cells then adopt secondary fates in quick succession, and the remaining cells eventually adopt a tertiary fate. In order to determine which cells adopt which fates, we plot the number of cells $V, V1, V2, V3$ at each of the 6 positions as shown in Fig. 29, where each circle represents a precursor cell. The horizontal coordinate denotes the cell position, which ranges from 3 to 8. The colour denotes the cell fate, where an undifferentiated cell V is shown in grey, and a cell that adopts a primary, secondary or tertiary fate is shown in red, blue or yellow, respectively. The plots are generated automatically during the simulation, by using the parameter x in processes $V, V1, V2, V3$ as the horizontal coordinate for each process. As a given cell adopts a particular fate, the process $V(x, l, r, k)$ of each cell transitions to either $V1(x, l, r)$, $V2(x)$ or $V3(x)$, and the processes at each position x are plotted accordingly. In this way, the SPiM simulation generates a movie of cell differentiation over time. We can take snapshots of the simulation to separate out the individual steps, as shown in Fig. 29. For example, the first frame plots processes $V(3, s3, s4, low) \mid V(4, s4, s5, low) \mid$

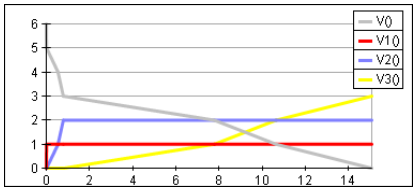


Fig. 28. Simulation results for the VPC model of Fig. 27. The results show the populations of undifferentiated, primary, secondary and tertiary cells over time.

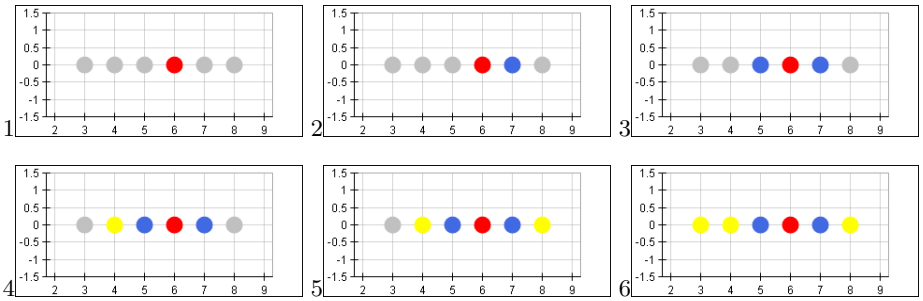


Fig. 29. Coordinate geometry plots for the simulation results of Fig. 28. The results show the change in cell fate over time for cells at positions 3 to 8, where the horizontal coordinate denotes the cell position. Initially all the cells start out as undifferentiated (not shown). The cell at position 6 then adopts a primary fate, after which the cells at positions 7 and 5 adopt a secondary fate. One by one the remaining cells then adopt a tertiary fate.

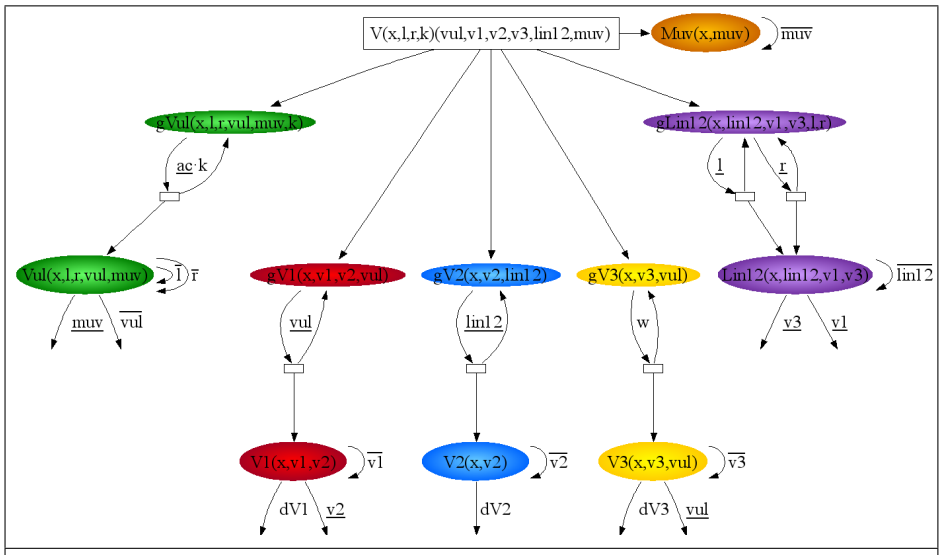
$V(5, s5, s6, med) \mid V1(6, s6, s7) \mid V(7, s7, s8, med) \mid V(8, s8, s9, low)$, with process $V1(6, s6, s7)$ at position 6 representing a primary fate and the remaining processes representing undetermined fates. As the simulation progresses, the processes interact with each other according to the behaviour outlined in Fig. 26, with the processes $V(7, s7, s8, med)$ and $V(5, s5, s6, med)$ transitioning to $V2(7)$ and $V2(5)$, respectively, followed by the remaining processes transitioning to $V3(4)$, $V3(8)$ and $V3(3)$.

We can interpret the simulation results based on the model construction and the chosen parameters. The precursor cell at position 6 adopts a primary fate by interacting with the anchor cell at rate $high = 10$. The average duration of this interaction is 0.1, which explains the rapid transition time observed in the simulation. Next, the primary cell at position 6 can signal to its left and right neighbours to adopt a secondary fate at rate $ls = 1.0$. The average duration of these transitions is 1, which is 10 times slower than the initial inductive signal to the cell at position 6. In absence of external signals, the remaining cells adopt a tertiary fate at rate $w = 0.0001$, which explains the length of time taken for these cells to differentiate.

Although the rates are chosen arbitrarily, we observe that if we increase the rate w then the cells are able to adopt a tertiary fate more quickly. Since the simulation is stochastic, this means that the cells at positions 5 to 7 will have a greater chance of adopting a tertiary fate, even in the presence of inductive and lateral signals, which could potentially result in abnormal development. Note also that the cells at positions 5 and 7 receive an inductive signal at rate 1, which is 10 times weaker than the inductive signal to the cell at position 6. Although the inductive signal is much weaker, there is still a one in ten chance for a cell at position 5 or 7 to adopt a primary fate, resulting in abnormal development. Indeed, if we simulate the model several times we observe that cells 5 or 7 adopt the wrong fate roughly 10 percent of the time. Clearly the VPC model is not satisfactory, since in reality the process of VPC differentiation is extremely robust. Simply changing the ratio of the *high* to *med* inductive signals to something like 1000 : 1 in order to improve robustness is not acceptable either, since in reality the strength of the inductive signal does not differ by such a large amount between neighbouring cells. Thus, although the high-level model we have presented can mechanistically reproduce some of the stages of VPC differentiation, it cannot reproduce the inherent robustness of the process, and more refined models are needed.

4.4 Refined Model

To illustrate our modelling approach, we describe a refined version of the VPC model of Fig. 26, based on the early results of [37]. The refined model is presented in Fig. 30. One of the simplifying assumptions of the initial model was to represent a change in cell fate by the activation of a single process inside the cell, representative for example of the activation of a single gene. In the refined model we now represent a change in cell fate by the accumulation of a particular protein, where $V1, V2, V3$ represent proteins with primary, secondary and tertiary functions, respectively. As with the simplified model, an undifferentiated cell V is parameterised by its position x, the strength k of its interaction with the anchor cell, and the channels l and r on which it can communicate with its left and right neighbours. As a result, the overall model of the VPC system remains the same as in Fig. 27, while the behaviour of an undifferentiated cell $V(x, l, r, k)$ is substantially refined. Instead of consisting of a single process that changes state over time, the cell V now consists of multiple processes running in parallel. The processes $gV1, gV2, gV3$ represent the genes for proteins $V1, V2, V3$, while the processes $gVul$ and $gLin12$ represent the genes for proteins Vul and $Lin12$. The genes are executed in parallel with each other inside the cell, alongside the Muv proteins. The gene $gVul$ produces proteins in the presence of a persistent inductive signal on channel ac, where the rate of production is determined by the parameter k. The Vul proteins then initiate a persistent communication with neighbouring cells by sending on $\overline{l}$ and $\overline{r}$. The Vul proteins can also induce the production of proteins $V1$ by sending on $\overline{vul}$. The proteins $V1$ are specific to cells that adopt a primary fate and can persistently send on $\overline{v1}$, which causes the $Lin12$ receptors to degrade. This prevents the cell from responding to its own



$$
\begin{aligned}
V(x,l,r,k) &= (vul, v1, v3, v3, lin12, muv) \\
&\quad (\ gVul(x,l,r,vul,muv,k) \mid gLin12(x,lin12,v1,v3,l,r) \\
&\quad \mid gV1(x,v1,v2,vul) \mid gV2(x,v2,lin12) \mid gV3(x,v3,vul) \mid Muv(x,muv)) \\
gVul(x,l,r,vul,muv,k) &= \underline{ac} \cdot k.(gVul(x,l,r,vul,muv,k) \mid Vul(x,l,r,vul,muv)) \\
gLin12(x,lin12,v1,v3,l,r) &= \underline{l}.(gLin12(x,lin12,v1,v3,l,r) \mid Lin12(x,lin12,v1,v3) \\
&\quad + \underline{r}.(gLin12(x,lin12,v1,v3,l,r) \mid Lin12(x,lin12,v1,v3) \\
gV1(x,v1,v2,vul) &= \underline{vul}.(gV1(x,v1,v2,vul) \mid V1(x,v1,v2)) \\
gV2(x,v2,lin12) &= \underline{lin12}.(gV2(x,v2,lin12) \mid V2(x,v2)) \\
gV3(x,v3,vul) &= w.(gV3(x,v3,vul) \mid V3(x,v3,vul)) \\
Vul(x,l,r,vul,muv) &= \overline{l}.Vul(x,l,r,vul,muv) + \overline{r}.Vul(x,l,r,vul,muv) + \overline{vul} + \underline{muv} \\
Lin12(x,lin12,v1,v3) &= \overline{lin12}.Lin12(x,lin12,v1,v3) + \underline{v3} + \underline{v1} \\
Muv(x,muv) &= \overline{muv}.Muv(x,muv) \\
V1(x,v1,v2) &= \overline{v1}.V1(x,v1,v2) + \underline{v2} + dV1 \\
V2(x,v2) &= \overline{v2}.V2(x,v2) + dV2 \\
V3(x,v3,vul) &= \overline{v3}.V3(x,v3,vul) + \underline{vul} + dV3
\end{aligned}
$$

Fig. 30. A refined SPiM model of the Vulval Precursor Cell of Fig. 26. The proteins $V1, V2, V3$ represent primary, secondary and tertiary functions which correspond to particular cell fates.

lateral signals on $\underline{l}$ and $\underline{r}$, thereby preventing it from adopting a secondary fate.

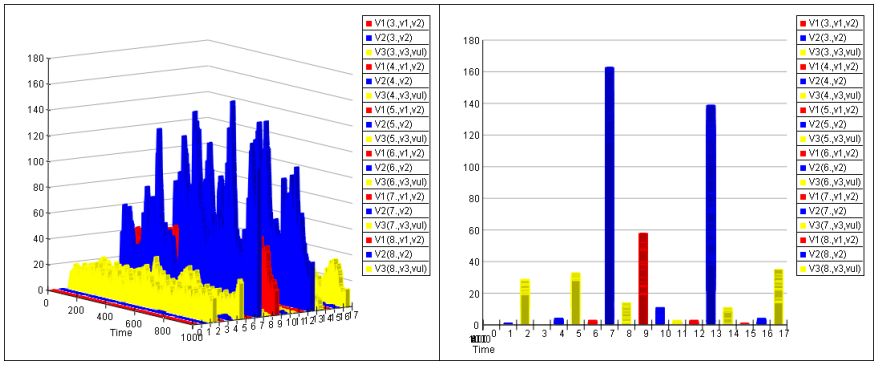


Fig. 31. Simulation results for the VPC model of Fig. 30, with $low = 0.01$, $med = 2$, $high = 10$, $v1 = 10$, $v2 = 0.1$, $v3 = 1$, $dV1 = 0.1$, $dV2 = dV3 = 0.05$ and all remaining rates set to 1. The results indicate the populations of processes $V1$, $V2$ and $V3$, which represent primary, secondary and tertiary fates, respectively, for each of the 6 cells. Both plots represent different views of the same simulation. The left plot represents a 3D view of the 18 different protein concentrations over time, 3 for each cell. The right plot gives another view of the same results, which displays the maximum populations for each of the proteins. Each cell has a dominant population of a particular protein, which corresponds to a particular cell fate.

Conversely, the gene $gLin12$ produces active receptor proteins when it receives a signal on $\underline{l}$ or $\underline{r}$ from its left or right neighbours. The $Lin12$ receptors send on $\overline{lin12}$ and cause the production of proteins $V2$, which are specific to cells that adopt a secondary fate. The proteins $V2$ also cause the degradation of proteins $V1$ by sending on $\overline{v2}$, thereby preventing the cell from adopting a primary fate. Finally, the gene $gV3$ is constitutively expressed at rate w and produces proteins $V3$ that are specific to cells with a tertiary fate. These proteins are in turn inhibited by receiving on $\underline{vul}$. Thus, there is a complicated network of interactions to ensure that the cells reliably adopt specific fates depending on the signals they receive. The interactions inside a particular cell take place on channels $vul, v1, v2, v3, lin12$ and muv. In order to ensure that the signals inside a given cell do not interfere with those inside neighbouring cells, the channels are *restricted* to each cell, represented by placing them in brackets inside the cell definition. It is this notion of restricted channels that enables boundaries between cells to be established in the model.

Initial simulation results for the VPC model of Fig. 30 are presented in Fig. 31. The results show the populations of processes $V1$, $V2$ and $V3$ for each of the 6 cells. Since these processes are parameterised by their location x, there are 18 different processes altogether. The first three lanes represent populations of $V1, V2, V3$ for the cell at position 3, the next three lanes represent populations of proteins for the cell at position 4, etc. We observe that the cell at position 6 adopts a primary fate, since it has a predominance of proteins $V1$, while the cells at positions 5 and 7 adopt a secondary fate, with a predominance of $V2$ proteins.

Finally, the cells at positions 3,4 and 8 adopt a tertiary fate, with a predominance of $V3$ proteins. Rather than making a decision after receiving a single signal from a neighbouring cell, repeated signalling is required for a cell to adopt a particular fate. This improves the robustness of the system, since the decision to adopt a particular fate is now made over a period of time. An important mechanism is the amplification of a relatively small difference in inductive signal between cells, so that each cell can reliably adopt the correct fate. Some of these issues are discussed in more detail in [9]. The main point we wish to make here is that we can start with a highly simplified model of a cell, and instantiate multiple copies of this model with different parameters to represent a system of communicating cells. We can then refine the model of a cell without changing the overall system definition. Additional details can be included in subsequent refinements, based on published results in [35,9,12,21]. This highlights the main motivation for our approach, which is to construct complex systems from simpler building blocks, and to be able to refine these building blocks without having to reconstruct the entire system.

5 SPiM with Compartments

The previous section described how the SPiM calculus can be used to model interactions between cells with fixed locations. In this section we describe how the calculus can be extended with a notion of *mobile compartments*, in order to model interactions between cells that can move relative to each other. The ambient calculus [5] by Cardelli and Gordon originally introduced a notion of mobile compartments to model computation carried out in mobile devices, together with mobile code that can move between devices. Since the original calculus, a range of variants have been introduced to model various aspects of security and mobility of distributed computer systems. More recently, the bioambient calculus [31] was introduced by Regev and colleagues to model molecular localisation and compartmentalisation in biological systems. In this section we present an extension of the SPiM calculus with compartments, which is essentially a graphical variant of the bioambient calculus.

5.1 Basic Primitives

We introduce the basic primitives of the SPiM calculus with compartments by means of a simple example. The example models some of the interactions that can take place between cells of the human immune system, and represents a tiny fraction of the processes taking place during an immune response.

Dendritic cells are part of the *adaptive immune system*, one of the main defence mechanisms of the human body against viral infections. Dendritic cells act as sentinels, patrolling areas of the body that can be exposed to infection, such as the lungs and the tissue underneath the skin or surrounding the intestines. If a dendritic cell becomes infected by a virus such as measles, the virus hijacks some of the internal machinery of the cell to produce more copies of itself, consuming

some of the resources of the cell in the process. When infected with a virus, a dendritic cell can detect the infection by means of internal receptors, which cause the cell to become activated. Once activated, the dendritic call can travel to a nearby *lymph node*, where it presents fragments of some of the proteins being made by the virus. The fragments are presented by means of *MHC class I molecules*, which capture the fragments from inside the cell and present them at the cell surface. The MHC class I molecules provide an external signal that the cell is infected, by presenting specific parts of the viral proteins as evidence of the infection. Once inside a lymph node, the dendritic cell with MHC class I molecules on its surface will encounter a large number of naive *CD8 T cells*, some of which will recognise the viral protein fragments being presented and become activated as a result. Only a very small proportion of the total population of CD8 T cells will be able to recognise a particular protein fragment from a particular virus. This is because the T cell repertoire needs to be broad enough to recognise potentially billions of viral proteins. This broad repertoire is achieved by a process of *genetic recombination*, involving a random mix-and-match of different genetic sequences during T cell development. Thus, each T cell develops receptors that can only recognise specific protein fragments. When the receptors of a given CD8 T cell recognise a particular protein fragment being presented on the surface of an infected dendritic cell, the T cell can become activated, provided it also receives the necessary co-stimulatory signals. The activated CD8 T cell then proliferates to make more copies of itself, creating an army of such cells to effectively handle the viral infection. The activated T cells leave the lymph node, travelling through the blood stream until they reach the site of infection. During their journey, they are guided by specific *chemokine* signalling molecules, while specific *addressin* and *selectin* molecules allow them to identify and enter the site of infection. On arrival, the activated CD8 T cells are able to use their receptors to target the infected cells. This is because the infected cells are also presenting protein fragments from the virus with which they are infected, by means of MHC class I molecules. The T cell army can recognise the specific protein fragments being presented by the infected cells, and can target those cells for destruction in order to help contain the viral infection.

Here we have described only a tiny fraction of the events taking place during a CD8 T cell response. More details are available in standard text books such as [34,20], some of which are searchable online[3]. A recent review of the presentation of protein fragments by MHC class I molecules can also by found in [38].

Fig. 32 presents a SPiM model of some of the basic processes that can take place during an immune response to a viral infection. The graphical representation at the top is equivalent to the textual representation at the bottom. The example illustrates the main primitives of the SPiM calculus with compartments. We model the site of infection as a particular compartment, which represents a region of tissue cells belonging to an infected organ (such as the liver or a lung). The compartment is drawn as a box around a collection of processes, all of which are executing in parallel inside the compartment. We also model a cell

[3] http://www.ncbi.nlm.nih.gov/

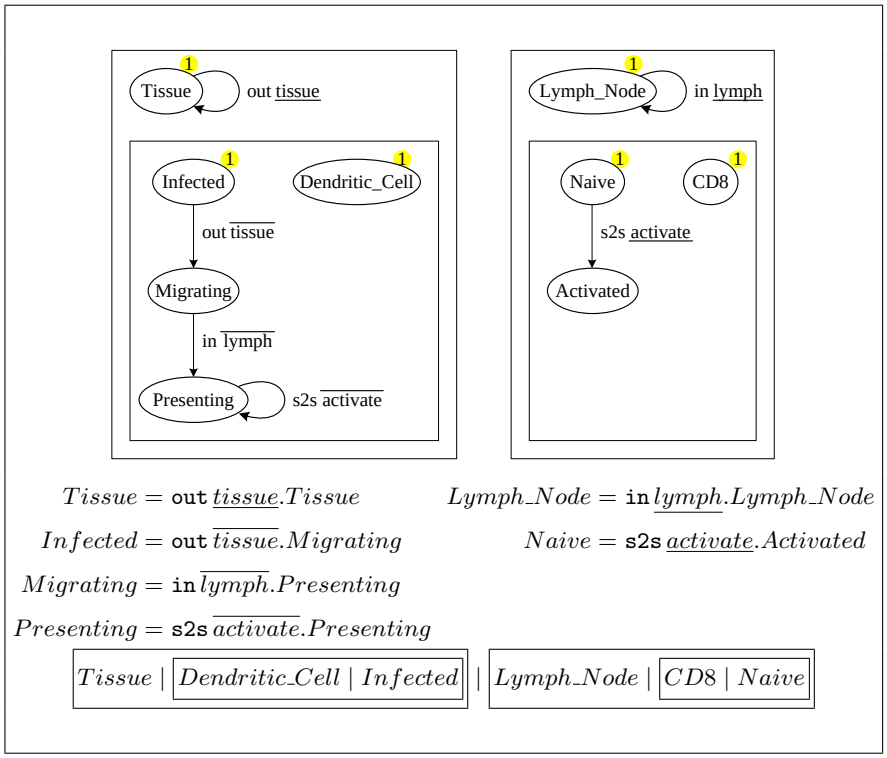


Fig. 32. A SPiM model of some of the basic processes that can take place during an immune response to a viral infection.

as a compartment, with its own internal processes. In this case, the compartment is a means of separating the computation taking place inside the cell boundary from the computation taking place outside the cell. When a dendritic cell becomes infected, it can become activated and migrate from the site of infection towards a *secondary lymphoid organ*, such as a nearby lymph node. In the SPiM calculus, the movement is represented by the $\texttt{out}\,\overline{tissue}$ action. The `out` keyword denotes the direction of movement, while the send on $\overline{tissue}$ denotes the channel over which the movement takes place. In parallel, the tissue contains a persistent $\texttt{out}\,\underline{tissue}$ action, which allows cells to leave the tissue. Thus, the send $\texttt{out}\,\overline{tissue}$ is executed inside the compartment that is leaving, while the receive $\texttt{out}\,\underline{tissue}$ is executed inside the compartment that is allowing the leave to take place. The sender and receiver can then interact on channel $tissue$, after which the sender compartment moves outside the receiver compartment. Note that the entire compartment moves together with all of its contents after the interaction takes place. The dendritic cell is then in a $Migrating$ state and can enter a lymph node by the $\texttt{in}\,\overline{lymph}$ action. The `in` keyword denotes the direction of movement, while the send on $\overline{lymph}$ denotes the channel over which the movement

takes place. In parallel, the lymph node contains a persistent `in` $\underline{lymph}$ action, which allows cells to enter the lymph node. The send `in` $\overline{lymph}$ is executed by the compartment that is entering, while the receive `in` $\underline{lymph}$ is executed by the compartment that is allowing the enter to take place. Once the dendritic cell has entered the lymph node, it can present the MHC class I molecules on its surface to nearby T cells. This is represented by a persistent send on `s2s` $\overline{activate}$. The `s2s` keyword stands for *sibling to sibling*, and denotes a communication from one sibling compartment to another.

In general, the lymph nodes are meeting places where dendritic cells can present protein fragments of the viruses with which they are infected, and can recruit T cells to help fight the infection. Here we represent a CD8 T cell as a compartment, containing the $CD8$ process in parallel with the $Naive$ process. The $CD8$ process can represent a range of cellular processes which are necessary for the survival and normal functioning of the CD8 T cell. In our model, the $Naive$ process represents the current state of the CD8 T cell. The cell can transition from $Naive$ to $Activated$ by receiving on `s2s` $\underline{activate}$ from a nearby cell. Here we represent the activation step as a single communication from the dendritic cell to the naive CD8 T cell on the *activate* channel. In reality, of course, the mechanism is much more complicated. In particular, the activation will only be successfully if the MHC class I molecules and their contents on the surface of the dendritic cell match the receptors on the surface of the T cell. We can represent this match in an abstract way, by requiring the sender and receiver cells to communicate on the same *activate* channel. Once the T cell becomes activated, it can then leave the lymph node and circulate in the blood until it reaches the infected tissue, though these later events are not represented in Fig. 32.

Fig. 33 illustrates the sequence of events that take place when executing the model of Fig. 32. For each frame in the figure, the graphical representation at the top is equivalent to the textual representation at the bottom. The number of copies of a given process is indicated next to the corresponding node in the figure. By default we assume that there is a single copy of each compartment. The example illustrates the `in` and `out` movement primitives, which allow compartments to move in and out of each other, respectively. When a compartment moves it travels with its entire contents, and when it reaches its new location it can interact with additional compartments that may not have been present in its previous location. For example, in the case of the dendritic cell moving inside a lymph node, on arrival it can interact with Naive CD8 T cells that were not accessible in its previous location. The interaction between two compartments is achieved using the `s2s` communication primitive, which allows two sibling compartments to communicate with each other. There are also primitives for allowing communication from a parent to a child (`p2c`) and from a child to its parent (`c2p`), respectively, together with a `merge` primitive for merging two compartments. These primitives, together with the full definition of the SPiM calculus with compartments, are given in Appendix 7.5.

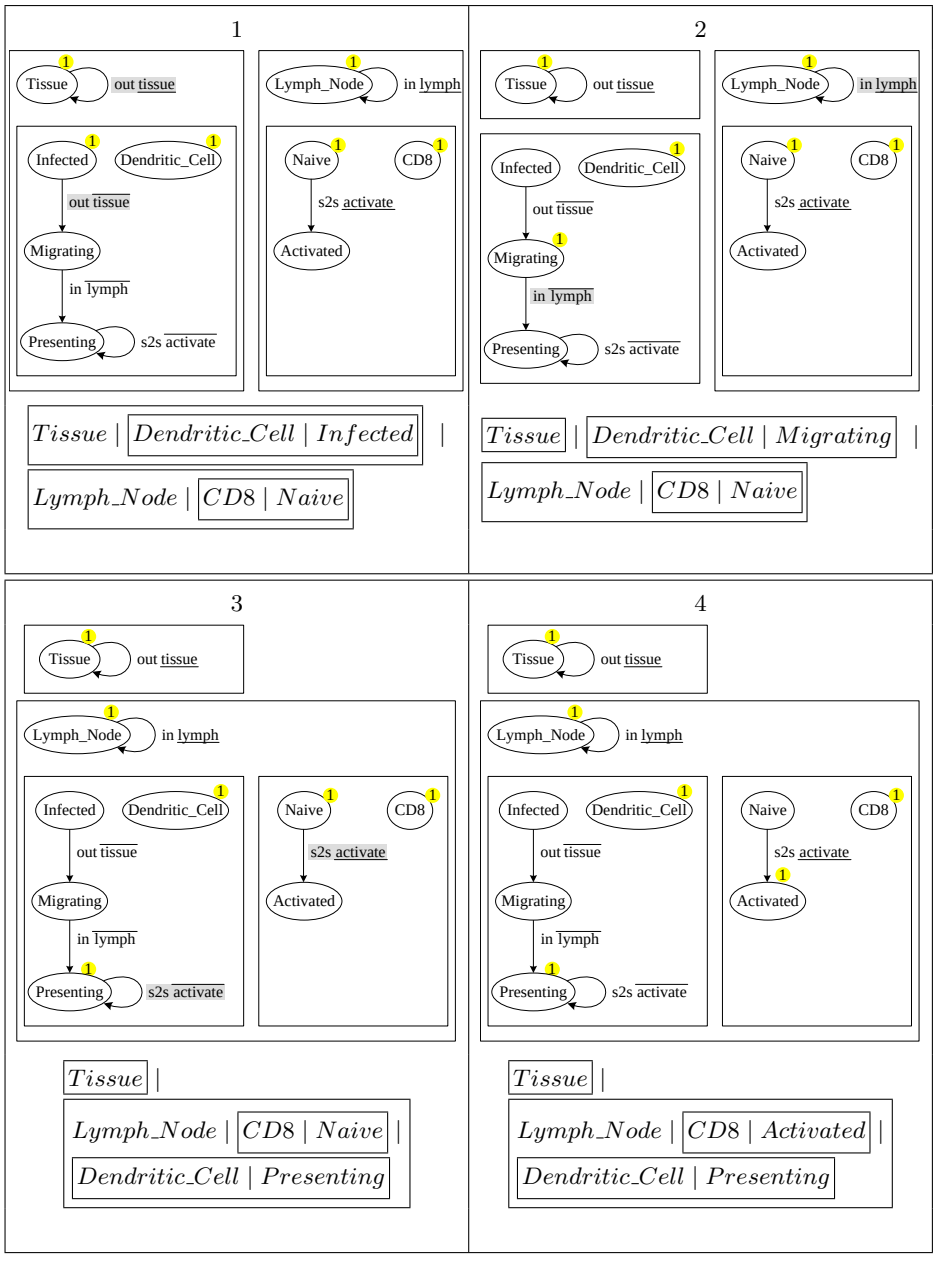


Fig. 33. The sequence of events that take place when executing the model of Fig. 32. (1) The infected dendritic cell can leave the tissue by sending on $\texttt{out}\,\overline{tissue}$. (2) The migrating dendritic cell has left the tissue, and can now enter the lymph node by sending on $\texttt{in}\,\overline{lymph}$. (3) The presenting dendritic cell can activate a nearby Naive CD8 T cell by sending on $\texttt{s2s}\,\overline{activate}$. (4) The CD8 T cell is now activated.

5.2 Simple Model of Viral Infection and T Cell Response

Fig. 34 extends the simple model of Fig. 32 to include activation and proliferation of T cells. The model of a dendritic cell is extended to include an initial *Clean* process, which can receive on `s2s` $\underline{infect}$ from a nearby cell and become $Infected$. We also allow the *Presenting* process to *die*, which represents the fact that the dendritic cell will not continue presenting indefinitely. We start with a nominal population of 10 dendritic cells, where the initial population is indicated next to the compartment boundary. We provide a basic model for the other cells in the tissue, which start out in an uninfected state and can receive on `s2s` $\underline{infected}$, becoming infected as a result. Once infected, the cell can then infect other neighbouring cells by repeatedly sending on `s2s` $\overline{infect}$. In this way, a virus inside a single cell can infect a potentially infinite number of neighbouring cells. The infected cell can also be killed by receiving on `s2s` $\underline{kill}$. We start with a nominal population of 1000 uninfected cells, together with a single infected cell. We also extend the model of the tissue by allowing multiple cells to enter or leave, represented by persistent receive actions on `in` $\underline{tissue}$ and `out` $\underline{tissue}$, respectively. Similarly, we allow multiple cells to enter or leave the lymph node, represented by persistent receive actions on `in` $\underline{lymph}$ and `out` $\underline{lymph}$, respectively. Finally, we extend the model of a CD8 T cell to represent the processes that take place after activation, namely T cell proliferation and migration to the site of infection, followed by killing of the infected cells. Once activated, the CD8 T cell can either proliferate or die, which ensures that it does not continue to proliferate indefinitely. After the $proliferate$ action is executed, a new cell is created in parallel with the *Parent* process. The creation of a new cell is represented by a triangle node, where the successor nodes to the triangle represent the contents of the newly created cell. In this example, the new cell executes the *Child* process, which sends on `out` $\overline{divide}$. This is a simple way of representing cell division, where the newly created cell leaves it parent and then executes alongside the parent. The *Parent* process allows the newly created cell to leave by receiving on `out` $\underline{divide}$. Once the child cell has left its parent, it executes the *Activated* and $CD8$ processes, representing a newly created CD8 T cell in an activated state. Meanwhile, the parent cell can leave the lymph node by sending on `out` $\overline{lymph}$ and then enter the tissue by sending on `in` $\overline{tissue}$. On arrival, it can kill neighbouring infected cells by repeatedly sending on `s2s` $\overline{kill}$, until eventually it dies. Note that the newly created T cell can perform the same actions as the activated parent. In this way, a single activated T cell can produce a large population of cells, in order to effectively fight a viral infection.

Of course, this model is still an extreme simplification of the events that actually take place during T cell activation. In particular, the activated CD8 T cell will only be able to kill infected cells that it recognises as being infected. As with T cell activation, this will depend on whether or not the T cell receptors recognise the viral protein fragments being presented on the surface of the infected cell by MHC class I molecules. We can model this complementarity between the presenter and the receiver by requiring the T cells and the infected cells to interact on the same *kill* channel. Thus, different signalling and receptor

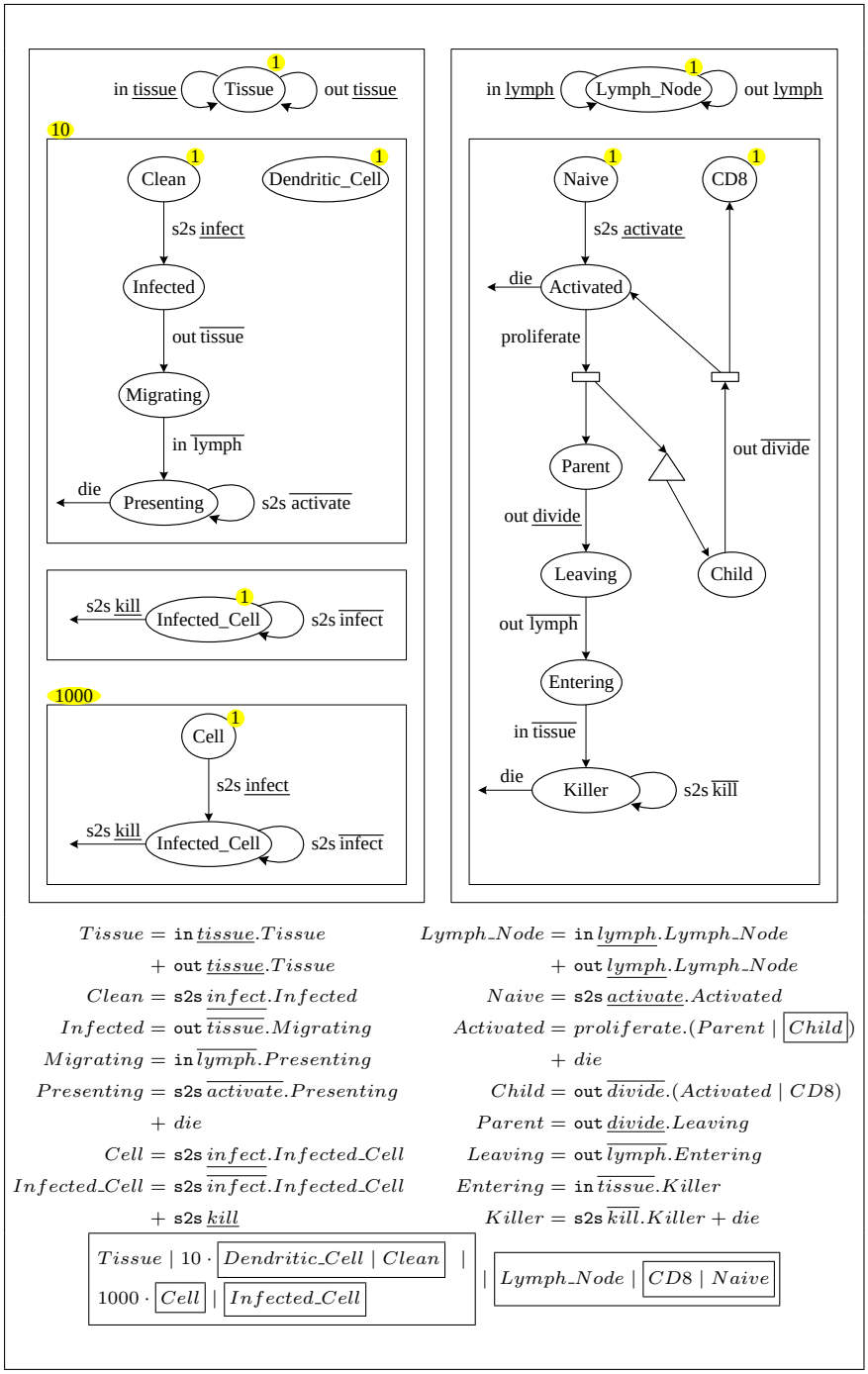


Fig. 34. A simple model of viral infection and immune response.

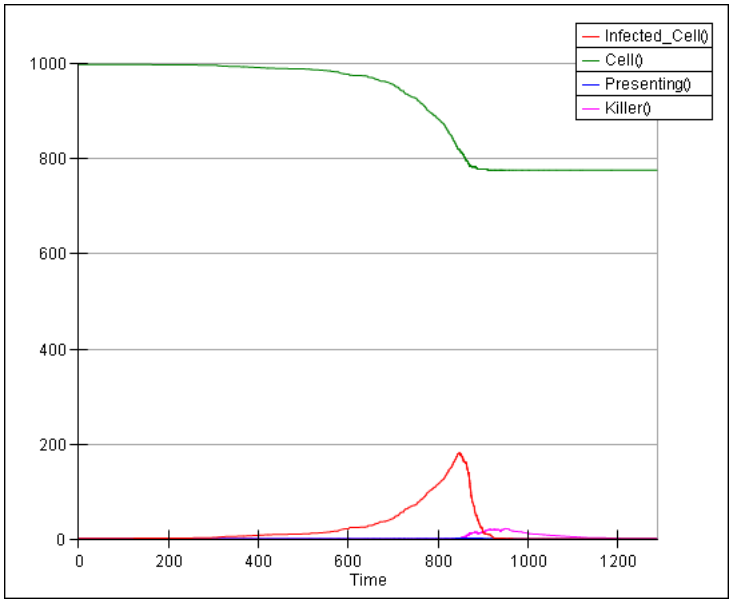


Fig. 35. Simulation results for the viral infection model of Fig. 34. The results show the total populations of uninfected and infected tissue cells, together with the populations of presenting dendritic cells and killer T cells over time. We assume that $proliferate = 0.5$, $divide = 0.01$, $activate = 0.2$, $killer = 0.005$, $infect = 0.00001$ and the remaining rates are equal to 1. We start with a single infected tissue cell among a population of 1000 uninfected cells and 10 dendritic cells. When a dendritic cell becomes infected, it moves to a lymph node where it activates a corresponding CD8 T cell. The T cell proliferates and the newly produced T cells move to the site of infection, where they kill the infected cells to contain the spread of the virus. After the infection is cleared, the T cells eventually die off.

molecules can be represented by sending and receiving on different channels. In this simplified model we have also represented proliferation by only allowing a given cell to divide once before leaving the lymph node, when in reality the same parent cell can divide multiple times. More generally, the lymph node also has a much more complicated structure, and the behaviours of the individual cells will probably require millions of lines of code in order to be accurately represented. Nevertheless, we can continually refine our models to include additional detail as needed, and we can use our highly simplified models to study the dynamic interactions between cells at an abstract level.

Fig. 35 shows a simulation of the model of Fig. 34. We start with 1000 tissue cells and a single infected tissue cell. In addition, 10 dendritic cells are assumed to be patrolling this particular area of tissue. When one of the dendritic cells becomes infected, it moves to a nearby lymph node, where it chances upon a Naive CD8 T cell that recognises the viral protein fragments being presented

on its surface and becomes activated as a result. The CD8 T cell proliferates and moves to the site of infection. Meanwhile, the infection is spreading between neighbouring cells. We can see this by the decrease in population of uninfected cells, shown in green, and the increase in population of infected cells, shown in red. As the killer T cells arrive, they destroy the infected cells as a means of preventing the spread of the virus. We observe a sudden drop in the population of infected cells, which coincides with a rise in the population of killer T cells. Once the infection has been cleared, there are no more infected cells to stimulate an immune response, and the killer T cells die off. We can try different parameter values for the simulation to observe under what conditions the infection is contained and how many of the cells become infected. We can also observe the trade-offs between rapid proliferation with a large population of killer T cells, and slower proliferation with less T cells. In reality, the immune system is often trying to strike a balance between responding too strongly and using unnecessary resources, or not responding strongly enough and being unable to clear the infection. Thus, the T cells need to proliferate at just the right rate depending on the level of presentation by the dendritic cells, and to quickly die off once the infection has been contained.

The are a number of simple ways in which the model can be extended. For instance, rather than having a fixed population of tissue cells we can include a steady-state population, where cells are continually dying and proliferating. We can also include many other types of cells such as helper T cells, which send activation signals to regulate the overall immune response. Furthermore, when a dendritic cell becomes infected by a virus such as measles, the dendritic cells can also cause the virus to spread to other cells in the lymph node, leading to a sophisticated battle between immune system and virus.

6 Conclusion

In this chapter we have presented a visual process calculus for designing and simulating computer models of biological systems. We have started with simple models of interacting genes and proteins, followed by more complex models of gene networks. We have also presented simple models of developmental processes by constructing a generic model of a particular cell type, and then instantiating this model to represent a number of identical cells in different locations with different environmental signals. Finally, we have used the calculus to model cells that can move, proliferate and interact with each other in different physical and logical locations. The calculus we have presented builds on previous work of [29,28,32,31] among many others, but there is still scope for many extensions, for example to handle spatiotemporal dynamics or more realistic biological interactions. One example is the use of primitives for biological membranes, as described in [4], which more accurately represent membrane formation and interaction. More generally, there are a wide variety of process calculi being developed for biology, such as [27,10,8,7,23] to name but a few. In future, these and other process calculi could help to form the theoretical foundations for novel program-

ming languages for biology. Such languages will enable us to take modelling, simulation and analysis of biological systems to a stage where we can handle models consisting of millions of lines of code in a scalable, modular fashion, just as we have achieved for more traditional software systems.

Acknowledgements Thanks to Hillel Kugler for discussions and helpful comments on *C. elegans* modelling and to Rosie Bloxsom for contributing a stochastic VPC model based on [37]. Thanks also to Alistair Bailey, Tim Elliott and Sandra Phillips for helpful feedback.

$\pi ::=$	r	Delay at rate r
$\mid$	$\underline{x}(\tilde{m})^r$	Receive names $\tilde{m}$ on channel x at rate r
$\mid$	$\overline{x}\langle\tilde{n}\rangle^r$	Send names $\tilde{n}$ on channel x at rate r
$\mid$	$\overline{x}(\tilde{m})^r$	Send restricted names $\tilde{m}$ on channel x at rate r
$M ::=$	$\pi_1.P_1 + \ldots + \pi_N.P_N$	Choice between actions, $N \geq 0$
$P ::=$	$\mathbf{0}$	Null
$\mid$	$X(\tilde{n})$	Species X with parameters $\tilde{n}$
$\mid$	$P_1 \mid \ldots \mid P_N$	Parallel composition of processes, $N \geq 2$
$\mid$	$(x_1) \ldots (x_N)\, P$	Restricted channels $x_1, \ldots, x_N$ in P
$D ::=$	P	Definition of a process
$\mid$	M	Definition of a choice
$E ::=$	$X_1(\tilde{m}_1) = D_1, \ldots, X_N(\tilde{m}_N) = D_N$	Definition of species X_i with parameters $\tilde{m}_i$
$S ::=$	$E \vdash P$	System with environment E and process P

Definition 1. *Syntax of* SPiM.

7 The SPiM calculus

This appendix presents the technical definition of the SPiM calculus, a variant of stochastic pi-calculus for biological modelling.

7.1 Syntax

The syntax of the calculus is presented in Definition 1, where r denotes a *rate* belonging to the set of real numbers, x denotes a *channel,* X denotes a *species,* $\tilde{n}$ denotes a *list* of names $n_1, \ldots, n_N$, and $\tilde{m}$ denotes a list of names that are pairwise distinct. Each channel x is also associated with a corresponding rate given by $\rho(x)$. Rates are used to characterise the speed of a reaction according to an exponential distribution, such that the probability of a reaction with rate r occurring within time t is given by $F(t) = 1 - e^{-r \cdot t}$. The average duration of the reaction is given by the mean $1/r$ of this distribution.

An *action* π can be a delay at rate r, a receive $\underline{x}(\tilde{m})^r$ of names $\tilde{m}$ on channel x at rate r, a send $\overline{x}\langle\tilde{n}\rangle^r$ of names $\tilde{n}$ on channel x at rate r, or a send $\overline{x}(\tilde{m})^r$ of restricted names $\tilde{m}$ on channel x at rate r. A choice M denotes a competition between zero or more actions of the form $\pi.P$, where process P is executed after performing action π. A process P can be a empty $\mathbf{0}$, a species $X(\tilde{n})$ with name X and parameters $\tilde{n}$, a parallel composition of two or more processes $P_1 \mid \ldots \mid P_N$, or a process $(x_1) \ldots (x_N)\, P$ with restricted channels $x_1, \ldots, x_N$. The restricted

channels are used to represent the formation of a complex. For example, the process $(x)\,(X_1(x) \mid X_2(x))$ represents a complex of two species, X_1 and X_2, bound to each other on channel x. The restriction denotes a local channel x on which the two species can interact in order to split the complex.

An environment E consists of a set of species definitions of the form $X(\tilde{m}) = D$, where X is the name of the species, $\tilde{m}$ are its parameters and D is its corresponding definition, which can be a process P or a choice M. It is assumed that $\mathrm{fn}(D) \subseteq \tilde{m}$, where $\mathrm{fn}(D)$ denotes the set of free names of D, given that $(x)\,P$ binds the name x in P, and $\underline{x}(\tilde{m})^r.P$ and $\overline{x}(\tilde{m})^r.P$ bind the set of names $\tilde{m}$ in P. Definitions D are also assumed to be equal up to renaming of bound names. In addition, for each definition $X(\tilde{m}) = D$, any calls to X inside D can only occur after an action π. This prevents potentially infinite processes such as $X() = (X() \mid X())$. Finally, a system S consists of a global constant environment E, together with a process P.

We define a number of syntactic conventions: we assume that the parallel composition operator $(|)$ has the lowest precedence among the operators of the calculus; we abbreviate an action $\pi.\mathbf{0}$ followed by the empty process to π, and a sequence of restricted channels $(x_1) \ldots (x_N)\,P$ to $(x_1, \ldots, x_N)\,P$; we abbreviate a parallel composition of N identical processes $P \mid \ldots \mid P$ to $N \cdot P$. We also allow some or all of the parameters of a species to be hidden in cases where these parameters are unchanged throughout the system. We can recover the parameters by observing the free names of the species definition. In order to model the binding and unbinding of species we assume that each channel x is associated with a unique *unbinding channel* ${}^{-}x$ and vice-versa, where x and ${}^{-}x$ represent two distinct names. We write ${}^{+}\overline{x}$ as an abbreviation for $\overline{x}({}^{-}x)$ and ${}^{+}\underline{x}$ as an abbreviation for $\underline{x}({}^{-}x)$. We illustrate these abbreviations with the following three identical systems, which model the binding and unbinding of proteins X and Y, as described in Fig. 9:

$$\begin{array}{l} X = {}^{+}\overline{b}.X', \;\; Y = {}^{+}\underline{b}.Y', \\ X' = {}^{-}\overline{b}.X, \;\; Y' = {}^{-}\underline{b}.Y \end{array} \vdash X \mid Y \tag{1}$$

$$\begin{array}{l} X = \overline{b}({}^{-}b).X', \;\; Y = \underline{b}({}^{-}b).Y', \\ X' = {}^{-}\overline{b}.X, \qquad Y' = {}^{-}\underline{b}.Y \end{array} \vdash X \mid Y \tag{2}$$

$$\begin{array}{l} X(b) = \overline{b}({}^{-}b).X'(b, {}^{-}b), \;\; Y(b) = \underline{b}({}^{-}b).Y'(b, {}^{-}b), \\ X'(b, {}^{-}b) = {}^{-}\overline{b}.X(b), \qquad Y'(b, {}^{-}b) = {}^{-}\underline{b}.Y(b) \end{array} \vdash X(b) \mid Y(b) \tag{3}$$

The first system models the binding and unbinding of proteins X and Y on channel b, using the syntactic abbreviations for binding. The second system expands the binding abbreviations, while the third system explicitly shows all of the parameters of the different proteins. The abbreviations allow a more compact representation of systems, without loss of information. Note that here we

$$\frac{\begin{array}{c} X_1(\tilde{n}_1) = \overline{x}\langle\tilde{n}\rangle^{r_1}.P_1 + M_1 \\ X_2(\tilde{n}_2) = \underline{x}(\tilde{m})^{r_2}.P_2 + M_2 \end{array}}{X_1(\tilde{n}_1) \,|\, X_2(\tilde{n}_2) \overset{\rho(x)\cdot r_1 \cdot r_2}{\longrightarrow} P_1 \,|\, P_{2\{\tilde{n}/\tilde{m}\}}} \quad (4) \qquad \frac{X(\tilde{n}) = r.P + M}{X(\tilde{n}) \xrightarrow{r} P} \quad (6)$$

$$\frac{P \xrightarrow{r} P'}{(x)\,P \xrightarrow{r} (x)\,P'} \quad (7)$$

$$\frac{P \xrightarrow{r} P'}{P \mid Q \xrightarrow{r} P' \mid Q} \quad (8)$$

$$\frac{\begin{array}{c} X_1(\tilde{n}_1) = \overline{x}(\tilde{m})^{r_1}.P_1 + M_1 \\ X_2(\tilde{n}_2) = \underline{x}(\tilde{m})^{r_2}.P_2 + M_2 \end{array}}{X_1(\tilde{n}_1) \,|\, X_2(\tilde{n}_2) \overset{\rho(x)\cdot r_1 \cdot r_2}{\longrightarrow} (\tilde{m})(P_1 \,|\, P_2)} \quad (5) \qquad \frac{Q \equiv P \xrightarrow{r} P' \equiv Q'}{Q \xrightarrow{r} Q'} \quad (9)$$

Definition 2. *Reaction rules in* SPiM. *The condition $X(\tilde{n}) = D'$ is true if there is a definition in the system environment such that $X(\tilde{m}) = D$ and $D' = D_{\{\tilde{n}/\tilde{m}\}}$.*

$$P \mid \mathbf{0} \equiv P \quad (10) \qquad (x)\,\mathbf{0} \equiv \mathbf{0} \quad (14)$$

$$P_1 \mid P_2 \equiv P_2 \mid P_1 \quad (11) \qquad (x)\,(y)\,P \equiv (y)\,(x)\,P \quad (15)$$

$$P_1 \,|\, (P_2 \mid P_3) \equiv (P_1 \mid P_2) \,|\, P_3 \quad (12) \qquad (x)\,(P_1|P_2) \equiv P_1|\,(x)P_2 \textbf{ if } x \notin \mathrm{fn}(P_1) \quad (16)$$

$$\pi_1.P_1 + \pi_2.P_2 \equiv \pi_2.P_2 + \pi_1.P_1 \quad (13) \qquad X(\tilde{n}) \equiv P_{\{\tilde{n}/\tilde{m}\}} \textbf{ if } X(\tilde{m}) = P \quad (17)$$

Definition 3. *Structural congruence axioms in* SPiM. Structural congruence is reflexive, symmetric and transitive, and holds in any context inside a process or a choice.

have defined binding and unbinding primitives purely by means of syntactic abbreviations. A more detailed treatment of such primitives is given in [6], which defines a novel calculus for binding and unbinding.

7.2 Reaction

The reaction rules of the calculus are presented in Definition 2. The rules are of the form $\frac{condition}{reaction}$, where the given *condition* must be satisfied in order for the given *reaction* to be possible. The notation $P \xrightarrow{r} P'$ states that the process P can evolve to P' by performing a reaction at rate r. If a species $X(\tilde{n})$ can do a delay $r.P$ in competition with actions M, then the species can evolve to a process P at rate r (6). If a species $X_1(\tilde{n}_1)$ can do a send $\overline{x}\langle\tilde{n}\rangle^{r_1}.P_1$ in competition with actions M_1, and in parallel a species $X_2(\tilde{n}_2)$ can do a receive $\underline{x}(\tilde{m})^{r_2}.P_2$ in competition with actions M_2, then the two species can interact at the rate of the channel x times the rates r_1 and r_2 of the actions. After the interaction takes place, the processes P_1 and P_2 are executed in parallel, where the parameters $\tilde{m}$ are replaced with $\tilde{n}$ in process P_2, written $P_{2\,\{\tilde{n}/\tilde{m}\}}$ (4). If the species $X_1(\tilde{n}_1)$ can do a restricted send $\overline{x}(\tilde{m})^{r_1}.P_1$ in competition with actions M_1 then the two species become bound on channels $\tilde{m}$ after the interaction takes place, and the

result of the interaction is the complex $(\tilde{m})\,(P_1 \mid P_2)$ (5). All of these reactions can also take place inside a restriction (7), inside a parallel composition (8) and up to re-ordering of processes (9). The re-ordering is defined by structural rules, according to Definition 3. Essentially, the rules allow the re-ordering of parallel processes (11), actions (13) and restricted channels (15), and the removal of empty parallel processes (10) and unused restricted channels (14). Parallel composition is associative (12), and the scope of a restricted channel can be extended over parallel processes that do not use the channel (16). Finally, a species $X(\tilde{n})$ defined in the environment as $X(\tilde{m}) = P$ can be replaced with its process definition, where the parameters $\tilde{m}$ are replaced with $\tilde{n}$ in process P (17).

7.3 Reaction Probability

According to standard principles of chemical kinetics [13], the probability of a reaction should be proportional to its rate. In order to compute this probability, we therefore need a way of computing the rates of each of the individual reactions in the system. We do this using a notion of standard form, presented in Definition 4, together with a set of indexed reaction rules, presented in Definition 5. The approach is inspired by previous work presented in [16]. A given process can be converted to standard form by application of the structural rules in Definition 3. In particular, all of the restricted channels can be moved to the top-level by application of rule (16), and each of the species defined as a process can be expanded by application of rule (17), until only species defined as a choice of actions remain. Once a process is in standard form, we can identify each species $X_i(n_i)$ by its position i in the parallel composition $X_1(n_1) \mid \ldots \mid X_N(n_N)$, and we can identify each action $\pi_j.P_j$ of a given species by its position j inside the choice $\pi_1.P_1 + \ldots + \pi_M.P_M$. Thus, each action can be identified by a pair (i, j). These action identifiers can then be used to identify all of the reactions in a system. A reaction identifier w can be a pair of indices (i, j) denoting a delay action j inside choice i, or a tuple of four indices (i_1, j_1, i_2, j_2) denoting a send action j_1 inside choice i_1 interacting with a receive action j_2 inside choice i_2. The indexed reaction rules of Definition 5 are analogous to those of Definition 2, except that they assign an identifier w to each reaction, based on the position of each action in the system. The notation $P \xrightarrow{r,w} P'$ states that the process P can evolve to P' by performing a reaction with identifier w at rate r.

We can use the indexed reaction rules to compute the probability of individual reactions, as shown in Definition 6. The propensity $\rho(P)$ of a process P is defined as the sum of the rates of all the reactions of the process (24). The probability $\Pr(P \xrightarrow{r,w} P')$ that the process P can perform a particular reaction w with rate r and evolve to P' is given by the rate of the reaction divided by the propensity of the process (25). Similarly, the probability $\Pr(P \longrightarrow P')$ that the process P can reduce to P' is given by the sum of the probabilities of all the reactions for which P can reduce to P' (26). These reaction probabilities form the basis of the SPiM stochastic simulation algorithm. More details about the algorithm and its corresponding implementation are presented in [25].

$$(x_1)\ \ldots\ (x_M)\ (X_1(\tilde{n}_1) \mid \ldots \mid X_N(\tilde{n}_N)) \tag{18}$$

Definition 4. *A process P is in* standard form *if it is in the form given by (18), such that all restricted channels are at the top-level, and each species $X_i(n_i)$ is defined as a choice of actions M_i in the environment.*

$$\frac{\begin{array}{lll} X_{i_1}(\tilde{n}_{i_1}) = \sum_{j\in J_1} \pi_j.P_j & \pi_{j_1} = \overline{x}\langle\tilde{n}\rangle^{r_1} & j_1 \in J_1 \\ X_{i_2}(\tilde{n}_{i_2}) = \sum_{j\in J_2} \pi_j.P_j & \pi_{j_2} = \underline{x}(\tilde{m})^{r_2} & j_2 \in J_2 \end{array} \quad i_1 \neq i_2 \quad i_1, i_2 \in I}{\prod_{i\in I} X_i(\tilde{n}_i) \xrightarrow{(\rho(x)\cdot r_1\cdot r_2),(i_1,j_1,i_2,j_2)} P_{j_1} \mid P_{j_2\ \{\tilde{n}/\tilde{m}\}} \mid \prod_{i\in I\setminus i_1,i_2} X_i(\tilde{n}_i)} \tag{19}$$

$$\frac{\begin{array}{lll} X_{i_1}(\tilde{n}_{i_1}) = \sum_{j\in J_1} \pi_j.P_j & \pi_{j_1} = \overline{x}(\tilde{m})^{r_1} & j_1 \in J_1 \\ X_{i_2}(\tilde{n}_{i_2}) = \sum_{j\in J_2} \pi_j.P_j & \pi_{j_2} = \underline{x}(\tilde{m})^{r_2} & j_2 \in J_2 \end{array} \quad i_1 \neq i_2 \quad i_1, i_2 \in I}{\prod_{i\in I} X_i(\tilde{n}_i) \xrightarrow{(\rho(x)\cdot r_1\cdot r_2),(i_1,j_1,i_2,j_2)} (\tilde{m})\,(P_{j_1} \mid P_{j_2}) \mid \prod_{i\in I\setminus i_1,i_2} X_i(\tilde{n}_i)} \tag{20}$$

$$\frac{X_{i_1}(\tilde{n}_{i_1}) = \sum_{j\in J} \pi_j.P_j \qquad \pi_{j_1} = r \qquad i_1 \in I, j_1 \in J}{\prod_{i\in I} X_i(\tilde{n}_i) \xrightarrow{r,(i_1,j_1)} P_{j_1} \mid \prod_{i\in I\setminus i_1} X_i(\tilde{n}_i)} \tag{21}$$

$$\frac{P \xrightarrow{r,w} P'}{(x)\,P \xrightarrow{r,w} (x)\,P'} \tag{22}$$

$$\frac{P \xrightarrow{r,w} P' \equiv Q'}{P \xrightarrow{r,w} Q'} \tag{23}$$

Definition 5. *Indexed reaction rules in* SPiM. Processes are assumed to be in standard form, in accordance with Definition 4. A choice between zero or more actions $\pi_1.P_1 + \ldots + \pi_N.P_N$ is abbreviated to a sum $\sum_{i\in\{1\ldots N\}} \pi_i.P_i$ and a parallel composition of zero or more processes $P_1 \mid \ldots \mid P_N$ is abbreviated to a product $\prod_{i\in\{1\ldots N\}} P_i$.

$$\rho(P) \triangleq \sum_{P \xrightarrow{r,w}} r \tag{24}$$

$$\Pr(P \xrightarrow{r,w} P') \triangleq \text{ if } P \xrightarrow{r,w} P' \text{ then } \frac{r}{\rho(P)} \text{ else } 0 \tag{25}$$

$$\Pr(P \longrightarrow P') \triangleq \sum_{P \xrightarrow{r,w} P'} \frac{r}{\rho(P)} \tag{26}$$

Definition 6. *Reaction Probability in* SPiM. (24) defines the propensity of process P, where $P \xrightarrow{r,w}$ means that P can perform a reaction w at rate r. (25) defines the probability that process P can perform a particular reaction w at rate r and evolve to P'. (26) defines the probability that process P can evolve to P'.

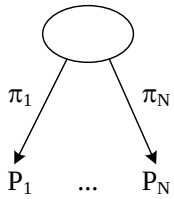
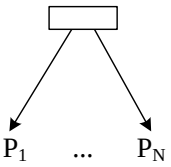
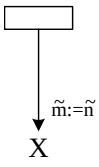
Definition	Choice	Parallel	Species	Restriction	Null
D	$\pi_1.P_1 + \ldots + \pi_N.P_N$	$P_1 \mid \ldots \mid P_N$	$X(\tilde{n}), X(\tilde{m}) = D$	$(\tilde{m})\, P$	$\mathbf{0}$
D				(m̃) P	∅

Fig. 36. Graphical representation of a static environment E in SPiM, where the graphical representation at the bottom is equivalent to the textual representation at the top.

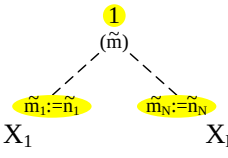
Process	Species	Restriction	Null
P	$X(\tilde{n}), X(\tilde{m}) = D$	$(\tilde{m})\,(X_1(\tilde{n}_1) \mid \ldots \mid X_N(\tilde{n}_N))$	$\mathbf{0}$
			∅

Fig. 37. Graphical representation of a dynamic process P in SPiM, where the graphical representation at the bottom is equivalent to the textual representation at the top.

7.4 Graphical Representation

The syntax of the SPiM calculus supports a concise graphical representation, originally presented in [26]. 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 reaction.

The static environment E is displayed as a *graph* of *nodes* and *edges*, as shown in Fig. 36. Each choice M or process P in the environment is displayed as a node in the graph, and each definition $X(\tilde{m}) = D$ assigns an *identifier* X and a *label* $X(\tilde{m})$ to a given node D. A choice $\pi_1.P_1 + \ldots + \pi_N.P_N$ is displayed as an elliptical node with edges to nodes $P_1, \ldots, P_N$. Each edge to a node P_i is labelled with an action π_i and denotes an alternative execution path in the system. A parallel composition $P_1 \mid \ldots \mid P_N$ is displayed as a rectangular node with edges to nodes $P_1, \ldots, P_N$. Each edge to a node P_i denotes a concurrent execution path in the system. A species $X(\tilde{n})$ with definition $X(\tilde{m}) = D$ is displayed as a

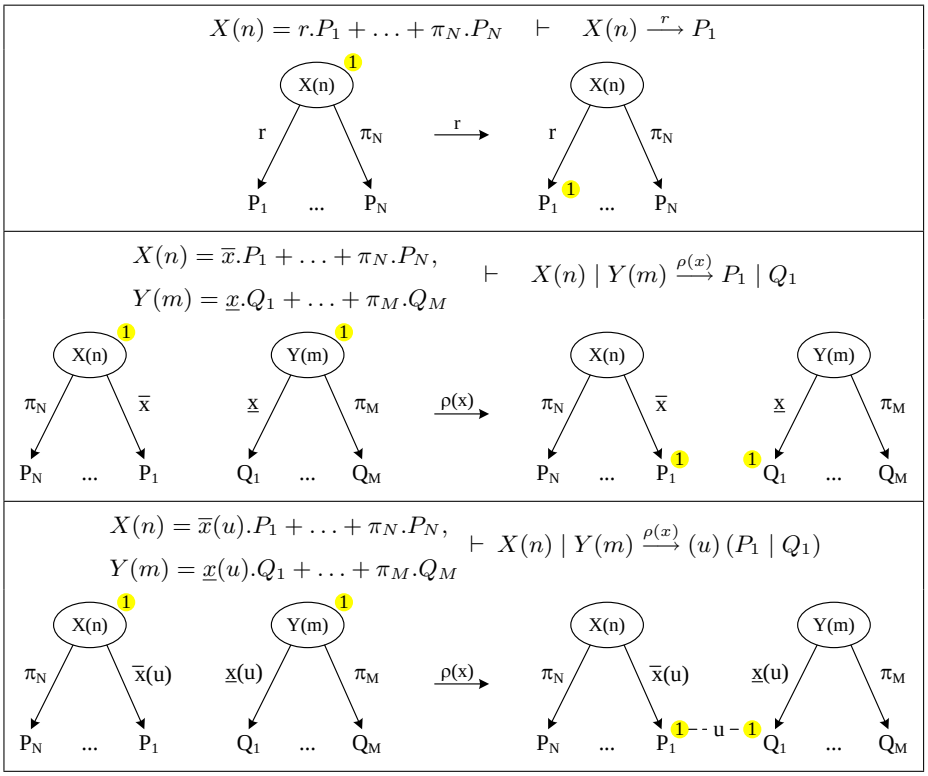


Fig. 38. Example graphical reactions in SPiM, where the graphical representation at the bottom is equivalent to the textual representation at the top. The examples illustrate the three main reaction rules (6), (4) and (5) of Definition 2, respectively. A given process is executed graphically by first applying one of the calculus reaction rules and then displaying the resulting process.

rectangular node with an edge to the node identified by X. If $\tilde{m} \neq \tilde{n}$ then the tip of the edge is labelled with the *substitution* $\tilde{m} := \tilde{n}$. This represents the passing of parameters from one node to another. Note that we only need to display the mappings in the substitution that are different from the identity, since we only need to indicate those parameters that are modified by the substitution. We also abbreviate the notation so that edges from a choice or parallel composition node to a species $X(\tilde{n})$ are connected directly to node X, without displaying the additional rectangle. For example, an action $\pi.X(\tilde{n})$ with $X(\tilde{n}) = D$ is displayed as $\bigcirc \xrightarrow{\pi} X$ instead of $\bigcirc \xrightarrow{\pi} \square \longrightarrow X$. A restriction $(\tilde{m})\,P$ is displayed by placing the restricted channels $(\tilde{m})$ next to node P, and the null process $\mathbf{0}$ is not displayed.

The dynamic process P is displayed by attaching *tokens* to nodes in the graph to represent currently executing species, and by connecting restricted names to

tokens to represent the formation of *complexes*, as shown in Fig. 37. A species $X(\tilde{n})$ with definition $X(\tilde{m}) = D$ is displayed by attaching a substitution token $\tilde{m} := \tilde{n}$ to the node identified by X. A parallel compositions of species $X_1(\tilde{n}_1) \mid \ldots \mid X_N(\tilde{n}_N)$ that share restricted names $(\tilde{m})$ is displayed by drawing a dotted edge from names $\tilde{m}$ to the tokens of each of the species. This represents the formation of a complex of species. The null process $\mathbf{0}$ is not displayed.

We introduce a number of convenient abbreviations for the graphical representation of dynamic processes. If N identical tokens are attached to the same node, they can be replaced with a single token preceded by the number N. Furthermore, if $\tilde{m} = \tilde{n}$ the token can be displayed as just the number N. If there are N copies of a restriction $(\tilde{m})\,P$ they can be displayed by placing the number N next to the restricted names $\tilde{m}$. The scope of a restricted name can be further clarified by only drawing a dotted edge to species that use this name. For example, a restriction $(x_1, x_2)\,(X_1(x_1) \mid X_2(x_1, x_2) \mid X_3(x_2))$ can be displayed as: $X_{1\,(1)} \cdots x_1 \cdots {}_{(1)} X_{2\,(1)} \cdots x_2 \cdots {}_{(1)} X_3$, assuming that the substitution tokens are all empty and displayed as (1). The graphical representation clearly shows that X_1 and X_3 do not share any restricted names, and are therefore not connected to each other directly. Finally, in some cases a graph can appear crowded when multiple tokens with different substitutions are attached to the same node. In order to improve readability in these cases, it is convenient to display a separate copy of the entire graph for each of the different tokens.

Once we have formally defined a graphical syntax in this way, we can use it to display the execution of a model. Essentially, the execution of a system can be visualised by applying the reaction rules to a given process in SPiM, and then displaying the resulting process after each reaction. Thus, we do not need to formally define the graphical reactions rules. We illustrate a number of example reactions by displaying the process before and after the reaction takes place, as shown in Fig. 38.

7.5 SPiM with Compartments

The SPiM calculus can be extended with mobile compartments, resulting in a calculus similar to the bioambient calculus originally presented in [31]. The syntax of SPiM with compartments is a direct extension of Definition 1. The syntax of processes is extended with the notion of an *ambient* $\boxed{P}$ consisting of a process P inside a compartment. The syntax of actions is extended so that an action π can also be a receive $\gamma\underline{x}(\tilde{m})$ of names $\tilde{m}$ on channel x from an ambient, a send $\gamma\overline{x}\langle\tilde{n}\rangle$ of names $\tilde{n}$ on channel x to an ambient, or a send $\gamma\overline{x}(\tilde{m})$ of restricted names $\tilde{m}$ on channel x to an ambient, where γ denotes the direction of communication. This can be from a child ambient to its parent (`c2p`), from a parent ambient to a child (`p2c`), or from one sibling ambient to another (`s2s`). In addition, an action π can be a move $\mu\overline{x}$ on channel x or an accept $\mu\underline{x}$ on channel x, where μ denotes the direction of movement. This can be an ambient entering one of its siblings (`in`), a child ambient leaving its parent (`out`) or a merge of two sibling ambients (`merge`).

$P, Q ::=$	$\ldots$		$\gamma ::=$	`s2s`	Sibling
	$\mid \boxed{P}$	Ambient containing process P		$\mid$ `c2p`	Parent
				$\mid$ `p2c`	Child
$\pi ::=$	$\ldots$				
	$\mid \gamma\underline{x}(\tilde{m})$	Ambient receive	$\mu ::=$	`in`	Enter
	$\mid \gamma\overline{x}\langle\tilde{n}\rangle$	Ambient send		$\mid$ `out`	Leave
	$\mid \gamma\overline{x}(\tilde{n})$	Ambient restricted send		$\mid$ `merge`	Merge
	$\mid \mu\overline{x}$	Ambient move			
	$\mid \mu\underline{x}$	Ambient accept			

Definition 7. *Syntax of* SPiM *with compartments, which extends the syntax of Definition 1, where γ represents a direction of communication and μ represents a direction of movement.*

The reaction rules for SPiM with compartments are presented in Definition 8, and extend the reaction rules of Definition 2. The rules (27) - (29) allow two ambients to communicate with each other. If a species $X_1(\tilde{n}_1)$ can do a send $\gamma\overline{x}\langle\tilde{n}\rangle^{r_1}.P_1$ in competition with actions M_1, and in parallel a species $X_2(\tilde{n}_2)$ can do a receive $\gamma\underline{x}(\tilde{m})^{r_2}.P_2$ in competition with actions M_2, then the two species can interact at the rate of the channel x times the rates r_1 and r_2 of the actions. After the interaction takes place, process P_1 continues executing in the sending ambient, and process P_2 continues executing in the receiving ambient, where the parameters $\tilde{m}$ are replaced with the names $\tilde{n}$ in process P_2, written $P_{2\,\{\tilde{n}/\tilde{m}\}}$. The communication can be from a child to a parent (27), from a parent to a child (28) or from one sibling to another (29).

The rules (30) - (32) allow two ambients to move in or out of each other, or to merge with each other. If a species $X_1(\tilde{n}_1)$ can do a move $\mu\overline{x}^{r_1}.P_1$ in competition with actions M_1, and in parallel a species $X_2(\tilde{n}_2)$ can do an accept $\mu\underline{x}^{r_2}.P_2$ in competition with actions M_2, then the two species can interact at the rate of the channel x times the rates r_1 and r_2 of the actions. After the interaction takes place, process P_1 continues executing in the moving ambient, and process P_2 continues executing in the accepting ambient. The move can be one sibling entering another (30), a child leaving its parent (31) or two siblings merging (32). In the case of the merge, the contents of both ambients are merged in a single ambient. In addition, all of these reactions can take place inside an ambient (33).

The structural rules for SPiM with compartments are extended with the rules in Definition 9, which allow empty ambients to be deleted (34) and the scope of restricted names to be extended outside an ambient (35). The reaction probabilities for SPiM with compartments can be derived in a similar way to Definition 5 and Definition 6, but we omit the details here. More information

$$\frac{X_1(\tilde{n}_1) = \mathtt{c2p}\,\overline{x}\langle\tilde{n}\rangle^{r_1}.P_1 + M_1 \qquad X_2(\tilde{n}_2) = \mathtt{c2p}\,\underline{x}(\tilde{m})^{r_2}.P_2 + M_2}{\boxed{Q_1 \mid X_1(\tilde{n}_1)} \mid Q_2 \mid X_2(\tilde{n}_2) \xrightarrow{\rho(x)\cdot r_1 \cdot r_2} \boxed{Q_1 \mid P_1} \mid Q_2 \mid P_{2\{\tilde{n}/\tilde{m}\}}} \tag{27}$$

$$\frac{X_1(\tilde{n}_1) = \mathtt{p2c}\,\overline{x}\langle\tilde{n}\rangle^{r_1}.P_1 + M_1 \qquad X_2(\tilde{n}_2) = \mathtt{p2c}\,\underline{x}(\tilde{m})^{r_2}.P_2 + M_2}{Q_1 \mid X_1(\tilde{n}_1) \mid \boxed{Q_2 \mid X_2(\tilde{n}_2)} \xrightarrow{\rho(x)\cdot r_1 \cdot r_2} Q_1 \mid P_1 \mid \boxed{Q_2 \mid P_{2\{\tilde{n}/\tilde{m}\}}}} \tag{28}$$

$$\frac{X_1(\tilde{n}_1) = \mathtt{s2s}\,\overline{x}\langle\tilde{n}\rangle^{r_1}.P_1 + M_1 \qquad X_2(\tilde{n}_2) = \mathtt{s2s}\,\underline{x}(\tilde{m})^{r_2}.P_2 + M_2}{\boxed{Q_1 \mid X_1(\tilde{n}_1)} \mid \boxed{Q_2 \mid X_2(\tilde{n}_2)} \xrightarrow{\rho(x)\cdot r_1 \cdot r_2} \boxed{Q_1 \mid P_1} \mid \boxed{Q_2 \mid P_{2\{\tilde{n}/\tilde{m}\}}}} \tag{29}$$

$$\frac{X_1(\tilde{n}_1) = \mathtt{in}\,\overline{x}^{r_1}.P_1 + M_1 \qquad X_2(\tilde{n}_2) = \mathtt{in}\,\underline{x}^{r_2}.P_2 + M_2}{\boxed{Q_1 \mid X_1(\tilde{n}_1)} \mid \boxed{Q_2 \mid X_2(\tilde{n}_2)} \xrightarrow{\rho(x)\cdot r_1 \cdot r_2} \boxed{\boxed{Q_1 \mid P_1} \mid Q_2 \mid P_2}} \tag{30}$$

$$\frac{X_1(\tilde{n}_1) = \mathtt{out}\,\overline{x}^{r_1}.P_1 + M_1 \qquad X_2(\tilde{n}_2) = \mathtt{out}\,\underline{x}^{r_2}.P_2 + M_2}{\boxed{\boxed{Q_1 \mid X_1(\tilde{n}_1)} \mid Q_2 \mid X_2(\tilde{n}_2)} \xrightarrow{\rho(x)\cdot r_1 \cdot r_2} \boxed{Q_1 \mid P_1} \mid \boxed{Q_2 \mid P_2}} \tag{31}$$

$$\frac{X_1(\tilde{n}_1) = \mathtt{merge}\,\overline{x}^{r_1}.P_1 + M_1 \qquad X_2(\tilde{n}_2) = \mathtt{merge}\,\underline{x}^{r_2}.P_2 + M_2}{\boxed{Q_1 \mid X_1(\tilde{n}_1)} \mid \boxed{Q_2 \mid X_2(\tilde{n}_2)} \xrightarrow{\rho(x)\cdot r_1 \cdot r_2} \boxed{Q_1 \mid P_1 \mid Q_2 \mid P_2}} \tag{32}$$

$$\frac{P \xrightarrow{r} P'}{\boxed{P} \xrightarrow{r} \boxed{P'}} \tag{33}$$

Definition 8. *Reaction rules for* SPiM *with compartments, which extend the reaction rules of Definition 2. The condition $X(\tilde{n}) = D'$ is true if there is a definition in the system environment such that $X(\tilde{m}) = D$ and $D' = D_{\{\tilde{n}/\tilde{m}\}}$. For each of the communication rules (27), (28) and (29) there is also a corresponding rule for restricted communication, along the lines of (4) (not shown).*

$$\boxed{\mathbf{0}} \equiv \mathbf{0} \tag{34}$$

$$(x)\boxed{P} \equiv \boxed{(x)\,P} \tag{35}$$

Definition 9. *Structural congruence axioms for* SPiM *with compartments, which extend the axioms of Definition 3.*

about a stochastic simulation algorithm for bioambients and its corresponding implementation are presented in [24].

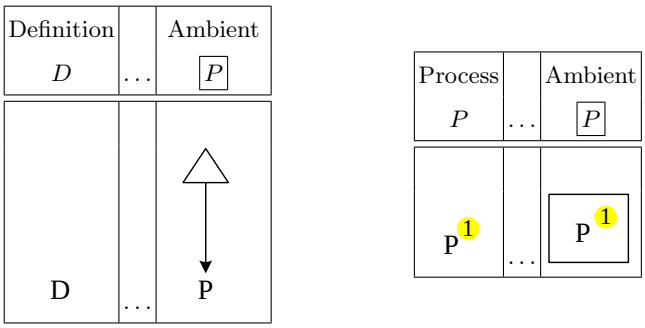


Fig. 39. Graphical representation for SPiM with compartments. The static representation of an ambient extends the display of static environments E in Fig. 36, while the dynamic representation extends the display of dynamic processes P in Fig. 37.

The SPiM calculus with compartments also supports a concise graphical representation, as shown in Fig. 39. In a static environment, an ambient $\boxed{P}$ is displayed as a triangular node with an edge to process P. In a dynamic process, the ambient is displayed as a box around the process P, where each box contains its own copy of the graph representing P. If N identical ambients are executing in parallel, only one copy of the ambient is represented, with the number N next to the surrounding box. Once again, the graphical reaction rules do not need to be defined explicitly. Instead, the execution of a system can be visualised by displaying the system before and after each reaction. We illustrate a number of example reactions, as shown in Fig. 40.

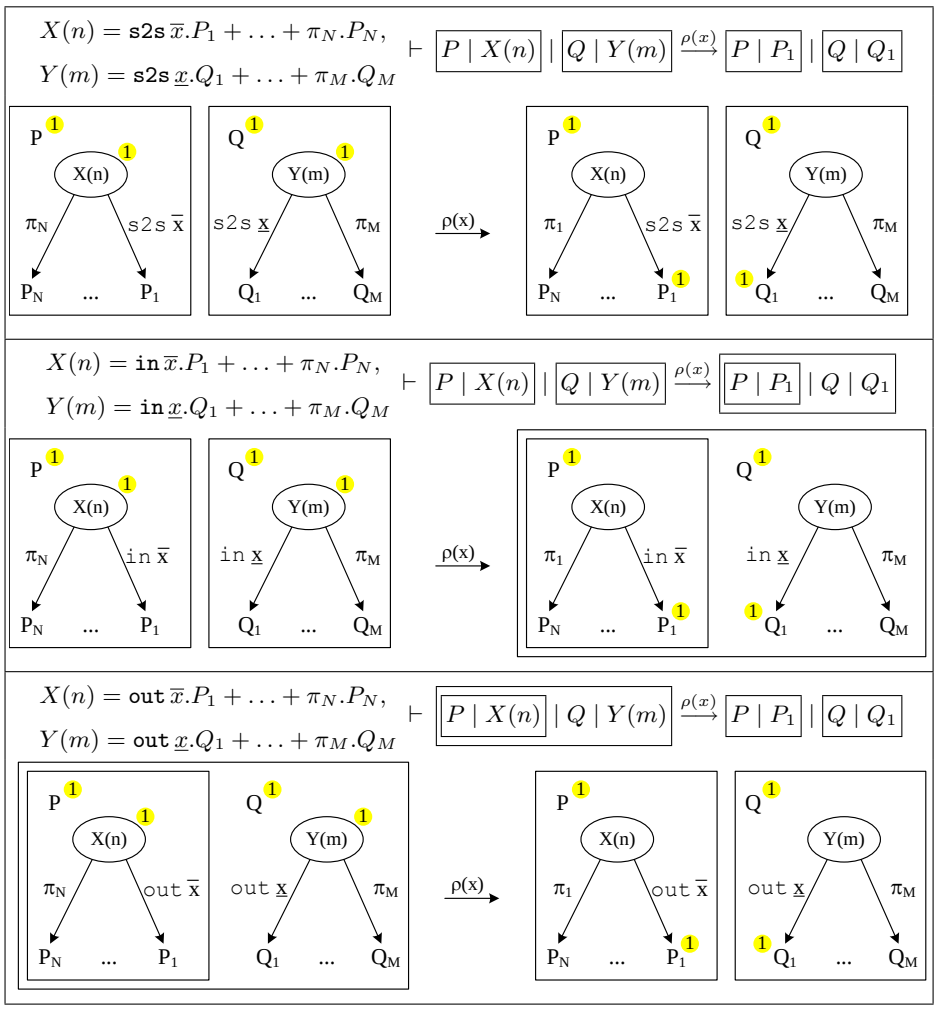


Fig. 40. Example graphical reactions in SPiM with compartments, where the graphical representation at the bottom is equivalent to the textual representation at the top. The examples illustrate the reaction rules (29), (30) and (31) of Definition 8, respectively.

References

1. 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.
2. Ralf Blossey, Luca Cardelli, and Andrew Phillips. Compositionality, stochasticity and cooperativity in dynamic models of gene regulation. *HFSP Journal*, 2(1):17–

28, February 2008.
3. Johannes Borgstroem, Andrew Gordon, and Andrew Phillips. A chart semantics for the pi-calculus. *Electronic Notes in Theoretical Computer Science*, 194(2):3–29, January 2008.
4. Luca Cardelli. Brane calculi. In *CMSB'04*, pages 257–278, 2004.
5. Luca Cardelli and Andrew D. Gordon. Mobile ambients. *Theoretical Computer Science*, 240(1):177–213, 2000. An extended abstract appeared in *FoSSaCS '98*: 140–155.
6. Luca Cardelli and Gianluigi Zavattaro. On the computational power of biochemistry. In *Algebraic Biology*, volume 5147 of *LNCS*, pages 65–80. Springer, July 2008.
7. Nathalie Chabrier-Rivier, François Fages, and Sylvain Soliman. The biochemical abstract machine BIOCHAM. In V. Danos and V. Schächter, editors, *Proc. CMSB*, volume 3082 of *LNCS*, pages 172–191. Springer, 2004.
8. Federica Ciocchetta and Jane Hillston. Bio-pepa: An extension of the process algebra pepa for biochemical networks. *Electron. Notes Theor. Comput. Sci.*, 194(3):103–117, 2008.
9. Paul W. Sternberg Claudiu A. Giurumescu and Anand R. Asthagiri. Intercellular coupling amplifies fate segregation during caenorhabditis elegans vulval development. *PNAS*, 103(5):1331–1336, January 2006.
10. Vincent Danos and Cosimo Laneve. Formal molecular biology. *Theor. Comput. Sci.*, 325(1):69–110, 2004.
11. Michael B. Elowitz and Stanislas Leibler. A synthetic oscillatory network of transcriptional regulators. *Nature*, 403(6767):335–338, January 2000.
12. Jasmin Fisher, Nir Piterman, Alex Hajnal, and Thomas A. Henzinger. Predictive modeling of signaling crosstalk during c. elegans vulval development. *PLoS Computational Biology*, 3(5):0862–0873, May 2007.
13. Daniel T. Gillespie. Exact stochastic simulation of coupled chemical reactions. *J. Phys. Chem.*, 81(25):2340–2361, 1977.
14. Călin C. Guet, Michael B. Elowitz, Weihong Wang, and Stanislas Leibler. Combinatorial synthesis of genetic networks. *Science*, 296:1466–1470, May 2002.
15. Michael Hucka et al. The systems biology markup language (sbml): a medium for representation and exchange of biochemical network models. *Bioinformatics*, 19(4):524–531, 2003.
16. Céline Kuttler, Cédric Lhoussaine, and Joachim Niehren. A stochastic pi calculus for concurrent objects. In *Algebraic Biology*, volume 4545 of *LNCS*, pages 232–246. Springer, August 2007.
17. Paola Lecca and Corrado Priami. Cell cycle control in eukaryotes: a biospi model. In *BioConcur'03*. ENTCS, 2003.
18. Robin Milner. *Communicating and Mobile Systems: the π-Calculus*. Cambridge University Press, 1999.
19. Robin Milner, Joachim Parrow, and David Walker. A calculus of mobile processes, part I/II. *Journal of Information and Computation*, 100:1–77, September 1992.
20. Kenneth M. Murphy, Paul Travers, and Mark Walport. *Janeway's Immunobiology*. Garland Science, seventh edition, 2008.
21. Rami Marelly Lara Appleby Jasmin Fisher Amir Pnueli David Harel Michael J. Stern E. Jane Albert Hubbard Na'aman Kam, Hillel Kugler. A scenario-based approach to modeling development: A prototype model of c. elegans vulval fate specification. *Developmental Biology*, Volume 323(1):1–5, November 2008.
22. Denis Noble. *The Music of Life: Biology Beyond the Genome*. Oxford University Press, 2006.

23. Michael Pedersen and Gordon Plotkin. A language for biochemical systems. In M. Heiner and A. M. Uhrmacher, editors, *Proc. CMSB*, LNCS. Springer, 2008.
24. Andrew Phillips. An abstract machine for the stochastic bioambient calculus. *Electronic Notes in Theoretical Computer Science*, 227:143–159, January 2009.
25. Andrew Phillips and Luca Cardelli. Efficient, correct simulation of biological processes in the stochastic pi-calculus. In *Computational Methods in Systems Biology*, volume 4695 of *LNCS*, pages 184–199. Springer, September 2007.
26. Andrew Phillips, Luca Cardelli, and Giuseppe Castagna. A graphical representation for biological processes in the stochastic pi-calculus. *Transactions in Computational Systems Biology*, 4230:123–152, November 2006.
27. C. Priami and P. Quaglia. Beta binders for biological interactions. In V. Danos and V. Schächter, editors, *Proc. CMSB*, volume 3082 of *LNBI*, pages 20–33. Springer, 2005.
28. 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.
29. Corrado Priami. Stochastic π-calculus. *The Computer Journal*, 38(6):578–589, 1995.
30. A. Regev, W. Silverman, and E. Shapiro. Representation and simulation of biochemical processes using the pi- calculus process algebra. In *Pacific Symposium on Biocomputing*, volume 6, pages 459–470, 2001.
31. Aviv Regev, Ekaterina M. Panina, William Silverman, Luca Cardelli, and Ehud Y. Shapiro. Bioambients: an abstraction for biological compartments. *Theor. Comput. Sci.*, 325(1):141–167, 2004.
32. Aviv Regev and Ehud Shapiro. The π-calculus as an abstraction for biomolecular systems. *Modelling in Molecular Biology*, pages 219–266, 2004.
33. Davide Sangiorgi and David Walker. *The π-calculus: a Theory of Mobile Processes*. Cambridge University Press, 2001.
34. Lauren M. Sompayrac. *How the Immune System Works*. Wiley-Blackwell, third edition, 2008.
35. Paul W. Sternberg. Vulval development. *WormBook*, June 2005.
36. Paul W. Sternberg and H. Robert Horvitz. Pattern formation during vulval development in c. elegans. *Cell*, 44:761–772, March 1986.
37. Paul W. Sternberg and H. Robert Horvitz. The combined action of two intercellular signaling pathways specifies three cell fates during vulval induction in c. elegans. *Cell*, 58:679–693, August 1989.
38. Jonathan W. Yewdell, Eric Reits, and Jacques Neefjes. Making sense of mass destruction: quantitating mhc class i antigen presentation. *Nature Reviews Immunology*, 3:952–961, 2003.