



Applications of Process Algebra in Systems Biology

Nicolas Wu^a

^a*Brasenose College, University of Oxford*

Abstract

This report introduces some of the ongoing work in the application of process algebra in systems biology. This survey reveals that although progress has been made in using process algebra to create stochastic simulations of the systems in question, no work has been done to emphasise the importance of refinement theory. A brief description of the notation and some of the semantics of Communicating Sequential Processes, a process algebra, are presented, with the aim of applying refinement in biological models. As an example, bacterial chemotaxis is modeled using the principles discussed. This results in an incremental approach to modelling, and allows the direct comparison of behaviour between systems. Finally, the weaknesses of this methodology are examined and proposals to address them are suggested.

1 Introduction

The aim of systems biology is to be able to formally and quantitatively reason about biological systems in an effort to understand them, and eventually predict behaviour. One such system of interest is the signalling pathway in cells, and in particular, chemotaxis. Many aspects of chemotaxis in bacteria have been well understood, and various models have been proposed to explain the cascade of signals from the chemoreceptors on the cell surface to the switching mechanisms of the flagellar motors.

It is becoming clear that concurrency theory and process algebras have particular relevance in systems biology and recently these have been used to describe various interactions. Originally, process algebras were developed in order to facilitate the formal reasoning of processes in computer science, allowing the specification and verification of these processes. In systems biology, we view cells analogously to complex computer systems and this abstraction allows us to apply the rich library of theory available to processes in cells.

In Section 2 of this report, the concept of process algebra will be introduced and a brief overview of the cur-

rent ongoing work in the applications of process algebra in systems biology is given. Section 3 presents a brief account of *Communicating Sequential Processes* (CSP), a process algebra which considers the refinement of one process to another. Following this, a simple description of chemotaxis in *E. coli* is presented in Section 4. This system is then modelled using CSP in Section 5, and the model is improved upon in Section 6. A discussion concerning the suitability of probabilistic CSP and of further work is found in Section 7. Finally, concluding remarks can be found in Section 8.

2 Process Algebra in Systems Biology

One of the first applications of process algebra in systems biology was by Regev et al. (2000). In their paper they investigate a ‘molecule-as-computation’ abstraction, where a domain such as a receptor molecule is a process, residues such as phosphorylation sites are channels on which communication can take place, and molecular interaction or modification is the communication of a message. By abstracting in this way, we are able to analyse the dynamics of a biological system on a molecular level by using techniques from process algebra.

This abstraction is made with the specific aim of using a

rich language of description which can be applied to reason about molecular processes. In particular, four goals are identified that an appropriate formal approach to modelling molecular processes should fulfil (Regev and Shapiro, 2002), it must be:

Relevant The formal representation must offer a unifying view of both the molecular data and dynamics of behaviour. This should be similar to the intuitive biological understanding of the underlying system.

Computable The representation must be well defined and computable. This goal is with the aim of simulating dynamics using a computer.

Understandable There should be some way of measuring the differences and similarity in terms of structure, dynamics and function between different systems.

Extensible Scalability and modularisation are essential given the sheer size of the systems which are being simulated.

They argue that contrary to methods such as kinetic models based on differential equations, petri-nets and object oriented databases, process algebra offers a unique and fully encompassing framework to describe molecular processes which satisfies all four goals. They suggest that π -calculus should be used to model biochemical networks as mobile communication systems, with the aim of integrating dynamics, molecular and biochemical detail.

In their paper and subsequent ones (Regev et al., 2001), they describe a model for the RTK-MAPK signal transduction pathway. In these early works, the models capture the functional aspects of the pathway, but every interaction has an equal chance of occurring. This is obviously too coarse an approximation of the real system where reaction rates play a crucial role in determining the resulting behaviour. This issue was later addressed by the same authors in Regev (2001) and Priami et al. (2001), in an implementation of stochastic π -calculus called BioSPI, based on the Gillespie algorithm (Gillespie, 1977).

One contribution that these models provide is the simulation of mutant behaviour (Regev et al., 2001). This is achieved by modifying parts of the code describing the system. This proved particularly useful when trying to understand a mutation which had surprising results; by ‘debug tracing’ their simulation, they were able to understand that increasing MP1, a facilitator of Mek and Erk phosphorylation, had in fact an inhibitory effect in large quantities due to unforeseen interactions with other proteins.

A criticism of the methods used here is that the model proposed simulates interactions on the molecular level and yet only a handful of individual molecules are being simulated. In reality, thousands of molecules will

be present in a cell and this fact is not taken into account. On this note, work by Calder et al. (2004) focuses on alternative approaches to constructing a representation. They use *Performance Evaluation Process Algebra* (Hillston, 1996) to model the Extracellular signal Regulated Kinase (ERK) signalling pathway, and how it is influenced by the Raf Kinase Inhibitor Protein (RKIP). PEPA is a Markovian process algebra that resembles π -calculus, but has been extended to incorporate stochastic durations and probabilistic choices.

Two different approaches to modelling the signalling pathway are proposed: a ‘reagent-centric’ view and a ‘pathway-centric view’. As the names suggest, the reagent-centric view focuses on describing the different states that the reagents themselves can undergo. This is done by describing each possible reaction in terms of its producer and consumer reagents, as well as the reaction rate. The alternative model is the pathway-centric view, which emphasises how a substrate is consumed and eventually replenished. Calder et al. (2004) go on to show us that the reagent and pathway-centric views are equivalent models since they contain exactly the same transition state space, although they do not prove this isomorphism between the views to be true in general. Being able to model the system in different ways to produce equivalent models is a common concept in process algebra, and in our discussion of CSP we will emphasise this idea. Overall, their models provide an elegant means of simulating the effects of increasing rates of certain reactions, allowing the models to make predictions of behaviour based on Continuous-Time Markov Chain models.

Further to this work, Calder et al. (2005) have shown a way of automatically generating ordinary differential equations (ODEs) from their models of signalling pathways. This is particularly useful from two perspectives. First, in terms of validation they are able to compare the ODEs produced by their models with those already accepted in the scientific community. Secondly, this allows ODEs to be automatically generated which are free from deadlock and are intuitive to understand. From these equations, numerical methods such as the Runge-Kutta algorithm are used to analyse their model further.

An alternative use of process algebra is to use it to verify that the implementation of a system satisfies certain conditions stipulated in a specification. In terms of systems biology, this will help us with the validation of our models. In order to do this, we will be using *Communicating Sequential Processes* (CSP), a language developed by Hoare (1985) which has particular emphasis on the refinement of one process from another based on their event traces and failures, to describe the systems. A brief introduction is presented in the following section, and further details can be found in Roscoe (1998).

3 Communicating Sequential Processes

3.1 Simple processes

In CSP, we think of a *process* in terms of the *events* that it can communicate. Events occur instantaneously and last an indivisibly small amount of time, so we do not worry about when in time they happen, but rather, the order in which events occur is what we are concerned with.

Each event p is taken from the set Σ of all events in a particular context. Given a process P and an event $p \in \Sigma$, $p \rightarrow P$ is the process which can offer to communicate p and then behave like P . The simplest process of all is the one which does not communicate. We call this *STOP*.

A simple process would therefore be $p \rightarrow \text{STOP}$, the process which communicates p , then nothing else. We can also define recursive processes, for example, $P = a \rightarrow b \rightarrow P$ is the process which successively performs a and then b forever.

3.2 External Choice

So far, the processes we have described can only behave in a completely linear fashion in that the systems will always communicate the same sequence of events from beginning to end. In reality, we are much more interested in processes which offer alternative behaviours. A choice of processes can be expressed by saying that

$$P \sqcap Q$$

is the process that offers the first events of P and Q , and then behaves accordingly; we call this the *external choice* operator¹. For example, given the process $p \rightarrow P \sqcap q \rightarrow Q$, if we decide to choose the event p which has been offered by the process, then the process will perform p and continue to behave like P .

3.3 Nondeterminism and Internal Choice

It may not have been obvious, but the last definition introduced some ambiguity in the behaviour of our processes. Consider the process

$$p \rightarrow P_1 \sqcap p \rightarrow P_2$$

Quite clearly, the first event which will be communicated is p , since this is common to both sides of the operator, but whether the process will behave like P_1 or P_2 is

¹ Roscoe (1998) discusses the guarded alternative “|” and presents external choice as a semantically different choice operator. For brevity, we will not discuss this here.

not known. In fact, the process is able to behave like either, but cannot behave like both (assuming of course, that $P_1 \neq P_2$). We call this process *nondeterministic*, meaning that there is uncertainty about exactly what will occur next.

It is important that we are able to reason formally about nondeterministic behaviour. To this end we define an operator to distinguish between a deterministic external choice, and a nondeterministic internal choice. The process

$$P \sqcap Q$$

behaves either like P , or like Q , but we do not know which. The difference between *external* choice $P \sqcap Q$, and *internal* choice $P \sqcap Q$ may appear subtle at first, but there is an important distinction between the two. In both cases, we know that the process will behave like one process or like the other. The difference between the two operators is that with external choice, we can decide which of the two processes it will behave like, whereas with internal choice, we do not have this choice; it genuinely is an internal event. In fact, the process P can be used anywhere the process $P \sqcap Q$ is suitable, since it is capable of behaving like either.

3.4 The Parallel and Interleave Operators

Now that we have a basic means of describing the behaviour of single processes, we can begin to talk about the interaction of multiple processes. The first operator which we will look at is the *parallel* operator. Given αP and αQ , the sets of all events that can be communicated in P and Q respectively, we write

$$P \parallel Q$$

to mean the parallel composition² of the processes P and Q . This is the process which behaves like P and Q , but ensures that both processes agree on any event $a \in \alpha P \cap \alpha Q$.

A more general version of the parallel operator is the *alphabetised parallel*. This is written

$$P \parallel_{\phi} \parallel_{\psi} Q$$

Here the processes P and Q are free to communicate any events in ϕ and ψ respectively, but must agree on events in $\phi \cap \psi$. We therefore have $P \parallel Q = P \parallel_{\alpha P} \parallel_{\alpha Q} Q$.

² This is not the standard definition of the parallel operator as described by Roscoe (1998), which is defined to agree (or synchronise) on all actions which occur in Σ . We make the alphabets implicit when applying this operator to keep from having to describe them at each point of use.

Sometimes we will not want our processes to synchronise on any events at all. In this case, we want to use *interleaving*, written $P \parallel Q$. This allows the processes P and Q to run simultaneously with no interaction at all. This implies that in an interleaving, we do not place any restriction on the order in which the events occur between the processes.

3.5 Traces and Failures

When trying to understand the behaviour of a process, it is useful to have a *trace* of the events which it communicates. Informally, this is a set of all the possible finite pathways of events which a process P could take, and is written $traces(P)$. A few examples of traces are:^{3 4}

$$\begin{aligned} traces(STOP) &= \{\langle \rangle\} \\ traces(a \rightarrow STOP) &= \{\langle \rangle, \langle a \rangle\} \\ traces(\mu P. a \rightarrow P) &= \{n : \mathbb{N} \bullet \langle a \rangle^n\} \\ traces(P \square Q) &= traces(P) \cup traces(Q) \\ traces(P \sqcap Q) &= traces(P) \cup traces(Q) \\ traces(P \parallel Q) &= \{s : seq \Sigma \mid s \upharpoonright \alpha P \in traces(P) \\ &\quad \wedge s \upharpoonright \alpha Q \in traces(Q)\} \\ trace(P \parallel\parallel Q) &= \{s : traces(P); t : traces(Q) \\ &\quad \bullet s \parallel\parallel t\} \end{aligned}$$

One interesting observation that can be made by looking at the traces of processes is that $traces(P \square Q) = traces(P \sqcap Q)$. The traces are identical since in both cases the process is able to behave like P or like Q . This demonstrates that traces alone are not enough to fully describe the behaviour of a process. In order to get a broader description of process behaviour, we need to analyse the *refusal set* of a process. For a process P , this is written $refusals(P)$. This is the set of events which a process can initially choose not to communicate, no matter how long it is offered by the environment. By comparing the refusal set with the traces of a process, we can see which events the process may perform, but is not required to perform. Assuming $\alpha P = \{p\}$ and $\alpha Q = \{q\}$, the refusals of the choice operators are:

$$\begin{aligned} refusals(p \rightarrow P \square q \rightarrow Q) &= \{\} \\ refusals(p \rightarrow P \sqcap q \rightarrow Q) &= \{p, q\} \end{aligned}$$

It is here that we see a clear difference in the way that external and internal choice behave.

Since we are often interested in the events which a process might refuse during the course of its execution, we

³ The definition of $traces(P \parallel\parallel Q)$ requires us to define an interleave operator on sequences. Informally, it is all the sequential combinations of the heads of both sequences.

⁴ Here the *restriction* operation $s \upharpoonright T$ is the sequence of elements in s which appear in T .

are interested in knowing what the refusals of a process is after a certain trace. We define this as the *failures* of a process. To describe this set, we first define the *failure* of a process s , which is a pair (s, X) such that for some $s \in traces(P)$, we have $X = refusals(P/s)$, where P/s represents the process P after the trace s . We can then write $failures(P)$ to mean the set of all of P 's failures⁵.

3.6 Refinement

We now have a vocabulary for defining a full spectrum different processes, and describing their behaviour in terms of what they can and may not do. One key concept which CSP offers when analysing processes is *refinement*. We write $P \sqsubseteq Q$, when P is refined by Q . It is defined

$$P \sqsubseteq Q \Leftrightarrow P \sqcap Q = P$$

This relation tells us that Q is more deterministic than P , and in a sense tells us that Q is better than P in that it is able to act in place of P (since $P \sqcap Q$ could act like Q in every instance), yet its behaviour is more predictable to the environment.

The idea of refinement is useful in that we can use it to verify that a *specification* of process behaviour is satisfied by the behaviour of a particular *implementation*. A specification could be a description of the properties of the trace which must be satisfied, and it turns out that this specification can be described fully by a process called the *characteristic* process. We could check that a specification described by the characteristic process $SPEC$ is satisfied by the process $IMPL$ by checking that $SPEC \sqsubseteq IMPL$. Since refinement is transitive and monotonic, we can move from one implementation to another by checking refinement holds at each step.

A slightly weaker form of refinement can be described in terms of traces. The definition of trace refinement is similar to that of full refinement

$$P \sqsubseteq_T Q \Leftrightarrow P \sqcap Q =_T P$$

which is more easily understood by

$$P \sqsubseteq_T Q \Leftrightarrow traces(P) \supseteq traces(Q)$$

Essentially this tells us that P must be able to perform any trace performed by Q .

Similarly, we can define failures refinement as

$$P \sqsubseteq_F Q \Leftrightarrow traces(P) \supseteq traces(Q) \wedge failures(P) \supseteq failures(Q)$$

⁵ For a full description of behaviour, we also need to take note of the points where a process may *diverge*, but for now traces and failures will be enough.

which indicates not only that every trace performed by Q can be performed by P , but also that any refusal after such a trace can only be performed by Q if it is also performed by P . In other words, Q is able to refuse no more than P .

4 Bacterial Chemotaxis

With this theory in mind, we now turn to understanding the system that we will model. Living organisms survive by continually responding to various signals in their surrounding environment. In bacteria, chemotaxis is the process by which external stimuli cause a cascade of signals which eventually result in a reaction. In particular, *Escherichia coli* respond by moving towards attractants such as aspartate and away from repellents such as Ni^{2+} . This signalling pathway is one of the most well-understood of all sensory pathways (Wadhams and Armitage, 2004), and many models have been proposed to explain its behaviour (Kollmann et al., 2005; Bray et al., 1993).

E. coli displace themselves by the coordinated actions of 6–10 rotating flagella, driven by rotary motors attached to the cell. The overall movement of a single bacteria is relatively fast; the cell itself is approximately $2\mu\text{m}$ long, and yet is able to swim at speeds of $20\text{--}30\mu\text{m}$ per second, a result of the motors rotating at up to 300 revolutions per second (Bray, 2001). When the flagella rotate in an anticlockwise direction (looking from the outside in towards the bacteria), the bacteria is pushed forward. If the direction of rotation is reversed, the flagella pull the cell in different directions, causing it to tumble and move randomly. In this way, *E. coli* stochastically move away from environments which are hostile towards favourable ones.

The detection of the environment and ensuing response is controlled by a series of chemotaxis specific proteins, usually prefixed with *Che*. Figure 1 shows how the different proteins interact with one another to cause the bacteria to tumble in response to a repellent.

A striking property of the chemotaxis pathway of *E. coli* is that it maintains a high level of sensitivity to its environment through a large range of possible values. The system is remarkably robust (Kollmann et al., 2005), and yet has a relatively simple design. We can break down the pathway into three main categories, each containing proteins specific to their jobs.

Receptors The pathway begins when chemotactic signals are detected by transmembrane sensory receptors that respond to chemical stimuli called chemoreceptors. In particular, these receptors are methyl-accepting chemotaxis proteins (MCPs), in that they show methylation-dependant adaption. *E. coli* has four different types of MCP, and another MCP-like

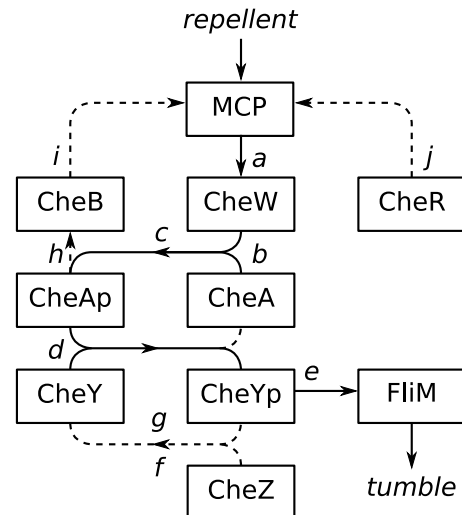


Fig. 1. An informal diagram of the chemotactic signalling pathway in *E. coli* in response to a repellent. Dashed lines represent interactions which do not immediately affect the signalling pathway cascade.

protein, which span the membrane in linked pairs. These MCPs cluster in patches of approximately 200nm in diameter, which are thought to form higher-order arrays at the poles of cells. Interestingly, it has been suggested that MCPs might further pack as trimers of dimers.

Transmitters The MCP is linked to a histidine protein kinase called CheA by an adaptor protein CheW. It is known that CheW is essential for the formation of the quaternary MCP-CheW-CheA complex at the poles of cells, and is essential for signal transduction. However, it is also thought that beyond its role as scaffolding between the MCP and CheA, CheW may also play a more significant role in the process of signal transduction since the structure of CheW is also conserved in other species, such as *R. sphaeroides*. Curiously, there is evidence that a direct MCP-CheA interaction can also occur.

CheA plays a more active role in that it shows varying levels of autophosphorylation which enables the competitive binding of two response regulators, CheY and CheB. In response to a decrease in attractant binding, CheA shows an increased level of autophosphorylation, whilst a decreased level is observed when repellent binds to the MCP. Phosphotransfer from CheA is at a greater rate to CheY than to CheB. This is important since CheY leads to the system response whereas CheB is related to system adaption.

Response regulators The first of the response regulators, CheY, is a flagellar-motor-binding protein. Once phosphorylated by CheA, this protein can interact with FliM in the switch complex of the flagellar motor which then causes the motor to switch direction of rotation. Increased levels of phosphorylated CheY (CheY-P) result in the motor rotating in a clockwise direction, which causes the bacteria to “tumble”. The

default state of the motor in the absence of CheY-P is that the motors rotate in an anti-clockwise direction, which causes the bacteria to “run” in a particular direction. By switching between these two states, the bacteria can stochastically move towards environmental concentrations that are more favourable, and away from those that are hostile.

After a while, the signal to tumble is terminated by the dephosphorylation of CheY-P. Under normal conditions, CheY-P has a half-life of ~ 20 seconds. This time is decreased to ~ 200 milliseconds in the presence of an allosteric activator of CheY dephosphorylation, CheZ. Unsurprisingly, *E. coli* that lack CheZ behave abnormally, and continually tumble since high concentrations of CheY-P accumulate.

Highly methylated MCPs are better able to stimulate CheA phosphorylation. Methyl groups are transferred to an MCP by the methyltransferase CheR, and these are subsequently removed by the methyl-esterase CheB. Varying levels of CheB therefore help the bacteria to adapt to background levels of attractants and repellents.

One way of understanding this complex pathway is by looking at how information received in the MCP is passed down to the FliM. Informally, we can see this in Figure 1 where the solid lines show the cascade of events from the MCP to FliM, and dashed lines represent interactions which do not immediately affect the signalling pathway cascade. Each of the lettered interactions represent an essential element of the model.

- a* The presence of CheW interacting with the MCP.
- b* The presence of CheW interacting with CheA.
- c* Phosphorylation of CheA from the MCP.
- d* Phosphorylation of CheY, coupled with the dephosphorylation of CheA-P.
- e* Interaction between CheY-P and FliM.
- f* Dephosphorylation of CheY-P through interaction with CheZ.
- g* Phosphorylation of CheB, coupled with the dephosphorylation of CheA-P.
- h* Methylation of the MCP, coupled with the dephosphorylation of CheB-P.
- i* Demethylation of the MCP through interaction with CheR.

We can now begin the process of modelling the system. The following section discusses a modelling strategy that not only views the various proteins involved in the system as processes, but the entire system as a process in and of itself. By analysing the behaviour of the systems we produce, we aim to produce better approximations to the real system in order to gain an understanding of some of its properties.

5 A Model of Bacterial Chemotaxis in CSP

Our first model is extremely simple. We require only that the introduction of a repellent causes the bacteria to tumble. Our first approximation of how the system behaves is therefore

$$SYS = repellent \rightarrow tumble \rightarrow SYS$$

This is the coarsest model that we can propose to describe the system, and as we will see later, it makes some unrealistic assumptions about how it can behave.

Further to this, we can begin to describe the system by talking about the interactions of each of the proteins involved. We begin by describing the possible actions of each protein, and how it can return to its original state.

$$\begin{aligned} MCP &= repellent \rightarrow a \rightarrow c \rightarrow MCP \\ CheW &= a \rightarrow b \rightarrow CheW \\ CheA &= b \rightarrow c \rightarrow d \rightarrow CheA \\ CheY &= d \rightarrow e \rightarrow f \rightarrow CheY \\ CheZ &= f \rightarrow CheZ \\ FliM &= e \rightarrow tumble \rightarrow FliM \end{aligned}$$

As individual processes, each of these is well defined and if left unbounded, will happily perform all its actions independently. These processes can then be put together in parallel to produce a more complex system. By placing them in parallel, we constrain their behaviour such that they may only perform certain events that are shared if all other processes are also able to perform those events.⁶

$$\begin{aligned} SYS' &= (((((MCP \parallel CheW) \parallel CheA) \\ &\parallel CheY) \parallel CheZ) \parallel FliM) \setminus \{a, b, c, d, e, f\} \end{aligned}$$

At this point, it is interesting to note that

$$SYS \sqsubseteq SYS'$$

does not hold, since SYS' is able to perform the trace $\langle repellent, repellent \rangle$ for example. However, we do have that $SYS' \sqsubseteq_T SYS$, showing that our initial description SYS only encompasses *some* of the possible behaviour of SYS' .

In actual fact, our initial system was too simplistic, since it stipulates that each *repellent* must be followed by a *tumble*, when it is entirely possible that we receive more than one repellent before a tumble occurs. However, we also want to be able to show that there cannot be more

⁶ In this model we are required to use the *hiding* operator. The process $P \setminus \varepsilon$ is the process which behaves like P , but does not expose communications of the events in the set ε to the environment.

tumbles than repellents. This can be expressed concisely by the statement:

$$s \in \text{traces}(SYS'') \\ \Rightarrow \#(s \upharpoonright \{\text{repellent}\}) \geq \#(s \upharpoonright \{\text{tumble}\})$$

In addition to this property, we would like to ensure that whenever all *tumble* events have been performed in response to *repellent* events, that more *repellent* must be accepted if offered.

$$(s, X) \in \text{failures}(SYS'') \\ \wedge \#(s \upharpoonright \{\text{tumble}\}) = \#(s \upharpoonright \{\text{repellent}\}) \\ \Rightarrow \text{repellent} \notin X$$

We also want to ensure that the system cannot refuse to perform a *tumble*, so long as a *repellent* has been received.

$$(s, X) \in \text{failures}(SYS'') \\ \wedge \#(s \upharpoonright \{\text{tumble}\}) < \#(s \upharpoonright \{\text{repellent}\}) \\ \Rightarrow \text{tumble} \notin X$$

The reader experienced in CSP will recognise these conditions as a modified version of the *buffer* specification. We also require that this is a bounded buffer, since we expect that our model of the system can only take a finite number of states (this must be the case since all the states we have described are finite).

The properties above can all be concisely described by the *characteristic process* that satisfies exactly the conditions that are prescribed. For a buffer that can store up to N elements at once, the characteristic process is:

$$BUFF_0^N = \text{repellent} \rightarrow BUFF_1^N \\ BUFF_N^N = \text{tumble} \rightarrow BUFF_{N-1}^N \square STOP \\ BUFF_n^N = (\text{repellent} \rightarrow BUFF_{n+1}^N \sqcap STOP) \\ \square \text{tumble} \rightarrow BUFF_{n-1}^N$$

Now we can establish $BUFF_0^4 \sqsubseteq_T SYS'$. This is useful since we now know that our system can buffer up to 4 *repellent* events before actually performing a *tumble*. By carefully looking at the system we have described, we can see that this is because 3 other proteins involved in the cascade can be in different states whilst the MCP is receiving a new repellent. With a little intuition we can see that if there are many duplicate MCPs and other proteins that this buffer size is vastly increased.

On the other hand, only $SYS' \sqsubseteq_T BUFF_0^4$ holds. This indicates to us that a buffer may fail in ways that our system will not. By analysing the behaviour of our system further, we notice that a slightly weaker assertion

can be made about the *repellent* events it is willing to refuse; our system cannot *refuse* to perform a *repellent* event unless it is full.

We therefore define a weaker version of *BUFF* that never refuses input unless it is full.

$$BUFF'_0^N = \text{repellent} \rightarrow BUFF'_1^N \\ BUFF'_N^N = \text{tumble} \rightarrow BUFF'_{N-1}^N \square STOP \\ BUFF'_n^N = \text{repellent} \rightarrow BUFF'_{n+1}^N \\ \square \text{tumble} \rightarrow BUFF'_{n-1}^N$$

Notice that this change to the buffer specification is done by simply removing the choice of nondeterministically behaving like *STOP* when performing a *repellent* event. It is therefore not surprising that

$$BUFF_0^N \sqsubseteq BUFF'_0^N$$

We expect this since $BUFF'$ is able to do all the things that $BUFF$ is able to do. In terms of functionality, we would could always use a $BUFF'$ in place of a $BUFF$.

We can now assert that

$$BUFF_0^4 = SYS'$$

since $BUFF_0^4 \sqsubseteq SYS'$ and $SYS' \sqsubseteq BUFF_0^4$. What we have achieved is a proof that the system we are interested in can be concisely described in terms of its behaviour, and that this identifies with that of a buffer.

6 Extending the Model

So far, we have not considered the methylation of the MCP or the interactions involving CheB or CheR. We have also neglected to model the behaviour of CheY appropriately, since our model stipulates that CheY dephosphorylates after every interaction with FliM. We therefore propose a new model which takes these interactions into consideration. Rather than redefining the entire system, we can reuse some of the definitions made earlier and create new processes to describe the more complex interactions.

$$MCP' = MCP \parallel MCP_B \parallel MCP_R \\ MCP_B = i \rightarrow MCP_B \\ MCP_R = j \rightarrow MCP_R \\ CheA' = b \rightarrow c \rightarrow (d \rightarrow CheA' \square h \rightarrow CheA') \\ CheB = h \rightarrow i \rightarrow CheB \\ CheR = j \rightarrow CheR \\ CheY' = d \rightarrow CheY_P \\ CheY_P = e \rightarrow CheY_P \square f \rightarrow CheY' \square g \rightarrow CheY'$$

The definition of $CheY_P$ is interesting in that we could equally have defined it

$$CheY'_P = (e \rightarrow CheY'_P \sqcap STOP) \\ \sqcap f \rightarrow CheY' \sqcap g \rightarrow CheY'$$

One would use this definition if there were some uncertainty about the behaviour of $CheY_P$ in terms of when it decides to no longer interact with $FliM$ through event e . By placing an internal choice between this event and $STOP$, we are describing the system that can decide (for some unknown internal reason) to no longer perform e . This system would then have to offer either f or g and then recurse. Quite clearly, $CheY_P \sqsubseteq_F CheY'_P$ does not hold since $CheY'_P$ is able to refuse event e at any time. However, it turns out that modelling the system using $CheY_P$ or $CheY'_P$ is entirely equivalent, due to interactions with parallel processes.

We can then put these new processes together in parallel with some of the processes previously described to generate a description for a new model.

$$SYS'' = (((((((MCP' \parallel CheW) \parallel CheA') \parallel CheB) \\ \parallel CheR) \parallel CheY') \parallel CheZ) \parallel FliM) \\ \setminus \{a, b, c, d, e, f, g, h, i\}$$

By changing our model, we do not have $SYS'' = BUFF'$, although $SYS'' \sqsubseteq_T BUFF'$ does hold⁷. The problem is that our new definitions have introduced many points at which the system is able to diverge. For example, in the MCP, the system could decide to perform an infinite number of events j before accepting any repellent.

In fact, the system is equivalent to $BUFF''^A_0$, as defined

$$BUFF''^N_0 = DIV \sqcap repellent \rightarrow BUFF''^N_1 \\ BUFF''^N_N = DIV \sqcap repellent \rightarrow BUFF''^N_N \\ \sqcap tumble \rightarrow (BUFF''^N_{N-1} \sqcap BUFF''^N_N) \\ \sqcap STOP \\ BUFF''^N_n = DIV \sqcap repellent \rightarrow BUFF''^N_{n+1} \\ \sqcap tumble \rightarrow (BUFF''^N_{n-1} \sqcap BUFF''^N_n)$$

where DIV is defined:⁸

$$DIV = \mu.X = \tau \rightarrow X \setminus \{\tau\}$$

DIV can be understood as a *diverging* process, one that communicates hidden events forever. This means that

⁷ $SYS'' \sqsubseteq_{FD} BUFF'$ also holds, though this has not been defined since we leave a discussion of divergence until later.

⁸ In this definition we have used a nameless process for convenience. Given a recursion $X = F(X)$, where $F(X)$ is a CSP term involving X , it can be written $\mu X.F(X)$.

the environment could be waiting for this process to terminate or offer to perform an event, when neither may occur.

Clearly something has gone wrong. It is not possible for a bacteria to accept an infinite dose of repellent without tumbling. The problem arises in that we need to take into account the probability that this occurs — it would seem that this is essential to a correct understanding of the system. We will therefore have to use a process algebra that assigns a probability to certain events occurring, with *probabilistic CSP* being the prime candidate. Another clue that we will need to use probabilistic events is in the interaction between $CheB$, $CheR$ and the MCP , which should result in different behaviour when phosphorylating $CheA$.

7 Probabilistic CSP and Future Work

In the previous sections, CSP worked in an environment with no notion of time. One limitation of this is that performance analysis based on probability is not possible without time. Similar observations were made by Zic (1987) concerning both CSP and CCS (a predecessor of π -calculus) for performance specification and verification of protocols. It was then proposed that *Timed CSP* ought to be extended with probabilities, and further work on this was researched by Lowe (1993).

Timed CSP essentially introduces several operators which allow us to better understand how timings can affect the behaviour of a system. It was initially developed by Reed and Roscoe (1988) and allows one to consider how one process might become another by simply allowing time to pass, or by performing an event at a specific time. Operators such as the delay process, written $WAIT\ t$, which waits for t units before changing states, and the timeout operator $P \stackrel{u}{\triangleright} Q$, which initially acts like P , but acts like Q if no synchronisation occurs within time u , allow a much richer description of a system.

In addition to these operators, Lowe (1995) presents us with a means of extending CSP further by removing all non-determinism in order to introduce a *probabilistic choice*.

$$P \text{ } {}_m\sqcap_n \text{ } Q$$

This is the operator which behaves like P with probability $\frac{m}{m+n}$, and like Q with probability $\frac{n}{m+n}$. Other additions include biased external choice and parallel operators and there appears to be no work where these are used when modelling biological systems.

Using these methods would certainly increase the accuracy of our models. However, this would still only allow us to make rudimentary statements about system behaviour. In biological systems, concentrations of

reagents and the rates at which they react are of great importance. By nature, they are continuous and not discrete systems, and this is where CSP fails to capture the key properties of the systems appropriately, since it is as if probabilistic choices are made at every possible point in time. Currently, no obvious way is known of modelling continuous rated behaviour in CSP, and this would certainly be an area of research worth investigating.

For example, an open question in this field would be how to consider the actions of N different processes if they all had the same stochastic pathway of behaviour, but N is very large. An obvious example of such behaviour is in cells where there are thousands of copies of a particular functional protein. An alternative consideration would be finding a way of predicting the behaviour of a single component if what is known is how N such components act in parallel.

8 Conclusion

The field of applying process algebra in systems biology problems appears to be increasingly active. Currently, the modelling methods are dominated by stochastic methods using π -calculus and PEPA, which produce simulations of the systems in question by using variants of Gillespie's algorithm (Regev and Shapiro, 2004; Kwiatkowska et al., 2006) and Markov chains (Calder et al., 2004).

The use of refinement methodology to model these systems in an incremental fashion is an alternative approach to applying process algebra that has not yet been exploited, and this is where CSP has a rich theoretical foundation. This allows the comparison of systems in terms of what how they behave, allowing a more formal approach to the description of system behaviour. This opens up the possibility of directly comparing two systems.

This report has briefly shown the application of refinement in the context of systems biology, and has found the limitations of this method to be in the quality of assertion that can be made about system behaviour.

References

- D. Bray. *Cell Movements: From Molecules to Motility*. Garland Publishing New York, 2001.
- D. Bray, R. Bourret, and M. Simon. Computer simulation of the phosphorylation cascade controlling bacterial chemotaxis. *Mol. Biol. Cell*, 4(5):469–482, 1993.
- M. Calder, S. Gilmore, and J. Hillston. Modelling the influence of RKIP on the ERK signalling pathway using the stochastic process algebra PEPA. *Proceedings of Bio-Concur*, 2004.
- M. Calder, S. Gilmore, and J. Hillston. Automatically deriving ODEs from process algebra models of signalling pathways. *submitted for publication*, 2005.
- W. Fontana and L. Buss. *The barrier of objects: from dynamical systems to bounded organizations*. International Institute for Applied Systems Analysis, 1996.
- D. Gillespie. Exact stochastic simulation of coupled chemical reactions. *The Journal of Physical Chemistry*, 81(25):2340–2361, 1977.
- J. Hillston. *A Compositional Approach to Performance Modelling*. Cambridge University Press, 1996.
- C. Hoare. *Communicating Sequential Processes*. Series in Computer Science, 1985.
- M. Kollmann, L. Lovdok, K. Bartholome, J. Timmer, and V. Sourjik. Design principles of a bacterial signalling network. *Nature*, 438, 2005.
- M. Kwiatkowska, G. Norman, D. Parker, O. Tymchyshyn, J. Heath, and E. Gaffney. Simulation and verification for computational modelling of signalling pathways. *Proceedings of the 2006 Winter Simulation Conference (to appear)*, 2006.
- G. Lowe. *Probabilities and Priorities in Timed CSP*. PhD thesis, D. Phil thesis, Oxford, 1993.
- G. Lowe. Probabilistic and Prioritized Models of Timed CSP. *TCS*, 138(2):315–352, 1995.
- 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.
- G. Reed and A. Roscoe. A timed model for communicating sequential processes. *Theoretical Computer Science*, 58(1-3):249–261, 1988.
- A. Regev. Representation and simulation of molecular pathways in the stochastic-calculus. *2nd Workshop on Computation of Biochemical Pathways and Genetic Networks*, 2001.
- A. Regev and E. Shapiro. Cellular abstractions: Cells as computation. *Nature*, 419:343, 2002.
- A. Regev and E. Shapiro. The π -calculus as an abstraction for biomolecular systems. *Modelling in Molecular Biology*. Springer, 2004.
- A. Regev, W. Silverman, and E. Shapiro. Representing biomolecular processes with computer process algebra: π -calculus programs of signal transduction pathways. *Proceedings of the Pacific Symposium of Biocomputing*, 2000.
- A. Regev, W. Silverman, and E. Shapiro. Representation and simulation of biochemical processes using the π -calculus process algebra. *Pacific Symposium on Biocomputing*, 6:459470, 2001.
- A. Roscoe. *The theory and practice of concurrency*. Prentice Hall, 1998.
- G. Wadhams and J. Armitage. Making sense of it all: bacterial chemotaxis. *Nature Reviews Molecular Cell Biology*, 5(12):1024–1037, 2004.
- J. J. Zic. Extensions to communicating sequential processes to allow protocol performance specification. *SIGCOMM Comput. Commun. Rev.*, 17(5):217–227, 1987. ISSN 0146-4833.

A CSP code listing

```
channel a,b,c,d,e,f,g,h,i,j,repellent,tumble,tau
epsilon = {a,b,c,d,e,f,g,h,i,j}
```

```
-- The first model is extremely simplistic and only designed
-- to introduce the idea of process development
SYS = repellent -> tumble -> SYS
```

```
-- The second model ignores methylation in the MCP
MCP = repellent -> a -> c -> MCP
CheW = a -> b -> CheW
CheA = b -> c -> d -> CheA
CheY = d -> e -> f -> CheY
CheZ = f -> CheZ
FliM = e -> tumble -> FliM
```

```
SYS' = (((((MCP [{repellent,a,c}||{a,b}] CheW)
[{{repellent,a,b,c}||{b,c,d}}] CheA)
[{{repellent,a,b,c,d}||{d,e,f}}] CheY)
[{{repellent,a,b,c,d,e,f}||{e,tumble}}] FliM)
[{{repellent,a,b,c,d,e,f,tumble}||{f}}] CheZ)
```

```
-- Here we show that SYS is a trace refinement of SYS'
assert SYS [T= SYS' \ epsilon --F
assert SYS [F= SYS' \ epsilon --F
assert SYS [FD= SYS' \ epsilon --F
assert SYS' \ epsilon [T= SYS --T
assert SYS' \ epsilon [F= SYS --F
assert SYS' \ epsilon [FD= SYS --F
```

```
-- The specification is a modified buffer such that the
-- channels left and right are replaced with events
-- repellent and tumblxxx respectively.
BUFF(s,N) = if (s==0) then (repellent -> BUFF(1,N))
else ((tumble -> BUFF(s-1,N))
[] (STOP |~| (if (s==N) then STOP
else (repellent -> BUFF(s+1,N))))))
```

```
-- These asserts tell us just quite how similar our model is
-- to our specification.
assert BUFF(0,4) [T= SYS' \ epsilon --T
assert BUFF(0,4) [F= SYS' \ epsilon --T
assert BUFF(0,4) [FD= SYS' \ epsilon --T
assert SYS' \ epsilon [T= BUFF(0,4) --T
assert SYS' \ epsilon [F= BUFF(0,4) --F
assert SYS' \ epsilon [FD= BUFF(0,4) --F
```

```
-- This slightly weaker specification is that of a buffer
-- that never refuses input when it is not full. It is an
-- equivalent process to our system.
```

```
BUFF'(s,N) = if (s==0) then (repellent -> BUFF'(1,N))
else ((tumble -> BUFF'(s-1,N))
[] ((if (s==N) then STOP
else (repellent -> BUFF'(s+1,N))))))
```

```
assert BUFF'(0,4) [T= SYS' \ epsilon --T
assert BUFF'(0,4) [F= SYS' \ epsilon --T
assert BUFF'(0,4) [FD= SYS' \ epsilon --T
assert SYS' \ epsilon [T= BUFF'(0,4) --T
assert SYS' \ epsilon [F= BUFF'(0,4) --T
assert SYS' \ epsilon [FD= BUFF'(0,4) --T
```

```
-- The third model introduces methylation in the MCP
```

```
-- The MCP should be able to perform i or j at any time,
-- choice isn't right therefore
--MCP = repellent -> MCP'
--MCP' = (a -> c -> MCP [] i -> MCP' [] j -> MCP')
-- and
--MCP = repellent -> a -> c -> MCP [] i -> MCP [] j -> MCP
-- are invalid models
```

```
MCP' = MCP ||| MCPb ||| MCPc
MCPb = i -> MCPb
MCPc = j -> MCPc
CheA' = b -> c -> (d -> CheA' [] h -> CheA')
CheB = h -> i -> CheB
CheR = j -> CheR
-- The initial version of CheY was too simple in that
-- a CheY-P may cause more than one tumble, but eventually
-- it will be dephosphorylated.
--CheY' = d -> e -> (f -> CheY' [] g -> CheY')
CheY' = d -> CheYp
--CheYp = (STOP |~| e -> CheYp) [] f -> CheY' [] g -> CheY'
CheYp = e -> CheYp [] f -> CheY' [] g -> CheY'
```

```
CheY'' = d -> CheYp'
CheYp' = (STOP |~| e -> CheYp') [] f -> CheY' [] g -> CheY'
```

```
SYS'' = ((((((MCP' [{repellent,a,c,i,j}||{a,b}] CheW)
[{{repellent,a,b,c,i,j}||{b,c,d,h}}] CheA')
[{{repellent,a,b,c,d,h,i,j}||{d,e,f,g}}] CheY')
[{{repellent,a,b,c,d,e,f,g,h,i,j}||{e,tumble}}] FliM)
[{{repellent,a,b,c,d,e,f,g,h,i,j,tumble}||{f}}] CheZ)
[{{repellent,a,b,c,d,e,f,g,h,i,j,tumble}||{h,i}}] CheB)
[{{repellent,a,b,c,d,e,f,g,h,i,j,tumble}||{j}}] CheR)
```

```
SYS''' = (((((((MCP' [{repellent,a,c,i,j}||{a,b}] CheW)
[{{repellent,a,b,c,i,j}||{b,c,d,h}}] CheA')
[{{repellent,a,b,c,d,h,i,j}||{d,e,f,g}}] CheY')
[{{repellent,a,b,c,d,e,f,g,h,i,j}||{e,tumble}}] FliM)
[{{repellent,a,b,c,d,e,f,g,h,i,j,tumble}||{f}}] CheZ)
[{{repellent,a,b,c,d,e,f,g,h,i,j,tumble}||{h,i}}] CheB)
[{{repellent,a,b,c,d,e,f,g,h,i,j,tumble}||{j}}] CheR)
```

```
assert SYS'' \epsilon [T= SYS'' \epsilon --F
assert SYS'' \epsilon [F= SYS'' \epsilon --F
assert SYS'' \epsilon [FD= SYS'' \epsilon --F
assert SYS'' \epsilon [T= SYS'' \epsilon --T
assert SYS'' \epsilon [F= SYS'' \epsilon --F
assert SYS'' \epsilon [FD= SYS'' \epsilon --F
```

```
assert CheYp [T= CheYp'
assert CheYp [F= CheYp'
assert CheYp [FD= CheYp'
assert CheYp' [T= CheYp
assert CheYp' [F= CheYp
assert CheYp' [FD= CheYp
```

```
assert BUFF'(0,4) [T= SYS'' \ epsilon --F
assert BUFF'(0,4) [F= SYS'' \ epsilon --F
assert BUFF'(0,4) [FD= SYS'' \ epsilon --F
assert SYS'' \ epsilon [T= BUFF'(0,4) --T
assert SYS'' \ epsilon [F= BUFF'(0,4) --F
assert SYS'' \ epsilon [FD= BUFF'(0,4) --T
```

```
-- This is a terrible buffer, it may diverge at any time,
DIV = let X = tau -> X
within X\{tau}
BUFF''(0,N) = DIV
[] repellent -> BUFF''(1,N)
BUFF''(n,N) = DIV
```

```

[] tumble -> (BUFF''(n-1,N) [] BUFF''(n,N))
[] if (n==N)
then STOP [] repellent -> BUFF''(N,N)
else repellent -> BUFF''(n+1,N)

BUFF''(0,N) = repellent -> (BUFF''(1,N) |~| BUFF''(0,N))
|~| BUFF''(0,N)
BUFF''(n,N) = tumble -> BUFF''(n-1,N)
[] tumble -> BUFF''(n,N)
[] if (n==N)
then STOP [] repellent -> BUFF''(N,N)
else (repellent -> BUFF''(n+1,N))

assert BUFF''(0,4) [T= BUFF''(0,4) --T
assert BUFF''(0,4) [F= BUFF''(0,4) --F
assert BUFF''(0,4) [FD= BUFF''(0,4) --T
assert BUFF''(0,4) [T= BUFF''(0,4) --T
assert BUFF''(0,4) [F= BUFF''(0,4) --T
assert BUFF''(0,4) [FD= BUFF''(0,4) --T

```

```

-- Unfortunately, that's the kind of system we're dealing with
assert BUFF''(0,4) [T= SYS'' \ epsilon --T
assert BUFF''(0,4) [F= SYS'' \ epsilon --T
assert BUFF''(0,4) [FD= SYS'' \ epsilon --T
assert SYS'' \ epsilon [T= BUFF''(0,4) --T
assert SYS'' \ epsilon [F= BUFF''(0,4) --T
assert SYS'' \ epsilon [FD= BUFF''(0,4) --T

```

```

assert BUFF''(0,4) [T= BUFF(0,4) --T
assert BUFF''(0,4) [F= BUFF(0,4) --F
assert BUFF''(0,4) [FD= BUFF(0,4) --T

```

```

assert BUFF(0,4) [T= BUFF'(0,4) --T
assert BUFF(0,4) [F= BUFF'(0,4) --T
assert BUFF(0,4) [FD= BUFF'(0,4) --T

```