Modelling and Analysis of Structure in Cellular Signalling Systems

Robin Donaldson

Submitted in fulfilment of the requirements for the Degree of

Doctor of Philosophy

School of Computing Science
College of Science and Engineering
University of Glasgow

August 2012

Abstract

Cellular signalling is an important area of study in biology. *Signalling pathways* are well-known abstractions that explain the mechanisms whereby cells respond to signals. Collections of pathways form *signalling networks*, and interactions between pathways in a network, known as *cross-talk*, enables further complex signalling behaviours. Increasingly, computational modelling and analysis is required to handle the complexity of such systems.

While there are several computational modelling approaches for signalling pathways, none make cross-talk explicit. We present a modular modelling framework for pathways and their cross-talk. Networks are formed by composing pathways: different cross-talks result from different synchronisations of reactions between, and overlaps of, the pathways. We formalise five types of cross-talk and give approaches to reason about possible cross-talks in a network.

The complementary problem is how to handle *unstructured* signalling networks, i.e. networks with no explicit notion of pathways or cross-talk. We present an approach to better understand unstructured signalling networks by modelling them as a set of signal flows through the network. We introduce the Reaction Minimal Paths (RMP) algorithm that computes the set of signal flows in a model. To the best of our knowledge, current algorithms cannot guarantee both correctness and completeness of the set of signal flows in a model. The RMP algorithm is the first.

Finally, the RMP algorithm suffers from the well-known state space explosion problem. We use suitable partial order reduction algorithms to improve the efficiency of this algorithm.

Contents

Acknowledgements

First I wish to thank my supervisor, Muffy Calder, and my (unofficial) co-supervisor, Carolyn Talcott. Both excellent scientists have taught me so much, while allowing me the academic freedom that made this process so enjoyable. Thanks for everything.

I wish to thank my thesis examiners, Rainer Breitling and Stephen Gilmore, for their time in reading this thesis. Their suggestions greatly enhanced this thesis.

I wish to thank David Gilbert and Monika Heiner for introducing me to the research area of Computational Biology.

Declaration

This thesis is submitted in accordance with the rules for the degree of Doctor of Philosophy at the University of Glasgow. None of the material contained herein has been submitted for any other degree. The material contained herein is the work of myself unless stated otherwise.

Below is a list of my publications. The work in Chapter 5 has been published in [A, C] and the work in Chapter 6 has been published in [B].

Robin Donaldson

[A] **R. Donaldson** and M. Calder (2012). *Modular modelling of signalling pathways and their cross-talk.* Theoretical Computer Science (TCS).

[B] **R. Donaldson**, C. Talcott, M. Knapp and M. Calder (2010). *Understanding Signalling Networks as Collections of Signal Transduction Pathways.* ACM (Proc. of 8th International Computational Methods in Systems Biology) pp. 86–95.

[C] **R. Donaldson** and M. Calder (2010). *Modelling and Analysis of Biochemical Signalling Pathway Cross-talk.* EPTCS (Proc. of 3rd Workshop From Biology To Concurrency and back) 19, pp. 40–54.

[D] R. Breitling, **R. Donaldson**, D. Gilbert and M. Heiner (2010). *Biomodel Engineering – From Structure to Behavior.* Position paper in Trans. on Computational Systems Biology XII, Springer Lecture Notes in Bioinformatics 5945, pp. 1–12.

[E] D. Gilbert, R. Breitling, M. Heiner and **R. Donaldson** (2009). *An introduction to BioModel Engineering, illustrated for signal transduction pathways.* 9th International Workshop on Membrane Computing. Lecture Notes in Computer Science 5391, pp. 13-28.

[F] **R. Donaldson** and D. Gilbert (2008). *A Model Checking Approach to the Parameter Estimation of Biochemical Pathways.* Lecture Notes in Computer Science (Proc. of 6th international Computational Methods in Systems Biology) 5307 pp. 269-287.

[G] **R. Donaldson** and D. Gilbert (2008). *A Monte Carlo Model Checker for Probabilistic LTL with Numerical Constraints.* Technical Report TR-2008-282 at the Dept. of Computer Science at University of Glasgow.

[H] M. Heiner, **R. Donaldson** and D. Gilbert (2008). *Petri Nets for Systems Biology.* In "Symbolic Systems Biology: Theory and Methods," edited by Sriram Iyengar. Jones and Bartlett publishers.

[I] M. Heiner, D. Gilbert, and **R. Donaldson** (2008). *Petri Nets for Systems and Synthetic Biology.* In Formal Methods for Computational Systems Biology. Lecture Notes in Computer Science 5016, pp. 215-264.

Chapter 1

Introduction

Systems biology Systems biology [68, 108] is fast emerging as the new approach to understand the complex behaviour of biological systems by analysing the interaction of numerous components of the system simultaneously. The success of systems biology is made possible by advances in both high-throughput laboratory techniques [5] and computational modelling and analysis [84]. Typical areas of study in systems biology include metabolic networks [39], gene regulation networks [73] and cellular signalling [8], which is the focus of this thesis.

Cellular signalling Cellular signalling is the mechanism by which cells communicate with other cells and detect and respond to the environment [29]. Signalling is initiated by a ligand (signal molecule)[1] binding to a receptor (typically on the cell membrane), and thus the presence of the ligand is detected. This initiates a series of biochemical reactions within the cell resulting in a cellular response. The network of biochemical reactions that generate cellular responses to ligands is called a *signalling network*. The steps involved in a cell detecting and responding to a signal are shown in Figure 1.1.

Signalling pathway and cross-talk abstraction The current study of cellular signalling is largely based on an abstraction of signalling networks to *signalling pathways* and *cross-talk*. A signalling pathway is the biochemical reactions that generate the cellular response for one type of signal. Biologists have organised signalling networks into a number of well-known signalling pathways. The interaction between two or more signalling pathways is called cross-talk. Biologists handle the complexity involved in studying cellular signalling by only considering signalling pathways in isolation or a small number of signalling pathways with cross-talk.

[1] signalling can also be initiated without a ligand (e.g. photons, biochemical stress, changes in temperature) though this is not covered in this thesis

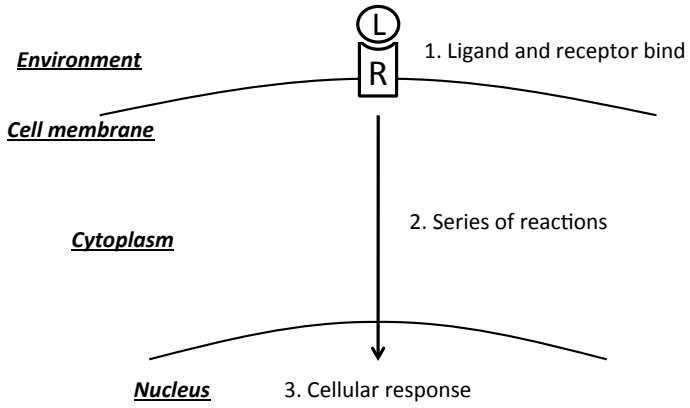


Figure 1.1: An example of the steps involved in a cell detecting and responding to a signal (ligand). The receptor and ligand bind, initiating a series of reactions within the cell, resulting in the cellular response. There are variations on this example, e.g. signalling can be initiated in the cytoplasm.

Computational modelling and analysis Computational modelling and analysis plays an important role in the study of cellular signalling. A model of a signalling network can be used to generate hypotheses of the mechanism behind certain diseases. For example, cancer can be caused by a mutation in a signalling network causing uncontrolled growth in the number of cells in response to a signal [54]. Applying suitable analysis techniques to a signalling network model, targets for therapeutic intervention can be identified, aiding the drug discovery process.

Models of cellular signalling are either unicellular or multicellular. Unicellular models are concerned only with the signalling mechanism within one cell, whereas multicellular models also consider the communication and interaction between different cells. This thesis considers unicellular models.

Problem: signalling pathway cross-talk There are a wide range of modelling paradigms in the literature. Early models of signalling pathways were written using flat systems of equations [8]. More recently, modelling languages have been used to describe signalling pathways [10]. These languages provide an abstraction from the implementation detail (such as the mathematical equations). Several modelling languages have a modular approach in which they describe systems in terms of their components and the interaction between components. For example, process algebras [12] have been used to model signalling pathways, with molecules as a processes and reactions as communication between processes. However, no modelling language has made explicit,

or permits reasoning about, pathway cross-talk.

Problem: unstructured signalling networks The complementary problem is how to understand unstructured models of signalling networks, i.e. models with no explicit notion of pathways or cross-talk. A particularly interesting example is Pathway Logic [102]. At the core of Pathway Logic is a knowledge base of known biochemical reactions. Models are generated automatically from the knowledge base by defining the initial conditions for a cell of interest (i.e. the proteins, ligands, receptors that are present). These models are typically larger and more complex than models that are manually built. However, they can be harder to understand and analyse due to their size, complexity and lack of structure.

Thesis contributions In this thesis we give solutions to both problems.

First, we describe a new framework for modelling and analysing signalling networks formally. The framework is based on generic biological modules that have programmable interfaces defining how they can be connected. A signalling pathway is built by connecting modules, and a signalling network is built from a set of pathways with optional cross-talk. Within this work we have developed a formal definition of cross-talk, including a novel categorisation based on the use of process algebraic operators. The resulting models of signalling networks are analysed by model checking temporal properties. Cross-talk can be detected and characterised, and the effect of the cross-talk measured, by different temporal properties. Formally defining cross-talk permits us to express different instances of cross-talk. We give an algorithm to enumerate all possible instances of cross-talk between a set of pathways, thus generating different network hypotheses that can be tested against data. We apply this framework to a prominent case study: the network of the TGF-β/BMP, WNT and MAPK pathways and their cross-talk. This part of the thesis is illustrated schematically in Figure 1.2.

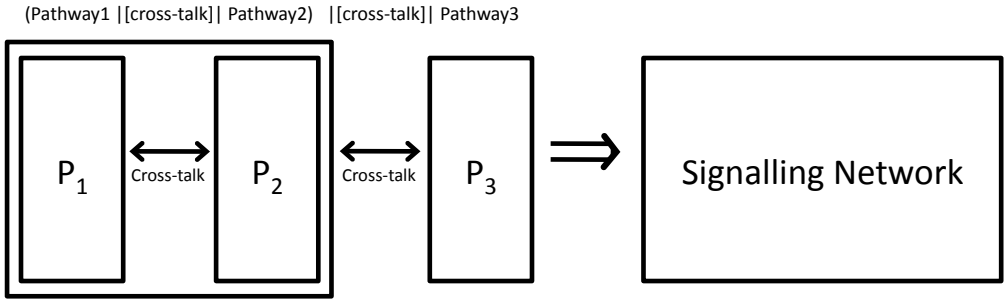


Figure 1.2: Modelling signalling pathways and cross-talk to create a signalling network.

Second, we give an approach to model *unstructured* signalling network models as a set of signal flows. A signal flow is a series of biochemical reactions from a set of inputs (e.g. ligands) to a set of outputs (e.g. activated proteins). We find that current techniques are unsuitable to compute the set of signal flows in a network; we introduce a new algorithm based on exploring the state space (all possible behaviours) of a model. The algorithm is applied to signalling network models generated from the Pathway Logic knowledge base. We also define techniques to handle *large* unstructured signalling networks, in particular, more efficient versions of the signal flows algorithm using partial order reduction [69]. We apply the approach to analyse a larger, more complex version of the signalling network model generated from the Pathway Logic knowledge base. This part of the thesis is illustrated schematically in Figure 1.3.

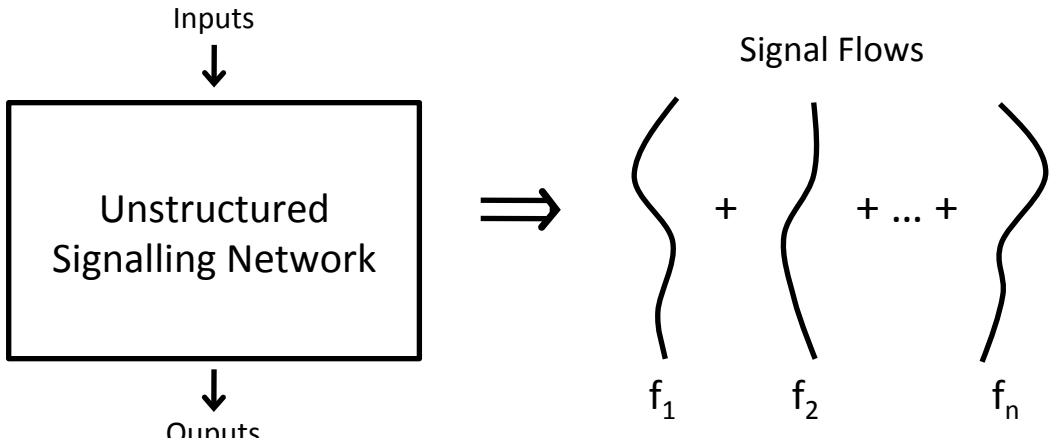


Figure 1.3: Modelling unstructured signalling networks as a set of signal flows.

In developing the solution to the first problem (signalling pathway cross-talk), we use continuous time Markov chains defined with an extension of the PRISM modelling language [71]. In the solution to the second problem (unstructured signalling networks), we use Petri nets [88]. However, both solutions provide generic techniques and the choice of underlying model is not important.

This thesis is concerned with qualitative models, with reactions rates and stoichiometry 1, and strictly presence or absence of biochemical entities. There may be limitations because of this, for example we are not able to express dimerisation, $A + A- > 2A$. We use qualitative models because quantitative models are often difficult to create, especially due to the difficulty in measuring the rates of biochemical reactions. Future work would be to extend this work to quantitative models.

Our methods and approaches are applicable to man-made abstractions of cellular signalling systems (i.e. models), rather than directly to the systems themselves. As one would expect, there is likely significant detail lost in dealing only with the abstractions of the systems.

Thesis statement The role of computational modelling and analysis in systems biology is now well-established. Modelling paradigms range from systems of equations to modelling languages. However, no current modelling paradigm deals with structural information such as pathway cross-talk. Furthermore, approaches to understand unstructured signalling networks are lacking. We propose two modelling and analysis approaches to handle these important, but different, problems. A signalling network can be modelled in a structured manner with pathways as entities and cross-talk as interactions between these entities. Unstructured signalling networks can be modelled as a set of signal flows. We demonstrate the utility of each approach by answering biological questions in prominent case studies.

Organisation of this thesis This thesis is organised as follows. Chapter 2 gives the biological background and Chapter 3 gives the computational background required for this thesis. Chapter 4 reviews the related computational work on cellular signalling. Chapter 5 describes the modelling framework for signalling pathway cross-talk and gives results of the application to a prominent case study. Chapter 6 describes the modelling approach to understand unstructured signalling networks as a set of signal flows and gives results of the application to Pathway Logic. Chapter 7 describes an approach to understand large unstructured signalling networks and gives results of the application to the Pathway Logic. Chapter 8 gives directions for future work and Chapter 9 concludes this thesis.

Chapter 2

Biological background

In this chapter we introduce the required biological background for this thesis.

In Section 2.1 we give basic biological principles and related graphical notation. Section 2.2 gives an overview of the research area of systems biology. In Section 2.3 we introduce cellular signalling, an important area of study in systems biology. Cellular signalling is studied using an abstraction to signalling pathways and cross-talk. In Section 2.4 and Section 2.5 we explore further the concepts of signalling pathways and cross-talk respectively.

2.1 Biological principles and graphical notation

In this section we explain the basic biological principles used in this thesis and where appropriate we give the related graphical notation.

Protein A *protein* is a chain of amino acids folded into a 3-dimensional structure. Most proteins have a specific biological function that is affected by its structure. A protein is represented by an oval with the name of the protein inside. A protein that is initially present is shaded grey.

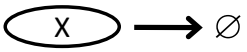
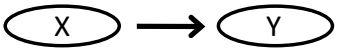
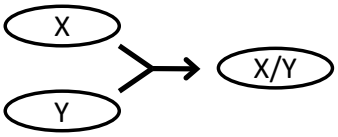
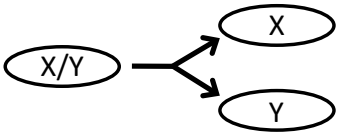
Reactions A *reaction*[1] turns a set of molecules (called *substrates*) into another set of molecules (called *products*). We distinguish five types of *reactions*.

- **Production:** a protein is created.

[1] we consider only protein-based reactions

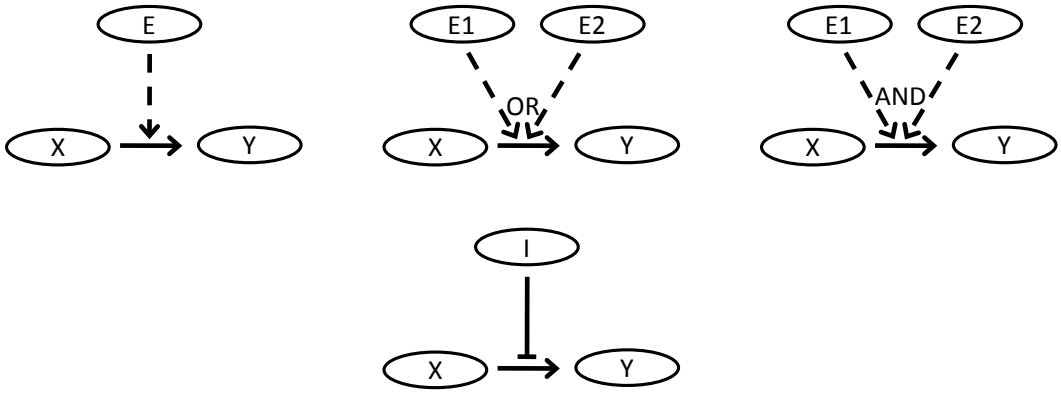
- **Degradation:** a protein is broken down and recycled by a cell.
- **Transformation:** a protein changes state or moves location.
- **Complexation:** two or more proteins bind to form a complex.
- **Decomplexation:** a complex breaks up into its constituent proteins.

A reaction is represented by a solid line with an arrow. The type of reaction is given by the arity of the arc (e.g. an arc with arity $N \to 1$ is a complexation reaction). X/Y denotes a complex of X and Y.

Reaction	Example	Arity
Production		$0 \to 1$
Degradation		$1 \to 0$
Transformation		$1 \to 1$
Complexation		$N \to 1$
Decomplexation		$1 \to N$

Enzyme An *enzyme* is a protein that increases the rate of a reaction (the enzyme *catalyses* the reaction). The substrate binds to the enzyme's *active site* which lowers the energy required for the reaction. An enzyme is represented by a dashed arrow from the enzyme to the reaction (this is called an *enzymatic arc*). There is sometimes more than one enzyme for a reaction. We use the following abstraction. If the text "OR" is between the enzymatic arcs then, if any enzyme is present, the rate of the reaction increases. Otherwise, the text "AND" is between the arcs and all enzymes are required to be present.

Inhibitor An *inhibitor* is a protein that decreases the rate of a reaction by binding to and blocking the enzyme's active site. An inhibitor is represented by a solid line with a blunt end from the inhibitor to the reaction (this is called an *inhibitory arc*).



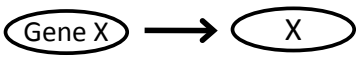
Active protein A protein can be *active*, meaning that a particular function has been enabled. A group of atoms can be added to a particular site on a protein that changes the structure, and therefore the function, of the protein. *Phosphorylation* is the addition of a phosphate group to a protein that is common in cellular signalling. *Dephosphorylation* is the removal of a phosphate group from a protein. Typically, but not always, phosphorylation activates a protein and dephosphorylation deactivates a protein. In this thesis we often only distinguish between inactive and active proteins rather than the various mechanisms by which a protein changes state. An active protein is decorated with $*$.

Gene A *gene* is a segment of genomic DNA that can be *expressed* to create a specific protein (this process is called *gene expression*). A gene is represented by an oval with "Gene" followed by the name of the gene inside. Gene expression is represented by an arrow from a gene to a protein with the same name as the gene.

Transcription factor A *transcription factor* is a protein that regulates (controls the rate of) gene expression. A transcription factor has an enzymatic or inhibitory arc to a gene expression reaction, either *up-regulating* or *down-regulating* gene expression respectively. Like other proteins, transcription factors may need to be activated to enable their function.

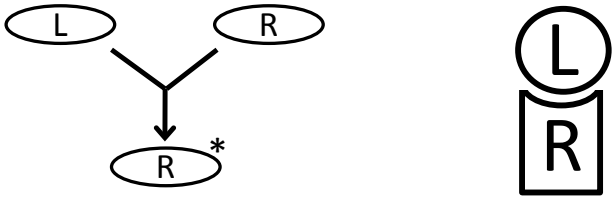
Ligand A *ligand* is a biochemical signal, of which there are many types.

Receptor A *receptor* is a molecule that binds to a ligand.



Ligand-receptor binding Ligand-receptor binding is represented as a complexation reaction of the ligand and receptor, represented as proteins, forming a complex where the receptor protein is said to have been activated.

If the process of ligand-receptor binding is not of importance to the diagram, ligand-receptor binding can be represented in an alternative manner. A receptor is represented by a rectangle with a concave edge with the ligand, represented as a circle, next to it. The names of the receptor and ligand are either inside or beside the receptor or ligand respectively.

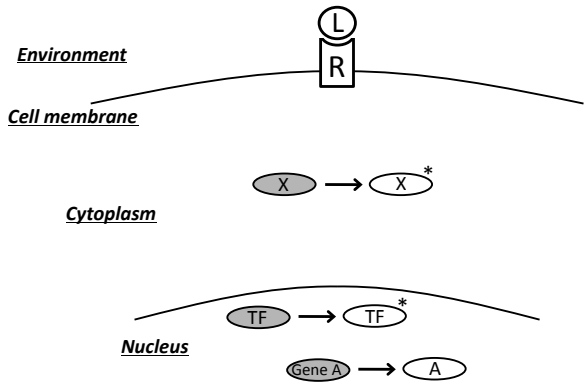


Cellular locations Of the several locations in a typical cell, we refer mainly to four locations.

- **Environment:** the space outside the cell, containing the ligands.
- **Cell membrane:** the space between the environment and the inside of the cell, typically containing the receptors.
- **Cytoplasm:** the space inside the cell (excluding the nucleus) and where many of the protein reactions occur.
- **Nucleus:** the space inside the cell that contains the genes.

Translocation is the movement of a molecule between locations, e.g. a transcription factor gets activated in the cytoplasm and then translocates to the nucleus to regulate gene expression. Translocation is represented by a reaction from a protein in one location to the same protein in a different location.

Biochemical species A *biochemical species* is a type of molecule in a biochemical system, e.g. protein X, active protein Y or receptor Z.



2.2 Systems biology

Systems biology [68, 108] is fast emerging as the new approach to understand the complex behaviour of biological systems by analysing the interaction of numerous components of the system simultaneously. The success of systems biology is made possible by advances both in high-throughput laboratory techniques [5] and computational modelling and analysis [84].

There are several areas of study in systems biology, including:

- Cellular metabolism (e.g. [39]), the network of reactions that consume and produce biochemical species which are essential to maintain life.
- Genetic regulation (e.g. [73]), the network of interactions between proteins and segments of DNA/RNA that control the rate of gene expression.
- Cellular signalling (e.g. [8]), the network of reactions that govern how a cell responds to signals.

In this thesis we focus on cellular signalling.

2.3 Cellular signalling

Cellular signalling is the mechanism by which cells communicate with other cells and detect and respond to the environment. A cell detects a ligand by the ligand binding to one of the cell's receptors. This initiates a series of reactions within the cell, eventually causing a cellular response

to the ligand. Typical cellular responses include proliferation (increase in the number of cells), apoptosis (cell death) and cellular differentiation (changing cell type).

The cellular responses are governed by a large, complex network of reactions called a *signalling network*. Biologists study signalling networks by measuring the change in concentration of several biochemical species in a cell (typically proteins and their activated form) after being treated with a ligand, to infer the reactions that cause the cellular response. For example, in [5] a high-throughput technique is used to analyse 518 protein interactions under various signalling conditions to infer the reactions that cause the cellular response. By repeating these experiments while inhibiting proteins that are thought to be important in generating the cellular response, biologists can validate their hypotheses. These hypotheses are often communicated using informal diagrams that we call *biological cartoons*. A biological cartoon depicting a signalling network is shown in Figure 2.1.

Because signalling networks are large and complex, the study of cellular signalling is often based on an abstraction to *signalling pathways* and *cross-talk*. We summarise the basic concepts of a signalling network, signalling pathways and cross-talk below.

- **Signalling network:** a large, complex network of biochemical reactions that generate cellular responses to ligands.
- **Signalling pathway:** the biochemical reactions that generate the cellular response for one type of signal.
- **Cross-talk:** the interaction between two or more signalling pathways.

We now explore the concepts of signalling pathways and cross-talk further.

2.4 Signalling pathways

A signalling pathway[2] is an abstraction used by biologists. We focus on the most common usage of the term signalling pathway, the biochemical reactions that generate the cellular response for one type of signal. There are a number of well-known signalling pathways, e.g. the Egf and Ngf pathways [8], the Interferon pathway [89] and the Integrin pathway [98].

The basic outline of a pathway is as follows. A ligand is detected by binding to one of the cell's receptors. A receptor is typically located on the cell membrane. Some receptors (called intracellular receptors) exist within the cytoplasm and detect ligands that have passed through the

[2] often shortened to pathway

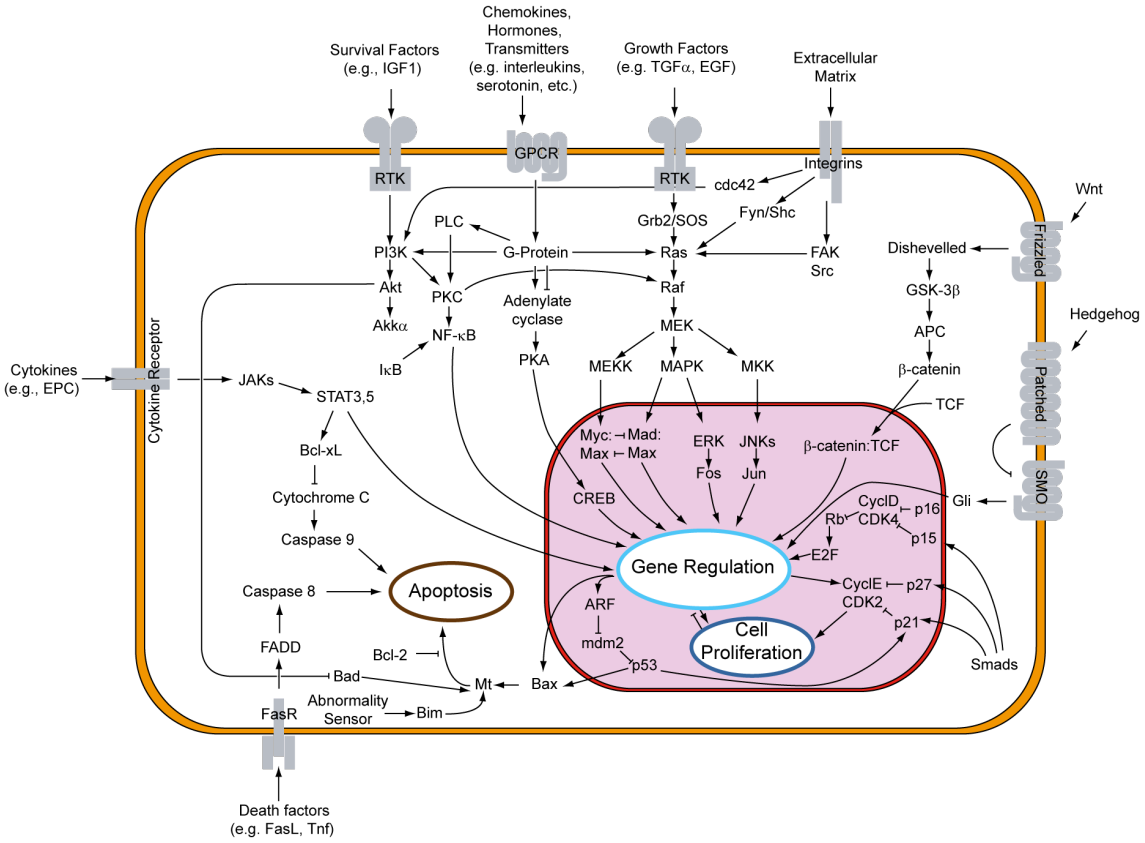


Figure 2.1: A biological cartoon of a signalling network. Figure reproduced from [75].

cell membrane [65]. Ligand-receptor binding initiates a series of reactions within the cell. These reactions are often enzyme-driven complexation, decomplexation, phosphorylation or dephosphorylation reactions, and cause a change in the activity of proteins and ultimately transcription factors. The activity of the transcription factors affects the rate of gene expression and causes the cellular response. An example basic pathway is shown in Figure 2.2.

Protein activation reactions are sometimes arranged in a *signalling cascade* in which an active protein is the enzyme for the activation reaction of another type of protein, and so on. The number of activation reactions involved in the cascade is the number of *stages*, hence a 3-stage cascade is the activation of protein X, that is an enzyme for the activation of protein Y, that is an enzyme for the activation of protein Z. The effect of the signalling cascade is to amplify the response to a small amount of signal. An example of a 3-stage cascade is shown in Figure 2.2.

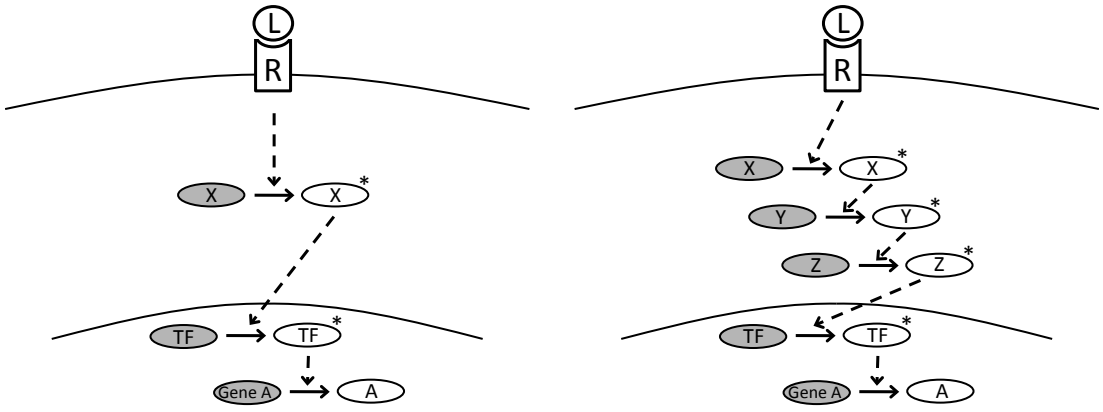


Figure 2.2: (left) a basic signalling pathway. (right) a signalling pathway including a 3-stage cascade.

The reactions that comprise a pathway were first thought to be a linear chain of reactions. However, more recently, detailed understanding of these pathways shows that they are nonlinear [104]. The reactions can diverge and the products of the reactions can catalyse or inhibit other reactions, perhaps forming a feedback or feedforward loop. This is illustrated further in Figure 2.3.

Signal flow [36] (also called signal propagation [20] or signal transduction [49]) is a term used by biologists for the reactions starting from the cell detecting a ligand and ending in a change in some output(s), e.g. protein activation, transcription factor activation or gene expression. The *rate of signal flow* is some measure of how fast these reactions are.

We note that signalling pathways are a human abstraction. Although there is a well-known set

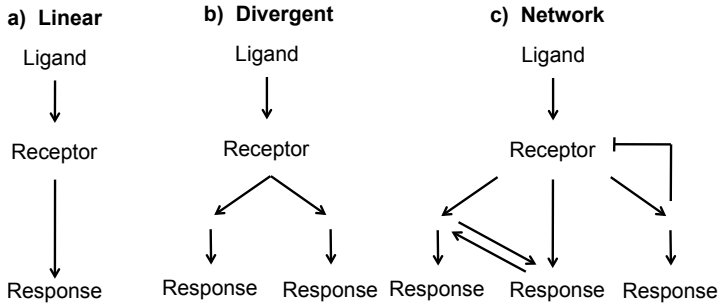


Figure 2.3: Signalling pathways were first thought to be a) linear, then b) divergent, and now c) nonlinear. Note that c) contains an inhibitory feedback loop in which the product of one reaction inhibits the receptor. Figure reproduced from [104].

of pathways, there is a lack of rigorous definitions of what constitutes a single pathway—it is an inexact process to define the boundaries between pathways. There are several signalling pathways in the signalling network cartoon in Figure 2.1, including the Egf, Tnf and Wnt pathways. The exact boundaries between these pathways are unclear.

The term signalling pathway is used in two other ways that we call protein- and response-centric pathways.

Protein-centric pathways There are several signalling pathways whose focus is explaining the activity of a protein. For example, the NF-κB pathway [34] and the MAPK/Erk pathway [92] contain the signalling reactions that affect the activity of the NF-κB and MAPK/Erk proteins respectively.

Response-centric pathways There are several signalling pathways whose focus is explaining a cellular response. For example, apoptosis pathway [22] contains the signalling reactions that affect the apoptosis response.

These pathways attempt to explain the activity of a protein or a cellular response rather than the cellular response to a single type of ligand. These pathways are more likely to overlap and the ligands that affect the protein/response are often different depending on the biologist and data. Therefore, they are not a basic unit of study in cellular signalling and we do not focus on these pathways in this thesis.

2.5 Cross-talk

The term cross-talk was first applied to electronic circuits to describe a signal in one circuit having an undesired effect on another circuit [18]. Cross-talk in this setting is a design flaw: the electronic

circuit has been specified and built, and has resulted in an undesired interaction between signals called "signal interference."

In biology, cross-talk is an interaction between two or more signalling pathways in a cell [98]. An example of cross-talk is given in Figure 2.4. Cross-talk can sometimes result in signal interference, such as the oncogenic positive feedback loop formed by cross-talk in [67]. Most often, however, cross-talk is the normal interaction between pathways.

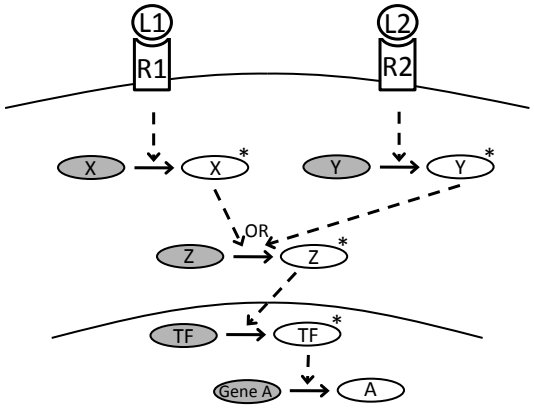


Figure 2.4: An example of cross-talk: both pathways can activate protein Z.

We now give several examples of cross-talk. The Integrin pathway can increase the rate of signal flow through growth factor pathways [98]. There is cross-talk between the signalling pathways in [67]; however, the effect is delayed because the protein involved has first to be expressed from the gene. Growth factor pathways can enhance the expression of estrogen receptors and also activate the receptor in absence of the estrogen ligand [65]. Several pathways can affect the activity of a transcription factor, e.g. the activity of the NF-κB transcription factor is controlled by several pathways [32].

Cross-talk accounts for many of the complex dynamics exhibited by cellular signalling, some of which are as follows.

- **Multi-signal responses:** cross-talk allows certain cellular responses to be decided by several signals, e.g. apoptosis is decided by both pro- and anti-apoptotic signals [63].
- **Signal multi-response:** cross-talk allows certain signals to produce multiple responses by interacting with other pathways, e.g. [17, 101].
- **Signalling history:** cross-talk allows certain cellular responses to be dependent on the

signalling history of the cell, e.g. [64].

- **Protein reuse:** cross-talk allows different pathways to propagate their signal through one or more of the same proteins, e.g. [76].

The process of discovering cross-talk between two pathways involves measuring the cellular response to the two ligands independently and then in combination. There is cross-talk if the cellular response to the combination of ligands is fundamentally unpredictable from the responses to other combinations.

Note that the term cross-talk is sometimes applied to two cells interacting using ligands [38]. This is often considered a misnomer within the biological community—the interaction is between two cells and therefore is more suited to the term (intercellular) communication. On the other hand, communication within the same cell using ligands, called (intracellular) communication [99], can more reasonably be termed cross-talk. Cross-talk can also be applied to pathways other than signalling pathways, such as the interaction between regulatory and metabolic pathways [72]; however, this use is much less common.

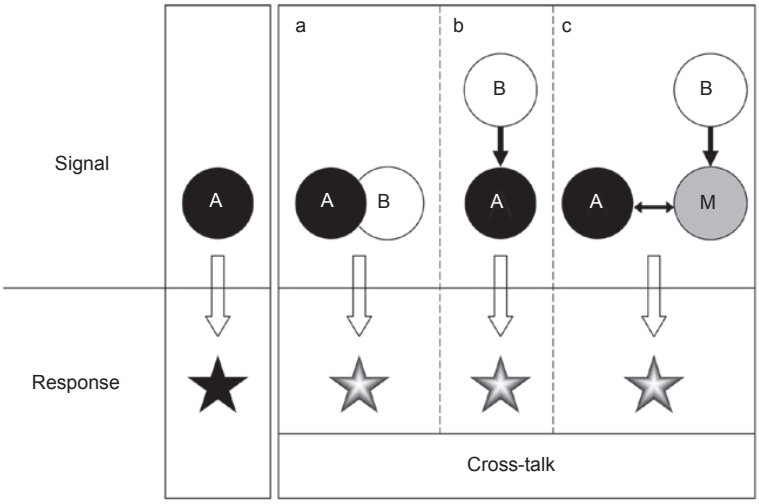


Figure 2.5: Three types of cross-talk. a) components of A and B interact, b) components of A are enzymatic or transcriptional targets of B, and c) pathways compete for the key molecule M. Figure reproduced from [50].

Given the variety of cross-talk scenarios outlined above, it is surprising that there is little discussion of types of cross-talk. To our knowledge there are only two papers that discuss cross-talk types, three types are given in [50] (shown in Figure 2.5) and four types are alluded to in [81], but are not made specific. There is no universal categorisation.

2.6 Summary

In this chapter we introduced the biological background for this thesis. In Section 2.1 we gave basic biological principles and related graphical notation. Section 2.2 gave an overview of the research area of systems biology and Section 2.3 introduced cellular signalling, an important area of study within systems biology. In Section 2.4 and Section 2.5 we explored the concepts of signalling pathways and cross-talk respectively. An important finding was that cross-talk is a rather informal notion and there is a lack of rigorous definitions of what constitutes a single signalling pathway.

In the next chapter we give the fundamental computational concepts we will employ in this thesis.

Chapter 3

Computational background

In this chapter we introduce the fundamental computational concepts used in this thesis.

In Section 3.1 we define continuous time Markov chains and show how they can be be used to model an example biological system. Section 3.2 gives an overview of temporal logic properties and model checking, which we will use to verify properties of Markov chains. In Section 3.3 we define Petri nets and show how they can be used to model the same example biological system. Section 3.4 and Section 3.5 describe the dynamic and steady-state behaviour of Petri nets respectively.

3.1 Continuous time Markov chains

We largely follow the continuous time Markov chain notation of Kwiatkowska et al. [70].

Definition 1 (CTMC)**.** *A continuous time Markov chain (CTMC) is a tuple* (S, s_0, R, L) *where*

- S *is a finite set of states*
- s_0 *is the initial state*
- $R : S \times S \to R_{\geq 0}$ *is the rate matrix, and*
- $L : S \to 2^{AP}$ *is a labelling of states with a finite set of atomic propositions AP that are true.*

There is said to be a *transition* t from s to s', written $s \to_t s'$, if $R(s, s') > 0$. The *rate* of the transition is $R(s, s')$. If there is more than one transition from a state s then there is a *race* between the transitions. The probability that a transition from s to s' completes within t time units is drawn from the probability distribution $1 - e^{-R(s,s') \cdot t}$. S is the set of states that can be reached by zero or more transitions from the initial state.

We can build a CTMC that represents the behaviour of a biological system as follows. The discrete set of states in a CTMC represent the possible configurations of a system, in this case each biochemical species is mapped to a value that represents the level of the species in the state. In the following we use reactions with stoichiometry 1 only. A transition between states represents a reaction that is possible, the difference in the level of each species between the states is governed by the reaction. The continuous time value in the CTMC denotes the time at which each reaction occurs.

In a *CTMC with levels* [25] the amount of each biochemical species is characterised by a concentration range that is discretised uniformly into N levels where $N \geq 1$. Hence, if we choose $N = 3$ then a species X can have four values $X = 0$, $X = 1$, $X = 2$ and $X = 3$. We could also choose $N = 1$, therefore a species is considered *present* ($X = 1$) or *absent* ($X = 0$).

A CTMC with levels is given diagrammatically as follows. Each state is represented by an ellipse, the initial state indicated by an ellipse with a thick line. The amount of each biochemical species in the state is shown inside the ellipse. If $N > 1$ then we explicitly give the amount of each species, e.g. $X = 2$, otherwise we give only the species that are present, e.g. X. Each transition is a directed arc from one state to another, labelled with the rate of the transition.

Example 1 Modelling a biological system as a CTMC with levels.

Consider the biological system depicted in Figure 3.1.

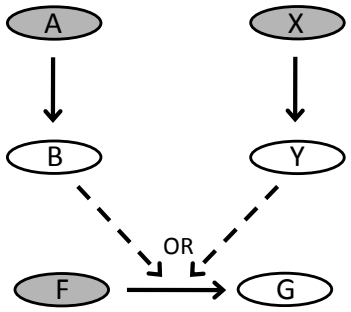


Figure 3.1: A biological system in which protein B or protein Y can enable protein F turning into protein G.

We model the biological system using a CTMC with levels (S, s_0, R, L) with $N = 1$ as follows.

There are 7 states in this model, $S = \{s_0, s_1, s_2, s_3, s_4, s_5, s_6\}$.

We now compute the labelling L of the states in S. Because $N = 1$ we label the states with only the species that are present. The species that are present in the initial state s_0 are indicated by the shaded ellipses in Figure 3.1. The initial state is labelled as follows.

$s_0 \rightarrow \{A, X, F\}$

The labelling of subsequent states is found following transitions from the initial state. From the initial state the transition that turns A into B reaches a new state s_1. Then, from this state the transition that turns X into Y reaches a new state s_4. And so on.

$s_1 \rightarrow \{B, X, F\}$

$s_2 \rightarrow \{A, Y, F\}$

$s_3 \rightarrow \{B, X, G\}$

$s_4 \rightarrow \{B, Y, F\}$

$s_5 \rightarrow \{A, Y, G\}$

$s_6 \rightarrow \{B, Y, G\}$

The rate matrix of the transitions between states is as follows. For simplicity we assume unit reaction rates, hence each reaction has a rate of either 1 (possible) or 0 (impossible).

$$R = \begin{array}{c} \\ s_0 \\ s_1 \\ s_2 \\ s_3 \\ s_4 \\ s_5 \\ s_6 \end{array} \begin{array}{c} \begin{array}{ccccccc} s_0 & s_1 & s_2 & s_3 & s_4 & s_5 & s_6 \end{array} \\ \begin{pmatrix} 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \end{array}$$

The CMTC with levels model of the biological system is shown in Figure 3.2.

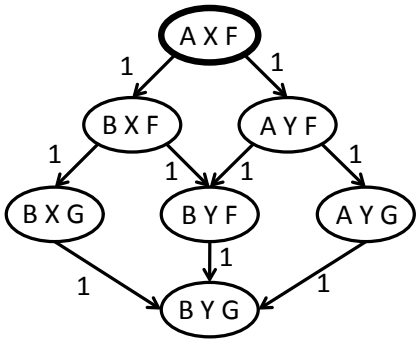


Figure 3.2: The CMTC with levels model of the biological system from Figure 3.1.

The PRISM modelling language [70, 71] is a state-based modelling language based on the

Reactive Modules formalism [1]. The language can be used to define CTMCs with levels. We use the PRISM modelling language in Chapter 5.

3.2 Model checking

A *model checker* is a programme that takes as input a discrete state model and a property and returns whether, or the probability that, the model satisfies the property.

Explicit state model checking [27] checks properties against a model by searching, from the initial state s_0, the states S that can be reached. However, as the number of components in a model grows, the number of states in S can grow exponentially. Explicit state model checking can quickly become infeasible for non-trivial models. Two different approaches are used to overcome this limitation.

- **Partial order reduction:** reduces the size of the state space by considering a subset of the different orders in which concurrent transitions can execute [69].
- **Symbolic model checking:** instead of enumerating the full state space, symbolic model checking considers large numbers of states at a single step by representing sets of states as formulae [9], e.g. $A > 0$ is the set of states in which the value of A is greater than 0.

The background of partial order reduction is covered in more detail in Section 3.4 where we explore state space searching in the context of Petri nets.

3.2.1 Temporal logics

A model checker checks properties expressed in a temporal logic. We refer to safety properties ("bad" properties that are to be avoided) and liveness properties ("good" properties that capture required functionality). We give a brief overview of two temporal logics: Computational Tree Logic and Continuous Stochastic Logic. The latter is a quantitative extension of the former with probabilities and timing.

Computational Tree Logic *Computational Tree Logic* (CTL) [27] is a qualitative logic. Properties expressed in CTL can be used to reason about whether a behaviour is possible, impossible or inevitable.

An *Atomic Proposition* (AP) is a formula in propositional logic that can be evaluated to a boolean value for a state in a Markov chain. An AP may compare combinations of variables in a

Path quantifiers:		
Universal	**A**	all paths from the state
Existential	**E**	at least one path from the state
Temporal operators:		
Next	$X\ \phi$	ϕ holds in the next state
Until	$\phi_1\ U\ \phi_2$	ϕ_1 holds in every state before ϕ_2
Finally	$F\ \phi$	ϕ holds in some future state
Globally	$G\ \phi$	ϕ holds in every state

Table 3.1: The definition of the path quantifiers and temporal operators.

Markov chain and constant values, using equalities and inequalities $=$, $<$, $\leq$, etc. The arithmetic operations $+$, $-$, $*$ and $/$ may be applied to any combination of variables and constant values.

Given variables X, Y and Z, examples of APs are: $(X = 1)$, $(X > 0)$ and $(2 * X > Y + Z)$.

A *path* through a CTMC is a (possibly infinite) sequence of states $s_0, s_1, \ldots$ such that $s_0 \rightarrow_{t_1} s_1 \rightarrow_{t_2} s_2 \rightarrow \ldots$ where $s_{i-1} \rightarrow_{t_i} s_i$ denotes the transition from s_{i-1} to s_i by transition t_i. A path is not necessarily from the initial state of a model.

A CTL formula ϕ is defined as follows:

$\phi ::= AP \mid \neg\, \phi \mid \phi \wedge \phi \mid \phi \vee \phi \mid \mathbf{A}\ X\ \phi \mid \mathbf{E}\ X\ \phi \mid \mathbf{A}\ \phi\ U\ \phi \mid \mathbf{E}\ \phi\ U\ \phi \mid \mathbf{A}\ F\ \phi \mid \mathbf{E}\ F\ \phi \mid \mathbf{A}\ G\ \phi \mid \mathbf{E}\ G\ \phi$

where $\neg$, $\wedge$ and $\vee$ denote "not," "and" and "or" respectively. The definitions of the *path quantifiers* A and E, and *temporal operators* X, U, F and G are given in Table 3.1.

We also use the non-standard *filter construct* $\phi\ \{\ \psi\ \}$ as implemented by PRISM. A filter allows a property ϕ to be checked from a state other than the initial state of the Markov chain, in this case a state that satisfies ψ where $\psi ::= AP \mid \neg\, \psi \mid \psi \wedge \psi \mid \psi \vee \psi$.

Continuous Stochastic Logic *Continuous Stochastic Logic* (CSL) [4] is the quantitative extension of CTL with probabilities and continuous time. Properties expressed in CSL can be used to reason about the time at which events occur and the probability of a behaviour.

In CSL the path quantifiers **A** and **E** are replaced with the probability operator $\mathbf{P}_{\bowtie \mathbf{x}}$ where $x \in [0..1]$ and $\bowtie \in \{>, \geq, <, \leq\}$. The quantifier **A** is equivalent to $\mathbf{P}_{\geq \mathbf{1}}$ and **E** is equivalent to $\mathbf{P}_{>\mathbf{0}}$. The probability of a formula can be returned in the PRISM model checker using $\mathbf{P}_{=?}$.

Temporal operators can have a time bound thus $F_{\leq 10}\ \phi$ expresses ϕ must become true within 10 time units.

Example 2 Temporal logic properties.

The following temporal logic properties are true in the CTMC with levels from Example 1.

- Property: "it is possible that at all times either X or Y are present"

CSL: $\mathbf{P}_{>\mathbf{0}}\,[\,G\,(X\ =\ 1\ \vee\ Y\ =\ 1)\,]$

CTL: $\mathbf{E}\,[\,G\,(X\ =\ 1\ \vee\ Y\ =\ 1)\,]$

- Property: "it is not possible that G is never present"

  CSL: $\mathbf{P}_{\leq\mathbf{0}}\,[\,\neg\,F\,(G\ =\ 1)\,]$

  CTL: $\neg\mathbf{E}\,[\,\neg\,F\,(G\ =\ 1)\,]$

- Property: "it is inevitable that G will become present"

  CSL: $\mathbf{P}_{\geq\mathbf{1}}\,[\,F\,(G\ =\ 1)\,]$

  CTL: $\mathbf{A}\,[\,F\,(G\ =\ 1)\,]$

- Property: "it is possible that G is present within 1.5 time units"

  CSL: $\mathbf{P}_{>\mathbf{0}}\,[\,F_{\leq 1.5}\,(G\ =\ 1)\,]$

  CTL: no CTL equivalent

- Query: "what is the probability that B is present before Y?"

  CSL: $\mathbf{P}_{=?}\,[\,F\,(B\ =\ 1\ \wedge\ Y\ =\ 0)\,]$

  CTL: no CTL equivalent

The PRISM model checker [70, 71] is a popular tool that can check CTL and CSL properties against PRISM models. We employ the PRISM model checker to check properties of CTMCs with levels in Chapter 5.

3.3 Petri nets

We largely follow the Petri net notation of Heiner et al. [55].

Definition 2 (Petri net)**.** *A Petri net, or net for short, is a tuple $\mathcal{N} = (T, P, f, m_0)$. P is a finite set of places and T is a finite set of transitions such that $P \cap T = \emptyset$. f is the set of (nonnegatively) weighted directed arcs between places and transitions, $f : ((P \times T) \cup (T \times P)) \to \mathbb{N}$. m is a state*[1]*, an assignment of places to a number of tokens $m : P \to \mathbb{N}$, where m_0 is the initial state. The number of tokens on a place $p \in P$ in a state m is $m(p)$.*

The set of *pre-* and *post-places* of a transition t is ${}^\bullet t = \{p \in P \mid f(p,t) > 0\}$ and $t^\bullet = \{p \in P \mid f(t,p) > 0\}$ respectively. Likewise the set of *pre-* and *post-transitions* of a place p is ${}^\bullet p = \{t \in T \mid f(t,p) > 0\}$ and $p^\bullet = \{t \in T \mid f(p,t) > 0\}$ respectively.

[1] the Petri net community use the term *marking*, however to be consistent with other chapters, we use the term *state*

We can represent a biological system using a Petri net as follows. Biochemical species are places and reactions are transitions. Reactions change the number of tokens on a set of places (the amount of a set of biochemical species). An *enzyme* in a Petri net is a place that is both a pre- and post-place of a transition.

It is possible to express that multiple tokens are consumed/produced from a place in a transition using an arc with a weight > 1, however in this thesis we only consider arcs with weight 0 or 1.

A Petri net is represented graphically by circles (places), rectangles (transitions), arcs with arrows (directed arcs), and dots or numbers within places (tokens). We use the shortcut of a dashed directed arc from the enzyme to the transition instead of an arc directed in both directions (a bidirectional arc)—we call this an *enzymatic edge* or *enzymatic arc*.

In standard Petri net notation there is no explicit notion of *inhibition*, i.e there are no inhibitory arcs between a place and transition. Inhibitors are modelled by including the exact mechanism that causes inhibition, e.g. the inhibitor and protein bind to form an inactive complex.

We show how the Petri net notation differs from biological notation by reproducing the biological system from Figure 3.1 as a Petri net in Figure 3.3.

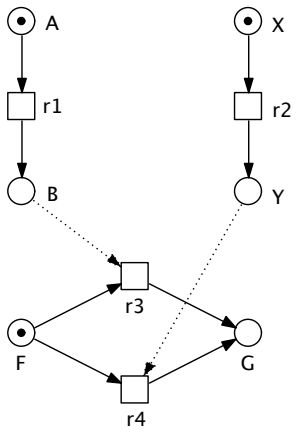


Figure 3.3: A Petri net model of the biological system from Figure 3.1.

Pathway Logic [103] is a framework for modelling biological systems. At the heart of Pathway Logic is a "knowledge base" of biochemical reactions from which Petri net models can be generated. We explore the analysis of Petri net models from Pathway Logic further in Chapter 6.

3.4 Dynamic behaviour of Petri nets

The dynamic behaviour of a Petri net is defined by the firing of transitions in T. A transition t is *enabled* in a state m, written $m \rightarrow_t$, if $\forall p \in {}^\bullet t \,.\, m(p) \geq f(p,t)$. If a transition t is *not* enabled then it is *disabled*, written $m \nrightarrow_t$. A transition t that is enabled in m *may* fire to produce a new state m', written $m \rightarrow_t m'$ where $\forall p \in P \,.\, m'(p) = m(p) - f(p,t) + f(t,p)$.

Definition 3 (Execution)**.** *An execution from m reaching m', written $R \vdash m \rightarrow m'$, is a sequence of transitions $R = t_1, \ \ldots, \ t_n$ where $m \rightarrow_{t_1} m_1 \ \ldots \ \rightarrow_{t_n} m'$. We also use the shorthand notation $m \rightarrow_{t_1, \ \ldots, \ t_n} m'$ for the execution of a sequence of transitions.*

Definition 4 (Reachable state)**.** *A reachable state in a Petri net is the initial state m_0 or a state m' that is reachable by an execution from the initial state, $R \vdash m_0 \rightarrow m'$.*

Note, the following definition uses multisets which are defined in Appendix A.

Definition 5 (Path)**.** *A path $\overline{R}$ from m reaching m', written $\overline{R} \vdash m \rightsquigarrow m'$, is a multiset representation of an execution R such that $R \vdash m \rightarrow m'$.*

To reiterate, an execution R is a sequence of transitions whereas a path $\overline{R}$ is a multiset of transitions. This distinction will be important later.

From some state m, an execution R of $\overline{R}$ is an ordering of the transitions in $\overline{R}$ such that $R \vdash m \rightarrow m'$.

The analysis of the dynamic behaviour of a Petri net is concerned with computing and studying (at least some of) the executions of a system. The dynamic behaviour of cellular signalling models is important because biologists are interested in the transient changes within the cell that lead to the response.

We give an example of the dynamic behaviour of a Petri net in Figure 3.4.

An important concept concerning the dynamic behaviour of a Petri net is the state space.

Definition 6 (State space)**.** *The state space of a Petri net (T, P, f, m_0) is the set of reachable states, i.e. the states that can be reached by any execution from m_0.*

A Petri net is k-bounded (has a finite set of reachable states) if there is some $k \geq 0$ such that no place in the net can have more than k tokens. If the Petri net is k-bounded then the state space is finite and can be given diagrammatically.

The state space is given diagrammatically in a similar manner to a CTMC. Each reachable state is represented by an ellipse, the initial state is indicated by an ellipse with a thick line. The

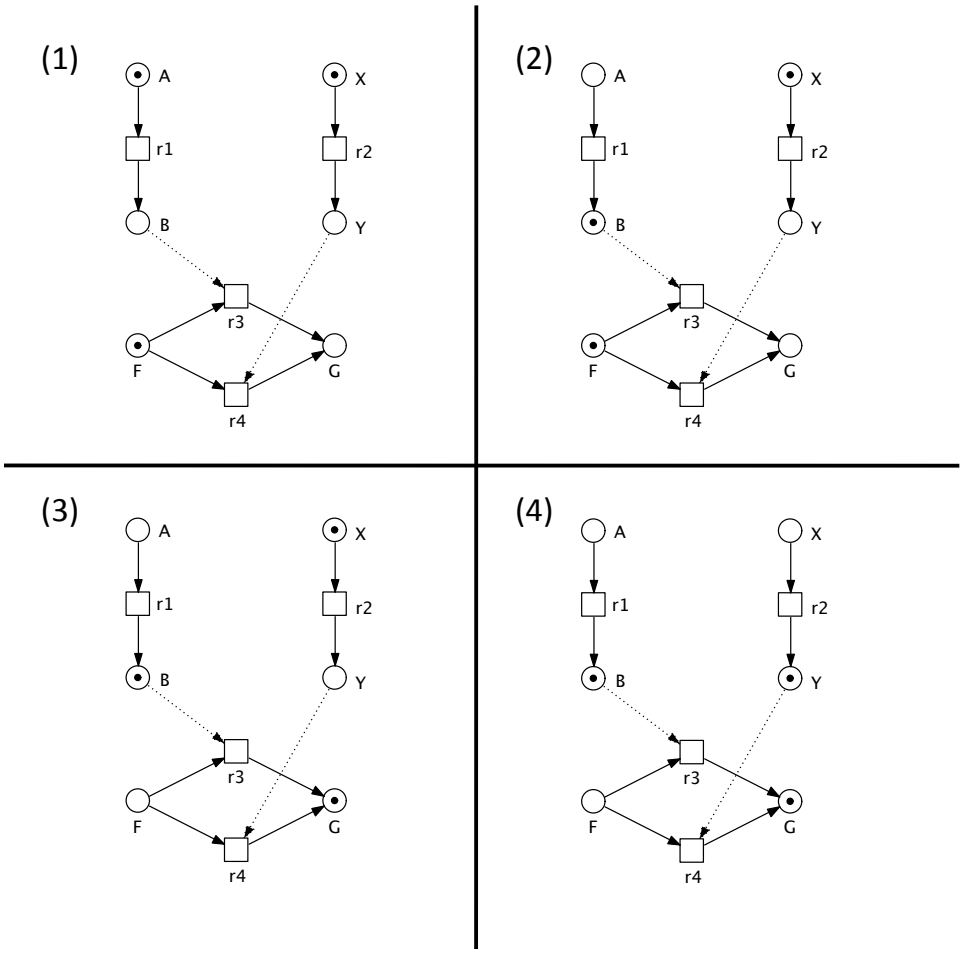


Figure 3.4: An execution of the Petri net from Figure 3.1. This execution is the sequence $R = r1, r3, r2$ and the path $\overline{R} = \{r1, r2, r3\}$.

number of tokens on each place is shown inside the ellipse. If the net is k-bounded where $k = 1$ then we give only the places that have a token, e.g. X, otherwise we explicitly give the number of tokens on each place, e.g. $X = 2$. There is an arc labelled t between a pair of reachable states (m_1, m_2) if $m_1 \rightarrow_t m_2$.

The state space can be searched using breadth-first search (BFS) or depth-first search (DFS). An overview of the difference between BFS and DFS is given in Appendix B. In this thesis we are interested in BFS for reasons that will become clear in Chapter 6.

Definition 7 (State space search)**.** *The state space of a Petri net $\mathcal{N} = (T, P, f, m_0)$ can be searched with BFS or DFS using either algorithm below. This is called the full state space search because all states are searched.*

Breadth-first search

The set of seen states $S = \emptyset$
Add initial state m_0 to the queue Q
while Q is not empty **do**
  Remove state m from the front of Q
  Fire all enabled transitions in state m to produce states $M = \{m_1, \ldots, m_n\}$
  Add $M \setminus S$ to the back of Q
  Add M to S
end while

Depth-first search

Add initial state m_0 to the stack S
while S is not empty **do**
  Remove state m from the top of S
  Fire all enabled transitions in state m to produce states $M = \{m_1, \ldots, m_n\}$
  Add $M \setminus S$ to the top of S
end while

Example 3 Example of a (BFS) state space search.

We show how the state space of the Petri net in Figure 3.3 is searched with BFS from the initial state AXF (i.e. places A, X and F are marked).

The algorithm searches the state space as follows.

Queue: AXF

Seen states: {AXF}

AXF - Transitions r1 and r2 can be fired producing two states, BXF and AYF respectively.

Queue: BXF, AYF

Seen states: {AXF, BXF, AYF}

BXF - Transitions r2 and r3 can be fired producing two states, BYF and BXG respectively.

Queue: AYF, BXG, BYF

Seen states: {AXF, BXF, AYF, BXG, BYF}

AYF - Transitions r1 and r4 can be fired producing two states, BYF and AYG respectively. State BYF has already been seen, so it is not added to the queue.

Queue: BXG, BYF, AYG

Seen states: {AXF, BXF, AYF, BXG, BYF, AYG}

BXG - Transition r2 can be fired producing state BYG.

Queue: BYF, AYG, BYG

Seen states: {AXF, BXF, AYF, BXG, BYF, AYG, BYG}

BYF - Transitions r3 and r4 can be fired producing the same state BYG. State BYG has already been seen, so it is not added to the queue.

Queue: AYG, BYG

Seen states: {AXF, BXF, AYF, BXG, BYF, AYG, BYG}

AYG - Transition r1 can be fired producing the state BYG. State BYG has already been seen, so it is not added to the queue.

Queue: BYG

Seen states: {AXF, BXF, AYF, BXG, BYF, AYG, BYG}

BYG - No enabled transitions and the queue is empty, therefore the search terminates.

Queue: *empty*

Seen states: {AXF, BXF, AYF, BXG, BYF, AYG, BYG}

The state space that was searched is given in Figure 3.5.

As the number of components in the Petri net grows, the size of the state space can grow exponentially. To combat this growth, we can employ a reduced state space search using a suitable partial order reduction algorithm. *Partial order reduction* [69] removes many of the states that are produced when firing different orders of a set of concurrent transitions. In other words, a set of transitions may be fired in many orders but to answer some questions, only a subset of these orders

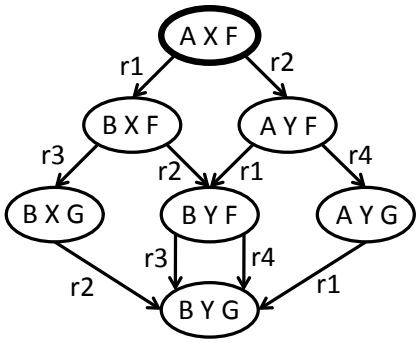


Figure 3.5: The searched state space of the Petri net in Figure 3.3.

need to be explored. Different partial order reduction algorithms guarantee different properties of the reduced state space search, from preserving deadlock states (states with no enabled transitions) to verifying a CTL∗-X (CTL∗ without the Next operator) property.

Definition 8 (Reduced state space search)**.** *The reduced state space of a Petri net* $\mathcal{M} = (T, P, f, m_0)$ *can be searched with BFS or DFS by firing a subset of the enabled transitions at each state. The subset is chosen using a suitable partial order reduction algorithm.*

There are three classes of partial order reduction algorithms: stubborn sets [69], persistent sets [46] and ample sets [87]. We explore reduced state space search algorithms in Chapter 7.

3.5 Steady-state behaviour of Petri nets

A Petri net exhibits steady-state behaviour when an equilibrium is reached despite ongoing processes that attempt to change the system. A simple example to illustrate steady-state behaviour is that of a bathtub with an inflow of water from the tap and an outflow of water from the drain. The bathtub is initially empty, but given suitable rates of inflow and outflow, the volume of water in the bathtub after some time will be constant—a steady-state will be reached.

Place and transition invariants formalise the steady-state behaviour of Petri nets.

Definition 9 (Incidence matrix)**.** *The incidence matrix (also called the stoichiometric matrix) of a Petri net* (T, P, f, m_0) *is a matrix* $C : P \times T \to \mathbb{Z}$*, indexed by* P *and* T*, such that* $C(p, t) = f(t, p) - f(p, t)$*. Hence,* $C(p, t)$ *is the change in the number of tokens on* p *by firing* t*.*

Definition 10 (P invariant)**.** *A P invariant is a place vector* $x : P \to \mathbb{Z}$ *such that* x *is a nontrivial nonnegative integer solution of* $x \cdot C = 0$*. The weighted sum of the tokens on the places in a P invariant is constant while firing any sequence of transitions (i.e. the places are mass conserving).*

Definition 11 (T invariant)**.** *A T invariant is a transition vector $y : T \to \mathbb{Z}$ such that y is a nontrivial nonnegative integer solution of $C \cdot y = 0$. Given some state (not necessarily reachable), the sequential firing of the transitions in a T invariant reproduces the state (i.e. cyclic behaviour). For a T invariant y, there is an execution R (an ordering of the transitions in y) such that $R \vdash m \to m$ for some m.*

We treat P and T invariants more conveniently as multisets rather than vectors. For example, the T invariant $t_1 \to 1, t_2 \to 2, t_3 \to 1$ is treated as the multiset $\{t_1, 2 * t_2, t_3\}$.

The *support* of an invariant x is the set of nodes corresponding to the nonzero entries of x, written $supp(x)$. An invariant x is minimal if there is no invariant z such that $supp(z) \subset supp(x)$ and the greatest common divisor of all nonzero entries of x is 1.

The *minimal* P invariants and T invariants in a Petri net are computed by enumerating all minimal nontrivial nonnegative integer solutions of $x \cdot C = 0$ and $C \cdot y = 0$ respectively. In the following text we consider *only* minimal invariants and often omit the word minimal.

A Petri net is guaranteed to be k-bounded if all places belong to at least one minimal P invariant (i.e. all places are mass conserving).

We now give an example of P and T invariants.

Example 4 Example of P and T invariants.

We compute the set of P and T invariants in the Petri net in Figure 3.6. Notice that the Petri net contains no tokens—this is because P and T invariants are independent of the state.

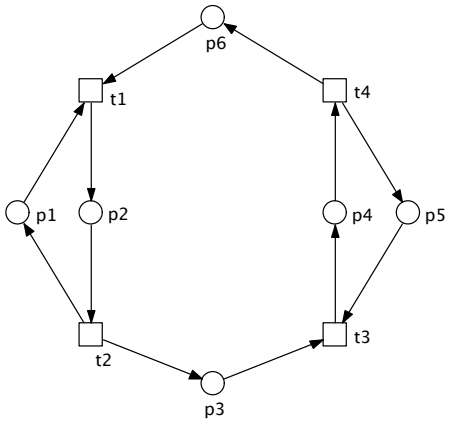


Figure 3.6: A toy Petri net used to illustrate P and T invariants.

P invariants:	T invariants:
$\{p1, p2\}$	$\{t1, t2, t3, t4\}$
$\{p2, p3, p4, p6\}$	
$\{p4, p5\}$	

The P invariants are multisets of places such that the weighted sum of the tokens on the places is always the same regardless of which transitions are fired, e.g. the sum of the tokens on $p1$ and $p2$ is always the same. The T invariant is a multiset of transitions such that firing the transitions in some order reaches the same state, e.g. from any state, firing transitions $t1$, $t2$, $t3$ and $t4$ reaches the same state.

Flux Balance Analysis (FBA) [83] is a form of steady-state analysis that is commonly applied to metabolic systems.[2] Given an $m \times r$ incidence matrix C of a Petri net, FBA is concerned with analysing solutions V to $C \cdot V = 0$ where V is a $r \times 1$ vector of real-value flux levels. Solutions V are constrained such that for at least one element v_i in V, $v_i \geq 0$. $v_i \geq 0$ says that transition (reaction) i can consume substrates and produce products, but cannot produce substrates and consume products.

There is a constant flow in metabolic systems. Any metabolite that is not produced from other metabolites has a metabolic uptake reaction (a reaction that can produce an unlimited number of tokens for that metabolite). Similarly, any metabolite that does not turn into another metabolite has a metabolic secretion reaction (a reaction that can consume an unlimited number of tokens).

FBA requires two transformations to the metabolic network model in order to produce correct results.

1. Transitions arranged in a cycle in a metabolic network cause problems for FBA, leading to unrealisable or non-robust solutions. Two possible solutions would be to either selectively break the cycles (by hand) or to introduce an additional objective function in the FBA, i.e. to minimise the total flux through the system given that the primary objective is optimised already. In the latter case, all futile cycles would be guaranteed to carry zero flux in the optimal flux distribution

2. A reaction that both produces and consumes a metabolite must be replaced by a pair of reactions.

Finite descriptions of the solution space of $C \cdot V = 0$ are important for analysis [85]. An example is the set of elementary modes, $M = \{V_1, ..., V_k\}$, which are the real-value counterpart to

[2] Although FBA is a generic technique, in this chapter we give it in the context of Petri nets.

minimal T invariants (which are integer solutions of the same equation). The set of elementary modes M has the following properties. Any solution V is a positive linear combination of vectors in M. Each $V_i \in M$ is a maximally zero solution (the same minimality condition for minimal T invariants). Finally, every maximally zero solution is in M.

We use T invariant analysis of Petri nets in Chapter 6.

3.6 Summary

In this chapter we introduced the two fundamental computational concepts for this thesis. The first paradigm was continuous time Markov chains in Section 3.1 and in Section 3.2 we discussed model checking of these models. The second paradigm was Petri nets in Section 3.3. In Section 3.4 and Section 3.5 we described the dynamic and steady-state behaviour of Petri nets respectively and introduced the concepts of P and T invariants.

In the next chapter we review how computational techniques have been applied to model and analyse biological systems.

Chapter 4

Related work

In this chapter we review the application of various computational techniques to model and analyse biological systems.

We start in Section 4.1 with an overview of how models of biological systems are created. In Section 4.2 we give two important, broad categories of models, mathematical and computational models, and another important distinction, qualitative vs. quantitative models. Next we explore the analysis of biological models, with dynamic analysis in Section 4.3 and with steady-state analysis in Section 4.4. Finally, we review the application of modelling and analysis techniques to biological cross-talk in Section 4.5 and to non-biological cross-talk in Section 4.6.

4.1 Biological models

The role and importance of computational modelling in systems biology is now well-established. The complexity of biological systems, as well as the volume of experimental data generated to capture this complexity, means that models are crucial. Models are used to generate new insight into biological systems, answering questions such as how they work, why they do not work and what is the response to system perturbations such as drugs.

Examples of how models are created are given in [23] and [84]. Models of biological systems comprise two main parts, the *structure* and the *parameters*. The structure is the set of reactions and the parameters are the values used in the reactions such as initial concentrations and reaction rates.

The structure of a model of a biological system can be inferred by measuring how the concentration of certain biochemical species changes over time. An intuitive toy example is that of three proteins, X, Y and Z. If we observe that the concentration of protein X is decreasing and

protein Y is increasing over time, we could infer a reaction where protein X turns in to protein Y. If the removal (or inhibition) of protein Z stops this observation, we could infer that protein Z is an enzyme for this reaction.

The parameters of a model of a biological system are usually very difficult to obtain. Initial concentration values can be found by measurements of the amount of protein in a sample of cells, e.g. using western blotting [84]. However, accurate methods to measure the rate of reactions are rare. Often reaction rates are guesses from a range of "reasonable" values and known relationships between reactions, e.g. "reaction X happens 10 times faster than reaction Y."

4.2 Types of biological models

We follow Fisher and Henzinger [40] and distinguish two types of biological models, mathematical and computational models. Also important to the work in this thesis is the distinction between qualitative and quantitative models.

4.2.1 Mathematical models

Mathematical models are the most common approach to modelling biological systems. Mathematical models are *solved* to produce the behaviour of the system.

Ordinary differential equation (ODE) models [43,84] are the most popular modelling technique for biological systems. In an ODE model, each species is represented by a single differential equation. The system of ODEs are solved, producing the concentration of each biochemical species in the model at various time points. The computational complexity is lower compared with many other techniques, however the output is only the average behaviour of the system.

Models such as ODE models are called *deterministic models* because the output of the model is the same upon successive runs. Models that take stochastic effects (e.g. from low molecule numbers and gene expression) into account are called *stochastic models*—there is a range of outputs of these models, depending on the stochastic effects.

The *Chemical Langevin Equation (CLE)* is a set of stochastic differential equations, one for each biochemical species in the system [58]. The solution of the CLE gives a real value concentration for each species at time t.

The *Chemical Master Equation (CME)* is a set of ODEs, one for each possible state of the system. The solution of the CME gives the probability distribution of the states at a particular time t. Solving the CME is usually too computationally intensive. The Stochastic Simulation

Algorithm [45] (also called Gillespie's algorithm) is often used to produce single simulations from the CME.

Markov processes are models that hold the Markov property that the probability distribution of a future state depends only on the current state. Markov processes include Markov decision processes, discrete time Markov chains and, of particular importance to this thesis, *continuous time Markov chains (CTMCs)*. CTMCs have been used to model biological systems in [25], amongst others.

Finally, Boolean networks have been used to model biological systems, especially gene regulatory networks [3].

As pointed out by [40], mathematical models are restricted to mathematical analysis e.g. simulation [23], sensitivity analysis of the parameters [77] and steady-state behaviour analysis [84].

The main problem with mathematical models are that they do not have well-defined notions of structure, modularity and composition. There is also no *formal* graphical representation of these models and no automatic way to go from equations to diagrammatic representations.

4.2.2 Computational models

Computational models have origins in the design and verification of hardware and software systems where correctness is especially important, for example embedded systems or aeronautic software. Computational models use a *modelling language* to describe the system and are *executed* to mimic the system.

Computational models of biological system have become increasingly popular in recent years. We now explore the application of computational modelling paradigms to biological systems.

Process algebras were first used to model biological systems in [93] using π-calculus [79]. Each molecule in the system is modelled as a process and reactions are modelled as communication between processes. Later, beta binders [91] were used to model biological systems with an enriched syntax.

Performance Evaluation Process Algebra (PEPA) [59] is a process algebra that has enjoyed considerable success in modelling biological systems. For example, PEPA is used to model the RKIP influence on the ERK signalling pathway in [10]. Later, a process algebra called Bio-PEPA [25] was created which includes higher level constructs specifically for modelling biological systems. Applications of Bio-PEPA include modelling the cAMP/PKA/MAPK pathway [26], the Gp130/JAK/STAT pathway [48] and the NF-κB pathway [24].

While the PRISM modelling language (discussed in the previous chapter) is not strictly a

process algebra, it shares some of the same features such as modularity, composition and the use of process algebraic operators for communication between subsystems. The PRISM modelling language with underlying semantics of a CTMC has been used to model the RKIP inhibited ERK pathway in both [11] and [15].

Petri nets have been used to model a number of biological systems in [43, 44, 55–57].

Finally, rule-based models have been used to model biological systems in [37, 60].

Of particular importance to this thesis is computational modelling approaches that have well-defined notions of structure, modularity and composition. In Chapter 5 we introduce a modular modelling approach for signalling pathway cross-talk using the PRISM modelling language.

4.2.3 Qualitative vs. quantitative models

Another important aspect of biological models is the distinction between qualitative and quantitative models.

Qualitative models focus only on the structure of a biological system, i.e. the reactions in the system. The rationale for this approach is often that the parameter values are difficult to obtain or estimate. Qualitative models can optionally include an initial state to allow simulation, state space analysis and model checking. Examples of qualitative models are Boolean networks [3, 51]. The Toll-like receptor map [82] (essentially a comprehensive, formal diagram) is also a qualitative model, but it has no initial state, so analyses are limited.

Quantitative models extend qualitative models with detailed parameter values permitting rich behavioural analysis and comparison to laboratory data, e.g. [23].

4.3 Dynamic analysis

We review two types of dynamic analysis: simulation-based analysis and model checking.

4.3.1 Simulation-based analysis

Simulation-based analysis involves analysing the output of a model in the form of simulation runs, e.g. a time-series of protein concentration values. Applications include comparing the output of a model with laboratory data to drive the model construction process [23] and the analysis of the transient behaviour of proteins [8].

4.3.2 Model checking

Model checking is also considered to be dynamic analysis. In [44] model checking is used to analyse qualitative, continuous and stochastic Petri nets models of the MAPK pathway. Similarly, in [16] model checking is use to analyse models of the cell cycle with boolean, concentration and population semantics. Model checking of PEPA and PRISM models of the RKIP inhibited ERK pathway using CSL is the topic of [11]. Finally, recent work has used temporal logic to describe the desired behaviour of a biochemical system in the parameter estimation of biological models [35, 95].

4.4 Steady-state approaches

Steady-state approaches have traditionally been used to analyse models of metabolic systems. There is constant flow through metabolic systems because there is constant uptake and excretion of metabolites, whereas cellular signalling systems respond to transient incoming signals dynamically and should not operate in a steady state. We review the application of steady-state techniques to metabolic systems and recent applications to cellular signalling systems.

4.4.1 Application to metabolic systems

Metabolic systems are usually analysed using steady-state analysis because there is constant consumption and production of metabolites. FBA and elementary modes have been used for analysis of metabolic systems including [85, 97]:

- Prediction of minimal sets of nutrients required for a cell to produce a metabolite of interest.
- Finding unused reactions that can point to gaps in the model.
- Finding correlated reactions, i.e. reactions that are always “on” or “off” together.

The application of steady-state techniques to metabolic systems is sound, due to the constant flow in such systems.

4.4.2 Application to cellular signalling systems

Steady-state analysis of cellular signalling systems is much less common because signalling initiates transient changes within the cell. Despite this, steady-state analysis has been applied to cellular signalling systems resulting in useful analysis results.

FBA has been used to modularise the Toll-like receptor signalling network into "distinct input/output signalling (DIOS) pathways" [74]—DIOS pathways are the same as signal flows[1]. The network was decomposed into 10 DIOS pathways and resulted in the identification of novel inhibition targets.

T invariants were used to analyse the Pheromone response pathway in yeast [47]. Clustering of T invariants revealed functional modules that allowed better understanding of the pathway. Finally, T invariant analysis of the apoptosis network in [57] discovered that some T invariants describe basic signal flows and some describe cross-talk between signal flows.

We discuss in Chapter 6 how the steady-state analysis of cellular signalling models can produce incorrect results and introduce a sound alternative based on dynamic analysis.

Note that the PRISM model checker can be used for "steady-state analysis." For example, in [15] the probability distribution of the concentration of a protein in the "steady-state" is calculated using the PRISM model checker. Rather than analysing the incidence (stoichiometric) matrix, the stationary distribution of the Markov chain is computed. This analysis can be more accurately described as analysis of the model's behaviour *in the long run*.

4.5 Modelling and analysis of signalling pathway cross-talk

There are a limited number of applications of computational techniques to study signalling pathway cross-talk.

There are several mathematical models of specific pathways that include an aspect of cross-talk. Ordinary differential equation models have been used to model the cross-talk between the MAPK and AKT pathways [53], the MAPK and PKC pathways [101], and the hyperosmolar and the pheromone MAPK pathways [76].

The analysis depends on the pathways involved. In [76] the motivation for modelling is to answer how the pathways maintain signal specificity, given shared common proteins. Two models are proposed, one that contains mutual inhibition between pathways to limit signal bleed-through and one that contains scaffold proteins. In [101] the goal is to investigate whether cross-talk has an effect on bistability, namely whether the signal switches from transient to sustained activation as a consequence of varying the duration of the signal. Cross-talk is expressed implicitly in mathematical models, i.e. it is part of the system of equations, with no explicit reference to pathways or interactions between pathways. Therefore, there is no direct way to reason about cross-talk,

[1] recall that signal flow was informally defined on page 13

especially to detect or classify the cross-talk.

Computational models are also employed. For example, Petri nets are used to model apoptosis decision-making in the Fas-induced and mitochondrial DNA damage pathways, and this includes the Bid controlled cross-talk between them [57]. In [41] there is a discrete, state-based model of the multiple modes of intercellular "cross-talk" between the EGFR and LIN-12/Notch signalling pathways, developed in the language of Reactive Modules. Model checking is used to check the validity of the model and to generate new biological insights. But, intercellular "cross-talk" is considered a misnomer within parts of the life science community—it is often called cellular communication. This work bears little relation to the *intracellular cross-talk* that is of interest in this thesis.

In general, it is difficult to draw any generic methods or techniques from these specific models.

We are aware of only one paper [30] that addresses a more generic concept of pathway and cross-talk. Models in [30] are defined using the rewrite rules of the κ calculus; the notion of a "story" corresponds to a pathway and an "influence map" defines how rules can inhibit each other. Superposition of an influence map with a pathway suggests ways in which a story's ending can be delayed or prevented (i.e. delay or prevent pathway output)—this can be interpreted as detecting cross-talk. The pathways of [30] are minimal executions to an output (equivalent to signal flows), and thus cannot be compared directly with the established signalling pathways that we consider. Nonetheless, we note that superposition (via renaming and synchronisation) is also fundamental to the work in Chapter 5.

4.6 Cross-talk in non-biological systems

The analysis of cross-talk between signalling pathways bears similarities to cross-talk in other systems (although it is often not called cross-talk in these contexts).

We have already learned that cross-talk was first applied to describe signal interference between electronic circuits [18]. Software systems that are developed independently and communicate on a shared medium may exhibit unusual behaviour that needs to be both detected and resolved, e.g. distributed systems, web services or e-mail [52]. The recent increase of mobile systems and multimedia services that operate on the internet further this problem [6], exacerbated by the commercial sensitivity that often inhibits openness about system/service architecture [14].

Related fields have a common theme: they comprise distinct elements of a system that are developed independently, then composed.

Feature interaction in telecommunication systems A well-studied area of research is feature interaction in telecommunication systems [13]. The feature interaction problem can be described as the following three components.

Basic service: the basic functionality of the telecommunication system, e.g. place/answer calls, busy in call.

Feature: a component of functionality additional to the basic service that users subscribe to, typically added incrementally, e.g. Call Waiting.

Feature interaction: two or more features added to a system causing a behavioural modification of the feature or basic service, possibly harmful to the user experience.

Consider a telecommunication system with two features, Answer Call (AC) and Call Waiting (CW). CW sounds a call waiting tone to the user being called upon receiving a second call. AC diverts the user that is calling to an answering service upon the called user being busy. If user A subscribes to both AC and CW, then the behaviour of the system when user A is in a call to user B and receives a second call from user C is undetermined.

We suggest that feature interaction is similar to cross-talk between signalling pathways, though there are some important differences that we highlight below.

	Features in telecom. systems	**Signalling pathways**
Development	Developed independently	Evolved in a tightly coupled manner
Specification	Engineered from specification	Hypothesised specification
Interference	Bad user experience	Disease
Goal	Detect & resolve interference	Explore potential interactions

There are three broad categories of analysis approaches; software engineering (avoiding interference) [42], formal methods (analysing models of a system) [13], and online analysis (analysing real systems) [94]. The importance of formal methods in this field provides support for our approach to analysing cross-talk in Chapter 5.

4.7 Summary

We have reviewed the application of various computational techniques to model and analyse biological systems. In Section 4.1 we gave an overview of how models are built. In Section 4.2 we reviewed two types of models, mathematical and computational models, and another important distinction, qualitative vs. quantitative models. We found that mathematical models do not

have well-defined notions of structure, modularity and composition. We also found that qualitative models are sometimes used in place of quantitative models because the parameter values, especially reaction rates, are hard to obtain. Next we covered the analysis of biological models, with dynamic analysis in Section 4.3 and steady-state analysis in Section 4.4. Steady-state approaches are well-suited to metabolic systems, but not to cellular signalling—because of transient flow—though some successful applications exist. We reviewed work on modelling and analysis of cross-talk in Section 4.5 and found that there is a focus on mathematical models, which have no explicit notion of cross-talk. Finally, in Section 4.6 we looked at modelling and analysis of cross-talk in non-biological systems, especially telecommunication systems. We found that the use of formal methods has been particularly successful in this area.

In the next chapter we tackle the problem of modelling signalling pathway cross-talk in a rigorous, modular fashion.

Chapter 5

Modelling signalling pathway cross-talk

In this chapter we introduce a formal framework for pathway and network modelling that allows us to explain, categorise, and detect cross-talk in a systematic way.

We start by outlining the basic modelling framework for signalling pathways and networks. The framework is based on composing generic pathway modules written in the PRISM modelling language. In Section 5.1 we give the motivation for the framework. In Section 5.2 we explain how the PRISM modelling language can be used to model CTMCs with levels. In Section 5.3 we extend the PRISM modelling language with the abstractions required to model generic pathway modules. In Section 5.4 we introduce our modular modelling approach and show how we can build a pathway by composing generic pathway modules. In Section 5.5 we show how, in a similar way, we can compose pathways to build a signalling network, in this case the pathways are composed independently.

Next, we show how the modelling framework can be used to model cross-talk in a formal and rigorous way. Cross-talk is expressed by different synchronisations of reactions between, and overlaps of, pathways written in the PRISM modelling language—using this approach, we can formally define, and reason about, cross-talk. In Section 5.6 we discuss auxiliary reactions which are added to a pathway to allow additional, optional pathway behaviours. In Section 5.7 we give the main contributions of this chapter: a categorisation and formalisation of cross-talk and a modelling approach for cross-talk, using the auxiliary reactions from Section 5.6. In Section 5.8 we introduce an algorithm to enumerate all instances of cross-talk between two pathways.

We then give preliminary results on how to analyse a model that does not contain a explicit notion of cross-talk: in Section 5.9 and Section 5.10 we show how to detect and characterise cross-talk respectively. In Section 5.11 we demonstrate our framework with a case study of the TGF-β, WNT and MAPK pathways.

Finally, in Section 5.12 we discuss our modelling assumptions and possible extensions of the framework.

Background material We assume the following background material: continuous time Markov chains (Section 3.1) and model checking (Section 3.2).

5.1 Motivation

Signalling pathways[1] are well-known abstractions that explain the mechanisms whereby cells respond to signals. They comprise biochemical reactions that transfer information from a receptor to a target such as the nucleus or mitochondria. Several computational modelling paradigms from computer science have been extended and applied to signalling pathways in recent years, for example, rewrite rules [30], Petri nets [55] and process algebras [10, 15, 19, 33, 105]. However, there has been less focus on collections of pathways that form networks, and very little on the interactions between pathways, known in the life sciences as *cross-talk*. Cross-talk accounts for many useful behaviours, for example, producing a variety of responses to a single signal, and reuse of proteins between pathways. Cross-talk is an essential aspect of network behaviour, yet there are no known computational models of pathways with cross-talk.

5.2 The PRISM modelling language

The PRISM modelling language [70,71] is a state-based modelling language based on the Reactive Modules formalism [1]. We focus on using the language to build continuous time Markov chains (CTMCs).

We adopt a reagent-centric approach [12] to modelling in which each of the reagents in a reaction is mapped to a process, whose variation reflects increase or decrease in amount of the reagent, through production or consumption. The processes can model individuals (molecules) or populations (concentrations of biochemical species); we assume the latter here, and an underlying semantics of CTMCs with levels [25].

A CTMC with levels model is defined using the PRISM modelling language as follows. A PRISM model contains one or more *modules*.

[1] we refer simply to pathways henceforth

States A model can contain *global variables* and each module can contain *local variables*. Each *state* in a CTMC is labelled with an assignment of values to the variables—the initial state is labelled with an assignment of the initial values to the variables. The definition of an integer variable has the following syntax.

```
name : [min_val .. max_val] init val;
```

Transitions Each module can have a number of *commands* that represent the transitions in the CTMC. The definition of a command has the following syntax.

```
[label] guard -> rate:update_statement;
```

A guard is a Boolean expression, often used to check the values of one or more variables in a state. Commands are only executable in a state where their guard is true. If there is more than one executable command in the current state then each executable command executes with a probability that is proportional to the rate. An update statement is an assignment of new values to the variables in the model, hence the update statement $X' = X + 1$ increments the value of X by 1. Executing a command reaches the state as governed by the update statement. The time at which the command executes is drawn from the probability distribution $1 - e^{-rate \cdot t}$ where rate is the rate of the command. Commands can be labelled so that they can be synchronised, renamed or hidden.

Synchronisation, renaming and hiding A PRISM model can contain a *system equation* in which modules can be composed concurrently, *synchronising* (multiway) on the commands whose labels occur in the synchronisation set. For example, given modules $M1$ and $M2$, and set of labels L, $M1$ |[L]| $M2$ denotes the concurrent composition of $M1$ and $M2$, synchronising on all labels in L. If the label set is omitted, $M1$ || $M2$, then $M1$ synchronises with $M2$ on the intersection of labels occurring in $M1$ and $M2$. Conversely, $M1$ ||| $M2$ synchronises $M1$ with $M2$ on no labels (independent). If $M1$ and $M2$ synchronise on a label l then the command l in $M1$ and l in $M2$ execute at the same time and only after both guards become true. The rate at which synchronised commands execute is the product of the rates of the individual commands. Labels can be *renamed*, denoted thus $M1$ $\{old\_label \leftarrow new\_label\}$, and *hidden*, denoted thus $M \setminus \{label1, \ldots, labeln\}$. Hidden labels are not available for synchronisation. A system equation is given within the *system* ... *endsystem* construct. If a system equation is not given then all modules are composed using the || operator.

Example 5 Comparison of two PRISM models.

Two PRISM models of a reaction called $r1$, the complexation of X and Y forming Z, are given

below. The model on the left has a single module with a command that represents reaction $r1$. The model on the right has three modules that are composed in parallel, causing the three commands to be synchronised to create reaction $r1$. Both models create the same underlying CTMC with levels.

Model 1

```
module example
  X : [0..1] init 1;
  Y : [0..1] init 1;
  Z : [0..1] init 0;

  [r1] X = 1 & Y = 1 & Z = 0 ->
      1:(X' = 0) & (Y' = 0) & (Z' = 1);
endmodule
```

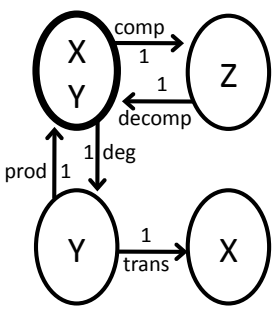
Model 2

```
module module1
  X : [0..1] init 1;

  [r1] X = 1 -> 1:(X' = 0);
endmodule

module module2
  Y : [0..1] init 1;

  [r1] Y = 1 -> 1:(Y' = 0);
endmodule

module module3
  Z : [0..1] init 0;

  [r1] Z = 0 -> 1:(Z' = 1);
endmodule

system
      module1 || module2 || module3
endsystem
```

Example 6 A PRISM model containing each type of biochemical reaction.

A PRISM model that contains each type of biochemical reaction (production, degradation, transformation, complexation and decomplexation) and the related CTMC with levels is given below. The arcs in the CTMC diagram are labelled with both the rate and label of the command that causes the transition.

```
module example
  X : [0..1] init 1;
  Y : [0..1] init 1;
  Z : [0..1] init 0;

  [prod] X = 0 -> 1:(X' = 1);
  [deg] X = 1 -> 1:(X' = 0);
  [trans] X = 0 & Y = 1 ->
      1:(X' = 1) & (Y' = 0);
  [comp] X = 1 & Y = 1 & Z = 0 ->
      1:(X' = 0) & (Y' = 0) & (Z' = 1);
  [decomp] X = 0 & Y = 0 & Z = 1 ->
      1:(X' = 1) & (Y' = 1) & (Z' = 0);
endmodule
```



5.3 PRISM language extensions

The PRISM modelling language does not include all the abstractions required for the generic pathway modules given in the next section. We therefore introduce two extensions to make the language more convenient for modelling—they do not add any expressive power to the language.

We express pathways by composing generic pathway modules, and networks by composing pathways. Variable sharing between modules/pathways provides a convenient way to express overlapping pathways, e.g. cross-talk in which pathways share the same protein. Synchronisation within modules provides a convenient way to build larger reactions by synchronising smaller reactions in a module, e.g. we could synchronise the reaction for the degradation of protein X with the reaction for the production of protein Y to create the reaction $X \rightarrow Y$.

Variable sharing To implement variable sharing between modules, we cannot use PRISM global variables because they cannot be updated within a labelled transition. So, we use PRISM local variables and introduce new syntax as follows: $M1 \; |[L,\; V]| \; M2$, where $V = \{(v_1, w_1),\; \ldots,\; (v_n, w_n)\}$. (v_i, w_i) is called a *variable sharing* where v_i (local to $M1$) and w_i (local to $M2$) are shared. We implement a variable sharing (v_i, w_i) in PRISM by a pre-processing step in which we substitute w_i for v_i. For each command r in $M1$, we remove all references to v_i from r and define a new command r in $M2$[2], substituting w_i for v_i; we then synchronise $M1$ and $M2$ over r.[3] We assume that the PRISM modules have the same number of levels N for all variables, so the ranges of the shared variables are the same. The initial value of the shared variable is $max(init(v_i), init(w_i))$ where $init(var)$ is the initial value of the variable var. Because we can now share variables between two modules, we extend the hide operator to hide local variables so that they are unavailable for sharing. Hence we can hide labels and variables in a module thus $M \setminus \{label_1,\; \ldots,\; label_n,\; var_1,\; \ldots,\; var_n\}$.

Synchronisation within modules Synchronisation within modules is currently not implemented in PRISM. Suppose we have two labels $r1$ and $r2$ in module $M1$, then renaming one by the other will not force a synchronisation, i.e. $M1\ \{r2 \leftarrow r1\}$ will create two $r1$ labels in $M1$, and a non-deterministic choice between the labels. So, we give an alternative semantics for renaming when the labels are in the same module. With the new semantics, our example $M1\ \{r2 \leftarrow r1\}$ means $r1$ and $r2$ are *synchronised within a module*, which we implement by pre-processing (to form a single transition $r1$ that is the conjunction of the transitions for $r1$ and $r2$).

[2] we have taken care to avoid naming conflicts in our examples; however, in general this may be problematic

[3] Modules that share a variable are still independent; the shared variable is local to one module and access to this variable is only through local labelled commands.

5.4 Modelling a pathway

We define a *generic pathway module* to be a behavioural pattern within a pathway. For example, commonly occurring pathway modules are: Receptor, 3-stage Cascade and Gene Expression (shown in Figure 5.1).

The Receptor module has three species (L for ligand, R for receptor and R* for active receptor) and two reactions (r1 and r2). The 3-stage Cascade module has 3 species (proteins X, Y and Z) and 4 reactions (r3, r4, r5 and r6). The Gene Expression module has 2 species (Gene and Protein) and one reaction (r7). While Gene is not strictly a biochemical species, for modelling purposes we treat it as a species. We use shading to indicate species with initial concentrations (the species present in the initial state).

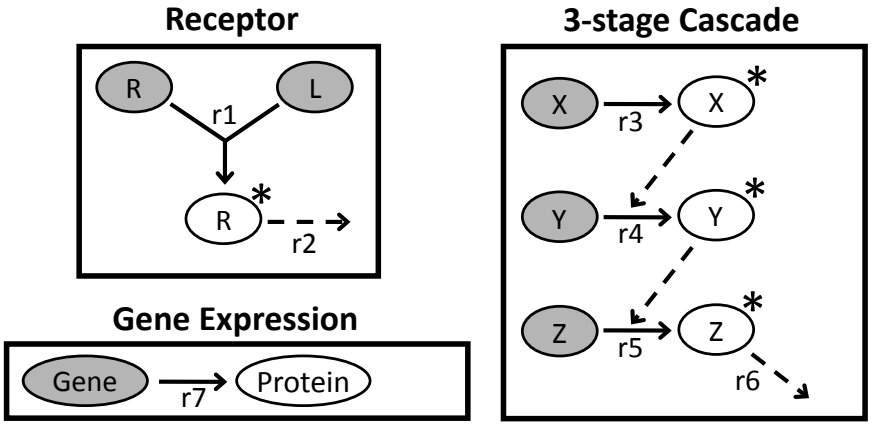


Figure 5.1: Three generic pathway modules: Receptor, 3-stage Cascade and Gene Expression.

Note that r2 and r6 are not reactions in their own right—they do not update the state of the model. r2 and r6 are the enzymatic activities of R^* and Z^* respectively.

We represent the three generic pathway modules as PRISM modules with $N = 1$ below.[4]

```
module Receptor
   R : [0..1] init 1;      L : [0..1] init 1;      R* : [0..1] init 0;

   [r1] R = 1 & L = 1 & R* = 0 -> 1:(R' = 0) & (L' = 0) & (R*' = 1);
   [r2] R* = 1 -> 1:true;
endmodule

module 3StageCascade
   X : [0..1] init 1;   X* : [0..1] init 0;
   Y : [0..1] init 1;   Y* : [0..1] init 0;
   Z : [0..1] init 1;   Z* : [0..1] init 0;
```

[4] PRISM reserves certain names such as X and does not allow names with the $*$ symbol—strictly we use names such as *XInactive* instead of X and *XActive* instead of X^*.

```
   [r3] X = 1 & X* = 0 -> 1:(X' = 0) & (X*' = 1);
   [r4] Y = 1 & Y* = 0 & X* = 1 -> 1:(Y' = 0) & (Y*' = 1);
   [r5] Z = 1 & Z* = 0 & Y* = 1 -> 1:(Z' = 0) & (Z*' = 1);
   [r6] Z* = 1 -> 1:true;
endmodule

module GeneExpression
   Gene : [0..1] init 1;      Protein : [0..1] init 0;

   [r7] Gene = 1 & Protein = 0 -> 1:(Gene' = 0) & (Protein' = 1);
endmodule
```

We treat these modules as *generic*, that is, we instantiate them (strictly, duplicate and rename in PRISM) for multiple occurrences. We adopt the following convention. For generic module M, M_i denotes an *instance* of M with every variable and reaction renamed by an indexed form. For example, variable v becomes v_1 in module M_1.

We can compose modules synchronising over sets of labels as follows. Synchronising reaction a in module A with b in module B is achieved by renaming a to b and synchronising the modules over b, i.e. $A\ \{a \leftarrow b\}\ |[b]|\ B$. In this chapter we use the term label and reaction synonymously.

A pathway is a parallel composition of instances of generic modules, renaming reactions to coordinate synchronisation within the pathway.

Definition 12 (Pathway)**.** *Let G be a set of generic pathway modules. A pathway P has the form $(X_1 f_1\ |[L_1]|\ \ldots\ |[L_{n-1}]|\ X_n f_n)\ \setminus\ H$ where $X_1\ \ldots\ X_n$ are instances of modules in G, $f_1\ \ldots\ f_n$ are sets of renamings, $L_1\ \ldots\ L_{n-1}$ are labels (reactions) and H is a set of hidings.*

Definition 13 (Renaming pathway reactions)**.** *The reactions in a pathway P can be renamed creating a new pathway $P' = P\ \{renamings\}$ where renamings is a set of renamings.*

As an example, consider pathway $Pathway_1$ comprising instances of the Receptor, 3-stage Cascade and Gene Expression modules:

$$
\begin{aligned}
Pathway_1 \;=\; & (\text{Receptor}_1\ \{r2_1 \leftarrow r3_1\} \\
& |[r3_1]| \\
& \text{3StageCascade}_1\ \{r6_1 \leftarrow r7_1\} \\
& |[r7_1]| \\
& \text{GeneExpression}_1) \\
& \setminus \{r1_1, r3_1,\ r4_1, r5_1, R_1, L_1, R_1^*, Gene_1, Protein_1\}
\end{aligned}
$$

Receptor_1 and 3StageCascade_1 modules synchronise on $r2_1$ and $r3_1$, and 3StageCascade_1 and GeneExpression_1 synchronise on $r6_1$ and $r7_1$ (strictly, we rename $r2_1$ to $r3_1$ and synchronise the modules on $r3_1$, and similarly for $r6_1$ and $r7_1$). Because of these synchronisations, the active

receptor catalyses the activation of protein X and active protein Z catalyses the expression of Gene. Reactions $r1_1$, $r3_1$, $r4_1$ and $r5_1$ and variables R_1, L_1, R_1^*, $Gene_1$ and $Protein_1$ are hidden using the \ operator.

Reactions and (local) variables are considered to be external or internal. Reactions that are not external are internal.

Definition 14 (External reactions and variables)**.** *For a pathway P, the set of external reactions, $ext_r(P)$, is the set of reactions, modulo renamings, that have not been hidden and the set of external variables, $ext_v(P)$, is the set of (local) variables that have not been hidden. External reactions are available for synchronisation and external variables are available for sharing.*

Hence, $ext_r(Pathway_1) = \{r7_1\}$ and $ext_v(Pathway_1) = \{X_1, Y_1, Z_1, X_1^*, Y_1^*, Z_1^*\}$.

As a further example we define pathway $Pathway_2$:

$$
\begin{aligned}
Pathway_2 \;=\; & (\text{Receptor}_2\ \{r2_2 \leftarrow r3_2\} \\
& |[r3_2]| \\
& \text{3StageCascade}_2\ \{r6_2 \leftarrow r7_2\} \\
& |[r7_2]| \\
& \text{GeneExpression}_2) \\
& \setminus \{r1_2, r3_2,\ r4_2, r5_2, R_2, L_2, R_2^*, Gene_2, Protein_2\}
\end{aligned}
$$

With $ext_r(Pathway_2) = \{r7_2\}$ and $ext_v(Pathway_2) = \{X_2, Y_2, Z_2, X_2^*, Y_2^*, Z_2^*\}$.

Pathways $Pathway_1$ and $Pathway_2$ are shown graphically in Figure 5.2.

We now consider *networks* of pathways.

5.5 Modelling a network of independent pathways

Here we give the general definition of a network, and then consider the special case of networks of independent pathways. Later we consider networks with cross-talk.

A network is a parallel composition of pathways, with optional synchronisation of external reactions and sharing of variables between pathways.

Definition 15 (Network)**.** *A network is a composition of two pathways of the form P_1 $\{renamings_1\}$ $|[E\ \cup\ U,\ V]|$ P_2 $\{renamings_2\}$ where $|[E\ \cup\ U,\ V]|$ defines the interaction between P_1 and P_2. $renamings_1$ and $renamings_2$ are optional sets of renamings of reactions. E is the intersection of the sets of external reactions, modulo renamings, in P_1 and P_2, $E = ext_r(P_1\ \{renamings_1\}) \cap ext_r(P_2\ \{renamings_2\})$ and $U \subseteq ext_r(P_1\ \{renamings_1\})\ \cup\ ext_r(P_2\ \{renamings_2\})$. V is a set of variable sharings between P_1 and P_2.*

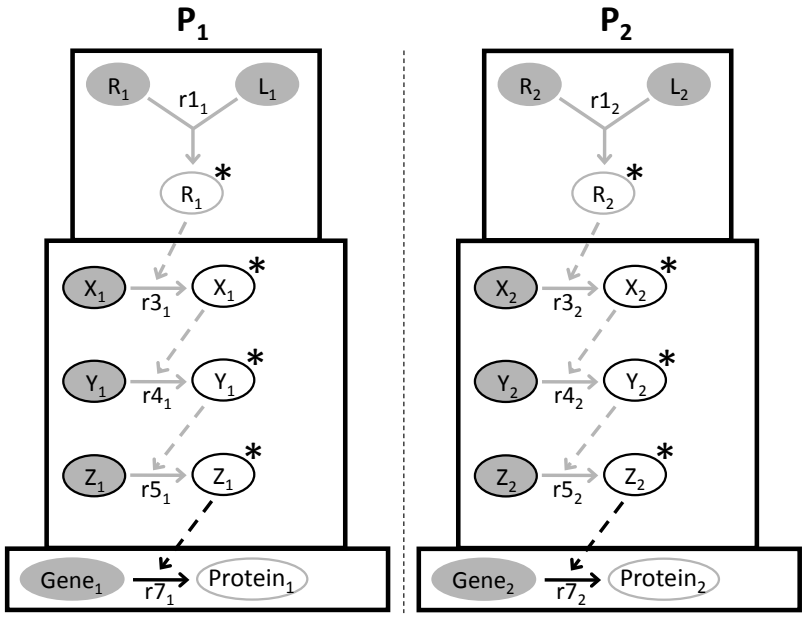


Figure 5.2: The two pathways $Pathway_1$ and $Pathway_2$ each comprise three instances of the generic pathway modules Receptor, 3-stage Cascade and Gene Expression. External reactions and variables are denoted by black lines, internal reactions and variables by grey lines and species that are present in the initial state by shaded ellipses.

Note, P_1, P_2, $renamings_1$, $renamings_2$, U and V determine the network. At this stage in the chapter, U is unimportant. U becomes important in the next section when we include auxiliary reactions to our pathways—U is then the auxiliary reactions (which are a subset of the external reactions) that are *unused*.

Now consider the special case of a network of independent pathways.

Definition 16 (Independent pathways)**.** *A network of two pathways* P_1 $\{renamings_1\}$ $|[E \cup U, V]|$ P_2 $\{renamings_2\}$ *is independent if there is no synchronisation of reactions and sharing of variables between the pathways, hence* $E = \emptyset$ *and* $V = \emptyset$*.*

We can compose our two example pathways independently as follows.

$Pathway_1$ $|[E \cup U, V]|$ $Pathway_2$

where $E = V = U = \emptyset$.

We now turn our attention to the case where there is synchronisation of reactions or sharing of variables between the pathways, i.e. there is cross-talk. However, before doing so we introduce the concept of auxiliary reactions, and ultimately how they result in the set of unused reactions U.

5.6 Auxiliary reactions

Auxiliary reactions are additional basic reactions and modifiers that can be used to express interactions between pathways.

Definition 17 (Auxiliary reactions)**.** *There are four types of auxiliary reactions for a species X as given in Figure 5.3.*

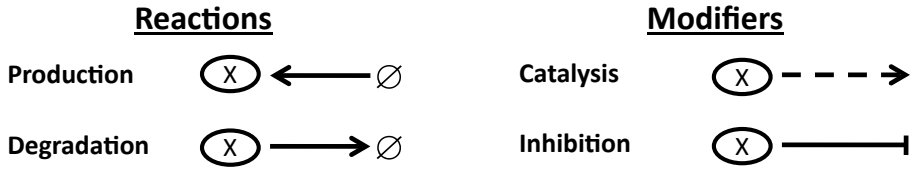


Figure 5.3: The four types of auxiliary reactions.

Production and degradation reactions are the two basic reactions for any species. All other reactions can be defined by synchronising production and degradation reactions. For example, we can express the formation of Z from X and Y by synchronising the degradation of X and Y with the production of Z.

Catalysis and inhibition are modifiers: they change the precondition of reactions (in PRISM). Catalysis and inhibition auxiliary reactions must synchronise with a reaction to make (biological) sense.

For any species we can add any number of any type of auxiliary reactions.

Definition 18 (PRISM implementation of auxiliary reactions)**.** *For any species X in a module any of the 4 types of auxiliary reactions can be added as follows.*

```
[prod]  X = 0 -> 1:(X' = 1);
[deg]   X = 1 -> 1:(X' = 0);
[cat]   X = 1 -> 1:true;
[inhib] X = 0 -> 1:true;
```

Note, although we have not defined an explicit syntax for adding auxiliary reactions, we assume that for any given pathway P we can augment it with a given set of auxiliary reactions $aux(P)$. For a pathway P there is an infinite number of augmentations of auxiliary reactions.

Definition 19 (Pathway auxiliary reactions)**.** *For a given pathway P, the set of auxiliary reactions is $aux(P)$ and we extend $ext_r(P)$ to include $aux(P)$, i.e. all auxiliary reactions are external.*

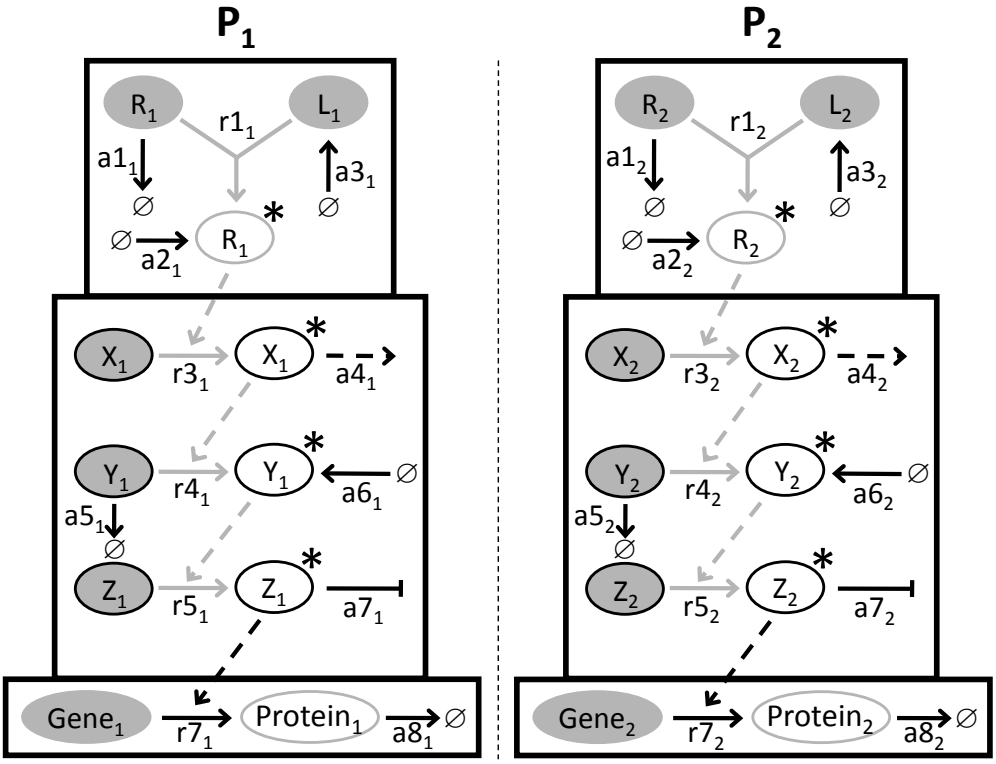


Figure 5.4: The two pathways $Pathway_1$ and $Pathway_2$ with added auxiliary reactions.

We add to our two example pathways some auxiliary reactions, the motivation for our choice of external reactions will be given in the next section.

We adopt the following convention. In $Pathway_j$ we label auxiliary reaction i as ai_j (or ai_j in PRISM).

In the Receptor module in $Pathway_j$ we add:

```
[a1_j] R = 1 -> 1:(R' = 0);
[a2_j] R* = 0 -> 1:(R*' = 1);
[a3_j] L = 0 -> 1:(L' = 1);
```

In the 3-stage Cascade module in $Pathway_j$ we add:

```
[a4_j] X* = 1 -> 1:true;
[a5_j] Y = 1 -> 1:(Y' = 0);
[a6_j] Y* = 0 -> 1:(Y*' = 1);
[a7_j] Z* = 0 -> 1:true;
```

In the Gene Expression module in $Pathway_j$ we add:

```
[a8_j] Protein = 1 -> 1:(Protein' = 0);
```

The pathways with added auxiliary reactions are shown graphically in Figure 5.4 and the PRISM model is given in Appendix C.

Auxiliary reactions are an integral part of our approach to modelling cross-talk. We model cross-talk by different combinations of synchronisation of external reactions (which includes the auxiliary reactions) and sharing of variables.

Definition 20 (Cross-talk)**.** *Given a network of two pathways* P_1 $\{renamings_1\}$ $|[E \cup U, V]|$ P_2 $\{renamings_2\}$*, there is cross-talk if there is at least one reaction* $e \in E$ *or one variable sharing* $v \in V$*. The number of cross-talks is* $|E| + |V|$*.*

We now introduce the concept of unused reactions U with the aid of the following functions.

Definition 21 (The "mapped" functions)**.** *Given a network of two pathways* P_1 $\{renamings_1\}$ $|[E \cup U, V]|$ P_2 $\{renamings_2\}$*, for any* $e \in E$*, we define the function* $mapped(e) = \{x_i \mid x_i \leftarrow e\} \cup \{u_i \mid u_i \leftarrow e\}$ *where* $renamings_1 = \{x_1 \leftarrow y_1, \ldots, x_n \leftarrow y_n\}$ *and* $renamings_2 = \{u_1 \leftarrow v_1, \ldots, u_n \leftarrow v_n\}$*. We also define* $mapped(E) = \bigcup_{e \in E} mapped(e)$*. In other words,* $mapped(e)$ *is the set of reactions involved in one synchronisation* e *between the pathways and* $mapped(E)$ *is the set of reactions involved in any synchronisation* $e \in E$ *between the pathways.*

The unused reactions U are the auxiliary reactions that are not used for synchronisation between two pathways, and so $U = (aux(P_1) \cup aux(P_2)) \setminus mapped(E)$. Note that since each reaction in U occurs in only one pathway, they cannot synchronise and their corresponding transitions never execute. This will be explored in detail in the next section, where we turn our attention to a network of two pathways in which there is cross-talk.

5.7 Categorisation of cross-talk

We model cross-talk by expressing instances of different types of cross-talk between pathways

Although there is some discussion of types of cross-talk [50,81], there is no universal categorisation in the literature. In this section we propose that there are five types of cross-talk: substrate availability, signal flow, receptor function, gene expression and intracellular communication. We note that four of the five types are alluded to in [81] but are not made specific.

We give the motivation for the five types using indicative examples from the literature. We define several functions that are used in the formalisation. We then formalise the types and prove

that the types are distinct. Finally, we give examples of each type of cross-talk using our example pathways $Pathway_1$ and $Pathway_2$.

5.7.1 Motivation for types

In this section we give evidence of each of the five types. While there is no proof that there are no other types of cross-talk, we have found no examples after performing an exhaustive literature search and discussing this with domain experts.

Substrate availability cross-talk In [76] there are two pathways that compete for the activation of the MAPK cascade. The pathways share the MAPKKK protein STE11 and have homologous MAPKK and MAPK proteins.

Signal flow cross-talk In [98] there is signal flow cross-talk between the MAPK and Integrin signalling pathways. Activation of the Integrin pathway enhances signalling through the MAPK pathway by increased rate of activation of key proteins in the pathway.

Receptor function cross-talk In [65] other signalling pathways can activate the Estrogen receptor in the absence of the Estrogen ligand.

Gene expression cross-talk In [32] two pathways contain cross-talk within the nucleus. One pathway contains a transcription factor GR that resides outside the nucleus. Upon signalling, GR relocates to the nucleus and inhibits the transcription factor NF-κB that is activated by another pathway.

Intracellular communication cross-talk In [50] the TGF-β and WNT pathways reciprocally regulate the production of their ligands. There is some contention in the literature as to whether this is genuine cross-talk: the interaction is less direct than other types of cross-talk and involves (temporally) long processes such as gene expression and ligand excretion.

We now turn to formalising the 5 types, but before doing so, we define several useful functions on the generic modules.

5.7.2 Functions on modules

For the convenience of the reader, we repeat the three modules (with auxiliary reactions) below. Recall that r2 and r6 are not reactions in their own right because they do not update the state of the system.

```
module Receptor
   R : [0..1] init 1;      L : [0..1] init 1;      R* : [0..1] init 0;

   [r1] R = 1 & L = 1 & R* = 0 -> 1:(R' = 0) & (L' = 0) & (R*' = 1);
   [r2] R* = 1 -> 1:true;

   [a1_j] R = 1 -> 1:(R' = 0);
   [a2_j] R* = 0 -> 1:(R*' = 1);
   [a3_j] L = 0 -> 1:(L' = 1);
endmodule

module 3StageCascade
   X : [0..1] init 1;   X* : [0..1] init 0;
   Y : [0..1] init 1;   Y* : [0..1] init 0;
   Z : [0..1] init 1;   Z* : [0..1] init 0;

   [r3] X = 1 & X* = 0 -> 1:(X' = 0) & (X*' = 1);
   [r4] Y = 1 & Y* = 0 & X* = 1 -> 1:(Y' = 0) & (Y*' = 1);
   [r5] Z = 1 & Z* = 0 & Y* = 1 -> 1:(Z' = 0) & (Z*' = 1);
   [r6] Z* = 1 -> 1:true;

   [a4_j] X* = 1 -> 1:true;
   [a5_j] Y = 1 -> 1:(Y' = 0);
   [a6_j] Y* = 0 -> 1:(Y*' = 1);
   [a7_j] Z* = 0 -> 1:true;
endmodule

module GeneExpression
   Gene : [0..1] init 1;      Protein : [0..1] init 0;

   [r7] Gene = 1 & Protein = 0 -> 1:(Gene' = 0) & (Protein' = 1);

   [a8_j] Protein = 1 -> 1:(Protein' = 0);
endmodule
```

We now define several functions that operate on modules, of type $func : Module \rightarrow \{Labels\}$.

all – all reactions

all(Receptor) = $\{a1, a2, a3, r1\}$

all(3StageCascade) = $\{a4, a5, a6, a7, r3, r4, r5\}$

all(GeneExpression) = $\{a8, r7\}$

trans – all transformation reactions

trans(Receptor) = $\{a1, a2, a3, r1\}$

trans(3StageCascade) = $\{a5, a6, r3, r4, r5\}$

$trans$(GeneExpression) = $\{a8, r7\}$

mod – all modifiers

mod(Receptor) = $\emptyset$

mod(3StageCascade) = $\{a4, a7\}$

mod(GeneExpression) = $\emptyset$

Note that $\forall x \,.\, all(x) = trans(x) \cup mod(x)$.

$catalysis$ – all catalysis reactions

$catalysis$(Receptor) = $\emptyset$

$catalysis$(3StageCascade) = $\{a4\}$

$catalysis$(GeneExpression) = $\emptyset$

$inhib$ – all inhibition reactions

$inhib$(Receptor) = $\emptyset$

$inhib$(3StageCascade) = $\{a7\}$

$inhib$(GeneExpression) = $\emptyset$

$prod$ – all production reactions

$prod$(Receptor) = $\{a2, a3\}$

$prod$(3StageCascade) = $\{a6\}$

$prod$(GeneExpression) = $\emptyset$

deg – all degradation reactions

deg(Receptor) = $\{a1\}$

deg(3StageCascade) = $\{a5\}$

deg(GeneExpression) = $\{a8\}$

$receptor\_deg$ – all degradation of (inactive) receptor reactions

$receptor\_deg$(Receptor) = $\{a1\}$

$receptor\_deg$(3StageCascade) = $\emptyset$

$receptor\_deg$(GeneExpression) = $\emptyset$

receptor_act – all ligand-receptor binding reactions

receptor_act(Receptor) = $\{r1\}$

receptor_act(3StageCascade) = $\emptyset$

receptor_act(GeneExpression) = $\emptyset$

active_receptor_prod – all production of active receptor reactions

active_receptor_prod(Receptor) = $\{a2\}$

active_receptor_prod(3StageCascade) = $\emptyset$

active_receptor_prod(GeneExpression) = $\emptyset$

ligand_prod – all production of ligand reactions

ligand_prod(Receptor) = $\{a3\}$

ligand_prod(3StageCascade) = $\emptyset$

ligand_prod(GeneExpression) = $\emptyset$

gene_expression – all gene expression reactions

gene_expression(Receptor) = $\emptyset$

gene_expression(3StageCascade) = $\emptyset$

gene_expression(GeneExpression) = $\{r7\}$

5.7.3 Cross-talk types

We now formalise the 5 types of cross-talk: substrate availability, signal flow, receptor function, gene expression and intracellular communication, in terms of the three modules introduced so far. Extending the formalisation to include extra modules that behave in a similar way is trivial.

Given a network of the form P_1 $\{renamings_1\}$ $|[E \cup U, V]|$ P_2 $\{renamings_2\}$, a cross-talk is either an $e \in E$ or a $v \in V$.

A cross-talk $v \in V$ is always **substrate availability cross-talk**.

A cross-talk $e \in E$ is categorised according to the rules below. Rules are either necessary rules or biological constraints. Biological constraints prevent infeasible cross-talk due to, for example, different cellular locations or different species types. These constraints concern which reactions can be synchronised between two pathways, thus a constraint always concerns members

of $mapped(e)$.

We use the notation $Module\_Type \in P_i$ to mean a *module* in P_i of type $Module\_Type$ and $x \in P_i$ to mean a reaction x defined in P_i. The operator $\exists!x \,.\, f(x)$ means one and only one x such that $f(x)$ holds and is defined by $\exists x \,.\, (f(x) \wedge \forall y \,.\, (f(y) \rightarrow y = x))$

Finally, we note by the definition of E in a network the following properties hold.

- $\forall e \in E \,.\, \forall x \in mapped(e) \,.\, (x \in P_1 \vee x \in P_2)$
- $\forall e \in E \,.\, \exists x \in mapped(e) \,.\, (x \in P_1)$
- $\forall e \in E \,.\, \exists x \in mapped(e) \,.\, (x \in P_2)$

Signal flow cross-talk e is signal flow cross-talk if and only if the rules in Case 1 or Case 2 hold.

Case 1: P_1 affects a transformation reaction in P_2 or vice-versa.

Rules
$\exists x \in mapped(e) \,.\, \exists$3StageCascade $\in (P_1 \cup P_2) \,.\, x \in trans$(3StageCascade)
Biological constraints
$\forall x \in mapped(e) \,.\, \forall$Receptor $\in (P_1 \cup P_2) \,.\, x \notin trans$(Receptor)
$\forall x \in mapped(e) \,.\, \forall$GeneExpression $\in (P_1 \cup P_2) \,.\, x \notin all$(GeneExpression)

Case 2: P_1 produces a protein in P_2 or vice-versa, or else P_2 catalyses P_1's production of a protein, or vice-versa, or P_2 inhibits P_1 from producing a protein, or vice-versa.

Rules
$\exists! x \in mapped(e) \,.\, \exists$GeneExpression $\in (P_1 \cup P_2) \,.\, x \in deg$(GeneExpression)
$\exists! x \in mapped(e) \,.\, \exists$3StageCascade $\in (P_1 \cup P_2) \,.\, x \in prod$(3StageCascade)
Biological constraints
$\forall x \in mapped(e) \,.\, \forall$Receptor $\in (P_1 \cup P_2) \,.\, x \notin all$(Receptor)
$\forall x \in mapped(e) \,.\, \forall$3StageCascade $\in (P_1 \cup P_2) \,.\, x \in trans$(3StageCascade)
$\rightarrow x \in prod$(3StageCascade)
$\forall x \in mapped(e) \,.\, \forall$GeneExpression $\in (P_1 \cup P_2) \,.\, x \in all$(GeneExpression)
$\rightarrow x \in deg$(GeneExpression)

Receptor function cross-talk e is receptor function cross-talk if and only if the rules in Case 1, Case 2 or Case 3 hold.

Case 1: P_1 catalyses P_2's receptor degradation, or vice-versa, with possible modifiers from 3-stage Cascades.

Rules
$\exists! x \in mapped(e)$. $\exists$Receptor $\in P_i$. $x \in receptor\_deg$(Receptor)
$\exists x \in mapped(e)$. $\exists$3StageCascade $\in P_j$. $x \in catalysis$(3StageCascade)
where $i \neq j$
Biological constraints
$\forall x \in mapped(e)$. $\forall$GeneExpression $\in (P_1 \cup P_2)$. $x \notin all$(GeneExpression)
$\forall x \in mapped(e)$. $\forall$3StageCascade $\in (P_1 \cup P_2)$. $x \notin trans$(3StageCascade)
$\forall x \in mapped(e)$. $\forall$Receptor $\in (P_1 \cup P_2)$. $x \in all$(Receptor)
$\rightarrow x \in receptor\_deg$(Receptor)

Case 2: The activation of P_1's receptor is inhibited by P_2, or vice-versa, with possible extra modifiers from 3-stage Cascades.

Rules
$\exists! x \in mapped(e)$. $\exists$Receptor $\in P_i$. $x \in receptor\_act$(Receptor)
$\exists x \in mapped(e)$. $\exists$3StageCascade $\in P_j$. $x \in inhib$(3StageCascade)
where $i \neq j$
Biological constraints
$\forall x \in mapped(e)$. $\forall$GeneExpression $\in (P_1 \cup P_2)$. $x \notin all$(GeneExpression)
$\forall x \in mapped(e)$. $\forall$3StageCascade $\in (P_1 \cup P_2)$. $x \notin trans$(3StageCascade)
$\forall x \in mapped(e)$. $\forall$Receptor $\in (P_1 \cup P_2)$. $x \in all$(Receptor)
$\rightarrow x \in receptor\_act$(Receptor)

Case 3: P_1's receptor is activated without the need for a ligand and this is catalysed by P_2, or vice-versa, with possible extra modifiers from 3-stage Cascades.

Rules
$\exists! x \in mapped(e)$. $\exists$Receptor $\in P_i$. $x \in receptor\_deg$(Receptor)
$\exists! x \in mapped(e)$. $\exists$Receptor $\in P_i$. $x \in active\_receptor\_prod$(Receptor)
$\exists x \in mapped(e)$. $\exists$3StageCascade $\in P_j$. $x \in catalysis$(3StageCascade)
where $i \neq j$
Biological constraints
$\forall x \in mapped(e)$. $\forall$3StageCascade $\in (P_1 \cup P_2)$. $x \notin trans$(3StageCascade)
$\forall x \in mapped(e)$. $\forall$GeneExpression $\in (P_1 \cup P_2)$. $x \notin all$(GeneExpression)
$\forall x \in mapped(e)$. $\forall$Receptor $\in (P_1 \cup P_2)$. $x \in all$(Receptor)
$\rightarrow (x \in receptor\_deg$(Receptor) $\vee$ $x \in active\_receptor\_prod$(Receptor))

Gene expression cross-talk The rate of P_1's gene expression reaction is modified by a species in a 3-stage Cascade module in P_2 or vice-versa.

Rules
$\exists! x \in mapped(e)$. $\exists$GeneExpression $\in (P_1 \cup P_2)$.
$x \in gene\_expression$(GeneExpression)
$\exists x \in mapped(e)$. $\exists$3StageCascade $\in (P_1 \cup P_2)$. $x \in mod$(3StageCascade)
Biological constraints
$\forall x \in mapped(e)$. $\forall$Receptor $\in (P_1 \cup P_2)$. $x \notin all$(Receptor)
$\forall x \in mapped(e)$. $\forall$3StageCascade $\in (P_1 \cup P_2)$. $x \notin trans$(3StageCascade)
$\forall x \in mapped(e)$. $\forall$GeneExpression $\in (P_1 \cup P_2)$. $x \in all$(GeneExpression)
$\rightarrow x \in gene\_expression$(GeneExpression)

Intracellular Communication cross-talk A protein is released from P_1 that is the ligand for P_2, or vice-versa, with possible extra modifiers from 3-stage Cascades.

Rules

$(\exists! x \in mapped(e)\ .\ \exists \text{GeneExpression}\ \in P_i\ .\ x \in deg(\text{GeneExpression})) \vee$

$(\exists! x \in mapped(e)\ .\ \exists \text{3StageCascade}\ \in P_i\ .\ x \in deg(\text{3StageCascade}))$

$\exists x \in mapped(e)\ .\ \exists \text{Receptor}\ \in P_j\ .\ x \in ligand\_prod(\text{Receptor})$

where $i \neq j$

Biological constraints

$\forall x \in mapped(e)\ .\ \forall \text{GeneExpression}\ \in (P_1 \cup P_2)\ .\ x \in all(\text{GeneExpression})$

$\rightarrow x \in deg(\text{GeneExpression})$

$\forall x \in mapped(e)\ .\ \forall \text{3StageCascade}\ \in (P_1 \cup P_2)\ .\ x \in trans(\text{3StageCascade})$

$\rightarrow x \in deg(\text{3StageCascade})$

$\forall x \in mapped(e)\ .\ \forall \text{Receptor}\ \in (P_1 \cup P_2)\ .\ x \in all(\text{Receptor})$

$\rightarrow x \in ligand\_prod(\text{Receptor})$

5.7.4 Categorisation is well-defined

We prove below that the categorisation is well-defined.

Theorem 1 (Well-defined categorisation)**.** *The cross-talk categorisation is well-defined, i.e. any cross-talk that has been categorised has only one type.*

Proof. Trivially, any cross-talk v is of type substrate availability cross-talk.

We now in-turn assume a cross-talk e of each type and give a *witness*, a label that must/must not be part of the cross-talk, that prevents it from being another type of cross-talk.

Suppose e has been categorised as signal flow cross-talk. In both cases of signal flow cross-talk, no transformation reactions from a Receptor module are allowed in $mapped(e)$, hence if $x \in \{a1, a2, a3, r1\}$ then $x \notin mapped(e)$.

A receptor function cross-talk must have a label from $receptor\_deg$(Receptor) (Case 1 and Case 3) or a label from $receptor\_act$(Receptor) (Case 2). Therefore, $a1 \in mapped(e) \ \vee \ r1 \in mapped(e)$.

The witness to e, a signal flow cross-talk, not being a receptor function cross-talk is $a1 \notin mapped(e)$ and $r1 \notin mapped(e)$.

Assuming e is each type of cross-talk in turn, we list below the witnesses that prove e can have no other type.

e is signal flow cross-talk	Witness
cannot be receptor function	$\forall x \in \{a1, r1\} \,.\, x \notin mapped(e)$
cannot be gene expression	$r7 \notin mapped(e)$
cannot be intracellular communication	$a3 \notin mapped(e)$

e is receptor function cross-talk	Witness
cannot be signal flow	$\forall x \in \{a5, a6, r3, r4, r5\} \,.\, x \notin mapped(e)$
cannot be gene expression	$r7 \notin mapped(e)$
cannot be intracellular communication	$a3 \notin mapped(e)$

e is gene expression cross-talk	Witness
cannot be signal flow	$\forall x \in \{a5, a6, r3, r4, r5\} \,.\, x \notin mapped(e)$
cannot be receptor function	$\forall x \in \{a1, r1\} \,.\, x \notin mapped(e)$
cannot be intracellular communication	$a3 \notin mapped(e)$

e is intracellular communication cross-talk	Witness
cannot be signal flow	$a3 \in mapped(e)$
cannot be receptor function	$\forall x \in \{a1, r1\} \,.\, x \notin mapped(e)$
cannot be gene expression	$r7 \notin mapped(e)$

□

5.7.5 Examples of cross-talk

Given two pathways, we can generate all instances of cross-talk (comprising up to k synchronisations) that our formalisation allows using the $Generate(P_1, P_2, k)$ algorithm given in Section 5.8; it depends upon k, the maximum number of synchronisations on a reaction.

Applying $Generate(Pathway_1, Pathway_2, 3)$ and quantifying over all possible renamings and synchronisations, yields 757 *candidate* examples of cross-talk (that do not necessarily satisfy the biological constraints) and 175 *actual* examples of cross-talk. The latter are categorised as follows: 36 substrate availability, 65 signal flow, 18 receptor function, 28 gene expression, and 28 intracellular communication cross-talks. Below, we give an example of each type.

Substrate availability cross-talk The pathways compete for activation of protein X, hence the pathways share variables X_1 and X_2.

$Pathway_1 \,|[E \cup U, V]|\, Pathway_2$

where $U = aux(Pathway_1) \cup aux(Pathway_2)$, $E = \emptyset$ and $V = \{(X_1, X_2)\}$. This example is shown in Figure 5.5a.

Signal flow cross-talk An alternative reaction to activate Y_1 through the X_2^* enzyme. Synchronise $a5_1$ (the degradation of Y_1) with $a6_1$ (the production of Y_1^*) and with $a4_2$ (the enzymatic activity of X_2^*)[5].

$Pathway_1 \{a5_1 \leftarrow r_{new}, a6_1 \leftarrow r_{new}\} \,|[E \cup U, V]|\, Pathway_2 \{a4_2 \leftarrow r_{new}\}$

where $U = (aux(Pathway_1) \cup aux(Pathway_2)) \setminus \{a5_1, a6_1, a4_2\}$, $E = \{r_{new}\}$ and $V = \emptyset$. This example is shown in Figure 5.5b.

Receptor function cross-talk An alternative reaction to activate receptor R_2 by the enzyme X_1^*. Synchronise $a1_2$ (the degradation of receptor R_2) with $a2_2$ (the production of the active receptor R_2^*) and with $a4_1$ (the enzymatic activity of X_1^*).

$Pathway_1 \{a4_1 \leftarrow r_{new}\} \,|[E \cup U, V]|\, Pathway_2 \{a1_2 \leftarrow r_{new}, a2_2 \leftarrow r_{new}\}$

where $U = (aux(Pathway_1) \cup aux(Pathway_2)) \setminus \{a4_1, a1_2, a2_2\}$, $E = \{r_{new}\}$ and $V = \emptyset$. This example is shown in Figure 5.5c.

Gene expression cross-talk Inhibit the expression of $Gene_1$ by the Z_2^* protein. Synchronise $a7_2$ (the inhibiting activity of Z_2^*) with $r7_1$ (the expression of $Gene_1$).

$Pathway_1 \{r7_1 \leftarrow r_{new}\} \,|[E \cup U, V]|\, Pathway_2 \{a7_2 \leftarrow r_{new}\}$

where $U = (aux(Pathway_1) \cup aux(Pathway_2)) \setminus \{a7_2, r7_1\}$, $E = \{r_{new}\}$ and $V = \emptyset$. This example is shown in Figure 5.5d.

Intracellular communication cross-talk The output of expressing $Gene_1$ is the ligand for $Pathway_2$. Synchronise $a8_1$ (the degradation of $Protein_1$) with $a3_2$ (the production of the ligand L_2).

$Pathway_1 \{a8_1 \leftarrow r_{new}\} \,|[E \cup U, V]|\, Pathway_2 \{a3_2 \leftarrow r_{new}\}$

where $U = (aux(Pathway_1) \cup aux(Pathway_2)) \setminus \{a8_1, a3_2\}$, $E = \{r_{new}\}$ and $V = \emptyset$. This example is shown in Figure 5.5e.

[5] notice that this synchronisation includes a synchronisation within a module

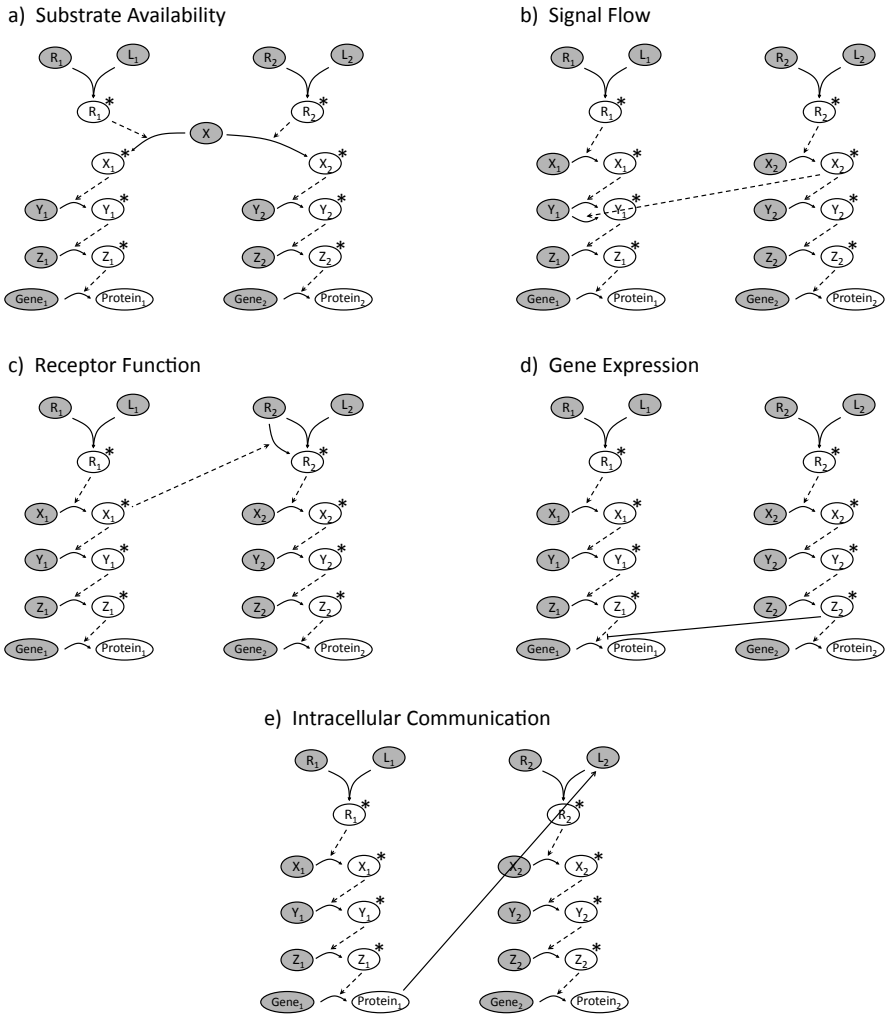


Figure 5.5: An example of each of the five types of cross-talk. a) two pathways compete for a protein. b) a pathway up-regulates signal flow through another pathway. c) a pathway activates the receptor of another pathway in the absence of a ligand. d) two pathways have conflicting transcriptional responses. e) a pathway releases a ligand for another pathway.

5.8 Cross-talk generation – Generate()

We now give the algorithm $Generate(P_1, P_2, k)$ to generate all possible instances of cross-talk between two pathways P_1 and P_2. First, we consider substrate availability cross-talk and then, all other types of cross-talk.

To generate every substrate availability cross-talk we share between pathways every pair of (external) variables.

---

for variable $v \in ext_v(P_1)$ **do**
  for variable $w \in ext_v(P_2)$ **do**
    $P_1\ \{renamings_1\}\ |[E\ \cup\ U,\ V]|\ Pathway_2\ \{renamings_2\}$ where $U = aux(P_1)\ \cup\ aux(P_2)$, $E = \emptyset$, $renamings_1 = \emptyset$, $renamings_2 = \emptyset$ and $V = \{(v, w)\}$
  end for
end for

To generate every cross-talk of all other types we create all possible *candidate* cross-talks by synchronising up to k (external) reactions.

---

for $i \geq 1, j \geq 1$ such that $i + j \leq k$ **do**
  for X = choose i reactions from $ext_r(P_1)$ **do**
    for Y = choose j reactions from $ext_r(P_2)$ **do**
      if $X \cup Y$ contains only modifiers **then** skip **else**
      $P_1\ \{renamings_1\}\ |[E\ \cup\ U,\ V]|\ P_2\ \{renamings_2\}$ where $renamings_1$ such that $\forall x \in X\ .\ x \leftarrow r_{new}$, $renamings_2$ such that $\forall y \in Y\ .\ y \leftarrow r_{new}$, $E = \{r_{new}\}$ and
      $U = (aux(P_1\ \{renamings_1\})\ \cup\ aux(P_2\ \{renamings_2\}))\ \setminus\ mapped(r_{new})$,
    end for
  end for
end for

We then filter the candidate cross-talks according to the categorisation—those cross-talks that are not categorised are removed.

5.8.1 Higher order networks

We have defined how to model networks of two pathways. Higher order networks can be modelled by composing a network with a single pathway, hence:

$Network_2 = Pathway_1\ |[E_1\ \cup\ U_1,\ V_1]|\ Pathway_2$

$Network_3 = Network_2\ |[E_2\ \cup\ U_2,\ V_2]|\ Pathway_3$

...

$Network_i = Network_{i-1}\ |[E_{i-1}\ \cup\ U_{i-1},\ V_{i-1}]|\ Pathway_i$

However, to the best of our knowledge all cross-talk are between pairs of pathways. Our definition of a network allows a single cross-talk in which three or more pathways participate,

however we have found no biological examples of such an interaction have been reported.

This concludes modelling cross-talk. We now change focus to analysing models of cross-talk using logical properties.

5.9 Detecting cross-talk

So far we have discussed the main contribution of this chapter, how to model cross-talk in a rigorous way by looking at the form of the model description, e.g. the synchronisations between PRISM modules. We now give preliminary results on the complementary problem, how to analyse at the model level (i.e. at the level of the CTMC rather than the form of the model description[6]). We aim to both detect and characterise cross-talk. We first tackle detecting cross-talk in these models.

This section makes use of the two pathways $Pathway_1$ and $Pathway_2$ introduced in Section 5.4 and the five example cross-talk models of Section 5.7.5.

The presence of cross-talk can be detected by checking a set of temporal logic properties as follows.

We choose CSL because we need a quantitative logic—it is a change in the probability of a formula being true that allows us to detect the presence of a cross-talk. The probabilities are used to measure the number of paths that satisfy a property. For example, in the signal flow cross-talk example (compared to the independent pathways model) there is a greater number of paths to the expression of $Protein_1$. There are other ways to detect cross-talk, however we use model checking of CSL properties as it is relatively straightforward and familiar to a large part of the community.

Given pathways $Pathway_1$ and $Pathway_2$ that conclude with gene expression ($Protein_1$ and $Protein_2$ being produced respectively), we detect cross-talk by comparing the probabilities of the following three CSL formulae with the probability of the formulae in the independent pathways model. Namely, we compare probabilities for the five example cross-talk models of Section 5.7.5 with $Pathway_1\ \{renamings_1\}\ |[E\ \cup\ U,\ V]|\ Pathway_2\ \{renamings_2\}$ where $E = V = \emptyset$. In each case, cross-talk is indicated by a change of probability of at least one formula.

Competitive Signal Flow ($Pathway_1$ before $Pathway_2$): probability of signal flow through $Pathway_1$ before $Pathway_2$

[6]for example, we define CTMCs in PRISM using the PRISM *language* whereas in a tool like Matlab, we would define them with *equations*

	ψ_1	ψ_2	ψ_3
Substrate Availability Example	=	↓	↓
Signal Flow Example	↑	↑	=
Receptor Function Example	↓	=	↑
Gene Expression Example	↓	↓	=
Intracellular Communication Example	=	=	=

Table 5.1: The change in probability for each of the 5 cross-talk models compared with the independent pathways model for the three CSL properties.

$$\psi_1 \;=\; \mathbf{P}_{=?}\,[\; F\;(\mathit{Protein}_1 \;=\; 1 \;\wedge\; \mathit{Protein}_2 \;=\; 0)\;]$$

Independent Signal Flow ($Pathway_1$): probability of signal flow through $Pathway_1$ within a time bound (3 time units)

$$\psi_2 \;=\; \mathbf{P}_{=?}\,[\; F_{\leq 3}\;(\mathit{Protein}_1 \;=\; 1)\;]$$

Independent Signal Flow ($Pathway_2$): probability of signal flow through $Pathway_2$ within a time bound (3 time units)

$$\psi_3 \;=\; \mathbf{P}_{=?}\,[\; F_{\leq 3}\;(\mathit{Protein}_2 \;=\; 1)\;]$$

The change in probability for each of the 5 cross-talk models, as compared to the independent pathways model, is given in Table 5.1. ↑ denotes an increase, ↓ denotes a decrease and = denotes no change in probability. Results were obtained using the PRISM model checker (run times are negligible).

Notice that there is no change in probability for the intracellular communication cross-talk model. In our qualitative model of this cross-talk, one pathway produces a ligand for another pathway only after the original ligand molecule has been consumed in a reaction. This means that the cross-talk has no effect on the rate of the activation reactions in either pathway. In a model with a greater level of quantitative detail, as discussed in Section 5.12, this cross-talk would change the rate of the activation reactions. This result is not unexpected as we have already identified that intracellular communication cross-talk is a source of contention in the life sciences.

We now move on to characterising cross-talk in models in which there is no model description.

5.10 Characterising cross-talk

The type of cross-talk can be characterised at the model level using different temporal logic properties.

We choose CTL because we only need a qualitative logic—it is a difference in the structure of the Markov chain rather than the transition rates that allows us to distinguish between types of cross-talks. We define 5 CTL properties, each of which characterises a type of cross-talk. The properties are simple liveness or safety properties.

As before, the activation of a pathway is reflected by the expression of *Protein*.

Substrate availability example It is not possible to activate X in both pathways (i.e. the pathways compete for a limited protein).

$$\mathbf{A}\ [\ G \neg\ (X_1^* = 1 \wedge X_2^* = 1)\ ]$$

Signal flow example It is possible to activate $Pathway_1$ without activating receptor R_1.

$$\mathbf{E}\ [\ F\ (R_1^* = 0 \wedge Protein_1 = 1)\ ]$$

Receptor function example It is possible to activate the receptor R_2 without using the ligand L_2.[7]

$$\mathbf{E}\ [\ F\ (R_1^* = 0 \wedge R_2^* = 1 \wedge L_2 = 1)\ ]$$

Gene expression example It is not possible to activate $Pathway_1$ if the signal has already passed through $Pathway_2$.

$$\mathbf{A}\ [\ G \neg\ (Protein_1 = 1)\ \{Y_1^* = 1 \wedge Z_1^* = 0 \wedge Protein_2^* = 1\}\ ]$$

Intracellular communication example It is possible to use and replenish ligand L_2.

$$\mathbf{E}\ [\ (L_2 = 1)\ U\ (\ (L_2 = 0) \wedge \mathbf{E}\ [\ (L_2 = 0)\ U\ (L_2 = 1)\ )\ ]\ ]$$

We now demonstrate our approach on a prominent case study of the cross-talk between the TGF-β, WNT and MAPK pathways.

[7] We include $R_1^* = 0$ because signalling in $Pathway_1$ in the intracellular communication model can produce L_2.

5.11 Case study: TGF-β, WNT and MAPK pathways

We apply our approach to a prominent biological case study of the cross-talk between the TGF-β, WNT and MAPK pathways. Details are taken from [50]. We use the approach to classify the cross-talk in the model and to understand the effects of the cross-talk on the TGF-β pathway. We note that the effects of cross-talk are not discussed in [50].

5.11.1 Biological background

Transforming Growth Factor β (TGF-β) is a family of cytokines (ligands used for cellular communication) including the TGF-β and Bone Morphogenic Protein (BMP) ligands. The TGF-β pathway controls many biological functions including cellular proliferation, differentiation and apoptosis, and immune function [50]. The study of this pathway is also crucial in understanding many diseases, especially the study of tumour invasion and metastasis.

Cross-talk is a "perennial theme" in the study of the TGF-β pathway [2]. It has been known for some time to play a large role in the complexity of the pathway response. A recent review paper [50] explains developments in this field, especially using the diagrams in Figure 5.6. We review the cross-talk between the TGF-β and two other pathways below.

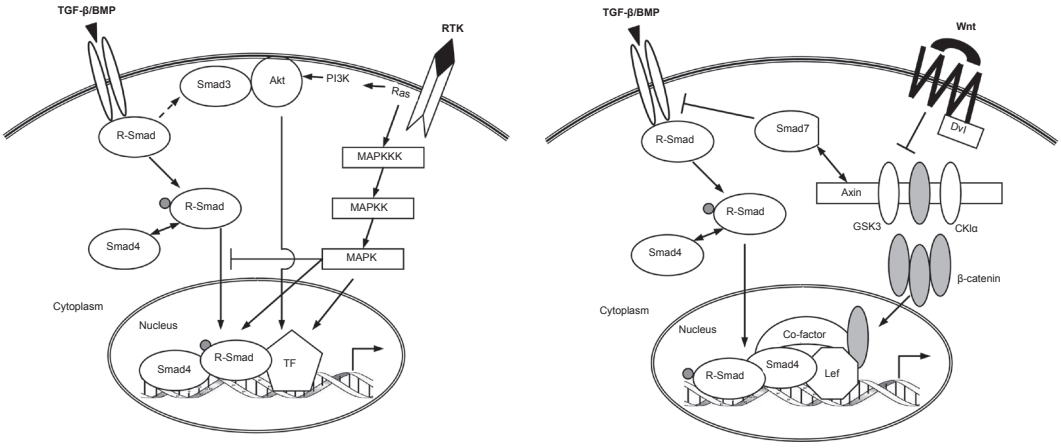


Figure 5.6: Two biological cartoons outlining the cross-talk between the TGF-β and MAPK (left) and TGF-β and WNT (right) pathways. Figure reproduced from [50].

TGF-β pathway The TGF-β pathway contains many signal transducers called Smads. The activated ligand-receptor complex phosphorylates a subset of Smads called R-Smads (Receptor regulated Smads). Activated R-Smads bind to Smad4, translocate to the nucleus and regulate gene expression by binding directly to the DNA. The Smad7 protein can deactivate the receptor, how-

ever Smad7 can be degraded by Axin (from the WNT pathway). MAPK proteins (from the MAPK pathway) inactivates R-Smads and degrades Smad4.

WNT pathway The effect of the WNT pathway is controlled by the state of the β-Catenin protein, which can translocate to the nucleus and cause gene expression. The Axin scaffold proteins combine two other proteins GSK3 and CKIα, then degrades β-Catenin. Active Smad7 is also implied in β-Catenin degradation. However, β-Catenin degradation is inhibited by WNT signalling, through the activation of Dvl, causing Axin down-regulation and β-Catenin stabilisation.

MAPK pathway Receptor tyrosine kinases, a class of receptors, stimulate both the activation of MAPK proteins and the AKT protein through the activation of a protein called Ras. The active MAPK proteins and AKT both enhance the expression of genes in the nucleus. Note that the inclusion of the PI3K and AKT proteins indicates an implicit cross-talk as these proteins are integral parts of the PI3K and AKT pathways respectively.

5.11.2 Modelling the pathways

To apply our modelling approach we need to expand our set of modules to Receptor, Protein Activation, 2-stage Cascade, 3-stage Cascade, Translocation, Protein Binding and Gene Expression. This is a natural extension of our approach.

Because there are extra modules, there may be new instances of cross-talks that our formalisation does not capture.

The TGF-β, WNT and MAPK pathways and their cross-talk are shown in Figure 5.7.

We define the following three pathways (for brevity, we omit the synchronisation sets and renamings).

$$
\begin{aligned}
MAPK \;=\; & (\text{Receptor } \{\ldots\} \;|[\ldots]|\; \text{ProteinActivation } \{\ldots\} \;|[\ldots]|\; \text{3StageCascade } \{\ldots\} \;|[\ldots]| \\
& \text{2StageCascade } \{\ldots\} \;|[\ldots]|\; \text{Translocation } \{\ldots\} \;|[\ldots]|\; \text{Translocation}) \\
TGFB \;=\; & (\text{Receptor } \{\ldots\} \;|[\ldots]|\; \text{ProteinActivation } \{\ldots\} \;|[\ldots]|\; \text{ProteinActivation } \{\ldots\} \;|[\ldots]| \\
& \text{ProteinBinding } \{\ldots\} \;|[\ldots]|\; \text{Translocation } \{\ldots\} \;|[\ldots]|\; \text{GeneExpression } \{\ldots\} \;|[\ldots]| \\
& \text{GeneExpression}) \\
WNT \;=\; & (\text{Receptor } \{\ldots\} \;|[\ldots]|\; \text{ProteinActivation } \{\ldots\} \;|[\ldots]|\; \text{Translocation } \{\ldots\} \;|[\ldots]| \\
& \text{GeneExpression})
\end{aligned}
$$

The following auxiliary reactions are added. In the MAPK pathway we add catalysis auxiliary reactions to the MAPK*, AKT* and TF* species. In the TGF-β pathway we add degradation

auxiliary reactions to the Smad4 and Smad7 species. In the WNT pathway we add catalysis auxiliary reactions to the Axin and β-Catenin$*$ species.

We then consider four networks,

$TGFB$,

$TGFB \,|[\ldots]|\, MAPK$,

$TGFB \,|[\ldots]|\, WNT$ and

$(TGFB \,|[\ldots]|\, MAPK) \,|[\ldots]|\, WNT$ (referred to as the full network).

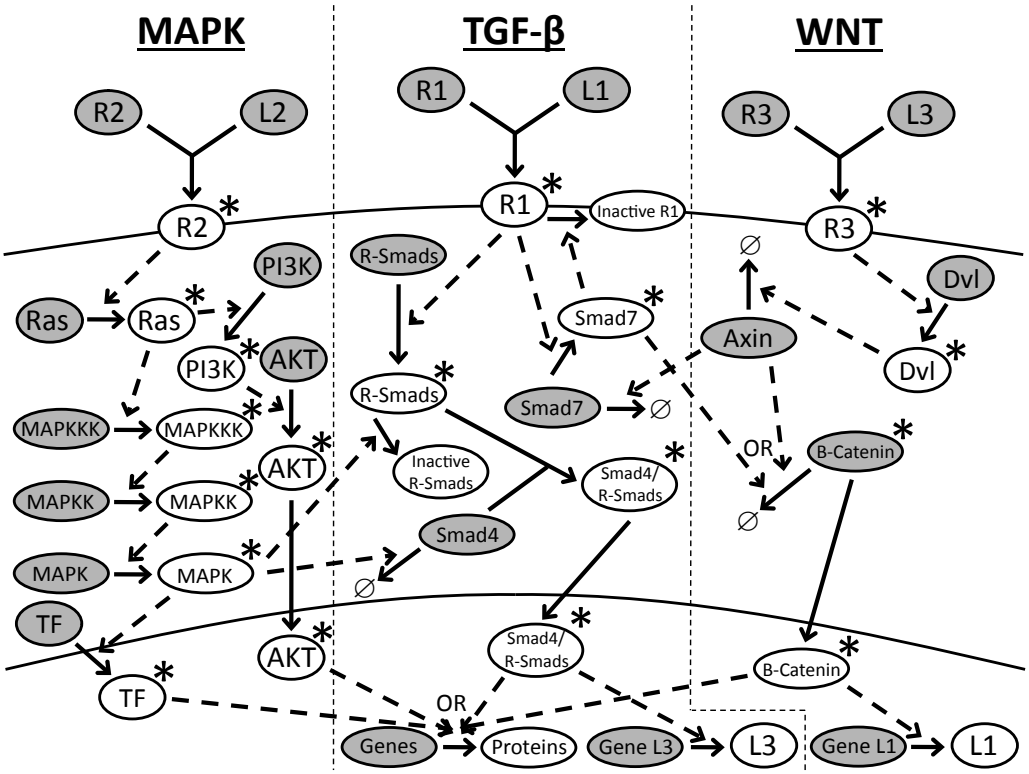


Figure 5.7: Our model of the cross-talk between the TGF-β, WNT and MAPK pathways.

5.11.3 Analysis of cross-talk

We detect the presence of 9 cross-talks in the full network using the approach outlined in Section 5.10—no new cross-talks are identified compared with the literature. We then characterise each cross-talk using the approach outlined in Section 5.9 as follows.

We measure the output of the TGF-β pathway by the activity of the expression of *Proteins* (a set of proteins in the TGF-β pathway). We use the following CSL properties to compare the effects of cross-talk: ψ_1, the eventual expression of Proteins, and ψ_2, the time-dependent expression of Proteins (within 5 time units).

$$\psi_1 = \mathbf{P}_{=?}\,[\,F\,(Proteins = 1)\,]$$

$$\psi_2 = \mathbf{P}_{=?}\,[\,F_{\leq 5}\,(Proteins = 1)\,]$$

We follow with a detailed analysis of each of the four networks.

Independent network In the independent network, $TGFB$, the activation of the TGF-β pathway leads to the expression of Proteins within 5 time units, ψ_2, with probability 0.47 and to the eventual expression of Proteins, ψ_1, with probability < 1 due to the inactivation of the receptor.

TGF-β and MAPK cross-talk In the TGF-β and MAPK network, $TGFB \,|[\ldots]|\, MAPK$, there are two types of cross-talk.

Signal flow: MAPK* proteins slow signal flow through the TGF-β pathway by deactivating the R-Smads and degrading Smad4.

Gene expression: the TF* and AKT* proteins upregulate gene expression in the TGF-β pathway.

The inclusion of cross-talk with the MAPK pathway can both provide alternative gene expression reactions and block signal flow through the TGF-β pathway, overall causing the probability of the expression of Proteins within 5 time units, ψ_2, to increase to 0.73. The probability of the eventual expression of Proteins, ψ_1, is 1 because there is an uninterrupted route to express proteins through the MAPK pathway.

TGF-β and WNT cross-talk In the TGF-β and MAPK network, $TGFB \,|[\ldots]|\, WNT$, there are three types of cross-talk.

Signal flow: the Smad7* protein degrades β-Catenin and the Axin protein degrades Smad7.

Gene expression: the β-Catenin protein upregulates gene expression in the TGF-β pathway.

Intracellular communication: the WNT pathway can cause the production of a ligand for the TGF-β pathway, and vice-versa.

The inclusion of cross-talk with the WNT pathway can both provide an alternative gene expression reaction and inhibit Smad7 which can inactivate the receptor for the TGF-β pathway. Overall this causes the probability of the expression of Proteins within 5 time units, ψ_2, to increase to 0.76. The probability of the eventual expression of Proteins, ψ_1, is < 1 due to the degradation of the β-Catenin protein.

TGF-β, WNT and MAPK cross-talk The TGF-β, WNT and MAPK network, $(TGFB \,|[\ldots]|\, MAPK) \,|[\ldots]|\, WNT$, is the union of the two cross-talk scenarios above. The effect of both WNT and MAPK cross-talk on the TGF-β pathway is additive—the probability of ψ_2 rises to 0.88, com-

pared with the single cross-talks of WNT and MAPK with probability 0.76 and 0.73 respectively. The inclusion of the MAPK cross-talk provides an uninterrupted route to express proteins and hence the probability of ψ_1 is 1.

We remark that we have categorised the complicated cross-talk in which Axin degrades Smad7 unambiguously as signal flow. Whereas, in [50] there is a suggestion that the cross-talk is receptor function because Axin degrades the receptor (via Smad7, an intermediate). Our approach does not classify this cross-talk as receptor function cross-talk.

5.12 Discussion

Reversible reactions We discuss two simplifications of the biochemistry that are used in our models. The first simplification is that we only consider irreversible reactions, e.g. the activation reaction $X \rightarrow X^*$. If our models were to include deactivation reactions, e.g. $X^* \rightarrow X$, then the temporal logic properties would need to be strengthened. For example, the property characterising signal flow cross-talk expresses that at some point in time R_1 is inactive and $Protein_1$ is expressed. If the activation of R_1 is a reversible reaction then this property is too weak. The property could be satisfied if R_1 becomes active, $Protein_1$ is expressed and then R_1 becomes inactive. Thus, the correct property with reversible reactions is:

$\mathbf{E}\,[(R_1^* = 0)\,U\,(R_1^* = 0 \wedge Protein_1 = 1)]$.

The enzyme-substrate complex The second simplification is that enzyme driven reactions are modelled without the intermediate complex of the enzyme bound to the substrate. This is a common abstraction used in models of biological systems. The activation of a protein X with an enzyme E is modelled as $X \rightarrow X^*$ if $E = 1$. We could include the enzyme-substrate complex by modelling the reaction as $X + E \rightarrow X/E \rightarrow X^* + E$. This would have an effect on our analysis. Consider the signal flow cross-talk example where X_2^* is also an enzyme for the activation of Y_1. If we included the enzyme-substrate complex then Y_1 and X_2^* would bind forming Y_1/X_2^*. This would slow the signal flow through pathway P_2 because enzyme X_2^* would be bound to Y_1 for a period of time and hence it would be unavailable to activate proteins in pathway P_2.

Cross-talk formalisation Our cross-talk formalisation depends on the set of modules being considered. One reason for this is that the modules act as a proxy for the cellular location. For example, in the definition for Gene Expression cross-talk, we disallow reactions from the Receptor

module because gene expression occurs in the nucleus which is 'far' from the receptor. Future work will be to introduce a formalisation that is not so strongly tied to current set of modules. We expect to need, at the very least, a mapping from the set of modules to the location of the modules.

Cross-talk generation and pathway generation Our method to generate all cross-talk models from a set of pathways (Section 5.8) can also be applied to generate all pathway models from a set of modules. However, to generate a pathway from a set of modules we need to ensure that all modules are connected, and sometimes connected together in a specific manner. Therefore, we require a constraint to our method: a set of reactions in each module that must synchronise with at least one reaction in another module.

Quantitative detail Recall the distinction between qualitative and quantitative models from Section 4.2.3. We have applied our approach to qualitative models, which have a low level of quantitative detail. As such, the probability values resulting from CSL model checking can only be used to compare the models with each other. Applying our approach to quantitative models would allow further interpretation of our analysis results. For example, the properties concerning the probability of time-dependent gene expression between cross-talk models would become a meaningful assessment of the strength of the cross-talk. However, this is left as future work.

Model-checking runtimes The state spaces for all the models presented here are small, of the order of 10^2. Runtimes for checking properties are therefore trivial.

Feature interaction One inspiration for the approach to pathway cross-talk presented here is work on using temporal logics to detect and characterise feature interaction in telecommunication networks [14]. A common problem is lack of universal definition of pathway/feature. In [90] the feature construct is introduced; the feature has an implementation δ and requirements ϕ, such that successful integration into base system P would be $P + \delta \models \phi$[8]. This parallels our approach of checking properties of different compositions of pathways (pathways with different instances of cross-talk).

Finally, we note that in telecommunications, 3-way feature interactions (a interaction between three features, that does not occur between only two features) are very rare: most detection algorithms depend on a pairwise analysis. This again parallels our approach in which we have not found an example of a single instance of cross-talk involving three pathways.

[8] the $\models$ symbol expresses that the system on the left-hand side satisfies the property on the right-hand side

Cross-talk categorisation An interesting question that also plagues the feature interaction community is what is a feature and a feature interaction? This is analogous to what is a pathway and a cross-talk, which begs the question, is our cross-talk categorisation complete? We believe this is *future* work for the biological, rather than the formal computer science, community.

The Molecular Nose project This work has been developed as part of the Molecular Nose project [80] that aims to develop new in vivo sensor technologies for analysing and interpreting signalling networks. The term "molecular nose" refers to sensor technology "sniffing out" pathways within a cell. Long term, we aim to generate hypotheses about the structure of pathways and networks, comparing those in normal cells with those in diseased cells.

5.13 Summary

In this chapter we have defined a framework for modular modelling of pathways and their cross-talk, based on generic modules and composition with synchronisation, variable sharing, and reaction renaming.

In Section 5.1 we gave the motivation for the framework. In Section 5.2 we explained how to build CTMCs with levels using the PRISM modelling language. In Section 5.3 we gave extensions to the PRISM modelling language required to model generic pathway modules that we used as the basis for our modelling approach. In Section 5.4 we introduced the modular modelling approach and showed how it can be used to build a pathway by composing generic pathway modules. In Section 5.5 we showed how, in a similar way, we can compose pathways to build a signalling network, in this case the pathways were composed independently. In Section 5.6 we introduced a new concept called auxiliary reactions, which were used when expressing cross-talk in the next section. Section 5.7 was the main contribution of this chapter, a formalisation and modelling approach for cross-talk based on synchronising reactions (including auxiliary reactions) and sharing variables between pathways. We defined five types of cross-talk and proved that they are well-defined. Although we could not prove completeness, we have not found an instance of cross-talk that did not belong to one of the five categories. In Section 5.8 we introduced an algorithm to enumerate all instances of cross-talk between two pathways. We applied this algorithm to two example pathways and were able to categorise each biologically meaningful instance of cross-talk. We then gave preliminary results on how to analyse models that do not contain a explicit notion of cross-talk. In Section 5.9 we showed how to detect cross-talk using CSL properties and in Section 5.10 we showed how to characterise each type of cross-talk using CTL properties. In Section 5.11

we demonstrated our framework with a case study of the TGF-β, WNT and MAPK pathways. In the case study we were able to detect, categorise and analyse the effect of the different instances of cross-talk in the network. Finally, in Section 5.12 we discussed our modelling assumptions and possible extensions of the framework.

We now turn our attention in the next chapter to the complementary problem, how to analyse models of signalling networks in which there is no structure.

Supplemental material An open-source Java application that implements the algorithm in Section 5.8 as well as the models used in this chapter can be found at
www.dcs.gla.ac.uk/~radonald/tcs2012/.

Chapter 6

Modelling unstructured signalling networks as signal flows

In this chapter we turn to the problem of how to analyse signalling network models in which there is no structure, i.e. there exists no explicit notion of signalling pathway and cross-talk.

In Section 6.1 we give the motivation for this chapter. In Section 6.2 we define signal flow in a signalling network. In Section 6.3 and Section 6.4 we explore the steady-state approach and current dynamic techniques (respectively) used to compute the set of signal flows in a signalling network model. In Section 6.5 we introduce a new algorithm called the Reaction Minimal Paths (RMP) algorithm to compute the set of signal flows in a model. In Section 6.6 we introduce the Pathway Logic modelling framework. In Section 6.7 we apply the RMP algorithm to signalling network models from the Pathway Logic modelling framework. We show how the set of signal flows and various network metrics computed using the RMP algorithm can be used to better understand the models. In Section 6.8 we show how signal flows can be clustered to reveal structure within signalling network models. In Section 6.9 we discuss the computational complexity and scalability of the RMP algorithm.

Background material We assume the following background material: Petri nets (Section 3.3), the dynamic behaviour of Petri nets (Section 3.4), and the steady-state behaviour of Petri nets (Section 3.5). Of particular importance are multisets (Appendix A), paths (Definition 5 on page 25) and state space searches (Definition 7 on page 27).

6.1 Motivation

In the previous chapter we introduced a framework for modelling signalling networks in a structured manner with an explicit notion of signalling pathway and cross-talk. We now turn our attention to the complementary problem, what do we do when a model does not have such structure? We call these models *unstructured models*. A particularly interesting modelling framework is Pathway Logic [103]. In Pathway Logic, models are automatically generated from a "knowledge base" of reactions—these models are unstructured and typically large, complex and difficult to understand.

Recall that a signal flow is the reactions starting from the cell detecting one or more ligands and ending in a change in some output of interest, e.g. gene expression, protein activation or a cellular response. Unstructured networks can be better understood by modelling them as a set of signal flows. To the best of our knowledge, no current technique for computing the set of signal flows in a model can guarantee both *completeness* and *correctness*, i.e. in some cases signal flows are missing or incorrect.

6.2 Signal flows

Signal flow in cellular signalling is a well-established concept in both the biological [36, 62, 66] and computational communities [30,57,83]. In the biological community signal flow is also called signal propagation [20] or signal transduction [49]. We now define signal flow.

Definition 22 (Signal flow)**.** *A signal flow s is a multiset of reactions that when fired from the initial state of a Petri net m_0 produce a set of outputs X, and all the reactions in the multiset are required to produce X—hence, the multiset is minimal. Usually the output is some measure of signalling pathway activity, e.g. gene expression, protein activation or a cellular response.*

We illustrate the term signal flow with the example below.

Example 7 Example of signal flow.

Consider the signalling pathway given in biological notation in Figure 6.1. One measure of signalling pathway activity is the expression of protein A. There are two signal flows to this output, one that uses reaction $r2$, $\{r1, r2, r4, r5\}$, and one that uses reaction $r3$, $\{r1, r3, r4, r5\}$.

Given a Petri net $\mathcal{M}$ and a set of outputs X, an algorithm SIG($\mathcal{M}, X$) computes the set of signal flows in a model if it returns a set S of signal flows that produce X.

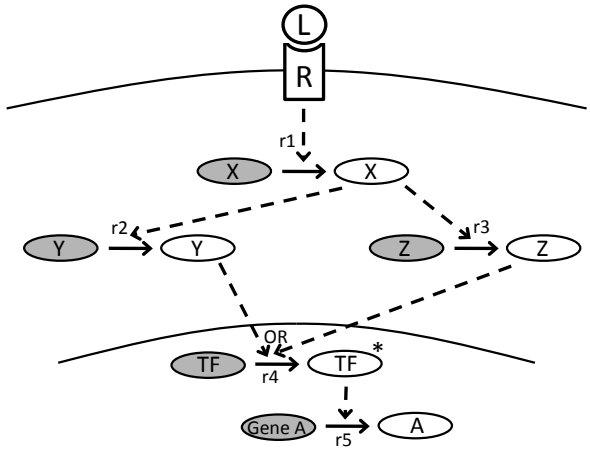


Figure 6.1: A signalling pathway ending with the expression of protein A. Each reaction is named so that it can be referenced in the signal flows.

Definition 23 (Completeness)**.** *An algorithm SIG*$(\mathcal{M}, X)$ *that returns a set of signal flows S is complete if for all possible signal flows s that produce X,* $s \in S$.

Definition 24 (Correctness)**.** *An algorithm SIG*$(\mathcal{M}, X)$ *that returns a set of signal flows S is correct if* $\forall s \in S$*, s is a signal flow that produces X.*

Algorithms that compute the set of signal flows in a model either analyse the steady-state behaviour or the dynamic behaviour. We now explore the shortcomings of current techniques, starting with the steady-state approach.

6.3 Demonstration of the steady-state approach using T invariant analysis

We demonstrate the shortcomings of the steady-state using T invariant analysis but we could equally use FBA, for example. We show that in the steady-state approach, completeness and correctness do not hold for models that contain certain network structures. We have found three network network structure patterns that cause the steady-state approach to produce incorrect results.

To compute T invariants that correspond to signal flows we first apply transformations to the Petri net. The purpose of these transformations is to force each T invariant to repeat the empty state (all places have no tokens) for example by introducing source and sink transitions. If a T invariant repeats the empty state then the T invariant contain all transitions to produce and consume the species required for the signal flow it represents—no species are assumed present.

We follow the approach taken by [56, 74] and apply the following three transformations (illustrated in Figure 6.2)

(1) T invariant analysis is not concerned with the initial state of a Petri net, only the net structure. To structurally encode the initial state, any place that is initially marked is given a source transition that can generate an infinite number of tokens on the place.

(2) To allow the possibility of repeating the empty state, places that are never consumed in a transition are given a sink transition that can consume an infinite number of tokens on the place.

(3) Again, to allow the possibility of repeating the empty state, places that are both pre- and post-places of a transition are changed to be only pre-places of the transition. Hence all bidirectional arcs are changed to unidirectional arcs from the place to the transition. In biological terms this means that all enzymes are consumed in the transitions.

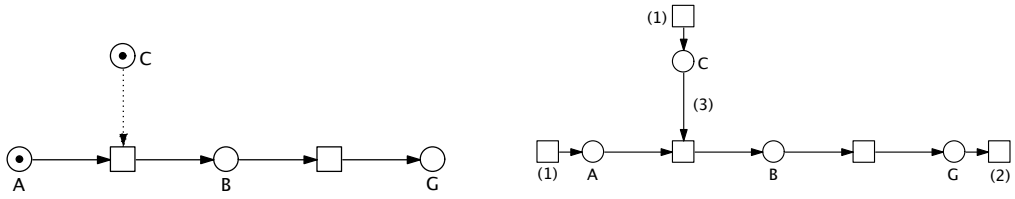


Figure 6.2: A Petri net before (left) and after (right) applying the transformations required to compute signal flows. Labels (1), (2) and (3) denote the transformation applied to the net. Recall that the dashed directed arc from place C to the transition represents a bidirectional arc and hence C is both a pre- and post-place of the transition in the Petri net on the left.

These transformations allow us to use T invariant analysis to compute signal flows in a model. The signal flows are computed to outputs (places with sinks): if we wish to designate a place as an output that was not given a sink through transformation (2), we can explicitly add a sink to this place.

An example of how T invariants relate to signal flows is shown in Figure 6.3.

In some cases the steady-state approach (such as T invariant analysis) computes signal flows very efficiently (discussed further in Section 6.3.5). However, steady-state analysis is not well-suited as both completeness and correctness are not guaranteed for all models. We have found three network structure patterns that cause incorrect results. We describe the patterns (place traps, consumption conflicts and protein degradations) below. It is also nontrivial to decide whether completeness or correctness will hold for a given model.

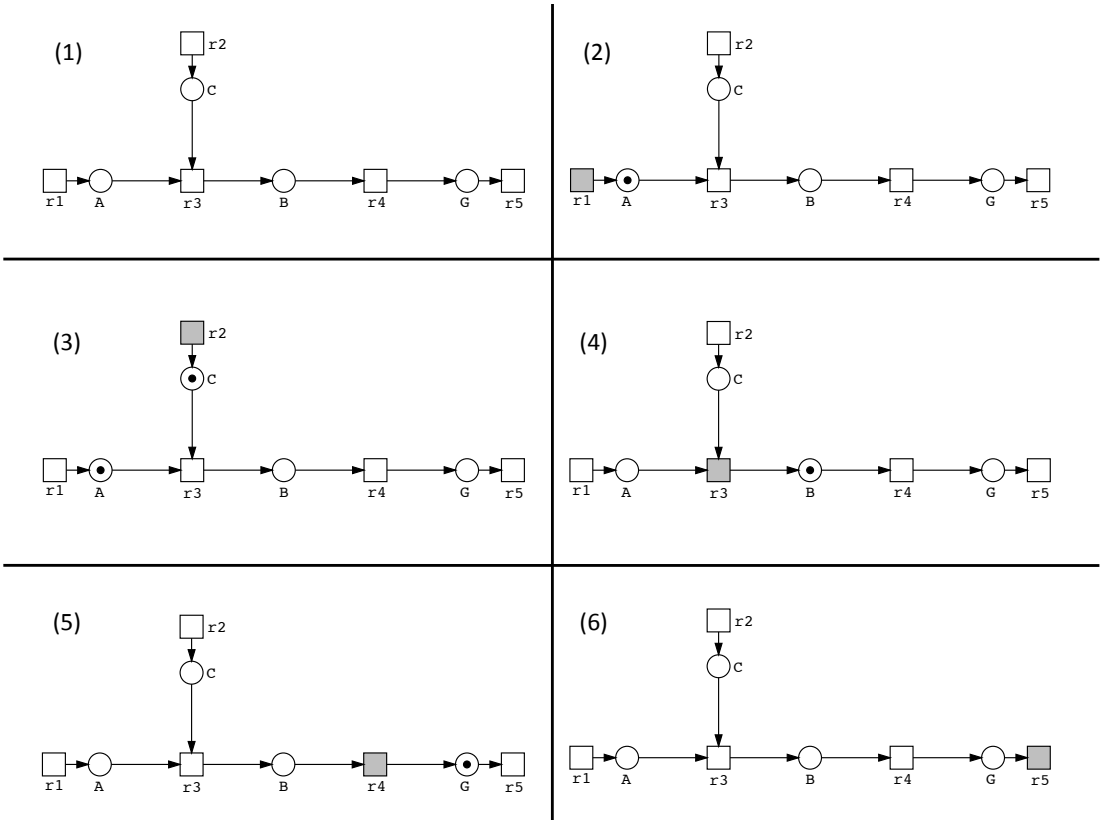


Figure 6.3: The signal flow from $\{A, C\}$ to $\{G\}$ is represented by the T invariant $\{r1, r2, r3, r4, r5\}$ as shown in steps (1) to (6) above. Notice that this T invariant repeats the empty state.

6.3.1 Place traps

A *place trap* is a set of places that once marked cannot become unmarked [55]. A set of places $Q \subseteq P$ is a place trap if $Q^{\bullet} \subseteq {}^{\bullet}Q$ (every transition that subtracts tokens from the place trap also puts tokens into the place set). Place traps are found in many models of biological systems, for example protein phosphorylation, protein ubiquitination and enzymes often involve place traps [106].

A model that contains a place trap cannot repeat an empty state because once the place trap is marked, it cannot become unmarked. Because of this, there can be no T invariant that places a token onto the place trap. However, a T invariant can include a place trap by repeating a state that is empty for all places except at least one place in the place trap—it assumes one of the species is present. This is an example where transformation (1), structurally encoding the initial state, is not followed.

Consider the problem of computing the signal flows from $\{A, C\}$ to $\{G\}$ in the Petri net in Figure 6.4. The Petri net has a single T invariant that starts with the place trap $\{B, D\}$. The T invariant repeats a state that is empty for all places except B. The T invariant is not an execution because it is not realisable—there is no possible execution of the T invariant.

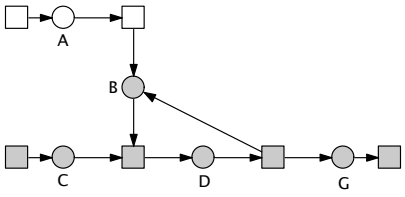


Figure 6.4: A Petri net containing a place trap on {B, D}. The single T invariant and all related places are highlighted in grey.

We note that the place trap in Figure 6.4 could be caused by an enzymatic reaction where B is the enzyme, C is the substrate, D is the enzyme-substrate complex and G is the product. The enzyme and enzyme-substrate complex is the place trap in this case.

Enzymes are often abstracted such that the enzyme-substrate complex is not modelled explicitly and hence we have a dashed arc from an enzyme to a transition. The enzyme is a place trap (with a single place) if there are no unidirectional outgoing arcs from the place. Transformation (3) removes place traps with a single place, however this can cause extra consumption conflicts as described below. Place traps with multiple places can be handled similarly by removing (by hand) an arc that will break the trap, as performed in [74]. However, we wish to avoid manual alteration of models for obvious reasons.

Steady-state analysis of models with place traps is likely to cause incorrect results.

6.3.2 Consumption conflicts

Consider the problem of computing the signal flows from $\{A, D\}$ to $\{G\}$ in the Petri net in Figure 6.5. There is no state that can be repeated by a T invariant. The production of E and F is coupled because E and F are produced by the same transition (perhaps a decomplexation or protein cleavage reaction). The number of tokens produced on E and F is always the same, however there is a difference in the number of transitions from E and F, 2 and 1 respectively. These transitions are required to produce a token on place G. We call this problem a *consumption conflict*—there is no true steady-state. Producing a token on G will always leave one more token on F than E, therefore it is never possible to repeat any state.

Note that arcs x, y and z in Figure 6.5 can be either unidirectional or bidirectional arcs (E and/or F can be an enzyme) because transformation (3) will convert all bidirectional arcs to unidirectional arcs. Hence, transformation (3) can cause extra consumption conflicts in a model.

Steady-state analysis of models with consumption conflicts is likely to cause incomplete results.

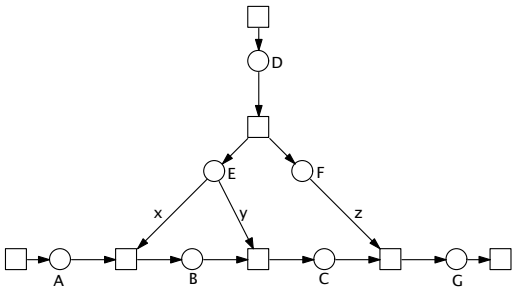


Figure 6.5: A Petri net containing a consumption conflict between places E and F. There are no T invariants in this Petri net because of the conflict.

6.3.3 Protein degradations

Consider the problem of computing the signal flows from $\{A, B\}$ to $\{G\}$ in the Petri net in Figure 6.6. The minimal sequence of transitions to produce G will leave a token on D. The token on D must be consumed because the empty state must be repeated. An extra two transitions must fire to consume the token on D, hence the T invariant contains extra transitions that are not part of the signal flow from $\{A, B\}$ to $\{G\}$. We call this problem a *protein degradation*.

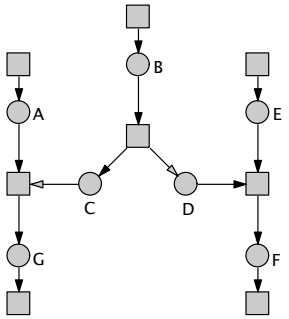


Figure 6.6: A Petri net with the single T invariant and all related places highlighted in grey. The T invariant does not correspond to a signal flow from A to G because transitions are included that are not required.

Steady-state analysis may produce incorrect results because all proteins must be degraded, therefore including extra transitions in the signal flows.

To summarise, the three network structure patterns above illustrate types of models where steady-state analysis is inappropriate to compute signal flows. We now discuss an alternative approach to T invariant analysis.

6.3.4 Alternative T invariant approach

In this section we investigate how the transformations might be altered to permit computing signal flows by T invariant analysis in more circumstances. An intuitive alternative approach is to use a different set of transformations in which we change transformation (3), the consumption of enzymes, to:

(3') All places that are both pre- and post-places ${}^\bullet t \cap t^\bullet$ of a transition t are given a sink transition that can consume an infinite number of tokens on the place.

Rather than consume the enzyme in the transition, we allow the enzyme to be consumed by a sink transition (as in Figure 6.7). This removes consumption conflicts caused by transformation (3). However this is not a complete fix because not all consumption conflicts are caused by transformation (3). Also, unrelated transitions may be included due to protein degradations, and place traps with multiple places may exist.

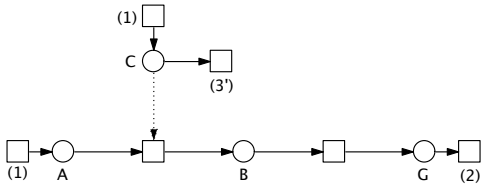


Figure 6.7: An alternative set of transformations applied to the Petri net from Figure 6.2. Note the sink transition added by transformation (3').

If the model contains no bidirectional arcs, there will be the same T invariants as before otherwise there will be a larger number of smaller T invariants. In this case we must compose T invariants to find signal flows, however composing T invariants is a nontrivial task (discussed briefly in [44]). Also note that composing T invariants to find signal flows is similar to the initial problem, composing transitions to find signal flows. The alternative approach is therefore not suitable. There may be other alternative approach (different sets of transformations) however we believe we have given strong evidence that the steady-state approach is inappropriate for computing signal flows in a signalling network.

6.3.5 Computational complexity

Before concluding this section, we comment on the computational complexity of the steady-state approach, again using T invariant analysis as the example. T invariants are computed by enumerating solutions to a linear algebra equation of the incidence matrix being in steady-state. There

are two approaches to enumerating the solutions: constraint programming [100] and an elimination algorithm such as Fourier-Motzkin elimination [21]. The complexity of these algorithms is exponential in the worst case, however the complexity in a given Petri net is difficult to characterise [100]. Note that transformation (3), consuming the enzyme, has the effect of increasing the complexity of the algorithm. Each time the enzyme is consumed it must be produced by firing at least one more transition.

6.3.6 Conclusion

The steady-state approach answers different questions than the dynamic approach. They are complementary methods to study models. The steady-state approach is concerned with "flows/firing rates" that maintain a given state whereas the dynamic approach is concerned with changing state and how information (local state change) propagates through a network, to control other processes. We have shown in this section how the steady-state approach can be used to compute the set of signal flows in a signalling network. However, we have also demonstrated that completeness and correctness are not guaranteed. We have demonstrated this approach using T invariants, but the shortcomings are experienced when using any steady-state technique.

6.4 Overview of current dynamic techniques

Dynamic techniques have been applied to compute the set of signal flows in models of cellular signalling systems. Current dynamic techniques can guarantee *correctness* however, to the best of our knowledge, no current technique also guarantees *completeness*. We discuss the LoLA and SPIN model checkers and stories below.

6.4.1 The LoLA model checker

The *Pathway Logic Assistant* (PLA) [103] allows the user to generate a single signal flow to a set of goal places. An individual signal flow is computed using the model checker within the *LoLA* (Low Level Analyzer) Petri net analysis tool [96] as described below.

Given a property ϕ, an state m is an *error state* if it violates ϕ. An error trace is a sequence of transitions $t_1, \ldots, t_n$ from the initial state m_0 to an error state such that $m_0 \rightarrow_{t_1} m_1 \ldots \rightarrow_{t_n} m_n$ where m_n is an error state and no states in $m_1, \ldots, m_{n-1}$ are error states. To compute error traces that are sequences of transitions to mark a set of goal places $G = \{g_1, \ldots, g_v\}$, we use a temporal logic property that asserts that the places in G cannot be marked. In LTL the property is:

$\phi = \neg\ F\ (g_1 \geq 1 \wedge\ \ldots\ \wedge g_v \geq 1)$

An error state for ϕ is a state m where $m(g_1) \geq 1 \wedge\ \ldots\ \wedge m(g_v) \geq 1$.

LoLA, using stubborn set reduction [69], can efficiently answer reachability queries such as ϕ. If ϕ is false, i.e. it is possible to mark the places in G, then LoLA will return an error trace for ϕ. The error trace is not guaranteed to be minimal. Transitions in the error trace that are not required to reach the goals are removed automatically using the relevant subnet algorithm (Appendix D), however there is no proof of correctness. Enforcing the minimality property on the traces using the relevant subnet algorithm is discussed further in Section 7.2.3 where it is shown to be insufficient in some models. LoLA returns only one error trace for each property ϕ—subsequent traces can be found by manually removing transitions in from the network, however there can be no guarantee that all traces are generated. Clearly this technique is insufficient for generating all signal flows in a model because correctness does not always hold and completeness is not guaranteed.

6.4.2 The SPIN model checker

The *SPIN model checker* [61] can return all *state* error traces in a model for a property ϕ. A state error trace is a sequence of states $m_0 \rightarrow m_1\ \ldots\ \rightarrow m_n$ where m_n is an error state and no state in $m_0,\ \ldots,\ m_{n-1}$ is an error state, and for each pair of states m_{i-1} and m_i, there exists at least one t_i such that $m_{i-1} \rightarrow_{t_i} m_i$. Note that only the sequence of states are given in the SPIN output, and not the transitions. We map a state error trace to the set of error traces that can generate it. SPIN permits breadth-first search of the state space, which produces *minimal length* error traces. One may expect minimal length error traces to equate to signal flows, however this is not the case.

Consider the Petri net in Example 8 on page 91. The error states in the model are BXG, AYG and BYG. The minimal length error traces are as follows. Trace $(r1, r3)$ for state BXG. Trace $(r2, r4)$ for state AYG. Traces $(r1, r2, r3)$ and $(r1, r2, r4)$ for state BYG. Because minimal length error traces are produced for *all* error states, there is no guarantee that all transitions are required to mark the set of goal places. Error traces $(r1, r2, r3)$ and $(r1, r2, r4)$ contain a redundant transition $r2$ and $r1$ respectively.

Furthermore, consider the Petri net in Example 9 on page 91. There is one error state labelled G and the minimal length error trace is $(r1)$. Because only the minimal length error trace is returned for each error state, the algorithm misses an error trace to state G that is a signal flow, $(r2, r3)$.

In general, minimal length error traces do not equate to signal flows.

6.4.3 Stories

Stories [30] in a rule-based language capture the events that are required to reach an event of interest. Stories are equivalent to signal flows. A story is a sequence of events that; starting from the initial state, reaches an event of interest called the *observable*; consists only of events that are required to reach the observable; and, contains no event subsequence that has the same property. The authors however do not discuss in detail the method used to compute stories. Stories are generated from paths through the state space (stochastic simulations) [31]. The "story sampler" converts a stochastic simulation into a story—however, no proof of correctness of the story sampler is given. To compute subsequent stories, more stochastic simulations are required, but with this process there can be no guarantee that all stories are generated. Only by exploring the state space, rather than paths through the state space in isolation, can we be guaranteed to find all stories.

6.5 A new dynamic technique: the Reaction Minimal Paths algorithm

In this section we introduce the major contribution of this chapter, the *Reaction Minimal Paths (RMP) algorithm*. This algorithm is a dynamic technique to computing the set of signal flows in a model, and is the first to guarantee both completeness and correctness. The algorithm works by exploring the state space of a model and computing reaction minimal paths. We are interested in reaction minimal paths that produce the goal from the initial state of a model without using certain places (avoid). We now give the formal definition of a reaction minimal path.

An avoid set A is a set of places $A \subseteq P$ to be avoided. A transition t satisfies the avoid constraint, written $t \models A$, if the transition does not have a pre- or post-place in the avoid set, $(\bullet t \cup t\bullet) \cap A = \emptyset$. A path $\overline{R}$ from m to m' satisfies the avoid constraint, written $\overline{R} \vdash_A m \rightsquigarrow m'$, if $\forall t \in \overline{R} \,.\, t \models A$.

A goal set G is a set of places $G \subseteq P$ that we wish to have marked. A state m satisfies the goal constraint, written $m \models G$, if $\forall g \in G \,.\, m(g) \geq 1$. States that satisfy the goal constraint are shown graphically as an oval with a dashed line. A path $\overline{R}$ from m to m' satisfies the goal constraint, written $\overline{R} \vdash^G m \rightsquigarrow m'$, if $m' \models G$.

Definition 25 (Goal/avoid path)**.** *A goal/avoid path is a path from the initial state in a model satisfying both the goal and the avoid constraints, written* $\overline{R} \vdash_A^G m_0 \rightsquigarrow m'$. *Clearly no path can have a place as both a goal and an avoid. Thus we require that the sets of goals and avoids are disjoint,* $G \cap A = \emptyset$.

Definition 26 (Reaction minimal path)**.** *Given a goal set G and avoid set A, a goal/avoid path* $\overline{R} \vdash_A^G m_0 \rightsquigarrow m$ *is called a reaction minimal path (RMP) if there is no goal/avoid path* $\overline{R'} \vdash_A^G m_0 \rightsquigarrow m'$ *that is a proper submultiset,* $\overline{R'} \subset \overline{R}$*.*

An RMP is a signal flow because it marks a set of outputs and all transitions in the RMP are required to mark the set of outputs.

Below, Theorem 2 shows that the multiset semantics used in paths is sufficient to describe executions. In the algorithm that follows, we are not concerned with the order the transitions fired in to reach a state.

Theorem 2 (Relationship between executions and paths)**.** *All executions R of a path*[1] $\overline{R}$ *starting at m reach the same final state.*

Proof. An execution R of $\overline{R}$ starting at m reaches m' where $\forall p \in P \,.\, m'(p) = m(p) + \sum_{t \in R} f(t,p) - \sum_{t \in R} f(p,t)$. m' is independent of the order of transitions in R because $\sum_{t \in R} f(t,p)$ and $\sum_{t \in R} f(p,t)$ are independent of the order of the transitions in R. Because each R of $\overline{R}$ is an ordering of transitions, and the state reached by R is independent of the order of the transitions, all R of $\overline{R}$ reach m'. □

6.5.1 Algorithm

We present an algorithm for computing all reaction minimal paths in a *k-bounded* Petri net $\mathcal{M} = (T, P, f, m_0)$.

The set of reaction minimal paths from m_0 reaching G without using A can be found by generating (state, path) tuples. The tuples are generated in stages following a breadth-first search of the state space such that Stage(n) contains tuples $(m, \overline{R})$ where $|\overline{R}| = n$ and $\overline{R} \vdash m_0 \rightsquigarrow m$.

Note that two (or more) distinct executions R and R' may equate to the same path $\overline{R}$. Therefore we may encounter the tuple $(m, \overline{R})$ multiple times by following different executions of the same path. We ignore duplicate tuples as these represent different orderings of the same multiset of transitions.

Definition 27 (Goal/avoid subsumption)**.** *A tuple* $(m, \overline{R})$ *is subsumed to a goal set G by* $(m', \overline{R'})$ *if* $\overline{R'}$ *is a proper submultiset of* $\overline{R}$*,* $\overline{R'} \subset \overline{R}$*, and* $m' = m$ *or* $m' \models G$*.*

Definition 28 (Goal/avoid stages)**.** *Suppose we have a goal set G and avoid set A. Stage(0) contains one tuple, the initial state and the empty path* $(m_0, \emptyset)$*. Stage(n) for* $n \geq 1$ *contains all tuples*

[1] recall that an execution of a path was informally defined on Page 25

$(m, \overline{R})$ with $\overline{R} \vdash m_0 \rightsquigarrow m$ and $|\overline{R}| = n$ such that $(m, \overline{R})$ is not subsumed by a member of Stage(j) for $j < n$. This ensures that all paths that satisfy the goal and avoid constraint are reaction minimal paths.

We use a breadth-first search because checking whether $(m, \overline{R})$ in Stage(n) is subsumed by some $(m', \overline{R'})$ requires checking $(m', \overline{R'})$ in Stage(j), $j < n$. Hence, $\overline{R'} \subset \overline{R}$ requires $|\overline{R'}| < |\overline{R}|$.

The algorithm to compute all reaction minimal paths to G avoiding A in a k-bounded Petri net $\mathcal{M}$ is RMP$(\mathcal{M}, G, A)$ as follows.

Pre-processs: To make all paths satisfy the avoid constraint, we remove any transition that has a pre- or post-place in the avoid set: $T^* = \{t \in T \mid (\bullet t \cup t\bullet) \cap A = \emptyset\}$.

Stage 0:

Stage(0) = $\{(m_0, \emptyset)\}$
Paths = $\emptyset$

Stage n: Given Stage(n-1)

Stage(n) = $\emptyset$
for $(m, \overline{R}) \in$ Stage(n-1) **do**
  if $m \models G$ **then**
    Add $\overline{R}$ to *Paths*
  else
    for $t_i \in T^*$ such that $m \rightarrow_{t_i} m'$ **do**
      $\overline{R'} = Add(\overline{R}, t_i)$
      Add $(m', \overline{R'})$ to Stage(n) if it is not subsumed by a tuple in Stage(j) for some $j < n$
    end for
  end if
end for
if Stage(n) == $\emptyset$ **then**
  Return *Paths*
end if

Theorem 3 (Termination)**.** *For any k-bounded Petri net there exists an $n \geq 1$ such that Stage(n) is empty.*

Proof. The set of possible states in a k-bounded Petri net is finite. The set of paths to each state such that there is no proper submultiset that reaches the same state is finite because the set of transitions is finite. Therefore, the set of (state, path) tuples is finite and hence there must be some n such that Stage(n) is empty. □

Theorem 4 (Completeness)**.** *For a given G and A if there exists a reaction minimal path $\overline{R} \vdash_A^G m_0 \rightsquigarrow m$ then $(m, \overline{R})$ is in Stage(n) where $n = |\overline{R}|$.*

Proof. Definition 28 ensures that all paths found in the stages that satisfy the goal and avoid constraints are reaction minimal paths. The set of stages contains all states except those reachable

only from a goal/avoid path that is not a reaction minimal path. Therefore if there exists a reaction minimal path $\overline{R} \vdash_A^G m_0 \rightsquigarrow m$ then $(m, \overline{R})$ is in Stage(n) where $n = |\overline{R}|$. □

Theorem 5 (Correctness)**.** *For any k-bounded Petri net, all reaction minimal paths to a goal set avoiding an avoid set are found by generating the set of stages.*

Proof. Multiset semantics are sufficient to describe an execution (Theorem 2). The set of stages is finite for any k-bounded Petri net (Theorem 3). If there exists a reaction minimal path then it is in some Stage(n) (Theorem 4). Therefore, all reaction minimal paths to a goal set avoiding an avoid set are found by generating the set of stages. □

Algorithm correctness In the algorithm above, paths are multisets, stages are generated as per Definition 28 and the set of stages is finite for any k-bounded Petri net. Therefore, Theorem 5 holds for this algorithm.

Relevant subnet optimisation To optimise the computation we can apply the relevant subnet algorithm with respect to G and A, Subnet(T, m_0, G, A). This function removes any transition that has a pre- or post-place in the avoid set or does not contribute to reaching the goal set. This algorithm was introduced in [103] where it is shown that the resulting network contains all reaction minimal paths for a given G and A. The algorithm to compute the relevant subnet is given in Appendix D.

Computational complexity The computational complexity of the algorithm is the number of (state, path) tuples. In the worst case of no reaction minimal paths, the number of tuples generated is at least the number of states in the state space and possibly greater than the number of states if multiple paths reach the same state.

Approximation An *unbounded Petri net* is a Petri net where there does not exist a k such that all places have at most k tokens in any reachable state. An unbounded Petri net has an infinite set of reachable states and therefore the algorithm in the previous section *may* not terminate. To obtain approximate results for unbounded Petri nets (or Petri nets with very large state spaces), we can compute the set of reaction minimal paths with an upper bound on the number of stages used, n. Hence the algorithm will terminate after Stage(n) and the paths will have a maximum cardinality (length) of n. Note that this is the same approach taken by bounded model checking.

6.5.2 Examples

Below we give five examples to illustrate different scenarios that the RMP algorithm may encounter.

Example 8 Simple example.

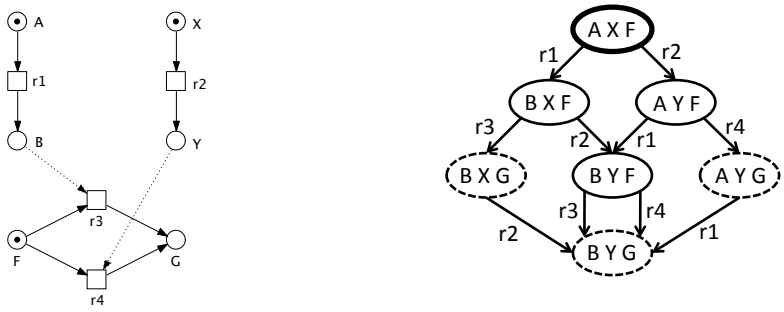


Figure 6.8: An example Petri net (left) and related state space (right). States that satisfy the goal constraint are indicated by a dashed line.

Consider the Petri net in Figure 6.8 with a goal set $\{G\}$ and avoid set $\emptyset$. The algorithm produces the 3 stages below.

Stage 0: $(\text{AXF}, \emptyset)$

Stage 1: $(\text{BXF}, \{r1\})$ $(\text{AYF}, \{r2\})$

Stage 2: $(\text{BXG}, \{r1, r3\})$ $(\text{BYF}, \{r1, r2\})$ $(\text{AYG}, \{r2, r4\})$

The set of reaction minimal paths is: $\{\{r1, r3\}, \{r2, r4\}\}$

Stage 3 is empty because all (state, path) tuples are subsumed by some tuple in a previous stage. $(\text{BYG}, \{r1, r2, r3\})$ is subsumed by $(\text{BXG}, \{r1, r3\})$ because BXG satisfies the goal constraint. Likewise, $(\text{BYG}, \{r1, r2, r4\})$ is subsumed by $(\text{AYG}, \{r2, r4\})$ because AYG satisfies the goal constraint.

Example 9 Difference between minimal length and reaction minimal paths.

Consider the Petri net in Figure 6.9 with a goal set $\{G\}$ and avoid set $\emptyset$. The algorithm produces the 3 stages below.

Stage 0: $(\text{A}, \emptyset)$

Stage 1: $(\text{G}, \{r1\})$ $(\text{B}, \{r2\})$

Stage 2: $(\text{G}, \{r2, r3\})$

The set of reaction minimal paths is: $\{\{r1\}, \{r2, r3\}\}$

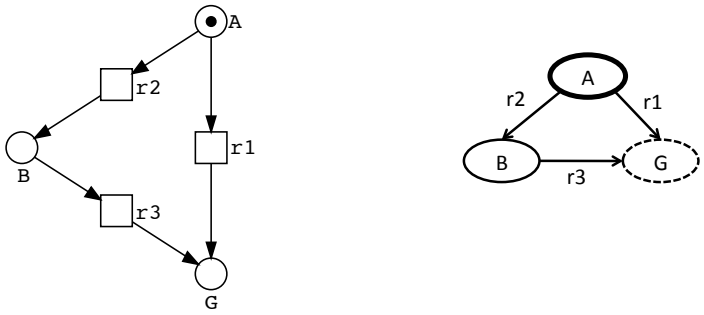


Figure 6.9: An example Petri net (left) and related state space (right). States that satisfy the goal constraint are indicated by a dashed line. Notice that there is one minimal length path, $\{r1\}$, and two reaction minimal paths, $\{r1\}$ and $\{r2, r3\}$.

Although $\{r2, r3\}$ is a longer path to state G than $\{r1\}$, it is distinct and all transitions are required to reach G, therefore it is reaction minimal.

Example 10 Rationale for multiset semantics.

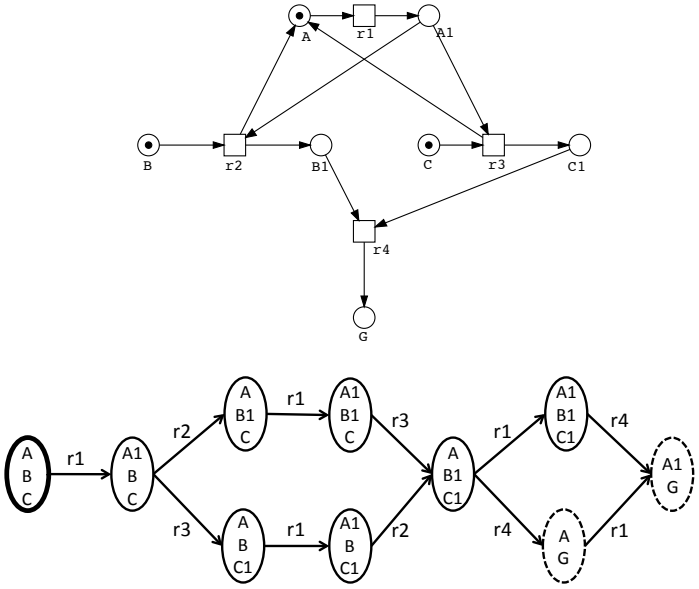


Figure 6.10: An example Petri net (top) and related state space (bottom). States that satisfy the goal constraint are indicated by a dashed line.

Consider the Petri net in Figure 6.10 with a goal set $\{G\}$ and avoid set $\emptyset$. The algorithm produces the 7 stages below.

Stage 0: (A B C, $\emptyset$)

Stage 1: (A1 B C, $\{r1\}$)

Stage 2: (A B1 C, $\{r1, r2\}$) (A B C1, $\{r1, r3\}$)

Stage 3: (A1 B1 C, $\{2 * r1, r2\}$) (A1 B C1, $\{2 * r1, r3\}$)

Stage 4: (A B1 C1, $\{2 * r1, r2, r3\}$)

Stage 5: (A1 B1 C1, $\{3 * r1, r2, r3\}$) (A G, $\{2 * r1, r2, r3, r4\}$)

Stage 6: (A1 G, $\{3 * r1, r2, r3, r4\}$)

The set of reaction minimal paths is: $\{\{2 * r1, r2, r3, r4\}\}$

Even though this is a 1-bounded Petri net, the transition $r1$ must fire more than once to reach $\{G\}$. This is the rationale for using multiset (rather than set) representations of executions even in models with only presence/absence of biochemical species.

Example 11 Demonstration of termination.

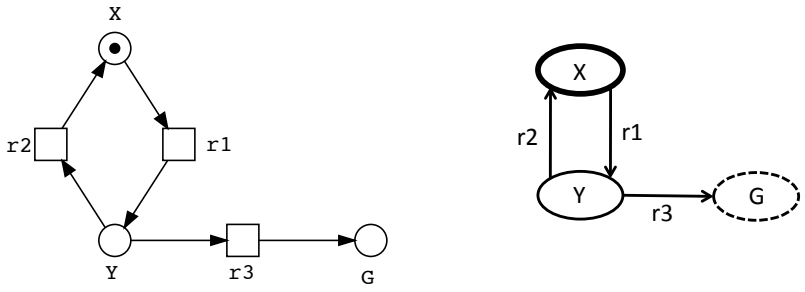


Figure 6.11: An example Petri net (left) and related state space (right). States that satisfy the goal constraint are indicated by a dashed line.

Consider the Petri net of a loop in Figure 6.11 with a goal set $\{G\}$ and avoid set $\emptyset$. The algorithm produces the 3 stages below.

Stage 0: (X, $\emptyset$)

Stage 1: (Y, $\{r1\}$)

Stage 2: (G, $\{r1, r3\}$)

The set of reaction minimal paths is: $\{\{r1, r3\}\}$

The algorithm terminates even with a loop because the tuple (X, $\{r1, r2\}$) is subsumed by (X, $\emptyset$) in Stage 0, therefore the exploration of the loop stops.

Example 12 Comparison to T invariants.

Consider the Petri net in Figure 6.12. Transition r3 requires both X and Y to be marked however it is only possible to mark one of X or Y. In this case the RMP algorithm will terminate after exploring all (state, path) tuples, returning an empty set of reaction minimal paths.

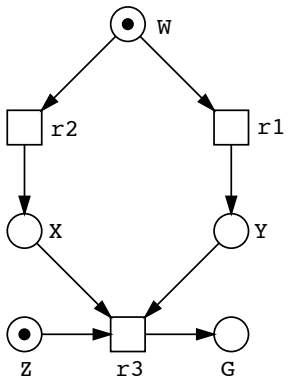


Figure 6.12: An example 1-bounded Petri net with a transition r3 that is always disabled.

T invariant analysis will find a signal flow in this model because there are source transitions for the initially marked places. Places W and Z will be given source transitions, say rW and rZ, such that an infinite number of tokens can be generated for each place. This approach will find a signal flow to $\{G\}$, $\{2 * rW, rZ, r1, r2, r3\}$. The application of T invariant analysis is well-suited in this case because signal flows involve a flow of many molecules rather than restricted to single molecules.

6.6 Pathway Logic

Pathway Logic (PL) [103] is a framework for modelling biological systems based on rewriting logic [28, 78]. The Pathway Logic framework can be thought of as comprising three components: curation, knowledge base and models.

Curation Experimental data about the cellular signalling response to a variety of stimuli is *curated* from results reported in biological papers. Curation is a lengthy, manual process and as such it is not possible to provide consistent, detailed coverage of all systems involved in signalling. For example, there has been more curation on the reactions initiated by Egf stimulation than those initiated by Ngf stimulation because the curator, with limited resources, has deemed Egf more interesting.

A *cell line* is a type of cell that is used for biological research, e.g. the PC12 cell line is from a tumour in a rat adrenal gland. A sample of a cell line is produced by growing the cells under controlled conditions. Crucially, a sample of a cell line can be traced back to a single source, therefore experiments on different samples of the same cell line are comparable, even between labs in different locations. Data in the Pathway Logic project is curated from any cell line.

The experimental data that is curated is stored as *datums*, a formal representation of biological results.

Knowledge base From these datums *reactions* are manually written as rewrite rules; the datums form the *evidence* for each reaction.

Rewrite rules are of the form current_state → new_state where current_state and new_state are constraints on the states that the rule applies to and reaches respectively. If a biochemical species is in the current_state then it is required for the rewrite rule to execute. If a biochemical species is in the new_state then it is present in the state that is reached when the rewrite rule executes. If a biochemical species is in the current_state and new_state then it is required but unchanged in the reaction, hence it is an enzyme.

The *knowledge base* is a database of reactions written as rewrite rules.

At the time of writing there are two versions of the knowledge base in use, *version 5* and *version 6* (also written kb v5 and kb v6 respectively). Version 6 extends version 5 with further curation, however version 5 still stands as a useful knowledge base.

Model Executable Petri nets are automatically constructed using the knowledge base.

An initial state (the biochemical species that are initially present) is specified by the user. The forward collection algorithm (part of the relevant subnet algorithm in Appendix D) is used to collect all rewrite rules (reactions) in the knowledge base that can fire from any reachable state.[2] The rewrite rules then become the transitions, and the biochemical species that are referenced by the rules are the places. There is an arc from a place to a transition if the species is in the current_state of the rewrite rule, and there is an arc from a transition to a place if the species is in the new_state of the rewrite rule. Enzymes are in both the current_state and new_state of the rule, so the enzyme place has a directed arc from and to the transition. Recall that we use the shortcut of a dashed arc from the place to the transition for enzymes. For any species in the initial state, the corresponding place is marked.

The rewrite rules used in Pathway Logic express only presence of biochemical species. Because the rules have been written such that there is never more than one "copy" of each biochemical species, the resulting Petri nets are guaranteed to be 1-bounded.

To illustrate the difference in size between version 5 and version 6 of the knowledge base, we build a model of a cell with the initial state containing all ligands in the knowledge base. In version

[2] Checking whether a reaction can fire is approximated, therefore a superset of the reactions that can fire is generated.

5 the result is a model with 8 ligands, 353 biochemical species and 235 reactions and in version 6 the result is a model with 11 ligands, 664 biochemical species and 542 reactions.

The curation step involves writing reactions using datums from any cell line. Some reactions may occur only in cancerous cell lines, whereas other reactions may be critical in all cell types in an organism. Pathway Logic models contain all reactions from the knowledge base (that are fireable from the initial state) and so they are an over approximation of the behaviour of any particular cell. This is a particularly interesting feature of Pathway Logic because, for the first time, experimental results from many cell lines are brought together to build an executable model.

The Pathway Logic framework is also interesting because the approach to building models is more automated than the more common, manual approach. The manual approach to building models uses experimental data to build a biological cartoon of the whole signalling pathway/network. The cartoon is then transformed into an executable model. This process can be error prone especially with complex models such as signalling networks.

Pathway Logic on the other hand builds individual reactions in isolation, depositing them into a knowledge base from which models are automatically constructed. The curator has only to focus on one reaction at a time rather than the whole model in the manual approach. This allows *larger* and *more complex* models to be built, however these models can be more difficult to understand. Unlike the signalling network models that are created manually in the previous chapter, Pathway Logic models have no notion of pathway or cross-talk—they are *unstructured*.

Example 13 Example of the Pathway Logic framework.

Consider the example of a 3-stage cascade that is initiated in response to a ligand L.

Curation Suppose we have curated the following set of datums. We give the datums in a shorthand form as we wish to focus on the modelling approach rather than the details of curation and experimental data.

0. With no ligands, proteins X, Y and Z are inactive.
1. Given ligand L, protein X is active.
2. Given ligand L, protein Y is active.
3. Given ligand L, protein Z is active.
4. Given ligand L and protein X knocked-out, protein Y is inactive.

5. Given ligand L and protein X knocked-out, protein Z is inactive.

6. Given ligand L and protein Y knocked-out, protein Z is inactive.

Knowledge base Reactions are built from the datums as follows. There is clearly a reaction that causes Y to become active (comparing datums 0 and 2). The biochemical species that must be present in order for Y to become active are L (datums 2) and X (datum 4). We create a reaction for the activation of Y with L and X as enzymes

A knowledge base (below) is built containing the following reactions built using datums 0 ... 6.

Reaction name	Rewrite rule	Evidence (datums)
r1	$(X\ L) \Rightarrow (XActive\ L)$	0, 1
r2	$(Y\ L\ XActive) \Rightarrow$ $(YActive\ L\ XActive)$	0, 2, 4
r3	$(Z\ L\ XActive\ YActive) \Rightarrow$ $(ZActive\ L\ XActive\ YActive)$	0, 3, 5, 6

Model Given the initial state with X, Y, Z and L, a Petri net model of the 3-stage cascade is automatically constructed from the knowledge base. This model is given in Figure 6.13

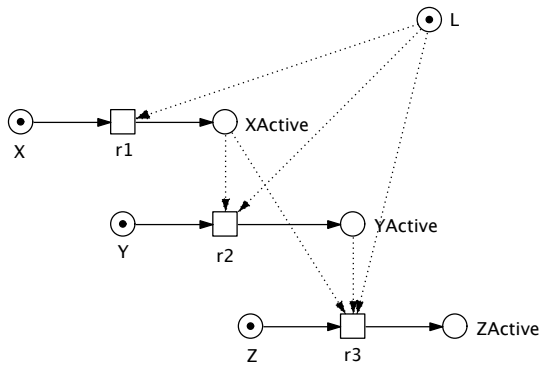


Figure 6.13: The Petri net model of the 3-stage cascade that was automatically constructed from the knowledge base.

6.7 Results

We have used the RMP algorithm to compute the reaction minimal paths (signal flows) in two Petri net models generated from the Pathway Logic knowledge base of cellular signalling response. The

models are the signalling events for the *activation of ERKs* and the *activation of RelA*. We have also used the reaction minimal paths to compute four metrics for characterising network behaviour. We show how the reaction minimal paths and metrics allow us to better understand the Pathway Logic models.

6.7.1 Pathway Logic models

Recall from Section 6.6 that at the time of writing there are two versions of the Pathway Logic knowledge base in use: version 5 and version 6. We apply the RMP algorithm to version 5 of the knowledge base; in the next chapter we show how we can adapt the RMP algorithm to be applicable to the more computationally intensive version 6.

We generate a model from version 5 of the Pathway Logic knowledge base with 11 ligands in the initial state. From this model we have generated the relevant subnets (Appendix D) for the activation of two proteins of interest, ERKs and RelA, and analyse the set of reaction minimal paths, summarised in Table 6.1.

Diagrams of the Pathway Logic models are given in Appendix E.

For the ERK activation model the initial event is Egf (Epidermal Growth Factor) binding to its receptor, EgfR, and the goal is activation of ERK1 and ERK2 (ERKs) in the EgfR complex (EgfRC), therefore G = {Erks–act–EgfRC}. We first generated the relevant subnet for G using the Pathway Logic Assistant. The relevant subnet for this goal contains none of the problematic network structures discussed in Section 6.3, and in this case, the set of T invariants corresponds exactly to the set of signal flows activating ERKs. A further Petri net transformation was required for the algorithm to compute the set of T invariants to terminate. It is interesting to note that T invariant analysis had a significantly shorter execution time (less than 1s) compared with the RMP algorithm.

For the RelA activation model there are two potential stimuli IL1 (Interleukin 1) and Tnf (Tumor Necrosis Factor) and the goal is activation of RelA in the nucleus, therefore G = {Rela–act–Nuc}. The relevant subnet for this goal contains several place traps due to the ubiquitination reactions, for example the E2 ubiquitin ligase Traf5 causes phosphorylated Irak1 to become ubiquitinated. The standard set of transformations required for T invariant analysis results in consumption conflicts, and therefore no T invariants are found. We have also applied the alternative T invariant approach outlined in Section 6.3.4. This resulted in a large number of small T invariants which did not fully cover the relevant subnet, hence there is no possibility to connect the T invariants to find signal flows. Clearly the T invariant approach is insufficient for this model.

	Erks–act–EgfRC kb v5	**Rela–act–Nuc kb v5**
Places	54	92
Transitions	38	57
Reachable State	149,014	95,096
Number of tuples	618,861	171,237
Number of stages	24	34
Runtime	50s	9s
Reaction Minimal Paths	144	39
T Invariants	144	0

Table 6.1: The result of the RMP algorithm for G = {Erks–act–EgfRC} and G = {Rela–act–Nuc} contrasted with T invariant analysis. Number of tuples is the number of (state, path) tuples. Runtime is on a workstation with a 2.53GHz dual core processor with 4GB of memory.

We now proceed with an analysis of what reaction minimal paths tell us about these models. We compute four metrics for characterising network behaviour: essential transitions, used places, knockouts and multi-signal cellular responses. In the following we assume a Petri net $\mathcal{M} = (T, P, f, m_0)$, a set of goals G and a set of avoids A. Note that we parameterise our metrics by $[\mathcal{M}, G, A]$. Let RMP be the set of reaction minimal paths computed using RMP$(\mathcal{M}, G, A)$.

6.7.2 Essential transitions

A transition t is essential, written $ess[\mathcal{M}, G, A](t)$, if there is no goal/avoid path $\overline{R}$ from m_0 using only transitions in $(T - t)$. We can compute the set of essential transitions using the set of reaction minimal paths RMP as follows.

$$ess[\mathcal{M}, G, A] = \{t \mid \forall \overline{R} \in RMP \,.\, t \in \overline{R}\}$$

More generally, a set of transitions T' is essential, $ess[\mathcal{M}, G, A](T')$, if every path satisfying G and A contains a member of T', and no proper subset of T' has this property. This can be checked using the set of reaction minimal paths for G and A as follows.

$$\begin{aligned} ess[\mathcal{M}, G, A](T') \Leftarrow \\ &(\forall \overline{R} \in RMP \,.\, \exists t \in T' \,.\, t \in \overline{R}) \wedge \\ &(\forall T'' \subset T' \,.\, \exists \overline{R} \in RMP \,.\, (T'' \cap \overline{R} = \emptyset)) \end{aligned}$$

6.7.3 Used places

A path $\overline{R} \vdash_A^G m_0 \rightsquigarrow m'$ uses a place p, $uses[\mathcal{M}, G, A](\overline{R}, p)$, if there is no path $\overline{R''} \vdash_A^G m_0 \rightsquigarrow m'$ where $\overline{R''} \subseteq \overline{R'}$ and $\overline{R'}$ is the result of removing from $\overline{R}$ any transition t with p as a pre-place, $p \cap {}^\bullet t \neq \emptyset$. This holds if $\exists \overline{R} \in RMP \,.\, \exists t \in \overline{R} \,.\, p \cap {}^\bullet t \neq \emptyset$. The reduction to reaction minimal paths

is valid because if $uses[\mathcal{M}, G, A](\overline{R}, p)$ then $uses[\mathcal{M}, G, A](\overline{R'}, p)$ for every reaction minimal path $\overline{R'} \subseteq \overline{R}$. Furthermore, if $\overline{R}$ is reaction minimal then $uses[\mathcal{M}, G, A](\overline{R}, p)$ if $\exists t \in \overline{R} \,.\, (p \cap {}^\bullet t \neq \emptyset)$.

The set of all used places in a path $\overline{R}$ is:

$$uses[\mathcal{M}, G, A](\overline{R}) = \{p \in P \mid uses[\mathcal{M}, G, A](\overline{R}, p)\}.$$

6.7.4 Knockouts

A place p is a (single) knockout for G and A, written $KO[\mathcal{M}, G, A](p)$, if there is no path $\overline{R} \vdash_A^G m_0 \rightsquigarrow m$ using only transitions that do not use p, hence only transitions in $\{t \in T \mid p \cap {}^\bullet t = \emptyset\}$. To check if p is a knockout, we need only check that all reaction minimal paths use p:

$$KO[\mathcal{M}, G, A](p) \Leftarrow \forall \overline{R} \in RMP \,.\, uses[\mathcal{M}, G, A](\overline{R}, p).$$

More generally, $P' \subseteq P$ is a knockout set, $KO[\mathcal{M}, G, A](P')$, if every path $\overline{R} \vdash_A^G m_0 \rightsquigarrow m$ uses some element of P' and there is no proper subset of P' with this property. This can be checked using RMP as follows.

$$\begin{aligned} KO[\mathcal{M}, G, A](P') \Leftarrow \\ &(\forall \overline{R} \in RMP \,.\, \exists p \in P' \,.\, uses[\mathcal{M}, G, A](\overline{R}, p)) \wedge \\ &(\forall P'' \subset P' \,.\, \exists \overline{R} \in RMP \,.\, \forall p \in P'' \,.\, \neg uses[\mathcal{M}, G, A](\overline{R}, p)) \end{aligned}$$

6.7.5 Multi-signal cellular responses

Often a cellular response requires more than one signal to be present. Activation of effector cells of the immune system is one example. Thus it is interesting to ask, *are there paths reaching G from m_0 that require more than one stimulus?*

Let $S \subseteq P$ be the places considered stimuli. Then $\overline{R}$ uses more than one stimulus if $S \cap uses[\mathcal{M}, G, A](\overline{R})$ has more than one element. Again, this can be checked using only the paths in RMP.

6.7.6 Analysis of ERKs and RelA activation

As seen in Table 6.1 the relevant subnet for the activation of ERKs has many more reaction minimal paths than the subnet for activation of RelA. This can be partly explained by the choice of enzymes at several points in the network which leads to a combinatorial number of paths. This also partly

explains why the computation for ERKs involved exploring over four times as many (state, path) tuples than reachable states, whereas the computation for RelA involved less than two times.

From Table 6.2 we see that the RelA relevant subnet has no essential transitions while the ERKs relevant subnet has several. An explanation for this is that RelA has two possible stimuli whereas ERKs only has one stimulus. Conversely, the RelA relevant subnet has considerably more essential transition pairs than the ERKs relevant subnet. The essential transition pairs for RelA are formed by taking one transition downstream from the Tnf stimulus and one transition downstream from the IL1 stimulus. Finally, there are more single knockouts for ERKs than RelA, and more double knockouts for RelA than ERKs.

	Erks–act–EgfRC kb v5	**Rela–act–Nuc kb v5**
Signals	1	2
Essential Transitions	8	0
Essential Transition Pairs	11	183
Used Places	53	84
Single Knockouts	26	9
Double Knockouts	20	674

Table 6.2: The results of computing essential transitions, used places and knockouts for G = {Erks–act–EgfRC} and G = {Rela–act–Nuc}.

The enumeration of essential transition and knockout sets has "discovered" a difference in the structure of the two subnets. For example the RelA subnet is composed of two "uber flows," one for each stimulus, while the ERKs subnet is one signal flow with many local variations.

6.8 Clustering reaction minimal paths

The set of reaction minimal paths can be clustered as follows.

We use a well-known clustering technique called hierarchical clustering. Given a set of reaction minimal paths, *Paths*, each path $\overline{R_i} \in Paths$ is assigned a cluster C_i. The two clusters with minimum distance between them are merged. The distance between two clusters C_i and C_j is calculated using the equation $max(\ \{d(\overline{R_i}, \overline{R_j}) \mid \overline{R_i} \in C_i, \overline{R_j} \in C_j\}\ )$. We use the following simple metric to compute the distance between two paths, $d(\overline{R_i}, \overline{R_j}) = max(|\overline{R_i}|, |\overline{R_j}|) - |\overline{R_i} \cap \overline{R_j}|$.

The results of clustering reaction minimal paths help us understand the underlying structure of signalling network models.

We have clustered the 39 reaction minimal paths for the Rela–act–Nuc kb v5 model. The resulting dendrogram is shown in Figure F.1 in Appendix F. The dendrogram shows that there are three main clusters: one for Tnf-stimulated signal flows, one for IL1-stimulated signal flows and

one for Tnf&IL1-stimulated signal flows.

We have clustered the 144 reaction minimal paths for the Erks–act–EgfRC kb v5 model. The resulting dendrogram is shown in Figure F.2 in Appendix F. The dendrogram shows that there are two main clusters. There is only one stimulus in this model, EGF. The clusters indicate that there are two main "routes" of signal flow through the EGF signalling pathway.

6.9 Discussion

The RMP algorithm searches the state space of a model and therefore suffers from the *state space explosion problem* [86]. Both the time and space complexity of the algorithm can be exponential in the number of components in the model. The analysis of the ERKs activation model from Section 6.7.1 searched over 600,000 tuples. With 4GB of memory, a maximum of around 3 million tuples can be searched in a model with the same number of places. Clearly this approach does not scale sufficiently for larger models.

We have applied this algorithm to models from version 6 of the Pathway Logic knowledge base and found that, for some models, the set of reaction minimal paths was uncomputable. This motivates our next chapter which deals with large unstructured signalling networks.

6.10 Summary

In this chapter we gave an approach to analysing unstructured signalling network models.

In Section 6.1 we gave the motivation for this chapter. In Section 6.2 we defined signal flows and the completeness and correctness properties of algorithms to compute signal flows. In Section 6.3 and Section 6.4 we reviewed the steady-state approach and current dynamic techniques (respectively) to compute signal flows in a signalling network model. We found that, to the best of our knowledge, no current technique guarantees both completeness and correctness. In Section 6.5 we introduced a new algorithm called the Reaction Minimal Paths (RMP) algorithm to compute the set of signal flows in a model. This algorithm guarantees both completeness and correctness. In Section 6.6 we introduced the Pathway Logic modelling framework, which we used to build unstructured signalling network models from the knowledge base of reactions. In Section 6.7 we applied the RMP algorithm to the Pathway Logic models. We computed the set of signal flows and a set of network metrics, each of which provided a better understanding of the models. In Section 6.8 we showed how clustering the set of reaction minimal paths reveals structure within the models. In Section 6.9 we discussed the computational complexity and scalability of the RMP

algorithm and found that it does not scale well for large models.

We now extend this approach to *large* unstructured signalling network models.

Supplemental material An open-source Java application that computes all reaction minimal paths in Pathway Logic models as well as the models used in this chapter can be found at **www.dcs.gla.ac.uk/~radonald/cmsb2010/**. The Pathway Logic Assistant, knowledge bases and documentation can be found at **pl.csl.sri.com**.

Chapter 7

Extension to large unstructured signalling networks

In this chapter we extend the approach from the previous chapter to be applicable to *large* unstructured signalling networks.

In Section 7.1 we give the motivation for this chapter. In Section 7.2 we adapt the Reaction Minimal Paths (RMP) algorithm from the previous chapter to search the reduced state space using two versions of stubborn sets partial order reduction. We prove the algorithms correct, i.e. that they find all reaction minimal paths. We also introduce the Hide Edges algorithm that simplifies certain models. We apply the algorithms to a set of signalling network models from the Pathway Logic framework, including models previously uncomputable using the (original) RMP algorithm. In Section 7.3 we introduce a partial order reduction algorithm that is simpler than stubborn sets called dependence sets. We adapt the RMP algorithm to search the reduced state space using dependence sets partial order reduction. We again prove the algorithm correct and apply it to the same set of signalling network models.

Background material We assume the following background material: Petri nets (Section 3.3), the dynamic behaviour of Petri nets (Section 3.4), the RMP algorithm (Section 6.5.1), and Pathway Logic (Section 6.6). Of particular importance are multisets (Appendix A), paths (Definition 5 on page 25) and state space searches (Definition 7 on page 27).

7.1 Motivation

Although the the longest runtime of the RMP algorithm with version 5 of the Pathway Logic knowledge base was only 50s, it is clear this algorithm does not scale well. For example, one model required searching over 600,000 tuples. With 4GB of memory, a maximum of around 3 million tuples can be searched in a model with the same number of places. Given that the state space can be exponential in the number of components (the state space explosion problem [86]), a small increase in model complexity could result in state spaces that are infeasible for the RMP algorithm.

This observation was confirmed when applying the RMP algorithm to models from version 6 of the Pathway Logic knowledge base. The set of reaction minimal paths in some models was uncomputable.

7.2 Stubborn sets

Stubborn sets [69] are a class of partial order reduction algorithms. We start with some definitions related to stubborn sets and a discussion of an important problem called the *ignoring problem*. We then give a reduced state space search algorithm using stubborn sets. We adapt the RMP algorithm to search the reduced state space using stubborn sets—we refer to this as the *RMP using stubborn sets algorithm*. We prove the algorithm correct, i.e. that it finds all reaction minimal paths. We also introduce an algorithm called the *Hide Edges algorithm* and prove it correct. The Hide Edges algorithm simplifies certain models and thus may improve the effect of partial order reduction. We give results of using the RMP using stubborn sets and Hide Edges algorithms on a set of Pathway Logic models. We repeat the analyses with an alternative RMP using stubborn sets algorithm. Finally, we discuss the results of these algorithms.

7.2.1 Definitions

We use $stub(m) \subseteq T$ to denote a *stubborn set* of transitions in a state m. The reduced state space search fires at each state m only the enabled transitions in a single stubborn set in m, $\{t \in stub(m) \mid m \rightarrow_t\}$, instead of all enabled transitions in the state.

Two desirable properties of stubborn sets are Properties D1 and D2.

Property D1. *If* $t \in stub(m)$, $t_1, \ldots, t_n \notin stub(m)$, $m \rightarrow_{t_1, \ldots, t_n} m_n$ *and* $m_n \rightarrow_t m'_n$, *then there exists an* m' *such that* $m \rightarrow_t m'$ *and* $m' \rightarrow_{t_1, \ldots, t_n} m'_n$.

Property D2. *If m has an enabled transition, then there is at least one transition $t_k \in stub(m)$ such that if $t_1, \ldots, t_n \notin stub(m)$ and $m \rightarrow_{t_1, \ldots, t_n} m_n$, then $m \rightarrow_{t_k}$. Any such t_k is called a key transition of $stub(m)$.*

Definition 29 (D1-D2 stubborn set)**.** *A set $stub(m) \subseteq T$ is a D1-D2 stubborn set if Properties D1 and D2 hold.*

In the remainder of this chapter, we consider only D1-D2 stubborn sets.

The reduced state space search of some models suffers from the *ignoring problem*. The ignoring problem is where one or more enabled transitions never fire, hence they are ignored. This problem affects some signalling network models, such as the example in Figure 7.1.

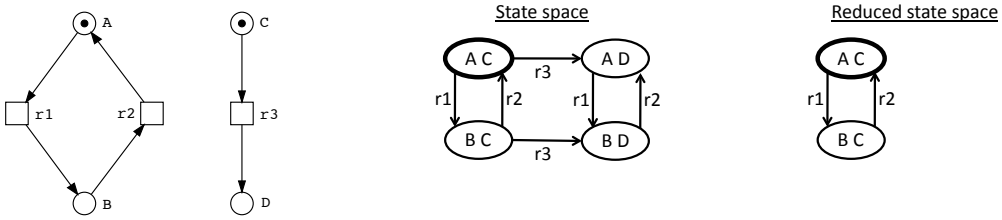


Figure 7.1: A Petri net model (left), the state space (centre) and a possible reduced state space (right). The reduced state space exhibits the ignoring problem, with transition r3 being ignored. This model can be found in biology—transitions $r1$ and $r3$ may be activation reactions and $r2$ may be a deactivation reaction.

A reduced state space search algorithm that satisfies Property S does not suffer from the ignoring problem.

Property S. *For all states m of the reduced state space and every transition $t \in T$ such that $m \rightarrow_t$, there is a sequence of transitions $t_1, \ldots, t_n$ where $m = m_0 \rightarrow_{t_1, \ldots, t_n} m_n$, $t \in stub(m_n)$ and t_i is a key transition of $stub(m_{i-1})$. Hence, if a transition is enabled in a state then it is enabled in some future state reachable by firing transitions in stubborn sets.*

Given a reduced state space search in which there is a finite number of states (i.e. the search terminates) and that satisfies Property S, then for all states m of the reduced state space and all transitions t such that $m \rightarrow_t$, t will either fire in m or in some future state.

A reduced state space search using D1-D2 stubborn sets reaches all terminal states [69]. A reduced state space search that satisfies Property S does not suffer from the ignoring problem [69] and Theorem 6 below holds [69].

Definition 30 (Representative path)**.** *Given two paths $\overline{F}$ and $\overline{R}$, $\overline{R}$ is a representative of $\overline{F}$ if $\overline{R} \supseteq \overline{F}$, i.e. $\overline{R}$ contains the path $\overline{F}$.*

Theorem 6 (Representative paths (stubborn sets))**.** *Assuming a finite state space, for any path $\overline{F}$ in the full state space search there exists a path in the reduced state space search $\overline{R}$ such that $\overline{R}$ is a representative of $\overline{F}$ (Definition 30), i.e. $\overline{R} \supseteq \overline{F}$.*

We now give a reduced state space search algorithm using stubborn sets.

7.2.2 Reduced state space search

We first define an algorithm that computes stubborn sets in a state. A D1-D2 stubborn set in a state m, $stub(m)$, can be computed as follows.

Initialise: pick an enabled transition in m and add it to $stub(m)$
Recurse: apply rules (1) and (2) to newly added transitions in $stub(m)$
(1) if $m \rightarrow_t$ then the add $({}^\bullet t)^\bullet$ to $stub(m)$
(2) if $m \nrightarrow t$ then select a p where $p \in {}^\bullet t \wedge m(p) < f(p,t)$, add ${}^\bullet p$ to $stub(m)$

Rule (1) ensures that no transition outside of $stub(m)$ can disable an enabled transition in $stub(m)$, by adding all post-transitions of the pre-places of the transition to $stub(m)$. Therefore all enabled transitions in $stub(m)$ are key transitions. Rule (2) ensures that no transition outside of $stub(m)$ can enable a disabled transition in $stub(m)$ by selecting a place that causes the transition to be disabled and adding all pre-transitions of the place to $stub(m)$.

Note that often some transitions in $stub(m)$ are disabled in m.

At each state m we fire a subset of the enabled transitions, the enabled transitions in a single stubborn set $stub(m)$. We use the heuristic of choosing $stub(m)$ with the fewest enabled transitions in m. The intuition is that this will produce the fewest unseen states and thus the reduced state space is likely to be as small as possible, though this is not always the case. To find a stubborn set with the fewest enabled transitions, we enumerate all stubborn sets by picking a different enabled transition in the initialise step. If we create a stubborn set of size 1, then we use this stubborn set.

In this chapter we consider only breadth-first search (BFS) because we later adapt the RMP algorithm to follow the reduced state space search using stubborn sets, which requires BFS.

A (BFS) reduced state space search algorithm using stubborn sets is given below.

The set of seen states $S = \emptyset$
Add initial state m_0 to the queue Q
while Q is not empty **do**
  Remove state m from the front of the queue Q
  Fire the enabled transitions in $stub(m)$ to produce states $M = \{m_1, \ldots, m_n\}$
  Add $M \setminus S$ to the back of the queue Q
  Add M to S
end while

We now alter the reduced state space search algorithm to satisfy Property S, therefore avoiding the ignoring problem. The most efficient approach to satisfy Property S involves analysing the terminal strongly connected components in the state space search [69]. However, this approach is only applicable using depth-first search. We require a solution that is applicable using BFS.

The approach used by the SPIN model checker to overcome the ignoring problem using BFS is as follows [7]. For a state m and stubborn set $stub(m)$, if there does not exists a $t \in stub(m)$ such that $m \rightarrow_t m'$ where m' is an unseen state, then fire all enabled transitions in m, $\{t \in T \mid m \rightarrow_t\}$. This is a less effective solution than the solutions applicable when using a DFS because the criterion to identify the ignoring problem is not as specific. In some cases, no unseen states are produced by firing the transitions in a stubborn set, but the ignoring problem is not encountered.

We follow the SPIN approach to satisfy Property S. We add a condition that for a stubborn set $stub(m)$ in state m, it must hold that $\exists t \in stub(m) \,.\, m \rightarrow_t m'$ and m' is an unseen state. If no such stubborn set can be found then we take a step according to the full state space search, i.e. we fire all enabled transitions in the current state, and then resume the reduced state space search.

A (BFS) reduced state space search algorithm using stubborn sets that satisfies Property S is given below.

Add initial state m_0 to the queue Q
while Q is not empty **do**
  Remove state m from the front of the queue Q
  Let $stub(m)$ be the stubborn set in m with the fewest enabled transitions such that $\exists t \in stub(m) \,.\, m \rightarrow_t m'$ and m' is an unseen state.
  if $stub(m)$ exists **then**
    Fire the enabled transitions in $stub(m)$ to produce states $m_1, \ldots, m_n$
  else
    Fire all enabled transitions $\{t \in T \mid m \rightarrow_t\}$ to produce states $m_1, \ldots, m_n$
  end if
  Add any unseen states in $m_1, \ldots, m_n$ to the back of the queue Q
end while

Example 14 Example of a (BFS) reduced state space search using stubborn sets.

We show how to perform a reduced state space search using stubborn sets of the Petri net in Figure 7.2. We follow a BFS search from the initial state, AXF (i.e. places A, X and F are marked).

The reduced state space search using stubborn sets is as follows.

Queue: AXF

Seen states: {AXF}

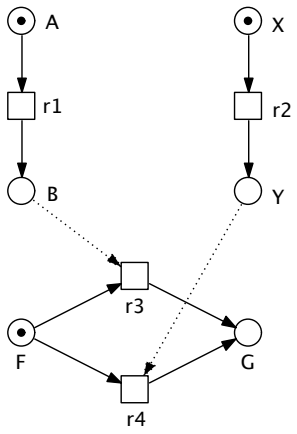


Figure 7.2: An example Petri net used to illustrate the reduced state space search using stubborn sets. Recall that the dashed directed arc from B to r3 is an enzymatic arc defined on Page 24.

AXF - Pick a random enabled transition, $r1$. $stub(AXF) = \{r1\}$. Transition $r1$ is fired producing state BXF.

Queue: BXF

Seen states: {AXF, BXF}

BXF - Pick a random enabled transition, $r3$. The pre-places of $r3$ are B and F. Add $r3$ and $r4$, the post-transitions of B and F, to $stub(BXF)$. A place that causes $r4$ to be disabled in BXF is Y, therefore add all transitions that put a token onto Y, in this case $r2$, to $stub(BXF)$. $stub(BXF) = \{r2, r3, r4\}$. Transitions $r2$ and $r3$ are fired producing states BYF and BXG respectively.

Queue: BYF, BXG

Seen states: {AXF, BXF, BYF, BXG}

BYF - Pick a random enabled transition, $r3$. The pre-places of $r3$ are F and Y. Add $r3$ and $r4$, the post-transitions of F and Y, to $stub(BYF)$. $stub(BYF) = \{r3, r4\}$. Transitions $r3$ and $r4$ are fired producing the same state, BYG.

Queue: BXG, BYG

Seen states: {AXF, BXF, BYF, BXG, BYG}

BXG - Pick a random enabled transition, $r2$. $stub(BXG) = \{r2\}$. Transition $r2$ is fired producing

state BYG. The state BYG has already been seen.

Queue: BYG

Seen states: {AXF, BXF, BYF, BXG, BYG}

BYG - No enabled transitions and the queue is empty, therefore the search terminates.

Queue: *empty*

Seen states: {AXF, BXF, BYF, BXG, BYG}

The reduced state space that was searched above is given in Figure 7.3.

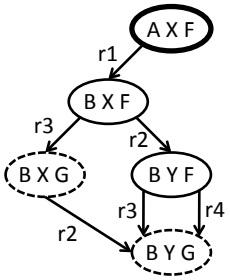


Figure 7.3: The reduced state space search of the Petri net in Figure 7.2.

7.2.3 The RMP using stubborn sets algorithm

We now introduce the *RMP using stubborn sets algorithm*, ssRMP($\mathcal{M}, G, A$). The algorithm is an adaption of the (original) RMP algorithm to follow a reduced state space search using stubborn sets. Recall that reaction minimal paths are paths through the state space to a goal set without using an avoid set. In this section we introduce the algorithm and prove correctness, i.e. that the algorithm finds all reaction minimal paths. Finally, we demonstrate the algorithm with an example.

Algorithm The RMP using stubborn sets algorithm for a k-bounded Petri net $\mathcal{M} = (T, P, f, m_0)$ is ssRMP($\mathcal{M}, G, A$) as follows.

Pre-processs: To make all paths satisfy the avoid constraint, we remove any transition that has a pre- or post-place in the avoid set: $T^* = \{t \in T \mid ({}^\bullet t \cup t^\bullet) \cap A = \emptyset\}$.

Stubborn sets: Compute the set of stubborn sets using T^* in a state m as per the algorithm in Section 7.2.2.

The algorithm for computing stages assuming a pre-processed network works as follows.

Recall from Definition 27 on page 88 that a tuple $(m, \overline{R})$ is subsumed by another tuple $(m', \overline{R'})$ if $\overline{R} \supseteq \overline{R'}$, and $m = m'$ or $m' \models G$.

Stage 0:
Stage(0) = $\{(m_0, \emptyset)\}$
$Paths = \emptyset$
Stage n: Given Stage(n-1)
Stage(n)= $\emptyset$
for $(m, \overline{R}) \in$ Stage(n-1) **do**
  if $m \models G$ **then**
    Add $process(\overline{R})$ to $Paths$
  else
    Let $stub(m)$ be the stubborn set in m with the fewest enabled transitions such that $\exists t \in stub(m)\ .\ m \rightarrow_t m'$ and $(m', Add(\overline{R}, t))$ is not subsumed.
    if $stub(m)$ exists **then**
      $trans = \{t \in stub(m) \mid m \rightarrow_t\}$
    else
      $trans = \{t \in T^* \mid m \rightarrow_t\}$
    end if
    for $t \in trans$ such that $m \rightarrow_t m'$ **do**
      $\overline{R'} = Add(\overline{R}, t)$
      Add $(m', \overline{R'})$ to Stage(n) if it is not subsumed by a tuple in Stage(j) for some $j < n$
    end for
  end if
end for
if Stage(n) == $\emptyset$ **then**
  Return $Paths$
end if

Enforcing the reaction minimal property Theorem 6 guarantees that for each reaction minimal path in the full state space search there exists a representative path in the reduced state space search. Therefore the paths in *Paths* are representative paths—the reaction minimal property is *not* guaranteed. Representative paths require further processing to guarantee the reaction minimal property because they are possibly an extension of reaction minimal paths.

The approach taken by the Pathway Logic Assistant [103] is to generate a single goal/avoid path using LoLA as outlined in Section 6.4.1. The reaction minimal property is enforced on the path using the relevant subnet algorithm (Appendix D). However, there is no proof of correctness given. We have found a counter example—the reaction minimal property is not enforced on the goal/avoid path in Figure 7.4 using the relevant subnet algorithm. Only by using the RMP($\mathcal{M}, G, A$) algorithm can the reaction minimal property of a path be guaranteed.

For each representative path, we apply the reaction minimal path algorithm from Section 6.5.1 using only transitions from the path, i.e. taking the representative path as a subnet of the net. This turns the representative path into a reaction minimal path.

$process(\overline{R})$ =RMP($(T', P, f, m_0), G, A$) where $T' = \{t \mid t \in \overline{R}\}$

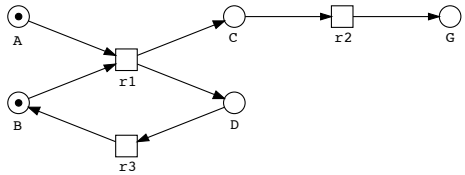


Figure 7.4: A goal/avoid path with $G = \{G\}$ and $A = \emptyset$. The reaction minimal property cannot be enforced on this path using the relevant subnet algorithm. The transition $r3$ is not required to reach $\{G\}$ and will not be removed from the path using the relevant subnet algorithm.

Note that for each representative path, there is only one reaction minimal path. This is because transitions that represent a choice between two or more reaction minimal paths are included in the same stubborn set (rule (1) in the stubborn set algorithm). Therefore the transitions form different paths through the reduced state space. A similar argument holds for dependence sets later in this chapter.

Theorem 7 (The RMP using stubborn sets algorithm is correct)**.** *The RMP using stubborn sets algorithm is correct, i.e. it produces all reaction minimal paths.*

Proof. We know by Theorem 6 that for each reaction minimal path in the full state space search, there exists a representative path in the reduced state space search. We convert the representative paths into reaction minimal paths using the RMP algorithm which is correct (Theorem 5 on page 90), therefore this algorithm is correct. □

Example 15 Example of the RMP using stubborn sets algorithm.

Consider the Petri net in Figure 7.2 with a goal set $\{G\}$ and avoid set $\emptyset$. We compute the set of reaction minimal paths with the RMP using stubborn sets algorithm. The algorithm produces 3 stages as follows. The explanation of how the stubborn sets are computed is given in Example 14.

Stage 0: (AXF, $\emptyset$)

From state AXF, $stub(AXF) = \{r1\}$ and transition $r1$ is fired producing state BXF.

Stage 1: (BXF, $\{r1\}$)

From state BXF, $stub(BXF) = \{r2, r3, r4\}$ and transitions $r2$ and $r3$ are fired producing states BYF and BXG respectively.

Stage 2: (BXG, $\{r1, r3\}$) (BYF, $\{r1, r2\}$)

From state BXG, no transitions are fired because the goal constraint is satisfied.

From state BYF, $stub(BYF) = \{r3, r4\}$ and transitions $r3$ and $r4$ are fired producing the same state, BYG.

Stage 3: (BYG, $\{r1, r2, r4\}$) From state BYG, no more enabled transitions.

The set of *representative paths* is: $\{\{r1, r3\}, \{r1, r2, r4\}\}$

The set of *reaction minimal paths* is: $\{\{r1, r3\}, \{r2, r4\}\}$

7.2.4 The Hide Edges algorithm

We introduce an algorithm called the *Hide Edges algorithm* that simplifies certain models, especially Pathway Logic models. The simplification process removes edges in a Petri net that do not affect the behaviour of the net (i.e. the state space). This algorithm makes Petri nets easier to understand visually and may also enhance the effect of partial order reduction.

We start with the biological justification for the algorithm, then give the algorithm and an example, and finally prove the algorithm correct.

Motivation Models in Pathway Logic are built from the Pathway Logic knowledge base as outlined in Section 6.6. Datums, generated from laboratory experiments, are used to construct single reactions in the knowledge base. Given an initial state (the proteins, ligands, etc. that are initially present), a model is automatically built by collecting all reactions in the knowledge base that can fire.

To explain the motivation of this algorithm, we continue with Example 13 from Page 96. Recall the set of curated datums.

0. With no ligands, proteins X, Y and Z are inactive.
1. Given ligand L, protein X is active.
2. Given ligand L, protein Y is active.
3. Given ligand L, protein Z is active.
4. Given ligand L and protein X knocked-out, protein Y is inactive.
5. Given ligand L and protein X knocked-out, protein Z is inactive.
6. Given ligand L and protein Y knocked-out, protein Z is inactive.

Three reactions are generated from this set of datums: the activation of X, Y and Z. The enzymes for each reaction are clear from the results of the datums. The enzyme for the activation of X is L (datum 1). The enzymes for the activation of Z are L, X and Y (datums 3, 5 and 6). The result is a model as shown in Figure 7.5.

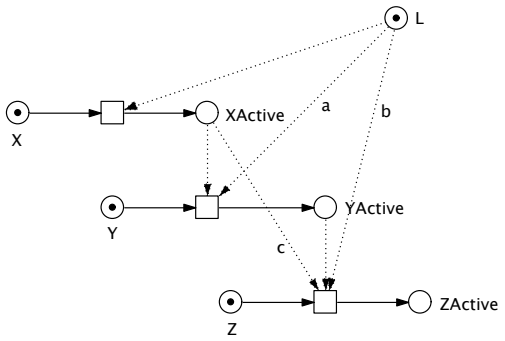


Figure 7.5: The model built using the Pathway Logic approach from datums 0 ... 6.

The enzymatic edges a, b and c in the model are not required. Edges a and b are redundant because *XActive* is an enzyme for the activation of Y and Z respectively. In other words, if *XActive* is present then we can guarantee that L is also present and therefore the enzymatic edge from L to the reaction is redundant. Likewise, edge c is redundant because *YActive* is an enzyme for the activation of Z—*XActive* is present if *YActive* is present.

Edge c is a result of datum 6. With X knocked-out, Z is not activated so we assume that X is an enzyme for the activation of Z. Only when the full model is uncovered do we see that when X is knocked-out, Y cannot become activated which in-turn cannot activate Z. The enzymatic interaction between X and Z *may* simply be a product of X being required for the activation of Y which is required for the activation of Z. Without checking whether X and Z physically interact, we do not know which model is biologically consistent.

We hide these edges to produce a logically equivalent model that is simpler to reason about.

The model with the edges hidden is shown in Figure 7.6.

Algorithm We now introduce the Hide Edges algorithm, HideEdges($\mathcal{M}$), that removes edges in a Petri net $\mathcal{M}$ that do not affect the behaviour of the Petri net, such as edges a, b and c in Figure 7.5.

The algorithm labels each place in a Petri net with the set of enzymes that can be guaranteed present if the place is marked. If pre-place p of transition t is labelled with an enzyme e, then any enzymatic edge between e and t can be safely removed.

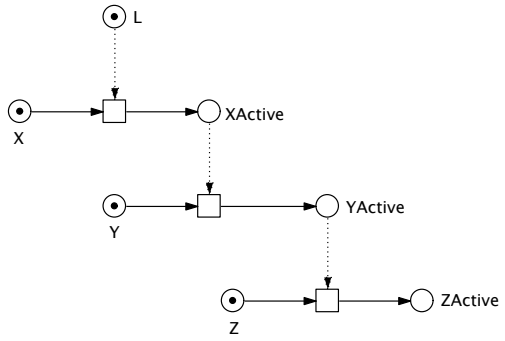


Figure 7.6: The model built using the Pathway Logic approach from datums 0 ... 6 after applying the Hide Edges algorithm.

The HideEdges($\mathcal{M}$) algorithm is in three steps below: seed, propagate and hide.

Seed step Label each place with the set of enzymes that can be guaranteed present by firing a single transition that puts tokens on the place.

---

for each place p in the initial state **do**
  Label p with $\emptyset$
end for
for each place p not in the initial state **do**
  for each transition t that can mark p **do**
    l = the enzymes for t that are never consumed by any transition
  end for
  Label p with the intersection of all such l
end for

Propagate step Propagate the intersection of the labels on the transitions that can mark a place to the place (strictly, add the intersection of the labels to the label on the place). The labels on a transition is the union of the labels on the pre-places of the transition. Apply propagation until convergence, i.e. the set of labels do not change.

---

for each place p not initially marked **do**
  for each transition t that can mark p **do**
    l = the union of the set of labels on the pre-places of t
  end for
  Add to the labels on p the intersection of all such l
end for
Repeat until convergence, i.e. the set of labels do not change

---

Hide step Remove the enzymatic edges *of weight 1* that do not affect the behaviour of the net as indicated by the enzyme being in the label on the transition. The labels on a transition is the union

of the labels on the pre-places of the transition.

---

for each transition t **do**
  l = the union of the set of labels on the pre-places of t
  Remove any enzymatic edge *of weight 1* from enzyme e to t if e is in l
end for

Example 16 Example of the Hide Edges algorithm.

We apply the Hide Edges algorithm to the Petri net in Figure 7.7.

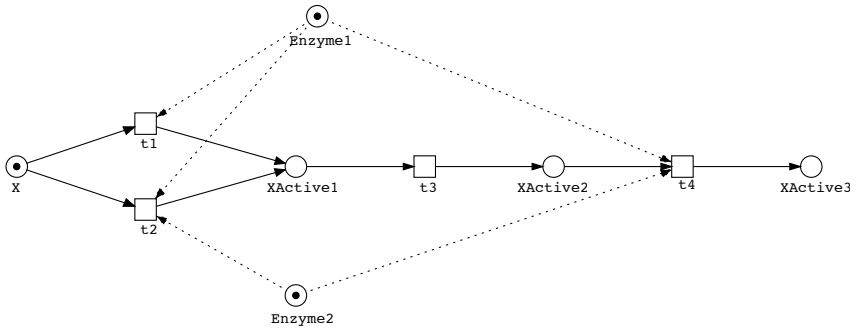


Figure 7.7: The original Petri net before applying the Hide Edges algorithm.

The application of the Hide Edges algorithm follows Figure 7.8.

The seed step labels *XActive*3 with {*Enzyme*1, *Enzyme*2} because $t4$ is the only transition that can produce *XActive*3 and it has *Enzyme*1 and *Enzyme*2 as enzymes. *XActive*1 is labelled with {*Enzyme*1} because *Enzyme*1 is an enzyme for both transitions $t1$ and $t2$ that produce *XActive*1. The label on *XActive*1 does not include *Enzyme*2 because *Enzyme*2 is only an enzyme for $t2$ and therefore cannot be guaranteed present if *Xactive*1 is marked.

In the propagate step, propagation is applied twice. In the first propagation, *enzyme*1 is propagated to (strictly, added to the label on) *XActive*2 because *enzyme*1 is in the label of *XActive*1 which is a pre-place for the only transition $t3$ that produces *Xactive*2. In the second propagation, no labels are changed and therefore propagation converges.

The hide step removed the enzymatic edge between *Enzyme*1 and $t4$ because a pre-place of $t4$, *XActive*2 has a label that contains *Enzyme*1. This is the only enzymatic edge that is removed.

We prove the algorithm is correct for any Petri net (note, 1-safeness is not required). In the following proof we use the shorthand notation of $p \in m$ for $m(p) > 0$, i.e. p is marked in m.

Theorem 8 (Termination)**.** *For any Petri net, the algorithm will always terminate.*

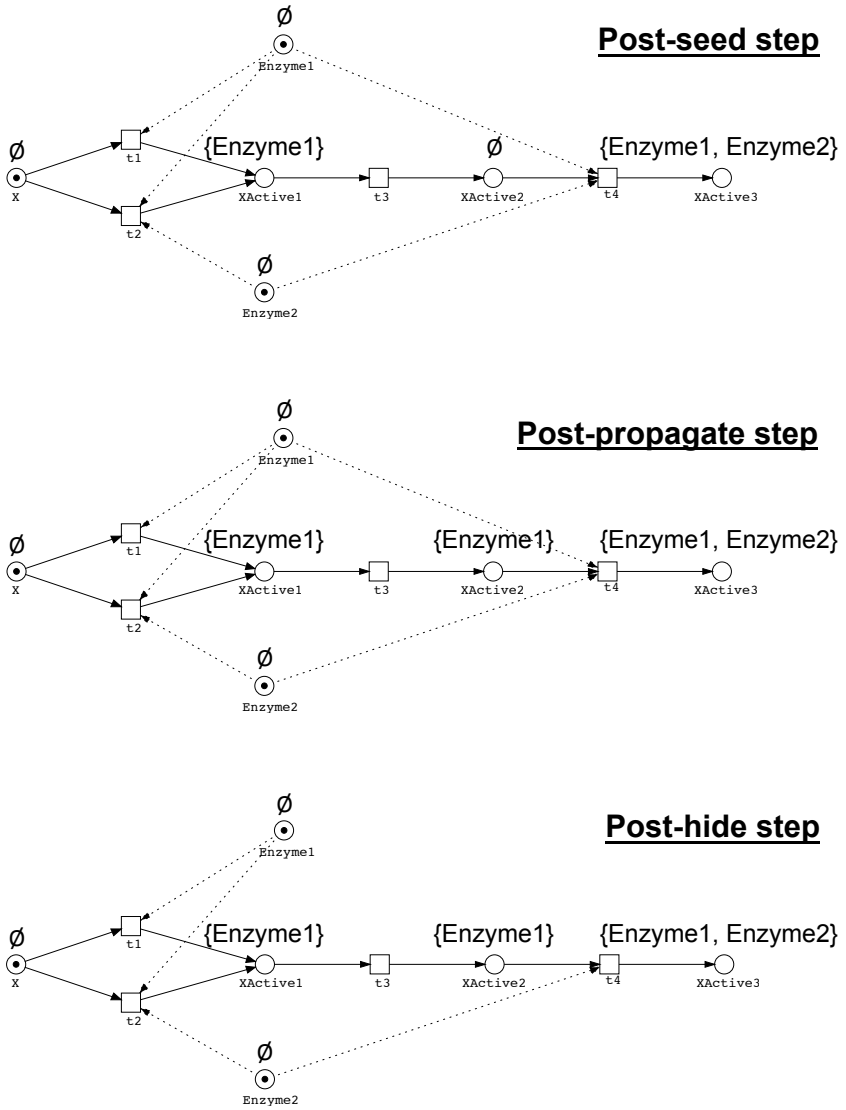


Figure 7.8: The application of the Hide Edges algorithm to the Petri net from Figure 7.8. The Petri net returned by the algorithm is shown in the "post-hide step."

Proof. The seed and hide steps iterate over places and transitions which are both finite, therefore these steps terminate. Propagation iterates over places and transitions which are both finite, therefore propagation will terminate. However, propagation is repeated until convergence, i.e. until the labels do not change. Propagation only adds to labels, which are sets of places. Convergence will be reached, and thus the propagation step is finite and will terminate, because in the extreme there will be no more places to add to the labels (there is a finite number of places). Therefore because all three steps terminate, the algorithm always will terminate. □

Theorem 9 (Labelling correctness)**.** *Given a Petri net, a place p and associated labelling l after zero or more propagations, $\forall e \in l$, in any reachable state if p is marked then e is guaranteed to be marked, i.e. the labelling is correct.*

Proof. We prove this theorem by a proof by induction over the number of propagations.

Base case

The base case is 0 propagations, therefore only the seed step has been applied. If $p \in m_0$ then $l = \emptyset$ so the theorem is true. Otherwise, p is marked at some future state and we reason about all possible sequences of transitions $t_0, \ \ldots, \ t_k$ such that $m_0 \rightarrow_{t_0, \ \ldots, \ t_{k-1}} m_k \rightarrow_{t_k} m_{k+1}$, $p \notin m_0 \wedge \ \ldots \ \wedge p \notin m_k$ and $p \in m_{k+1}$. l is the intersection of the enzymes for any transition that can produce p. Therefore e is an enzyme for t_k and because $m_k \rightarrow_{t_k}$ then $e \in m_k$. The enzymes in l are never consumed, therefore if $e \in m_k$ then $e \in m_j$ for $j \geq k$. Because $e \in m_j$ for $j \geq k$ and p is first marked in m_{k+1}, whenever p is marked, e is guaranteed to be marked.

Therefore the theorem holds for the seed step.

Inductive step

Suppose the theorem holds after n propagations, we prove that the theorem holds after $n + 1$ propagations. If $p \in m_0$ then no e is added to l so the theorem is true. Otherwise, p is marked at some future state and we reason about all possible sequences of transitions $t_0, \ \ldots, \ t_k$ such that $m_0 \rightarrow_{t_0, \ \ldots, \ t_{k-1}} m_k \rightarrow_{t_k} m_{k+1}$, $p \notin m_0 \wedge \ \ldots \ \wedge p \notin m_k$ and $p \in m_{k+1}$. In the $n + 1$th propagation, we only add e to l if e is in the label of at least one pre-place of any transition that can produce p. Therefore e must be in a label on a pre-place p of t_k and because $m_k \rightarrow_{t_k}$ and we assume the nth propagation is correct, then $e \in m_k$. The enzymes in l are never consumed, therefore if $e \in m_k$ then $e \in m_j$ for $j \geq k$. Because $e \in m_j$ for $j \geq k$ and p is first marked in m_{k+1}, whenever p is marked, e is guaranteed to be marked.

Therefore the theorem holds after $n + 1$ propagations and by induction the theorem is correct.

□

Theorem 10 (Hiding correctness)**.** *The state space of a Petri net is not changed by hiding the edges using the hide step based on a labelling satisfying Theorem 9.*

Proof. We prove this theorem by a proof by induction over the number of transitions fired.

Base case

Trivially, with no transitions fired, m_0 is unaffected by hiding edges.

Therefore the theorem holds with no transitions fired.

Inductive step

Suppose the theorem holds after firing n transitions, we prove that the theorem holds after firing $n + 1$ transitions. We reason about all possible sequences of $n + 1$ transitions $t_0, \ldots, t_n$ such that $m_0 \rightarrow_{t_0, \ldots, t_{n-1}} m_n \rightarrow_{t_n} m_{n+1}$. $hidden(t_n)$ is the transition t_n with enzymatic edges *of weight 1* removed between t_n and enzyme e if e is in the label of at least one pre-place p of t_n.[1] Suppose $hidden(t_n)$ can fire in m_n then because of Theorem 9, $e \in m_n$. Therefore t_n can fire in m_n because all $e \in m_n$ as above. Trivially if t_n can fire, then $hidden(t_n)$ can fire because $hidden(t_n)$ has fewer enzymes. Finally, because removing the enzymatic edge between e and t_n does not affect the resultant state, m_{n+1} is the same. Therefore the set of possible m_{n+1}'s is unchanged by hiding edges.

Therefore the theorem holds after firing $n + 1$ transitions and by induction the theorem is correct.

□

Theorems 8, 9 and 10 together prove the algorithm is correct and always terminates.

7.2.5 Pathway Logic results

We compare the performance of the RMP using stubborn sets algorithm with the (original) RMP algorithm. We also show how pre-processing with the Hide Edges algorithm affects reduction.

We compute the set of reaction minimal paths for both the ERK activation and RelA activation models generated from version 5 of the Pathway Logic knowledge base (kb v5), as per Section 6.7. We extend the comparison of the algorithms to models generated from version 6 of the knowledge base (kb v6)—these models are significantly larger and more complex than their counterparts from version 5.

The results of the algorithms are given in Table 7.1.

Comparing the RMP using stubborn sets algorithm with the (original) RMP algorithm shows that the reduction is either trivial or non-existent. In both ERKs activation models there is no

[1] Note, we only remove enzymatic edges of weight 1—we cannot remove enzymatic edges of weight > 1 because the labelling only guarantees that a place is marked, not how many tokens are on it.

	RMP	ssRMP	ssRMP & HideEdges
Erks–act–EgfRC kb v5 (144 paths)	Tuples: 618,861 Runtime: 50s	Tuples: 618,861 Runtime: 333s	Tuples: 437,484 Runtime: 68s
Rela–act–Nuc kb v5 (39 paths)	Tuples: 171,237 Runtime: 9s	Tuples: 162,818 Runtime: 102s	Tuples: 161,806 Runtime: 39s
Erks–act–EgfRC kb v6 (160 paths)	Tuples: 1,358,465 Runtime: 207s	Tuples: 1,358,465 Runtime: 987s	Tuples: 970,354 Runtime: 244s
Rela–act–Nuc kb v6 (??? paths)	Not computable	Not computable	Not computable

Table 7.1: The performance of the RMP using the stubborn sets algorithm, and the Hide Edges algorithm, for four Pathway Logic models. Tuples is the number of (state, path) tuples. Runtime is on a workstation with a 2.53GHz dual core processor with 4GB of memory. Not computable means no results were obtained within 24 hours.

reduction. This is because all transitions have a shared enzyme—the active EGF receptor. This causes any stubborn set computed in any state to be populated with all transitions in the model, therefore for any m, $stub(m) = T$. There is some reduction in the RelA activation model from version 5 of the knowledge base because there are two receptors, i.e. not all transitions have a shared enzyme.

Pre-processing with the Hide Edges algorithm allows more significant reduction. The Hide Edges algorithm removes many of the connections between transitions, causing $stub(m)$ to have fewer transitions in general. The reduction in the state space of the ERKs activation models is greater than the reduction in the RelA activation model—approximately 25% of the states are removed compared with 5% of the states respectively. This reflects the structure of these models; the ERKs models contain many sequences of transitions which are more amenable to partial order reduction, whereas RelA is a more interconnected network of reactions.

Note that the runtime is much higher in the RMP using stubborn sets algorithm. Computing a stubborn set at each state is computationally expensive compared with creating the next state. In all cases, partial order reduction increased the execution time of the RMP algorithm.

Finally, even with pre-processing with Hide Edges, the set of reaction minimal paths in the RelA activation model from version 6 of the knowledge base is not computable.

7.2.6 Alternative stubborn sets algorithm

After personal correspondence with Prof. Valmari [107], we were made aware of an alternative stubborn sets algorithm that is in general more efficient in terms of state space reduction. The algorithm differs in the $stub(m)$ function defined in Section 7.2.2. The following step in the algorithm:

(1) if $m \rightarrow_t$ then add $({}^\bullet t)^\bullet$ to $stub(m)$

is changed to:

(1′) if $m \rightarrow_t$ then add t' to $stub(m)$ if $\exists p\ .\ (min(W(t,p), W(t',p)) < min(W(p,t), W(p,t')))$

where $W(t,p)$ and $W(p,t)$ is the weight of the arc from t to p and from p to t respectively.

Given that the Petri nets in Pathway Logic have arcs of weight 1, we can reduce this step to a boolean equation with propositions on the presence of arcs.

(1″) if $m \rightarrow_t$ then add t' to $stub(m)$ if $\exists p\ .\ (\neg(t \rightarrow p \wedge t' \rightarrow p)) \wedge (p \rightarrow t \wedge p \rightarrow t'))$

where $t \rightarrow p$ and $p \rightarrow t$ is the presence of an arc from t to p and from p to t respectively.

The *RMP using alternative stubborn sets algorithm* ssRMP'$(\mathcal{M}, G, A)$ is the RMP using stubborn sets algorithm using rule (1″) instead of rule (1).

We repeat the analyses from Section 7.2.5 with the RMP using alternative stubborn sets algorithm. The results of the algorithms are given in Table 7.2.

	RMP	ssRMP'	ssRMP' & HideEdges
Erks–act–EgfRC kb v5 (144 paths)	Tuples: 618,861 Runtime: 50s	Tuples: 1,258 Runtime: 2s	Tuples: 1,258 Runtime: 2s
Rela–act–Nuc kb v5 (39 paths)	Tuples: 171,237 Runtime: 9s	Tuples: 2,145 Runtime: 3s	Tuples: 1,969 Runtime: 2s
Erks–act–EgfRC kb v6 (160 paths)	Tuples: 1,358,465 Runtime: 207s	Tuples: 1,463 Runtime: 3s	Tuples: 1,463 Runtime: 3s
Rela–act–Nuc kb v6 (156 paths)	Not computable	Tuples: 252,728 Runtime: 343s	Tuples: 247,938 Runtime: 317s

Table 7.2: The performance of the RMP using alternative stubborn sets algorithm, and the Hide Edges algorithm, for four Pathway Logic models. Tuples is the number of (state, path) tuples. Runtime is on a workstation with a 2.53GHz dual core processor with 4GB of memory. Not computable means no results were obtained within 24 hours.

In all cases, the RMP using alternative stubborn sets algorithm performs well compared to the (original) RMP algorithm. The memory requirements for the Erks–act–EgfRC kb v6 model is three orders of magnitude less and the runtime is two orders of magnitude less. Pre-processing with Hide Edges provides a small but noticeable improvement in reduction. The RelA activation model from version 6 of the knowledge base was previously uncomputable—with the RMP using alternative stubborn sets algorithm, results are returned after around 6 minutes. This is a significant improvement in performance.

7.2.7 Discussion

Models of biochemical systems typically contain sequences of independent reactions that propagate signal through the cell. Partial order reduction works particularly well for such sequences.

The size of the state space for a model of N sequences of M independent transitions is N^{M+1}. Applying partial order reduction algorithm, the transitions are fired in a linear sequence, resulting in $(N * M) + 1$ states. Consider the model in Figure 7.9 comprising two independent sequences of two transitions. The state space has $2^{2+1} = 8$ states whereas the reduced state space has $(2 * 2) + 1 = 5$ states.

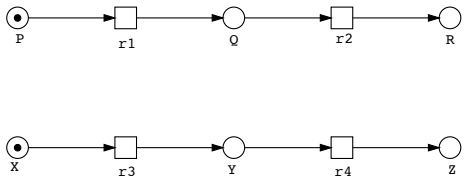


Figure 7.9: A model of two independent sequences of two transitions.

Suppose now that the model of two independent sequences of two transitions has a shared enzyme E for all transitions, as shown in Figure 7.10. This is common in Pathway Logic models. A stubborn set for any state in this model is created as follows. A seed transition is chosen, t, where the pre-places of t include the place E. The transitions that have E as a pre-place are added to the stubborn set, which is all transitions in the model. Therefore, $stub(m) = T$ for any state m, resulting in no reduction in the size of the state space. This example explains why using the (original) stubborn sets algorithm gives no reduction in the ERKs activation models. The Hide Edges algorithm removes some of the connections between transitions, thus enhancing the effect of partial order reduction, shown in Figure 7.10.

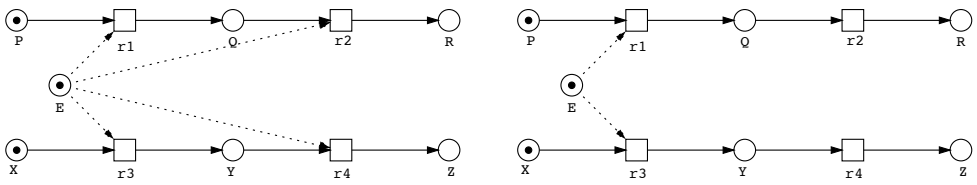


Figure 7.10: (left) a model of two sequences of independent transitions with a common enzyme. (right) the same model after applying the Hide Edges algorithm.

The RMP using alternative stubborn sets algorithm does not suffer from the above problem. The results from this algorithm are encouraging. A previously uncomputable model is now computable after around 6 minutes. The Hide Edges algorithm also provides a small but noticeable improvement in reduction in the alternative stubborn sets algorithm.

The stubborn sets algorithms are however unintuitive and their relationship to biological structure in the models is not obvious. This is especially true because the stubborn sets change depending on the current state of the model.

We now investigate a purely structural (i.e. state independent) partial order reduction algorithm called dependence sets that relates well to biological concepts.

7.3 Dependence sets

In this section we introduce an algorithm called dependence sets, which is a partial order reduction algorithm within the persistent sets class [46]. We start with the biological motivation for dependence sets—dependence sets formalise important biological concepts. We give some definitions of dependence sets and a reduced state space search algorithm using dependence sets. We adapt the RMP algorithm to search the reduced state space using dependence sets—we refer to this as the *RMP using dependence sets algorithm*. We prove the algorithm correct, i.e. that it finds all reaction minimal paths. We give results of using the RMP using dependence sets algorithm on a set of Pathway Logic models. Finally, we discuss these results and compare them to the results of using the RMP using stubborn sets algorithm.

7.3.1 Biological motivation

Two important observations about signalling networks models are as follows.

1. There are often many reactions that are independent steps in the signal propagation, e.g. protein activation, translocation or composition/decomposition.

2. Some reactions represent a choice in the network, e.g. reactions that activate a protein in different ways or reactions that vary only in the choice of enzyme.

These observations can be formalised using the dependency between transitions. Two transitions are dependent if firing one transition can disable the other transition. This is illustrated in Figure 7.11.

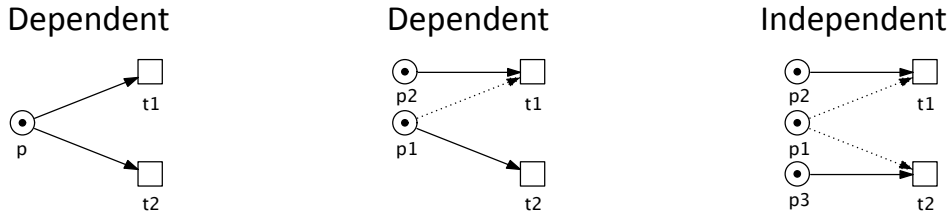


Figure 7.11: (left) $t1$ and $t2$ are dependent because they both consume place p. (centre) $t1$ and $t2$ are dependent because $t2$ consumes place $p1$ which is an enzyme for $t1$. (right) $t1$ and $t2$ are independent because the only place they share is an enzyme for both transitions.

Finally, we show in Figure 7.12 how dependency can be used to partition the transitions in a signalling network model. The transitions are partitioned into "dependence sets" (formalised later). Dependence sets of size 1 are independent steps in the signal propagation.

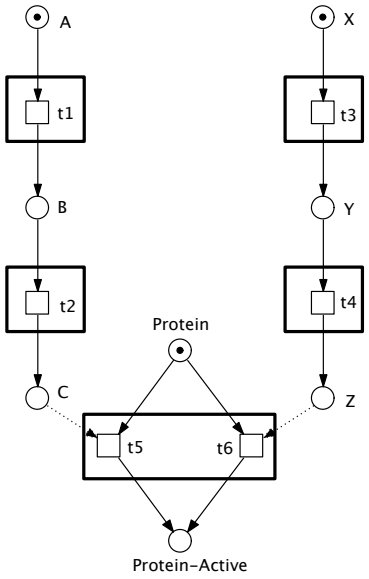


Figure 7.12: The transitions in this Petri net can be partitioned into dependence sets. Transitions $t1$, $t2$, $t3$ and $t4$ are each in dependence sets of size 1, i.e. they are independent steps in the signal propagation. Transitions $t5$ and $t6$ are in the same dependence set because these transitions can disable each other.

7.3.2 Definitions

We now define transition dependency.

Definition 31 (Transition dependency)**.** *Two transitions* $t1$ *and* $t2$ *are dependent if firing* $t1$ *can disable* $t2$ *or vice-versa. Hence,* $dependent(t1, t2) = ((consumed(t1) \cap {}^\bullet t2) \cup ({}^\bullet t1 \cap consumed(t2)) \neq \emptyset)$ *where* $consumed(t) = {}^\bullet t - t^\bullet$ *(the set of places that are consumed by firing* t*). If two transitions are not dependent,* $\neg dependent(t1, t2)$*, they are independent.*

Note that we define dependency by analysing the Petri net structure, i.e. dependency is computed independently from the current state. This is an over approximation because two transitions could be labelled dependent when the set of reachable states in the Petri net never allows the transitions to disable each other, for example Figure 7.13. We use this over approximation because it is simple to understand and removes the need to recompute the dependence sets at each state. Furthermore, dependence in standard Petri nets is rather simple to compute because there is no explicit inhibition—a transition cannot be disabled by the presence of a token on a place.

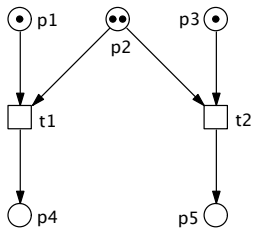


Figure 7.13: Transitions $t1$ and $t2$ are dependent, but in no reachable state can either disable the other. Note that place $p2$ has two tokens, therefore this Petri net is not in the class of models we consider in this thesis.

Definition 32 (Dependence set)**.** *A set of transitions D is a dependence set with $|D| \geq 1$ and if $|D| > 1$ then $\forall t \in D \,.\, \exists t' \in (D - t) \,.\, dependent(t, t')$.*

Definition 33 (Dependence partition of a Petri net)**.** *A dependence partition is a set of dependence sets $\{D_1, \ \ldots, \ D_n\}$. A Petri net $\mathcal{M} = (T, P, f, m_0)$ has a dependence partition of maximal size, found by computing the transitive closure of $dependent(t, t')$ over T.*

In what follows we assume dependence sets are of maximum size with respect to the given Petri net.

We now give a reduced state space search algorithm using dependence sets.

7.3.3 Reduced state space search

We first define an algorithm that computes the dependence partition of a Petri net $\mathcal{M}$. Recall that our definition of dependency is purely structural, therefore the dependence sets need to be computed only once.

Initialise: $\{D_1 = \{t_1\}, \ \ldots, \ D_n = \{t_n\}\}$ where $t_i \in T$ and $n = |T|$
for each D_i, D_j and $j > i$ **do**
  if $\exists t \in D_i \,.\, \exists t' \in D_j \,.\, dependent(t, t')$ **then**
    $merge(D_i, D_j)$
  end if
end for

At each state m we fire a subset of the enabled transitions, a single dependence set D_i that satisfies the following two conditions.

- All transitions in the dependence set must be enabled, $\forall t \in D_i . m \rightarrow_t$.
- To overcome the ignoring problem (as per stubborn sets in Section 7.2.2), there must exist a transition in the dependence set which, when fired from m, reaches an unseen state, i.e. $\exists t \in D_i \,.\, m \rightarrow_t m'$ and m' is unseen.

If a dependence set D_i cannot be found satisfying these conditions, we take a step according to the full state space search, i.e. we fire all enabled transitions in the current state, and then resume the reduced state space search.

We use the heuristic of choosing the D_i that satisfies these conditions that is of smallest size. The intuition is that this will produce the fewest number of unseen states and thus the reduced state space is likely to be as small as possible, though this is not always the case.

Again, we consider only BFS because we later adapt the RMP algorithm to follow the reduced state space search using dependence sets, which requires BFS.

A (BFS) reduced state space search algorithm using dependence sets is given below.

The set of seen states $S = \emptyset$
Add initial state m_0 to the queue Q
while Q is not empty **do**
  Remove state m from the front of the queue Q
  Let D_i = smallest dependence set such that $\forall t \in D_i \,.\, m \rightarrow_t$ and $\exists t \in D_i \,.\, m \rightarrow_t m'$ and m' is an unseen state.
  if D_i exists **then**
    Fire the transitions in D_i to produce states $M = \{m_1, \ \ldots, \ m_n\}$
  else
    Fire the transitions in $\{t \in T \mid m \rightarrow_t\}$ to produce states $M = \{m_1, \ \ldots, \ m_n\}$
  end if
  Add $M \setminus S$ to the back of the queue Q
  Add M to S
end while

Example 17 Example of a (BFS) reduced state space search using dependence sets.

We show how to perform a reduced state space search using dependence sets of the Petri net in Figure 7.14. We follow a BFS search from the initial state, AXF (i.e. places A, X and F are marked).

The reduced state space search using dependence sets is as follows.

The dependence sets in this model are $D_1 = \{r1\}$, $D_2 = \{r2\}$ and $D_3 = \{r3, r4\}$.

Queue: AXF

Seen states: {AXF}

AXF - Pick $D_1 = \{r1\}$ because it is the smallest dependence set such that all transitions are enabled. Transition $r1$ is fired producing state BXF.

Queue: BXF

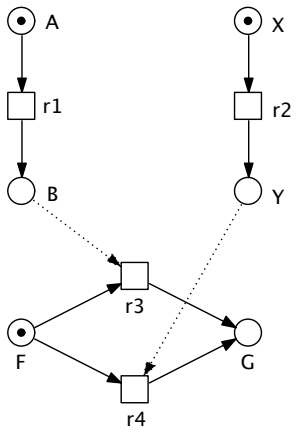


Figure 7.14: An example Petri net used to illustrate the reduced state space search using dependence sets.

Seen states: {AXF, BXF}

BXF - Pick $D_2 = \{r2\}$ because it is the smallest dependence set such that all transitions are enabled. Transition $r2$ is fired producing state BYF.

Queue: BYF

Seen states: {AXF, BXF, BYF}

BYF - Pick $D_3 = \{r3, r4\}$ because it is the smallest dependence set such that all transitions are enabled. Transitions $r3$ and $r4$ are fired producing the same state, BYG.

Queue: BYG

Seen states: {AXF, BXF, BYF, BYG}

BYG - No enabled transitions and the queue is empty, therefore the search terminates.

Queue: *empty*

Seen states: {AXF, BXF, BYF, BYG}

The reduced state space that was searched above is given in Figure 7.15.

7.3.4 Dependence sets propositions/theorems

We now prove two propositions and one theorem for the reduced state space generated by dependence sets.

We first introduce some short-hand notation for firing sets of transitions and firing transitions

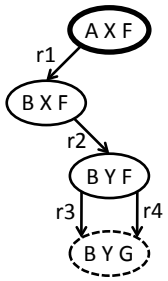


Figure 7.15: The reduced state space search of the Petri net in Figure 7.14.

from sets of states.

We can fire a set of transitions from a single state.
$m \rightarrow_D M_2$ denotes $\forall t \in D \,.\, m \rightarrow_t m'$ where $M_2 = \{m' \mid m \rightarrow_t m', t \in D\}$

We can fire a transition from a set of states.
$M_1 \rightarrow_t M_2$ denotes $\forall m \in M_1 \,.\, m \rightarrow_t m'$ where $M_2 = \{m' \mid m \rightarrow_t m'\}$
$M_1 \rightsquigarrow_t M_2$ denotes $\exists m \in M_1 \,.\, m \rightarrow_t m'$ where $M_2 = \{m' \mid m \rightarrow_t m'\}$

We can fire a set of transitions from a set of states.
$M_1 \rightarrow_D M_2$ denotes $\forall m \in M_1 \,.\, m \rightarrow_D M_2'$ where $M_2 = \bigcup M_2'$
$M_1 \rightsquigarrow_D M_2$ denotes $\exists m \in M_1 \,.\, m \rightarrow_D M_2'$ where $M_2 = \bigcup M_2'$

Proposition 1 (Enabledness preserving property of dependence sets)**.** *Consider two dependence sets D_1 and D_2 and a third transition $t \notin (D_1 \cup D_2)$. Consider further a state m where $m \rightarrow_{D_1}$, $m \rightarrow_{D_2}$ and $m \not\rightarrow_t$. We can fire $m \rightarrow_{D_1} M_1 \rightarrow_{D_2} M_3$ or we can fire $m \rightarrow_{D_2} M_2 \rightarrow_{D_1} M_3$, as shown in Figure 7.16. Suppose $M_2 \rightsquigarrow_t$, then it must follow that $M_3 \rightsquigarrow_t$, i.e. by considering either order of firing the dependence sets, we do not lose the ability to fire t.*

Proof. If t is enabled in some state in M_2 ($M_2 \rightsquigarrow_t$) then t must be enabled in some state in M_3 ($M_3 \rightsquigarrow_t$) because $\forall t' \in D_1 \,.\, \neg dependent(t, t')$, hence t' cannot disable t.

□

Proposition 2 (An enabled transition will fire now or in the future)**.** *Assuming a finite state space, at each state in the reduced state space search m, if $m \rightarrow_t$ then t it either fires from m or some state reachable from m in the search, $m \rightarrow_{t_1, \ldots, t_n} m'$.*

Proof. We prove this proposition by a proof by induction over the number of transitions fired from

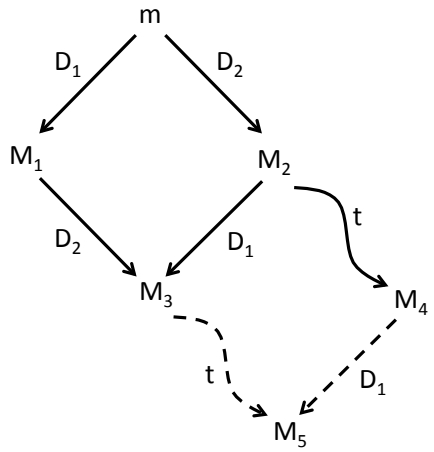


Figure 7.16: By following the branch $m \rightarrow_{D_1} M_1 \rightarrow_{D_2} M_3$ we do not miss the opportunity to fire transition t.

m.

Base case

Suppose no transitions have fired. At m_0 a dependence set D is chosen such that $\forall t' \in D \,.\, m_0 \rightarrow_{t'}$ and $\exists t' \in D \,.\, m_0 \rightarrow_{t'} m'$ and m' is an unseen state. If there is no such D, then all enabled transition are fired and hence t fires from m_0. Otherwise, either $t \in D$ therefore t fires from m_0, or else $m_0 \rightarrow_D M_1 \rightarrow_t$ because $\forall t' \in D \,.\, \neg dependent(t, t')$. $\forall m' \in M_1$ the number of unexplored state from m' is less than the number of unexplored states from m_0, therefore at m' or some state reachable from m' we will pick a D with $t \in D$ or we will fire all enabled transitions and thus t will fire in a state reachable from m_0.

Therefore the proposition holds for m_0.

Inductive step

Suppose the proposition holds after firing n transitions, $m \rightarrow_{t_1} m_1 \ \ldots \ m_{n-1} \rightarrow_{t_n} m_n$, we prove that the proposition holds after firing $n+1$ transitions. At m_n a dependence set D is chosen such that $\forall t' \in D \,.\, m_n \rightarrow_{t'}$ and $\exists t' \in D \,.\, m_n \rightarrow_{t'} m'$ and m' is an unseen state. If there is no such D, then all enabled transition are fired and hence t fires from m_n. Otherwise, either $t \in D$ therefore t fires from m_n, or else $m_n \rightarrow_D M_{n+1} \rightarrow_t$ because $\forall t' \in D \,.\, \neg dependent(t, t')$. $\forall m' \in M_{n+1}$ the number of unexplored states from m' is less than the number of unexplored states from m_n, therefore at m' or some state reachable from m' we will pick a D with $t \in D$ or we will fire all enabled transitions and thus t will fire in a state reachable from m_n.

Therefore the proposition holds after firing $n+1$ transitions and by induction the proposition holds.

□

Theorem 11 (Representative paths (dependence sets))**.** *Assuming a finite state space, for any path* $\overline{F}$ *in the full state space search there exists a path in the reduced state space search* $\overline{R}$ *such that* $\overline{R}$ *is a representative of* $\overline{F}$ *(Definition 30 on page 106), i.e.* $\overline{R} \supseteq \overline{F}$.

Proof. We prove this theorem by a proof by induction over the depth i of the reduced state space search. $\overline{R_i}$ denotes a path where $|\overline{R_i}| = i$. At each depth we prove that there exists a path $\overline{R_i}$ where the transitions that are in $\overline{F}$ but not in $\overline{R_i}$, $\overline{Y_i} = \overline{F} \setminus \overline{R_i}$, can fire in some order from m_i, i.e. $\overline{Y_i} \vdash m_i \rightsquigarrow m_i'$.

Base case

In the base case of depth 0 through the reduced state space search, $\overline{R_0} \vdash m_0 \rightsquigarrow m_0$ and $\overline{Y_0} = \overline{F}$ can fire in some order from m_0 because $\overline{F}$ can fire in some order from m_0 in the full state space.

Inductive step

Suppose at depth n in the reduced state space search there exists a path $\overline{R_n}$ such that $\overline{R_n} \vdash m_0 \rightsquigarrow m_n$ and $\overline{Y_n} = \overline{F} \setminus \overline{R_n}$ can fire in some order from m_n. We prove that if $\overline{Y_n} \neq \emptyset$ then there exists a path $\overline{R_{n+1}}$ in the reduced state space with $\overline{R_{n+1}} \supset \overline{R_n}$ such that $\overline{R_{n+1}} \vdash m_0 \rightsquigarrow m_{n+1}$ and $\overline{Y_{n+1}} = \overline{F} \setminus \overline{R_{n+1}}$ can fire in some order from m_{n+1}. One of the following three cases must happen:

- A dependence set D is chosen where $D \cap \overline{Y_n} = \emptyset$. Trivially $\forall t \in D \,.\, m_n \rightarrow_t m_{n+1}$, $\overline{R_{n+1}} = \overline{R_n} \cup t$ and $\overline{Y_{n+1}}$ can fire from m_{n+1}, because $\forall t \in D \,.\, \forall t' \in \overline{Y_n}.\neg dependent(t, t')$.

- A dependence set D is chosen where $D \cap \overline{Y_n} \neq \emptyset$. We know there is an ordering of $\overline{Y_n}$ that can fire from m_n. Let t be the transition in $D \cap \overline{Y_n}$ that fires first in an ordering of $\overline{Y_n}$ that can fire from m_n. Because t is independent of all transitions that fire before t in the ordering of $\overline{Y_n}$, t can fire from m_n and $\overline{Y_{n+1}}$ can fire from m_{n+1}.

- There is no dependence set D such that $\forall t \in D \,.\, m \rightarrow_t$ and $\exists t \in D \,.\, m \rightarrow_t m'$ and m' is an unseen state. Therefore, all enabled transitions in the Petri net fire from m_n and because $\overline{Y_n}$ can fire in some order from m_n, we are guaranteed to fire a transition t such that $\overline{R_{n+1}} = \overline{R_n} \cup t$ and $\overline{Y_{n+1}}$ can fire in some order from m_{n+1}.

Concluding remark

Consider a state m_i where $\overline{Y_i} \neq \emptyset$ can fire in some order from m_i. There is a state m_j with $j \geq i$ reachable from m_i where $\overline{Y_j}$ can fire in some order and a transition $t \in \overline{Y_j}$ will fire, because there will be no more transitions outside of $\overline{Y_j}$ reaching an unseen state (i.e. there is a finite number of unseen sates and at some point we will have exhausted them). Repeating this argument, we will reach a state m_k where $\overline{Y_k} = \emptyset$ and there is a path $\overline{R_k} \vdash m_0 \rightsquigarrow m_k$ such that $\overline{R_k} \supseteq \overline{F}$. □

7.3.5 The RMP using dependence sets algorithm

We now introduce the *RMP using dependence sets algorithm*, dsRMP($\mathcal{M}, G, A$). The algorithm is an adaption of the (original) RMP algorithm to follow a reduced state space search using dependence sets. Recall that reaction minimal paths are paths through the state space to a goal set without using an avoid set. In this section we introduce the algorithm and prove correctness, i.e. that the algorithm finds all reaction minimal paths. Finally, we demonstrate the algorithm with an example.

Algorithm The RMP using dependence sets algorithm for a k-bounded Petri net $\mathcal{M} = (T, P, f, m_0)$ is dsRMP($\mathcal{M}, G, A$) as follows.

Pre-processs: To make all paths satisfy the avoid constraint, we remove any transition that has a pre- or post-place in the avoid set: $T^* = \{t \in T \mid (\bullet t \cup t\bullet) \cap A = \emptyset\}$.

Dependence sets: Compute the set of dependence sets using T^* as per the algorithm in Section 7.3.3.

The algorithm for computing stages assuming a pre-processed network works as follows.

Recall from Definition 27 on page 88 that a tuple $(m, \overline{R})$ is subsumed by another tuple $(m', \overline{R'})$ if $\overline{R} \supseteq \overline{R'}$, and $m = m'$ or $m' \models G$.

Stage 0:
Stage(0) = $\{(m_0, \emptyset)\}$
$Paths = \emptyset$
Stage n: Given Stage(n-1)
Stage(n) = $\emptyset$
for $(m, \overline{R}) \in$ Stage(n-1) **do**
  if $m \models G$ **then**
    Add $process(\overline{R})$ to $Paths$
  else
    Let D_i = smallest dependence set such that $\forall t \in D_i \,.\, m \rightarrow_t$ and $\exists t \in D_i \,.\, m \rightarrow_t m'$ and $(m', Add(\overline{R}, t))$ is not subsumed.
    if D_i exists **then**
      $trans = D_i$
    else
      $trans = \{t \in T^* \mid m \rightarrow_t\}$
    end if
    for $t \in trans$ and $m \rightarrow_t m'$ **do**
      $\overline{R'} = Add(\overline{R}, t)$
      Add $(m', \overline{R'})$ to Stage(n) if it is not subsumed by a tuple in Stage(j) for some $j < n$
    end for
  end if
end for
if Stage(n) == $\emptyset$ **then**

Return *Paths*
end if

Enforcing the reaction minimal property Theorem 11 guarantees that for each reaction minimal path in the full state space search there exists a representative path in the reduced state space search. Therefore the paths in *Paths* are representative paths—the reaction minimal property is *not* guaranteed. Representative paths require further processing to guarantee the reaction minimal property because they are possibly an extension of reaction minimal paths. We enforce the reaction minimal property using the approach outlined in Section 7.2.3. For each representative path, we apply the reaction minimal path algorithm from Section 6.5.1 using only transitions from the path. This turns the representative path into a reaction minimal path.

$process(\overline{R})$ =RMP$((T', P, f, m_0), G, A)$ where $T' = \{t \mid t \in \overline{R}\}$

Theorem 12 (The RMP using dependence sets algorithm is correct)**.** *The RMP using dependence sets algorithm is correct, i.e. it produces all reaction minimal paths.*

Proof. We know by Theorem 11 that for each reaction minimal path in the full state space search, there exists a representative path in the reduced state space search. We convert the representative paths into reaction minimal paths using the RMP algorithm which is correct (Theorem 5 on page 90), therefore this algorithm is correct. □

Example 18 Example of the RMP using dependence sets algorithm.

Consider the Petri net in Figure 7.14 with a goal set $\{G\}$ and avoid set $\emptyset$. We compute the set of reaction minimal paths with the RMP using dependence sets algorithm. The algorithm produces 3 stages as follows. The explanation of how the dependence sets are computed is given in Example 17.

The dependence sets in this model are $D_1 = \{r1\}$, $D_2 = \{r2\}$ and $D_3 = \{r3, r4\}$.

Stage 0: (AXF, $\emptyset$)

From state AXF, pick $D_1 = \{r1\}$. Transition $r1$ is fired producing state BXF.

Stage 1: (BXF, $\{r1\}$)

From state BXF, pick $D_2 = \{r2\}$. Transition $r2$ is fired producing state BYF.

	RMP	RMP using dependence sets
Erks–act–EgfRC PL v5 (144 paths)	Tuples: 618,861 Runtime: 50s	Tuples: 1,258 Runtime: <1s
Rela–act–Nuc PL v5 (39 paths)	Tuples: 171,237 Runtime: 9s	Tuples: 3,416 Runtime: 2s
Erks–act–EgfRC PL v6 (160 paths)	Tuples: 1,358,465 Runtime: 207s	Tuples: 1,471 Runtime: <1s
Rela–act–Nuc PL v6 (156 paths)	Not computable	Tuples: 401,178 Runtime: 419s

Table 7.3: The performance of the RMP using dependence sets algorithm for four Pathway Logic models. Tuples is the number of (state, path) tuples. Runtime is on a workstation with a 2.53GHz dual core processor with 4GB of memory. Not computable means no results were obtained within 24 hours.

Stage 2: (BYF, $\{r1, r2\}$)

From state BYF, pick $D_3 = \{r3, r4\}$. Transitions $r3$ and $r4$ are fired producing the same state, BYG.

Stage 3: (BYG, $\{r1, r2, r3\}$) (BYG, $\{r1, r2, r4\}$)

From state BYG, no more enabled transitions.

The set of *representative paths* is: $\{\{r1, r2, r3\}, \{r1, r2, r4\}\}$

The set of *reaction minimal paths* is: $\{\{r1, r3\}, \{r2, r4\}\}$

7.3.6 Pathway Logic results

We repeat the analysis from Section 7.2.5 and compare the performance of the RMP using dependence sets algorithm with the (original) RMP algorithm. The results of the new algorithm are given in Table 7.3.

In all cases the RMP using dependence sets algorithm performs well compared to the (original) RMP algorithm. Pre-processing with the Hide Edges algorithm does not affect performance, therefore we have omitted these results. The Rela activation model from version 6 of the knowledge base is now computable (as with the RMP using alternative stubborn sets algorithm in Section 7.2.6).

We now compare the results using the RMP using dependence sets algorithm with results using the RMP using alternative stubborn sets algorithm. While the state space reduction of stubborn sets is better in three of the four Pathway Logic models, the runtime of dependence sets is shorter in three of the four models. This reflects the simple and purely structural definition of dependency.

However, with the Rela model, the RMP using stubborn sets algorithm performs slightly better with a runtime of 317 seconds compared with 419 seconds. More significantly, the RMP using stubborn sets algorithm explores roughly half the number of tuples compared with the RMP using dependence sets algorithm.

7.3.7 Discussion

The state state space reduction using stubborn sets is better compared to dependence sets in three of the four Pathway Logic models. However, the runtime using dependence sets is shorter in three of the four Pathway Logic models. As discussed, our definition of dependency is an over approximation. It is possible that a definition of dependency that takes into account the set of reachable states would result in better reduction—however, this would be at the cost of runtime because the dependence partition would be recomputed at each state.

Another benefit from using dependence sets is that, because they are purely structural, they can be used to identify subnets in a model that cause the state space to become infeasible. These subnets can be fed back to the model design/curator in hopes of simplifying the model or using appropriate abstraction, thus generating a computationally tractable model. For example, the Petri net in Figure 7.17 could be a subnet in a model and would harm the performance of partial order reduction using dependence sets.

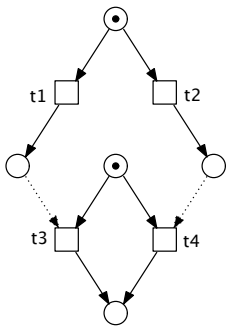


Figure 7.17: An example problematic subnet for dependence sets. Transitions $t3$ and $t4$ comprise a dependence set. Because $t3$ and $t4$ cannot both be enabled in the same reachable state, *when* either $t3$ or $t4$ fires, all transitions in the Petri net must fire.

7.4 Summary

In this chapter we extended the approach to understanding unstructured signalling networks to be applicable to *large* networks.

In Section 7.1 we gave the motivation for this chapter. In Section 7.2 we adapted the RMP algorithm to use two versions of stubborn sets partial order reduction. We introduced the RMP using stubborn sets and the RMP using alternative stubborn sets algorithms. We also introduced the Hide Edges algorithm which simplifies certain models. Even applying the Hide Edges algorithm, the RMP using stubborn sets algorithm was not significantly faster than the (original) RMP algorithm. The RMP using alternative stubborn sets algorithm performed well compared with the (original) RMP algorithm; the memory requirements for one model was three orders of magnitude less and the runtime was two orders of magnitude less. Furthermore, for a previously uncomputable model, results were returned after around 6 minutes. In Section 7.3 we introduced (the state independent) dependence sets partial order reduction. We adapted the RMP algorithm to use dependence sets partial partial order reduction, calling this algorithm the RMP using dependence sets algorithm. The performance of this algorithm compared well to the RMP using alternative stubborn sets algorithm. In three of four models the memory requirements of alternative stubborn sets is smaller, however in three of four models, the runtime of dependence sets is shorter. This reflects the simple nature of dependence sets.

In the next chapter we give future directions of this research.

Supplemental material An open-source Java application that computes all reaction minimal paths in Pathway Logic models (using the various partial order reduction algorithms) as well as the models used in this chapter can be found at **www.dcs.gla.ac.uk/~radonald/por2012/**. The Pathway Logic Assistant, knowledge bases and documentation can be found at **pl.csl.sri.com**.

Chapter 8

Future work

In this chapter we suggest some future directions of the work in this thesis.

Application to quantitative models Recall the distinction between qualitative and quantitative models from Section 4.2.3.

The modelling framework for cross-talk can, in principle, be applied to quantitative models. Because the underlying semantics of the framework is a CTMC with levels, all species in the model must have a consistent number of levels. Reaction rates can be included using the rates of the transitions in the Markov chain. As a result of using quantitative models, model checking can be used to measure the quantitative effect of cross-talk (e.g. the effect on rate of signal flow through the cell). There may be interesting ways to detect or characterise cross-talk using the rate of reactions.

We believe that the RMP algorithm can also, in principle, be applied to quantitative models. Currently the goal and avoid constraints reason about marked vs. unmarked places. These constraints can be extended to reason about the number of tokens on a place, e.g. the goal constraint $(X \geq 2 * Y)$ states that the number of tokens on X is at least double the number of tokens on Y. There may be interesting ways to categorise or reason about the set of signal flows using the rate of the reactions in the flows.

Specific drug targets We can use the approaches in this thesis to identify drug targets that control cellular signalling behaviour. Because we model cross-talk explicitly, we can measure any target's effect on unrelated pathways. One would expect that more specific targets (i.e. less effect on other pathways) would make better drug targets.

Formally represented biological data In Pathway Logic each reaction has a set of formally represented biological data, used as evidence for the reaction. No current analysis methods for cellular signalling models take advantage of such data. We believe that there is much to be gained by reasoning about this data, for example we could infer reactions from this data, generate evidence for algorithmic results and choose targets that are backed up with biological evidence.

Disease data We can use approaches in this thesis to generate hypothesis about cross-talk or signal flows that explain disease data. Most current models attempt to explain disease data using a single signalling pathway. Cross-talk between pathways, or signal flows in a network, could provide new hypotheses. The availability of high(er)-throughput protein measurements is a *current* limitation.

The Molecular Nose project data We have generated a hypotheses using the Pathway Logic framework and the RMP algorithm. We have created a model of the reactions from the Egf and IL1 ligands downstream to the activation of a set of transcription factors from the Pathway Logic knowledge base. The set of signal flows through the network was computed using the RMP algorithm. The signal flows from the Egf ligand turn on "immediate early genes" whereas the signal flows from the IL1 ligand turn on "late response" genes. From the literature we know that the Ngf ligand has two receptors in the well-studied PC12 cell line: the NgfR receptor which is similar to the IL receptor and the TrkA receptor which is similar to the Egf receptor. We have generated the following hypothesis, illustrated further in Figure 8.1.

Hypothesis: between 0 – 60 minutes, the response of Ngf looks like Egf, and after 60 minutes the response of Ngf looks like IL1.

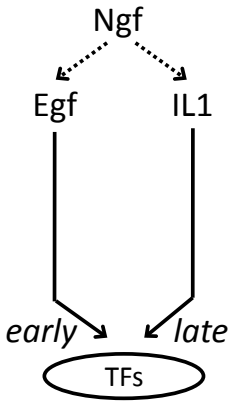


Figure 8.1: The Egf and IL1 ligands have an early and late response respectively. The Ngf ligand attaches to two receptors, one Egf-like and one IL1-like.

The Molecular Nose sensor is used to measure ("sniff out") the activity of transcription factors in a sample of cells. Currently the sensor can measure the activity of around 50 transcription factors. We can measure the transcription factor response to the following ligands: Egf, IL1, (the combination of) Egf&IL1, and Ngf. Comparing the transcription response of Egf to Ngf at early time points and IL1 to Ngf at late time points would validate/invalidate the above hypothesis. Finally, we can use this data to make models of the cross-talk between the Egf and IL1 pathways using the cross-talk modelling framework in this thesis.

Work has begun on experiments to prove/disprove the above hypothesis.

Chapter 9

Conclusion

This thesis is concerned with modelling and analysis of cellular signalling, which is an important area of study in systems biology. Cellular signalling is often studied using the abstraction to signalling pathways and cross-talk; however, both terms remain rather informal.

Our first contribution (Chapter 5) is a modular modelling framework for pathways and their cross-talk. This is the first modelling framework that has an explicit notion of cross-talk, expressed using different synchronisations of reactions between, and overlaps of, pathways. We gave a categorisation and formalisation of cross-talk and a modelling approach for cross-talk. We also gave model checking techniques to detect, characterise, and measure the effect of, cross-talk in a model. We demonstrated the framework using a prominent case study: the cross-talk between the TGF-β, WNT and MAPK pathways. The framework can be used to generate all pathway or network hypotheses given a suitable formalisation of permissible compositions. Our long term aim is to generate hypotheses to inform both systems and synthetic biology.

Our second contribution (Chapter 6) tackles the problem of unstructured signalling networks, i.e. networks with no explicit notion of pathways or cross-talk. We showed how signalling networks can be broken down into a set of signal flows, essentially a (minimal) multiset of reactions that work together to produce some output of interest. Current techniques to compute the set of signal flows are largely based on steady-state analysis. We have argued that steady-state analysis is appropriate for metabolic systems, but not cellular signalling systems which are concerned with transient flow of information through the cell. We reviewed current algorithms and showed them to be insufficient, either not guaranteeing completeness (generating all signal flows) or correctness (some signal flows can be incorrect). We then introduced the Reaction Minimal Paths (RMP) algorithm, the first algorithm to guarantee both correctness and completeness, and prove it correct. Then, we showed how to better understand signalling network models using the set of signal flows,

demonstrating this using Pathway Logic models. Finally, we showed how the set of signal flows can be used to compute several network metrics, and how clustering of signal flows can uncover structure within the network.

Our final contribution (Chapter 7) was to employ partial order reduction algorithms to improve the efficiency of the RMP algorithm. We started with two versions of the stubborn sets partial order reduction algorithm and then introduced the dependence sets algorithm. These algorithms have different computational complexities depending on the model being analysed. We also introduced the Hide Edges algorithm which simplifies certain categories of models, making them more amenable to reduction. An important result was that a previously incomputable model is now computable using partial order reduction.

Appendix A

Multisets

Definition 34 (Multiset)**.** *A multiset is a pair* (A, f) *where* A *is the underlying set of elements and* $f : A \to \mathbb{N}^+$ *is the (positive) multiplicity of each element in* A*. The multiplicity of* $a \in A$ *is* $f(a)$*.*

Given a multiset $M = (A, f)$:

- The elements of M are written $\{f(a_1) * a_1, \ \ldots, \ f(a_n) * a_n\}$ where $n = |A|$ and if $f(a) = 1$ then $f(a)*$ is omitted.
- The cardinality of M, written $|M|$, is $\sum_{a \in A} f(a)$.
- An element a belongs to M, written $a \in M$, if $a \in A$.
- An element a is added to M, written $Add(M, a)$, returning $M' = (A', f')$ where $A' = A \cup \{a\}$, $\forall b \in (A - \{a\}) \ . \ f'(b) = f(b)$ and if $a \in A$ then $f'(a) = f(a) + 1$ else $f'(a) = 1$.

Given two multisets $M_1 = (A_1, f_1)$ and $M_2 = (A_2, f_2)$:

- M_1 and M_2 are equivalent, written $M_1 = M_2$, if $A_1 = A_2$ and $\forall a \in A_1 \ . \ f_1(a) = f_2(a)$.
- M_1 is a submultiset of M_2, written $M_1 \subseteq M_2$, if $\forall a \in A_1 \ . \ (a \in A_2 \wedge f_1(a) \leq f_2(a))$.
- M_1 is a (proper) submultiset of M_2, written $M_1 \subset M_2$, if $M_1 \neq M_2$ and $M_1 \subseteq M_2$.
- $M_1 \cap M_2$ is the intersection of two multisets returning (A', f') where $A' = (A_1 \cap A_2)$ and $\forall a \in A' \ . \ f'(a) = min(f_1(a), f_2(a))$.

Appendix B

Breadth- vs. depth-first search

Breadth- and depth-first search are algorithms are used to explore the nodes in a graph—they differ in the order in which the nodes are explored.

Breadth-first search Starting at the initial node, each child is visited (in order from left-to-right). Then the children's children are visited, and so on, until all nodes have been explored. Hence, the nodes are visited in order of their depth from the initial node.

Depth-first search Starting at the initial node, the left-most unexplored child is visited, then that child's left-most unexplored child, and so on, until we reach a node with no more unexplored children. Then, the algorithm resumes from the nearest ancestor with unexplored children. Hence, the nodes are visited in order of their branches, from left-to-right, starting at the initial node.

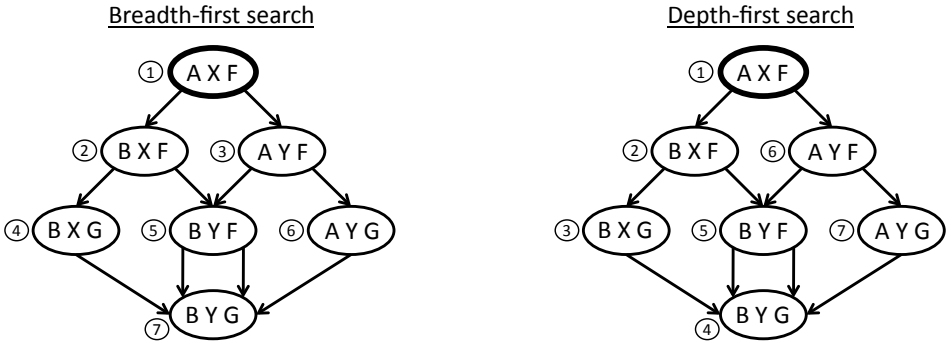


Figure B.1: The state space from Example 3 on page 27 explored using breadth- (left) and depth-first search (right). Numbers 1 ... 7 indicate the order in which the states are visited.

Appendix C

PRISM model of pathway cross-talk

The PRISM model of the example pathways and cross-talk from Sections 5.4 to 5.7 is given below. Note that we use the non-standard `pathway ... endpathway` construct. This allows us to define the system equations for our two example pathways, $pathway_1$ and $pathway_2$.

```
ctmc

module P1_Receptor
   R1 : [0..1] init 1;      L1 : [0..1] init 1;      R1Active : [0..1] init 0;

   [r1_1] R1 = 1 & L1 = 1 & R1Active = 0 -> 1:(R1' = 0) & (L1' = 0) & (R1Active' = 1);
   [r2_1] R1Active = 1 -> 1:true;

   [aux1_1] R1 = 1 -> 1:(R1' = 0);
   [aux2_1] R1Active = 0 -> 1:(R1Active' = 1);
   [aux3_1] L1 = 0 -> 1:(L1' = 1);
endmodule

module P1_Cascade3
   X1Inactive : [0..1] init 1;      X1Active : [0..1] init 0;
   Y1Inactive : [0..1] init 1;      Y1Active : [0..1] init 0;
   Z1Inactive : [0..1] init 1;      Z1Active : [0..1] init 0;

   [r3_1] X1Inactive = 1 & X1Active = 0 -> 1:(X1Inactive' = 0) & (X1Active' = 1);
   [r4_1] Y1Inactive = 1 & Y1Active = 0 & X1Active = 1 ->
      1:(Y1Inactive' = 0) & (Y1Active' = 1);
   [r5_1] Z1Inactive = 1 & Z1Active = 0 & Y1Active = 1 ->
      1:(Z1Inactive' = 0) & (Z1Active' = 1);
   [r6_1] Z1Active = 1 -> 1:true;
```

```
   [aux4_1] X1Active = 1 -> 1:true;
   [aux5_1] Y1Inactive = 1 -> 1:(Y1Inactive' = 0);
   [aux6_1] Y1Active = 0 -> 1:(Y1Active' = 1);
   [aux7_1] Z1Active = 0 -> 1:true;
endmodule

module P1_GeneExpression
   Gene1 : [0..1] init 1;      Protein1 : [0..1] init 0;

   [r7_1] Gene1 = 1 & Protein1 = 0 -> 1:(Gene1' = 0) & (Protein1' = 1);

   [aux8_1] Protein1 = 1 -> 1:(Protein1' = 0);
endmodule

module P2_Receptor = P1_Receptor
   [R1 = R2, L1 = L2, R1Active = R2Active, r1_1 = r1_2, r2_1 = r2_2, aux1_1 = aux1_2,
   aux2_1 = aux2_2, aux3_1 = aux3_2]
endmodule

module P2_Cascade3 = P1_Cascade3
   [X1Inactive = X2Inactive, X1Active = X2Active, Y1Inactive = Y2Inactive,
   Y1Active = Y2Active, Z1Inactive = Z2Inactive, Z1Active = Z2Active, r3_1 = r3_2,
   r4_1 = r4_2, r5_1 = r5_2, r6_1 = r6_2, aux4_1 = aux4_2, aux5_1 = aux5_2,
   aux6_1 = aux6_2, aux7_1 = aux7_2]
endmodule

module P2_GeneExpression = P1_GeneExpression
   [Gene1 = Gene2, Protein1 = Protein2, r7_1 = r7_2, aux8_1 = aux8_2]
endmodule

pathway
   ((P1_Receptor {r2_1 <- r3_1} |[r3_1]| P1_Cascade3) {r6_1 <- r7_1} |[r7_1]|
   P1_GeneExpression) / {r1_1, r3_1, r4_1, r5_1, R1, L1, R1Active, Gene1, Protein1}
endpathway

pathway
   ((P2_Receptor {r2_2 <- r3_2} |[r3_2]| P2_Cascade3) {r6_2 <- r7_2} |[r7_2]|
   P2_GeneExpression) / {r1_2, r3_2, r4_2, r5_2, R2, L2, R2Active, Gene2, Protein2}
endpathway
```

Appendix D

Relevant subnet algorithm

The relevant subnet algorithm Subnet(T, m_0, G, A) for a k-bounded Petri net with respect to G and A is a three step process as follows. Note that $out(t) = t^\bullet - {}^\bullet t$.

Remove avoids Remove all transitions in T containing a pre- or post-place in A.

$T' = \{t \in T \mid ({}^\bullet t \cup t^\bullet) \cap A = \emptyset\}$

Backward collection The backward collection T relative to G is a collection of all transitions that can put a token on a place in G or on a pre-place of a transition that, through a sequence of transitions, can put a token on G.

$T' = \bigcup_{j \in \mathbb{N}} T_j$ where

$G_0 = G$ $\qquad$ $G_{j+1} = G_j \cup (\bigcup_{t \in T_j} {}^\bullet t)$

$T_j = \{t \in T \mid (out(t) \cap G_j) \neq \emptyset\}$

There is a j such that $G_{j+1} = G_j$ since T is finite, hence the collection terminates.

Forward collection The forward collection of T relative to m_0 is a collection of all transitions that can fire from the initial state m_0.

$T' = \bigcup_{j \in \mathbb{N}} T_j$ $\qquad$ $I = \bigcup_{j \in \mathbb{N}} I_j$

$I_0 = \{p \in P \mid m_0(p) \geq 1\}$ $\qquad$ $I_{j+1} = \bigcup_{t \in T_j} t^\bullet$

$T_j = \{t \in T \mid {}^\bullet t \subseteq I_j\}$

There is a n such that $I_n = I_j$ for some $n > j$ since T is finite, hence the collection terminates.

Appendix E

Pathway Logic model diagrams

Below are the Pathway Logic model diagrams for the activation of ERKs (kb v5) and the activation of RelA (kb v5).

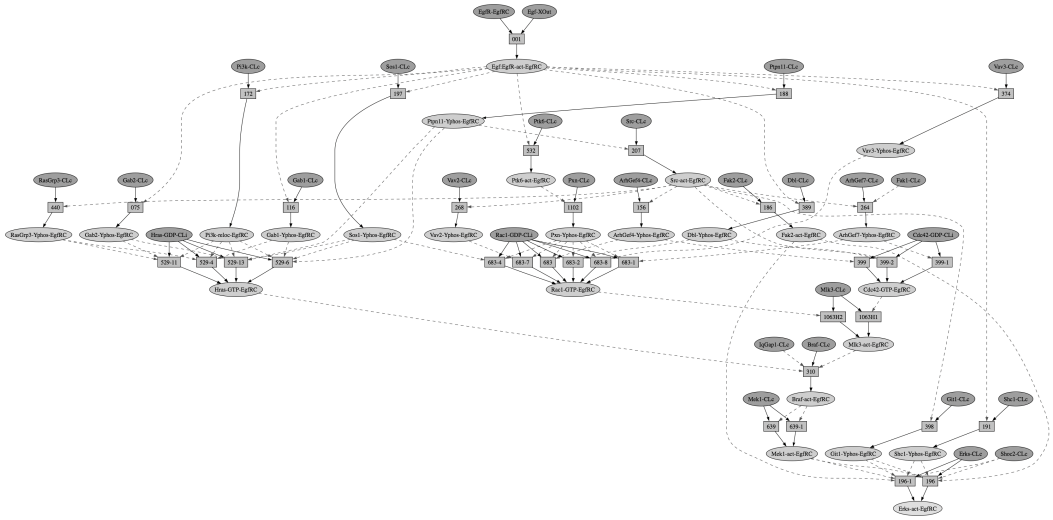


Figure E.1: The Pathway Logic model diagram for the activation of ERKs (kb v5).

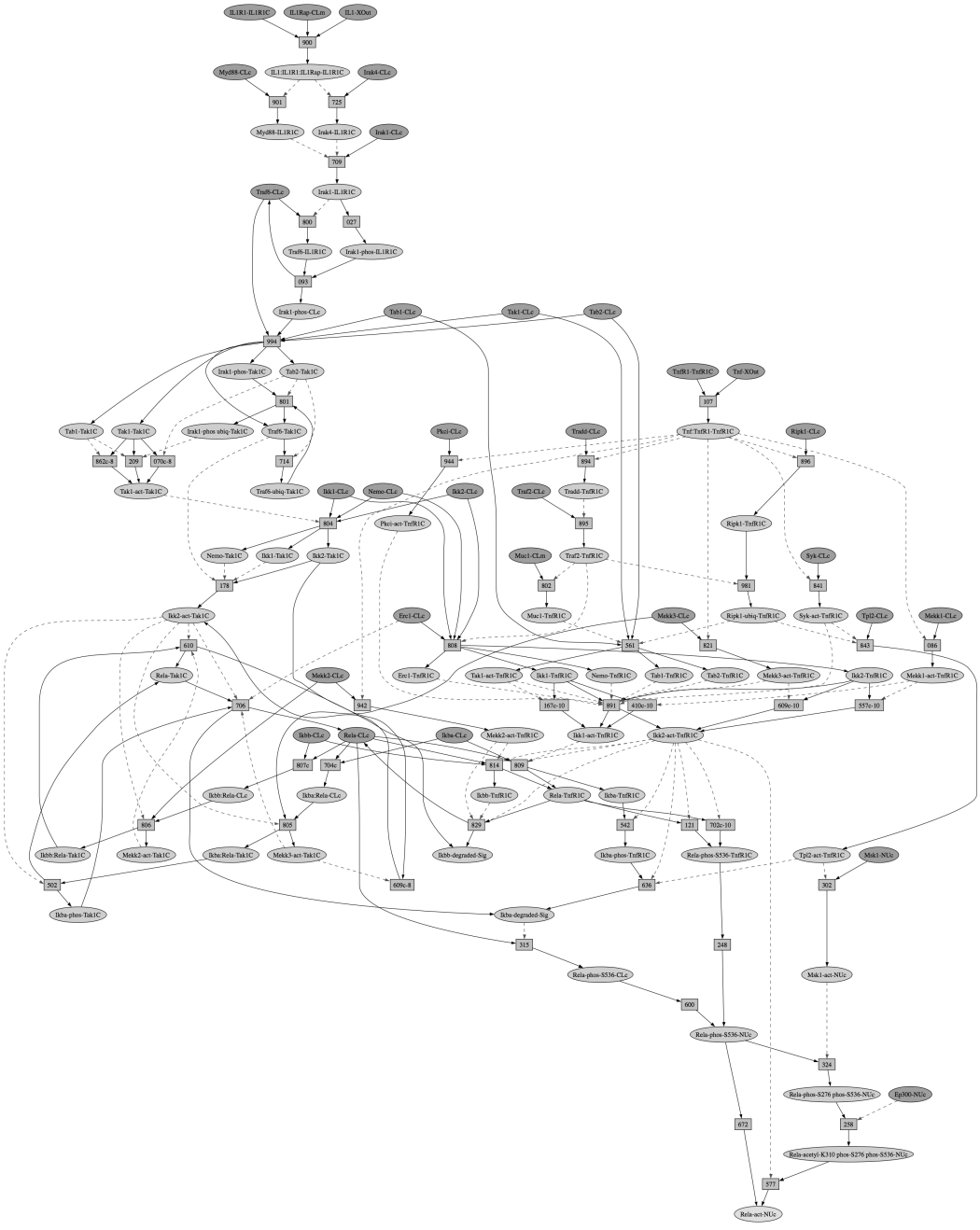


Figure E.2: The Pathway Logic model diagram for the activation of RelA (kb v5).

Appendix F

Signal flow cluster diagrams

Below are the cluster diagrams for the paths in the activation of RelA (kb v5) and the activation of ERKs (kb v5) models.

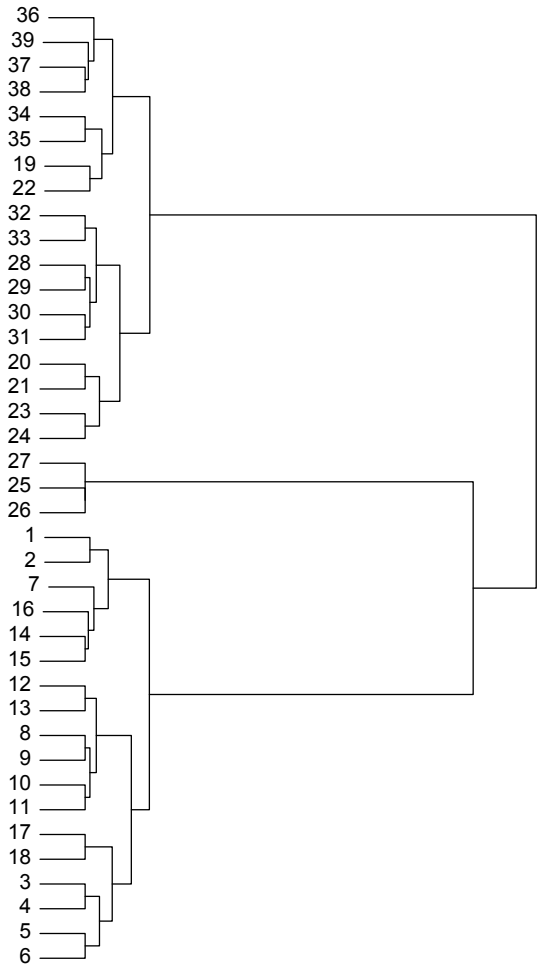


Figure F.1: The dendrogram from clustering the 39 paths in the activation of RelA (kb v5) model.

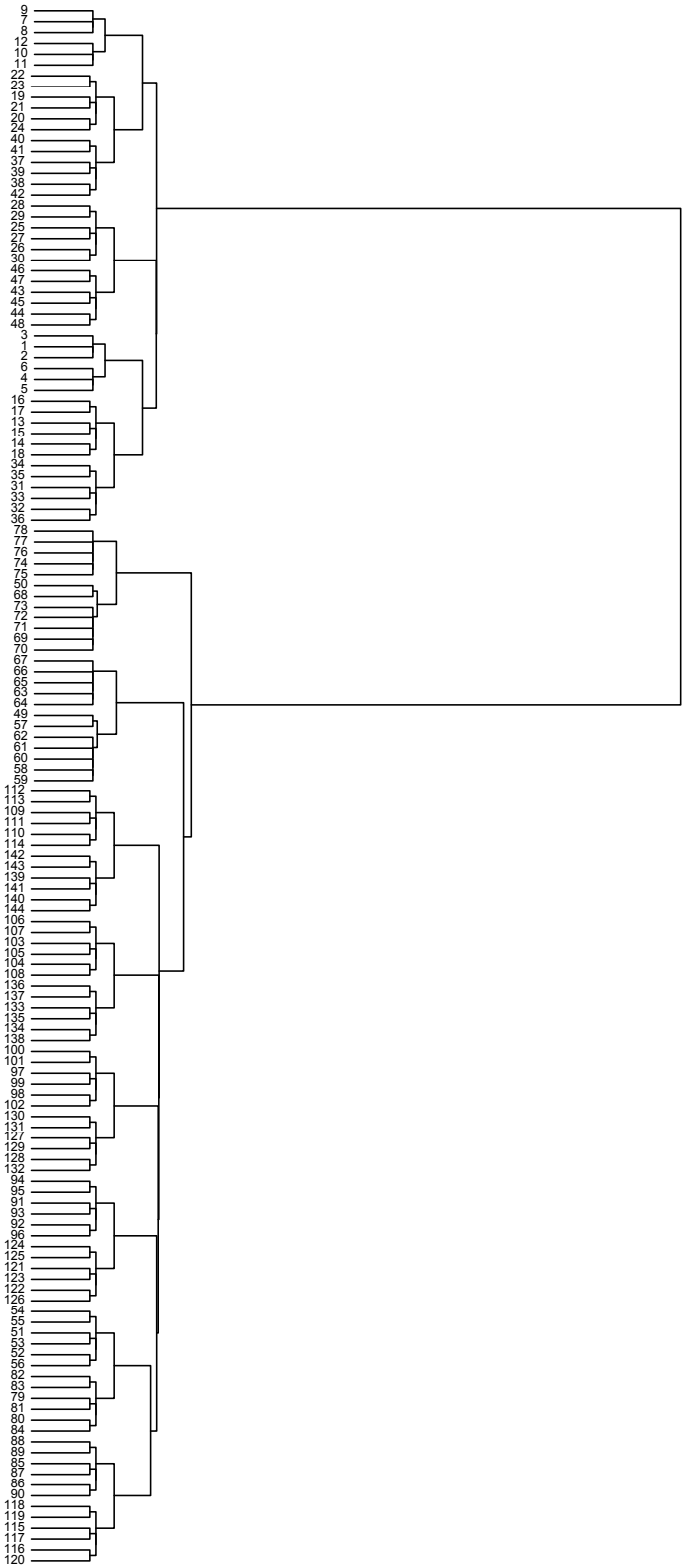


Figure F.2: The dendrogram from clustering the 144 paths in the activation of ERKs (kb v5) model.

Bibliography

[1] R. Alur and T. Henzinger. Reactive modules. *Formal Methods in System Design*, 15(1):7–48, 1999.

[2] M. Anzano, A. Roberts, C. Meyers, A. Komoriya, L. Lamb, J. Smith, and M. Sporn. Communication: Synergistic Interaction of Two Classes of Transforming Growth Factors from Murine Sarcoma Cells . *Cancer Research*, 42(11):4776–4778, 1982.

[3] F. Ay, F. Xu, and T. Kahveci. Scalable Steady State Analysis of Boolean Biological Regulatory Networks. *PLoS ONE*, 4(12):e7992, 2009.

[4] A. Aziz, K. Sanwal, V. Singhal, and R. Brayton. Verifying Continuous-Time Markov Chains. In Rajeev Alur and Thomas A. Henzinger, editors, *Computer Aided Verification*, volume 1102, pages 269–276. Springer-Verlag, 1996.

[5] M. Barrios-Rodiles, K. Brown, B. Ozdamar, R. Bose, Z. Liu, R. Donovan, F. Shinjo, Y. Liu, J. Dembowy, I. Taylor, et al. High-throughput mapping of a dynamic signaling network in mammalian cells. *Science*, 307(5715):1621–1625, 2005.

[6] L. Blair and J. Pang. Feature Interactions – Life Beyond Traditional Telephony. In M. Calder and E. Magill, editors, *Feature Interactions in Telecommunications and Software Systems VI*, pages 83–93. IOS Press (Amsterdam), 2000.

[7] D. Bosnacki and G. J. Holzmann. Improving Spin's Partial-Order Reduction for Breadth-First Search. In *SPIN*, volume 3639 of *Lecture Notes in Computer Science*, pages 91–105. Springer-Verlag, 2005.

[8] F. Brightman and D. Fell. Differential feedback regulation of the MAPK cascade underlies the quantitative differences in EGF and NGF signalling in PC12 cells. *FEBS Letters*, 482(3):169–174, 2000.

[9] J. Burch, E. Clarke, K. McMillan, D. Dill, and L. Hwang. Symbolic model checking: 10^{20} states and beyond. *Information and Computation*, 98(2):142–170, 1992.

[10] M. Calder, S. Gilmore, and J. Hillston. Modelling the influence of RKIP on the ERK signalling pathway using the stochastic process algebra PEPA. *Transactions on Computational Systems Biology VII*, 4230:1–23, 2006.

[11] M. Calder, S. Gilmore, J. Hillston, and V. Vyshemirsky. Formal methods for biochemical signalling pathways. In P. Boca, J. P. Bowen, and J. Siddiqi, editors, *Formal Methods: State of the Art and New Directions*, pages 185–215. Springer, 2009.

[12] M. Calder and J. Hillston. Process algebra modelling styles for biomolecular processes. *Transactions on Computational Systems Biology XI*, 5750:1–25, 2009.

[13] M. Calder, M. Kolberg, E. Magill, and S. Reiff-Marganiec. Feature Interaction: A Critical Review and Considered Forecast. *Computer Networks*, 41(1):115–141, 2003.

[14] M. Calder and A. Miller. Feature interaction detection by pairwise analysis of LTL properties – A case study. *Formal Methods in System Design*, 28(3):213–261, 2006.

[15] M. Calder, V. Vyshemirsky, R. Orton, and D. Gilbert. Analysis of Signalling Pathways using Continuous Time Markov Chains. *Transactions on Computational Systems Biology VI*, 4220:44–67, 2006.

[16] L. Calzone, N. Chabrier-Rivier, F. Fages, and S. Soliman. Machine learning biochemical networks from temporal logic properties. *Transactions on Computational Systems Biology*, 4220:68–94, 2006.

[17] A. Carracedo and P. Pandolfi. The PTEN–PI3K pathway: of feedbacks and cross-talks. *Oncogene*, 27(41):5527–5541, 2008.

[18] I. Catt. Crosstalk (Noise) in Digital Systems. *Electronic Computers, IEEE Transactions on*, EC-16(6):743–763, 1967.

[19] N. Chabrier-Rivier, M. Chiaverini, V. Danos, F. Fages, and V. Schächter. Modeling and Querying Biomolecular Interaction Networks. *Theoretical Computer Science*, 325(1):25–44, 2004.

[20] S. Chapman and A. Asthagiri. Quantitative effect of scaffold abundance on signal propagation. *Molecular Systems Biology*, 5(1), 2009.

[21] Charlie Website. Charlie – Petri Net Analyser. `http://www-dssz.informatik.tu-cottbus.de/software/charlie.html`.

[22] L. Chen, G. Qi-Wei, M. Nakata, H. Matsuno, and S. Miyano. Modelling and simulation of signal transductions in an apoptosis pathway by using timed Petri nets. *Journal of Biosciences*, 32(1):113–127, 2007.

[23] K. Cho, S. Shin, H. Kim, O. Wolkenhauer, B. Mcferran, and W. Kolch. Mathematical modeling of the influence of RKIP on the ERK signaling pathway. In C. Priami, editor, *Computational Methods in Systems Biology (CSMB03)*, volume 2602, pages 127–141. Springer-Verlag, 2003.

[24] F. Ciocchetta, A. Degasperi, J. K. Heath, and J. Hillston. Modelling and Analysis of the NF-κB Pathway in Bio-PEPA. *Transactions on Computational Systems Biology XII*, 5945:229–262, 2010.

[25] F. Ciocchetta, A. Degasperi, J. Hillston, and M. Calder. Some Investigations Concerning the CTMC and the ODE Model Derived From Bio-PEPA. *Electronic Notes in Theoretical Computer Science*, 229(1):145–163, 2009.

[26] F. Ciocchetta, A. Duguid, and M. L. Guerriero. A compartmental model of the cAMP/PKA/MAPK pathway in Bio-PEPA. In G. Ciobanu, editor, *MeCBIC*, volume 11, pages 71–90, 2009.

[27] E. Clarke, O. Grumberg, and D. Peled. *Model Checking*. MIT Press 1999, third printing, 2001.

[28] M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, and C. Talcott. *All About Maude: A High-Performance Logical Framework*. Springer-Verlag, 2007.

[29] G. Cooper and R. Hausman. *The Cell, A Molecular Approach, 5th Edition*. Sinauer Associates Inc, 2009.

[30] V. Danos, J. Feret, W. Fontana, R. Harmer, and J. Krivine. Rule-based Modelling of Cellular Signalling. In *Proceedings of CONCUR'07, Lecture Notes In Computer Science*, volume 4703, pages 17–41, 2007.

[31] V. Danos, J. Feret, W. Fontana, R. Harmer, and J. Krivine. Rule-based modelling and model perturbation. *Transactions on Computational Systems Biology XI*, pages 116–137, 2009.

[32] K. De Bosscher, W. Vanden Berghe, and G. Haegeman. Cross-talk between nuclear receptors and nuclear factor κb. *Oncogene*, 25(51):6868–6886, 2006.

[33] P. Degano, D. Prandi, C. Priami, and P. Quaglia. Beta-binders for Biological Quantitative Experiments. *Electronic Notes in Theoretical Computer Science*, 164(3):101–117, 2006.

[34] E. Dejardin. The alternative NF-κB pathway from biochemistry to biology: pitfalls and promises for future drug development. *Biochemical Pharmacology*, 72(9):1161–1179, 2006.

[35] R. Donaldson and D. Gilbert. A model checking approach to the parameter estimation of biochemical pathways. In M. Heiner and A. M. Uhrmacher, editors, *CMSB*, volume 5307 of *Lecture Notes in Computer Science*, pages 269–287. Springer-Verlag, 2008.

[36] A. Dutt, S. Canevascini, E. Froehli-Hoier, and A. Hajnal. EGF signal propagation during *C. elegans vulval* development mediated by ROM-1 Rhomboid. *PLoS Biology*, 2:1799–1814, 2004.

[37] S. Eker, M. Knapp, K. Laderoute, P. Lincoln, J. Meseguer, and K. Sonmez. Pathway logic: Symbolic analysis of biological signaling. *Proceedings of the Pacific Symposium on Biocomputing, 400–412*, 2002.

[38] N. Fernandez, A. Lozier, C. Flament, P. Ricciardi-Castagnoli, D. Bellet, M. Suter, M. Perricaudet, T. Tursz, E. Maraskovsky, and L. Zitvogel. Dendritic cells directly trigger NK cell functions: Cross-talk relevant in innate anti-tumor immune responses in vivo. *Nature Medicine*, 5:405–411, 1999.

[39] O. Fiehn. Metabolic networks of cucurbita maxima phloem. *Phytochemistry*, 62(6):875–886, 2003.

[40] J. Fisher and T. A. Henzinger. Executable cell biology. *Nature Biotechnology*, 25:1239–1249, 2007.

[41] J. Fisher, N. Piterman, A. Hajnal, and T. A. Henzinger. Predictive Modeling of Signaling Crosstalk during *C. elegans Vulval* Development. *PLoS Computational Biology*, 3(5):92+, 2007.

[42] P. Georgatsos, T. Nauta, and H. Velthuijsen. Role of service management in service interaction handling in an in environment. In P. Dini, R. Boutaba, and L. Logrippo, editors, *FIW*, pages 213–225. IOS Press, 1997.

[43] D. Gilbert and M. Heiner. From Petri Nets to Differential Equations – An Integrative Approach for Biochemical Network Analysis. In *ICATPN*, pages 181–200, 2006.

[44] D. Gilbert, M. Heiner, and S. Lehrack. A unifying framework for modelling and analysing biochemical pathways using Petri nets. In *Proceedings of CMSB 2007*, pages 200–216. Lecture Notes in Computer Science 4695, Springer, 2007.

[45] D. Gillespie. Exact stochastic simulation of coupled chemical reactions. *The Journal of Physical Chemistry*, 81(25):2340–2361, 1977.

[46] P. Godefroid, J. van Leeuwen, J. Hartmanis, G. Goos, and P. Wolper. *Partial-order methods for the verification of concurrent systems: an approach to the state-explosion problem*, volume 1032. Springer-Verlag, 1996.

[47] E. Grafahrend-Belau, F. Schreiber, M. Heiner, A. Sackmann, B. H. Junker, S. Grunwald, A. Speer, K. Winder, and I. Koch. Modularization of biochemical networks based on classification of Petri net T-invariants. *BMC Bioinformatics*, 9, 2008.

[48] M. Guerriero. Qualitative and quantitative analysis of a bio-pepa model of the gp130/jak/stat signalling pathway. *Transactions on Computational Systems Biology XI*, pages 90–115, 2009.

[49] J. Gunawardena. Signals and Systems: Towards a Systems Biology of Signal Transduction. *Proceedings of the IEEE*, 96(8):1386 – 1397, 2008.

[50] X. Guo and X. Wang. Signaling cross-talk between TGF-β/BMP and other pathways. *Cell Research*, 19:71–88, 2009.

[51] S. Guptaa, S. S. Bishta, R. Kukretia, S. Jainb, and S. K. Brahmacharia. Boolean network analysis of a neurotransmitter signaling pathway. *Journal of Theoretical Biology*, 244:463–469, 2007.

[52] R. J. Hall. Feature Interactions in Electronic Mail. In M. Calder and E. Magill, editors, *Feature Interactions in Telecommunications and Software Systems VI*, pages 67–82. IOS Press (Amsterdam), 2000.

[53] M. Hatakeyama, S. Kimura, T. Naka, T. Kawasaki, N. Yumoto, M. Ichikawa, J. Kim, K. Saito, M. Saeki, M. Shirouzu, S. Yokoyama, and A. Konagaya. A computational model on the modulation of mitogen-activated protein kinase (MAPK) and Akt pathways in heregulin-induced ErbB signalling. *Biochemical Journal*, 373 Pt. 2:451–463, 2003.

[54] H. Hatzikirou, D. Basanta, M. Simon, K. Schaller, and A. Deutsch. 'Go or Grow': the key to the emergence of invasion in tumour progression? *Mathematical Medicine and Biology*, 2010.

[55] M. Heiner, D. Gilbert, and R. Donaldson. Petri Nets in Systems and Synthetic Biology. In *Schools on Formal Methods (SFM)*, pages 215–264. Springer Lecture Notes in Computer Science 5016, 2008.

[56] M. Heiner and I. Koch. Petri net based model validation in systems biology. In *ICATPN, 216–237*, 2004.

[57] M. Heiner, I. Koch, and J. Will. Model validation of biological pathways using Petri nets–demonstrated for apoptosis. *Journal BioSystems*, 75:15–28, 2004.

[58] D. J. Higham. Modeling and Simulating Chemical Reactions. *SIAM Rev.*, 50(2):347–368, 2008.

[59] J. Hillston. *A Compositional Approach to Performance Modelling*. PhD thesis, Cambridge University, 1996.

[60] W. S. Hlavacek, J. R. Faeder, M. L. Blinov, R. G. Posner, M. Hucka, and W. Fontana. Rules for modeling signal-transduction systems. *Science STKE*, 344, July 2006.

[61] G. J. Holzmann. *The SPIN Model Checker: Primer and Reference Manual*. Addison-Wesley Professional, 2003.

[62] D. Hong and S. Man. Signal propagation in small-world biological networks with weak noise. *Journal of Theoretical Biology*, 262(2):370–380, 2010.

[63] M. Jarpe, C. Widmann, C. Knall, T. Schlesinger, S. Gibson, T. Yujiri, G. Fanger, E. Gelfand, and G. Johnson. Anti-apoptotic versus pro-apoptotic signal transduction: checkpoints and stop signs along the road to death. *Oncogene*, 17(11):1475–1482, 1998.

[64] M. R. Junttila, S. Li, and J. Westermarck. Phosphatase-mediated crosstalk between MAPK signaling pathways in the regulation of cell survival. *FASEB Journal*, 22:954–965, 2008.

[65] B. Katzenellenbogen. Estrogen receptors: bioactivities and interactions with cell signaling pathways. *Biol. Reprod.*, 54:287–293, 1996.

[66] U. Kaupp, J. Solzin, E. Hildebrand, J. Brown, A. Helbig, V. Hagen, M. Beyermann, F. Pampaloni, and I. Weyand. The signal flow and motor response controling chemotaxis of sea urchin sperm. *Nature Cell Biology*, 5(2):109–117, 2003.

[67] D. Kim, O. Rath, W. Kolch, K. H. Cho, and and. A hidden oncogenic positive feedback loop caused by crosstalk between wnt and erk pathways. *Oncogene*, 2007.

[68] H. Kitano. Systems biology: a brief overview. *Science*, 295(5560):1662–1664, 2002.

[69] L. M. Kristensen, K. Schmidt, and A. Valmari. Question-guided stubborn set methods for state properties. *Formal Methods in System Design*, 29(3):215–251, 2006.

[70] M. Kwiatkowska. Model Checking for Probability and Time: From Theory to Practice. In *Proceedings of 18th Annual IEEE Symposium on Logic in Computer Science (LICS'03)*, pages 351–360. IEEE Computer Society Press, 2003.

[71] M. Kwiatkowska, G. Norman, and D. Parker. PRISM: Probabilistic Symbolic Model Checker. In *Computer Performance Evaluation / TOOLS*, pages 200–204, 2002.

[72] M. Kyung, K. Hyun, S. Park, and S. Yoo. Crosstalk between Metabolic and Regulatory Pathways. *Genome Informatics*, 14:372–373, 2003.

[73] T. Lee, N. Rinaldi, F. Robert, D. Odom, Z. Bar-Joseph, G. Gerber, N. Hannett, C. Harbison, C. Thompson, I. Simon, et al. Transcriptional Regulatory Networks in *Saccharomyces cerevisiae*. *Science*, 298(5594):799–804, 2002.

[74] F. Li, I. Thiele, N. Jamshidi, and B. Palsson. Identification of Potential Pathway Mediation Targets in Toll-like Receptor Signaling. *PLoS Computational Biology*, 5(2):e1000292+, 2009.

[75] H. Lodish, A. Berk, S. L. Zipursky, P. Matsudaira, D. Baltimore, and J. Darnell. *Molecular Cell Biology*. New York : W. H. Freeman and Co., 2003.

[76] M. McClean, A. Mody, J. R. Broach, and S. Ramanathan. Cross-talk and decision making in MAP kinase pathways. *Nature Genetics*, 2007.

[77] I. Merelli, D. Pescini, E. Mosca, P. Cazzaniga, C. Maj, G. Mauri, and L. Milanesi. Grid computing for sensitivity analysis of stochastic biological models. In V. Malyshkin, editor, *PaCT*, volume 6873 of *Lecture Notes in Computer Science*, pages 62–73. Springer-Verlag, 2011.

[78] J. Meseguer. Conditional Rewriting Logic as a Unified Model of Concurrency. *Theoretical Computer Science*, 96(1):73–155, 1992.

[79] R. Milner. *Communicating and Mobile Systems: the Pi-Calculus*. Cambridge University Press, 1st edition, 1999.

[80] Molecular Nose Website. The Molecular Nose Project. `http://www.molecularnose.org`.

[81] NIH. "Insulin Signaling And Receptor Cross-Talk" Program Announcement PA-07-058. November 20, 2006.

[82] K. Oda and H. Kitano. A comprehensive map of the toll-like receptor signaling network. *Molecular Systems Biology*, 2, 2006.

[83] J. Orth, I. Thiele, and B. Palsson. What is flux balance analysis? *Nature Biotechnology*, 28(3):245–248, 2010.

[84] R. Orton, O. Sturm, V. Vyshemirsky, M. Calder, D. Gilbert, and W. Kolch. Computational modelling of the receptor tyrosine kinase activated MAPK pathway. *Biochemical Journal*, 392(2):249–261, 2005.

[85] J. Papin, N. Price, S. Wiback, D. Fell, and B. Palsson. Metabolic pathways in the post-genome era. *Trends in Biochemical Sciences*, 28(5):250–258, 2003.

[86] R. Pelánek. In Fighting State Space Explosion: Review and Evaluation. Formal Methods for Industrial Critical Systems, 37–52. Springer-Verlag, Berlin, Heidelberg, 2009.

[87] D. Peled. All from One, One for All: on Model Checking Using Representatives. In *Proceedings of the 5th International Conference on Computer Aided Verification*, CAV '93, pages 409–423. Springer-Verlag, 1993.

[88] C. Petri. *Communication with Automata (in German)*. Schriften des Instituts für Instrumentelle Mathematik, Bonn, 1962.

[89] L. C. Platanias and E. N. Fish. Signaling pathways activated by interferons. *Experimental Hematology*, 27(11):1583–1592, 1999.

[90] M. Plath and M. Ryan. The feature construct for SMV: Semantics. In M. Calder and E. Magill, editors, *Feature Interactions in Telecommunications and Software Systems VI*, pages 129–144. IOS Press (Amsterdam), 2000.

[91] C. Priami and P. Quaglia. Beta Binders for Biological Interactions. In *Computational Methods in Systems Biology*, pages 20–33. 2005.

[92] G. Querrec, V. Rodin, J. Abgrall, S. Kerdelo, and J. Tisseau. Uses of multiagents systems for simulation of mapk pathway. In *Proceedings of the Third IEEE Symposium on Bioinformatics and Bioengineering*, pages 421–425, 2003.

[93] A. Regev, W. Silverman, and E. Shapiro. Representation and simulation of biochemical processes using the pi-calculus process algebra. In R. Altman, A. Dunker, L. Hunter, and T. Klein, editors, *Pacific Symposium on Biocomputing, Volume 6*, pages 459–470, Singapore, 2001.

[94] S. Reiff. Identifying Resolution Choices for an Online Feature Manager. In M. Calder and E. Magill, editors, *Feature Interactions in Telecommunications and Software Systems VI*, pages 113–128. IOS Press (Amsterdam), 2000.

[95] A. Rizk, G. Batt, F. Fages, and S. Soliman. On a Continuous Degree of Satisfaction of Temporal Logic Formulae with Applications to Systems Biology. In *CMSB '08: Proceedings of the 6th International Conference on Computational Methods in Systems Biology*, pages 251–268. Springer-Verlag, 2008.

[96] K. Schmidt. LoLA: A Low Level Analyser. In M. Nielsen and D. Simpson, editors, *Application and Theory of Petri Nets, 21st International Conference (ICATPN 2000)*, volume 1825 of *Lecture Notes in Computer Science*, pages 465–474. Springer-Verlag, 2000.

[97] S. Schuster, T. Dandekar, and D. A. Fell. Detection of elementary flux modes in biochemical networks: a promising tool for pathway analysis and metabolic engineering. *Trends in Biotechnology*, 17(2):53–60, 1999.

[98] M. Schwartz and M. Ginsberg. Networks and crosstalk: integrin signalling spreads. *Nature Cell Biology*, 4(4):65–68, 2002.

[99] S. Shvartsman, M. Hagan, A. Yacoub, P. Dent, H. Wiley, and D. Lauffenburger. Autocrine loops with positive feedback enable context-dependent cell signaling. *American Journal of Physiology – Cell Physiology*, 282(3), 2002.

[100] S. Soliman. Finding minimal P/T-invariants as a CSP, May 2008.

[101] S. Sreenath, R. Soebiyanto, M. Mesarovic, and O. Wolkenhauer. Coordination of crosstalk between MAPK-PKC pathways: an exploratory study. *IET Systems Biology*, 1:33–40, 2007.

[102] C. Talcott. Symbolic modeling of signal transduction in Pathway Logic. In *WSC '06: Proceedings of the 38th conference on Winter simulation*, pages 1656–1665. Winter Simulation Conference, 2006.

[103] C. Talcott and D. Dill. Multiple representations of biological processes. *Transactions on Computational Systems Biology*, 2006.

[104] C. M. Taniguchi, B. Emanuelli, and R. C. Kahn. Critical nodes in signalling pathways: insights into insulin action. *Nature Reviews Molecular Cell Biology*, 7(2):85–96, 2006.

[105] O. Tymchyshyn and M. Kwiatkowska. Combining intra- and inter-cellular dynamics to investigate intestinal homeostasis. In *Proceedings of Formal Methods in Systems Biology (FMSB'08)*, volume 5054 of *Lecture Notes in Computer Science*. Springer-Verlag, 2008.

[106] J. J. Tyson. Modeling the cell division cycle: cdc2 and cyclin interactions. *Proceedings of the National Academy of Sciences of the United States of America*, 88(16):7328–7332, 1991.

[107] A. Valmari. Private correspondence, 2010-2011.

[108] O. Wolkenhauer. Systems Biology: the Reincarnation of Systems Theory Applied in Biology? *Briefings in Bioinformatics*, 2:258–270, 2001.