Qualitatively Modelling Genetic Regulatory Networks: Petri Net Techniques and Tools

Thesis by
Richard A Banks

In Partial Fulfillment of the Requirements
for the Degree of
Doctor of Philosophy

Newcastle University
Newcastle upon Tyne, UK

2009
(Defended March 17, 2009)

Contents

Acknowledgements

First and foremost, my parents Paul and Sue, as well as my sister Charlotte, have given me nothing but love and support throughout this work, and for that I am very grateful.

I would like to thank my supervisor Jason Steggles, who has gone above and beyond his duties as a supervisor, and has always made time for meetings to ensure that the work has run smoothly. Thanks also go to Neil Wipat and Victor Khomenko for many useful discussions, as well as Matthew Pockock, Keith Flanagan, Jennifer Hallinan and the rest of the bioinformatics group at Newcastle University.

Outside of work, I thank Colin and Sylvia Brown for many enjoyable meals, Simon Carter and Claire Hines for giving me somewhere to lodge during the last three months, the departmental basketball people for some interesting games and the infamous Lahore takeaway on Normanton Road in Derby for producing the best kebabs ever. I also thank the now-flattened cushion in the dining room for making the last year of write up a comfortable experience.

On a more serious note, my girlfriend Lejla, along with her mum and dad Lida and Ferid, have offered me much support, encouragement and tasty Yugoslavian snacks. For this I say hvala za hranu i smjestaj u vasoj kuci. I thank Lejla especially throughout all this for her love, understanding, and remarkable patience, as well as her nagging for me to work!

Abstract

The development of post-genomic technologies has led to a paradigm shift in the way we study *genetic regulatory networks (GRNs)* - the underlying systems which mediate cell function. To complement this, the focus is on devising scalable, unambiguous and automated *formal techniques* for holistically modelling and analysing these complex systems.

Quantitative approaches offer one possible solution, but do not appear to be commensurate with currently available data. This motivates *qualitative* approaches such as *Boolean networks (BNs)*, which abstractly model the system without requiring such a high level of data completeness. Qualitative approaches enable fundamental dynamical properties to be studied, and are well-suited to initial investigations. However, strengthened formal techniques and tool support are required if they are to meet the demands of the biological community.

This thesis aims to investigate, develop and evaluate the application of *Petri nets (PNs)* for qualitatively modelling and analysing GRNs. PNs are well-established in the field of computer science, and enjoy a number of attractive benefits, such a wide range of techniques and tools, which make them ideal for studying biological systems.

We take an existing qualitative PN approach for modelling GRNs based on BNs, and extend it to more general models based on *multi-valued networks (MVNs)*. Importantly, we develop tool support to automate model construction. We illustrate our approach with two detailed case studies on Boolean models for carbon stress in *Escherichia coli* and sporulation in *Bacillus subtilis*, and then consider a multi-valued model of the former. These case studies explore the analysis power of PNs by exploiting a range of techniques and tools.

A number of behavioural differences are identified between the two *E. coli* models which lead us to question their formal relationship. We investigate this by proposing a framework for reasoning about the behaviour of MVNs at different levels of abstraction. We develop tool support for practical models, and show a number of important results which motivate the need for multi-valued modelling.

Asynchronous BNs can be seen to be more biologically realistic than their synchronous counterparts. However, they have the drawback of capturing behaviour which is unrealisable in practice. We propose a novel approach for refining such behaviour using signal transition graphs, a PN formalism from asynchronous circuit design. We automate our approach, and demonstrate it using a BN of the lysis-lysogeny switch in phage λ. Our results show that a more realistic asynchronous model can be derived which preserves the stochastic switch.

Chapter 1

Introduction

The regulatory processes governing the growth and development of cellular organisms are so unimaginably complex that they will probably elude our understanding for many years to come. Despite some notable precursors, this understanding has only recently been accelerated by the tremendous developments in post-genomic technology of the ninetees [33]. Now in the 21st century, our goal is to unite research from the mathematical, computational and biological disciplines, to model organisms holistically at multiple levels of organisation, and to simulate their behaviour closely *in vivo* [112].

To achieve this goal, however, we must work from the ground up. Indeed, the American physicist Richard Feynman, who also had a deep interest in biology, once stated [66]:

> *"Everything is made of atoms. That is the key hypothesis. The most important hypothesis in all biology, for example, is that everything that animals do atoms can do. In other words, there is nothing that living things do that cannot be understood from the point of view that they are made of atoms acting according to the laws of physics."*

Thus, our understanding of complex cellular organisms first requires an understanding of the underlying processes which mediate their function.

At a high level, these processes can be conceptualised as interlinked control structures which work together in an orchestrated fashion, and are often referred to as *signal transduction pathways (STPs)* [170], *metabolic pathways (MPs)* [89] and *genetic regulatory networks (GRNs)* [206, 212]. STPs control how the cell receives, processes and responds to information from the environment, whereas MPs control the intake and usage of energy in the cell for it to perform its function. However, the general focus of this thesis is on GRNs, which control the complex process of gene expression. Specifically, GRNs control the synthesis of gene products (or proteins) by transcribing the genetic information coded by the gene into mRNA, and then translating this into the resulting protein.

An abstract example of a GRN for the *mammalian cell cycle* (taken from [65]) is shown in Figure 1.1. Specifically, interactions can be either activation (denoted by directed arrows) or inhibition (denoted by flat arrows). Even for this small system, we observe that incredibly rich behaviours can be manifested due to the tight integration of local interactions. This complexity increases further when we consider that gene expression can be regulated at numerous stages in the process, and synthesised proteins can

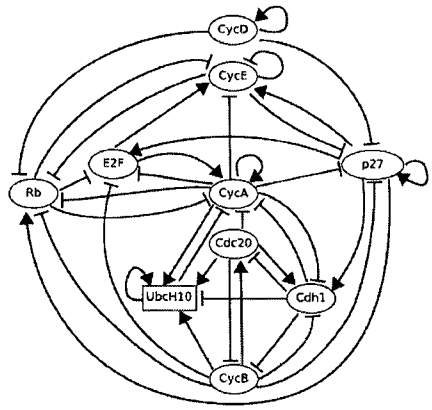


Figure 1.1: A high-level regulatory representation of the interactions present in the mammalian cell cycle network (taken from [65]), where directed edges represent activation interactions, whilst flat edges represent repression.

regulate other genes (including the gene from which they were synthesised), act as enzymes catalysing metabolic reactions or form components in a signal cascade [98]. Thus, there is an impending requirement for mathematically sound, unambiguous, scalable and systematic *formal techniques* which facilitate the study of the "sum of the parts".

Quantitative approaches offer one possible solution by facilitating realistic and detailed simulations, but are widely regarded as being incommensurate with currently available data [91]. This motivates *qualitative* approaches such as *Boolean networks (BNs)* [102], which abstractly model the system without requiring such a high level of data completeness. Qualitative approaches enable fundamental dynamical properties to be studied, and are well-suited to initial investigations. However, strengthened formal techniques and tool support are required if they are to meet the demands of the biological community.

In this thesis, we explore this issue by considering the *Petri net (PN)* formalism [147, 156]. PNs are well-established in the field of computer science, and enjoy a number of attractive benefits which make them a promising framework for studying biological systems. This leads us to state the high-level aim of this thesis.

> This thesis aims to investigate, develop and evaluate the application of Petri nets (PNs) to qualitatively model and analyse GRNs.

PNs are based on directed bipartite graphs (for example, see Figure 1.2(a)) consisting of: *places* (denoted by circles) which represent biological entities such as genes or proteins; *transitions* (denoted by rectangles) which represent events such as chemical reactions; *weighted arcs* (denoted by directed arrows) which connect together places and transitions; and *tokens* (denoted by black circles) which represent a quantity or concentration. Tokens can be moved around the places of the PN by *firing* the transitions [147]. A transition can fire if all its input places contain sufficient tokens (this is discussed more in Section 2.3).

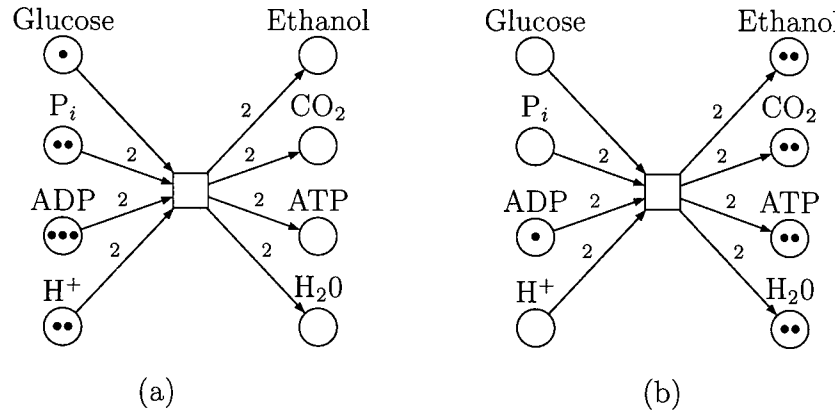


Figure 1.2: PNs modelling the conversion of glucose into ethanol $Glucose + 2P_i + 2ADP + 2H^+ \longrightarrow 2Ethanol + 2CO_2 + 2ATP + 2H_2O$ [80]: (a) initial marking representing quantities of each substance; and (b) subsequent marking after reaction.

For example, Figure 1.2(a) illustrates a PN modelling the well-known conversion of glucose into ethanol. The tokens present on its places represent the initial amounts of each substance in the reaction, and are sufficient so that the transition can fire. In this case, firing represents the conversion from glucose into ethanol, and the PN in Figure 1.2(b) shows the new token arrangement to reflect this.

A key advantage of PNs is that they are able to capture both the structural and dynamical information required to comprehensively model GRNs in a concise, executable and refinable way (we will explore this in more detail in Section 2.3). Furthermore, they are extensible with additional information (for example, time delays) whilst preserving the underlying net structure [44], thus supporting the incremental process of model development through both the qualitative and quantitative domains. Finally, PNs are a mature and developed formalism [1] with a wealth of techniques and tools to support the study of GRNs. Indeed, this thesis will explore how these key benefits can be exploited to strengthen qualitative techniques for modelling and analysing GRNs, with a particular emphasis on ensuring that these are available to the biological community.

1.1 Contributions

This thesis aims to investigate, develop and evaluate the application of PNs to qualitatively model and analyse GRNs. In particular, the contribution of this thesis is at a number of levels which we will now discuss.

Systematic PN framework for qualitative modelling GRNs

BNs [101, 102] are a logical formalism for qualitatively modelling GRNs which have received much attention in the literature [6, 7, 107, 203]. However, their analysis techniques and tools appear to require strengthening if they are to meet the demands of the biological community, and they do not cope well with the uncertainty that can plague practical data sets.

We address this in Chapter 3 by developing a systematic PN modelling framework based on BNs, and then generalise it in Chapter 4 by catering for MVNs [146, 206]. In particular, this work builds upon and strengthens similar approaches (for example, see [45,46]) in a number of important ways: (i) a range of PN representations for MVNs; (ii) the application of efficient *logic minimisation techniques* [146, 169] for compactness and scalability; (iii) catering for *both* synchronous and asynchronous MVNs in the PN environment; and (iv) coping with partial models. Importantly, we develop much-needed tool support to automate model construction. Thus, we contribute an important link which strengthens the availability of powerful PN techniques and tools to the biological community.

A further key contribution is made by considering the use of a PN formalism from asynchronous circuit design called *signal transition graphs (STGs)* [55]. STGs do not appear to have been applied before to GRNs, but offer a promising approach for the refinement of asynchronous models. In particular, we utilise STGs in a unique initial study which considers the refinement of asynchronous BNs.

Application and validation of qualitative techniques using case studies

We consider a number of detailed case studies which allow us to explore the application of our PN modelling approaches on practical systems. In particular, these case studies demonstrate the application of our tool support, as well as a number of existing PN analysis techniques and tools based on *unfolding* [109] and *model checking* [51].

Chapter 3 applies our Boolean PN framework to model part of the GRN for carbon stress response in *E. coli* [167]. We then consider a larger GRN for initiating sporulation in *B. subtilis* [99]. Both are well-studied in the literature, and this allows us to explore the correctness of our approach straightforwardly. Chapter 4 then applies our generalised PN framework by revisiting the *E. coli* GRN, and this highlights some interesting behavioural differences with the Boolean model which form the basis for a novel investigation in Chapter 5.

Chapter 6 applies our STG modelling approach to an asynchronous BN model of the lysis-lysogeny switch in phage λ [206]. We utilise powerful PN-based circuit analysis techniques in conjunction with our developed tool support to identify a number of key insights, and show how a more realistic asynchronous model can be developed which still captures the fundamental switch.

Relationship between models at different levels of abstraction

Case studies in Chapters 3 and 4 identify some key behavioural differences between the Boolean and multi-valued *E. coli* models. This raises a number of interesting questions concerning the scope and limitations of Boolean modelling, as well as when the extra expressive power of multi-valued modelling is required.

Chapter 5 presents an initial investigation by developing a framework for reasoning about the relationship between MVNs at different levels of abstraction. At its foundation is the proposal of a formal refinement theory capturing an assumption about this relationship, and we develop algorithms for its systematic application accordingly. This work highlights a number of important results which motivate the need for multi-valued

modelling, and sheds light into the initial behavioural differences noted between the two *E. coli* models. At a higher level, this work forms the foundation for promising future developments in biological model analysis.

Asynchronous qualitative model refinement techniques

Asynchronous BNs can be argued to be more biologically realistic than their synchronous counterparts [86]. However, such models have the problem of capturing too much information – some of which is unrealisable in practice - thus hampering analysis and interpretation. To address this, Chapter 6 proposes a novel application of PN techniques from asynchronous circuit design based on STGs [55]. We develop a systematic approach for incrementally developing realistic asynchronous BNs, and apply it to a case study on the phage λ lysis-lysogeny switch [206]. Our results show clear promise for the refinement of more generalised asynchronous models, and contribute a novel and interesting application of PN techniques which do not appear to have been explored before.

Integrated tool support

We place a strong emphasis on ensuring that the qualitative techniques developed are available to the biological community. As such, we implement a suite of integrated tools to enable the systematic construction and refinement of PN models.

1.2 Organisation of Thesis

This thesis is organised as follows.

Chapter 2 introduces BNs and PNs, and explores their application in the literature to biological modelling and analysis.

Chapter 3 builds upon an existing PN modelling framework for BNs by developing tool support to make it practical to the biological community, and by investigating its application to case studies on *E. coli* and *B. subtilis*.

Chapter 4 generalises this modelling framework to handle MVNs, and revisits a case study on *E. coli*, which raises some interesting questions concerning the relationship between MVNs at different levels of abstraction.

Chapter 5 formally investigates this relationship, and presents some important insights which motivate the MVN modelling framework of Chapter 4.

Chapter 6 develops a technique for incrementally developing realistic asynchronous BNs with a novel application of PN techniques from asynchronous circuit design.

Chapter 7 concludes this thesis by discussing the key insights gleaned, as well as future developments to take this work forward.

Chapter 2

Background

This chapter focuses primarily on the qualitative modelling and analysis of *genetic regulatory networks (GRNs)*, which have received much attention in the literature to date [98]. We start by briefly outlining some of the existing approaches that have been employed to model and analyse GRNs. We then focus on the key formalisms used throughout this thesis by introducing the background and theory of *Boolean networks (BNs)* [101, 102] and *Petri nets (PNs)* [147]. Finally, we review some of the success that PNs have enjoyed in the literature, and summarise a selection of available PN-based tools (amongst other approaches) for systematically modelling and analysing biological systems.

2.1 Existing modelling approaches

A wide variety of qualitative and quantitative techniques have emerged in the rapidly maturing area of systems biology (see [57, 98, 100] for comprehensive reviews). Here, we note some of the most prominent.

Boolean and multi-valued networks

BNs [57, 98, 101, 102, 137, 188] and their generalisation to *multi-valued networks (MVNs)* [172, 206, 208] model the underlying regulatory system discretely as a circuit of interconnected entities [139]. BNs in particular provide a suitable starting point for understanding complex systems, are scalable, and capture a number of properties which have been justified in the biological context [77, 86, 183, 188]. However, they appear to suffer from relatively limited analysis techniques and tools for comprehensively studying biological systems, and cannot cope straightforwardly with the uncertainty that is often present.

Petri nets

PNs [147] are a well-studied formalism for modelling concurrent, distributed systems that have received much attention in the biological community [44, 155, 160]. PNs offer a number of benefits to the biologist, including a graphical notation underpinned by a formal mathematical semantics, an ability to model and analyse biological systems at multiple levels of detail, and a wealth of available techniques and tool support. For these reasons

and more, PNs have been utilised to address the shortcomings of other approaches, such as BNs [45], and we will investigate this partnership in more detail throughout this thesis.

Bayesian networks

Bayesian networks [17, 67, 68, 148] are directed acyclic graphs of nodes representing random variables following a joint probability distribution [98]. Bayesian networks have solid foundations in statistics and probability theory, are well-suited for modelling complex stochastic processes and can cope with sparse and noisy data sets [182]. Furthermore, they allow for the discovery of multiple coherent models which best fit some experimental observation. However, they appear to be hampered by a number of shortcomings: (i) an inability to represent self-loops (i.e. the possibility of self-regulation); (ii) an NP-hard learning procedure for model construction; and (iii) an inherent static nature [182] which only appears to be addressed by extensions such as *dynamic Bayesian networks* [148], which result in further learning complexity. For these reasons, stochastic extensions to BNs [182] are often preferred.

Graph clustering

The idea of *graph clustering* approaches [26, 129, 175] is to group together regulatory entities whose behaviour over time is similar based on some metric. Regulatory entities belonging to the same cluster can therefore be viewed abstractly as a meta-entity and treated essentially as one functional unit. The problem is then to deduce the nature of the regulatory interactions between each meta-entity. Such clustering is therefore advantageous as it can reduce potentially large networks into smaller ones, aiding state space analysis and interpretation. However, clustering is computationally-intensive (often based on multiple heuristics) [181] and is usually employed in conjunction with other formalisms such as BNs [186] for analysis.

Piecewise-linear differential equations

To address the problems associated with constructing systems of quantitative differential equations with limited data, *piece-wise linear differential equations (PLDEs)* [23, 40, 71, 77, 99, 143] have been proposed. PLDEs are qualitative and use the notion of internally continuous, externally discrete variables which yield a number of desirable mathematical properties. In particular, PLDEs replace the non-linearity of sigmoidal curves found in traditional differential equations with discrete *step functions*. This discontinuous approximation has been justified based on the observation that genes often exhibit switch-like behaviour between different expression levels [105, 182, 203]. PLDEs allow the qualitative evolution of entity concentrations to be simulated through the phase space, and for a number of properties such as stability to be analysed. This formalism is also backed by tool support [21] which has been extended to allow for the possibility of more advanced analysis techniques such as *model checking* [22–25]. However, at present PLDEs appear to currently lack the mature and developed range of tools and techniques required for biological systems (although they could be modelled as PNs to address this), as well as an intuitive means of describing the dynamics and parameters of the system under study.

Stochastic π-calculus

Stochastic π-calculus [62, 162] provides a stochastic extension to a calculus developed for mobile communicating processes [144]. Based on a grammatical representation, this calculus allows for the incremental development of models in a compositional manner. Since its development, a graphical representation has also been proposed to overcome problems concerning the unintuitive syntax [158], and a number of tools have been developed to support its simulation and analysis [159]. However, the calculus has no notion of invariants which can provide many structural and global insights into a system. Furthermore, the description of (often) straightforward systems can be troublesome, making it a language for experts [211].

The rest of this chapter will now discuss BNs and PNs in more detail, as these form the foundation of this thesis. Note we discuss the generalisation of BNs to MVNs in Chapter 4.

2.2 Boolean networks

Boolean networks (BNs) [101, 102] are a qualitative formalism that have received much attention over the last few decades from both the bioinformatics (see [57, 98, 137, 188] for good reviews) and physics communities (see, for example, [61, 82, 184, 189]). A BN is a directed graph where entity can be either *on* or *off*. The behaviour of each entity is then described by a Boolean function over a subset of other entities (possibly including itself) which directly affect it.

This Boolean approximation for modelling complex networks can be traced back to the early sixtees with the work of Sugita *et al.* [198, 199] who analysed chemical systems using logic circuits. This work was then built upon by the seminal work of Kauffman in 1969 [101, 102] who introduced BNs as a formalism for modelling complex natural systems, and provided a comprehensive mathematical study of their properties [77,103–105,189]. Around the same time, Thomas developed a sequential logical approach [203, 204, 207] which provided a purely asynchronous logical description of a GRN using Boolean discretisation.

2.2.1 The Boolean idealisation

The representation of an entity using only two values is subject to much debate (see [77] for early studies, and [86, 183, 188] for interesting discussions). On one hand, genes are observed to make the transition between a floor and maximum level of expression in a switch-like fashion [105,182,203], and so a Boolean approximation of this steep sigmoidal curve appears to be justified. In addition, there is often a substantial amount of noise at the level of regulation which can result in some degree of uncertainty [138], making the reliability of more detailed approximations questionable [183]. Furthermore, Kauffman showed that BNs still capture a rich amount of behaviour present in practical GRNs despite their simplicity [105].

However, BNs have some obvious shortcomings. Firstly, they are not always capable of capturing behaviours present in finer-grained models [77,188] (we will formally investigate this more in Chapter 5). In addition, Mestl *et al.* [142] argued that the order in which gene products became active and inactive in a BN should be consistent, if nothing else, with that provided by differential equations. However, this has been shown to not always hold [188].

Despite this, BNs have become an important tool for modelling and analysing large GRNs [83,107,186,194,200,206,208,214]. For example, Shmulevich *et al.* [186] presented an approach for analysing gene expression data completely in the Boolean domain using a combination of binary discretisation and normalisation. Their approach offered both noise resiliance and computational efficiency, and was successfully applied in identifying different tumour subclasses. Furthermore, Thomas *et al.* [206,208] showed that many of the dynamics present in GRNs manifest themselves in BNs; moreover, the results they showed were coherent with those obtained using differential equations, and so provided further confidence in their application.

2.2.2 Basic notations

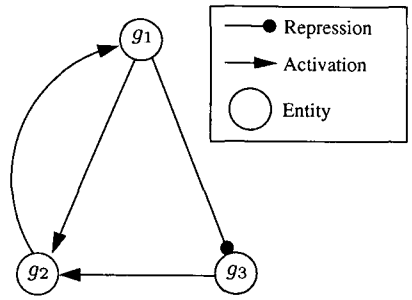
We now formally introduce BNs and their related notations which will be used throughout this thesis.

Definition 2.1 (Boolean network). *A Boolean network $\mathcal{BN}$ with k entities is a directed graph $\mathcal{BN} = (G, N, F)$ where: $G = \{g_1, \ldots, g_k\}$ is a finite set of entities; $N = (N(g_1), \ldots, N(g_k))$ is a finite list of sets, where $N(g_i) \subseteq G$ is the neighbourhood of g_i containing all the entities which directly affect its behaviour; and $F = (f_{g_1}, \ldots, f_{g_k})$ is a finite list of next-state Boolean functions, where $f_{g_i}: \mathbb{B}^{|N(g_i)|} \rightarrow \mathbb{B}$ defines the next state of g_i.*

Note in a slight abuse of notation, we use g_i to denote *both* the identity and state of entity g_i. We therefore write g_i and $\overline{g_i}$ to represent g_i being *on* ($g_i = 1$) and *off* ($g_i = 0$), respectively.

The structure of a simple BN with three entities g_1, g_2 and g_3 is illustrated in Figure 2.1(a) [6]. Such a representation specifies the neighbourhood $N(g_i)$ for each entity g_i using directed connections (either activation or repression). For example, the neighbourhood of entity g_1 is g_2, since g_2 activates g_1. To accompany this network structure, the state transition tables in Figure 2.1(b) describe the next-state functions F, i.e. the functional interactions between each entity. Such tables capture for each entity g_i, the next state reached $[g_i]$ for each possible input state of $N(g_i)$. If $|N(g_i)| = m$, then there are 2^m possible input states to consider. For example, the state transition table for entity g_2 contains $2^2 = 4$ input states, and shows us that one situation in which it will turn off is when g_1 is off and g_3 is on. Note throughout this thesis we may simply refer to a BN using its state transition tables, since they capture both its structure and behaviour precisely.

A more compact representation of the next-state functions F can be derived using *logic minimisation techniques* and is referred to as the *next-state equations* (we will discuss



g_2	$[g_1]$
0	0
1	1

g_1	g_3	$[g_2]$
0	0	0
0	1	0
1	0	0
1	1	1

g_1	$[g_3]$
0	1
1	0

$$[g_1] = g_2$$
$$[g_2] = g_1\ g_3$$
$$[g_3] = \overline{g_1}$$

(a) (b) (c)

Figure 2.1: (a) Diagrammatic representation of a BN with three entities, (b) the state transition tables capturing the corresponding next-state functions F, and (c) an equational representation of F after logic minimisation.

this in Section 3.1.1). This compact equational representation in *disjunctive normal form (DNF)* is shown in Figure 2.1(c), where the notation $x + y$ and $x\ y$ is used to represent Boolean disjunction and conjunction, respectively. For example, the next-state equation for entity g_2 shows us that it turns on when g_1 and g_3 are on, and we see that this relationship is reflected in its state transition table.

The collective state of each entity in a BN is called a *global state*. For example, if we have $g_1 = 1$, $g_2 = 0$ and $g_3 = 1$ then the global state is written $g_1\ \overline{g_2}\ g_3$, or sometimes in shorthand as 101 where it does not cause confusion. The state space of a model $\mathcal{BN}$ is therefore the set of all possible global states, which we denote by $S_{\mathcal{BN}}$. It is clear that if $\mathcal{BN}$ has k entities, then $|S_{\mathcal{BN}}| = 2^k$.

2.2.3 Update semantics

The dynamics of a BN can be interpretted using either synchronously or asynchronously, and we refer to these as the *update semantics* [86].

Definition 2.2 (Update semantics). *Let $\mathcal{BN} = (G, N, F)$ be a Boolean network. Then $\mathcal{BN}$ can be interpreted synchronously by applying each f_{g_i} to $N(g_i)$ for each $g_i \in G$ in unison. Similarly, $\mathcal{BN}$ can be interpreted asynchronously by applying f_{g_i} to $N(g_i)$ for only one $g_i \in G$ chosen randomly.*

Figure 2.2 shows the state graphs illustrating the dynamics of the BN in Figure 2.1 under both the synchronous and asynchronous semantics. States are represented by nodes and edges denote instantaneous transitions between states. Even for our simple model, the differences between the two update semantics are apparent. For example, consider global state 010. Under the synchronous semantics shown in Figure 2.2(a), there is only one possible next state, 101 resulting from the simultaneous application of f_{g_1}, f_{g_2} and f_{g_3} to state 010. As such, there is a deterministic pathway through the state graph from any initial state. However, under the asynchronous semantics in Figure 2.2(b), there are

three possible next states resulting from independent application of the corresponding next-state functions to state 010, namely: 110 if we apply f_{g_1}; 000 if we apply f_{g_2}; and 011 if we apply f_{g_3}. Thus, multiple pathways can exist through the state graph under the asynchronous semantics, resulting in non–determinism.

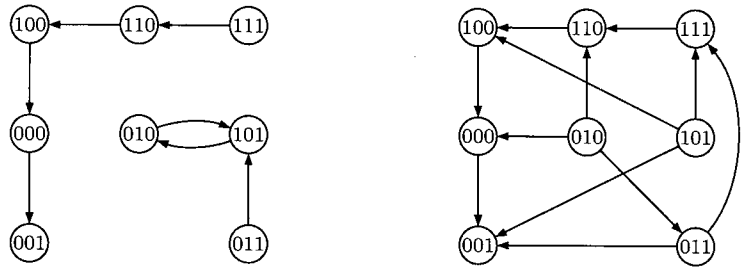


Figure 2.2: State graph showing the dynamics of the BN in Figure 2.1 under: (a) the synchronous semantics; and (b) the asynchronous semantics.

A sequence of state transitions from some initial state is called a *trajectory* [217]. For example, in Figure 2.2(a), one such trajectory is 111 $\rightarrow$ 110 $\rightarrow$ 100. Since the state space of a BN is finite, a trajectory will eventually fall into a cycle, which is often called an *attractor cycle* (see [105, 217] for a comprehensive study). In Figure 2.2(a), there are two attractor cycles: 001 $\rightarrow$ 001 (called a *point attractor* or *steady state*), and; 010 $\rightarrow$ 101 $\rightarrow$ 010 (called a *cyclic* attractor or *periodic cycle*). The number of states present in an attractor cycle is referred to as its *period.* For example, the period of the cycle 010 $\rightarrow$ 101 $\rightarrow$ 010 is two.

Point attractors exist and are inescapable under both the synchronous and asynchronous semantics [113]. However, Harvey *et al.* [86] showed that whilst numerous point attractors can exist under the synchronous semantics, the number of expected point attractors under the asynchronous semantics is one independent of network connectivity. The same authors also showed the presence of so-called *loose attractors* (cyclic behaviour from which it is possible to escape) under the asynchronous semantics, which result from the inherent non–determinism of random updates. As such, they argued that cyclic attractors seldom exist in asynchronous models. This is illustrated in Figure 2.2(b); the cyclic attractor 010 $\rightarrow$ 101 $\rightarrow$ 010 present in Figure 2.2(a) disappears under the asynchronous semantics. Instead, three pathways emerge from state 010 which all end at the point attractor 001 $\rightarrow$ 001.

A state outside an attractor cycle is called a *transient state* [216]. The combination of transient states and attractor cycles is refered to as a *basin of attraction* [216], since by definition transient states will eventually fall into the cycle. In Figure 2.2(a), there are two distinct basins of attraction, whilst in Figure 2.2(b) there is one. Note the dynamics of attractor cycles have been extensively studied, and we discuss some of this work in the next section.

Historically, the synchronous semantics appears to be favoured, especially in the bio-

logical community, as it avoids the need to consider the different reaction rates present in the underlying GRN by assuming they are all equal [86]. In addition, there is a deterministic pathway through the state space of a synchronous BN, which is appealing for both interpretation and automated analysis. Thus, it appears that the synchronous semantics is more readily used than its (more biologically-realistic) asynchronous counterpart, as it is simpler to interpret [56] and has more abundant tool support (for example, the DDLab software suite by Andrew Wuenshe[1]). Indeed, this thesis goes some way to address this issue.

There is also substantial work which supports the synchronous update semantics [5, 6, 120, 195, 214, 216]. For example, Wuensche [216] argued that since gene regulation at the molecular level comprises of discrete events occurring concurrently, a synchronous view is acceptable. In [214], Wen *et al.* showed that some systems exhibited natural synchrony; gene expression patterns in cervical spinal chord tissue were detected which occurred in a synchronous fashion. However, in [86], it is argued that the synchronous idealisation is only acceptable if fine time-splicing is used, and that this is exactly what BNs do not provide. See [86] for further interesting discussions on the comparison between the synchronous and asynchronous semantics, and [70] for a computational analysis of the differences between the two approaches.

2.2.4 Attractor dynamics and random Boolean networks

Attractor cycles are widely considered to represent the different cellular types, such as proliferation, apoptosis and differentiation [93–95, 183]. Thus, their theoretical and practical implications have warranted much attention in the literature (for example, see [61, 118, 190, 224]). Kauffman used so-called *random Boolean networks (RBNs)* to investigate the affect of neighbourhood size and functional connectivity on attractor dynamics [102, 105]. In an RBN, a random neighbourhood size and next-state function is assigned to each entity (for a comprehensive introduction and literature survey, see [73], and for biological justification, see [105]). By considering randomly wired BNs of up to $|G| = 10000$ entities, Kauffman postulated that for small neighbourhood sizes (one or two): (i) the average number of attractors was approximately $\sqrt{|G|}$; and (ii) the average length of an attractor grew proportionally to $\sqrt{|G|}$. These measures appear to be consistent with experimental knowledge. For example, [183] noted that the human genome has approximately 40000 known genes, and so we would expect 200 attractor cycles representing the various cell types, such as proliferation, apoptosis and differentiation. This approximation is remarkably close to the 254 known cell types in an adult human (the authors reference [12]).

Kauffman also observed that network dynamics were highly ordered for small neighbourhood sizes of one or two, but increasingly chaotic above that [104, 105]. In particular, a neighbourhood size of two was found to be the critical parameter in separating order from chaos (see [86] for more discussion on neighbourhood sizes). Such models are often called *critical Kauffman models (CKMs)*, and are the subject of much study as it is thought that complex organisms also operate at this critical balance between order and

[1]DDLab can be downloaded from *http://www.ddlab.com/*

chaos [105, 106]. The rationale for this is that under the order found with a neighbourhood size of one, the network is very stable and robust to perturbations. In other words, if the system is perturbed it will most likely remain in the same attractor (the same cell type). However, this appears to go against the principles of evolution, which require flexibility and adaptability. On the other hand, the network becomes very unstable and susceptible to perturbations in the chaotic phase. Again, this disagrees with the idea that an organism must have a certain level of stability in order to survive adverse conditions.

To date, a substantial amount of work in the literature has focused on the mathematical properties of CKMs and attractors. More recently, Samuelsson *et al.* [171] proved that, in fact, the number of attractors grows superpolynomially with $|G|$ for neighbourhoods of size two – much faster than originally suggested by Kauffman. Drossel *et al.* [61] built on these results by proving that both the average number and length of attractors scales superpolynomially for neighbourhoods of size one. Whilst it appears that proving superpolynomial growth for attractor lengths with neighbourhoods of size two is still an open question, Drossel *et al.* noted that, if this property does indeed hold, then it suggests that the number and length of attractors is too large to model cell differentiation realistically.

Shmulevich *et al.* [184] developed a method for measuring the importance of a Boolean function based on its sensitivity. Using this metric, they showed how the type of network (ordered or chaotic) could be determined without undertaking computationally-intensive simulations. In [82], Greil *et al.* investigated the effect of introducing delays on the number and length of attractors. Their findings showed that in the presence of delays, the average length of attractors naturally increased whilst their frequency decreased, and the size of the basins of attraction increased, making the network more stable. Similarly, Sontag *et al.* [190] investigated the relationship between negative feedback loops and attractor cycles. They too observed that as the number of negative feedback loops increased in a network, the average length of the attractors increased whilst their frequency decreased.

2.2.5 Constructing Boolean models of genetic networks

Since an initial study by Thomas [203], substantial work has focused on the application of BNs for learning the structure of GRNs from experimental data, such as microarray time series [33]. Liang *et al.* [125] proposed an information theoretic approach for network inference with the development of the reverse engineering algorithm *REVEAL*. This algorithm uses the *Shannon entropy* [176] to assess the mutual information between input-output pairs in the state transition table of the BN. In particular, REVEAL addresses the problem of inferring BNs from incomplete data. The authors showed that for small neighbourhood sizes, REVEAL performed well and was able to infer the original network structure from only $O(\log k)$ out of the 2^k state transition pairs, where k was the number of entities. They also outlined ideas for parallelising the algorithm, and discussed how it could be adapted to handle multi-state entities.

Akutsu *et al.* [4, 5] took a different approach by considering the results of multiple gene disruption and overexpression experiments. Here, the behaviour of one or more

genes was mutated so that they remained at either a high or low level of expression. By analysing the effects of this abnormal behaviour on the network, they showed how the neighbourhoods and next-state functions could be inferred to describe the behaviour of the underlying GRN. Furthermore, they used this method to develop a simulator and identification system for inferring the underlying GRN using $O(k^3)$ experiments. Although their method appears to work well with relatively small networks, it does not scale to more practical data sets, and is only applicable to networks with a maximum neighbourhood size of two.

Akutsu *et al.* later considered this scalability issue in [6] by: (i) formally proving the soundness of the REVEAL algorithm; and (ii) proposing a simpler set of algorithms which made mathematical analysis more tractable. In particular, they applied techniques from *computational learning theory* to BNs (see [108] for an introduction), such as the *consistency problem* paradigm, which is the problem of determining the existence of a network which is consistent with the data. In [7], the authors then extended this suite of algorithms by incorporating an efficient Monte-Carlo randomised method based on combining matrix multiplication with an efficient hashing function, resulting in substantial speedups. Furthermore, they extended their notion of a BN to incorporate a noise factor, which leaned away from the traditional deterministic view provided by BNs and more towards stochastic modelling.

The noise element in practical data sets results in a level of uncertainty [138] which must be addressed in order to reverse engineer meaningful qualitative models. As such, many subsequent strategies have been suggested to address this issue since the work reported in [7]. Shmulevich *et al.* [185] argued that the consistency problem paradigm was insufficient in the presence of noise, and applied a generalised paradigm from computational learning theory called the *best-fit extension problem* (see [30] for an extensive study). The best-fit extension problem is solvable in polynomial time, making it applicable in practice, and aims at inferring the underlying BN whilst making as few mis-classifications as possible. This approach was later successfully applied in [120] to the cdc15 yeast gene expression time series data.

Yun *et al.* [220, 221] proposed an information-theoretic approach called the *discrete function learning (DFL)* algorithm, which is superior to REVEAL in efficiency, flexibility and robustness to noise. In particular, DFL is a generalised approach and is able to infer the underlying GRN from both Boolean and multi-valued data. The authors compared DFL to a Bayesian network model using a yeast time series data set, and noted that DFL identified a network that more closely represented the known regulatory relationships reported in the literature.

In [182], Shmulevich *et al.* introduced a stochastic extension to BNs called *probabilistic Boolean networks (PBNs)*, which relaxed the deterministic constraints imposed by traditional BNs. In a PBN, the behaviour of each entity may be described by a *set* of Boolean next-state functions with associated probabilities. The rationale behind this is that, due to noise factors, it may not be possible to select a next-state function which best describes the behaviour of an entity; however, a most probable set of next-state functions can be determined which fit the data (for further justification of the rationale behind PBNs, refer to the later work of Shmulevich *et al.* in [183]). The authors also

considered the natural comparison between PBNs and Bayesian networks due to their similar probabilistic approach. They demonstrated that PBNs were advantageous over Bayesian networks because they are able to model networks with self-loops and were more efficient to compute (see [119] for further discussion). Since then, a comprehensive study [124] has indicated that whilst PBNs are indeed more efficient to compute, they do not seem able to identify as many regulatory interactions. For further work which has used PBNs, see [10, 96, 223].

Martin *et al.* [129] argued that multiple consistent candidate models should be inferred in the presence of noise. They developed an approach combining clustering techniques with standard BN inference. The clustering algorithm used a force directed graph layout and produced a two dimensional representation of the genes by grouping those with similar expression profiles. They then applied *support vector regression* (see [37] for a comprehensive tutorial) to fit a curve to the expression profile of each cluster, and binarised these to produce a progression of Boolean states over time. Finally, they applied *logic minimisation techniques* [140] to the Boolean profiles to infer a compact representation of the next-state functions describing the behaviour. The multiple inferred networks were then themselves clustered in a further processing step if they shared similar attractor dynamics, resulting in a relatively small number of candidate networks. This approach was further developed by the authors in [130] and successfully applied to two data sets. However, whilst their approach appears to combine many novel techniques, there are questions concerning its scalability due to the pipeline of intensive processing steps required.

Finally, there have been attempts at developing hybrid Boolean models which marry together Boolean and continuous formalisms. For example, McAdams *et al.* [139] constructed a model for the lysis-lysogeny switch mechanism in phage λ using a hybrid circuit consisting of both Boolean and continuous components, and through simulation showed that it faithfully captured the dynamics of this well-studied system. However, it is unclear how such models and related parameters can be inferred in practice without both significant knowledge of kinetic parameters and substantial computational power. Indeed, the inference of quantiative models still appears to be a problematic task, and this is a key motivation for using the qualitative formalisms discussed in this thesis. For studies following on from this work, see [8, 9, 11, 219], and for further discussion into this hybrid approach, refer to [137].

2.3 Petri nets

The theory of Petri nets (PNs) [147] combines a graphical notation with a formal mathematical semantics for modelling and reasoning about complex concurrent systems [147]. With their ability to model synchronous and asynchronous events, true concurrency, resource sharing and conflicts, PNs have been applied to many problem domains (see [226] for industrial applications), including system verification [151, 177, 210], manufacturing processes [3, 225], hardware design [166] and more recently biological systems [53, 79] (also, see [215] for examples of how PNs have been used in chemistry and medicine). In fact, the inventor of PNs, Carl Petri, noted their suitability for modelling natural

processes from the start [156].

Specifically, PNs offer a number of key advantages for modelling biological systems:

- they have an *intuitive graphical representation* underpinned by a rigorous and well-understood *formal semantics*;
- they are able to capture *both* the static structure and dynamical behaviour of a system in a concise way;
- they support *incremental model development* by allowing further information such as time to be added whilst preserving the underlying net;
- they are able to model a system *both* qualitatively and quantitatively;
- they support analysis at *varying levels* depending on the completeness of the model, from purely structural through to behavioural through to timed behavioural;
- and they have a *wealth* of formal techniques and tools for simulation and analysis [1].

In this section, we investigate these key motivations in more detail by formally introducing PNs and related notations. We then discuss some properties of PNs and their biological significance. For a more detailed introduction to PNs, we recommend [147].

2.3.1 Basic notations

A PN (see Figure 2.3) is a bipartite graph consisting of *places*, which represent conditions or resources, and *transitions*, which represent events or actions. The places of a PN can contain *tokens* which indicate the presence of a resource, and the distribution of tokens represent the state of the PN called a *marking*. Tokens can then be moved around the net by *firing* transitions, which can only occur if the input places to the transitions contain sufficient tokens.

We now give a more formal definition of a PN.

Definition 2.3 (Petri net). *A Petri net $\mathcal{PN}$ is a tuple $\mathcal{PN} = (P, T, F, W, M_0)$ where: $P = \{p_1, \ldots, p_n\}$ is a finite set of places; $T = \{t_1, \ldots, t_m\}$ is a finite set of transitions, such that $P \cap T = \emptyset$; $F \subseteq (P \times T) \cup (T \times P)$ is the flow relation specifying connections between places and transitions; $W : F \to \mathbb{N}^+$ is the weight function which assigns a weight to each arc $f \in F$; $M_0 : P \to \mathbb{N}$ is the initial marking specifying the number of tokens on each place.*

A PN can be represented diagrammatically, where the set of places P is denoted by circles, the set of transitions T is denoted by boxes and the flow relation F is denoted by directed arcs. Furthermore, the weight function W is denoted by labelling each arc with a weight to indicate its multiplicity (we do not label arcs whose weight is 1). If all arcs have a weight of one, then the PN is said to be *ordinary*. The initial marking M_0 gives the distribution of tokens on places and is denoted by small black circles. Note we represent a marking $M : P \to \mathbb{N}$ formally as a tuple $(M(p_1), \ldots, M(p_n))$ which gives the number of tokens for each place in the net.

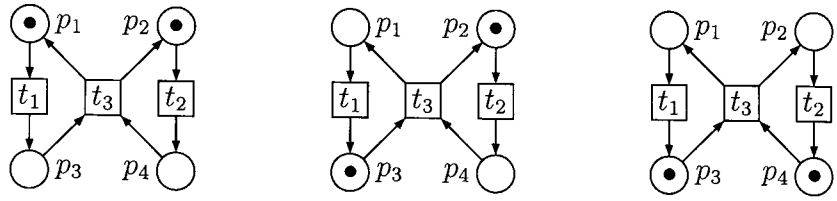


Figure 2.3: (a) Simple PN in initial marking with transitions t_1 and t_2 enabled, (b) PN after firing t_1 with t_2 enabled, and (c) PN after firing t_2 with t_3 now enabled.

A simple example of an ordinary PN is shown in Figure 2.3(a), consisting of four places p_1, p_2, p_3 and p_4, and three transitions t_1, t_2 and t_3. Its initial marking is such that only places p_1 and p_2 contain a token, and so we write $M_0 = (1, 1, 0, 0)$ to represent this, where $M(p_1) = 1$, $M(p_2) = 1$, $M(p_3) = 0$ and $M(p_4) = 0$.

A place $p \in P$ is called an *input place* (resp. *output place*) to a transition $t \in T$ if there exists an arc $f \in F$ from p to t (resp. t to p). For example, in Figure 2.3, place p_3 is an input place to t_3, whilst p_1 is an output place of t_3. The set of all input places to a transition t is called the *pre-set* of t, and is denoted ${}^\bullet t$ where ${}^\bullet t = \{p \mid (p, t) \in F\}$. Similarly, the set of all output places of a transition t is called the *post-set* of t, and is denoted $t^\bullet$ where $t^\bullet = \{p \mid (t, p) \in F\}$. In our running example, transition t_3 has pre-set ${}^\bullet t_3 = \{p_3, p_4\}$ and post-set $t_3{}^\bullet = \{p_1, p_2\}$.

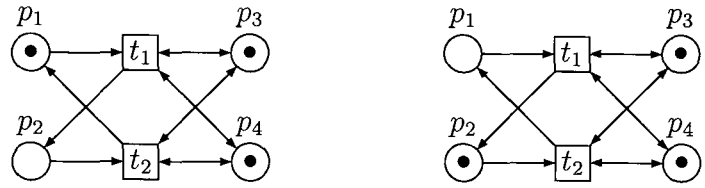
The dynamic behaviour of a PN is governed by the transition *firing rule*, which we now formally define.

Definition 2.4 (Firing rule). *For a transition t and a marking M, we say that t is enabled at M, denoted $M[t\rangle$, if for every place $p \in {}^\bullet t$, we have that $M(p) \geq W(p, t)$. If t is enabled, then it may fire, resulting in marking M', denoted $M[t\rangle M'$, where $M'(p) = M(p) - W(p, t) + W(t, p)$ for every place $p \in P$.*

Note PNs are a non–deterministic modelling formalism, and so a single transition is chosen at random to fire when more than one is enabled. A sequence of transitions is called a *firing sequence*, written $\sigma = \langle t_1, \ldots, t_n \rangle$, if each transition is able to fire (according to Definition 2.4) in the order specified. As such, a marking M' is said to be *reachable* from M if there exists a firing sequence which transforms M into M'. The set of all possible markings reachable from M is denoted by $RM(M)$.

To illustrate these ideas, consider Figure 2.3(a) which shows the PN in its initial marking $M_0 = (1, 1, 0, 0)$. Places p_1 and p_2 contain sufficient tokens to enable transitions t_1 and t_2, and so we have a non–deterministic choice between two firing sequences. If we fire t_1, then a token is removed from p_1 and deposited on p_3, resulting in the marking $M_1 = (0, 1, 1, 0)$ shown in Figure 2.3(b). In marking M_1, only transition t_2 is enabled, since transition t_3 requires both $M_1(p_3) = 1$ and $M_1(p_4) = 1$ to hold. Thus, our only choice is to fire t_2, resulting in marking $M_2 = (0, 0, 1, 1)$ which is shown in Figure 2.3(c). Now that both p_3 and p_4 are marked, we can fire transition t_3, and this returns us back to our initial marking $M_0 = (1, 1, 0, 0)$ shown in Figure 2.3(a). Our firing sequence was therefore

$\sigma_1 = \langle t_1,t_2,t_3 \rangle$. However, since both t_1 and t_2 are enabled in M_0, another possibility would be $\sigma_2 = \langle t_2,t_1,t_3 \rangle$, resulting in markings $M_3 = (1,0,1,0)$, $M_2 = (0,0,1,1)$ and $M_0 = (1,1,0,0)$, respectively. Therefore, the set of reachable markings from M_0 is $RM(M_0) = \{M_0, M_1, M_2, M_3\}$.



(a) (b)

Figure 2.4: (a) Simple PN with read arcs in initial marking with transition t_1 enabled, and (b) PN after firing transition t_1.

In some cases, bidirectional arcs called *read arcs* may be used to allow transitions to read (but not change) the tokens on places. For example, let us consider the PN in Figure 2.4(a), which contains read arcs between transitions t_1 and t_2 and places p_3 and p_4, respectively. In the initial marking, only transition t_1 is enabled. When we fire t_1, we obtain the marking shown in Figure 2.4(b), where a token has been removed from p_1 and added to p_2 only; the tokens on places p_3 and p_4 have not changed and were simply used as a firing condition. Now t_2 is enabled, and firing it returns us back to the initial marking shown in Figure 2.4(a). As such, the markings constantly oscillate between $M_0 = (1,0,1,1)$ and $M_1 = (0,1,1,1)$.

In the remainder of this section, we consider some of the properties of PNs which can be analysed using the wide range of tools and techniques available [1], and discuss their biological significance.

2.3.2 Structural analysis

The topology of a PN can be used to gain many insights into its properties through structural analysis. Such analysis is advantageous for the following reasons: (i) it does not rely on any initial marking – in fact, it applies to any marking, and so provides a useful means of studying fundamental behaviour; and (ii) it does not suffer from *state space explosion*, which sees an exponential blowup in possible states.

A PN is a directed bipartite graph whose structure can be described by a $|P| \times |T|$ *incidence matrix* $\mathbf{A} = [a_{ij}]$ of integers, such that for each entry a_{ij} we have:

$$a_{ij} = W(t_j,p_i) - W(p_i,t_j).$$

In other words, each a_{ij} describes the change in tokens on place p_i resulting from firing transition t_j, for $i = 1,\ldots,|P|$ and $j = 1,\ldots,|T|$. For example, the PN from Figure 2.3

has the following incidence matrix:

$$\begin{pmatrix} -1 & 0 & 1 \\ 0 & -1 & 1 \\ 1 & 0 & -1 \\ 0 & 1 & -1 \end{pmatrix}$$

Each column of the matrix shows the effect that firing the corresponding transition has on the four places. For instance, when t_1 is fired (first column), a token is taken away from place p_1 (first row) and added to p_3 (third row). As t_1 is not adjacent to places p_2 and p_4, no tokens can be affected and so we write 0 for the second and fourth rows. Note although non-adjacency implies a zero entry in the matrix, the reverse does not hold; if we have a place p_i connected to a transition t_j with a read arc, then $a_{ij} = 0$. Thus, PNs with read arcs can be problematic for structural analysis. However, these can be transformed into *pure PNs* which contain no read arcs, thus enabling meaningful structural analysis to be performed. Although we do not discuss this transformation here, we refer the reader to [147], and simply note that: (i) this transformation can be automated[2]; and (ii) the PN in Figure 2.3 is pure.

From the incidence matrix, a number of structural properties can be analysed using *linear algebraic techniques* [89, 91, 147]. Two of the most common structural properties are *place invariants* (P-invariants) and *transition invariants* (T-invariants):

- P-invariants represents state quantities which, starting from an initial state, are conserved throughout the running of the system. In biological terms, a P-invariant could represent a conservation relation. More formally, a P-invariant $\mathbf{x}$ is a non-zero vector solution to the linear equation $\mathbf{x} \cdot \mathbf{A} = 0$, i.e. the places and their corresponding token count which are conserved from some marking M.

- T-invariants are sequences of transitions which, when fired, reproduce a given marking. In biological terms, a T-invariant represents an attractor cycle or elementary mode [173] . Formally, a T-invariant $\mathbf{y}$ is a non-zero vector solution to the linear equation $\mathbf{A} \cdot \mathbf{y} = 0$. A T-invariant is only realisable in practice, however, if a marking is reachable such that all the transitions in the T-invariant are able to fire. Thus, in some cases, it is necessary to consider the dynamics of the PN when tokens are introduced in conjunction with invariant analysis.

For example, we can calculate the T-invariants for the PN in Figure 2.3 by deriving all column vectors $\mathbf{y}$ such that:

$$\begin{pmatrix} -1 & 0 & 1 \\ 0 & -1 & 1 \\ 1 & 0 & -1 \\ 0 & 1 & -1 \end{pmatrix} \cdot \mathbf{y} = 0,$$

[2]PETRIFY can perform many such transformations on a PN, and can be downloaded at *http://www.lsi.upc.es/ jordicf/petrify/home.html.*

and we find that there are infinitely many solutions $\mathbf{y} = (a, a, a)$, where $a > 1$. These solutions correspond to firing t_1, t_2 and t_3 a total of a times each, and we can see that no matter what marking we start from, we will return to this after such a firing sequence. Note in practice, we only need to consider $a = 1$.

2.3.3 Dynamical analysis

Structural analysis of a PN has the advantage that it avoids state space explosion, since only the network topology is considered. However, properties inferred in this way are not always realisable in practice, and so we often have to consider the dynamics of a PN when tokens are introduced. Of course, such analysis is reliant on the choice of initial marking, which requires additional knowledge of the underlying system and the properties that one wishes to study. Here, we discuss a number of PN properties along with their biological significances. For a more detailed discussion of these properties, see [147].

Reachability analysis

Reachability analysis is a fundamental technique, and forms the basis for checking many other properties, such as *boundedness*, *deadlocks* and *mutual exclusion* [147]. Specifically, reachability analysis allows one to check whether some marking is reachable from another. More formally, given a marking M, we check to see whether $M \in RM(M_0)$ holds, where M_0 is the initial marking.

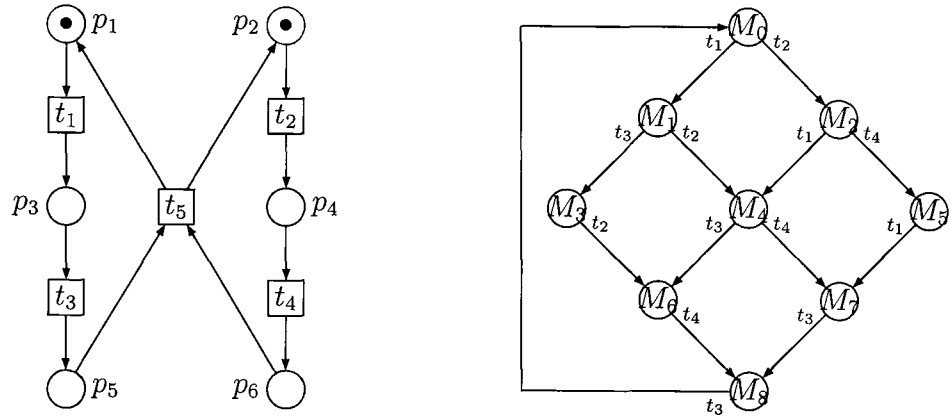
Biologically, reachability analysis can prove to be an insightful technique. For example, one might wish to check whether two proteins can ever be present at a high concentration; a property that could possibly indicate *positive feedback* [206]. In the PN model, this could be formulated as a mutual exclusion problem, and reachability analysis would therefore check for the existence of a marking representing the two proteins being high, i.e. a counter-example. Of course, the absence of such a marking would indicate that this property holds.

Reachability analysis is normally computed based on the so-called *reachability graph* [147], whose set representation is $RM(M_0)$. The reachability graph is a directed graph where nodes represent markings and edges represent transitions firing, and is constructed by a breadth-first search[3] from M_0 until no new markings are discovered. For example, consider Figure 2.5(a) which shows a PN, and Figure 2.5(b) which shows the corresponding reachability graph from initial marking $M_0 = (1, 1, 0, 0, 0, 0)$.

From the initial marking $M_0 = (1, 1, 0, 0, 0, 0)$, we see that both transitions t_1 and t_2 can fire. If we fire transition t_1, then we obtain marking $M_1 = (0, 1, 1, 0, 0, 0)$ for which there are a further two enabled transitions. This breadth-first search continues until we reach marking $M_8 = (0, 0, 0, 0, 1, 1)$, whereby firing transition t_5 returns us to the initial marking.

Even for this small example, we can see that the reachability graph grows quickly (often exponentially) due to state space explosion caused by concurrency in the PN. As such, reachability is known to be a decidable property [136], although it takes exponential

[3] A depth-first search can often fail to find all reachable markings.



(a) (b)

Figure 2.5: (a) PN with six places and five transitions, and (b) the corresponding reachability graph representing $RM(M_0)$.

time and space to verify [39]. In practice, the reachability graph can therefore be too large to compute and verify in reasonable time.

Numerous studies have considered this problem, and have looked at improving the efficiency of constructing and representing the reachability graph [109,110,145]. *Unfoldings* and their *canonical prefixes* have received much attention [63,109,110], as they provide a compact representation of the state space whilst capturing the desired behaviour. In particular, canonical prefixes are often exponentially smaller than the reachability graph, cope well with concurrency, are finite and are complete in the sense that they capture all the behaviour of the reachability graph. *Model checking* techniques [109] which use these prefixes to check for properties such as reachability, deadlock-freeness and mutual exclusion can then be utilised for efficient dynamical analysis. In fact, this thesis makes extensive use of the parallel PN unfolder PUNF and the linear programming model checker CLP, both of which were developed in [109].

Boundedness

Boundedness imposes an upper limit k on the number of tokens that the places of a PN can contain from an initial marking, and so puts an upper limit on $|RM(M_0)|$. A place is said to be k-bounded if it never contains more than k tokens from some initial marking. A PN is therefore said to be k-bounded if all its places are k-bounded from an initial marking, or more formally that for each $M \in RM(M_0)$ and for each $p \in P$, we have that $M(p) \leq k$. A PN is said to be *safe* if the respective bound is one [147].

An example of a safe PN is shown in Figure 2.6(a). Figure 2.6(b) illustrates a 2-bounded PN; here we have $W(t_2, p_3) = W(p_3, t_3) = 2$. Thus, firing transition t_2 puts two tokens on place p_3, which are then removed when t_3 fires. Finally, Figure 2.6(c) illustrates an unbounded PN, since the addition of an extra arc between t_3 and p_2 results in the premature firing of t_2, thus allowing an infinite number of tokens on p_1.

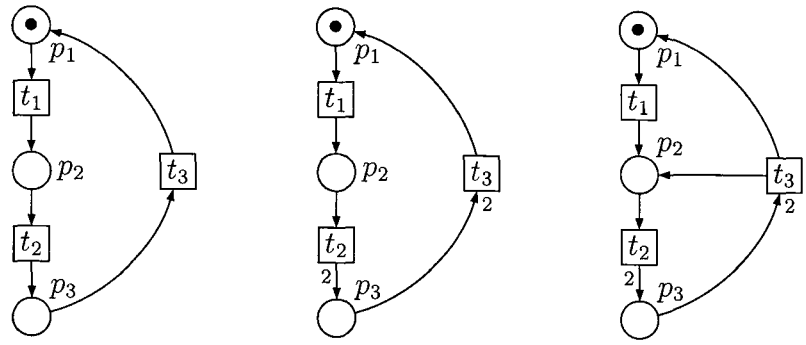


Figure 2.6: (a) Safe PN, (b) 2-bounded PN, and (c) unbounded PN.

Boundedness is an important property to check for during model construction; indeed, its significance is context-dependent. For example, in a PN model of a buffer system (which are always bounded), we may wish to verify whether the PN is bounded, otherwise there may exist hazardous buffer overflows which will have a detrimental impact on the system. Genes also appear to respect a notion of boundedness, since their expression is observed to vary between a floor and maximal value [105, 182, 203]. Thus, boundedness can prove a useful validation strategy before further analyses are performed.

This thesis focuses primarily on safe PNs, as they appear to be more efficient to analyse and are amenable to a wider range of techniques and tools. Note although it is possible to transform any bounded PN into a safe PN with equivalent behaviour [147], such transformations can often result in an unnecessarily large PN. This is therefore a clear motivation for focusing on the construction of compact safe PN.

Deadlocks and liveness

We say that a marking M is *deadlocked* if it does not enable any transitions. A PN is therefore deadlock-free if none of its markings contain a deadlock. As an example, consider Figure 2.7(a) which illustrates a PN with a deadlock and Figure 2.7(b) which shows the deadlocked marking.

There are a number of firing sequences which ensure the normal running of the net, such as $\sigma_1 = \langle t_1, t_3, t_2, t_4, t_6 \rangle$ or $\sigma_1 = \langle t_2, t_1, t_3, t_4, t_6 \rangle$. However, when we have $M(p_5) = 1$ and $M(p_6) = 1$, transitions t_5 and t_6 are enabled. If t_6 fires then we reach the deadlock marking $M = (0, 0, 0, 0, 0, 1, 1)$ in which no transitions are enabled.

In most cases, deadlocks do not represent intended behaviours of the system; in fact, they can often represent modelling errors or detrimental scenarios. Biologically, deadlocks can represent stable states of the system which may correspond to phenomenon such as apoptosis. In any case, deadlock detection plays an important part of system validation and can be automated by numerous tools such as CLP (see [109] for efficient model checking approaches to deadlock detection).

On the other hand, *liveness* is a property which relates to the complete absence of deadlocks. More specifically, a PN is said to be live if at any marking M reachable from

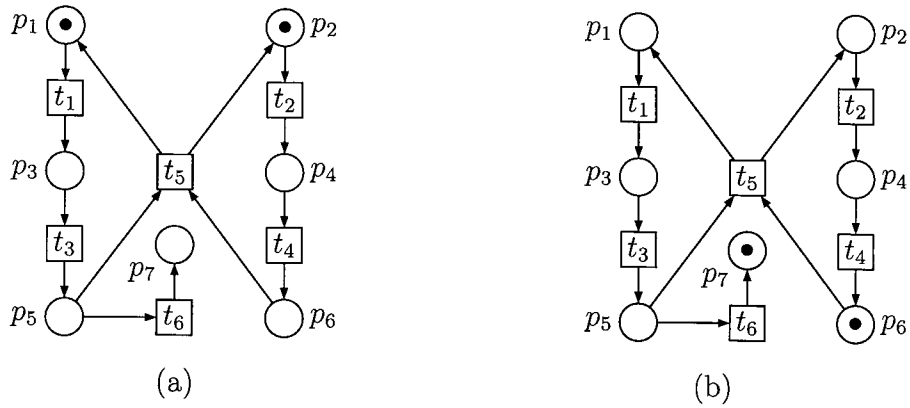


Figure 2.7: (a) PN in initial state with a deadlock, and (b) PN in deadlocked marking.

M_0, it is possible to fire any transition $t \in T$ via some further firing sequence σ. However, this property is costly to verify, and so the notion of liveness for a transition $t \in T$ is categorised into different levels:

- L0-live (dead) if t can never be fired for any sequence from M_0;
- L1-live if t can be fired at least once for some sequence from M_0;
- L2-live if t can be fired at least k times for some sequence from M_0, where $k \in \mathbb{N}^+$;
- L3-live if t can be fired infinitely for some sequence from M_0;
- L4-live if t is L1-live for each $M \in RM(M_0)$.

Biologically, the liveness property may correspond to a guarantee of a reaction or regulatory interaction occurring. Liveness can be checked for automatically by numerous tools such as INA, which is freely available for academic use from *http://www2.informatik.hu-berlin.de/ starke/ina.html.*

Choice and persistency

A *choice* exists between two transitions if they are both enabled at some marking M, and where firing one disables the other. A simple example of this is shown in Figure 2.8(a). One can see that transitions t_1, t_2 and t_3 are enabled in the specified marking, and that firing one of them will disable the others.

Persistency is a property of PNs related to the absence of such choices. More specifically, a persistent PN is one in which any enabled transition will *not* be disabled by some other transition. An example of a persistent PN is shown in Figure 2.8(b); at no reachable marking is there an enabled transition which can disable another.

Choices in the PN can represent non-deterministic events with only one outcome. Biologically, choices can represent race conditions between chemical reactions or competing proteins; the winner of the race will produce the corresponding chemical products or become synthesised first, respectively. As such, the identification of choices can be an

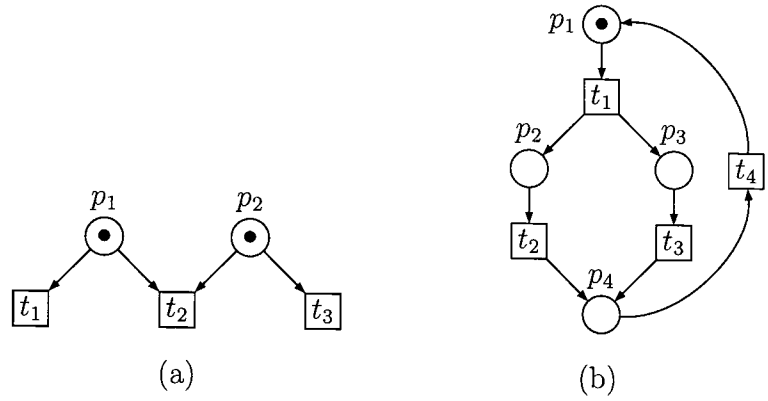


Figure 2.8: (a) Choice modelled in PN, and (b) persistent PN with no choices.

interesting means of analysing a biological PN model, and we discuss this in more detail in Chapter 6.

2.4 Modelling and analysis of biological systems using Petri nets

Having introduced the basics of PN theory, we go on to discuss some of the success that PNs have enjoyed in the literature for modelling and analysing biological systems. We start by looking at some of the studies that have applied standard PNs to GRNs, as well as metabolic and signalling pathways. Then, we consider a number of extensions which add additional information to the underlying PN graph structure, and review their application in this area. For further comprehensive reviews of PNs in biology, we recommend a number of detailed articles [44, 84, 155, 160, 215]

2.4.1 Standard Petri nets

Although Carl Petri noted the suitability of PNs for biological systems back in the 1960s [156] (and more comprehensively in the 1970s [157]), it was not until the work of Reddy *et al.* [163, 164] that the study of PNs became mainstream in the biological community. Reddy *et al.* motivated the use of PN techniques for qualitatively modelling *metabolic pathways (MPs)*. More specifically, they proposed a mapping mechanism which translated key biochemical features into PN structures, with a particular application to the combined glycolytic and pentose phosphate pathways in erythrocytes (this study was later extended with a thorough analysis in [115]). The idea of the mapping was to represent biochemical species using places, reactions using transitions, molecular quantities using tokens and reaction directions using arcs. As such, this representation provided a convenient means of capturing the stoichiometry of biochemical reactions, and thus formed the basis of many other subsequent studies [47, 72, 115, 222].

In addition, Reddy *et al.* were the first to comprehensively investigate the relationship between PN theory and biochemical theory. In particular, the main result was the relationship drawn between the incidence matrix of the PN and the stoichiometric matrix of the biochemical system. Due to the clear parallels, it was shown how the computation of P- and T-invariants could be used to shed light on conservation relations within the biochemical reactions [163] (note this relationship has been investigated further more recently in [174]). Specifically, many properties of PNs were shown to have corresponding properties in the biochemical domain, such as the correspondence between T-invariants and elementary modes [213, 222]. Subsequently, PNs are recognised as an insightful formalism for analysing biological systems, and this is demonstrated by approaches such as that of Shaw *et al.* [179], who considered a translation for biological models specified in SBML into the PNs formalism so that they are amenable to their wealth of techniques and tools.

Structural PN analysis has since played a key part in understanding fundamental properties of pathways. The main motivation for this is the apparent lack of reliable kinetic parameters needed for constructing quantitative models; indeed, this ability to analyse PNs at multiple levels of detail is one of their prime advantages. Furthermore, by analysing the structure of a PN, one circumvents the requirement to explore the state space of a model by considering fundamental dynamics only.

Koch and Heiner *et al.* [90] focused on the study of P- and T-invariants to validate constructed models of MPs prior to further analysis. Specifically, they focused on pathway models for apoptosis, carbon metabolism in the potato tuber and the glycolysis and pentose phosphate metabolism. In all studies, they identified the critical invariants relating to the fundamental processes governing these systems. In a later paper [114], the authors revisited the potato tuber pathway, and provided a more comprehensive analysis of this system by investigating both the structural and behavioural properties of their PN model.

Sackmann *et al.* [170] took these ideas further by considering how different functional forms of protein could be modelled within the PN framework. In particular, by means of a case study on the mating pheromone response pathway for budding yeast, they showed how structural analysis could provide a meaningful mechanism for network decomposition; something which they argued was crucial for understanding large systems. This decomposition based on structural properties was recently explored further in [80], where T-invariants were used to break up biochemical networks in to modules. Specifically, cluster analysis was used to biologically classify these T-invariants, and the approach was evaluated using both a *signal transduction pathways (STPs)* and GRN with promising results.

Work by Gilbert *et al.* [74] has also demonstrated how PNs can be used for both qualitative and quantitative analysis. Furthermore, their work highlighted how quantitative analysis can often be complemented by qualitative analysis. Gilbert *et al.* showed how qualitative techniques could prove a useful first step in deriving more detailed quantitative models by proposing a bridge between PNs and ordinary differential equations. In particular, they demonstrated how standard PNs could be used to derive the initial molecular concentrations of the continuous differential equation representation via an in-

termediate continuous PN model. This technique was then applied to model the inuence of the Raf Kinase Inhibitor protein on the extracellular signal Regulated Kinase signalling pathway, where they were able to identify all reasonable initial amounts which resulted in the desired behaviour specified in temporal logic. This work was later extended in a comprehensive report which considered the application and comparison of three PN approaches to modelling this pathway [123].

Hardy *et al.* [85] applied PN theory to develop a new technique for analysing the dynamics of signal propagation in STPs. In particular, they demonstrated how their approach could be used to infer temporal information about the signals in the pathway, and they developed a simplified graphical representation of the network. Moreover, they showed how their approach could be used to classify signaling routes in the network. Dill *et al.* [58] developed a PN framework for STPs, and implemented their theory in a support tool PATHALYZER, which was able to calculate reachable pathways, perform knockout analysis and calculate relevant subnets.

Moving away from STPs, PN theory has also been used in the study of GRNs where the semantics of interactions are different (regulators are usually not consumed) [44,180]. This therefore exemplies the flexibility of PNs in modelling a range of fundamentally different systems. Chaouiya *et al.* [45,165] considered the development of a rigorous rewriting mechanism from BNs to PNs in order to address the shortcomings of the former. In addition, focusing on the asynchronous update semantics of BNs, they outlined how redundant transition removal and *logic minimisation* [32] could be used to reduce the size of the PN model. They then considered a detailed case study which looked at the construction and analysis of a PN model of the *Drosophila* cell cycle of the flowering *Arabidopsis*, where reachability analysis techniques were applied to verify known properties. However, their approach does not appear to apply logic minimisation systematically to reduce the constructed PN, nor does their PN architecture scale to large models. Furthermore, they do not appear to make provisions for partial model, as they replace the non–deterministic choices with deterministic ones. In Chapter 3, we address these three issues by building upon and extending their approach.

Chaouiya *et al.* later extended their initial rewriting approach in [44] and generalised it to cater for multi-level logic. This approach was then applied by Simao *et al.* [187] to the regulated tryptophan biosynthesis pathway in *E. coli* and by Faure *et al.* to the budding yeast cell cycle [64] and mammalian cell cycle [65] (in the latter, Faure *et al.* also provided an interesting discussion on the affect of synchronous and asynchronous updating, and considered a combination of the two by introducing a notion of *priority classes*). Here, non-safe PNs were used to cater for the possibility of entities with multiple states, and the primary focus was on the integration of metabolic and regulated systems under the PN formalism. Their case study results indicated a close agreement with experimental knowledge, increasing confidence in the application of PNs in this setting.

Finally, Gambin *et al.* [69] proposed an approach for finding all steady states of a PN model with a particular application to GRNs. They demonstrated how their fully automated method was able to find all steady states in a model by considering a case study on the flower morphogenesis of *A. thaliana*.

In some cases, however, standard PNs may be too low level to describe some of the

complex dynamics of regulatory systems in a concise and intuitive manner. Indeed, some approaches such as that by Peleg *et al.* [154] have considered combining PNs with other formalisms such as workflows, to gain the benefits of both formalisms. On the other hand, a number of extensions to the underlying PN framework have also been proposed over the years, and we will now review some of these in the context of biological modelling.

2.4.2 Coloured Petri nets

A *coloured Petri net (CPN)* [97] extends the underlying PN by allowing places to contain distinguishable data types called *colours*, and Boolean expressions to describe both the enabling conditions and firing results of transitions. As such, CPNs provide a more expressive formalism than standard PNs, and facilitate the description of complex dynamics whilst maintaining a relatively simplistic structure. Furthermore, a key advantage of CPNs is that they are amenable to established techniques from PN theory [48,49,152].

Genrich *et al.* [72] proposed a CPN modelling framework for MPs to address the difficulty in visualising network structures from differential equations. Their approach allowed for the semi-automatic construction of a CPN model using data from the BRENDA database on enzymatic reactions (see *http://www.brenda-enzymes.info/*). In particular, they used colour to differentiate the name of the substrate and its concentration, and considered two modelling frameworks using CPNs. They evaluated both approaches for their simulation efficiency, and found that a more compact representation using a single transition in which the metabolic dynamics were encoded yielded significantly faster simulation times, but that it could be unintuitive to see pathway structures.

Heiner *et al.* [91,213] developed an alternative CPN approach for MPs in which colour was used to distinguish the source and destination of molecules of the same metabolite. In particular, they focused on qualitative analysis by studying the P- and T-invariants of the underlying net, and use a combination of structural and dynamic analysis to compute and study its steady states. Their results showed the correct confirmation of metabolite preservation laws as well as regenerative reactions.

In a similar manner, Taubner *et al.* [201] considered applying CPNs to model, simulate and analyse the TLR4 pathway, but took a more manual approach. They started by developing an object-orientated view of the pathway using UML diagrams to capture its static structure. Then, they looked at the iterative construction of its dynamics by considering each pathway interaction individually from a database. In particular, this work demonstrated the importance of human judgement in the modelling process, since a complete knowledge of such pathways is often unavailable from the database alone.

Comet *et al.* [53] proposed a CPN framework for modelling GRNs based on Thomas' generalised logical approach [206]. Their approach encoded the regulatory dynamics of the GRN into a single transition, and they used colours to represent model parameters. The idea was then to generate all possible models, and to filter these by specifying hypotheses expressed as temporal logic formulae. Only the models whose dynamical behaviour fitted the logic specified remained after such a process, and these were then analysed further.

Finally, in [122], Lee *et al.* included a timing element to quantitatively model and

analyse the epidermal growth factor signalling pathway, and provided an interesting discussion about the trade-offs between modelling power and analysis efficiency. Moreover, their results showed a close agreement with those obtained by differential equation approaches.

2.4.3 Stochastic Petri nets

In order to cope with the stochasticity of molecular interactions, *stochastic Petri nets (SPNs)* [16,79,192] have been developed. SPNs extend the underlying PN by associating an exponentially distributed timing delay to each transition; the same rules for enabling and firing still apply. As such, SPNs are suitable for capturing quantitative descriptions of small interacting systems [138,193], are amenable to probabilistic simulation and model checking techniques and tools such as PRISM [117], and are biologically justified [15, 76, 79,128].

The first attempt at applying SPNs to biological systems appears to be by Goss and Peccoud [79], who motivated their use as a format for replicating, transfering and extending knowledge between researchers. The authors used SPNs to model and simulate the affect of the Rom protein on ColE1 plasmid replication by making use of the MOBIUS tool (available from *http://www.mobius.uiuc.edu*). However, whilst their study motivated the applicability of SPNs for small molecular systems, it did not appear to yield any further insights over differential equation approaches of the same system.

Srivastava *et al.* [192] used SPNs to model the σ^{32} stress circuit in *E. coli* in the presence of heat and ethanol shock. Their SPN model was able to reproduce data which corresponds well to that observed by experimentation. Shaw *et al.* [178,180] addressed the problem of parameter estimation and stochastic simulation costs by employing SPNs. In particular, they developed a distributed stochastic simulator using the CONDOR platform [202]. They evaluated their approach using a case study on the stress pathway in *E. coli* which was able to correctly derive a coherent set of model parameters in correspondence with knowledge from the literature. However, it also identified a number of improvements relating to the efficiency of the genetic algorithm, which need to be addressed for their approach to scale to more practical models.

Gilbert *et al.* [75, 123] described a unifying framework for biological systems based on SPNs, which amalgamated the qualitative, continuous and stochastic PN approaches to modelling. The result was a family of models with a high analytical power. They showed the relationship between their SPN model and a standard PN and continuous representation using differential equations, and then evaluated these three approaches on a case study of the ERK signally pathway, with a particular emphasis on the model checking that could be performed.

2.4.4 Hybrid Petri nets

To address the lack of tools and techniques for managing ordinary differential equation models of biological systems, *hybrid Petri nets (HPNs)* [132] have been proposed. HPNs extend the PN formalism by including continuous places with associated real values in addition to standard discrete ones. Furthermore, they allow for continuous transitions

which fire continuously at some fixed rate, whilst discrete transitions can fire after a determined delay. As such, HPNs provide a rich and expressive environment for modelling biological networks both qualitatively and quantitatively.

Perhaps the most active work in this area has been that of Matsuno and collaborators [60,132,134,135,149,150]. In [132], a HPN framework for GRNs was proposed. This framework built upon and extended the hybrid circuit approach of McAdams *et al.* [139] by enabling both circuit and differential equation descriptions to be united under a common formalism which was then amenable to analysis. They showed how this framework could be utilised to incrementally develop models in a modular fashion, and illustrated their approach with a case study on the early stages of gene expression in phage λ.

A further extension to HPNs (based on [209]) was proposed by Matsuno *et al.* specifically for biological systems, namely *hybrid functional Petri nets (HFPNs)* [135, 150]. HFPNs allow for the firing rates of transitions to depend on the values of the corresponding input places, and are supported by the dedicated tools GON (Genomic Object Net) and CELL ILLUSTRATOR [131, 149].

HFPNs were proposed as an integrating approach for so-called biopathways (GRNs, MPs and STPs) in [59,135]. Here, HFPNs and the support tool GON [131] were evaluated against a number of biopathways, allowing for detailed simulations to be carried out. In [150], HFPNs were further extended with notions of object types, variables and methods, and were used to simulate various biological processes including Huntington's disease. In [133], Matsuno *et al.* used HFPNs to suggest the addition of an interaction in the mammalian circadian gene regulatory mechanism which was not currently known about, and showed that its presence resolved a number of observed discrepancies.

Whilst HFPNs offer a unified framework for biological processes, they appear to have a number of shortcomings. Firstly, their analysis appears to be currently limited to simulation, which will not scale for large models. In fact, it was Reddy who noted the trade-off between modelling power and analysis potential [163]; that is, as the expressiveness of a model increases, its availability to analysis techniques significantly reduces. This is exemplified by the current lack of analysis techniques for HFPNs such as model checking and invariant analysis. Moreover, there appears to be no suitable way in which HFPNs can be translated into simpler varieties of PN so that they are amenable to such techniques.

A second hinderance with HFPNs is that there seems to be no provision for the automatic model construction required for practical models, since many parameters and significant user knowledge is needed. One attempt at estimating parameters using HFPNs was proposed by Koh *et al.* [116], who used a decompositional approach based on the network topology to partition the model into subnets and estimate the parameters independently. Their results indicated significant improvements in estimation over existing attempts, but more case studies are clearly required to gain confidence in the biological assumptions made.

Finally, there appears to be little theoretical backing for the HFPN formalism which proves its completeness and justifies its correctness in the biological context. Moreover, the simulation algorithms do not appear to be documented, so it is hard to assess the quality of the results obtained.

2.5 Tool support for modelling and analysing biological systems

In this section, we briefly review some of the existing software support for constructing and analysing models of biological systems.

GINsim

GINSIM [78] is a modelling and simulation tool for GRNs based on Thomas' generalised logical approach [206]. A regulatory network can be specified intuitively by drawing the genes and their interactions as a directed graph. For each gene, the number of states can be specified, and arcs can be assigned with activation or inhibitory influences, along with the thresholds at which they become realisable. Once the genes and interactions are specified, GINSIM provides a number of functions to the user. Simulations may be run on the network, resulting in the generation of the state graph under both the synchronous and asynchronous update semantics, as well as priority classes. Searches for functional circuits as well as steady states can also be performed, as well as the translation to PNs and the export to a number of formats for further analysis. GINSIM is freely available from *http://gin.univ-mrs.fr/*.

GNA

GNA [21] allows for the specification of a regulatory network using PLDEs [23,40,71,77, 99,143]. The equations enable the qualitative description of the network to be specified using thresholds of influence under which the levels of biological entities tend towards some target value. GNA is then able to perform qualitative simulations on the system of equations, calculate steady states and export the model to a number of different formats for model checking with external tools. GNA is freely available from *http://www-helix.inrialpes.fr/article122.html*.

Cell Illustrator

CELL ILLUSTRATOR [131] is based on HFPNs and allows the user to comprehensively model a range of biological systems by providing a powerful drawing canvas onto which the biological entities, variables and interactions can be specified. CELL ILLUSTRATOR is then able to perform efficient simulations on the constructed models, draw graphs of the results and produce publication quality illustrations. Furthermore, CELL ILLUSTRATOR interfaces to a number of biological databases and supports the import of models in a range of formats including SBML and CellML. CELL ILLUSTRATOR is available commercially from *https://www.cellillustrator.com*.

E-Cell

E-CELL is an object-orientated software suite resulting from the collaboration of a number of international research groups, which aims at providing precise whole cell simulation. E-CELL consists of modelling, analysis and simulation modules which form part of a well-developed architecture, and interfaces with various data sources. In particular, a number

of simulation algorithms and analysis techniques appear to have been implemented as plugins for this growing software suite, which is currently maintained by a substantial team of developers. E-CELL is available commercially from *http://www.e-cell.org*.

Biocham

BIOCHAM [38] is a rule-based environment for specifying and simulating biochemical systems. A model can be specified using a rule-based syntax, and then simulated and analysed using powerful temporal logics. Furthermore, BIOCHAM provides an interface to external model checkers as well as a machine learning system for auto-completing models and estimating their parameters. BIOCHAM is freely available for academic use from *http://contraintes.inria.fr/biocham*.

2.6 Summary

This chapter has introduced BNs and PNs, the formalisms at the core of this thesis, as well as some of the key studies in which these have been applied in the literature. These studies have reflected some of the substantial work on qualitative modelling and analysis; in particular, the ability to construct models of biological systems, analyse them and then refine them in light of additional knowledge forms an important model development cycle. However, these studies have also highlighted an apparent lack of systematic PN model construction techniques. In the next chapter, we address this by combining the BN and PN formalisms introduced into a systematic modelling framework for GRNs, and develop much-needed tool support to make this available to the biological community.

Chapter 3

Boolean Models of Genetic Regulatory Networks: A Petri Net Approach

Boolean networks (BNs) provide an abstract modelling approach for *genetic regulatory networks (GRNs)* that allow each regulatory entity to be in one of two states [101, 102]. Due to this simplistic view, BNs provide many modelling advantages over finer-grained approaches: (i) they are scalable; (ii) they support the observation that genes exhibit switch-like behaviour [105, 182, 203]; (iii) they are robust to noisy data sets [138, 183]; and (iv) they capture fundamental behaviour, making them ideal for initial studies. However, there appears to be limited formal analysis techniques and tools in the biological community for studying BNs, and they do not cope well with the uncertainty that plagues most practical data sets.

Petri nets (PNs) [147] have emerged as a rich framework for modelling biological systems (see [84, 155, 160] for interesting reviews). With their strong mathematical foundation, non–deterministic firing semantics and abundant analysis techniques and tool support, PNs appear to be well-suited for this area. Moreover, by translating BNs into PNs (see, for example, [45, 165]), the latter has been shown to address the shortcomings of the former. Despite this, little appears to have been done in ensuring that model construction is systematic and compact. Furthermore, there appears to be a lack of support for *both* the synchronous and asynchronous semantics of BNs, including partial models.

Steggles *et al.* [194–196] built upon and extended [45, 165] by considering the application of efficient *logic minimisation techniques* [32, 127] to obtain compact PN models, catering for both the synchronous and asynchronous semantics of BNs and coping with non–determinism. The result was a flexible qualitative PN modelling framework for BNs. However, efficient tool support is now required to make this approach systematic and practical, and this is the key contribution of this chapter.

We introduce a tool called GNAPN (Genetic Networks as Petri Nets) which completely automates the theory reported in [196]. We then consider investigating and validating its application using two comprehensive case studies: the first considers the GRN responsible for carbon stress response in *E. coli*; and the second focuses on the

larger GRN responsible for initiating sporulation in *B. subtilis*. In particular, we apply a range of existing PN analysis techniques and tools to verify known properties of our models, and hence gain confidence in the correctness of the approach.

The rest of this chapter is structured as follows. In Section 3.1, we describe the systematic approach to translating BNs into PNs based on [194–196]. In particular, we show how both the synchronous and asynchronous semantics of BNs are catered for in the PN model, as well as how partial models can be represented in a useful way. Then, in Section 3.2, we discuss how this approach is automated by our Java tool GNAPN. We then investigate and validate the application of GNAPN in Section 3.3 using two case studies on carbon starvation in *E. coli* and sporulation in *B. subtilis*. Finally, Section 3.4 presents some concluding remarks.

3.1 From Boolean networks to Petri nets

In this section, we present the systematic approach for translating BNs into PNs proposed in [194–196]. We firstly show how the next-state functions F describing the regulatory relationships between each entity in a BN can be specified more compactly using *logic minimisation techniques* [32, 127]. We then show how this more compact representation can be used to construct PN models.

3.1.1 Obtaining compact next-state equations

Let $\mathcal{BN} = (G, N, F)$ be a BN with k entities as defined in Section 2.2.2 . The next-state functions F can be completely specified by k state transition tables, or by a single global table with 2^k rows. For example, consider the state transition tables shown in Figure 3.1 for a BN with three entities.

g_2	$[g_1]$
0	0
1	1

g_1	g_3	$[g_2]$
0	0	0
0	1	0
1	0	0
1	1	1

g_1	$[g_3]$
0	1
1	0

Figure 3.1: State transition tables from Figure 2.1 for three entities.

Alternatively, the behaviour of these state transition tables can be represented equationally as:

$$[g_1] = g_2, \quad \overline{[g_1]} = \overline{g_2}, \quad [g_2] = g_1\, g_3, \quad \overline{[g_2]} = \overline{g_1}\,\overline{g_3} + \overline{g_1}\, g_3 + g_1\,\overline{g_3}, \quad [g_3] = \overline{g_1}, \quad \overline{[g_3]} = g_1,$$

which specify the conditions for each entity g_i to be *on* and *off* in the next state $[g_i]$. For example, we see that g_2 will be on in the next state if $g_1 = 1$ and $g_3 = 1$. Each logical conjunction of variables is called a *term* [32], and the logical disjunction of such terms captures the behaviour for the corresponding entity in *disjunctive normal form* [32].

Collectively, such equations completely capture the behaviour of each entity in the BN, but often contain redundant logic which can be removed without changing the behaviour they specify. Since our PN construction approach relies on the translation of this logic, it is therefore advantageous to ensure that such redundancy is removed. This can be achieved with the application of *logic minimisation techniques* [32, 127], which enable an expression to be syntactically simplified whilst preserving its semantics. The basic idea is to eliminate redundancy by merging terms which differ by only one variable in an exhaustive fashion. For example, the only equation above that contains more than one term is for when g_2 turns off, and so we will look at how it can be minimised into a more compact description.

We start by considering the first term $\overline{g_1}\ \overline{g_3}$ and notice that it can be merged with the second term $\overline{g_1}\ g_3$, since it differs in only one variable g_3. This therefore results in the compact term $\overline{g_1}$ which is logically equivalent [32] to the original two. In a similar way, we can also merge the first term with the third $g_1\ \overline{g_3}$, resulting in the compact term $\overline{g_3}$. We then consider the second term $\overline{g_1}\ g_3$ and compare it to the last term only (since we have already compared it to the first) and note that they differ in both variables, meaning that they can not be merged. Finally, after one complete pass, we are left with two terms $\overline{g_3} + \overline{g_1}$ which cannot be merged any further, and which are logically equivalent to the original expression [32]. Thus, our complete set of next-state equations after minimisation is:

$$[g_1] = g_2, \quad \overline{[g_1]} = \overline{g_2}, \quad [g_2] = g_1\ g_3, \quad \overline{[g_2]} = \overline{g_1} + \overline{g_3}, \quad [g_3] = \overline{g_1}, \quad \overline{[g_3]} = g_1.$$

These minimised next-state equations completely describe the dynamics of each entity in Figure 3.1 in a compact way, which is advantageous for the PN translation discussed in the next section. For a comprehensive introduction to logic minimisation, see [32].

3.1.2 Constructing a qualitative Petri net model

We now consider how these compact next-state equations can be represented as PN structures, so that their logic can be simulated and analysed using the wide range of available PN techniques and tools [1]. In particular, this work extends similar approaches [45] by catering for *both* the synchronous and asynchronous semantics of BNs. Note we focus on constructing *safe* PNs (where each place can contain no more than one token) since they lend themselves better to automated analysis.

Asynchronous update semantics

Since PNs fire transitions asynchronously, it is straightforward to model the asynchronous update semantics of a BN in this setting. We adopt the standard approach based on representing the Boolean state of each entity g_i using two complementary places p_i and $\overline{p_i}$ (for example, see [45]). The idea is that g_i is on if p_i contains a token, and off if $\overline{p_i}$ contains a token (note the combined number of tokens on places p_i and $\overline{p_i}$ is always one). The PN construction process for a given BN can then be specified formally as follows.

Definition 3.1 (Asynchronous Boolean network to Petri net). *Assume we have a BN with n entities $g_1, \ldots, g_n$. A corresponding PN modelling its asynchronous update semantics can be constructed as follows:*

- *For $i = 1, \ldots, n$, we add two complementary places p_i and $\overline{p_i}$, such that $M(p_i) + M(\overline{p_i}) = 1$ always holds.*
- *Consider each minimised next-state equation $[g_i] = T_1 + \cdots + T_m$ which defines when g_i turns on. For each product term T_j, for $j = 1, \ldots, m$, we add a transition $t_{i/j}$ such that: (i) place $\overline{p_i}$ is an input place and p_i is an output place of $t_{i/j}$; and (ii) for each variable g_k (respectively $\overline{g_k}$) in T_j, we add a read arc connecting p_k (respectively $\overline{p_k}$) to $t_{i/j}$. The same approach is then used to model the minimised next-state equation $\overline{[g_i]} = T_1 + \cdots + T_m$ which defines when g_i turns off. We add a transition $\overline{t_{i/j}}$ to model each product term T_j, for $j = 1, \ldots, m$, using the same scheme as detailed above, but with $\overline{p_i}$ (respectively p_i) replaced by p_i (respectively $\overline{p_i}$) in part (i).*

As an example, consider the compact next-state equation for g_2:

$$[g_2] \;=\; g_1\, g_3,$$

which we would model using transition t_1 in Figure 3.2(a). Note transition t_1 is only enabled when $M(\overline{p_2}) = 1$, $M(p_1) = 1$ and $M(p_3) = 1$. When t_1 fires, a token is then removed from $\overline{p_2}$ and placed on p_2. We can then apply the same approach for when g_2 turns off, by considering the next-state equation:

$$\overline{[g_2]} = \overline{g_1} + \overline{g_3},$$

where transition t_2 represents term $\overline{g_1}$ and t_3 represents term $\overline{g_3}$.

Applying the construction process of Definition 3.1 to the remaining entities yields a PN modelling the asynchronous semantics of our example BN. More specifically, any sequence of global states obtained from asynchronous simulation of the BN can be represented by a corresponding sequence of markings in the constructed PN. Furthermore, it can be observed from Definition 3.1 that if we have a BN with n entities, where each has an on and off minimised next-state equation consisting of a total of m terms, then the corresponding PN will contain $2n$ places and mn transitions (equivalent to [45]). In contrast, without logic minimisation, the resulting PN would contain $(2^n)n$ transitions to cater for each possible state update that could occur.

Synchronous update semantics

In order to model the synchronous update semantics of a BN within the asynchronous PN framework, we make use of a two-phase commit protocol to synchronise updates in the PN model. In phase one, each entity decides what its next state will be and records this decision. When all the entities have made a decision about their next states, the second phase of the protocol begins, and the state of each entity is updated according to the recorded decisions. The process for constructing such a PN can be specified formally.

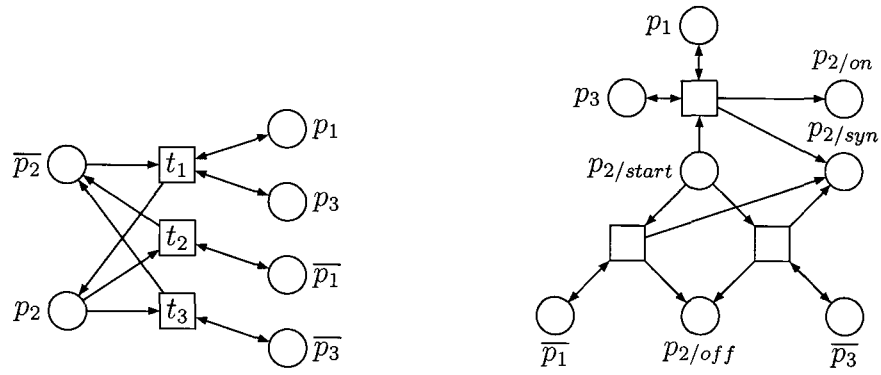


Figure 3.2: (a) PN structure for asynchronous semantics with transition t_1 modelling next-state equation $[g_2] = g_1\ g_3$, and transitions t_2 and t_3 modelling next-state equation $\overline{[g_2]} = \overline{g_1} + \overline{g_3}$, and (b) PN structure for phase one of synchronous semantics modelling next-state equations $[g_2] = g_1\ g_3$ and $\overline{[g_2]} = \overline{g_1} + \overline{g_3}$.

Definition 3.2 (Synchronous Boolean network to Petri net). *Assume we have a BN with n entities $g_1, \ldots, g_n$. A corresponding PN modelling its synchronous update semantics can be constructed as follows:*

- *For $i = 1, \ldots, n$, we add places p_i, $\overline{p_i}$, $p_{i/on}$ (to record g_i as on), $p_{i/off}$ (to record g_i as off), $p_{i/start}$ (to indicate when a decision about the next state of g_i is required), $p_{i/syn}$ (to indicate when an update decision has been made), $p_{i/com}$ (to indicate when the recorded decision should be committed) and $p_{i/done}$ (to indicate that the decision has been committed). Note in the initial marking, either p_i or $\overline{p_i}$ (but not both) must contain a token, and $p_{i/start}$ will contain a token, for $i = 1, \ldots, n$. All other places must be unmarked.*

- ***Phase One**: Consider each minimised next-state equation $[g_i] = T_1 + \cdots + T_m$ which defines when g_i turns on. For each product term T_j, for $j = 1, \ldots, m$, add a transition $t_{i/j}$ such that:*

  - *place $p_{i/start}$ is an input place and $p_{i/on}$ and $p_{i/syn}$ are output places of $t_{i/j}$;*
  - *for each variable g_k (respectively $\overline{g_k}$) in T_j, add a read arc connecting place p_k (respectively $\overline{p_k}$) to $t_{i/j}$.*

  We now use the same approach to model the minimised next-state equation $\overline{[g_i]} = T_1 + \cdots + T_m$ which defines when g_i turns off. We add a transition $\overline{t_{i/j}}$ to model each product term T_j, for $j = 1, \ldots, m$, using the same scheme as detailed above, but with $p_{i/on}$ replaced by $p_{i/off}$ as an output place for the transition.

- ***Phase Two**:*

- *We add a transition to initiate the update process which has input places $p_{i/syn}$ and output places $p_{i/com}$, for $i = 1, \ldots, n$.*
- *For each entity g_i, we add four transitions to update the state of g_i based on the recorded decisions:*
  - *move token from place $\overline{p_i}$ to p_i;*
  - *leave token on p_i;*
  - *move token from place p_i to $\overline{p_i}$;*
  - *and leave token on $\overline{p_i}$.*

  In addition, each transition takes input from place $p_{i/com}$ and outputs to $p_{i/done}$.
- *We add a transition which has input places $p_{i/done}$, for $i = 1, \ldots, n$, and output place p_{finish} (used as a convenient means of indicating when phase two has finished). An additional transition is then used which has input place p_{finish} and output places $p_{i/start}$, for $i = 1, \ldots, n$ which resets the protocol.*

As an example, Figure 3.2(b) shows the PN structure of phase one used to record the update decision for g_2. Once g_1 and g_3 have also made their decision (in a concurrent fashion), places $p_{1/syn}$, $p_{2/syn}$ and $p_{3/syn}$ will be marked, allowing the transition in Figure 3.3(a) to start phase two of the protocol. With places $p_{1/com}$, $p_{2/com}$ and $p_{3/com}$ marked, each entity is then able to commit the recorded changes concurrently. For example, the four transitions required to achieve this for g_2 are shown in Figure 3.3(b). Finally, once all three entities have committed their recorded changes, the transition shown in Figure 3.3(c) will be able to fire. Note place p_{finish} will be marked in between phase two finishing and phase one beginning, and is simply used for model checking convenience (for example, see the case study in Section 3.3.1).

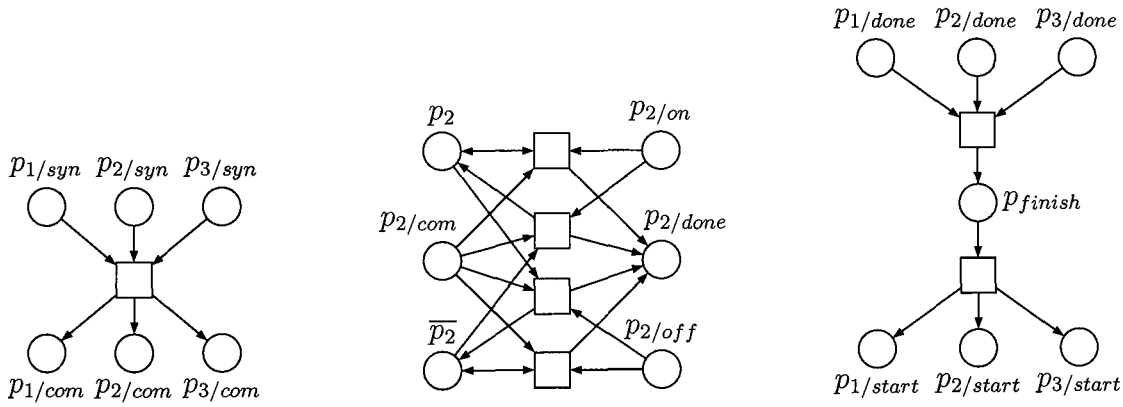


Figure 3.3: PN structures for phase two of the protocol using running example: (a) initiating phase two; (b) update of entity g_2; and (c) reset network for phase one.

A PN constructed from a BN using the process of Definition 3.2 will model the synchronous update semantics. More specifically, any sequence of global states resulting

from a synchronous simulation of the BN will be representable in the PN by considering only those markings reachable after complete passes of the two-phase protocol (all other intermediate markings are necessary only for the synchronisation logic, and can thus be safely ignored). Intuitively therefore, if we were to hide the extra places and transitions involved in the synchronisation logic of the two-phase protocol, the sequence of markings we would observe would directly correspond to the global states of the BN under the synchronous update semantics.

It can also be observed from Definition 3.2 that if we have a BN with n entities, where each has an on and off minimised next-state equation consisting of a total of m terms, then the corresponding PN will contain $8n+1$ places and $nm+4n+3$ transitions. Thus, the overhead of the synchronisation logic adds only $4n+3$ extra transitions, and therefore becomes negligible for large BNs where $n << m$ often holds. In particular, we avoid the potential 2^n transitions required to capture each possible state update under the synchronous semantics.

3.1.3 Partial models

So far, we have assumed that we always start with complete and consistent state transition tables. However, this may not always be the case in practice, as the underlying GRN may not be fully understood; indeed, this is one important reason for modelling such systems. The tables may be incomplete in the sense that information is missing about what happens in certain states, or they may be inconsistent in that they contain conflicting information. The result is that the behaviour of some entities under certain conditions may be non–deterministic.

Non–deterministic behaviour is problematic to model for BNs, as they are a deterministic formalism. Work in the literature has looked at a number of potential methods for making BNs more robust to uncertainty, including: stochastic extensions [182, 183]; algorithms for inferring complete BNs from sparse state transition specifications with a noise factor [7, 125, 221]; and noise-resiliant clustering and binarisation approaches [186]. However, to the best of our knowledge, no PN construction approaches appear to cater for this.

We propose a straightforward method for modelling uncertainty which makes use of the non–deterministic firing semantics of PN transitions. As such, our approach requires no extension to the underlying BN. The idea is to identify the problematic states for each entity, and then include these in both next-state equations, thus representing the uncertainty as a *choice* in the PN (see Section 2.3.3). For example, consider the state transition table for some entity g_1 shown in Figure 3.1.3, where the '-' represents an unknown outcome for state $g_2\ \overline{g_3}$.

Since the behaviour of g_1 is unknown at this state, we simply include state $g_2\ \overline{g_3}$ in both next-state equations for g_1:

$$[g_1] = \overline{g_2}\ g_3 + g_2\ \overline{g_3},\quad \overline{[g_1]} = \overline{g_2}\ \overline{g_3} + g_2\ \overline{g_3} + g_2\ g_3.$$

Applying logic minimisation to these next-state equations then gives us:

$$[g_1] = \overline{g_2}\ g_3 + g_2\ \overline{g_3},\quad \overline{[g_1]} = g_2 + \overline{g_3},$$

g_2	g_3	$[g_1]$
0	0	0
0	1	1
1	0	-
1	1	0

Figure 3.4: State transition table representing Boolean behaviour of g_1 with unknown behaviour.

which can be modelled using four transitions under both the asynchronous (see Figure 3.5(a)) and synchronous (see Figure 3.5(b)) update semantics. Note when places p_2 and $\overline{p_3}$ are marked, transitions t_2, t_3 and t_4 are enabled, and so a non–deterministic choice exists between g_1 becoming active or inactive.

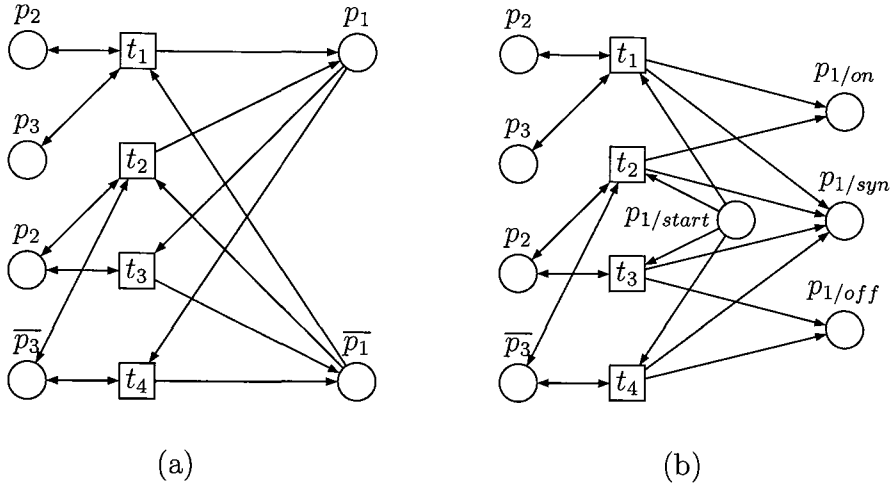


Figure 3.5: PN structures modelling unknown behaviour for g_1 under: (a) the asynchronous semantics; and (b) the synchronous semantics.

Such non–deterministic choices can be taken into account when analysing the dynamic behaviour of a PN, and so meaningful investigations are still possible. Furthermore, as more data becomes available for a GRN, the PN model can be refined to reduce the amount of non-determinism it contains. Thus, PNs provide an interesting means of documenting the progression of knowledge for a GRN.

3.2 Implementation issues

Whilst the modelling approach presented is theoretically well-founded, it remains impractical without support tools to automate model construction. In this section, we discuss the development of such a tool called GNAPN (Genetic Networks as Petri Nets).

GNAPN is a Java tool which completely automates the translation process from a BN representation of a GRN to a PN. BNs can be supplied to GNAPN either by defining the regulatory interactions using a GUI, or automatically from an input file of

state transition tables. In both cases, GNAPN represents the BNs in memory using state transition table data structures (note *binary decision diagrams (BDDs)* [35] offer a promising future extension to GNAPN to allow BNs to be represented more compactly).

A key feature of GNAPN is the application of logic minimisation techniques to the BN prior to PN construction. For this, GNAPN uses the auxiliary tool MVSIS [146]; the state transition tables are serialised and passed transparently to MVSIS, which then processes them and outputs a compact equational representation to file. GNAPN then parses these equations back into data structures, resulting in a set of compact next-state equations.

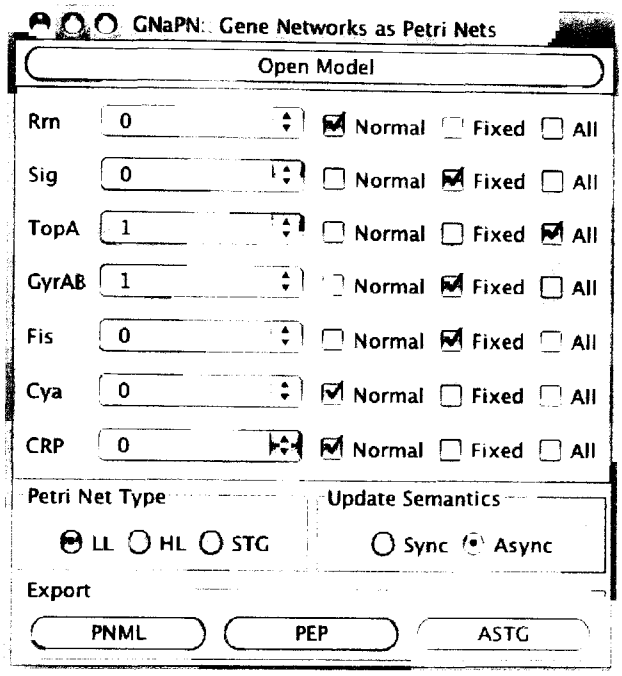


Figure 3.6: GUI of GNAPN allowing user to select initial states, specify update semantics and mutants as well as export formats.

GNAPN facilitates the construction of PNs for *both* the synchronous and asynchronous semantics of BNs. This is a crucial feature which so far does not appear to have been catered for. Furthermore, GNAPN can also cope with partial BNs following the approach described in Section 3.1.3. The GUI (see Figure 3.6) allows the user to specify initial markings for the PN as well as mutant models, by enabling the states of one or more entities to remain fixed once set. Constructed PNs are stored internally as graph structures, which GNAPN can then export to PNML (the *de facto* standard), PEP [81] (for use with the PEP tool) and ASTG [54] (for use with PETRIFY).

It is important to note here that GNAPN is implemented in a generic way that enables it to handle both BNs and their generalisation discussed in Chapter 4. GNAPN is freely available for academic use for Windows and Linux platforms only (since it requires the platform-specific binary MVSIS) and can be downloaded from *bioinf.ncl.ac.uk/gnapn*.

3.3 Applying and validating Petri net framework

We now focus on exploring the application of our PN approach using two independent case studies based on well-studied systems. In the first study, we consider a BN model of the carbon starvation stress response network in *E. coli* [167]. The second case study then considers the larger BN model for the initiation of sporulation in *B. subtilis* [99]. Both studies involve the application of GNAPN and a range of PN techniques and tools. In particular, we make use of the PEP tool [81] to analyse our PN models, as well as the PN unfolder PUNF and linear model checker CLP [109] for dynamical analysis. Note both case studies focus on the synchronous semantics of BNs, as it is interesting to investigate within the asynchronous PN framework.

3.3.1 Response to carbon starvation in *E. coli*

In this section, we apply the techniques introduced to construct and validate a PN model based on an adapted GRN for carbon starvation response in the bacterium *E. coli* [167].

Constructing the Petri net model

Under normal conditions with sufficient nutrient availability, the bacterium *E. coli* is able to develop rapidly entering an *exponential growth phase* [92]. However, under adverse conditions when the nutrient availability is depleted, *E. coli* enters a *stationary phase* in which a substantial slow down in growth occurs to help the bacterium survive. The GRN underlying this response for carbon starvation is shown abstractly in Figure 3.7 (adapted from [167]).

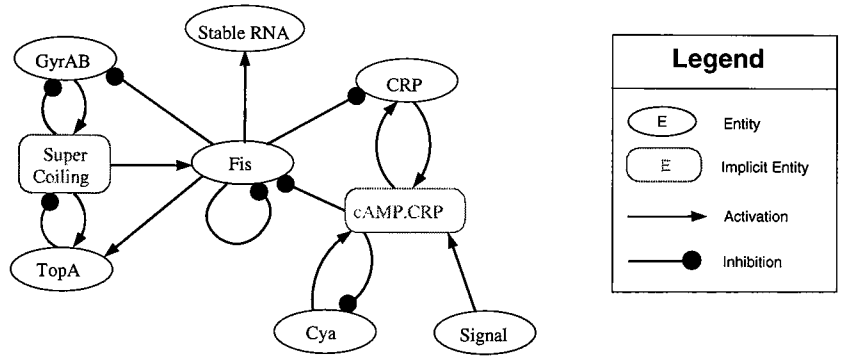


Figure 3.7: High-level GRN for the carbon starvation response network in *E. coli*.

The network has a single input signal indicating the presence of carbon starvation, which is transduced by the activation of adenylate cyclase (Cya), an enzyme which results in the production of the metabolite cAMP. This metabolite immediately binds with and activates the global regulator protein CRP, and the resulting cAMP.CRP complex is responsible for controlling the expression of key global regulators including Fis and CRP itself. The global regulatory protein Fis is central to the stress response and is responsible

for promoting the expression of stable RNA from the *rrn operon* [167]. Thus, during the exponential growth phase, high levels of Fis are normally observed and the mutual repression that occurs between Fis and cAMP.CRP is thought to play a key role in the regulatory network.

The expression of *fis* is also promoted by high levels of *negative supercoiling* being present in the DNA [18]. The level of DNA supercoiling is tightly regulated by two *topoisomerases*: GyrAB (composed of the products of genes *gyrA* and *gyrB*) which promotes supercoiling; and TopA which removes supercoils. An increase in DNA supercoiling results in increased expression of TopA and thus prevents excessive supercoiling. A decrease in supercoiling results in increased expression of *gyrA* and *gyrB*, and the resulting high level of GyrAB acts to increase supercoiling. For a more detailed introduction to the carbon starvation stress response network in *E. coli*, see [167].

Using the data provided in [167], we are able to derive state transition tables defining the Boolean behaviour of each regulatory entity in the carbon stress response network. For example, the state transition table defining the behaviour of Cya is shown in Figure 3.3.1. Note following the approach in [167], the states of cAMP.CRP and DNA supercoiling are not explicitly represented in our model.

CRP	Cya	Signal	[Cya]
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

Figure 3.8: State transition table representing Boolean behaviour of Cya.

Next, we apply logic minimisation to the state transition tables to derive the compact next-state equations describing their regulatory behaviour (see Figure 3.9). From these, we can construct a PN consisting of 45 places and 49 transitions modelling the synchronous update semantics based on the process detailed in Definition 3.2. This process is fully automated by GNAPN, and the model file and resulting PN can be downloaded from *bioinf.ncl.ac.uk/gnapn*.

Simulation

We first consider simulating our PN model to ensure that it captures a reasonable representation of the switch between the two fundamental growth phases [167]. The idea is to initialise our PN model to a given marking of interest, and then simulate it to observe the progression of behaviour after each synchronous state update.

The first simulation we perform is to ensure that the PN correctly switches from the exponential to the stationary growth phase in the presence of carbon stress. We therefore initialise our model to a marking representing the exponential growth phase (see [167])

$$
\begin{aligned}
[Cya] &= \overline{Signal} + \overline{Cya} + \overline{CRP} \\
[\overline{Cya}] &= (Signal\ Cya\ CRP) \\
[CRP] &= \overline{Fis} \\
[\overline{CRP}] &= Fis \\
[GyrAB] &= (\overline{GyrAB}\ \overline{Fis}) + (TopA\ \overline{Fis}) \\
[\overline{GyrAB}] &= (GyrAB\ \overline{TopA}) + Fis \\
[TopA] &= (GyrAB\ \overline{TopA}\ Fis) \\
[\overline{TopA}] &= \overline{GyrAB} + TopA + \overline{Fis} \\
[Fis] &= (\overline{Fis}\ \overline{Signal}\ GyrAB\ \overline{TopA}) + (\overline{Fis}\ \overline{Cya}\ GyrAB\ \overline{TopA}) + \\
&\quad (\overline{Fis}\ \overline{CRP}\ GyrAB\ \overline{TopA}) \\
[\overline{Fis}] &= (CRP\ Cya\ Signal) + Fis + \overline{GyrAB} + TopA \\
[SRNA] &= Fis \\
[\overline{SRNA}] &= \overline{Fis}
\end{aligned}
$$

Figure 3.9: Minimised next-state equations for *E. coli.*

and activate Signal to indicate carbon stress. The resulting simulation showing how the system responds to carbon starvation is shown in Figure 3.3.1; the first column shows the initial state and each subsequent column shows the progression of states after subsequent synchronised updates.

Signal	1	1	1	1	1
CRP	0	0	1	1	1
Cya	1	1	1	0	1
GyrAB	1	0	1	0	1
TopA	0	1	0	0	0
Fis	1	0	0	0	0
SRNA	1	1	0	0	0

Figure 3.10: Simulation results showing the switch from the exponential to the stationary growth phase in the presence of carbon stress.

The simulation shows that the PN model correctly makes the switch from the exponential to the stationary growth phase when carbon starvation stress is present. An attractor cycle of period two is entered (last three columns) in which Fis (and consequently SRNA) is not present at significant levels. Similarly, we can show that the model correctly switches back from the exponential to the stationary growth phase when Signal is inactive. Taking a marking representing stationary phase conditions from Figure 3.3.1, we switch Signal off and observe the resulting behaviour shown in Figure 3.3.1

The simulation shows how the PN model correctly returns back to the exponential growth phase by falling into an attractor cycle in which the level of Fis and SRNA oscil-

Signal	0	0	0	0	0
CRP	1	1	1	0	1
Cya	0	1	1	1	1
GyrAB	0	1	0	0	1
TopA	0	0	0	0	0
Fis	0	0	1	0	0
SRNA	0	0	0	1	0

Figure 3.11: Simulation results showing the switch back from the stationary to the exponential growth phase in the absence of carbon stress.

late. This corresponds to the physiological conditions present in the exponential phase, and is consistent with the behaviour described in [167].

Dynamic properties

We next consider using our PN model to investigate experimental hypotheses which we formulate using the literature and our simulations so far. In particular, we make use of powerful model checking techniques to achieve this. As an example, it appears that GyrAB and TopA should be mutually exclusive [167]; they should never both be active at the same time. In order to prove this property holds on our model, we need to verify that it holds from each possible initial state. We achieve this by inserting a simple PN control structure around the complementary places of each entity to enable us to explore all possibilities. For example, the control structure for entity Fis is illustrated by Figure 3.12.

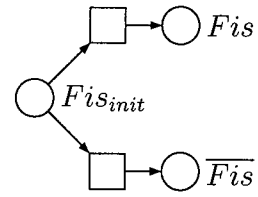


Figure 3.12: PN structure to enable both initial states of entity Fis to be explored.

When Fis_{init} is marked, the model checker CLP is able to explore both markings in which Fis is on and off using the non–deterministic transition firing semantics of PNs. By inserting this control structure around the remaining entities (this is automated by GNAPN), CLP is able to explore all possible behaviours. We then check for mutual exclusion between GyrAB and TopA using the following constraint:

$$GyrAB + TopA > 1,\ p_{finish} = 1$$

which characterises a marking in which the mutual exclusion property does not hold (where the condition $p_{finish} = 1$ is used to ensure we only consider markings reached after a complete pass of the two-phase commit protocol). The results are shown below,

and they correctly confirm that no state satisfying this constraint is reachable from any initial state, thus proving that GyrAB and TopA are indeed mutually exclusive in our model.

```
solving: 0.01 sec.
6 compatible vectors explored

the marking is unreachable
```

We can attempt to prove a similar mutual exclusion property for entities CRP and Fis using the same approach. However, this time a marking satisfying the constraint is confirmed, as shown below:

```
solving: 0.01 sec.
10 compatible vectors explored

 _SEQUENCE:
 t47,t49,t51,t53,t56,t57,t59,t0,t6,t17,t22,t35,t43,t4,t1,t13,t9,
 t20,t19,t31,t26,t39,t38,t45,t44,t0,t8,t15,t29,t35,t43,t4,t1,t12,
 t9,t21,t16,t31,t30
 path length: 39

the marking is reachable
```

This execution trace describes the firing sequence necessary to reach the offending witness marking, thus proving that CRP and Fis are not mutually exclusive in our PN model.

Mutant analysis

The final part of our study investigates the affect of "fixing" the state of a single entity to observe how the normal function of the model is affected. This corresponds with the experimental method of gene knockout and overexpression, and provides a means of testing the robustness of the model when key components do not function as normal. In order to fix the state of an entity, we ignore its corresponding state transition table so that it becomes an input to the model like Signal.

We investigate the affect that knocking out and overexpressing *crp*, *cya*, *gyrAB* and *topA* has on the production of Fis and consequently the expression of the *rrn* operon. Specifically, we look at two situations: where SRNA can be prevented from being active in the absence of carbon stress; and where SRNA can become active in the presence of carbon stress. We perform these tests by first setting Signal, Fis and SRNA to be inactive, and then we knock out and overexpress the remaining entities in turn. We then repeat this process with Signal on. The observed results of this analysis are summarised in Figure 3.3.1, where 'Yes' indicates that SRNA was able to become active, 'No' that this did not occur, and 's' the presence of carbon stress.

Entity	Knock out	Overexpressed	Knock out (s)	Overexpressed (s)
CRP	Yes	Yes	Yes	Yes
Cya	Yes	Yes	Yes	Yes
GyrAB	No	Yes	No	Yes
TopA	Yes	No	Yes	No

Figure 3.13: Results of mutant analysis, where 's' represents the presence of carbon stress.

We see that when both CRP and Cya are knocked out and overexpressed, the level of SRNA is able to become active. This is interesting as it indicates that they are in some way decoupled from the rest of the system under the Boolean abstraction. However, when we knockout GyrAB with no carbon stress, we see a special case in which SRNA does not become active. This can be explained by noting that since GyrAB indirectly activates Fis via supercoiling, TopA is allowed to reduce the amount of supercoiling without competition and so reduces the level of Fis. Meanwhile, the cAMP.CRP complex represses Fis, and so overall SRNA is repressed.

Another interesting case is when carbon stress is present and we overexpress GyrAB. One should note that under normal conditions when Signal is active, SRNA should never become active. However, with GyrAB overexpressed, the level of supercoiling increases without affect from TopA, and thus Fis increases. This increased level of Fis also reduces the amount of the cAMP.CRP complex and so its repression on Fis is reduced. Thus, the level of SRNA is activated under these abnormal conditions, which appears to be consistent with [167].

3.3.2 Sporulation in *B. subtilis*

We further explore our approach with a larger case study which focuses on the GRN responsible for sporulation in *B. subtilis* [99, 197].

Constructing the Petri net model

The soil bacterium *B. subtilis* is able to survive adverse environmental conditions by forming resistant dormant spores [197]. This process of sporulation is controlled by a complex regulatory network, a small part of which is shown abstractly in Figure 3.14 (adapted from [99]).

The presence or absence of extreme environmental conditions, such as nutrient starvation, is represented by the signalling entity Signal: when Signal is present, it indicates that the bacteria is under nutritional stress and that sporulation should occur. At the center of the network is the phosphorylation of the protein SpoOA, which in turn actives a cascade of sigma factors which direct the transcription of genes that initiate sporulation. This *phosphorelay* [99] transfers a phosphate from the kinase *kinA* (amongst others) via a number of intermediate steps to activate the protein SpoOA by producing SpoOA~P. This in turn activates the production of SigF, a key sigma factor whose expression we take as an indication that the sporulation process has been initiated. The phosphotase SpoOE is able to reverse the phosphorylation of SpoOA, thereby inactivating SpoOA

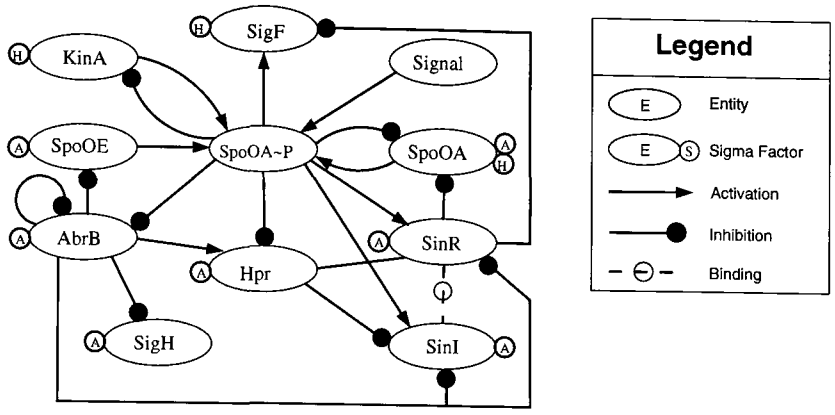


Figure 3.14: High-level GRN for initiating sporulation in *B. subtilis*

and preventing sporulation. The above interactions form part of a complex regulatory control mechanism involving a number of other genes which act as transition state regulators to inhibit or activate the sporulation process. For a more detailed account of the sporulation process, see [99].

Using the data provided in [99], we are able to derive state transition tables defining the Boolean behaviour of each regulatory entity in the sporulation network. For our case study, there are twelve entities: SigF, KinA, SpoOA, SpoOA~P, AbrB, SpoOE, SigH, Hpr, SinR, SinI, SigA and Signal. For example, entity AbrB is affected by three entities, namely SigA, AbrB and SpoOA~P, and its resulting behaviour is shown by the state transition table in Figure 3.3.2.

SigA	AbrB	SpoOA~P	[AbrB]
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	0

Figure 3.15: State transition table representing Boolean behaviour of entity AbrB.

Note SigA and Signal are not regulated within the scope of this network, and as such, their state remains fixed once initialised. Next, we apply logic minimisation to the state transition tables to derive the compact next-state equations describing their regulatory behaviour (see Figure 3.16). From these, we can construct a PN consisting of 75 places and 91 transitions modelling the synchronous update semantics based on the process detailed in Definition 3.2. This process is fully automated by GNAPN, and the model file and resulting PN can be downloaded from *bioinf.ncl.ac.uk/gnapn*.

$$
\begin{aligned}
[AbrB] &= (\overline{AbrB}\ SigA\ \overline{SpoOA{\sim}P}) \\
[\overline{AbrB}] &= SpoOA{\sim}P + \overline{SigA} + AbrB \\
[Hpr] &= (AbrB\ SigA\ \overline{SpoOA{\sim}P}) \\
[\overline{Hpr}] &= SpoOA{\sim}P + \overline{SigA} + \overline{AbrB} \\
[KinA] &= (SigH\ \overline{SpoOA{\sim}P}) \\
[\overline{KinA}] &= SpoOA{\sim}P + \overline{SigH} \\
[SigF] &= (SigH\ SinI\ SpoOA{\sim}P) + (SigH\ \overline{SinR}\ SpoOA{\sim}P) \\
[\overline{SigF}] &= \overline{SpoOA{\sim}P} + \overline{SigH} + (\overline{SinI}\ SinR) \\
[SigH] &= (\overline{AbrB}\ SigA) \\
[\overline{SigH}] &= \overline{SigA} + AbrB \\
[SinI] &= (\overline{AbrB}\ \overline{Hpr}\ SigA\ \overline{SinI}\ \overline{SinR}\ SpoOA{\sim}P) + \\
&\quad (\overline{AbrB}\ \overline{Hpr}\ SigA\ SinI\ SinR\ SpoOA{\sim}P) \\
[\overline{SinI}] &= \overline{SpoOA{\sim}P} + \overline{SigA} + Hpr + AbrB + (SinI\ \overline{SinR}) + (\overline{SinI}\ SinR) \\
[SinR] &= (\overline{AbrB}\ \overline{Hpr}\ SigA\ \overline{SinI}\ \overline{SinR}\ SpoOA{\sim}P) + \\
&\quad (\overline{AbrB}\ \overline{Hpr}\ SigA\ SinI\ SinR\ SpoOA{\sim}P) \\
[\overline{SinR}] &= \overline{SpoOA{\sim}P} + \overline{SigA} + Hpr + AbrB + (SinI\ \overline{SinR}) + (\overline{SinI}\ SinR) \\
[SpoOA] &= (SigH\ SinI\ \overline{SpoOA{\sim}P}) + (SigH\ \overline{SinR}\ \overline{SpoOA{\sim}P}) + \\
&\quad (SigA\ \overline{SpoOA{\sim}P}) \\
[\overline{SpoOA}] &= SpoOA{\sim}P + (\overline{SigA}\ \overline{SigH}) + (\overline{SigA}\ SinI\ SinR) \\
[SpoOA{\sim}P] &= (KinA\ Signal\ SpoOA\ \overline{SpoOE}) \\
[\overline{SpoOA{\sim}P}] &= SpoOE + \overline{SpoOA} + \overline{Signal} + \overline{KinA} \\
[SpoOE] &= (\overline{AbrB}\ SigA) \\
[\overline{SpoOE}] &= \overline{SigA} + AbrB
\end{aligned}
$$

Figure 3.16: Minimised next-state equations for *B. subtilis*.

Simulation

The main control structure is the phosphorelay [197], which is responsible for the phospory-lation of SpoOA under nutritional stress conditions to activate the expression of *sigF*, thus initiating sporulation. To assess whether this control structure has been correctly modelled, we perform some straightforward simulations to ensure that they capture the fundamental behaviour documented in [99].

We begin by considering an initial marking representing vegetative growth conditions, and we turn Signal on to signify the presence of nutritional stress. The sequence of states resulting from this simulation is presented in Figure 3.3.2, where the first column of the table represents the initial marking, and each subsequent column the next observed marking after a synchronised update.

SigF	0	0	0	0	0	1	0
KinA	0	0	0	1	0	0	0
SpoOA	1	1	1	1	1	0	1
SpoOA~P	0	0	0	0	1	0	0
AbrB	1	0	1	0	1	0	1
SpoOE	1	0	1	0	1	0	1
SigH	0	0	1	0	1	0	1
Hpr	1	1	0	1	0	0	0
SinR	1	0	0	0	0	0	0
SinI	0	0	0	0	0	0	0
Signal	1	1	1	1	1	1	1
SigA	1	1	1	1	1	1	1

Figure 3.17: Simulation results in the presence of nutritional stress.

The simulation shows that SpoOA~P accumulation is allowed to occur and SigF production is correctly activated, indicating that the bacterium is able to sporulate. We then turn Signal off to represent the absence of nutritional stress, and simulate the model again. The results of this simulation are shown in Figure 3.3.2.

SigF	0	0	0	0	0
KinA	0	0	0	1	0
SpoOA	1	1	1	1	1
SpoOA~P	0	0	0	0	0
AbrB	1	0	1	0	1
SpoOE	1	0	1	0	1
SigH	0	0	1	0	1
Hpr	1	1	0	1	0
SinR	1	0	0	0	0
SinI	0	0	0	0	0
Signal	0	0	0	0	0
SigA	1	1	1	1	1

Figure 3.18: Simulation results in the absence of nutritional stress.

The simulation correctly shows that sporulation does not occur when Signal is absent; SpoOA~P is not produced from the phosphorelay and so *sigF* is not expressed. It also shows that the model enters an attractor cycle of period two which describes the physiological conditions present in the exponential growth phase [99].

Dynamic properties

We further validate our PN model using the model checker CLP to confirm that basic properties are respected. For example, we know that in the absence of nutritional stress, sporulation should never be initiated [99]. We can use CLP to show that this holds

by proving that no reachable marking exists in which SigF is on from any initial state where Signal is off, SigF is off and SpoOA∼P is off. We achieve this by inserting the PN control structure shown in Figure 3.12 around the two complementary places of every other entity whose initial state we wish to explore, and then verify that SigF cannot turn on by checking the following condition:

$$SigF + p_{finish} = 2$$

where p_{finish} is used to ensure that we only consider states reached after a complete pass of the protocol. Having explored every possible behaviour vector, CLP is able to correctly verify that SigF never becomes active.

Having gained some confidence in the correctness of our model, we now investigate an important relationship noted in [99] between SigF and SpoOA∼P; the phosphorylated protein SpoOA∼P is reported to activate the production of SigF but also repress its own production. Thus, it seems likely that the two entities should be mutually exclusive. To verify this, we set SigF and SpoOA∼P off, and insert the additional PN structures shown in Figure 3.12 around all other entities. We then use CLP to check for mutual exclusion between SigF and SpoOA∼P using the following constraints:

$$(SigF + SpoOA{\sim}P) = 2, \quad p_{finish} = 1$$

The output from CLP is shown below, and it correctly confirms that no reachable marking exists which satisfy the constraints, thus proving that SigF and SpoOA∼P are indeed mutually exclusive in our model.

```
solving: 0.01 sec.
4 compatible vectors explored

the marking is unreachable
```

The above model checking technique provides a template which can be used to test other types of relationships between entities in a model. For example, by inspecting the simulation results above (see Figure 3.3.2 and 3.3.2) it appears that SpoOE and AbrB always have the same state. We can attempt to verify this hypothesis by using CLP to check if a contradictory state can be reached, which we specify using the following constraints:

$$(SpoOE + AbrB) = 1, \quad p_{finish} = 1$$

The results of CLP show that a contradictory state is in fact reachable, thus disproving our hypothesis.

```
solving: 0.01 sec.
254 compatible vectors explored
_SEQUENCE:
t89,t92,t94,t96,t97,t99,t101,t103,t106,t108,t7,t19,t24,t35,t42,
t46,t54, t66,t70,t81,t10,t9,t22,t21,t30,t27,t37,t36,t44,t43,t52,
```

```
t49,t60,t56,t69,t67,t79,t73,t87,t83,t8,t20,t24,t35,t42,t46,t54,
t61,t70,t84
path length: 50

the marking is reachable
```

CLP returns a transition firing sequence which leads to a contradictory marking in which SpoOE and AbrB have different states as a counter example, and this can then be automatically simulated in PEP, giving important insight into how this behaviour occurs.

Mutant analysis

We complete our study by investigating the affect of fixing a gene in the model to either be permanently active or inactive. In particular, we apply this technique to a range of entities in the sporulation model to investigate how it affects the sporulation process. The observed results are summarised in Figure 3.3.2 and correspond well with the ex-

Entity	Knock out	Overexpressed
KinA	No sporulation	Normal
SpoOA	No sporulation	Normal
AbrB	No sporulation	No sporulation
SpoOE	Normal	No sporulation
SigH	No sporulation	Normal
SinR	Normal	No sporulation

Figure 3.19: Results of mutant analysis.

perimental results available in the literature [99]. Interestingly, we see that *abrB* is the only gene which has an affect when either knocked out or overexpressed. This fits with our understanding of *abrB* being a transition state regulator gene that lies at the center of three competing feedback loops [99, 197].

3.4 Discussion

3.4.1 Conclusions

BNs have been studied extensively in biological modelling [105, 182, 203], but suffer from a number of shortcomings which can be addressed with PNs. In this chapter, we took as our starting point the work of Steggles *et al.* [194–196], who developed a systematic and flexible PN framework for BNs. This approach builds upon previous work for translating BNs into PNs [45, 165] in a number of important ways: (i) supporting *both* update semantics; (ii) coping with partial models; and (iii) the application of logic minimisation techniques [146] to enable a compact and systematic construction of the final PN model. The contribution of this chapter, however, has been the development and application of much-needed tool support to make this approach practical to the biological community.

GNAPN is a Java tool that was developed to completely automate the PN construction process [194–196]. Key features of GNAPN are its ability to model both the synchronous and asynchronous semantics of BNs within the PN framework, the application of logic minimisation techniques to construct compact PNs, and its ability to handle partial models in a useful way. Altogether, GNAPN provides a comprehensive tool which completely automates the systematic approach of [196], and thus contributes much needed support for PN model construction to the biological community.

We demonstrated our approach in Section 3.3 with two comprehensive case studies on carbon stress response in *E. coli* [167] and sporulation in *B. subtilus* [99, 197]. These studies exemplified the fact that complex behaviours can be manifested even in modest-sized models, thus motivating the formal PN techniques discussed. We automatically constructed PN models of both systems using our support tool GNAPN, and proceeded with some straightforward simulations using PEP. These simulations enabled us to gain initial insights into the fundamental qualitative behaviours of the models (for example, to verify that the switch from sporulation to normal growth was correctly captured for *B. subtilis*). The ability to perform such simulations plays a crucial part in validating and understanding complex models, and also allows hypotheses to be formulated which can then be used to incrementally refine the model. This is therefore a key benefit of PNs, and so motivates our approach.

We also utilised powerful PN analysis techniques, such as *model checking* [51, 109], which provided many deeper and more useful insights. For example, we hypothesised from our simulations on *E. coli* that GyrAB and TopA should be mutually exclusive, since they worked in opposite directions to affect Fis. Such an investigation would prove problematic from manual inspection of the models' behaviour, but model checking enabled such properties to be checked automatically and efficiently. In order to prove that this property held in all cases, we used GNAPN to automatically insert additional PN structures to allow each possible initial marking to be explored. The model checker CLP was then able to confirm that no marking violating mutual exclusion existed, which confirmed our theory. We were therefore able to demonstrate how efficient PN analysis techniques could be used to formally prove properties of the models; something which does not appear to be as straightforward using BNs alone.

Apart from their wide range of analysis techniques and tools, PNs also provide a convenient means of documenting our knowledge of a GRN concisely, by representing both static and dynamic information unambiguously in both a mathematical and visual way. Incomplete knowledge which is troublesome to represent in a BN can be captured usefully with the non–deterministic firing semantics of PN transitions, and later refined when more information is available. This is a clear advantage of our approach as it avoids the need for computationally-expensive learning techniques [125] or stochastic extensions to the underlying BN [182, 183] by allowing for all possible behaviours. Furthermore, PNs cater for modularity in a natural way, which supports the idea that genes often function in independent units [80]. This offers many attractive prospects for constructing and analysing models in a compositional fashion (see [48–50, 121, 126] for studies), which will aid interpretation as well as result in more efficient analysis. Our approach does, however, suffer from an inability to capture certain subtle regulatory interactions [87, 188] due to

its high level of abstraction, and we address this in Chapter 4.

3.4.2 Future work

Although BNs scale to large GRNs, they cannot always represent the qualitative behaviours captured by finer-grained models [77, 142, 188], and so this motivates us to consider extending our approach to cater for *multi-valued networks (MVNs)* [146] in Chapter 4. In an MVN, each entity assumes a range of discrete states and its dynamics are described using multi-valued functions. They therefore provide a means of capturing the possibility for multiple levels of interaction between entities, which are required for modelling more complex regulatory systems [206]. However, their increased modelling power results in a significant increase in possible behaviours, and so a key focus will be in ensuring that the resulting PN model is compact and amenable to analysis tools.

In addition, our framework can cater for both the synchronous and asynchronous semantics of BNs, but we focused on the synchronous semantics, as it appears to be widely used in the biological community [31]. Further work is now required for the analysis of practical GRNs under the asynchronous semantics. In particular, techniques are required for coping with the increase in state space complexity resulting from the inherent non–determinism present in asynchronous BNs. We will develop an approach for addressing this concern in Chapter 6.

Another consideration is investigating more expressive model checking techniques for PNs. In particular, we intend to strengthen our analysis by investigating the application of temporal logics such as *computational tree logic* [41]. Finally, we are interested in developing techniques for refining qualitative models (such as those based on BNs) into quantitative ones when more data is available; in particular, the development of systematic and automated translation techniques is crucial in ensuring that these are practical (see [161] for an initial study).

3.4.3 Sources

The theoretical PN modelling approach was proposed by L.J.Steggles and is presented in [196]. The development of the tool GNAPN and its application to the two case studies was primarily the authors own work. The application of the approach to the case study on *E. coli* appears in the proceedings of the International Conference on Computational Methods in Systems Biology in Trento, October 2006 [195]. Furthermore, the application to the case study on *B. subtilis* was published in Bioinformatics journal in November 2006 [194]. GNAPN and the model files used in this chapter can be freely downloaded for academic use from *bioinf.ncl.ac.uk/gnapn*.

Chapter 4

A Generalised Petri Net Framework for Genetic Regulatory Networks

BNs have enjoyed much success in biological modelling, but are limited by the fact that they cannot discriminate different levels of interaction [77, 142, 188]. This motivates the use of *multi-valued networks (MVNs)* [77, 141, 146, 205, 206], which generalise BNs by allowing for ranges of discrete values. As such, MVNs provide a compromise between the simplicity of BNs and more expressive differential equations, and have proven to be a promising qualitative approach for modelling GRNs [206, 208]. However, apart from some notable extensions (for example, see [27]), they too appear to require strengthened techniques for model construction and analysis.

To address this, we generalise the PN framework from Chapter 3 to cope with MVNs. The first key contribution is with the application of *multi-valued logic minimisation techniques* [146], which enable compact behavioural specifications of MVNs to be obtained. We then focus on developing efficient PN models to capture this compact but more complex logic. Specifically, we propose two complementary approaches: the first considers a novel technique using standard safe PN structures to minimise the number of transitions; and the second employs *high-level Petri nets (HLPNs)* [28], which provide an expressive modelling environment for describing complex logic whilst remaining amenable to standard PN techniques and tools. We investigate the application of HLPNs to develop a visually-compact generalised framework for MVNs, and show that, in general, it also offers superior analysis efficiency.

We implement both PN approaches as extensions to our support tool GNAPN, and demonstrate its application by revisiting the carbon stress response network in *E. coli* [167]. We automatically construct an HLPN model, and then apply a range of PN analysis techniques and tools to gain confidence in its correctness. In particular, this study identifies a number of behavioural differences between the corresponding BN model from Chapter 3, thus raising some interesting questions concerning their formal relationship.

A number of similar works developed around the same time as ours can be found in the literature. Chaouiya *et al.* [46] proposed a set of rewriting rules for translating MVNs into PN (see [187] for an application of this approach to tryptophan biosynthesis in *E. coli*). However, they do not make provisions for the synchronous network semantics, cater

for partial models or apply logic minimisation systematically. Furthermore, they utilise non-safe PNs which are not always amenable to the range of PN analysis techniques and tools that safe ones are. Comet *et al.* [53] proposed a compact HLPN framework for MVNs, but focused more on the derivation of consistent asynchronous PNs using model checking.

The rest of this chapter is structured as follows. In Section 4.1, we introduce MVNs. Then, in Section 4.2, we give a brief introduction to the theory of HLPNs. Section 4.3 describes the development of our two PN frameworks for modelling MVNs, and shows how logic minimisation techniques [146] can be utilised to achieve this compactly. We also outline an approach for coping with partial models in a useful way. In Section 4.4, we discuss these extensions to our support tool GNAPN, and then compare the safe and HLPN frameworks developed for analysis efficiency. In Section 4.5, we apply our framework with a detailed case study which revisits the carbon stress response network in *E. coli*. Section 4.6 then considers an informal comparison between our MVN and BN models, and highlights a number of interesting behavioural differences. Finally, Section 4.7 presents some concluding remarks.

4.1 Multi-valued networks

In this section, we introduce MVNs [46, 206, 208], a generalisation of the BN model which have also been extensively studied in circuit design (for example, see [146, 169]) and successfully applied to modelling GRNs [46, 208]. Note another variation of this approach is the so-called *qualitative networks* discussed in a recent paper [172].

An MVN model $\mathcal{MV}$ consists of a set of k logically linked entities $G = \{g_1, \ldots, g_k\}$ which regulate each other in a positive or negative way (note as with BNs, and in a slight abuse of notation, we use g_i to denote both the name and current state of an entity, and the shorthand notation $g_i \in \mathcal{MV}$ to denote the fact that $g_i \in G$ is an entity of model $\mathcal{MV}$). In particular, an MVN generalises upon the BN model by allowing each entity $g_i \in \mathcal{MV}$ to be in one of two or more states $g_i \in Y(g_i)$, where $Y(g_i) = \{0, \ldots, m_i\}$, for some $m_i \geq 1$. Note therefore that an MVN model $\mathcal{MV}$ is a BN if, for each entity $g_i \in \mathcal{MV}$, we have that $Y(g_i) = \{0, 1\}$.

Biologically, the rationale behind having two or more states is that an entity may interact at different points in the system, each at distinct threshold levels. The idea is therefore that an entity g_i involved in m interactions will have as many as m distinct threshold levels that split its real concentration range into $m + 1$ discrete regions [206]:

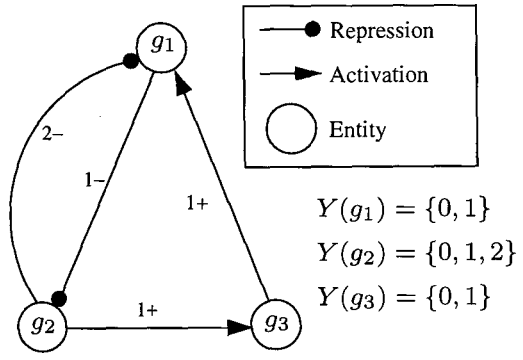
$$\theta_i^1 < \theta_i^2 < \cdots < \theta_i^m.$$

These regions can then be naturally interpreted in a logical fashion by assigning a logical value to each depending on their relation with the thresholds:

$$g_i = \begin{cases} 0 & \text{if } g_i < \theta_i^1 \\ 1 & \text{if } \theta_i^1 < g_i < \theta_i^2 \\ 2 & \text{if } \theta_i^2 < g_i < \theta_i^3 \\ \vdots & \end{cases}$$

Each entity g_i has an associated neighbourhood $N(g_i) \subseteq G$ of entities that directly affect it. Note we allow $g_i \in N(g_i)$ if g_i auto-regulates itself. The affect of one entity on another can therefore either be that of activation (i.e. an increase in state) or inhibition (i.e. a decrease in state). Furthermore, interactions between one entity and another only become functional if the state of the source entity is sufficiently above some threshold level (this threshold level is always at least one), and the result is that the state of the target will tend (or remain, depending on its current state) towards a new state. Note MVNs therefore allow one to discriminate between the strengths of different interactions, something which BNs are unable to capture.

As an example, consider Figure 4.1(a) which shows the structure of a small MVN consisting of three entities g_1, g_2 and g_3, such that $Y(g_1) = \{0, 1\}$, $Y(g_2) = \{0, 1, 2\}$ and $Y(g_3) = \{0, 1\}$, respectively. The idea is that entity g_1 acts as an inhibitor to g_2, whereas g_2 inhibits g_1 when $g_2 = 2$ and activates g_3 when $g_2 > 1$. In addition, g_3 acts as an activator to g_1; the corresponding functional ranges of the interactions are represented by the edge labels. Furthermore, the inhibitory affect of g_2 on g_1 is "overpowered" by g_3 when both interactions are functional, i.e. the state of g_1 will tend to one.



(a)

g_2	g_3	$[g_1]$
0	0	1
0	1	1
1	0	1
1	1	1
2	0	0
2	1	1

g_1	g_2	$[g_2]$
0	0	1
0	1	2
0	2	2
1	0	0
1	1	0
1	2	1

g_2	$[g_3]$
0	0
1	1
2	1

(b)

Figure 4.1: (a) An MVN with three entities, and (b) the state transition tables representing the corresponding next-state functions F.

As with BNs, the behaviour of each g_i can be precisely defined by a logical next-state function f_{g_i} which calculates the next-state of g_i given the current states of the entities in its neighbourhood. By applying the next-state function f_{g_i} to each neighbourhood state of g_i, one derives a state transition table which completely captures this behaviour. For our example MVN, this is shown in Figure 4.1(b). The behaviour captured by the state transition tables can often be simplified using *logic minimisation techniques* [146] which will be discussed in Section 4.3.

We can now define an MVN more formally as follows.

Definition 4.1 (Multi-valued network). *A multi-valued network $\mathcal{MV}$ with k entities is a four-tuple $\mathcal{MV} = (G, Y, N, F)$ where: $G = \{g_1, \ldots, g_k\}$ is a finite set of entities;*

$Y = (Y(g_1), \ldots, Y(g_k))$ is a finite list of sets, where each $Y(g_i) = \{0, \ldots, m_i\}$, for some $m_i \geq 1$, is the state space for entity g_i; $N = (N(g_1), \ldots, N(g_k))$ is a finite list of sets, such that $N(g_i) \subseteq G$ is the neighbourhood of g_i; and $F = (f_{g_1}, \ldots, f_{g_k})$ is a finite list of next-state multi-valued functions, such that if $N(g_i) = \{g_{i_1}, \ldots, g_{i_n}\}$ then $f_{g_i} : Y(g_{i_1}) \times \cdots \times Y(g_{i_n}) \to Y(g_i)$ defines the next state of g_i.

The collective state of each $g_i \in \mathcal{MV}$ is called a *global state*, and the same rules for synchronous and asynchronous updating apply here as they do for BNs [86]. The state space of a model $\mathcal{MV}$, denoted $S_{\mathcal{MV}}$, is therefore the set of all possible global states $S_{\mathcal{MV}} = Y(g_1) \times \cdots \times Y(g_k)$. An important focus of this work will therefore be in ensuring a compact PN representation of this state space. Finally, we remark that although entity states are usually considered to tend towards some focal value, the state transition tables place no restriction on this. Thus, multi-state jumps can either be left in the model if they can be biologically justified or guarded against; both of which represent important modelling decisions which should be documented.

4.2 High-level Petri nets

In this section, we introduce *high-level Petri nets (HLPNs)* [28] at a level suitable for the chapter. For a more comprehensive introduction to HLPNs, we refer the reader to [28]. In the sequel, we assume the reader is familiar with PNs (see Section 2.3).

An HLPN [28] is a PN in which tokens can be distinguished, for instance, by Boolean values, integers an so on. The idea is that each place is associated with a set or type specifying the tokens it can hold. Note the tokens of an HLPN include standard PN tokens for consistency. For example, in Figure 4.2(a), place p_2 has token type $\{0, \ldots, 5\}$ which restricts tokens for that place to integers between 0 and 5, and we observe that p_2 currently contains tokens 2 and 3. Note that multiple copies of the same token are allowed on places, i.e. places contain multi-sets of tokens. In a similar way to PNs, the marking an HLPN maps each place to the multi-set of tokens it contains. For example, the current marking M of Figure 4.2(a) is defined by $M(p_1) = \{1\}$, $M(p_2) = \{2, 3\}$ and $M(p_3) = \{\}$.

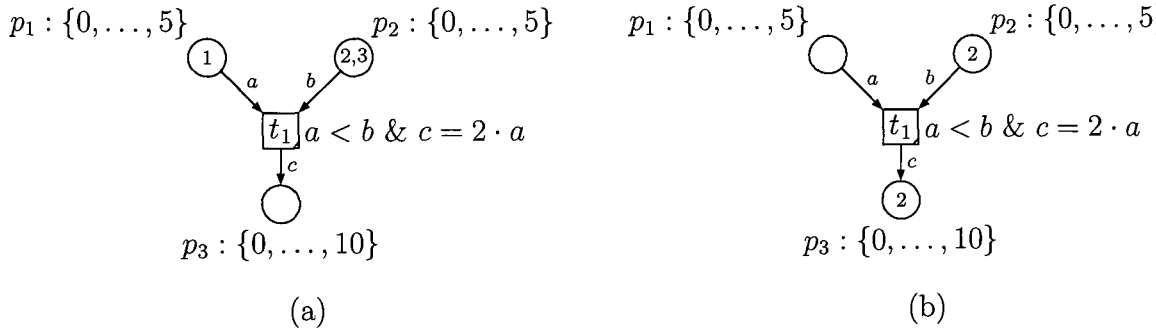


Figure 4.2: (a) A simple example of an HLPN, and (b) new marking after firing t_1 from enabling binding $\{a \mapsto 1, b \mapsto 3, c \mapsto 2\}$.

Each arc is associated with a variable which *binds* to the tokens of the place it comes from (called an *input variable*) or goes to (called an *output variable*). Furthermore, each transition is associated with a Boolean expression called a *guard* which evaluates to true or false for a given binding. A transition t therefore becomes enabled with a binding if: (i) the token value assigned to each input variable resides on the associated input place; and (ii) the transition's guard evaluates to true with the binding. An enabled transition may then fire by removing tokens from each of its input places and adding a new token to each output place as specified by the enabling binding. Note, as with PNs, if more than one transition is enabled then a transition is chosen non–deterministically to fire.

In Figure 4.2(a), transition t_1 is enabled by the binding $\{a \mapsto 1, b \mapsto 3, c \mapsto 2\}$. Note that binding $\{a \mapsto 1, b \mapsto 2, c \mapsto 2\}$ would also enable t_1, but we assume that the former is chosen non–deterministically. Transition t_1 can therefore fire, which results in token 1 and 3 being removed from places p_1 and p_2, respectively, and a new token 2 being added to place p_3. The new marking M' is therefore $M'(p_1) = \{\}$, $M'(p_2) = \{2\}$ and $M'(p_3) = \{2\}$ which we show in Figure 4.2(b). Note the normal definition of firing sequence and reachability for PNs also applies to HLPNs (see Section 2.3.3).

Overall, HLPNs provide an expressive and natural mechanism for modelling MVNs, by separating behaviour from representation, and by using high-level token types and transition guards. Furthermore, a key advantage of HLPNs is that they are amenable to the wide range of PN analysis techniques and tools [1] via the process of *unfolding* [109, 111]. Normally, the idea is that a HLPN is first expanded into a standard PN and then unfolded to obtain the canonical prefix over which analysis can be performed. However, this often proves a costly operation, since the intermediate PN can be exponentially larger than its corresponding prefix [109]. This issue was recently addressed by Khomenko *et al.* [109, 111] who developed an unfolding approach for generating the canonical prefix directly from the HLPN. This approach has been implemented in the parallel PN unfolder PUNF which we will use later in Section 4.5 to unfold our HLPN model for *E. coli*.

4.3 From multi-valued networks to Petri nets

In this section, we incrementally develop a generalised PN modelling framework for MVNs. Firstly, we show how the behaviour of an MVN can be compactly specified as next-state equations using *logic minimisation techniques* [146, 169]. We then propose a novel safe PN approach for modelling this behaviour compactly which aims to minimise the number of transitions. Finally, we explore the use of HLPNs for compactly modelling MVNs, and after a comparison between the two, discuss why, in general, the HLPN one is superior.

4.3.1 Obtaining compact next-state equations

Let $\mathcal{MV} = (G, Y, N, F)$ be an MVN as defined by Definition 4.1, whose behaviour F is specified by a series of state transition tables (e.g. see Figure 4.1(b)). This behaviour can be equivalently represented using Boolean terms called *literals* that formalise when an entity g_i is in one of the states in $Y(g_i)$. Literals have the form $g_i s$, where $s \subseteq Y(g_i)$, and

are defined to evaluate to *true* when $g_i \in s$, and to $false$ otherwise. For example, the literal $g_1\{0,1\}$ will evaluate to *true* only when $g_1 = 0$ or $g_1 = 1$, and to $false$ otherwise. As with Boolean functions (see Section 3.1), we refer to the conjunction of such literals as *terms*.

For example, from the table for g_1 in Figure 4.1(b), we can see that $[g_1] = 0$ when $g_1 = 0$, $g_2 = 2$ and $g_3 = 0$; in other words, when term $g_1\{0\}g_2\{2\}g_3\{0\}$ evaluates to *true*. The complete next-state equation specifying the conditions when an entity g_i updates to some state $j \in Y(g_i)$, written $[g_i\{j\}]$, is therefore the disjunction of all corresponding terms. Continuing with our example, we derive the following next-state equations which completely specify the behaviour of g_1:

$$\begin{aligned}
[g_1\{0\}] &= g_1\{0\}g_2\{2\}g_3\{0\} + g_1\{1\}g_2\{2\}g_3\{0\} \\
[g_1\{1\}] &= g_1\{0\}g_2\{0\}g_3\{0\} + g_1\{0\}g_2\{0\}g_3\{1\} + g_1\{0\}g_2\{1\}g_3\{0\} + \\
&\quad g_1\{0\}g_2\{1\}g_3\{1\} + g_1\{0\}g_2\{2\}g_3\{1\} + g_1\{1\}g_2\{0\}g_3\{0\} + \\
&\quad g_1\{1\}g_2\{0\}g_3\{1\} + g_1\{1\}g_2\{1\}g_3\{0\} + g_1\{1\}g_2\{1\}g_3\{1\} + g_1\{1\}g_2\{2\}g_3\{1\}
\end{aligned}$$

These equations can often contain redundant logic, and so can be simplified using *logic minimisation techniques* [146, 169]. Since our approach translates these equations into PN structures, it is important that they are compact. Logic minimisation for multi-valued functions works in a similar way to Boolean minimisation (see Section 3.1) in that we merge terms which differ by only one literal. For example, consider the next-state equation for $[g_1\{0\}]$:

$$[g_1\{0\}] = g_1\{0\}g_2\{2\}g_3\{0\} + g_1\{1\}g_2\{2\}g_3\{0\}$$

We note that these two terms can be merged, since they differ only by the value of g_1, giving us:

$$[g_1\{0\}] = g_1\{0,1\}g_2\{2\}g_3\{0\}$$

Now, we observe that the literal $g_1\{0,1\}$ will *always* evaluate to true, and so it is redundant with respect to the result of this expression. We therefore remove it, leaving:

$$[g_1\{0\}] = g_2\{2\}g_3\{0\}$$

which correctly captures the only logical condition in which g_1 reaches state 0.

We also observe that similar simplifications can be made to the next-state equation for $[g_1\{1\}]$. Starting with the first term $g_1\{0\}g_2\{0\}g_3\{0\}$, we compare it against all other terms, and notice that it can be merged with $g_1\{0\}g_2\{0\}g_3\{1\}$, $g_1\{0\}g_2\{1\}g_3\{0\}$ and $g_1\{1\}g_2\{0\}g_3\{0\}$, giving us the additional three merged terms (shown in bold) below:

$$\begin{aligned}
[g_1\{1\}] &= g_1\{0\}g_2\{0\}g_3\{1\} + g_1\{0\}g_2\{1\}g_3\{0\} + g_1\{0\}g_2\{1\}g_3\{1\} + g_1\{0\}g_2\{2\}g_3\{1\} + \\
&\quad g_1\{1\}g_2\{0\}g_3\{0\} + g_1\{1\}g_2\{0\}g_3\{1\} + g_1\{1\}g_2\{1\}g_3\{0\} + g_1\{1\}g_2\{1\}g_3\{1\} + \\
&\quad g_1\{1\}g_2\{2\}g_3\{1\} + \boldsymbol{g_1\{0\}g_2\{0\} + g_1\{0\}g_2\{0,1\}g_3\{0\} + g_2\{0\}g_3\{0\}}
\end{aligned}$$

Note we have removed the first term as we know its logic has been covered. We then perform the same process with the second term; removing it if it merges with another

term and keeping it otherwise, and adding any new terms created. Performing this process exhaustively until no more minimisation can be performed gives us the following simpler next-state equation:

$$[g_1\{1\}] = g_3\{1\} + g_2\{0,1\},$$

which captures the original logic specified using ten terms in only two.

These techniques are automated by tools such as MVSIS [2], and can be applied to each state transition table to obtain compact next-state equations representing F. The complete minimised next-state equations for our example MVN are given below:

$$\begin{aligned}
[g_1\{0\}] &= g_2\{2\}g_3\{0\} \\
[g_1\{1\}] &= g_3\{1\} + g_2\{0,1\} \\
[g_2\{0\}] &= g_1\{1\}g_2\{0,1\} \\
[g_2\{1\}] &= g_1\{0\}g_2\{0\} + g_1\{1\}g_2\{2\} \\
[g_2\{2\}] &= g_1\{0\}g_2\{1,2\} \\
[g_3\{0\}] &= g_2\{0\} \\
[g_3\{1\}] &= g_2\{0,1\}
\end{aligned}$$

We consider developing a PN modelling approach to enable the logic specified by these compact equations to be analysed using the wide range of available PN techniques and tools. For this, we will start by proposing a safe PN framework, and then explore the use of HLPNs.

4.3.2 A safe Petri net approach

In this section, we develop a safe PN modelling framework for the compact next-state equations derived previously. We propose a novel use of PN structures to model each logical term using a *single transition*. We then address the exponential blowup in possible state updates that occurs under both the synchronous and asynchronous semantics, and show how the two-phase commit protocol discussed in Section 3.1.2 can be adapted to cope with this compactly.

Compactly modelling logical terms

Let $g_i \in \mathcal{MV}$ be an entity with state space size $|Y(g_i)| = n$. These states can be modelled using n places $p_i\{j\}$, where $j \in Y(g_i)$. For convenience, we will refer to such places as the *state places* for g_i, and will denote these SP_{g_i}. In order to model the state of g_i using SP_{g_i}, we consider adopting a slightly less obvious approach which we will justify shortly. The idea is to capture all the states that g_i is *not* in by using a token on each corresponding place in SP_{g_i}. The current state of g_i is thus represented by the only place in SP_{g_i} that *does not* contain a token.

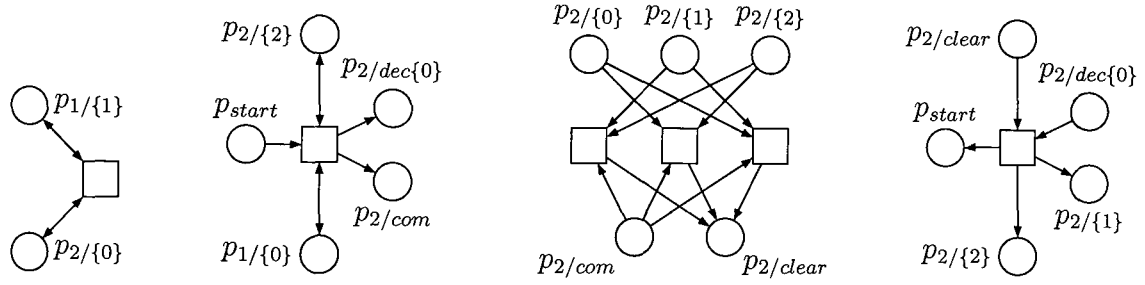
Whilst this approach may appear excessive, it turns out that the negative information present in logical terms can prove useful in modelling them compactly. For example, consider the minimised next-state equation:

$$[g_2\{2\}] = g_1\{0\}g_2\{1,2\},$$

which we derived in the previous section from the MVN in Figure 4.1. This says that entity g_2 updates to state 2 when $g_1 = 0$ and $g_2 = 1$ or $g_2 = 2$. Normally, one would model these two conditions as two separate transitions: one for when $g_1 = 0$ and $g_2 = 1$, and one for when $g_1 = 0$ and $g_2 = 2$. However, by utilising the negative information present in this term, we can equivalently say that g_2 updates to state 2 when $g_1 \neq 1$ *and* $g_2 \neq 0$, and thus we only require the *single* transition shown in Figure 4.3(a) to model this. We can now specify this approach more formally.

Definition 4.2 (Transition wiring using negative information). *Given a term* $\mathcal{T} = g_{l_1}s_1 g_{l_2}s_2 \ldots g_{l_n}s_n$, *we define a transition* t *such that for each* $v \in \{1, \ldots, n\}$, *we have read arcs connecting* $\overline{p_u}$ *and* t *for each* $u \in (Y(g_{l_v}) - s_v)$.

It is clear that for any term $\mathcal{T} = g_{l_1}s_1 g_{l_2}s_2 \ldots g_{l_n}s_n$, only a single transition t will be required to model its logic using the approach based on negative information, and that t will be enabled to fire iff $\mathcal{T}$ evaluates to *true*. Without utilising the negative information, however, we would require a transition to capture each possibility in the cartesian product $s_1 \times s_2 \times \cdots \times s_n$, which is significantly larger in most cases (except when each s_i is a singleton set, for $i = 1, \ldots, n$).



(a) (b) (c) (d)

Figure 4.3: (a) Transition modelling next-state equation $[g_2\{2\}] = g_1\{0\}g_2\{1,2\}$ from running example, and PN fragments for two-phase protocol under asynchronous semantics: (b) entity g_2 recording update to state 0; (c) flushing tokens from SP_{g_2}; and (d) SP_{g_2} repopulated with tokens to reflect new state 0.

With a compact means of representing each term in the next-state equations, we go on to discuss how both the synchronous and asynchronous semantics of MVNs can be implemented compactly using safe PN structures. In particular, we build upon the two-phase commit protocol from Section 3.1.2, and show how *both* update semantics of MVNs can be catered for in this way.

Asynchronous update semantics

As the state space of each entity increases, there is a need to handle the combinatorial blowup in possible state updates. To address this, we revisit the two-phase commit

protocol developed in Section 3.1.2 and adapt it to cater for the asynchronous update semantics. In particular, we combat the number of possible state updates for an entity g_i by first flushing SP_{g_i} of tokens and then repopulating them to reflect the new state. This strategy can be shown to reduce the number of transitions required to commit the recorded decision for g_i from j^2 to $2j$, where $|Y(g_i)| = j$. We now formally introduce this adapted two-phase protocol.

Definition 4.3 (Asynchronous multi-valued network to Petri net). *Assume we have an MVN with n entities $g_1, \ldots, g_n$, denoted $\mathcal{MV}$. A corresponding PN modelling the asynchronous update semantics of $\mathcal{MV}$ can be constructed as follows:*

- *For each entity $g_i \in \mathcal{MV}$, we add places $p_{i/com}$ (to indicate that an update decision has been made) and $p_{i/clear}$ (to indicate when a decision should be committed). Furthermore, for each state $j \in Y(g_i)$ of g_i, we add places $p_{i/\{j\}}$ (the state places SP_{g_i} representing the actual state of g_i) and $p_{i/dec\{j\}}$ (to record the update decision for g_i). Finally, we add a single place p_{start} (to ensure that only one entity can perform both phases of the protocol). Note that in the initial marking, SP_{g_i} will be marked accordingly, and p_{start} will contain a token. All other places must be unmarked.*

- ***Phase One:*** *Consider each minimised next-state equation $[g_i\{j\}] = T_1 + \cdots + T_m$ which defines when g_i updates to state j. For each product term T_a, for $a = 1, \ldots, m$, add a transition $t_{i/a}$ as specified in Definition 4.2. In addition, place p_{start} is an input place and $p_{i/dec\{j\}}$ and $p_{i/com}$ are output places of $t_{i/a}$.*

- ***Phase Two:***

  - *For each $g_i \in \mathcal{MV}$ and for each state $j \in Y(g_i)$, we add a transition whose input places are $p_{i/com}$ and $p_{i/\{q\}}$, for each $q \in Y(g_i) - \{j\}$, and whose output place is $p_{i/clear}$. Note only one transition will be enabled depending on the current state of g_i.*
  - *For each $g_i \in \mathcal{MV}$ and for each state $j \in Y(g_i)$, we add a transition which has input places $p_{i/dec\{j\}}$ and $p_{i/clear}$, and output places p_{start} and $p_{i/\{q\}}$, for each $q \in Y(g_i) - \{j\}$. Note firing this transition commits the recorded state to $Y(g_i)$ and remarks p_{start} so that some $g_i \in \mathcal{MV}$ can update its state.*

As an example, Figure 4.3(b) shows the transition in phase one modelling the term $g_1\{1\}g_2\{0,1\}$ in the next-state equation for $[g_2\{0\}]$. When the transition fires, it marks $p_{2/dec\{0\}}$ to record the fact that g_2 will update to state 0, and $p_{2/com}$ to indicate that an update decision has been made. In order to then commit this recorded decision and avoid the combinatorial blowup in possible cases, we first flush SP_{g_i} as shown in Figure 4.3(c). Depending on the current state of g_2, only one of these transitions will be enabled, which when fired, will mark $p_{i/clear}$ to indicate that the new state can be commited. Finally, Figure 4.3(d) shows the transition for committing the new state 0 to SP_{g_2}. When fired, this transition remarks p_{start} so that the protocol can begin again for a single entity in the model. With minor changes, we can adapt this protocol to also model the synchronous semantics of MVNs.

It can be observed from Definition 4.3 that if we have an MVN with n entities, where each entity has j states and a total of m logical terms in its minimised next-state equations, then the corresponding PN will contain $2jn+2n+1$ places and $mn+2jn$ transitions. Without logic minimisation, however, the PN would contain $(j^n)n$ transitions to cater for each possible state update that could occur.

Synchronous update semantics

We now discuss how we adapt the protocol for the synchronous semantics. The key difference is that instead of only allowing one entity $g_i \in \mathcal{MV}$ to perform both phases of the protocol, we now allow all entities to perform phase one concurrently, followed by phase two concurrently.

Definition 4.4 (Synchronous multi-valued network to Petri net). *Assume we have an MVN with n entities $g_1, \ldots, g_n$, denoted $\mathcal{MV}$. A corresponding PN modelling the synchronous update semantics of $\mathcal{MV}$ can be constructed as follows:*

- *For each entity $g_i \in \mathcal{MV}$, we add places $p_{i/start}$ (to indicate that g_i can make an update decision), $p_{i/syn}$ (to indicate that an update decision for g_i has been made), $p_{i/com}$ (to indicate that g_i can begin phase two), $p_{i/clear}$ (to indicate when a decision for g_i should be committed) and $p_{i/done}$ (to indicate that a decision for g_i has been committed). Furthermore, for each state $j \in Y(g_i)$ of g_i, we add places $p_{i/\{j\}}$ (the state places SP_{g_i} representing the actual state of g_i) and $p_{i/dec\{j\}}$ (to record the update decision for g_i). Finally, we add a single place p_{finish} for model checking convenience, indicating when a complete pass of the protocol has been made for all entities. Note that in the initial marking, SP_{g_i} will be marked accordingly, and $p_{i/start}$ will contain a token, for $i = 1, \ldots, n$. All other places must be unmarked.*

- ***Phase One:***

  - *Consider each minimised next-state equation $[g_i\{j\}] = T_1 + \cdots + T_m$ which defines when g_i updates to state j. For each product term T_a, for $a = 1, \ldots, m$, add a transition $t_{i/a}$ as specified in Definition 4.2. In addition, place $p_{i/start}$ is an input place and $p_{i/dec\{j\}}$ and $p_{i/syn}$ are output places of $t_{i/a}$.*
  - *We add a transition to model the fact that we want to wait for all entities to concurrently record a decision before they commit. The transition has input places $p_{i/syn}$ and output places $p_{i/com}$, for $i = 1, \ldots, n$.*

- ***Phase Two:***

  - *For each $g_i \in \mathcal{MV}$ and for each state $j \in Y(g_i)$, we add a transition whose input places are $p_{i/com}$ and $p_{i/\{q\}}$, for each $q \in Y(g_i) - \{j\}$, and whose output place is $p_{i/clear}$. Note only one transition will be enabled depending on the current state of g_i.*
  - *For each $g_i \in \mathcal{MV}$ and for each state $j \in Y(g_i)$, we add a transition which has input places $p_{i/dec\{j\}}$ and $p_{i/clear}$, and output places $p_{i/done}$ and $p_{i/\{q\}}$, for each $q \in Y(g_i) - \{j\}$.*

– *We add a transition which has input places $p_{i/done}$, for $i = 1, \ldots, n$, and output place p_{finish}. An additional transition is then used which has input place p_{finish} and output places $p_{i/start}$, for $i = 1, \ldots, n$, which resets the protocol.*

As an example, Figure 4.4(a) shows the transition in phase one modelling the term $g_1\{1\}g_2\{0,1\}$ in the next-state equation for $[g_2\{0\}]$. When the transition fires, it marks $p_{2/dec\{0\}}$ to record the fact that g_2 will update to state 0, and $p_{2/syn}$ to indicate that an update decision has been made. To ensure that all entities can record their update decision concurrently before phase two begins, we require the transition shown in Figure 4.4(b). Once this transition fires, each entity is then allowed to commit its recorded decision concurrently. First, the state places SP_{g_2} are flushed of tokens as previously shown in Figure 4.3(c). Then, the new state 0 is committed to SP_{g_2} using the transition shown in Figure 4.4(c). When fired, this transition also marks $p_{2/done}$, and once the remaining entities have also committed their new state, the transition shown in Figure 4.4(d) will be enabled which subsequently resets the protocol.

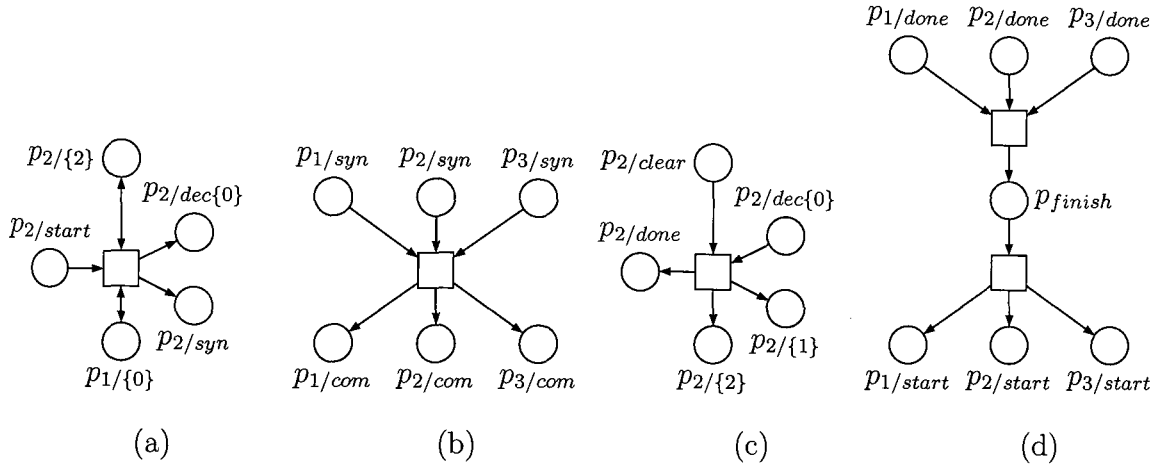


Figure 4.4: PN fragments for two-phase protocol under synchronous semantics: (a) entity g_2 recording update to state 0; (b) control structure to wait for all entities to complete phase one before beginning phase two; (c) state places for g_2 repopulated to reflect new state 0 by committing recorded state; and (d) control structure to reset protocol.

It can be observed from Definition 4.4 that if we have an MVN with n entities, where each entity has j states and a total of m logical terms in its minimised next-state equations, then the corresponding PN will contain $2jn + 5n + 1$ places and $mn + 2jn + 3$ transitions. Without logic minimisation, however, the PN would contain j^n transitions to cater for each possible state update that could occur.

This section has discussed the development of a novel safe PN framework for modelling MVNs under both the synchronous and asynchronous semantics. Although a number of additional control structures are used, these can be justified as they avoid the exponential blow-up in transitions required to model the possible combinations of state updates. Despite this, such a framework can become verbose for large MVNs. In the next section,

we investigate the application of HLPNs to modelling MVNs. HLPN provide a more natural and expressive means of modelling complex systems, often resulting in a simpler visual representation.

4.3.3 A high-level Petri net approach

This section proposes an HLPN modelling approach for MVNs which enjoys a compact visual representation whilst remaining amenable to the wide range of PN analysis tools and techniques.

We model each $g_i \in \mathcal{MV}$ using a single place p_i, whose token type (i.e. the tokens it can hold) corresponds to the integer values $Y(g_i)$. A place p_i communicates with a transition t using two arcs: one arc from p_i to t labelled with the input variable $c(p_i)$; and one arc from t to p_i labelled with the output variable $n(p_i)$. Each input variable $c(p_i)$ binds to the tokens on the corresponding input place p_i, and is used to specify a Boolean expression in HLPN syntax [81] describing the enabling conditions of t. Similarly, each output variable $n(p_i)$ binds to the tokens going to the corresponding output place p_i and is used to specify the result of firing t. The key to our approach is to keep the structure of the HLPN simple and encode the minimised next-state equations describing the behaviour as transition guards. This is achieved using the *transition guard translation (TGT)* operator δ which we now define.

Definition 4.5 (The TGT operator). *Let δ denote the TGT operator and $g_i s$ be a literal, where $s = \{x_1, \ldots, x_n\}$ for $n \geq 1$ such that $\{x_1, \ldots, x_n\} \subseteq Y(g_i)$. We apply δ to obtain the HLPN syntax [81] as follows:*

$$\delta(g_i s) \stackrel{def}{=} (c(p_i) = x_1 \mid \ldots \mid c(p_i) = x_n),$$

where $\mid$ represents disjunction. We can then apply δ to a term $g_{i_1} s_p \ldots g_{i_n} s_q$ naturally as:

$$\delta(g_{i_1} s_p \ \ldots \ g_{i_n} s_q) \stackrel{def}{=} (\delta(g_{i_1} s_p) \ \& \ \ldots \ \& \ \delta(g_{i_n} s_q)),$$

where $\&$ represents conjunction. We then lift δ to the next-state equation:

$$[g_i\{j\}] = T_1 + \cdots + T_m,$$

for some $j \in Y(g_i)$, where each T_a, for $a = 1, \ldots, m$, is a term:

$$\delta([g_i\{j\}] = \mathcal{T}_1 + \cdots + \mathcal{T}_m) \stackrel{def}{=} (n(p_i) = j \ \& \ (\delta(\mathcal{T}_1) \mid \ldots \mid \delta(\mathcal{T}_m))),$$

where $n(p_i) = j$ denotes the result of firing the transition using the output variable $n(p_i)$. Finally, to obtain the complete behaviour of some entity $g_i \in \mathcal{MV}$, such that $Y(g_i) = \{0, \ldots, r\}$, for $r \geq 1$, we apply δ to each next-state equation $[g_i\{j\}] = \mathcal{E}_j$, for $j \in Y(g_i)$:

$$\delta([g_i\{0\}] = \mathcal{E}_0, \ldots, [g_i\{r\}] = \mathcal{E}_r) \stackrel{def}{=} \delta([g_i\{0\}] = \mathcal{E}_0) \mid \ldots \mid \delta([g_i\{r\}] = \mathcal{E}_r),$$

where $\mathcal{E}_j$ denotes the terms of the equation.

For example, we can apply δ to the next-state equation $[g_1\{0\}] = g_2\{2\}g_3\{0\}$ from our running example as follows:

$$\delta\big([g_1\{0\}] = g_2\{2\}g_3\{0\}\big) \stackrel{def}{=} (n(p_1) = 0 \;\&\; (c(p_2) = 2) \;\&\; (c(p_3) = 0)).$$

As such, we can obtain the complete behaviour of g_1 in HLPN syntax by applying δ to its full set of minimised next-state equations:

$$\begin{aligned} [g_1\{0\}] &= g_2\{2\}g_3\{0\} \\ [g_1\{1\}] &= g_3\{1\} + g_2\{0,1\} \end{aligned}$$

resulting in the following:

$$\begin{aligned} \delta\big([g_1\{0\}] = g_2\{2\}g_3\{0\}, [g_1\{1\}] = g_3\{1\} + g_2\{0,1\}\big) \stackrel{def}{=} \\ (n(p_1) = 0 \;\&\; (c(p_2) = 2) \;\&\; (c(p_3) = 0)) \mid \\ (n(g_1) = 1 \;\&\; (c(p_3) = 1 \mid (c(p_2) = 0 \mid c(p_2) = 1))) \end{aligned}$$

Having defined δ for translating the next-state equations into HLPN syntax, it is then straightforward to implement the synchronous and asynchronous semantics of MVNs. The basic idea for the synchronous semantics is to capture the behaviour of all entities using a single transition. Thus, we apply δ to all the next-state equations, and then use their conjunction to form a corresponding transition guard. More specifically, for an MVN with k entities, we require a single transition t, whose guard encodes the complete behaviour of each entity $g_i \in \mathcal{MV}$ for $Y(g_i) = \{0, \ldots, n_i\}$, given by:

$$\delta\big([g_1\{0\}] = \mathcal{E}_0, \ldots, [g_1\{n_1\}] = \mathcal{E}_{n_1}\big) \;\&\ldots\&\; \delta\big([g_k\{0\}] = \mathcal{E}_0, \ldots, [g_k\{n_k\}] = \mathcal{E}_{n_k}\big).$$

On the other hand, the asynchronous semantics are modelled using a single transition for *each* entity. More specifically, for an MVN with k entities, we require k transitions. The complete behaviour of each entity $g_i \in \mathcal{MV}$, where $Y(g_i) = \{0, \ldots, n_i\}$, is then encoded in its corresponding transition as:

$$\delta\big([g_i\{0\}] = \mathcal{E}_0, \ldots, [g_i\{n_i\}] = \mathcal{E}_{n_i}\big).$$

For example, the HLPN modelling the asynchronous semantics of the MVN from our running examply is shown in Figure 4.5. Note we omit transition guards for brevity.

The result is a flexible generalised framework for modelling MVNs which provides a simple visual representation whilst remaining amenable to the wide range of PN analysis techniques and tools. We will explore how this framework compares to our safe PN one shortly.

4.3.4 Partial models

In this section, we extend our approach for handling partial BNs, thus providing much needed support in more general models.. Let $g_i \in \mathcal{MV}$ be an entity, such that $Y(g_i) =$

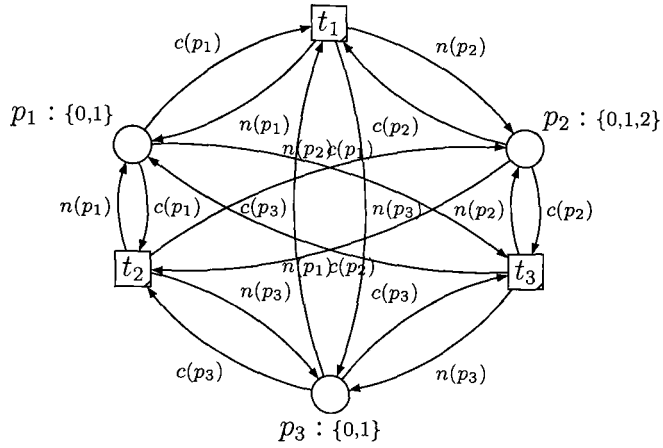


Figure 4.5: HLPN structure modelling asynchronous semantics of MVN from Figure 4.1, with the transition guards omitted for brevity.

$\{0, \ldots, m\}$ is its state space for some $m \geq 1$ and $N(g_i) = \{g_{i_1}, \ldots, g_{i_n}\}$ is its neighbourhood. Suppose for some current neighbourhood state $g_{i_1} \ldots g_{i_n} \in Y(g_{i_1}) \times \cdots \times Y(g_{i_n})$ that the next state $[g_i]$ is unknown; in other words, it could be any state $j \in \{0, \ldots, m\}$. Since we lack knowledge of the precise behaviour of g_i for this state, the most natural strategy for qualitatively modelling this phenomenon is to err on the side of caution and non–deterministically choose some next state $j \in Y(g_i)$.

To model this, we therefore add the term representing the problematic state to each next-state equation defining $[g_i\{j\}]$, for $j \in Y(g_i)$, before logic minimisation is applied. The result is that each compact next-state equation evaluates to true for the problematic state, and we then use a non–deterministic choice in the PN to cope with this.

g_1	g_2	$[g_1]$
0	0	0
0	1	0
0	2	-
1	0	0
1	1	0
1	2	2
2	0	1
2	1	1
2	2	2

Figure 4.6: Non–deterministic state transition table for an entity g_1.

For example, consider the state transition table for entity g_1 with state space $Y(g_1) = \{0, 1, 2\}$ in Figure 4.6, which specifies using the wild card '-' that the next state $[g_1]$ is unknown when $g_1 = 0$ and $g_2 = 2$, and as such can assume any of the values in $Y(g_1)$. In order to obtain the next-state equations for g_1, we simply add the term $g_0\{1\}g_2\{2\}$ representing the unknown state to the equations for $[g_1\{0\}]$, $[g_1\{1\}]$ and $[g_1\{2\}]$, resulting

in the following equations after logic minimisation:

$$\begin{aligned}
[g_1\{0\}] &= g_1\{0\} + g_1\{0,1\}g_2\{0,1\} \\
[g_1\{1\}] &= g_1\{0\}g_2\{2\} + g_1\{2\}g_2\{0,1\} \\
[g_1\{2\}] &= g_2\{2\}
\end{aligned}$$

Note that all three equations now evaluate to true when $g_1 = 0$ and $g_2 = 2$, thus modelling the fact that all three values are possible next states. Note for brevity, we do not show the PN model here, but simply remark that both frameworks can naturally cope with this situation without further modification.

It is also interesting to consider the case where we possess sufficient information to be able to impose restrictions on the possible next state $[g_i]$ at current state $g_{i_1} \ldots g_{i_n} \in Y(g_{i_1}) \times Y(g_{i_n})$. More specifically, we may know that $[g_i]$ assumes a value from a subset of $Y(g_i)$. For example, Figure 4.7 refines our example from Figure 4.6 with the knowledge that $[g_1]$ can only be the values 0 or 1 when $g_1 = 0$ and $g_2 = 2$.

g_1	g_2	$[g_1]$
0	0	0
0	1	0
0	2	{0,1}
1	0	0
1	1	0
1	2	2
2	0	1
2	1	1
2	2	2

Figure 4.7: Non–deterministic state transition table for an entity g_1.

In this case, the next-state equations are obtained straightforwardly by simply adding term $g_0\{1\}g_2\{2\}$ to the equations defining $[g_1\{0\}]$ and $[g_1\{1\}]$ and applying logic minimisation. As such, it is straightforward to refine the model in light of additional information, and PNs provide an interesting means of documenting this progression.

4.4 Framework comparison

The generalised modelling approaches presented in this chapter were used to extend the core implementation of GNAPN from Chapter 3. This was straightforward, since the existing data structures and construction algorithms were implemented generically. GNAPN provides a straightforward GUI for constructing PN models. Such models can either be specified as interactions in the interface, or can be loaded from an MVN file specification. In particular, logic minimisation techniques are applied prior to PN construction using MVSIS, both update semantics of MVNs are catered for and support is

in place for handling partial models. Altogether, GNAPN provides the biological community with a completely systematic means of studying MVN models of GRNs within the PN framework, and can be downloaded freely for academic use at *bioinf.ncl.ac.uk/gnapn*.

We now utilise GNAPN to assist in an interesting comparison which investigates the efficiency of the two PN frameworks against a number of MVN models. In particular, we focus on measuring the size of the canonical prefixes of the PN *unfoldings* [109] for both frameworks, as this appears to be a suitable means of ascertaining an approximation of their efficiency for analysis. Specifically, for each MVN model considered, our benchmarking approach is as follows:

- use GNAPN to construct both a safe and HLPN representation;
- use the tool PEP to expand the HLPN model into an equivalent safe one;
- record the number of places ($|P|$) and transitions ($|T|$) for both safe PN models;
- use the unfolder PUNF [109] to obtain canonical prefixes of both safe nets;
- record the number of events ($|E|$), which represent transition firings, and the number of cut-offs ($|C|$), which represent terminated behaviour due to repetition.

We take the value $|E| - |C|$ as a suitable size metric for the canonical prefix[1] and thus an indication of its efficiency for analysis (see [109] for a comprehensive discussion of unfolding). The table below presents our results for the following models:

(1) BN from Figure 2.1 consisting of three entities [6];

(2) Artificial MVN consisting of three entities; two Boolean and one with three states;

(3) BN model for the lysis-lysogeny switch in Lambda phage [206], consisting of five entities;

(4) BN case study model for carbon stress response in *E. coli* taken from Section 3.3.1 [167].

The "Safe PN" column shows results corresponding to our safe PN approach, whilst the "HLPN" column shows those corresponding to our HLPN one after expansion with PEP.

We observe in the $|T|$ column of both models that our safe PN framework enjoys substantially less transitions than the expanded PN produced by PEP. In fact, the number of transitions in our standard PN framework appear to increase in a linear fashion, whilst those from the expanded equivalent increase exponentially. This is due to the fact that PEP performs a naïve translation without considering any logical minimisation techniques.

Interestingly, however, the same cannot be said for the canonical prefix of our safe PN framework; in all cases, $|E| - |C|$ is significantly smaller for the expanded one. This appears to be due to the fact that the expanded PN exhibits more concurrency than our

[1]This metric was identified as a suitable measure for the size of the canonical prefix via personal communication with Victor Khomenko.

Model	Safe PN					HLPN				
	$\|P\|$	$\|T\|$	$\|E\|$	$\|C\|$	$\|E\| - \|C\|$	$\|P\|$	$\|T\|$	E	C	$\|E\| - \|C\|$
1	25	19	18	6	12	6	24	6	5	1
2	28	22	60	17	43	7	36	24	17	7
3	41	34	440	141	299	10	160	160	129	31
4	57	53	2662	1065	1597	14	896	896	769	127

Figure 4.8: Table comparing the standard and HLPN frameworks against a number of MVN data sets.

safe PN framework (which restricts concurrency in a number of areas due to the protocol it employs). As such, previously explored events are more readily visited by PUNF, and so the unfolding can be cut in more places, resulting in a smaller prefix. Thus, it appears that our HLPN framework is both more visually compact and more efficient to analyse. Similar results for different HLPN models have been reported in [72].

Note for the purpose of this comparison, we used PEP to expand the HLPNs into safe one before unfolding it with PUNF. However, PUNF is also able to calculate the canonical prefix directly from the HLPN, thus avoiding the costly translation into an (often) exponentially larger safe PN representation (see [111] for a comprehensive discussion). This further motivates the use of our HLPN framework for both its simple visual representation and efficiency for analysis.

4.5 Response to carbon stress in *E. coli*

In this section, we demonstrate our HLPN approach by revisiting the carbon stress response network in *E. coli* [167] (see Section 3.3.1 for initial Boolean study). This study applies GNAPN to a practical modelling situation, and makes use of a number of existing PN techniques and tools. In particular, we use PEP [81] to simulate and visualise our HLPN model, PUNF [109] to unfold it for analysis and CLP [109] for model checking. Note we focus here on the synchronous update semantics of MVNs following the approach taken in Chapter 3.

4.5.1 Constructing high-level Petri net model

Using the comprehensive data provided in [167], we are able to derive state transition tables defining the generalised logical behaviour of each regulatory entity in the carbon stress response network. For example, the state transition table defining the behaviour of SRNA is shown in Figure 4.9.

We then apply the logic minimisation techniques discussed in Section 4.3 to each state transition table to extract the minimised next-state equations shown in Figure 4.10. Finally, these equations are used to construct a HLPN model following the approach detailed in Section 4.3, which consists of seven places and a single transition with the behaviour of the MVN under the synchronous update semantics encoded as a transition

Fis	[SRNA]
0	0
1	0
2	0
3	1
4	1
5	1

Figure 4.9: State transition table showing SRNA behaviour under varying levels of Fis.

guard. Note this model construction process is fully automated by GNAPN, which is freely available for academic use at *bioinf.ncl.ac.uk/gnapn*, along with the model file.

4.5.2 Validating model

We now consider validating and exploring our HLPN model to see whether it captures a reasonable representation of the behaviour reported in [167]. In particular, we illustrate a range of PN techniques, from simulation through to model checking and mutant analysis.

Simulation

We simulate our HLPN model in PEP to ensure that it switches correctly between the exponential and stationary growth phases reported in [167]. To start, we check that our model captures the switch from the exponential to the stationary growth phase in the presence of carbon stress, by initialising the model to a marking representing exponential growth and by activating Signal. The sequence of steps resulting from this simulation are shown in Figure 4.11; the first column represents the initial marking and each subsequent column represents the observed behaviour thereafter.

The results show that our model correctly switches growth phases by entering a strong attractor cycle of period two that correctly represents the physiological conditions present in the stationary growth phase [167]. In particular, we see as expected that the level of SRNA and Fis decline to very low levels with the increasing concentration of CRP.

From this marking, we turn Signal off to represent an absence of carbon stress, and test that the model returns to the exponential growth phase. Figure 4.12 shows the trace of the simulation for each entity; the model enters an attractor cycle of period eight equivalent to the damped oscillationary behaviour observed in [167] caused by the mutual inhibition of *fis* and *crp*.

Dynamic properties

With a basic confidence in the correct working of our model, we now consider the application of more powerful analysis techniques based on model checking [109] to gain deeper insights into key behaviours. For example, it can be seen from the literature that the level of SRNA should remain low when Signal is active [167]. We can check this by considering all possible initial markings in which Signal is active (i.e. Signal = 1) and SRNA is inactive (i.e. SRNA = 0) using a similar idea to that presented in Figure 3.12

$$
\begin{aligned}
[SRNA\{0\}] &= Fis\{0,1,2\} \\
[SRNA\{1\}] &= Fis\{3,4,5\} \\
[Crp\{1\}] &= Crp\{0\} + Crp\{0,1,2\}Fis\{1,2,3,4,5\} \\
[Crp\{2\}] &= Crp\{3\}Fis\{1,2,3,4,5\} + Crp\{1\}Fis\{0\} \\
[Crp\{3\}] &= Crp\{2,3\}Fis\{0\} \\
[Cya\{1\}] &= Cya\{0\} \\
[Cya\{2\}] &= Cya\{1\} + Crp\{3\}Cya\{1,3\} + Signal\{1\} \\
[Cya\{3\}] &= Cya\{2,3\}Signal\{0\} + Cya\{2\} + Crp\{0,1,2\}Cya\{2,3\} \\
[Fis\{0\}] &= Crp\{1,2,3\}Cya\{1,2,3\}Fis\{0,1\}Signal\{1\} \\
[Fis\{1\}] &= Fis\{0\}Signal\{0\} + Fis\{0,1\}Signal\{0\}TopA\{2,3\} + Fis\{2\}TopA\{2,3\} + \\
&\quad Fis\{0,1\}GyrAB\{0,1\}Signal\{0\} + Fis\{2\}GyrAB\{0,1\} + Cya\{0\}Fis\{0\} + \\
&\quad Crp\{0\}Fis\{0\} + Cya\{0\}Fis\{0,1\}TopA\{2,3\} + Crp\{0\}Fis\{0,1\}TopA\{2,3\} + \\
&\quad Cya\{0\}Fis\{0,1\}GyrAB\{0,1\} + Crp\{0\}Fis\{0,1\}GyrAB\{0,1\} + \\
&\quad Crp\{1,3\}Cya\{1,3\}Fis\{2\}Signal\{1\} + Crp\{2,3\}Cya\{1,3\}Fis\{2\}Signal\{1\} + \\
&\quad Crp\{1,3\}Cya\{2,3\}Fis\{2\}Signal\{1\} + Crp\{2,3\}Cya\{2,3\}Fis\{2\}Signal\{1\} \\
[Fis\{2\}] &= Fis\{3\}GyrAB\{0,1\} + Crp\{1,2,3\}Cya\{1,2,3\}Fis\{3\}Signal\{1\} + \\
&\quad Fis\{3\}TopA\{2,3\} + Crp\{0\}Fis\{1\}GyrAB\{2,3\}TopA\{0,1\} + \\
&\quad Cya\{0\}Fis\{1\}GyrAB\{2,3\}TopA\{0,1\} + Fis\{1\}GyrAB\{2,3\}Signal\{0\}TopA\{0,1\} \\
[Fis\{3\}] &= Fis\{4\}GyrAB\{0,1\} + Crp\{1,2,3\}Cya\{1,2,3\}Fis\{4\}Signal\{1\} + \\
&\quad Fis\{4\}TopA\{2,3\} + Crp\{0\}Fis\{2\}GyrAB\{2,3\}TopA\{0,1\} + \\
&\quad Cya\{0\}Fis\{2\}GyrAB\{2,3\}TopA\{0,1\} + Fis\{2\}GyrAB\{2,3\}Signal\{0\}TopA\{0,1\} \\
[Fis\{4\}] &= Fis\{5\} + Crp\{0\}Fis\{3,5\}GyrAB\{2,3\}TopA\{0,1\} + \\
&\quad Cya\{0\}Fis\{3,5\}GyrAB\{2,3\}TopA\{0,1\} + \\
&\quad Fis\{3,5\}GyrAB\{2,3\}Signal\{0\}TopA\{0,1\} \\
[Fis\{5\}] &= Crp\{0\}Fis\{4\}GyrAB\{2,3\}TopA\{0,1\} + Cya\{0\}Fis\{4\}GyrAB\{2,3\}TopA\{0,1\} + \\
&\quad Fis\{4\}GyrAB\{2,3\}Signal\{0\}TopA\{0,1\} \\
[GyrAB\{0\}] &= Fis\{4,5\}GyrAB\{0,1\} \\
[GyrAB\{1\}] &= Fis\{4,5\}GyrAB\{2\} + Fis\{0,1,2,3\}GyrAB\{0\} \\
[GyrAB\{2\}] &= GyrAB\{3\}TopA\{0\} + Fis\{0,1,2,3\}GyrAB\{1\} + Fis\{4,5\}GyrAB\{3\} \\
[GyrAB\{3\}] &= Fis\{0,1,2,3\}GyrAB\{2\} + Fis\{0,1,2,3\}GyrAB\{3\}TopA\{1,2,3\} \\
[TopA\{0\}] &= TopA\{1\} + GyrAB\{0,1\}TopA\{0,1\} + Fis\{0,1,2,3\}TopA\{0,1\} \\
[TopA\{1\}] &= Fis\{4,5\}GyrAB\{2,3\}TopA\{0,2\} + TopA\{2\} \\
[TopA\{2\}] &= TopA\{3\}
\end{aligned}
$$

Figure 4.10: Minimised next-state equations for *E. coli* MVN.

from Chapter 3. We then use CLP with the constraint:

$$\text{SRNA} > 0,$$

resulting in the output shown below which correctly confirms that SRNA does not become active under these conditions.

```
solving: 0.01 sec.
1 compatible vector explored

the marking is unreachable
```

A similar check can be performed to see if entities TopA and GyrAB can both become inactive simultaneously during the stationary phase, as they appear to work in opposite directions to affect Fis. We check all possible initial states in which TopA and GyrAB are not simultaneously inactive with the constraint:

$$\text{TopA} + \text{GyrAB} = 2,$$

SRNA	1	1	1	0	0	0	0	0	0
Crp	1	1	1	1	1	2	3	3	3
Cya	3	3	3	3	3	3	3	3	3
TopA	0	0	0	0	0	0	0	0	0
Fis	4	3	2	1	0	0	0	0	0
Signal	1	1	1	1	1	1	1	1	1
GyrAB	2	3	2	3	2	3	2	3	2

Figure 4.11: Simulating the switch from exponential to stationary phase in *E. coli.*

SRNA	0	0	0	0	1	1	1	1	1	0	0	0	1	1	1	1
Crp	3	3	2	1	1	1	1	1	1	1	1	1	1	1	1	1
Cya	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3
TopA	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
Fis	0	1	2	3	4	5	4	3	2	1	2	3	4	5	4	3
Signal	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
GyrAB	2	3	2	3	2	1	0	0	1	2	3	2	3	2	1	0

Figure 4.12: Simulating re-entry from stationary to exponential phase in *E.coli.*

which specifies a state violating the mutual exclusion property, and CLP is able to confirm that no such state exists, which appears to be coherent with [167].

Mutant analysis

The final step in our study is to investigate the affect of "fixing" the state of an entity in the model. In order to achieve this, we simply ignore the state transition table for the entity so that it becomes an input signal. This corresponds to the experimental approach of creating mutants in which genes are knocked out or overexpressed, and provides a means of investigating the robustness of the network when key components do not function as normal. Furthermore, the results of such tests provide useful insights and allow hypotheses to be formed which can then be verified in the laboratory.

We investigate the affect that knocking out and overexpressing the entities *crp*, *cya*, *gyrAB* and *topA* has on the production of Fis and consequently the level of SRNA. In particular, we consider whether SRNA can be inhibited in the absence of carbon stress, and alternatively whether it can become inactive when no stress is present. We perform these tests by first setting Signal = Fis = SRNA = 0 and then by knocking out and overexpressing the remaining entities one at a time. We then repeat the analysis with Signal = 1. The observed results are summarised in Figure 4.13.

When carbon stress is absent, we notice that knockout and overexpression of *crp* and *cya* allows for the production of SRNA. However, this does not occur when we knockout and overexpress *gyrAB* and *topA*, respectively. In the case of knocking out *gyrAB*, a low concentration of GyrAB prevents an increase in negative DNA supercoiling; thus Fis production is reduced and so SRNA production is low. On the other hand, overexpression of *topA* inhibits the amount of negative DNA supercoiling, thus reducing Fis production

Entity	Knock out	Overexpressed	Knock out (s)	Overexpressed (s)
CRP	Yes	Yes	Yes	No
Cya	Yes	Yes	Yes	No
GyrAB	No	Yes	No	No
TopA	Yes	No	No	No

Figure 4.13: Results of knocking out and overexpressing entities, where (s) denotes that carbon stress is present.

and subsequently SRNA [167].

When carbon stress is present, however, we notice different behaviour; knocking out both *crp* and *cya* allows for the production of SRNA even under carbon stress. This is due to the reduced activation of the implicit complex cAMP.CRP, which in turn does not repress *fis* so strongly. Thus, SRNA is allowed to acculmulate. However, when we overexpress *crp* and *cya*, the opposite occurs and SRNA is not produced as expected [167].

4.6 Initial comparisons between *E. coli* models

Our case study demonstrated the application of a range of PN techniques and tools to validate our approach. In Section 3.3.1, we considered a similar study using Boolean modelling, and showed that a number of key dynamics could be captured that appeared to be consistent with the literature [167]. This therefore naturally raises questions about the relationship between these two models; one in particular is whether the Boolean model provides a "sufficient" representation of the *E. coli* network.

It is well understood that continuous models based on differential functions capture behaviour that does not correspond with that observed in the Boolean domain (see [77,188] for studies). However, such a distinction does not always appear to be so obvious for the discrete models considered here, especially when one starts to look at larger systems. It is therefore interesting to explore this in a little more depth by comparing and contrasting our two models for this *E. coli* GRN.

On inspection, both models appear to capture similar fundamental behaviour, and correctly switch between the two growth phases depending on whether carbon stress is present or not. Furthermore, the mutual inhibition between *fis* and *crp*, which plays an important part in the switch from the exponential to the stationary phase [167], appears to be faithfully represented in both models. However, we observe that TopA erroneously reaches an activated level in the BN model during the switch from the exponential to the stationary growth phase. In the MVN model, the level of TopA correctly remains low, allowing the amount of DNA supercoiling to increase. In fact, the only time that TopA can temporarily rise in the MVN model is when TopA = 0, Fis = 5 and GyrAB = 3; TopA then immediately becomes inactive. The BN representation, however, is unable to model this intermediate level for TopA, and so when Fis and GyrAB are fully active, TopA also becomes fully active, leading to questionable results.

Another interesting comparison concerns the results of mutant analysis. In both case studies, we performed the same mutant analysis on the entities *crp*, *cya*, *gyrAB* and *topA*

to compare how the production of SRNA is affected. In Figure 4.14, we show the mutant analysis results for the BN representation, and highlight in bold any discrepancies with the MVN equivalent.

Entity	Knock out	Overexpressed	Knock out (s)	Overexpressed (s)
CRP	Yes	Yes	Yes	**Yes**
Cya	Yes	Yes	Yes	**Yes**
GyrAB	No	Yes	No	**Yes**
TopA	Yes	No	**Yes**	No

Figure 4.14: Results of entity knock out and overexpression in the BN model.

The results show that in a number of cases, SRNA reaches an activated level in the Boolean mutants, whereas it remains inactive in the MVN model. This appears to be due to the BN model being less restrictive with respect to the activation of Fis; in other words, there are less conditions required for Fis to become active.

Although there are many similarities between the two models, we have noted some cases which indicate some possible limitations with the BN model. However, it is unclear whether this means that BNs are insufficient for this particular modelling task; indeed, it would be interesting to see if there exists an alternative BN model which does capture the required behaviour.

These considerations lead to more general questions concerning the level of detail required in a model to capture some desired behaviour. In particular, it is unclear how one reasons about such behaviour, and how "insufficient" models can be determined. Indeed, what we require is a formal framework for answering these questions, and we present this in Chapter 5.

4.7 Discussion

4.7.1 Conclusions

To address limitations with Boolean modelling, this chapter extended our PN approach from Chapter 3 to cater for MVNs [206]. In particular, efficient logic minimisation techniques [146] formed an integral part of model construction to ensure compactness. We developed two complementary PN approaches. The first was a novel safe PN framework which utilised the negative information present in the logical terms to significantly reduce the number of transitions required. This modelling technique was made possible by utilising the extra places in safe PNs, and so appears to be advantageous over non-safe ones for both compactness and amenability to analysis (for example, see [46,187] for a similar approach using non-safe nets). In particular, the increased number of places enabled us to model each logical term using only a *single* transition, and so offers promising scope for future approaches with compact safe PNs. Our second approach considered the use of HLPNs [28], and we showed how a visually-compact model could be systematically constructed by translating the next-state equations into transition guards. This resulted in a

generalised HLPN modelling framework which offers both visual simplicity and analysis potential.

We compared both PN frameworks to investigate how efficient they were to analyse, and the results were surprising. Whilst our safe PN framework was significantly more compact than the expanded HLPN equivalent, it turned out that this was not the case for the corresponding unfoldings. In fact, we found the unfoldings of the HLPN to be a factor of ten smaller in general using the $|E| - |C|$ metric. This appeared to be due to the fact that our safe PN approach imposed restrictions on the concurrency, and so it took more events before repetition was detected. On the other hand, the expanded HLPN had significantly more transitions with high concurrency, allowing more cut-offs to be inserted into the prefix. This therefore indicated that the HLPN framework was both visually-compact and more efficient to analyse than the safe framework. However, we argued that visual compactness could sometimes hamper the study of the regulatory interactions, and so proposed the safe PN framework as an interesting alternative in such situations.

The approaches presented have been used to extend our PN construction tool GNAPN. This was a straightforward task since the core classes of GNAPN were implemented in a generic way, highlighting their extensibility to future work. GNAPN is able to automatically construct safe and HLPN models of an MVN under both the synchronous and asynchronous semantics, utilise logic minimisation techniques with the auxiliary logic minimisation tool MVSIS [42] and cope with partial models. Furthermore, it facilitates the export of PN models into a number of widely used formats. Thus, GNAPN provides what appears to be one of the most comprehensive PN model construction approaches for MVNs, and contributes much-needed support to the biological community. GNAPN is freely available for academic use at *bioinf.ncl.ac.uk/gnapn*.

We demonstrated our HLPN framework by revisiting the carbon stress response network in *E. coli*. We used GNAPN to automatically construct a synchronous HLPN model from the comprehensive data provided in [167], and demonstrated the application of a number of PN analysis techniques which confirmed reported properties in the literature. We started by performing qualitative simulations using the PEP tool to ensure that the model captured the fundamental switch between growth phases. For example, we started the HLPN from an initial marking representing exponential phase conditions, and turned Signal on to represent the presence of carbon starvation. As such, we were able to verify that our model correctly captured this essential switch.

We then considered the application of powerful model checking techniques to our HLPN to further investigate these hypotheses. For instance, we observed from the literature [167] and from simulations that the level of SRNA should never become active during the stationary phase. Using the model checking tool CLP and unfolder PUNF [109], we were able to show that this property held from all reachable states in the stationary phase. Thus, we demonstrated the application of powerful model checking proofs which are not directly amenable to MVNs. Indeed, we also applied model checking to the study of mutants. We investigated the effects of knocking out and overexpressing key network entities, by fixing their state to both high and low values. We performed this one entity at a time, and observed how SRNA production was affected in the presence of abnormal

conditions. This yielded results consistent with biological intuition from the literature, thus providing further confidence in our model.

This case study also identified some interesting behavioural differences with the BN *E. coli* model from Chapter 3 which raise questions concerning their formal relationship. In particular, it has highlighted potential limitations which Boolean modelling, and led us to consider the level of detail required by a model to capture certain behaviours correctly. Clearly, for the purposes of analysis and interpretation, it is desirable to consider the simplest model which correctly captures the property of interest. However, it is not always clear whether a model is sufficient for purpose, or indeed whether it can be further simplified. For example, although we have shown doubt that the Boolean model captures certain subtle behaviours correctly, it is unclear whether this means it is an 'incorrect' representation; indeed, how do we define 'incorrect'? Moreover, does there exist an alternative Boolean model which *is* 'correct'? In order to answer these questions, we require a formal framework for reasoning about such models, and this is the contribution of Chapter 5.

4.7.2 Future work

Further case studies are now required to investigate the application of our HLPN framework and support tool GNAPN. In particular, we are interested in considering the asynchronous semantics of MVNs and the techniques required to cope with the inherent increase in behavioural complexity this entails.

In addition, the questions concerning the relationship between the two *E. coli* models are intriguing and call for further investigation. Specifically, the development of formal and systematic techniques appear to be of paramount importance if they are to cope with the challenges of large biological networks, and we address this in Chapter 5.

Compositional model construction and analysis appear to offer much promise for our techniques (see [29, 36, 52, 218] for studies). A compositional approach to model construction would be consistent with the biological intuition that GRNs function as a number of interlinked sub-units [87], and could lead to improved analysis efficiency. Specifically, we see compositional model checking techniques [36, 218] as an important area of research here.

Finally, we wish to investigate techniques for visualising the complex behaviour of a GRN. Model checking is a powerful analysis technique and can provide many insights in to the dynamics of a system, but can be hard to apply meaningfully if one does not have a basic understanding. We see the intuitive visualisation of a model's behaviour as crucial for obtaining this initial understanding, and techniques for coping with the subsequent state space explosion will need careful consideration.

4.7.3 Sources

The work presented in this chapter was presented at the 4th Integrative Bioinformatics Workshop 2007 [20]. Initial ideas for the HLPN framework were taken from [53]. Finally, the metric for determining the size of a canonical prefix was given by personal communication with Victor Khomenko.

Chapter 5

Relationships Between Models at Different Levels of Abstraction

Chapters 3 and 4 incrementally developed a generalised qualitative PN modelling approach for GRNs, and we demonstrated its application with case studies of the carbon stress response network in *E. coli* under both Boolean and multi-level discretisation. This then formed the basis of an interesting comparison in Section 4.6, in which we were able to compare and contrast the relative properties of the simpler Boolean and more expressive multi-valued models in an informal setting. Whilst this study found there to be significant agreement between the two approaches, it also highlighted a number of behavioural differences. Specifically, we found behaviours of the MVN model which did not appear to be captured in the BN equivalent. This therefore raised more general questions concerning their relationship. In particular, it is interesting to consider the scope and limitations of Boolean modelling and the situations in which multi-valued modelling is necessary.

Studies in the literature have considered comparisons between BNs and differential descriptions [14, 77, 142] (see [188] for a good review). All studies confirmed that steady state dynamics captured by BNs *do* correspond qualitatively to analogous steady states in the differential description (but that the converse it not always true), whilst cyclic behaviour in BNs *does not* always correspond. Glass *et al.* [77] exemplified the latter with the idea of a simple feedback loop where entity g_i activated g_j whilst g_j repressesed g_i. In the BN, this feedback loop yielded a four-state cycle, whilst in the differential description it fell into a single steady state. This led Glass *et al.* [77] (and later [14] who considered similar work) to develop a technique for determining whether cyclic behaviour in small BNs would appear in the differential description. However, there do not appear to be any such investigations for MVNs, nor much-needed systematic approaches for practical modelling.

In this chapter, we begin to address these concerns with an initial investigation into the development of a systematic framework for reasoning about the relationship between MVNs at different levels of abstraction. We propose a formal refinement theory which captures an assumption of what it means for an MVN to be representative of a more complex one, and consider the development of algorithms based on this for systemati-

cally deriving and validating such models. Application of our approach provides some interesting insights which motivate the necessity for multi-valued modelling in certain situations, and sheds light on the relationship between the BN and MVN models of the *E. coli* from Chapters 3 and 4. At a higher level, this work contributes the foundations of new techniques for systematically reasoning about a model's behaviour, and has real practical implications for addressing state space explosion in analysis.

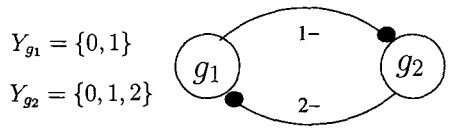
The rest of this chapter is structured as follows. Section 5.1 provides an introduction to the notation, theory and properties of our proposed refinement theory assumption. We then demonstrate the application of our framework using a small example which motivates the need for multi-valued modelling in Section 5.2. In Section 5.3, we develop algorithms for systematically simplifying and validating MVNs according to our refinement theory. A detailed benchmarking analysis of these algorithms is presented in Section 5.4 based on their implementation as a Java tool REFINER. In Section 5.5, we then revisit the two case study models for *E. coli* from Chapters 3 and 4, and apply our framework to shed light on the behavioural discrepancies noted. Finally, Section 5.6 provides some concluding remarks.

5.1 Developing a refinement theory

In this section, we introduce the basic notations, properties and theory of our framework for reasoning about MVNs at different levels of abstraction.

5.1.1 Basic notations

Consider Figure 5.1(a), which shows a simple MVN (see Definition 4.1) model $\mathcal{MV}_1$ consisting of two entities g_1 and g_2 with state spaces $Y(g_1) = \{0, 1\}$ and $Y(g_2) = \{0, 1, 2\}$, respectively.



(a)

g_2	$[g_1]$
0	1
1	1
2	0

g_1	g_2	$[g_2]$
0	0	1
0	1	2
0	2	2
1	0	0
1	1	0
1	2	1

(b)

Figure 5.1: (a) An MVN model $\mathcal{MV}_1$ with two entities, and (b) the state transition tables defining the corresponding next-state functions F.

The corresponding state transition tables shown in Figure 5.1(b) describe the complete behaviour of $\mathcal{MV}_1$ (as previously discussed, the state transition tables are a representation of the next-state functions F). The idea is that entity g_1 inhibits g_2, and g_2 inhibits g_1 when $g_2 = 2$. Note throughout the remainder of this chapter, we will simply

use the state transition tables to represent MVNs for convenience, and in a slight abuse of notation, will denote these tables as $\mathcal{MV}$.

In order to reason about MVN models, it is often convenient to formalise an individual input-output pair in the state transition table of an entity g_i, which we call a *rule*. Note rules are just a way of representing rows in the state transition table for an entity.

Definition 5.1 (Rule). *Let $g_i \in \mathcal{MV}$ be an entity with neighbourhood $N(g_i) = \{g_{i_1}, \ldots, g_{i_n}\}$. A rule for g_i is a tuple $r = (g_i, IS \rightarrow f_{g_i}(IS))$ where $IS \in Y(g_1) \times \cdots \times Y(g_n)$ and f_{g_i} is the next-state function for g_i; in other words, a function capturing the state transition table of g_i.*

For example, one possible rule for entity $g_1 \in \mathcal{MV}_1$ would be $r = (g_1, 0 \rightarrow 1)$. A state transition table for an entity g_i is therefore *deterministic* if there exists exactly one rule for each possible state of $N(g_i)$ (for instance, all tables for $\mathcal{MV}_1$ are deterministic), and *non-deterministic* if there exists more than one rule for some state of $N(g_i)$. Note in this chapter, we will assume deterministic models unless otherwise stated.

A trajectory of global states from some initial state in a model is called a *trace*. Note the sequence of these states depends on the update semantics used, but here we focus on the synchronous update semantics (see Section 2.2.3).

Definition 5.2 (Trace). *Let $S_0 \in S_{\mathcal{MV}}$ be a global state. A trace is a state list $\sigma(S_0) = \langle S_0, S_1, \ldots, S_n \rangle$ containing the sequence of global states, such that for each global state in the sequence, the next global state is obtained by updating each entity simultaneously, resulting in a synchronous simulation of $\mathcal{MV}$. Furthermore, we have that $S_0, \ldots, S_{n-1}$ are unique and $S_n = S_i$ for some $i \in \{0, \ldots, n-1\}$.*

Note that traces are infinite objects under the synchronous semantics since they will eventually fall into a cycle. Thus, the above definition defines a finite canonical representation which specifies a trace up to the first repeated state. For example, one possible trace of $\mathcal{MV}_1$ is $\sigma(00) = \langle 00, 11, 10, 10 \rangle$. The complete set of traces starting from each state $S \in S_{\mathcal{MV}}$ therefore captures the complete behaviour of some model $\mathcal{MV}$ under the synchronous semantics, and is referred to as the *trace semantics*.

Definition 5.3 (Trace semantics). *The trace semantics of a model $\mathcal{MV}$ is the set of traces $TS(\mathcal{MV}) = \{\sigma(S) \mid S \in S_{\mathcal{MV}}\}$.*

For example, $\mathcal{MV}_1$ has a state space of size $|S_{\mathcal{MV}_1}| = 6$, and so $TS(\mathcal{MV}_1)$ consists of the following six traces:

$$\begin{aligned}
\sigma(00) &= \langle 00, 11, 10, 10 \rangle \\
\sigma(01) &= \langle 01, 12, 01 \rangle \\
\sigma(02) &= \langle 02, 02 \rangle \\
\sigma(10) &= \langle 10, 10 \rangle \\
\sigma(11) &= \langle 11, 10, 10 \rangle \\
\sigma(12) &= \langle 12, 01, 12 \rangle
\end{aligned}$$

The trace semantics form a foundational part of this work, since it completely characterises the behaviour of an MVN, and thus acts as a medium through which models can be reasoned about. When comparing two models $\mathcal{MV}'$ and $\mathcal{MV}$, we assume that they both deterministic, and have the same entities and structure (i.e. same neighbourhoods and number of interactions). We then regard $\mathcal{MV}'$ as being *simpler* than $\mathcal{MV}$ (i.e. at a higher level of abstraction) if, for some entity $g'_i \in \mathcal{MV}'$ with corresponding entity $g_i \in \mathcal{MV}$, we have that $Y(g'_i) \subset Y(g_i)$, and for every other entity $g'_j \in \mathcal{MV}'$ with corresponding entity $g_j \in \mathcal{MV}$, we have that $Y(g'_j) = Y(g_j)$ (of course, it may be the case that multiple entities in $\mathcal{MV}'$ have smaller state spaces). Note we may also refer to one model being more *complex* than another, but this definition is now clear.

This chapter proposes a *refinement theory* which formally captures a relationship specifying what it means for a deterministic MVN to be represented by a simpler one. The basic idea is that the trace semantics of the latter should be consistent with the trace semantics of the former. However, before we can define this consistency, we must first introduce a means of relating the state spaces of the two MVNs.

5.1.2 Relating state spaces between models

The state space of an entity in an MVN represents a logical interpretation of the threshold levels at which it interacts at various points in the network. Intuitively, reducing this state space therefore corresponds to merging some of these thresholds at which interactions occur. Thus, such a reduction appears to provide a biologically-justifiable simplification, as well as an interesting means for exploring the necessity for the thresholds. We now formalise this with the notion of a *state mapping.*

Definition 5.4 (State mapping). *Let $g_i \in \mathcal{MV}$ be an entity such that $Y(g_i) = \{0, \ldots, m\}$ for some $m > 1$. Then a state mapping ϕ_{g_i} for g_i is a surjective mapping $\phi_{g_i} : \{0, \ldots, m\} \to \{0, \ldots, n\}$ where $0 < n < m$.*

The idea is that a state mapping reduces the states that an entity can be in by mapping its current state space to a smaller set of states. As such, the behaviour of the MVN is simplified as the range of states it can be in is restricted. Note we only consider state mappings with a codomain larger than one, since a singular state entity does not appear to be of biological interest.

As an example, consider entity $g_2 \in \mathcal{MV}_1$ from Figure 5.1 which has state space $Y(g_2) = \{0, 1, 2\}$. Since it is only meaningful to simplify $g_2 \in \mathcal{MV}_1$ to a Boolean entity, one possible state mapping to achieve this would be:

$$\phi_{g_2} = \{0 \mapsto 0, 1 \mapsto 0, 2 \mapsto 1\},$$

which merges states 0 and 1 into a single state 0, and translates state 2 into 1. In general, a state mapping can be applied to the trace semantics of an MVN and its state transition tables using the following approach.

Definition 5.5 (Model translation). *A state mapping ϕ_{g_i} can be applied to a global state $S = s_1 \ldots s_k \in S_{\mathcal{MV}}$ as $\phi_{g_i}(s_1 \ldots s_k) = s_1 \ldots \phi_{g_i}(s_i) \ldots s_k$. We can then lift ϕ_{g_i} to a trace*

$\sigma(S_0)$ from some initial state $S_0 \in S_{\mathcal{MV}}$ by applying it to each global state it contains $\phi_{g_i}(\langle S_0, \ldots, S_n \rangle) = \langle \phi_{g_i}(S_0), \ldots, \phi_{g_i}(S_n) \rangle$. Finally, we can lift ϕ_{g_i} to the trace semantics $TS(\mathcal{MV})$ of a model $\mathcal{MV}$ by applying it to each trace $\sigma(S) \in TS(\mathcal{MV})$ for $S \in S_{\mathcal{MV}}$, i.e. $\phi_{g_i}(TS(\mathcal{MV})) = \{\phi_{g_i}(\sigma(S)) \mid \sigma(S) \in TS(\mathcal{MV})\}$. Similarly, we can apply ϕ_{g_i} to the state transition tables of a model $\mathcal{MV}$ (i.e. $\phi_{g_i}(\mathcal{MV})$) in the obvious way by translating each occurrence of g_i in the corresponding rules.

In some cases, a trace may be translated by a state mapping resulting in a new sequence of states which is non–deterministic. For example, suppose we have the trace

$$\sigma(00) = \langle 00, 01, 02, 12, 02 \rangle ,$$

which we translate with state mapping $\phi_{g_2} = \{0 \mapsto 0, 1 \mapsto 1, 2 \mapsto 1\}$ (where g_2 is the second component of the global state). The resulting sequence of states is

$$\sigma(00) = \langle 00, 01, 01, 11, 01 \rangle ,$$

and it can be observed that this is now non–deterministic, since global state 01 leads to both 01 and 11. Such behaviour cannot possibly be captured by a deterministic model, and so we distinguish these sequences as *invalid traces* and ignore them during the refinement process (see Section 5.3).

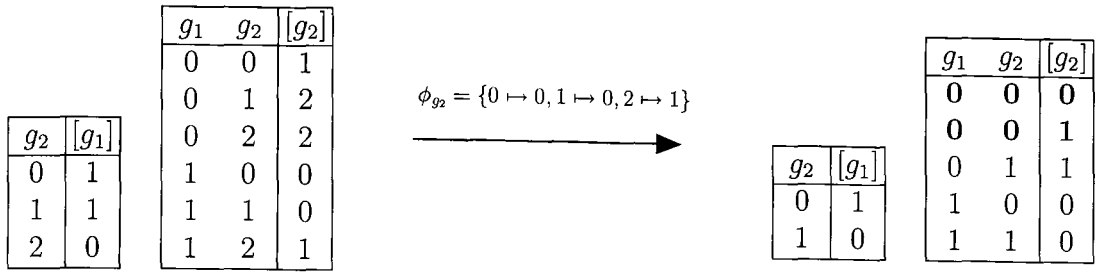
Continuing with our running example, ϕ_{g_2} can be applied to the trace semantics $TS(\mathcal{MV}_1)$ shown above, resulting in the translated trace semantics $\phi_{g_2}(TS(\mathcal{MV}_1))$ in which the states of g_2 have been reduced accordingly:

$$\begin{aligned}
\phi_{g_2}(\sigma(00)) &= \langle 00, 10, 10 \rangle \\
\phi_{g_2}(\sigma(01)) &= \langle 00, 11, 00 \rangle \\
\phi_{g_2}(\sigma(02)) &= \langle 01, 01 \rangle \\
\phi_{g_2}(\sigma(10)) &= \langle 10, 10 \rangle \\
\phi_{g_2}(\sigma(11)) &= \langle 10, 10 \rangle \\
\phi_{g_2}(\sigma(12)) &= \langle 11, 00, 11 \rangle
\end{aligned}$$

It can be noted that $\phi_{g_2}(TS(\mathcal{MV}_1))$ is now non–deterministic, as we have two different traces beginning with the same state 00. This occurs as we are viewing the more complex set of behaviours captured by $TS(\mathcal{MV}_1)$ from a simpler perspective. We can also apply ϕ_{g_2} to the state transition tables of Figure 5.1(b), and we show this in Figure 5.2. Again, we can see that this non–determinism is manifested in the table for g_2, since there are two rules for input state 00 (highlighted in bold) which give different outputs.

Usually, there is more than one state mapping which can be applied to reduce $|Y(g_i)|$ from m to n states, for $1 \leq n < m$. The complete set of all such state mappings is formalised by the *mapping set.*

Definition 5.6 (Mapping set). *The mapping set $MS_{g_i}^{m \to n}$ of an entity g_i, for $1 \leq n < m$, is the set $MS_{g_i}^{m \to n} = \{\phi_{g_i} \mid \phi_{g_i} : \{0, \ldots, m-1\} \to \{0, \ldots, n-1\}\}$ containing all possible state mappings which reduce $|Y(g_i)|$ from m to n states.*



g_2	$[g_1]$
0	1
1	1
2	0

g_1	g_2	$[g_2]$
0	0	1
0	1	2
0	2	2
1	0	0
1	1	0
1	2	1

g_2	$[g_1]$
0	1
1	0

g_1	g_2	$[g_2]$
0	**0**	**0**
0	**0**	**1**
0	1	1
1	0	0
1	1	0

Figure 5.2: Mapping (1) applied to the state transition tables of Figure 5.1.

It can be shown that $|MS_{g_i}^{m \to n}|$ will contain

$$\sum_{j=1}^{m-n+1} {}^{m-1}C_j \cdot n!$$

state mappings, where $n!$ is the number of *isomorphic* mappings for some n[1]. That is, we view this as the problem of counting the number of arrangements in which the same character can appear j times in an $m-1$ bit string (to ensure that the codomain of the mapping is two or more), where j cannot exceed $m-n+1$ for the reason that all n states must be present in the codomain.

For example, $g_2 \in \mathcal{MV}_1$ can be mapped to a Boolean entity by six state mappings, and so $MS_{g_2}^{3 \to 2}$ contains:

$$\begin{aligned}
(1)\ \phi_{g_2} &= \{0 \mapsto 0, 1 \mapsto 0, 2 \mapsto 1\} \\
(2)\ \phi_{g_2} &= \{0 \mapsto 0, 1 \mapsto 1, 2 \mapsto 1\} \\
(3)\ \phi_{g_2} &= \{0 \mapsto 0, 1 \mapsto 1, 2 \mapsto 0\} \\
(4)\ \phi_{g_2} &= \{0 \mapsto 1, 1 \mapsto 1, 2 \mapsto 0\} \\
(5)\ \phi_{g_2} &= \{0 \mapsto 1, 1 \mapsto 0, 2 \mapsto 0\} \\
(6)\ \phi_{g_2} &= \{0 \mapsto 1, 1 \mapsto 0, 2 \mapsto 1\}
\end{aligned}$$

Furthermore, we observe that there are $2! = 2$ isomorphic state mappings for each configuration in $MS_{g_2}^{3 \to 2}$: (1) $\cong$ (4); (2) $\cong$ (5); and (3) $\cong$ (6). Interestingly, it turns out that we can ignore such isomorphic mappings in $MS_{g_i}^{m \to n}$, as shown by the following theorem.

Theorem 5.1. *Let ϕ_{g_i} and ϕ'_{g_i} be two isomorphic state mappings for some entity $g_i \in \mathcal{MV}$. Then $\phi_{g_i}(TS(\mathcal{MV})) \cong \phi'_{g_i}(TS(\mathcal{MV}))$.*

Proof. Follows directly from the definition of an isomorphism. □

Intuitively, a trace can be seen simply as a sequence of symbols which have no meaning in this context. Thus, two traces which have the same structure but with consistently

[1]We say two mappings ϕ_g and ϕ_h are isomorphic to one another, written $\phi_g \cong \phi_h$, if there exists a bijective mapping ψ such that $\psi \circ \phi_g = \phi_h$ and $\psi^{-1} \circ \phi_h = \phi_g$.

different symbols (i.e. they are isomorphic to one another) can be seen as equivalent. As such, the size of $MS_{g_i}^{m \to n}$ now becomes

$$\sum_{j=1}^{m-n+1} {}^{m-1}C_j,$$

and consists of only structurally unique state mappings. For example, $MS_{g_2}^{3 \to 2}$ for $g_2 \in \mathcal{MV}_1$ now only consists of the state mappings (1), (2) and (3) (although any combination of the three structurally-unique mappings is sufficient).

To generalise the link between the state space of an MVN and a simpler one, we define a *mapping vector*.

Definition 5.7 (Mapping vector). *For a model $\mathcal{MV}$ with k entities, the mapping vector ϕ is the vector $\phi = \langle \phi_{g_1}, \ldots, \phi_{g_k} \rangle$, where each ϕ_{g_i}, for $i = 1 \ldots k$, either reduces $Y(g_i)$ or is the identity mapping $I_{g_i} : Y(g_i) \to Y(g_i)$, such that $I_{g_i}(x) = x$ for all $x \in Y(g_i)$.*

A mapping vector completely relates the state space of an MVN and a simpler one by specifying a mapping for each entity. Note $\phi(TS(\mathcal{MV}))$ and $\phi(\mathcal{MV})$ are performed in the obvious way by iteratively applying each mapping in the vector to the corresponding entity using the methodology presented in Definition 5.5.

5.1.3 Refinement theory: a relationship assumption

Having introduced mapping vectors as a means of relating the state spaces of two MVNs, we now define our refinement theory by introducing the notion of a *refinement*; an MVN whose behaviour is simpler yet *consistent* with another.

Definition 5.8 (Refinement). *Let $\mathcal{MV}$ and $\mathcal{MV}'$ be two deterministic MVNs with the same structure, and ϕ be a mapping vector which translates the state spaces of the entities in $\mathcal{MV}$ to those in $\mathcal{MV}'$. Then we say that $\mathcal{MV}'$ refines $\mathcal{MV}$ under ϕ, denoted $\mathcal{MV}' \lhd^{\phi} \mathcal{MV}$, iff $TS(\mathcal{MV}') \subseteq \phi(TS(\mathcal{MV}))$ holds.*

Observe that refinements are deterministic models, whilst $\phi(TS(\mathcal{MV}))$ can be non–deterministic (as shown previously). Refinements therefore nearly always capture less behaviour than the more complex model under ϕ, although we will consider the exception to this shortly. Subsequently, an MVN may have a number of possible refinements, and we formalise these with the notion of a *refinement set*.

Definition 5.9 (Refinement set). *A refinement set $Ref_{\mathcal{MV}}^{\phi}$ for a model $\mathcal{MV}$ is the set of all refinements $Ref_{\mathcal{MV}}^{\phi} = \{\mathcal{MV}' \mid \mathcal{MV}' \lhd^{\phi} \mathcal{MV}\}$ under mapping vector ϕ.*

In practice, the derivation of this refinement set becomes intractable for all but the smallest of MVNs. Specifically, if we have k entities each with n states, then we have a worst case upper bound of $(n^{n^k})^k$ possible candidate models to consider. For instance, there are $(2^{2^3})^3 = 16777216$ possible BNs consisting of just three entities! An interesting observation arises, however, by noting that for some model $\mathcal{MV}$ and mapping vector

ϕ, the trace semantics of the translated $\mathcal{MV}$ (i.e. $TS(\phi(\mathcal{MV}))$) is not the same as translating the trace semantics of $\mathcal{MV}$ (i.e. $\phi(TS(\mathcal{MV}))$). In fact, it turns out that an important relationship emerges between the two, in that $TS(\phi(\mathcal{MV}))$ will always contain at least the traces of $\phi(TS(\mathcal{MV}))$, as shown by the following theorem.

Theorem 5.2. *For a model $\mathcal{MV}$ and a mapping vector ϕ, we have that $\phi(TS(\mathcal{MV})) \subseteq TS(\phi(\mathcal{MV}))$.*

Proof. Let $\sigma \in TS(\mathcal{MV})$ be an arbitrary trace in model $\mathcal{MV}$ with k entities, and $\sigma_m \rightarrow \sigma_{m+1}$ be an arbitrary state step in σ. This results in a state step $\phi(\sigma_m) \rightarrow \phi(\sigma_{m+1})$ under ϕ and we show that this step is present in $TS(\phi(\mathcal{MV}))$. Let $j \in \{1, \ldots, k\}$, then the step can be broken up into k components $\sigma_m \rightarrow \sigma^j_{m+1}$, where j represents the j-th component of the state. Now $\phi(\sigma_m) \rightarrow \phi(\sigma^j_{m+1})$ under ϕ and this is equivalent to a rule in $\phi(\mathcal{MV})$. Thus, $\phi(TS(\mathcal{MV})) \subseteq TS(\phi(\mathcal{MV}))$ as required. □

From this result, it is clear that the trace semantics of any refinement $\mathcal{MV}'$ must also be contained within $TS(\phi(\mathcal{MV}))$.

Corollary 5.1. $\mathcal{MV}' \lhd^\phi \mathcal{MV} \implies TS(\mathcal{MV}') \subseteq TS(\phi(\mathcal{MV}))$.

Proof. Follows directly from Definition 5.8 and Theorem 5.2. □

Corollary 5.1 represents an important and useful relationship, since we know that the trace semantics of any refinements must be contained within $TS(\phi(\mathcal{MV}))$, resulting in a significantly smaller search space. Indeed, this relationship is exploited by our algorithms in Section 5.3.

In special cases, a refinement $\mathcal{MV}'$ may capture *all* the behaviour of an MVN under a mapping vector ϕ. We distinguish this strong case with the notion of an *exact refinement.*

Definition 5.10 (Exact refinement)**.** *A refinement $\mathcal{MV}'$ exactly refines $\mathcal{MV}$ under mapping vector ϕ, written $\mathcal{MV}' \trianglelefteq^\phi \mathcal{MV}$, iff $TS(\mathcal{MV}') = \phi(TS(\mathcal{MV}))$.*

Exact refinements are interesting as they can indicate redundant entity thresholds which have no affect on the qualitative behaviour of an MVN. Subsequently, an exact refinement $\mathcal{MV}'$ provides a simpler representation of an MVN whilst capturing all its behaviour under ϕ. Note exact refinements occur precisely when $\phi(\mathcal{MV})$ is deterministic, as shown by the following theorem.

Theorem 5.3. *Let $\mathcal{MV}$ be a model and ϕ be some mapping vector. Then we have the following:*

(1) if $\phi(\mathcal{MV})$ is deterministic, then $\phi(\mathcal{MV})$ is an exact refinement;

(2) if $\phi(\mathcal{MV})$ is non–deterministic, then no exact refinement can exist under ϕ.

Proof. To prove (1), we observe that if $\phi(\mathcal{MV})$ is deterministic, then $TS(\phi(\mathcal{MV}))$ must also be deterministic. As such, there must exist exactly one trace $\sigma(S) \in TS(\phi(\mathcal{MV}))$ for each state $S \in S_{\mathcal{MV}}$. From Theorem 5.2, we have $\phi(TS(\mathcal{MV})) \subseteq TS(\phi(\mathcal{MV}))$.

Thus, $\phi(TS(\mathcal{MV})) = TS(\phi(\mathcal{MV}))$ must hold in this case. Therefore, only one possible deterministic model can exist which captures this behaviour completely, and so from Definition 5.10, we can see that it must be an exact refinement $\mathcal{MV}'$ such that $TS(\mathcal{MV}') = \phi(TS(\mathcal{MV}))$.

To prove (2), we simply note that if $\phi(\mathcal{MV})$ is non–deterministic, then no single deterministic model can exist which captures this behaviour. □

Note for an exact refinement $\mathcal{MV}'$ of a model $\mathcal{MV}$, we have that $\mathcal{MV}' = \phi(\mathcal{MV})$, and so the refinement set $Ref^{\phi}_{\mathcal{MV}}$ will be the singleton set $Ref^{\phi}_{\mathcal{MV}} = \{\phi(\mathcal{MV})\}$. However, the converse is not necessarily true; if we have a refinement set $Ref^{\phi}_{\mathcal{MV}} = \{\mathcal{MV}'\}$ for some model $\mathcal{MV}'$, then it does not imply that $\mathcal{MV}'$ is exact. With this proposed refinement theory, we now go on to investigate its application.

5.2 Investigative application

In this section, we apply our approach to a small example to demonstrate its methodology. This walkthrough not only clarifies the ideas presented, but allows us to show an interesting result which motivates the need for multi-valued modelling. We then explore some of the limitations of analysing refinements with respect to the original model. Finally, we compare our results with existing work which considers the differences between BNs and continuous systems of differential equations [14, 77, 142] (see [188] for a good review), and we show that the same relationships do not hold between BNs and MVNs in the discrete modelling domain.

We start by revisiting model $\mathcal{MV}_1$ from Figure 5.1(a), which consists of two entities g_1 and g_2, such that $Y(g_1) = \{0, 1\}$ and $Y(g_2) = \{0, 1, 2\}$. Here, we will refine g_2 to a Boolean entity as defined by Definition 5.8. From Section 5.1.2, we know that the mapping set $MS^{3\to 2}_{g_2}$ for g_2 consists of three structurally unique state mappings, and so we have the following three mapping vectors to consider:

$$\begin{aligned} (1)\ \phi &= \langle \phi_{g_1}, \phi_{g_2} \rangle, \\ (2)\ \phi &= \langle \phi_{g_1}, \phi'_{g_2} \rangle, \\ (3)\ \phi &= \langle \phi_{g_1}, \phi''_{g_2} \rangle, \end{aligned}$$

where $\phi_{g_2} = \{0 \mapsto 0, 1 \mapsto 0, 2 \mapsto 1\}$, $\phi'_{g_2} = \{0 \mapsto 0, 1 \mapsto 1, 2 \mapsto 1\}$, $\phi''_{g_2} = \{0 \mapsto 0, 1 \mapsto 1, 2 \mapsto 0\}$ and ϕ_{g_1} is the identity mapping. For the purpose of this example, we will refine $\mathcal{MV}_1$ under mapping vector (1) as a refinement is known to exist. We start by applying (1) to the trace semantics $TS(\mathcal{MV}_1)$, which results in the following non–deterministic trace semantics $\phi(TS(\mathcal{MV}_1))$ in which only entity g_2 has been translated:

$$\begin{aligned} \sigma(00) &= \langle 00, 10, 10 \rangle \\ \sigma(00) &= \langle 00, 11, 00 \rangle \\ \sigma(01) &= \langle 01, 01 \rangle \\ \sigma(10) &= \langle 10, 10 \rangle \\ \sigma(11) &= \langle 11, 00, 11 \rangle \end{aligned}$$

In order to calculate the refinement set $Ref^{\phi}_{\mathcal{MV}_1}$, we search for all deterministic models whose trace semantics is contained within $\phi(TS(\mathcal{MV}_1))$. Using the results from Corollary 5.1, we therefore translate the state transition tables $\mathcal{MV}_1$ using mapping vector (1), and this gives us the non–deterministic state transition tables shown previously in Figure 5.2. We can see that the behaviour of g_2 is non-deterministic when $g_1 = 0$ and $g_2 = 0$. As such, we have two possible candidate models $\mathcal{MV}'_1$ and $\mathcal{MV}'_2$ shown respectively by Figure 5.3(a) and Figure 5.3(b), where the rules highlighted in bold are the only that differ.

g_2	$[g_1]$
0	1
1	0

g_1	g_2	$[g_2]$
0	**0**	**0**
0	1	1
1	0	0
1	1	0

(a)

g_2	$[g_1]$
0	1
1	0

g_1	g_2	$[g_2]$
0	**0**	**1**
0	1	1
1	0	0
1	1	0

(b)

Figure 5.3: (a) Candidate model $\mathcal{MV}'_1$, and (b) candidate model $\mathcal{MV}'_2$.

In order to verify that $\mathcal{MV}'_1$ and $\mathcal{MV}'_2$ are indeed refinements according to our theory, we check that their trace semantics are contained within $\phi(TS(\mathcal{MV}_1))$. The corresponding trace semantics for $\mathcal{MV}'_1$ and $\mathcal{MV}'_2$ are shown in Figure 5.2(a) and Figure 5.2(b), respectively. Now we can observe that $\mathcal{MV}'_1$ is not a refinement according to Definition

$$\begin{aligned}
\sigma(00) &= \langle 00, 10, 10\rangle \\
\sigma(01) &= \langle 01, 00, 10, 10\rangle \\
\sigma(10) &= \langle 10, 10\rangle \\
\sigma(11) &= \langle 11, 00, 10, 10\rangle
\end{aligned}$$

(a)

$$\begin{aligned}
\sigma(00) &= \langle 00, 11, 00\rangle \\
\sigma(01) &= \langle 01, 01\rangle \\
\sigma(10) &= \langle 10, 10\rangle \\
\sigma(11) &= \langle 11, 00, 11\rangle
\end{aligned}$$

(b)

Figure 5.4: (a) Trace semantics for $\mathcal{MV}'_1$, and (b) trace semantics for $\mathcal{MV}'_2$.

5.8, since $TS(\mathcal{MV}'_1) \not\subseteq \phi(TS(\mathcal{MV}_1))$; in other words, its behaviour is not regarded as being consistent with $\mathcal{MV}_1$. On the other hand, we find that $\mathcal{MV}'_2$ is a refinement as $TS(\mathcal{MV}'_2) \subseteq \phi(TS(\mathcal{MV}_1))$. Thus, we have the refinement set $Ref^{\phi}_{\mathcal{MV}_1} = \{\mathcal{MV}'_2\}$ for mapping vector (1).

In some cases, however, there may exist regulatory interactions in an MVN which are too subtle to be represented in the Boolean domain using our theory. This comparison between Boolean and multi-valued modelling is interesting to consider, and the following theorem applies our approach to show an important result motivating the latter.

Theorem 5.4. *Not every MVN has a Boolean representation.*

Proof. Suppose that we consider the addition of a third entity g_3 to model $\mathcal{MV}_1$ which is repressed by g_2 when $g_2 \geq 1$. This new MVN is shown in Figure 5.5 and will be denoted

g_2	$[g_1]$
0	1
1	1
2	0

g_1	g_2	$[g_2]$
0	0	1
0	1	2
0	2	2
1	0	0
1	1	0
1	2	1

g_2	$[g_3]$
0	1
1	0
2	0

Figure 5.5: Model $\mathcal{MV}_3$ with a third entity g_3, where $Y(g_3) = \{0, 1\}$.

$\mathcal{MV}_3$. We now see that $g_2 \in \mathcal{MV}_3$ acts in two subtly different ways: on one hand g_1 is inhibited when $g_2 = 2$; and on the other hand, g_3 is inhibited when $g_2 \geq 1$. This subtle behaviour can prove problematic for Boolean modelling according to our theory. For each possible ϕ, we: (i) enumerate each possible deterministic model $\mathcal{MV}'$ from $\phi(\mathcal{MV}_3)$; and (ii) check it for consistency $TS(\mathcal{MV}') \subseteq \phi(TS(\mathcal{MV}_3))$. Under all three possible vectors, we find that no refinement $\mathcal{MV}'$ exists such that $\mathcal{MV}' \lhd^{\phi} \mathcal{MV}_3$, thus concluding the proof. □

Of course, this result is centered around the relationship assumption formalised by our refinement theory, but it nevertheless represents an important application in reasoning about the expressive power of different models. In particular, it provides motivation for multi-valued modelling and the framework proposed in Chapter 4.

Let us now look more closely at the behaviour captured by an MVN and its refinements; in particular, the scope and limitations of the latter. For this, we first define some additional notations, starting with the notion of *corresponding states.*

Definition 5.11 (Corresponding state). *Let $S' \in S_{\mathcal{MV}'}$ be some global state of refinement $\mathcal{MV}'$, and $S \in S_{\mathcal{MV}}$ be a global state of the original model $\mathcal{MV}$, such that $\mathcal{MV}' \lhd^{\phi} \mathcal{MV}$ for mapping vector ϕ. Then we say that S' and S correspond with respect to ϕ iff $S' = \phi(S)$.*

In other words, there is a many-to-one relationship between a global state in an MVN and a global state in its refinements, since different states in the former can be merged into the same state in the latter. This then leads to the notion of a *corresponding trace.*

Definition 5.12 (Corresponding trace). *Let $\sigma(S) \in TS(\mathcal{MV})$ and $\sigma(S') \in TS(\mathcal{MV}')$ be the respective traces of models $\mathcal{MV}$ and $\mathcal{MV}'$, where $\mathcal{MV}' \lhd^{\phi} \mathcal{MV}$. Then we say that $\sigma(S)$ and $\sigma(S')$ correspond with respect to ϕ iff $\phi(\sigma(S)) = \sigma(S')$.*

Despite this, corresponding traces can be quite different due to the many-to-one relationship between analogous states. For example, suppose we have the following trace for some model with two entities g_a and g_b:

$$\sigma(00) = \langle 00, 11, 02, 12, 01, 11 \rangle$$

which has a cycle of period four. Now suppose we apply the mapping vector $\phi = \langle \phi_{g_a}, \phi_{g_b} \rangle$, where ϕ_{g_a} is the identity mapping and $\phi_{g_b} = \{0 \mapsto 0, 1 \mapsto 1, 2 \mapsto 1\}$. This results in the following trace:

$$\phi(\sigma(00)) = \langle 00, 11, 01, 11 \rangle$$

which now contains a simpler cycle of period two analogous to the original. One can quite easily see that it is also possible for a cycle of states in the original trace to be merged into a single steady state in the translated trace, which although corresponding according to our theory, represents quite different dynamics. Note it is also easy to see that since global states of a refinement are expanded in the more complex MVN, then a steady state of the former must correspond to either steady or cyclic states in the latter, and cyclic states of the former must correspond to cyclic states of the latter.

This therefore highlights the scope and some important limitations with the behaviour captured by a refinement. Despite this, reachability properties of a refinement will correspond to reachability properties in the more complex MVN. For this, we first introduce the notation $S_1 \xrightarrow{\mathcal{MV}} S_2$ to express the fact that global state $S_2 \in S_{\mathcal{MV}}$ is reachable from global state $S_1 \in S_{\mathcal{MV}}$ for some model $\mathcal{MV}$.

Theorem 5.5. *Let $\mathcal{MV}' \lhd^{\phi} \mathcal{MV}$ for some mapping vector ϕ. If $S_1' \xrightarrow{\mathcal{MV}'} S_2'$ then there exists two corresponding states $S_1 \in S_{\mathcal{MV}}$ and $S_2 \in S_{\mathcal{MV}}$ such that $S_1 \xrightarrow{\mathcal{MV}} S_2$.*

Proof. Since $S_1' \xrightarrow{\mathcal{MV}'} S_2'$, then there must exist a trace $\sigma(S_1') \in TS(\mathcal{MV}')$ containing S_2'. From Definition 5.8, we know that $TS(\mathcal{MV}') \subseteq \phi(TS(\mathcal{MV}))$ must hold. There must therefore exist a corresponding trace $\sigma(S_1) \in TS(\mathcal{MV})$ containing S_2, such that $\phi(\sigma(S_1)) = \sigma(S_1')$. Thus, $S_1 \xrightarrow{\mathcal{MV}'} S_2$ must hold. □

In other words, reachability properties of refinements have corresponding reachability properties in the more complex MVN. However, since refinements normally capture less behaviour than the original model, there are further limits on what can be deduced from the refinement, as shown by the following corollary.

Corollary 5.2. *Reachability properties of a model $\mathcal{MV}$ are deducible in exact refinements and semi-deducible otherwise.*

Proof. Let $S_1' \xrightarrow{\mathcal{MV}'} S_2'$ be some reachability property which holds in refinement $\mathcal{MV}'$ of model $\mathcal{MV}$. From Theorem 5.5, we therefore know that there exists a corresponding reachability property in $\mathcal{MV}$. However, suppose $S_1' \xrightarrow{\mathcal{MV}'} S_2'$ does not hold. If $\mathcal{MV}' \trianglelefteq^{\phi} \mathcal{MV}$, then from Definition 5.10 we have that $TS(\mathcal{MV}') = \phi(TS(\mathcal{MV}))$, and so we can deduce that the property also does not hold in $\mathcal{MV}$. Hence, we say that reachability is a deducible property for exact refinements. For non-exact refinements, however, we have that $TS(\mathcal{MV}') \subset \phi(TS(\mathcal{MV}))$, and so we cannot deduce whether or not this property holds in $\mathcal{MV}$ since some important traces may be missing. Hence, we can only verify reachability properties that hold in a non-exact refinement, and so we say that reachability is semi-deducible. □

In contrast, existing studies have tended to focus on comparisons between logical and differential descriptions [14, 77, 142, 206], and these have identified different results to the ones discussed here. For example, it is known that a steady state in a BN will correspond, qualitatively, to an analogous steady state in the differential representation, and that a steady state in the differential representation will not always correspond to

a steady state in the BN. However, whilst the latter is consistent with our view that refinements normally capture less behaviour than the more complex MVN, the former is not. As discussed above, if we have a steady state in a refinement, such as a BN, then it may either correspond to a steady state in the more complex MVN, or to a cycle of states according to our theory.

Furthermore, it is known that cyclic behaviour in a BN will not always correspond to cyclic behaviour in the differential description. The example given by Glass *et al.* [77] was that of a simple feedback loop where entity g_i activates g_j whilst g_j represses g_i. In the BN representation, this feedback loop yields a four-state cycle, whilst in the differential description it falls into a single steady state. However, from Definition 5.12 and the example that immediately follows, we have observed that cycles in the BN must always correspond to cycles in the more complex MVN according to our theory, although in some cases the cycles in the latter may have a larger period.

This has therefore formed an interesting study and application area for our refinement theory. We have investigated the scope and limitations of analysing more abstract MVN models, and have shown a result which identifies a situation in which multi-valued modelling is required. Although this initial investigation is reliant on our relationship assumption, it nevertheless begins to shed light on some of the questions raised from the previous two chapters regarding the level of detail required to model certain dynamics. Further investigations are left as an interesting area of future work.

5.3 Algorithm development

With our refinement theory introduced, we now consider how systematic approaches can be used to make it practical. In particular, we develop algorithms for both calculating the refinement set and for verifying whether one MVN is a refinement of another according to Definition 5.8.

5.3.1 Calculating the refinement set

We start by considering an exhaustive algorithm ExhaustiveRefinement, which automates the methodological approach presented in Section 5.2 by simply enumerating and checking all derivable deterministic models. Whilst this method identifies all refinements, its complexity is determined by an exponential number of possible models, and it is naïve to termination properties.

To address these shortcomings, we develop a more efficient tree-based algorithm TreeRefinement which computes the refinement set using logical deduction. This improved algorithm is shown to depend on the size of the refinement set which is in most cases significantly smaller than the number of possible models. Furthermore, it can detect termination properties early to avoid unnecessary computation when no refinements exist. We also outline further improvements by filtering the translated trace semantics prior to tree construction in a separate algorithm TreeRefinementPF.

Although TreeRefinementPF enjoys significant speed-ups over ExhaustiveRefinement, it can be hindered when the refinement set is large. A parallelised extension

PARALLELTREEREFINEMENT is therefore proposed which enables the refinement set to be computed concurrently across multiple processing nodes, making it practical and scalable to large MVNs.

An exhaustive approach

The EXHAUSTIVEREFINEMENT algorithm (see Algorithm 1) takes as input a model $\mathcal{MV}$ and mapping vector ϕ, and calculates the refinement set $Ref^{\phi}_{\mathcal{MV}}$ by exhaustively searching for all possible deterministic models $\mathcal{MV}'$ such that $\mathcal{MV}' \lhd^{\phi} \mathcal{MV}$. In particular, it makes use of Corollary 5.1 to significantly reduce the search space for such models. This is achieved by enumerating each possible deterministic model $\mathcal{MV}'$ derivable from the state transition tables $\phi(\mathcal{MV})$ (see the example in Section 5.2 for how this is done) and by checking that $TS(\mathcal{MV}') \subseteq \phi(TS(\mathcal{MV}))$ holds.

Algorithm 1 EXHAUSTIVEREFINEMENT: calculates $Ref^{\phi}_{\mathcal{MV}}$ from model $\mathcal{MV}$ and mapping vector ϕ.

Inputs: Model $\mathcal{MV}$, Mapping vector ϕ
Outputs: Set of refinements $Ref^{\phi}_{\mathcal{MV}}$

1: $Ref^{\phi}_{\mathcal{MV}} \longleftarrow \emptyset$
2: **for all** deterministic models $\mathcal{MV}'$ derivable from $\phi(\mathcal{MV})$ **do**
3: **if** $TS(\mathcal{MV}') \subseteq \phi(TS(\mathcal{MV}))$ **then**
4: $Ref^{\phi}_{\mathcal{MV}} \longleftarrow Ref^{\phi}_{\mathcal{MV}} \cup \{\mathcal{MV}'\}$
5: **end if**
6: **end for**
7: **return** $Ref^{\phi}_{\mathcal{MV}}$

Note an example application of this algorithm is provided in Section 5.2. If we let $Poss_{\mathcal{MV}}$ denote the total number of possible deterministic models $\mathcal{MV}'$ derivable from $\phi(\mathcal{MV})$, then the worst case complexity of this algorithm is $O(Poss_{\mathcal{MV}} \cdot |TS(\mathcal{MV}')|)$ if we use a suitable data structure to represent the trace semantics, e.g. a hash table.

A tree-based approach

The EXHAUSTIVEREFINEMENT algorithm exhaustively enumerates all possible deterministic models $\mathcal{MV}'$ from $\phi(\mathcal{MV})$ and checks that their trace semantics $TS(\mathcal{MV}')$ is contained in $\phi(TS(\mathcal{MV}))$. As such, EXHAUSTIVEREFINEMENT can struggle with large MVNs, and is naïve to termination properties which indicate when no refinements exist.

Here, we propose an improved algorithm TREEREFINEMENT which works with the individual rules of $\phi(TS(\mathcal{MV}))$. Whereas EXHAUSTIVEREFINEMENT blindly enumerates all possible deterministic models from $\phi(\mathcal{MV})$, TREEREFINEMENT searches for deterministic models directly from $\phi(TS(\mathcal{MV}))$, resulting in an exponentially smaller search space (this can be easily observed from the results of Theorem 5.2). Furthermore, since such deterministic models inherently obey the translated trace semantics, they must therefore be refinements by Definition 5.8.

The core of TREEREFINEMENT is the construction of a tree of rules from $\phi(TS(\mathcal{MV}))$ using a recursive percolate-up approach, where: (i) the root node contains rules common

to all refinements; (ii) each node contains rules which extend its parent node (i.e. they are disjoint); and (iii) the set union of all nodes on the path from the root to any leaf in a completed tree gives us the deterministic rules for a single refinement. Thus, one advantage of this structure is that we obtain a memory efficient representation of the refinement set $Ref^{\phi}_{\mathcal{MV}}$. Moreover, it facilitates the efficient detection of *rule contradictions* which enable early termination of the algorithm.

Definition 5.13 (Rule contradiction). *Let $r = (g_i, S \rightarrow S')$ be a rule for entity g_i, and let Node be a node in the tree. We say that r contradicts Node if there exists a rule $r' \in Node$ such that $r' = (g_i, S \rightarrow S'')$, where $S' \neq S''$.*

Rule contradictions indicate the presence of non–determinism which should be avoided, since refinements must be deterministic. That is, if we traverse the tree from some leaf node to the root with a rule, and find that the rule contradicts an existing node, then we can deduce that the rule is not part of that particular refinement. More powerful than that, however, is the fact that we can now also ignore any traces containing this rule with respect to the refinement, thus providing a means of significantly pruning the search space.

Example application

To illustrate the logic behind TREEREFINEMENT before providing a more formal definition, let us revisit model $\mathcal{MV}_1$ from Figure 5.1. As with the example given in Section 5.2, we will perform the refinement use mapping vector (1):

$$\phi = \langle \phi_{g_1}, \phi_{g_2} \rangle ,$$

where $\phi_{g_2} = \{0 \mapsto 0, 1 \mapsto 0, 2 \mapsto 1\}$. Based on this, we obtain the following translated trace semantics:

$$\begin{aligned} \sigma(00) &= \langle 00, 10, 10 \rangle \\ \sigma(00) &= \langle 00, 11, 00 \rangle \\ \sigma(01) &= \langle 01, 01 \rangle \\ \sigma(10) &= \langle 10, 10 \rangle \\ \sigma(11) &= \langle 11, 00, 11 \rangle \end{aligned}$$

We start by constructing the root node of the tree, denoted *Root*, which consists of the rules which are common to all refinements. Essentially, these are all the non-contradictory rules derivable from the translated trace semantics above, and are shown in Figure 5.6(a).

We then take a systematic approach to building the tree by considering all traces which start from a particular global state S, denoted by the set $AllTraces_S$. For example, if we take $S = 00$, then we have that $AllTraces_{00} = \{\langle 00, 10, 10 \rangle , \langle 00, 11, 00 \rangle\}$. The rules for both traces are shown below:

$$\{(g_1, 0 \rightarrow 1), (g_2, 00 \rightarrow 0), (g_2, 10 \rightarrow 0)\}, \quad \{(g_1, 0 \rightarrow 1), (g_2, 00 \rightarrow 1), (g_2, 11 \rightarrow 0)\}.$$

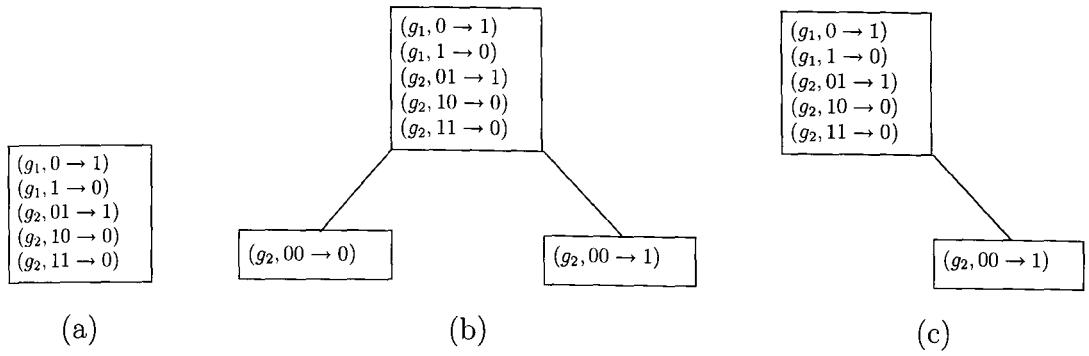


Figure 5.6: (a) Non-contradictory rules of $\phi(\mathcal{MV}_1)$ forming root node *Root*, (b) tree after all traces starting from global state 00 have been checked against *Root*, and (c) final tree after all global states $S \in \phi(S_{\mathcal{MV}_1})$ have been considered.

Taking each trace of $AllTraces_{00}$ in turn, we then simply check that the corresponding rules do not contradict the rules of *Root* shown in Figure 5.6(a). For the first set of rules, we can see that $(g_1, 0 \rightarrow 1)$ and $(g_2, 10 \rightarrow 0)$ already exist in *Root*, whilst $(g_2, 00 \rightarrow 0)$ neither exists nor contradicts. Similarly, for the second set of rules, we can see that $(g_1, 0 \rightarrow 1)$ and $(g_2, 11 \rightarrow 0)$ already exist in *Root*, whilst $(g_2, 00 \rightarrow 0)$ neither exists nor contradicts. As such, both traces are acceptable and the rules which are not already present in *Root* are used to create two new leaf nodes, which we show in Figure 5.6(b).

We then consider global state $S = 01$, where $AllTraces_{01} = \{\langle 01, 01\rangle\}$. The rules for this trace are shown below:

$$\{(g_1, 1 \rightarrow 0), (g_2, 01 \rightarrow 1)\},$$

and we can see that neither rule contradicts the two leaf nodes; in fact, they are already present in *Root*. We therefore move to the next global state $S = 10$, where $AllTraces_{10} = \{\langle 10, 10\rangle\}$. The corresponding rules are:

$$\{(g_1, 0 \rightarrow 1), (g_2, 10 \rightarrow 0)\},$$

and we note again that both are already present in *Root*. Finally, we consider the remaining global state $S = 11$, where $AllTraces_{11} = \{\langle 11, 00, 11\rangle\}$. The rules for this single trace are shown below:

$$\{(g_1, 1 \rightarrow 0), (g_1, 0 \rightarrow 1), (g_2, 11 \rightarrow 0), (g_2, 00 \rightarrow 1)\}.$$

When we compare these rules against the left leaf node, we notice a contradiction; rule $(g_2, 00 \rightarrow 1)$ of the trace contradicts $(g_2, 00 \rightarrow 0)$ (see Definition 5.13). Since these are the rules of the only trace starting from global state 11, we can remove the left leaf node as it cannot represent a valid refinement. In contrast, there are no contradictions with the right leaf node, and so we terminate the algorithm with the tree shown in Figure 5.6(c). Note the set union of these rules gives us the same refinement as shown in Figure 5.3(b) which was derived by ExhaustiveRefinement.

We can now introduce this refinement process more formally. In order to check whether some rule r can be part of a refinement, we use the recursive algorithm RULECONSISTENT (see Algorithm 2), which takes r and the corresponding leaf node $Leaf$ and checks for a contradiction. If a contradiction is found (i.e. the addition of r to $Leaf$ results in non–determinism), then RULECONSISTENT returns $false$. However, if r is already contained in $Leaf$ then RULECONSISTENT returns $true$ since we know at this stage that r has been previously accepted. Otherwise, we recurse to the parent of node $Leaf$ and perform the same checks up to and including the root node (in which case we return $true$ to indicate the absence of a contradiction).

Algorithm 2 RULECONSISTENT: recursively checks whether a rule r can be part of a refinement.

Inputs: Rule r, Node $Leaf$
Outputs: Boolean

```
1: if r contradicts Leaf then
2:    return false
3: else if r ∈ Leaf or Leaf = Root then
4:    return true
5: else
6:    return RULECONSISTENTFROMNODE(r, PARENT(Leaf))
7: end if
```

Thus, we can check whether a trace $\sigma \in \phi(TS(\mathcal{MV}))$ contradicts a leaf node $Leaf$ by applying RULECONSISTENT to each rule r of trace σ, as shown by TRACECONSISTENT in Algorithm 3. Note in order to ensure that trace σ is deterministic (i.e. valid and hence part of any potential refinements), we record the rules from previous iterations over σ in the set $AcceptedRules$, and check first whether the current rule r contradicts $AcceptedRules$ before traversing the tree. If any single rule r contradicts $AcceptedRules$ or any node on the path from the leaf node $Leaf$ to the root $Root$, then TRACECONSISTENT returns $false$, otherwise it returns $true$.

With this, TREEREFINEMENT (see Algorithm 4) computes the full tree representing the refinement set $Ref^{\phi}_{\mathcal{MV}}$ from $\phi(TS(\mathcal{MV}))$ by considering each set of traces $AllTraces_S$ in turn, such that $AllTraces_S = \{\sigma(S) \in \phi(TS(\mathcal{MV})) \mid S \in \phi(S_{\mathcal{MV}})\}$. TREEREFINEMENT checks the rules of each trace $\sigma \in AllTraces_S$ against each leaf node $Leaf \in LeafNodes$ using TRACECONSISTENT. If TRACECONSISTENT returns $true$ for some leaf node $Leaf$, then the corresponding trace σ is stored in the set $AcceptedTraces$ to indicate that its rules are non-contradictory with respect to $Leaf$. Note initially $LeafNodes = \{Root\}$, where $Root$ contains all non-contradictory rules of $\phi(TS(\mathcal{MV}))$ (in other words, rules that must be part of all potential refinements).

Once all traces in $AllTraces_S$, for some $S \in \phi(S_{\mathcal{MV}})$, have been checked against some leaf node $Leaf$, two cases are considered:

(1) $|AcceptedTraces| = 1$: The rules of the single trace $\sigma \in AcceptedTraces$ are added to the leaf node $Leaf$, and $Leaf$ is then added to the set $NewGeneration$ which stores the next generation of leaf nodes after an iteration using $AllTraces_S$;

Algorithm 3 TRACECONSISTENT: checks to see whether trace σ contains rules for a refinement.

Inputs: Trace σ, Node $Leaf$
Outputs: Boolean

```
1: AcceptedRules ⟵ ∅
2: for all rules r of trace σ do
3:   if r contradicts AcceptedRules or RULECONSISTENT(r, Leaf) = false then
4:     return false
5:   else
6:     AcceptedRules ⟵ Accepted ∪ {r}
7:   end if
8: end for
9: return true
```

(2) $|AcceptedTraces| = x$ for $x \geq 2$: We create x new leaf nodes $NewLeaf$ which extend the parent $Leaf$ with the corresponding rules of $\sigma \in AcceptedTraces$ that are not already present in the branch, and add these new leaf nodes to $NewGeneration$.

Once all leaf nodes have been covered by the traces in $AllTraces_S$, for some $S \in \phi(S_{\mathcal{MV}})$, we will have $|NewGeneration| \geq 0$. In the case that $|NewGeneration| = 0$, we terminate TREEREFINEMENT since determinism has been violated, making it impossible for refinements to exist. Otherwise, we consider the next $S \in \phi(\mathcal{MV})$ and compare all traces starting from S against the new generation of leaf nodes until all $S \in \phi(\mathcal{MV})$ have been covered.

As each iteration of TREEREFINEMENT compares the set of traces $AllTraces_S$ for some $S \in \phi(S_{\mathcal{MV}})$ against each leaf node $Leaf$, its performance depends on $|Ref^{\phi}_{\mathcal{MV}}|$; in contrast, the performance of EXHAUSTIVEREFINEMENT is dependent on the number of all possible deterministic models $Poss_{\mathcal{MV}}$ derivable from $\phi(\mathcal{MV})$, i.e. $O(Poss_{\mathcal{MV}} \cdot |TS(\mathcal{MV}')|)$, where $|Ref^{\phi}_{\mathcal{MV}}| << Poss_{\mathcal{MV}}$. Thus, in the worst case, the performance of TREEREFINEMENT is

$$O(|Ref^{\phi}_{\mathcal{MV}}| \cdot |\phi(TS(\mathcal{MV}))|),$$

since we must consider each trace in the translated trace semantics against each leaf node. However, the average and best case performance of TREEREFINEMENT is significantly better than that of EXHAUSTIVEREFINEMENT due to its ability to detect properties for terminating early and for discarding traces which can not be part of refinements.

A tree-based approach with pre-filtering

In this section, we present an optimisation to the TREEREFINEMENT algorithm called TREEREFINEMENTPF (pre-filtering) based around the fact that $\phi(TS(\mathcal{MV}))$ can be filtered prior to tree construction. Note we present the filtering algorithm only, and refer the reader to TREEREFINEMENT discussed previously for details of the tree building process once filtering is complete.

The rationale behind this optimisation is based around the observation that $Root$ contains all non-contradictory rules of $\phi(\mathcal{MV})$ (rules which must be part of all potential

Algorithm 4 TREEREFINEMENT: calculates $Ref^{\phi}_{\mathcal{MV}}$ by constructing a tree of rules using auxiliary algorithm TRACECONSISTENT.

Inputs: Node *Root*, Model $\mathcal{MV}$, Mapping vector ϕ
Outputs: Set of refinements $Ref^{\phi}_{\mathcal{MV}}$

$Ref^{\phi}_{\mathcal{MV}} \longleftarrow \emptyset$
$LeafNodes \longleftarrow \{Root\}$
for all $S \in \phi(S_{\mathcal{MV}})$ **do**
  $NewGeneration \longleftarrow \emptyset$
  $AllTraces_S \longleftarrow \{\sigma(S) \in \phi(TS(\mathcal{MV})) \mid S \in \phi(S_{\mathcal{MV}})\}$
  for all leaf nodes $Leaf \in LeafNodes$ **do**
    $AcceptedTraces \longleftarrow \emptyset$
    for all $\sigma \in AllTraces_S$ **do**
      if TRACECONSISTENT$(\sigma, Leaf) = true$ **then**
        $AcceptedTraces \longleftarrow AcceptedTraces \cup \{\sigma\}$
      end if
    end for
    if $|AcceptedTraces| = 1$ **then**
      $\sigma \in AcceptedTraces$
      Add rules of σ to $Leaf$
      $NewGeneration \longleftarrow NewGeneration \cup \{Leaf\}$
    else if $|AcceptedTraces| \geq 2$ **then**
      for all $\sigma \in AcceptedTraces$ **do**
        Create new leaf node $NewLeaf$ to extend $Leaf$
        Add rules of σ not already in branch to $NewLeaf$
        $NewGeneration \longleftarrow NewGeneration \cup \{NewLeaf\}$
      end for
    end if
  end for
  if $NewGeneration = \emptyset$ **then**
    return $\emptyset$
  else
    $LeafNodes \longleftarrow NewGeneration$
  end if
end for
return $Ref^{\phi}_{\mathcal{MV}}$ from tree representation

refinements). As such, the traces of any refinements must have rules which do not contradict those of *Root*. The idea is therefore to check the set of traces $AllTraces_S$, for each $S \in \phi(S_{\mathcal{MV}})$, against *Root* before tree construction begins, and to remove any traces where a contradiction is found. This process is performed by FILTERTRACES as shown by Algorithm 5.

Algorithm 5 FILTERTRACES: filters $\phi(TS(\mathcal{MV}))$ against the rules of *Root*.

Inputs: Node *Root*, Model $\mathcal{MV}$, Mapping vector ϕ
Outputs: Boolean

1: **for all** $S \in \phi(S_{\mathcal{MV}})$ **do**
2: $AllTraces_S \longleftarrow \{\sigma(S) \in \phi(TS(\mathcal{MV})) \mid S \in \phi(S_{\mathcal{MV}})\}$
3: **for all** $\sigma \in AllTraces_S$ **do**
4: **if** TRACECONSISTENT$(\sigma, Root) = false$ **then**
5: $AllTraces_S \longleftarrow AllTraces_S - \{\sigma\}$
6: **end if**
7: **end for**
8: **if** $|AllTraces_S| = 0$ **then**
9: **return** *false*
10: **else if** $|AllTraces_S| = 1$ **then**
11: $\sigma \in AllTraces_S$
12: $Root \longleftarrow Root \cup$ GETRULES(σ)
13: $AllTraces_S \longleftarrow AllTraces_S - \{\sigma\}$
14: **end if**
15: **end for**
16: **return** *true*

Once all traces $\sigma \in AllTraces_S$ have been compared against *Root*, we will have one of three possible situations:

- $|AllTraces_S| = 0$: We can terminate here as all traces for some $S \in \phi(S_{\mathcal{MV}})$ have been removed, making it impossible for any deterministic model to exist;
- $|AllTraces_S| = 1$: We add the rules of $\sigma \in AllTraces_S$ to *Root*, since they are also non-contradictory, and then we remove σ. Not only does this reduce the remaining traces to consider when building a tree, but it also tightens the criteria that other traces must respect;
- $|AllTraces_S| \geq 2$: Such traces contain non-contradictory rules with *Root*, but can be contradictory with each other. We therefore leave such traces in $\phi(TS(\mathcal{MV}))$ so that they can be revisited during tree construction and separated into different models.

Once FILTERTRACES has completed, *Root* will contain all non-contradictory rules, and the number of remaining traces to consider during tree building will be significantly reduced in most cases. TREEREFINEMENT can then be used on the remaining traces in the translated trace semantics as previously discussed. Since FILTERTRACES performs

a complete iteration over $\phi(TS(\mathcal{MV}))$, its complexity is $O(|\phi(TS(\mathcal{MV}))|)$. Thus, the worst case complexity of TREEREFINEMENTPF is

$$O(|\phi(TS(\mathcal{MV}))| + |Ref^{\phi}_{\mathcal{MV}}| \cdot |\phi(TS(\mathcal{MV}))|)$$

if FILTERTRACES removes few traces prior to tree construction. Despite this, FILTERTRACES will significantly reduce the size of the translated trace semantics in the best and average cases, and so will minimise the number of incorrect leaf nodes (i.e. leaf nodes which are created and later removed). Furthermore, we have the advantage of being able to detect immediately if no refinements exist, avoiding the potentially costly process of tree building.

A parallel tree-based approach

In this section, we outline a parallel adaption of the TREEREFINEMENT algorithm called PARALLELTREEREFINEMENT, which aims to alleviate (to some extent) the problem of tree building when a large number of refinements exist. In fact, in the worst case the number of leaf nodes can increase exponentially with each iteration of global state $S \in \phi(S_{\mathcal{MV}})$, and so TREEREFINEMENT can soon struggle for resources on a single processor.

A key property of the trees built by TREEREFINEMENT is the immutability of all non-leaf nodes, since we only add new rules to the leaf nodes. As such, leaf nodes are independent of one another, and so can be processed in parallel. The idea is that instead of iterating over all leaf nodes of the tree with the set of traces $AllTraces_S$, for some $S \in \phi(S_{\mathcal{MV}})$, we perform TREEREFINEMENT on each leaf node in parallel across multiple processors.

In reality, there are two strategies for performing this computation in parallel. The first is a 'breadth-first' approach which compares the traces starting from one $S \in \phi(S_{\mathcal{MV}})$ against all leaf nodes in parallel before the next S is considered. Whilst this approach proves useful for TREEREFINEMENT, it relies on a notion of synchronisation which is not suited to the possibility of a heterogenous computing environment.

A more suitable strategy is to compute the tree in parallel asynchronously; that is, each processor performs TREEREFINEMENT independently without synchronising. In this case, computation of $Ref^{\phi}_{\mathcal{MV}}$ is successful if each processor performs TREEREFINEMENT until each $S \in \phi(S_{\mathcal{MV}})$ has been covered, and only terminates if TREEREFINEMENT terminates prematurely on all processors. In addition, we introduce a parameter $TreeSize$ which denotes the maximum number of leaf nodes that must be present in the tree before it is split up. Thus, TREEREFINEMENT is performed as normal until the number of leaf nodes in the tree exceeds $TreeSize$, in which case the tree is partitioned and further processed in parallel. As such, the algorithm can be tuned to avoid the set-up costs of parallel computation for small models. The PARALLELTREEREFINEMENT algorithm is abstractly outlined by Algorithm 6.

If we have N processors, then the complexity of PARALLELTREEREFINEMENT is

$$O(|\phi(TS(\mathcal{MV}))| + \frac{|Ref^{\phi}_{\mathcal{MV}}| \cdot |\phi(TS(\mathcal{MV}))|}{N}),$$

Algorithm 6 PARALLELTREEREFINEMENT: performs TREEREFINEMENT in parallel by partitioning tree.

Inputs: Node *Root*, Model $\mathcal{MV}$, Mapping vector ϕ, Integer *TreeSize*,
Outputs: Set of refinements $Ref^{\phi}_{\mathcal{MV}}$

1: $Ref^{\phi}_{\mathcal{MV}} \longleftarrow \emptyset$
2: Perform FILTERTRACES to reduce size of $\phi(TS(\mathcal{MV}))$
3: Perform TREEREFINEMENT until number of leaf nodes exceeds *TreeSize*
4: **for all** leaf nodes L in **parallel do**
5: Perform TREEREFINEMENT on L using $\phi(TS(\mathcal{MV}))$
6: **if** TREEREFINEMENT returns $\emptyset$ **then**
7: terminate process
8: **end if**
9: **end for**
10: **if** TREEREFINEMENT terminates prematurely on all processors **then**
11: **return** $\emptyset$
12: **else**
13: **return** $Ref^{\phi}_{\mathcal{MV}}$ from tree representation
14: **end if**

if we perform FILTERTRACES on a single processor and then parallelise TREEREFINEMENT. Thus, we only gain a linear speed-up to an exponential problem, but this can still benefit the refinement of larger models. Furthermore, its performance relies on the *TreeSize* parameter. Indeed, it would be interesting to investigate methods for determining a suitable value for such a parameter in order to strike a balance between the set-up costs of parallel computation and the costs of performing TREEREFINEMENT on a single processor.

5.3.2 Validating a refinement

We conclude by considering the problem of verifying whether one MVN $\mathcal{MV}'$ is a refinement of another $\mathcal{MV}$. In contrast to searching for models which refine another as discussed previously, here we search for mapping vectors under which the refinement occurs.

Here, we consider a straightforward exhaustive approach. The idea is that for an entity $g_i \in \mathcal{MV}$, a mapping set $MS^{m\to n}_{g_i}$ contains all possible state mappings which reduce $Y(g_i)$ to $Y(g'_i)$, for $g'_i \in \mathcal{MV}'$. Note if $Y(g_i) = Y(g'_i)$, then the mapping set will just contain the identity mapping. Assuming that we have k entities, the total number of possible mapping vectors which map each entity is then given as the cartesian product of each mapping set:

$$MS^{m_1\to n_1}_{g_1} \times \cdots \times MS^{m_k\to n_k}_{g_k}.$$

This search can be done exhaustively as shown by Algorithm 7.

Note we will apply this exhaustive approach in Section 5.5 to verify whether our BN model for *E. coli* is a refinement of the corresponding MVN according to Definition 5.8.

Algorithm 7 VERIFYREFINEMENT: verifies for two MVN models $\mathcal{MV}$ and $\mathcal{MV}'$ whether $TS(\mathcal{MV}') \subseteq \phi(TS(\mathcal{MV}))$ holds by checking all possible mapping vectors ϕ.

Inputs: Model $\mathcal{MV}$, Model $\mathcal{MV}'$
Outputs: Boolean

```
1: for all possible mapping vectors φ such that φ(S_MV) = S_MV' do
2:   if TS(MV') ⊆ φ(TS(MV)) then
3:     return true
4:   end if
5: end for
6: return false
```

5.4 Benchmarking

In this section, we benchmark the performance of EXHAUSTIVEREFINEMENT (ER), TREEREFINEMENT (TR) and TREEREFINEMENTPF (TR$_{PF}$). To achieve this, we have implemented a prototype Java tool REFINER to act as a test bench for these three algorithms. REFINER provides a basic GUI for loading in, refining and saving MVN models, and integrates closely with our PN support tool GNAPN. REFINER is freely available as a separate tool for academic use from *bioinf.ncl.ac.uk/gnapn*.

We consider using REFINER to benchmark our algorithms against a range of MVN models. For each model $\mathcal{MV}$ and for each algorithm, we calculate the refinement set $Ref^{\phi}_{\mathcal{MV}}$ under a number of different mapping vectors ϕ. We record $|\phi(S_{\mathcal{MV}})|$ (the size of the translated state space), $|G|$ (the number of entities $g_i \in \mathcal{MV}$), $|Potential|$ (the total number of possible deterministic models derivable from the non–deterministic state transition tables $\phi(\mathcal{MV})$) and $|Ref^{\phi}_{\mathcal{MV}}|$. Algorithm run times are shown in seconds, and we record a '-' where completion does not occur in a reasonable time. These results are shown by the following table. Note all tests were performed on a 2.8GHz Pentium 4 with 2GB memory.

As expected, the run times for EXHAUSTIVEREFINEMENT become increasingly large as $|Potential|$ increases, since for each model it must check for the trace semantics containment property of Definition 5.8. Thus, even for modest values of $|Potential|$ (e.g. 256), we begin to see substantial algorithm run times, irrespective of whether refinements exist or not. Indeed, for model (4), the number of potential models becomes so large that it takes over five minutes to find all refinements, and for the third mapping which has $|Potential| = 2097152$, EXHAUSTIVEREFINEMENT fails to find all refinements in a reasonable amount of time (we estimate it to take approximately 160 minutes).

In contrast, we observe the performance of TREEREFINEMENT and TREEREFINEMENTPF to be substantially lower in all cases, since their run time depends on $|Ref^{\phi}_{\mathcal{MV}}|$. For the smaller models $(1)-(3)$ (and surprisingly for (6), the largest model tested), these run times are consistently less than 0.01 seconds when little or no refinements exist. This is due to both their ability to detect when no refinements exist early, and to converge quickly to a solution. Moreover, for the medium sized model (4) in which EXHAUSTIVEREFINEMENT struggled, we see run times of less than half a second for the first two cases, and for the third mapping vector under which EXHAUSTIVEREFINEMENT failed

Model	$\lvert\phi(S_{\mathcal{MV}})\rvert$	$\lvert G\rvert$	$\lvert Potential\rvert$	$\lvert Ref^{\phi}_{\mathcal{MV}}\rvert$	ER [s]	TR [s]	TR_{PF} [s]
(1)	4	2	2	0	< 0.01	< 0.01	< 0.01
	4	2	4	1	0.03	< 0.01	< 0.01
	4	2	16	1	0.13	< 0.01	< 0.01
(2)	16	3	288	0	0.24	< 0.01	< 0.01
	16	3	2048	0	1.14	< 0.01	< 0.01
	16	3	4096	0	2.22	< 0.01	< 0.01
(3)	24	3	32	1	0.08	< 0.01	< 0.01
	24	3	64	0	0.12	< 0.01	< 0.01
	24	3	256	0	3.02	< 0.01	< 0.01
(4)	128	4	65536	1	318.82	0.2	0.2
	128	4	65536	64	302.54	0.31	0.13
	128	4	2097152	1024	–	5.42	3.99
(5)	3072	7	$2.79 * 10^{41}$	8	–	9.04	2.30
	3072	7	$3.56 * 10^{44}$	16	–	16.67	5.54
	3072	7	$2.79 * 10^{44}$	512	–	534.17	61.98
(6)	4608	7	128	0	109.94	< 0.01	< 0.01
	4608	7	256	0	227.89	< 0.01	< 0.01
	4608	7	$2.79 * 10^{41}$	4	–	5.37	1.52
	4608	7	$2.79 * 10^{41}$	8	–	12.00	2.54

Figure 5.7: Benchmarking results for refinement algorithms.

to terminate, we see run times of no more than five seconds; a substantial improvement over 160 minutes!

Apart from showing clear improvements over ExhaustiveRefinement, these results also exemplify important differences between TreeRefinement and TreeRefinementPF. More specifically, the pre-filtering performed by TreeRefinementPF yields significant improvements over TreeRefinement, since it removes all invalid traces and thus minimises the number of erroneous leaf nodes that are added to the tree during construction. In fact, in some cases, the pre-filtering offers a tenfold reduction in run time.

Despite these promising results, there are indications that TreeRefinementPF will struggle when the models become larger and when the number of refinements increases (an interesting area of future work would be to investigate these practical limitations). As such, further improvements are clearly required if these techniques are to be scalable and manageable. One improvement which we discussed previously is the use of parallel computation, but as mentioned this can only yield a linear improvement to an exponential problem in the best case. Therefore, there is a real motivation for the the development of more intelligent algorithms which exploit key properties between a model and its refinements, but further theoretical understanding will be required to achieve this. In particular, heuristic techniques which approximate the refinement set may be required to solve this problem in a practical timescale.

5.5 Application to *E. coli* models

The key motivation for this chapter was the comparison presented in Section 4.6, which looked at the similarities and differences between our BN and MVN models of the carbon stress response network in *E. coli* [167]. Specifically, we noted a number of behavioural differences between the two, and this led to some more general questions concerning their relationship. However, with the lack of techniques for reasoning about the two models, we had no means of answering this question. Note we will denote the BN as $\mathcal{BN}_{ecoli}$ and the MVN as $\mathcal{MV}_{ecoli}$.

In this section, we investigate their relationship by formalising a validation problem to see whether $\mathcal{BN}_{ecoli}$ is a refinement of $\mathcal{MV}_{ecoli}$; that is, whether $\mathcal{BN}_{ecoli} \lhd^{\phi} \mathcal{MV}_{ecoli}$ holds, where ϕ is unknown. To achieve this, we must therefore check all possible mapping vectors which map the states of each entity in $\mathcal{MV}_{ecoli}$ to those in $\mathcal{BN}_{ecoli}$.

We start by considering the individual state spaces for each entity in $\mathcal{MV}_{ecoli}$:

$$\begin{aligned} |Y(Fis)| &= 6, \\ |Y(Cya)| &= 4, \\ |Y(Crp)| &= 4, \\ |Y(Top)| &= 4, \\ |Y(GyrAB)| &= 4, \\ |Y(SRNA)| &= 2, \\ |Y(Signal)| &= 2. \end{aligned}$$

Based on this, mapping sets can be specified containing all possible ways in which these state spaces can be translated into the Boolean domain using the results of Theorem 5.1:

$$\begin{aligned} |MS^{6\to 2}_{Fis}| &= 31, \\ |MS^{4\to 2}_{Cya}| &= 7, \\ |MS^{4\to 2}_{Crp}| &= 7, \\ |MS^{4\to 2}_{TopA}| &= 7, \\ |MS^{4\to 2}_{GyrAB}| &= 7, \\ |MS^{2\to 2}_{SRNA}| &= 1, \\ |MS^{2\to 2}_{Signal}| &= 1. \end{aligned}$$

Thus, the total number of possible mapping vectors is given as the cartesian product of each mapping set:

$$|MS^{6\to 2}_{Fis} \times MS^{4\to 2}_{Cya} \times MS^{4\to 2}_{Crp} \times MS^{4\to 2}_{TopA} \times MS^{4\to 2}_{GyrAB} \times MS^{2\to 2}_{SRNA} \times MS^{2\to 2}_{Signal}| = 74431.$$

For each possible mapping vector ϕ, we then simply check whether $TS(\mathcal{BN}_{ecoli}) \subseteq \phi(TS(\mathcal{MV}_{ecoli}))$ holds using the approach of Algorithm 7. This process is automated by our support tool REFINER, which, upon completion, is able to verify that no possible mapping vector ϕ exists such that $\mathcal{BN}_{ecoli} \lhd^{\phi} \mathcal{MV}_{ecoli}$. As such, we are able to conclude that $\mathcal{BN}_{ecoli}$ is not a refinement of $\mathcal{MV}_{ecoli}$ according to our theory.

This result is not surprising, as $\mathcal{BN}_{ecoli}$ was constructed manually by judging the manner by which the multi-valued states should be thresholded into Boolean values. As such, there may exist inconsistencies in the way certain states were translated. This could explain why we noted behavioural differences between $\mathcal{BN}_{ecoli}$ and $\mathcal{MV}_{ecoli}$ in Section 4.6, such as the erroneous activation of TopA in $\mathcal{BN}_{ecoli}$ during the stationary phase which was not captured in $\mathcal{MV}_{ecoli}$. Of course, this result is based on our relationship assumption, but it nevertheless offers a means of shedding light into their relationship, and provides a promising foundation for future development. Indeed, and to the best of our knowledge, this appears to be the first attempt at systematically answering such questions.

Work is now required to address another problem: whether there exists an alternative BN which *does* respect our refinement theory. For this, we would have to calculate the refinement set by considering each possible mapping vector in turn until a refinement is discovered. This is a challenging task and is left as an interesting area of future work.

5.6 Discussion

5.6.1 Conclusions

This chapter has proposed what appears to be the first approach for systematically exploring the relationship between MVNs at different levels of abstraction. This was motivated by some interesting behavioural differences observed between the BN and MVN *E. coli* models from Chapters 3 and 4 which raised questions concerning their relationship. In particular, more general questions were raised as a result, such as the scope and limitations of Boolean modelling, and the identifcation of situations where multi-valued modelling is required.

We developed a refinement theory which captured an assumption of what it meant for a simpler MVN to be representative of another. The basic idea was to relate the state spaces of the two MVNs using a mapping, and to check that the simpler one captured consistent behaviour with respect to the other. Based on this relationship assumption, we then developed algorithms for deriving and validating refinements, thus making our approach amenable to practical modelling in the biological community. In particular, some important issues were addressed during this development to improve efficiency. We started with a straightforward exhaustive approach for deriving refinements, and proved an interesting property in Theorem 5.2 that significantly reduced the search space. We showed that the complexity of this exhaustive approach was dictated by an exponential number of possible models, and so proposed a tree-based algorithm whose complexity depended on the number of refinements. Since we observed that the number of refinements was often relatively tiny in comparison to the number of possible models, this therefore represented a significant optimisation, as shown in our benchmarking. A parallel adaption of this algorithm was also proposed along with an implementation strategy, but this was left as an interesting and much-needed area of future work.

Based on our refinement theory assumption, we demonstrated how some interesting insights could be obtained into the above mentioned considerations. We gave an example

in Theorem 5.4 which motivated multi-valued over Boolean modelling. The example involved an entity in which one state was used to perform two different interactions. Although valid in practical modelling, it hinted at situations in which subtle behaviours could not be adequately represented by higher levels of abstraction. Indeed, more work is now needed to investigate more general motifs of behaviour which are problematic to model more abstractly.

The scope and limitations of analysing refinements were also identified. In particular, we showed a correspondence between reachability properties in an MVN and its refinements in Theorem 5.5, but identified in Corollary 5.2 that these were in fact semi-deducible for most refinements. Thus, there is a certain amount that one can deduce about an MVN using its refinements, but it is clear that further developments are needed to investigate more useful notions of refinement geared towards analysis. In particular, we see this essential in addressing a major application area of this work: circumventing state space explosion by deriving the simplest model which can answer the particular question of interest.

Finally, we showed how our framework could be applied to address the initial motivation for this chapter, by investigating the relationship between the Boolean and multi-valued models for *E. coli*. We demonstrated how this could be formulated as a verification of whether the Boolean model was a refinement of the multi-valued one, and discovered that this relationship did not hold according to our theory. This too demonstrates promising potential for this work, and thus motivates further development. In particular, the biological significance of the refinement theory proposed has been justified, but may require adaption for future investigations. Specifically, it would be interesting to assess whether a more relaxed or strict theory could provide a 'better' relationship, but 'better' needs to be biologically justifiable.

Overall, this work has indicated much potential for systematic reasoning in biological models, as well as the development of invaluable techniques for addressing state space explosion in practical analysis tasks.

5.6.2 Future work

Improved algorithmic approaches for systematic refinement are required if these techniques are to be applied to large practical models. A good starting point would be the implementation of the ParallelTreeRefinement algorithm. This parallel adaption to TreeRefinement was proposed based on the observation that the rule tree could be computed concurrently. Furthermore, since the worst case performance of TreeRefinement occurred when the refinement set was large, computation would be greatly optimised by using multiple processors working on sub-trees. One possibility for this would be to use a distributed computing framework such as Condor [202] in conjunction with a database to implement a shared memory environment.

We are also interested in exploring the extent to which entities need to be considered during refinement. Currently, refinement is based on a global view of the system where all entities are considered. This guarantees that the refinement is indeed consistent with the original model, but involves computation of the complete state space. If, however,

only a subset of the total number of entities can be used, then this will significantly improve the performance of any algorithms.

Finally, it would be interesting to develop more efficient and systematic approaches for verifying whether one model is a refinement of another. In Section 5.5, we used REFINER to search for a mapping vector under which our BN model for the carbon stress network in *E. coli* refined our MVN model of the same system. This involved searching over 70000 possible mapping vectors and checking for trace containment. Further studies would be to see if all mapping vectors are required, or whether symmetry properties can be used to reduce the search space.

5.6.3 Sources

Initial ideas and motivation for the refinement approach presented in this chapter form part of the paper [20] which was presented at the 4th Integrative Bioinformatics Workshop 2007.

Chapter 6

Developing Realistic Asynchronous Boolean Networks

BNs [101, 102] are traditionally interpreted in a synchronous fashion, and this has been reflected by their wide body of literature in the biological community [6, 101, 103]. In particular, synchronous BNs are often favoured for their simplicity, since they avoid the need to consider reaction rates. However, the assumption that all entities update their state at exactly the same time can be argued to be biologically unrealistic [86], and so this raises concern on the analysis of such models.

Motivation is therefore placed on their asynchronous counterparts [206]. In an asynchronous BN, each entity can update its state independently, which fits better with our understanding of gene regulation occurring at different rates [86]. However, asynchronous BNs also have shortcomings: they address the issue of reaction rates by erring on the side of caution and allowing for all possible scenarios to occur. The result is that they can often capture too much behaviour - some of which is *unrealisable in practice*. Furthermore, this behaviour is often highly non–deterministic due to non–converging *choices*, making analysis and interpretation problematic.

In theory, such choices can often be resolved either by assuming that the environment of the biological system is slow enough to allow its behaviour to stabilise, or by having some knowledge of the necessary reaction rates; that is, the system exhibits much less non–determinism than the model suggests. In practice, however, it can be unclear what these critical choices are and how to resolve them, since very little kinetic data is currently available (this appears to explain why synchronous networks have mostly been favoured by the biological modelling community [86]).

In this chapter, we begin to address these concerns by applying PN techniques from *asynchronous circuit design* based on *speed-independent (SI)* circuits [55, 191] and *signal transition graphs (STGs)* [54, 168]. SI circuits are a subclass of asynchronous circuits that function correctly according to their specification regardless of gate delays, and we argue that they are also well-suited to biological systems.

The foundation of this work is therefore the methodological assumption that *biological systems can be qualitatively modelled by SI circuits*, and we argue that if this is not

the case, then it implies that the model is either incorrect or misses some important information. Note due to clear similarities, we will use the terms 'BN' and 'circuit' interchangeably, and will sometimes refer to the nodes of a BN as 'gates'.

It turns out that whether a circuit is SI or not almost always depends on the behaviour of its environment, and we show that this environmental information cannot be correctly captured by the circuit alone. This consideration therefore motivates STGs, a PN formalism developed specifically for the specification of SI circuits [55, 191]. In particular, STGs enable the modeller to capture in a natural way both the behaviour of the circuit *and* its environment - something which is crucial for comprehensively modelling GRNs.

We investigate how the sufficient conditions ensuring that an STG can be implemented by an SI circuit can be interpreted in the biological context for GRNs. We observe that these properties provide important insights into a model and highlight areas which need to be refined. In particular, the violation of the *output-persistency (OP)* [55] condition indicates the presence of choices that either require further information to resolve or indicate some stochastic effects in the system that should be carefully documented. STGs provide a formal means of documenting and refining this information, and thus provide a well-supported formal framework for GRNs that allows realistic asynchronous models to be incrementally developed and analysed.

In order to make our STG approach practical, we develop appropriate tool support to enable STGs to be automatically constructed and refined. We extend the PN construction tool GNAPN from Chapters 3 and 4 to cater for STGs, and then develop a new tool STGTOOL to enable the automatic identification and resolution of OP violations in the constructed model. In particular, STGTOOL highlights only those choices which affect the qualitative behaviour of the system, and allows the user to resolve these by requesting *relative* (rather than absolute) reaction rates.

We demonstrate our proposed approach and tool support by considering a case study in which we develop and analyse a model of the GRN controlling the switch between the lysogeny and lysis cycles in phage λ. We begin by constructing an STG model based on the asynchronous BN presented in [206]. We then refine this by finding the points where the STG violates SI, and appropriately resolving the problems. In particular, we see how some violations of OP highlight timing assumptions about the environment's behaviour, whilst some represent the stochastic switching mechanism between the lysis and lysogeny modes which cannot be resolved and are modelled using arbitration.

The rest of this chapter is organised as follows. We start by briefly introducing STGs in Section 6.1. We then introduce the concepts and properties of SI circuits in Section 6.2 by considering the manual construction of an STG capturing the high-level behaviour of the *B. subtilis* sporulation network from Section 3.3.2. In Section 6.3, we show how STGs can be derived from circuit specifications, since these are what one often starts with in practice. Moreover, we propose an algorithmic approach for identifying OP violations in the constructed STG, and then develop a mechanism for refining these violations, ultimately resulting in an SI STG. In Section 6.4, we discuss the implementation of a tool STGTOOL which automates the theory presented. In Section 6.5, this theory and tool support are demonstrated using a detailed case study on the lysis-lysogeny switch in

phage λ. Finally, Section 6.6 concludes by summarising our results and discussing some future work.

6.1 Signal transition graphs

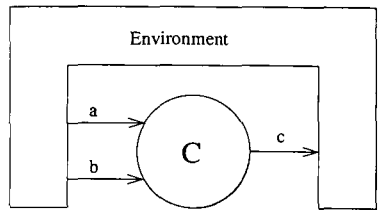
An STG [55, 168] is a particular type of labelled PN developed for the specification of asynchronous digital circuits. In an STG, a set of Boolean variables called *signals* is associated to the underlying PN, which collectively represent the global state of the digital signals (i.e. wires) within the actual circuit. The states of these signals are then changed by the transitions of the PN, which are labelled accordingly as follows: a transition label a^+ indicates that signal a goes from 0 to 1, and a label a^- indicates that signal a goes from 1 to 0. Note in some cases there may be several transitions with the same label (e.g. a^+), and so these are named a^+, $a^+/1$, $a^+/2$, and so on. Thus, the underlying PN specifies the causal relationship between signals which captures the behaviour of the circuit, and as such, the standard enabling and firing rules of PNs still apply. Moreover, STGs are amenable to both established PN techniques and tools [1], as well as specialist STG tools such as PETRIFY [54].

The signals of an STG are partitioned into *input*, *output* and *internal* signals. The input signals are controlled by the *environment* of the circuit - an abstraction representing its surroundings, such as the output of another circuit. The output signals, on the other hand, represent those produced by the circuit. Finally, the internal signals represent those signals inside the circuit that are required in order for it to work correctly, and are therefore invisible to the environment. For convenience, the output and internal signals are often referred to collectively as *local* signals, since they are the signals that the circuit must directly implement. Note the appropriate partitioning of signals into these three types represents an important design step when developing an STG model.

For example, Figure 6.1(a) shows a commonly used component in asynchronous circuit design called a *Muller C-element* [55]. The C-element takes two inputs a and b from the environment and produces a single output c back into the environment (there are no internal signals). The idea is that once the environment has raised a and b, the C-element raises c and it remains high until the environment has dropped both a and b. This behaviour can be completely captured by the state transition table in Figure 6.1(b), and specified by the STG shown in Figure 6.1(c) (note unlike PNs, we use common shorthand notation for STGs, where transitions are simple denoted by their labels and non-marked places with only one input and one output transition are contracted).

From the initial state specified in Figure 6.1(c) in which all signals are low, the system waits until the environment raises (in any order) the inputs a and b (transitions a^+ and b^+) before raising the output c (transition c^+). Note the environment is assumed not to reset the raised inputs until c^+ fires. Then the environment resets (in any order) the inputs a and b (transitions a^- and b^-), and in response the system resets its output c (transition c^-); again, the environment is assumed not to raise the inputs until c^- fires.

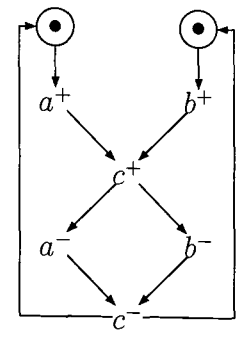
Intuitively, an STG captures a contract between the circuit and its environment, in that it specifies how the circuit should react to changes in the environment, and what the environment should expect from the circuit. More specifically, this contract is as follows:



(a)

a	b	c	$[c]$
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

(b)



(c)

Figure 6.1: (a) A Muller C-element, (b) state transition table for C-element, and (c) STG capturing behaviour of C-element using shorthand notation.

- if an input signal transition is enabled, then the environment is allowed (but not obliged) to send this input (the environment is not allowed to send inputs which are not enabled);
- if a local signal transition is enabled, then the system is obliged to eventually produce this signal (unless it is disabled by the firing of another transition, which we will consider later), and is not allowed to produce a signal if it is not enabled.

In other words, the STG specifies that the system must be able to cope with *at least* the specified inputs and produce *all and only* the specified outputs.

In this chapter, we interpret STGs biologically as follows. Signals are used to represent the states of biological entities and transitions are used to capture changes in these states, e.g. through chemical reactions or fluctuations in protein concentration. In particular, the input signals from the environment can be used to represent factors such as temperature, or simply the output from other sub-systems in the organism, and the internal signals can be used to represent any auxiliary biological entities that are required for the system to function correctly. The output signals can therefore be interpreted as the result of these environmental factors and internal mechanisms, such as a change in protein concentration.

It turns out that the ability of STGs to capture both the circuit and its environment is key for the modelling and analysis discussed in this chapter. Firstly, accounting for the behaviour of the environment is crucial in understanding how complex biological systems adapt to their surroundings. More important for this chapter, however, is the fact that environmental assumptions are critical for the development of SI circuits (and thus, according to our methodological approach, biological systems), since it is possible for a circuit to be SI in one environment yet non-SI in another.

For example, the STG in Figure 6.1(c) captures a circuit which is SI in the environment specified. This is because the environment must raise a and b before the output signal c is raised (and is assumed not to reset either of them until after this has occurred), and then the environment must drop a and b before c is dropped (again, the environment is assumed not to raise either before c has dropped). As such, the output signal c is

produced correctly no matter what delays are associated with signals a and b. However, in a slightly more demanding environment where the environment can drop a before c rises, this STG becomes non-SI. In this particular case, this is due to the violation of an important property known as *output-persistency (OP)* [55] which we will discuss in more detail later.

Of course, there are other properties apart from OP that a circuit must satisfy to be SI, and we will introduce these in the next section. In particular, we discuss why their violation can be used to indicate model construction errors or missing information in the context of biological systems.

6.2 Speed-independent circuits: properties and construction

In this section, we introduce the properties which ensure that a circuit is SI, and relate these to the identification of missing or erroneous information in an asynchronous model of a GRN. To accomplish this, we revisit the sporulation network for *B. subtilis* [197] from Section 3.3.2, and consider the step-by-step construction of an STG which models its high-level behaviour. As an overview, the properties that we will discuss here for SI to hold are *boundedness* (see Section 2.3.3), *consistency* (signal changes always alternate between rising and falling for each signal), *complete state coding (CSC)* (CSC ensures that the circuit always knows what to do next for any reachable state) and OP (a local signal cannot be disabled). We will then discuss some further properties which are not directly required for SI, but whose violation is nevertheless suspicious and should be investigated.

The *B. subtilis* sporulation model from Section 3.3.2 is a BN consisting of 12 entities, but here we consider a much simpler version for illustrative purposes. Specifically, we focus on a high-level representation requiring only three entities: Signal, which indicates the presence of nutrient starvation and which must be present for sporulation to occur as shown in Section 3.3.2; SigA, a sigma factor whose presence we showed was required for sporulation to occur in [194]; and SigF, a sigma factor whose presence we use as an indication that sporulation has occurred.

As already mentioned, the partitioning of input and local signals represents a key design step. In our case, we therefore assign *Signal* as an input because it represents an environmental condition. Also, since *SigA* is regulated outside the scope of the model from Section 3.3.2, we will also treat it as an input signal from the environment. On the other hand, *SigF* is used as an indication for sporulation, and should therefore be an output of the STG to the environment. This high-level circuit, which we will refer to as the *bacillus-circuit*, is shown abstractly in Figure 6.2.

The idea then is that we want to construct an STG which captures the behaviour of the bacillus-circuit in response to its environment. From what we have already discussed, we know that sporulation should only occur (i.e. *SigF* raises) in the presence of *Signal* and *SigA*. Moreover, sporulation should continue as long as both are present; that is, if either *Signal* or *SigA* drop then sporulation should stop and vegetative growth

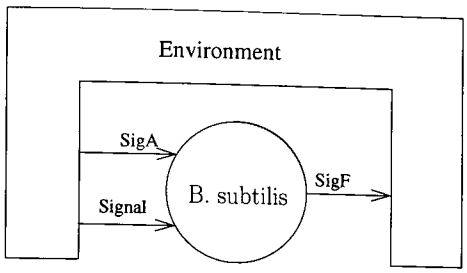


Figure 6.2: The bacillus-circuit, an abstract view of the *B. subtilis* sporulation network.

should occur (i.e. signal $SigF$ drops). Note such behaviour can be implemented by an AND-gate, but we will nevertheless discuss the construction of this STG to introduce the properties of SI circuits.

In order to formalise this behaviour, we begin to construct an STG by first deciding upon a suitable initial state. In most cases, it is prudent to choose a stable initial state in which the circuit is awaiting a request from the environment (note in general the choice of initial state is outside the scope of this chapter, and so we just remark that typically biological systems have cyclic behaviour, and that any state in the cycle can be chosen). We therefore choose an initial state in which all signals are low, where the bacillus-circuit waits for the environment to raise $Signal$ and $SigA$. The idea then is that once $Signal$ and $SigA$ have risen (in any order), the bacillus-circuit can raise $SigF$ to represent sporulation (note the environment is assumed to be relaxed and does not reset either $Signal$ or $SigA$ before this has occurred). This STG fragment is shown in Figure 6.3(a).

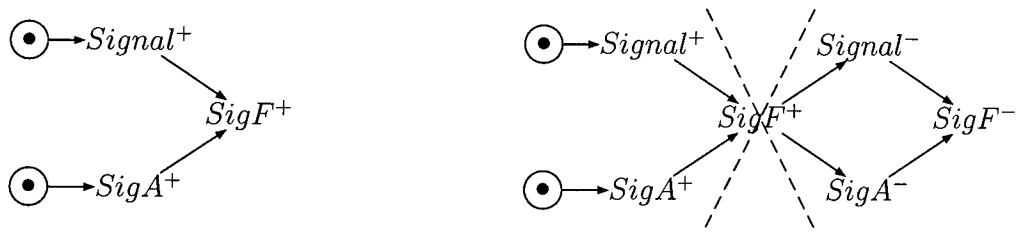


Figure 6.3: (a) STG fragment showing that circuit must wait for environment to raise $Signal$ and $SigA$ before sporulation can occur, and (b) incorrect STG for dropping $SigF$, since it does not allow sporulation to stop until *both* $Signal$ and $SigA$ have been dropped.

Once $SigF$ has been raised, the bacillus-circuit is again stable, and waits for the environment to drop $Signal$ and $SigA$. One possibility is for the circuit to wait for the environment to concurrently drop $Signal$ and $SigA$ before dropping $SigF$, as shown in Figure 6.3(b). However, whilst this situation accounts for when the environment is faster than the circuit, it does not include all the behaviour that we want, since sporulation should also stop when *either* $Signal$ or $SigA$ are low. We therefore need to account for two additional situations in the STG in which only one signal has been dropped. As

such, the following three environmental behaviours need to be considered: only *Signal* is dropped, only *SigA* is dropped and both are dropped (i.e. the environment is fast enough to drop both before the circuit can react). The STG fragment capturing these three situations is shown in Figure 6.4.

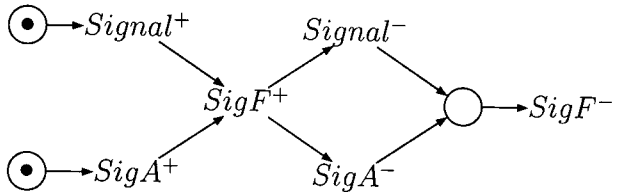


Figure 6.4: STG representing termination of sporulation in all three behaviours of the environment.

In the case where both *Signal* and *SigA* have dropped, there will be two tokens on the pre-place for $SigF^-$, and thus we say that this place is 2-bounded; from here, *SigF* should drop to reset the STG to its initial marking. In both other cases, however, there will be only one token, and so $SigF^-$ should be allowed to drop whilst still allowing the input which has not dropped to do so (again, we assume the most relaxed environment for the purpose of this example, which does not raise either input signal before this has occurred). These three situations can be compactly modelled with the use of a so-called dummy transition, which does not produce a signal and simply acts as a synchronisation mechanism. We show this logic in the STG in Figure 6.5.

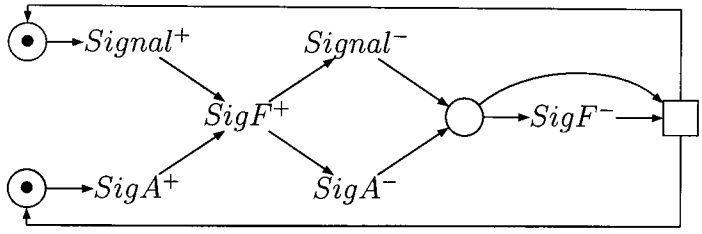


Figure 6.5: Incorrect STG with dummy transition.

The causal conditions for the dummy transition require that $SigF^-$ fires first, thus accounting for all three situations discussed before. Then, assuming that both input signals have dropped, the dummy will be enabled to fire, and will reset the STG to the initial marking. Therefore, this STG seems to capture the desired behaviour at first glance, but closer inspection reveals a problem: currently, it is possible for *SigF* to drop twice consecutively. This leads us to discuss our first two properties, *boundedness* and *consistency*, which all SI specifications must respect (note a third property called *deadlock-freeness* is also violated here, but is not required for SI and so will be discussed shortly).

6.2.1 Boundedness and consistency

Boundedness, as already discussed in Section 2.3.3, is a property which ensures that the number of tokens on places can never exceed some bound k. The rationale for this is that if all places are bounded, then the reachable markings of the STG will also be bounded. Since any digital circuit can have only finitely many reachable states, we therefore require the STG to have finitely many reachable markings. Note boundedness is a necessary implementability requirement for any digital circuit, not just those with SI.

The STG in Figure 6.5 is bounded; specifically, it is 2-bounded, as the pre-place to $SigF^-$ can contain up to two tokens. However, the implicit post-place of $SigF^-$ has been assumed to be safe, but since $SigF^-$ can fire twice, it is now also 2-bounded. If the STG has been constructed based on the assumption that this post-place is safe, then this clearly highlights a modelling error which must be addressed.

In fact, this boundedness violation is a direct result of the violation of another property, called *consistency*. Consistency is a property which requires that the transition labels for each signal a alternate between a^+ and a^-, and always begin with the same sign; in other words, the reachable signal values are binary. Note consistency, like boundedness, is a necessary implementability requirement for any digital circuit, not just SI ones.

In this case, signal $SigF$ does not respect this property, and so we need to ensure that $SigF$ cannot fire twice consecutively. To achieve this, we use an additional input place to $SigF^-$ to allow it to fire only once, resulting in the new STG shown in Figure 6.6.

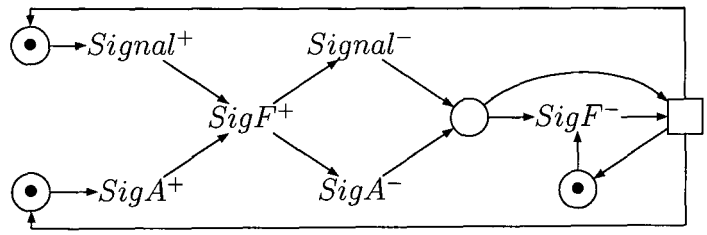


Figure 6.6: SI STG for bacillus-circuit in the most relaxed environment.

Now we observe that this STG respects boundedness and consistency. In fact, it turns out that it also respects another requirement implementability property, called *complete state coding (CSC)*.

6.2.2 Complete state coding

Two reachable markings in an STG are said to be in a complete state coding (CSC) confict if the values of all signals coincide, but the sets of enabled local signals differ. Thus, an STG satisfies the CSC property if no two reachable states are in a CSC conflict.

Intuitively, during the execution of the circuit, it can 'see' only the values of its signals, but not the marking of the STG. Hence, if two semantically different reachable states

with the same values of all the signals exist, the system cannot distinguish between them, and so cannot know what to do next. At the circuit level, CSC conflicts are resolved by inserting new internal signals helping to distinguish between the conflicting states, in such a way that its 'external' behaviour does not change. Intuitively, insertion of a signal introduces additional memory into the circuit, helping it to trace the current state.

In an STG modelling a biological system, CSC conflicts can be interpreted as a lack of information about its internal workings. That is, they indicate the presence of some auxiliary internal entities (e.g. proteins) which are not visible to the environment but help the system to accomplish its function. Discovering CSC conflicts during model construction can therefore prove useful for indicating the possibility of incomplete data, and should be documented as part of the development process.

6.2.3 Output persistency

Whilst boundedness, consistency and CSC are necessary properties for an STG to be implementable as a digital circuit, the final property, called *output-persistency (OP)*, is required for it to be SI. In particular, OP guarantees the robustness of the circuit under any delay of its gates, and violations of OP indicate the presence of choices in the model that require further information to be resolved. Thus, we use such choices as a means of highlighting areas of the model where further knowledge of reaction rates and environmental behaviours is required.

The OP property requires that once a signal is enabled, it cannot be disabled before it is produced. The rationale for this in circuit design is that once a signal becomes enabled, its voltage starts to change, and if it is disabled unexpectedly before this has completed, then it could be incorrectly interpreted by other gates listening to it. For example, if some signal a becomes enabled, its voltage may start to (say) rise from 0 to 1. However, if a is disabled unexpectedly during this process, its voltage is pulled down, resulting in a non-digital pulse on the corresponding wire which can be interpreted differently depending on whether the voltage has crossed the threshold between 0 and 1 or not. Hence, the behaviour of the circuit becomes non–deterministic (of course, these intermediate states can be handled when we introduce extra states, such as in the MVNs discussed in Chapter 4, but we do not consider this case here). Note we can interpret OP from a biological point of view by replacing voltage with, for example, protein concentration.

Visually, if OP is violated then there exists a *choice* between two transitions t and t' in the STG, where both are enabled simultaneously and where firing one disables the other. We can formalise this choice with the following notation.

Definition 6.1 (Choice). *Let t and t' be two transitions in the underlying PN of the STG. Then we write $t \rightharpoonup t'$ if at some reachable marking, both t and t' are enabled and firing t' disables t.*

However, OP violations in the STG represent a stronger property than simply the existence of such choices. More specifically, for some choice $t \rightharpoonup t'$ to be classed as an OP violation in the STG at a reachable marking, the following additional conditions are required:

(1) t and t' must be transitions of different signals;

(2) at least one transition must be labelled with a local signal;

(3) the signal of t must be disabled when t' fires.

Condition (1) is obvious, but (2) is more interesting, as it rules out choices between two input transitions. This is because such choices model a *non–deterministic choice in the environment* (for instance, a non–deterministic increase or decrease in temperature), and so does not need to be implemented by the system itself. Thus, SI circuits can still be obtained from STGs with input choices, as long as the other conditions are also met.

Even if $t \multimap t'$ satisfies conditions (1) and (2), it may still not be an OP violation if it does not satisfy condition (3). To explain this, consider a transition a^+ of some signal a. If a^+ is disabled by another transition of a different signal, then it is possible that some other enabled transition $a^+/1$ was not disabled, or that the new marking enables $a^+/1$. In both cases, signal a persists, and so an observer will not see it drop. Based on this, we can now formalise an OP violation.

Definition 6.2 (OP violation). *A choice $t \multimap t'$ at some reachable marking M is an OP violation, denoted $t \overset{OP}{\multimap} t'$, if: (i) t and t' are labelled with different signals; (ii) at least one of those signals is local; and (iii) firing t' disables the signal of t.*

Biologically, an OP violation can represent some missing information regarding the outcome of two competing reactions. This can therefore provide a crucial first step to the biologist in identifying precisely the information that should be gathered through further experimentation.

It can be observed that the STG in Figure 6.6 respects OP, since there are no choices involving local transitions. As this STG also respects boundedness, consistency and CSC, we therefore say that it is SI in the environment specified (recall that this is the most relaxed one). However, suppose we consider a slightly more demanding environment in which *Signal* can drop before *SigF* has been raised. This environmental behaviour is shown in Figure 6.7.

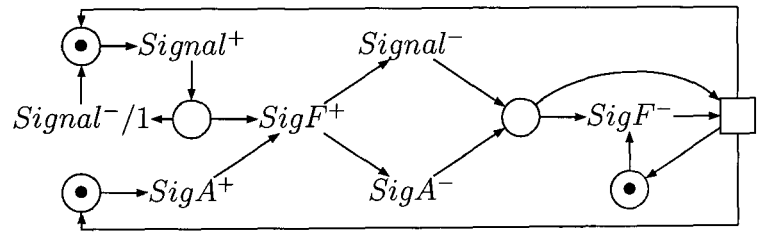


Figure 6.7: STG modelling bacillus-circuit in more demanding environment with an OP violation.

Now we see that if both *Signal* and *SigA* are raised, $SigF^+$ becomes enabled as before. But, the input transition $Signal^-/1$ is also now enabled, and if the environment

is faster than the circuit, $Signal^-/1$ can fire before the circuit has had time to raise $SigF$. We therefore have an OP violation $SigF^+ \overset{OR}{\rightarrow} Signal^-/1$.

Methodologically, OP violations can be automatically identified, but their resolution requires input from the user (we will discuss both of these issues later). In particular, two types of OP violation need to be considered: violations involving both input and local transitions; and violations involving only local transitions (recall that choices involving only inputs are not OP violations). The former is usually straightforward to resolve by assuming that the environment is slow enough for the circuit to stabilise. The latter is not so straightforward, since they require extra knowledge of the reaction rates, which are not always available. Nevertheless, such violations can be used to identify precisely the information required by the biologist, who can then focus on obtaining it through experimentation.

In some cases, it may not always be possible to resolve the latter if they represent some truly stochastic phenomenon, such as the lysis-lysogeny switch in phage λ (see Section 6.5). One should therefore leave such violations in the model, thus representing the stochasticity as a non–deterministic choice, and document this decision. Note such choices can still be handled in an SI manner using *arbitration* [55], but we will not elaborate on this here.

An STG is therefore implementable as an SI circuit (and thus, by our methodological assumption, a biological system) if it is bounded, consistent, has CSC and is OP. For example, the STG shown in Figure 6.6 can be implemented by the SI circuit $[SigF] = Signal\ SigA$, which, as previously mentioned, is called an AND-gate.

6.2.4 Additional properties

There are a number of other properties which are not directly required for SI, but whose violation is nevertheless suspicious and should be documented. At least, such properties can be used to identify potential model construction errors or indicate missing information.

Deadlock-freeness

A *deadlock* (see Section 2.3.3) occurs when no transitions are enabled at some reachable marking. Deadlocks indicate that the system can stop functioning, which in most cases is probably not an intended behaviour. Our SI STG in Figure 6.6 is free from deadlocks.

No self-triggering

A signal a is called *self-triggering* if firing one of its transitions (say) a^+ can immediately enable another of its transitions (say) a^-. Self-triggering can yield the same effects as OP violations (i.e. non-digital pulses) as well as cause CSC conflicts (note all the signals have the same values before firing the first transition and after firing the second one). In the context of biological systems, self-triggering could indicate some missing auxiliary internal entities whose transitions would separate the pair of transitions involved in self-triggering. Our SI STG in Figure 6.6 does not contain self-triggering.

Divergency-freeness

If infinitely many internal transitions of an STG can be executed from some reachable marking, then it is said to have *divergency*. Biologically, divergency can be interpreted as some infinite unproductive process in the system which, nevertheless, consumes resources. Our SI STG in Figure 6.6 respects the divergency-freeness property.

Note the properties introduced throughout this section can be automatically checked for using STG tools such as PETRIFY [54].

6.3 Constructing SI STGs from circuits

We now consider applying these analysis techniques to more practical modelling situations. In particular, we assume that we start with a circuit specification of a biological system - a BN derived by some means, such as from time series data - and show how an STG can be constructed to capture its behaviour. We then discuss how violations of SI, which, according to our methodological assumption, represent unrealisable behaviour, can be identified in the STG and refined.

6.3.1 Translating circuits into STGs

As introduced earlier, the bacillus-circuit should raise $SigF$ (to represent sporulation) only when $SigA$ and $Signal$ are high, and should drop $SigF$ (to represent vegetative growth) when $SigA$ or $Signal$ are low. This behaviour was captured by the STG in Figure 6.6 (which is SI in the environment specified), but can also be described by the state transition table in Figure 6.8, which represents the circuit specification that one would normally start with in practice.

$SigA$	$Signal$	$SigF$	$[SigF]$
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

Figure 6.8: State transition table for bacillus-circuit representing next-state function of $SigF$, which after logic minimisation, can be described by the next-state equation $[SigF] = SigA\ Signal$.

By applying the logic minimisation techniques discussed in Section 3.1 to this table,

we obtain the next-state equation $[SigF] = SigA\ Signal$ which directly corresponds to the logic of our SI STG from Figure 6.6. However, further study reveals one crucial difference: *it is impossible to deduce the environmental behaviour from the equation alone* (indeed, we have already seen an environment in Figure 6.7 where this circuit is not SI). In short, the environmental behaviour shown in Figure 6.6 has somehow been lost. This therefore exemplifies the key advantage of STGs in capturing *both* the circuit and the environment, and shows that having an STG specification can be much more useful for analysing a system than the circuit definition alone.

We therefore want to be able to translate this circuit into an STG to capture these environmental assumptions naturally. In fact, it turns out that any digital circuit can be modelled as an STG using a similar approach to the asynchronous PN construction of Definition 3.1, which we call the *circuit-STG* construction.

Definition 6.3 (The circuit-STG construction). *Let $\mathcal{BN} = (G, N, F)$ be a circuit. Then an STG can be constructed from $\mathcal{BN}$ as follows:*

- *Each signal (i.e. regulatory entity) $g_i \in G$ is represented by two places g_i and $\overline{g_i}$ which indicate whether the entity is active or inactive, respectively. Exactly one of these places is marked at any time.*

- *Since we do not have any information about the environment's behaviour, it is taken to be the most general, i.e. it can always change the value of any input. This is modelled for each input signal g_i by adding transitions g_i^+ (consuming a token from $\overline{g_i}$ and depositing a token on g_i) and g_i^- (consuming a token from g_i and depositing a token on $\overline{g_i}$).*

- *For each local signal g_i the circuit computes the next-state value $[g_i]$ of g_i using the corresponding minimised Boolean equation $[g_i] = E_i$ (e.g. $[SigF] = SigA\ Signal$ for our running example). For each term m_j in the minimised disjunctive normal form of $E_i|_{g_i=0}$ (where $E_i|_{g_i=b}$ denotes the Boolean expression resulting from substituting g_i by $b \in \{0,1\}$ in E_i), we add a transition g_i^+/j which switches on g_i. We add an arc from place $\overline{g_i}$ to g_i^+/j and from g_i^+/j to place g_i which switches on g_i. Then, for each g_k (resp. $\overline{g_k}$) occurring in m_j, we connect g_i^+/j to the place g_k (resp. $\overline{g_k}$) by a pair of arcs going in opposite directions (to model testing for the presence of a token without consuming it). We use a similar process to define the transitions g_i^-/j which reset g_i based on $\overline{E_i|_{g_i=1}}$.*

Using this circuit-STG construction process, we can translate the next-state equation $[SigF] = SigA\ Signal$ into the STG shown in Figure 6.9. One can see that as the behaviour of the environment could not be derived from the circuit, the most general environment was modelled. As such, the STG in Figure 6.9 strictly includes the behaviour of our manually constructed STG in Figure 6.6. However, now the STG captures too much behaviour which we argue to be unrealisable in practice. Thus, the next step is to apply the properties of SI circuits discussed in the previous section (i.e. boundedness, consistency, CSC and OP) to identify areas of the STG model where more information is required, and where this unrealisable behaviour can be removed.

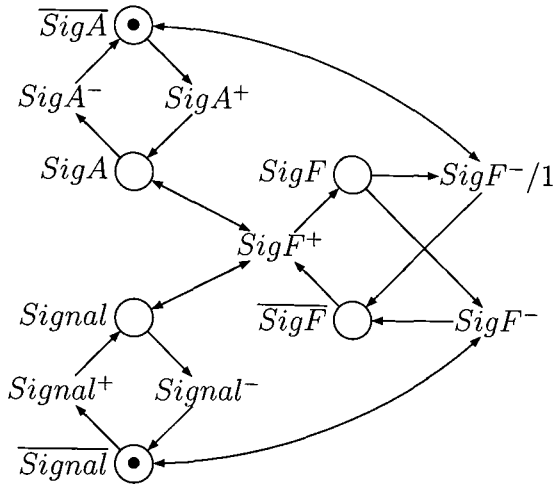


Figure 6.9: STG constructed by the circuit-STG process modelling the most general environment in which *Signal* and *SigA* can oscillate freely. This STG respects boundedness, consistency and CSC, but does not respect OP.

It turns out that STGs constructed from circuits using the circuit-STG process inherently respect boundedness, consistency and CSC, and so do not require checking for these properties. In fact, STGs constructed by the circuit-STG process respect some stronger properties: they are always *safe*, a strong boundedness property where the respective bound is one; and they respect *universal state coding*, a stronger property than CSC, requiring that no two different states have the same values of all the signals (note there is a one-to-one correspondence between the reachable markings and encodings of an STG derived from a circuit). In addition, STGs with at least one input signal are always free from deadlocks, since these inputs can oscillate freely.

However, OP is not always respected by the circuit-STG construction process, as the extra environmental behaviour captured can cause violations involving input and local transitions (of course, this is in addition to any violations between local transitions). We therefore require a means of identifying OP violations in an STG model so that the unrealisable behaviour can be removed.

6.3.2 Identifying output-persistency violations

The identification of OP violations for modest-sized STG models can often be achieved straightforwardly by visual inspection. For example, one can see that the STG in Figure 6.9 has the following OP violations between input and local transitions (remember, these are now present because the most general environment was modelled):

$$SigF^-/1 \stackrel{OP}{\prec} SigA^+, \quad SigF^- \stackrel{OP}{\prec} Signal^+, \quad SigF^+ \stackrel{OP}{\prec} SigA^-, \quad SigF^+ \stackrel{OP}{\prec} Signal^-.$$

For larger practical STG models, however, it is clear that we require automated techniques for such identification. Indeed, PETRIFY provides this functionality, but does

not appear to identify the actual transitions involved. To address this, let us start with an STG model $\mathcal{STG}$, which has been constructed from a circuit using the circuit-STG process from Definition 6.3. We know that $\mathcal{STG}$ respects boundedness, consistency, CSC and deadlock-freeness, but that it does not necessarily respect OP. We must therefore check $\mathcal{STG}$ for OP violations, and we do this by searching through each possible reachable marking $M \in RM(M_0)$ of $\mathcal{STG}$ from some initial marking M_0.

A model checking problem:

We formulate this search as a model checking problem. The idea is that at each marking M reachable from M_0, there will be one or more enabled transitions (we know this since STGs constructed from circuits are deadlock free) which, when fired, will yield a new marking M'. Therefore, at each marking M, we fire each enabled transition t in turn to give a new marking M'. Then, by inspecting which transitions t' have been disabled at M', one can identify all the choices in $\mathcal{STG}$. Of course, just because some transition t' has been disabled by the firing of some t, it does not mean that $t' \rightharpoonup t$ is an OP violation (see Definition 6.2). However, if $t' \rightharpoonup t$ is a violation of OP, then we record the choice. This process is then repeated for the other enabled transitions at marking M before the next marking $M \in RM(M_0)$ is considered (i.e. we do a breadth-first search of the reachability graph).

Clearly, this is a somewhat exhaustive approach for identifying OP violations in an STG, since an exponential number of markings must be checked. Whilst the development of more efficient approaches is outside the scope of this chapter, some improvements are worth mentioning:

- The first improvement can be made by noting that choices between transitions can be identified structurally; that is, two transitions which share some common pre-places. Whereas not all structurally-identifiable choices are actually realisable once tokens are introduced, their discovery will nevertheless be useful in reducing the search space for OP violations.

- A second improvement could be to use *binary decision diagrams (BDDs)* [34], which provide a compact and canonical representation of the state graph.

- Another improvement would be to use model checking approaches [109] which work on the unfolding of the STG (which is often exponentially smaller than its reachability graph). In such a case, a Boolean expression formulating the conditions for OP violations could be used for their efficient identification.

OP violations highlight the critical points in the STG model where there is uncertainty about the outcome of choices. The STG is therefore not SI, and thus captures behaviour which is unrealisable in practice according to our methodological assumption. With this in mind, we now focus on the resolution of OP violations and the derivation of an SI STG.

6.3.3 Resolving output-persistency violations

Whilst the identification of OP violations can be automated, their resolution requires user-given information, since it is impossible for a tool to derive this from the circuit. For example, the user may know that the environment is slow enough to allow the system to stabilise if the violation is between a local and input transition, or they may have some knowledge of reaction rates if the violation is between local transitions only. In practice, measuring reaction rates is a very effort-consuming task, but our approach addresses this issue by requiring only relative rates; that is, it is sufficient to know that one reaction is faster than another.

The approach we take to solving OP violations is to give priority to one of the transitions in the violation, so that it will always fire before the other in situations where they are both enabled. Of course, which transition is given the priority must be specified by the user. We therefore formalise such priority assumptions with the following notation.

Definition 6.4 (Transition priority). *Let $t \overset{OP}{\rightarrow} t'$ be an OP violation. Then we write $t \mapsto t'$ to denote the fact that, when both t and t' are enabled simultaneously, priority should be given to t.*

Note that since t is disabled by t' (but not the other way around) in the case where $t \rightharpoonup t'$, then assuming $t \mapsto t'$ will still allow t' to fire afterwards (provided that it is not disabled by t). For example, we can now formally express a slow environment for the STG in Figure 6.9 as follows:

$$SigF^-/1 \mapsto SigA^+, \quad SigF^- \mapsto Signal^+, \quad SigF^+ \mapsto SigA^-, \quad SigF^+ \mapsto Signal^-.$$

With this, we now discuss how such priority assumptions can be enforced as a *transformation* of the STG. In particular, we develop an initial approach, and then demonstrate with a counter-example a special case in which it can fail to resolve the violation. To address this shortcoming, we extend the definition of this transformation so that it is *commutative*, and formally prove that this property holds. Note in a slight abuse of notation, we will use $t \mapsto t'$ to denote both the priority assumption and the corresponding transformation in the STG.

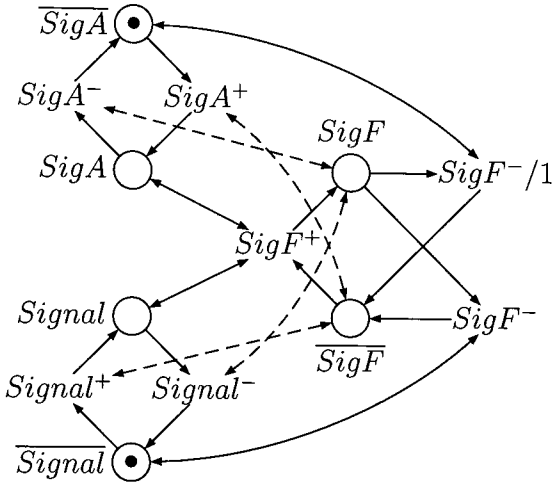
(1) A first approach

Lets assume that we want to express the priority $t \mapsto t'$. The basic idea is to replicate the transition with lower priority (in this case t') to capture each situation in which t is not enabled and t' can fire safely. We therefore formalise this as the *firing order enforcement (FOE)* transformation.

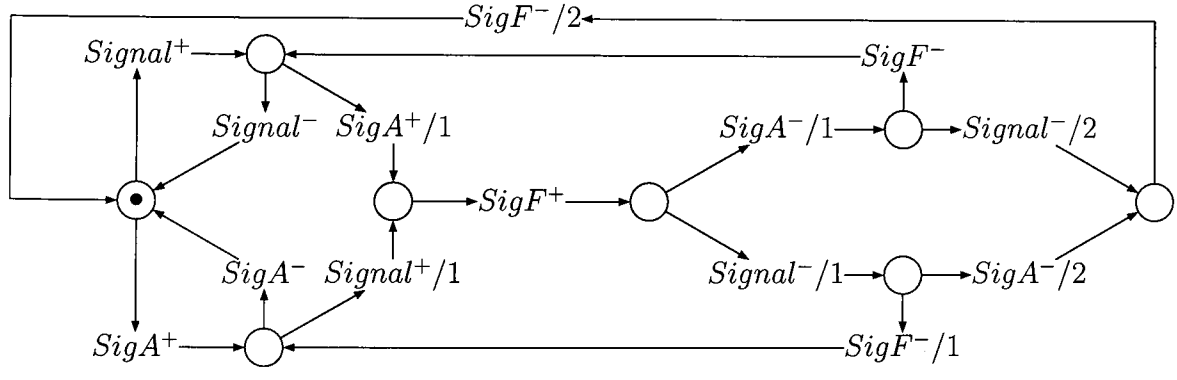
Definition 6.5 (Firing order enforcement (FOE) transformation). *Suppose $t \mapsto t'$ has been assumed and let $\{p_1, \ldots, p_n\} = {}^\bullet t \setminus {}^\bullet t'$. If $n = 0$ then t is enabled whenever t' is, and so t' along with all incident arcs can simply be removed from the STG, as it can never fire under the assumption $t \mapsto t'$. Otherwise, t' is replicated $n-1$ times so that there are n copies (denoted $t'_1 = t', t'_2, \ldots, t'_n$) of t' in total. All such replicas are labelled by the same signal as t' and have exactly the same connections. Furthermore, a read arc is*

added between t'_i and $\overline{p_i}$ for each $i = 1, \ldots, n$, where $\overline{p_i}$ is $\overline{g_j}$ if p_i corresponds to g_j, and g_j if p_i corresponds to $\overline{g_j}$. The FOE transformation guarantees that: (i) if t is enabled by some marking M, then none of $t'_1, \ldots, t'_n$ are enabled; and (ii) if t is not enabled by some marking M, but t' is enabled by M in the original STG, then at least one of $t'_1, \ldots, t'_n$ is enabled in the modified STG.

In other words, if $t \mapsto t'$ has been assumed, then the FOE transformation will resolve the choice between t and t' to favour t. For example, we can enforce the slow environmental assumptions specified above to the STG in Figure 6.9, resulting in the STG shown in Figure 6.10(a), where the additional read arcs (shown as dashed arcs) specify these environmental assumptions. This STG can then be simplified using PETRIFY [54], which returns the STG shown in Figure 6.10(b).



(a)



(b)

Figure 6.10: (a) STG from Figure 6.9 with slow environment assumptions, and (b) STG after simplification with PETRIFY.

Having resolved the four OP violations using the FOE transformation, the STG in

Figure 6.10(b) is now SI. Unsurprisingly, it contains less behaviour than the STG in Figure 6.9, since our assumptions have been used to remove unrealisable environmental behaviour. Interestingly though, it captures more behaviour than the manually-constructed STG in Figure 6.6. In particular, it can be observed that it is now more robust to the behaviour of the environment, since it poses fewer constraints on it. For example, the environment is now allowed to oscillate *Signal* and *SigA* from the initial marking, but once both have been raised, they are assumed to remain high so that the circuit can raise *SigF* (thus avoiding an OP violation). Similarly, once *SigF* has dropped as a result of *either Signal* or *SigA* dropping, the environment is now allowed to raise the corresponding signal again.

Intuitively, resolving OP violations is equivalent to the removal of arcs in the *state graph* [55] of the STG. The state graph is closely related to the reachability graph, but captures signal changes as opposed to transition firings. For example, the state graph for the STG in Figure 6.9 (modelling the most general environment) is shown in Figure 6.11(a), where the initial state $Signal = 0, SigA = 0, SigF = 0$ is highlighted. Now, consider OP violation $SigF^-/1 \stackrel{OP}{\prec} SigA^+$, which can be seen at state 101; *SigF* can drop at this state, but if *SigA* rises then this is prevented. Resolution of this violation is therefore equivalent to removing the outgoing arc $SigA^+$ from state 101. Solving the remaining violations therefore gives us the state graph in Figure 6.11(b), which corresponds to the STG in Figure 6.10(b).

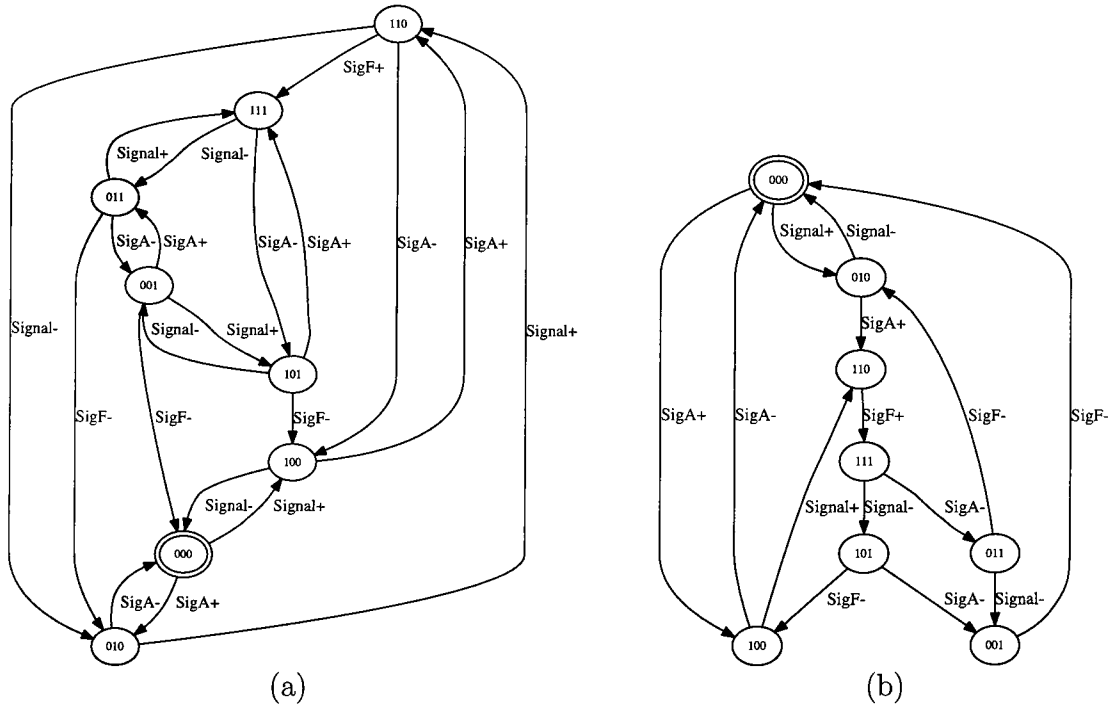


Figure 6.11: (a) State graph for the STG in Figure 6.9 modelling most general environment, with initial state $Signal = 0, SigA = 0, SigF = 0$ highlighted, and (b) state graph for SI STG in Figure 6.10(b).

This intuitive view of arc removal fits with our understanding that OP resolution removes unrealisable behaviour. In addition, since we clearly obtain the same state graph irrespective of the order in which the arcs are removed, then the order in which the violations are solved should also not matter - a property known as *commutativity*. It therefore follows that the FOE transformation on the STG should be commutative; the STG after solving a set of OP violations in any order should be the same. However, the following counter-example shows that this is not always the case.

Counter-example:

Consider the STG shown in Figure 6.12(a), which has the following OP violations: $b^- \stackrel{OP}{\prec} a^-$, $c^- \stackrel{OP}{\prec} a^-$ and $c^- \stackrel{OP}{\prec} d^-$. Suppose we look at resolving the first two viola-

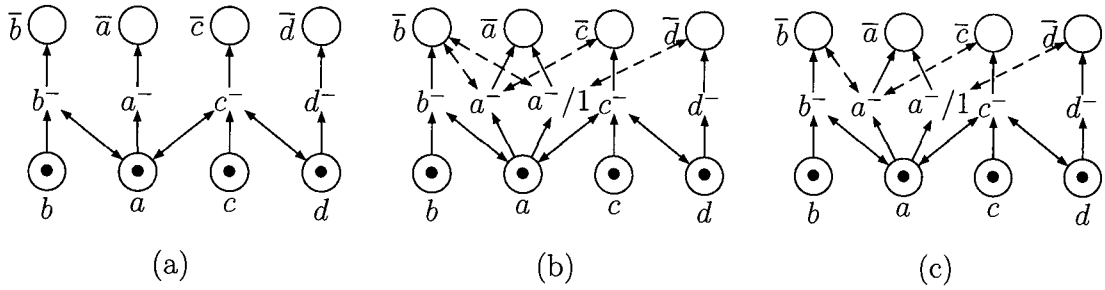


Figure 6.12: (a) STG with OP violations $b^- \stackrel{OP}{\prec} a^-$, $c^- \stackrel{OP}{\prec} a^-$ and $c^- \stackrel{OP}{\prec} d^-$, (b) resolving with priorities $b^- \mapsto a^-$ then $c^- \mapsto a^-$, and (c) resolving $c^- \mapsto a^-$ then $b^- \mapsto a^-$.

tions in both orders using the FOE transformation. If we apply assumptions $b^- \mapsto a^-$ then $c^- \mapsto a^-$, a read arc from $\overline{b}$ is added to a^- (to resolve the first violation), and then a^- (and the new arc) is replicated to form new transitions a^- and $a^-/1$ which capture the two conditions for resolving the second violation. This STG with the additional dashed arcs is shown in Figure 6.12(b). On the other hand, if we apply assumptions $c^- \mapsto a^-$ then $b^- \mapsto a^-$, we replicate a^- to form new transitions a^- and $a^-/1$ which capture the two conditions for resolving the second violation, and then add the read arc from $\overline{b}$ to a^- only. This different STG is shown in Figure 6.12(c), and one can clearly see that it is still possible for $a^-/1$ to fire before b^-, thus preserving the original OP violation.

The STG in Figure 6.12(b) correctly captures the slowness of a^-, whereas the STG in Figure 6.12(c) does not. Thus, there appears to be a discrepancy between the state graph view of arc removal and the FOE transformation view of transition prioritisation which clearly requires addressing for the correct resolution of OP violations. This discrepancy can be explained by noting that arc removal in the state graph works at the signal level, whereas the FOE transformation works at the transition level. Thus, for the FOE transformation to work at the signal level also, we propose a possible extension to its definition to make it *commutative*.

(2) A commutative approach to firing order enforcement

A commutative extension to the FOE transformation can be made by introducing the notion of a *transition group*; a 'virtual' set used for book-keeping which simply enables replicated transitions from the same originator to be treated as a logical unit. Each transition t is associated with such a group, denoted $TG(t)$, which initially contains only t itself. The idea then is that when t is replicated during some transformation, the replicas are added to this group, so that future transformations consider all transitions and do not miss any (as observed above).

For example, consider once again the STG from Figure 6.12(a). Initially, the transition group of all four transitions simply includes the one transition, e.g. $TG(a^-) = \{a^-\}$. Suppose we solve violation $b^- \overset{OP}{\prec} a^-$ first. As before, this only involves the addition of a read arc between a^- and $\overline{b}$. Solving violation $c^- \overset{OP}{\prec} a^-$, however, involves the replication of a^- to capture the two conditions under which a^- can fire (and c^- cannot), and so we create a replica transition $a^-/1$ and add this to the group $TG(a^-)$. We then add a read arc between a^- and $\overline{c}$ and a read arc between $a^-/1$ and $\overline{d}$ as before, resulting in the correct STG shown in Figure 6.12(b).

On the other hand, if we resolve $c^- \overset{OP}{\prec} a^-$ first, we replicate a^- to give us a^- and $a^-/1$, and then add $a^-/1$ to $TG(a^-)$. Read arcs between a^- and $\overline{c}$, and between $a^-/1$ and $\overline{d}$ are then used as before. Then, when we resolve $b^- \overset{OP}{\prec} a^-$, we consider *both* transitions in $TG(a^-)$, and add a read arc to place $\overline{b}$ for both of them. Again, this results in the correct STG shown in Figure 6.12(b).

We now formalise this extended transformation, which we refer to as the *transition group firing order enforcement (TGFOE)* transformation.

Definition 6.6 (TGFOE transformation). *Suppose $t \mapsto t'$ has been assumed and let $TG(t')$ be the transition group of t'. Then, we apply the FOE transformation to all $t'' \in TG(t')$, and add any new replicated transitions to $TG(t')$.*

The TGFOE transformation essentially creates the cartesian product of conditions for the slower transition to fire, and thus its application for a given set of OP violations will result in the same STG regardless of ordering. We show that this commutativity property holds more formally with the following theorem.

Theorem 6.1. *The TGFOE transformation is commutative.*

Proof. It suffices to consider two transformations T_1 and T_2. Then, we need to show that the resulting STG after applying T_1 and T_2 in both orders is the same. Specifically, there are two cases that should be considered:

(1) Assume T_1 and T_2 involve transforming two different transitions. This is trivial, and does not lead to the problems highlighted previously by our counter-example, since the two transitions are independent of one another. It is straightforward to see that their application is therefore commutative.

(2) Assume T_1 and T_2 involve transforming the same transition t. Let T_1 enforce the priority assumption $t_a \mapsto t$ and T_2 enforce the priority assumption $t_b \mapsto t$, where $\{p_{a_1}, \ldots, p_{a_j}\} = {}^\bullet t_a \setminus {}^\bullet t$ and $\{p_{b_1}, \ldots, p_{b_k}\} = {}^\bullet t_b \setminus {}^\bullet t$. We need to consider a further two cases:

(i) Let $j = 0$ or $k = 0$. Here, t_a or t_b are enabled precisely when t is, respectively. Thus, t will not be able to fire under either timing assumption, and so can be removed from the STG, along with incident arcs. Furthermore, since for each $t' \in TG(t)$, we have that ${}^\bullet t' \cap {}^\bullet t \neq \emptyset$ by definition, then these too will not be able to fire under either timing assumption, and so should be removed along with all incident arcs. Clearly, in either case, the resulting STG will be the same.

(ii) Let $j > 0$ and $k > 0$. Suppose $TG(t) = \{t_1, \ldots, t_m\}$. If we apply transformation T_1 first, then for each transition in $TG(t)$, we create $j - 1$ replicas with exactly the same connections, add these to $TG(t)$, and then link each to the corresponding place in ${}^\bullet t_a$ using read arcs. Thus, $TG(t)$ is now the set $TG(t) = \{t_1^1, \ldots, t_1^j, \ldots, t_m^1, \ldots, t_m^j\}$. Then, we apply T_2 by creating $k - 1$ replicas of each transition in $TG(t)$, adding them to $TG(t)$, and by linking them using read arcs to the corresponding places in ${}^\bullet t_b$. Thus, $TG(t)$ is now the set:

$$TG(t) = \{t_1^{1,1}, \ldots, t_1^{1,k}, \ldots, t_1^{j,1}, \ldots, t_1^{j,k}, \ldots, t_m^{1,1}, \ldots, t_m^{1,k}, \ldots, t_m^{j,1}, \ldots, t_m^{j,k}\}.$$

Similarly, applying transformation T_2 first then T_1 results in the transition group $TG(t)$ being:

$$TG(t) = \{t_1^{1,1}, \ldots, t_1^{1,j}, \ldots, t_1^{k,1}, \ldots, t_1^{k,j}, \ldots, t_m^{1,1}, \ldots, t_m^{1,j}, \ldots, t_m^{k,1}, \ldots, t_m^{k,j}\}.$$

Notice now that for every transition $t_r^{p,q} \in TG(t)$ after the first ordering, there is a corresponding transition $t_r^{q,p} \in TG(t)$ after the second ordering, with exactly the same incident arcs. Hence, the resulting STGs are the same.

Since we have shown that commutativity holds for any two TGFOE transformations, we have therefore shown that it holds for any sequence of TGFOE transformations, thus concluding our proof. □

From this, we observe that there is now a direct correspondence between the TGFOE transformation and the state graph, thus avoiding the problems identified by our counter-example for the FOE. In addition, the SI properties discussed in Section 6.2 which hold for STGs produced by the circuit-STG approach are preserved by the TGFOE transformation:

- boundedness and consistency are preserved, since the set of reachable markings can only be reduced by the transformation;
- since STGs constructed by the circuit-STG approach respect USC, and the TGFOE transformation preserves USC (as it can only eliminate reachable states and never adds new ones), then CSC is preserved also, since USC implies CSC;
- deadlock-freeness is preserved, since TGFOE can only disable some (but never all) of the transitions enabled at a reachable state.

In the next section, we go on to look at some of the interesting properties that emerge when violations are resolved.

6.3.4 Properties of violation resolution

So far, we have looked at the construction of STGs from circuits, the automatic identification of OP violations and their correct resolution. We now consider some of the interesting phenomena that can arise through OP violation resolution.

Introduction of new violations

In some cases, it is possible that resolving an OP violation can introduce one or more new ones. To illustrate this, let us consider a cut-down version of the STG in Figure 6.12(a), with the addition that signal b can oscillate, which we show in Figure 6.13(a) along with the corresponding state graph in Figure 6.13(b).

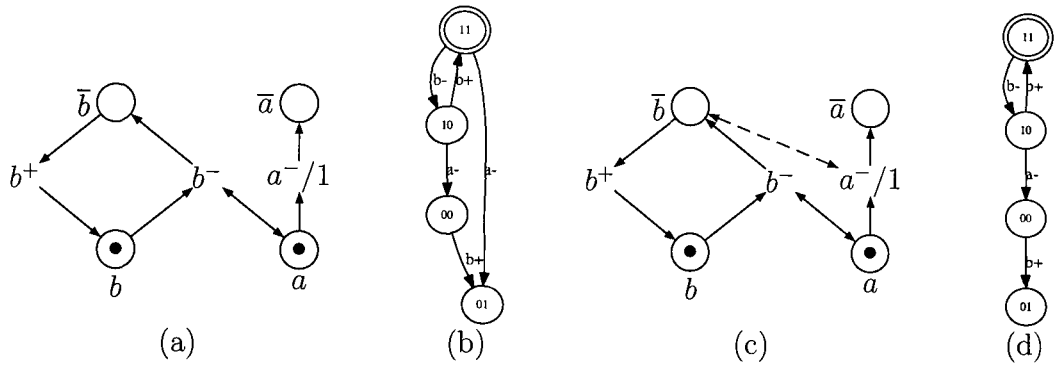


Figure 6.13: (a) STG with OP violation $b^- \stackrel{OP}{\prec} a^-$, (b) corresponding state graph, (c) new violation $a^- \stackrel{OP}{\prec} b^+$ introduced by resolving $b^- \stackrel{OP}{\prec} a^-$, and (d) corresponding state graph.

Initially, the only violation is $b^- \stackrel{OP}{\prec} a^-$. After resolution using the priority assumption $b^- \mapsto a^-$ under the TGFOE transformation, we get the STG shown in Figure 6.13(c), whose corresponding state graph is shown in Figure 6.13(d). The dashed read arc ensures that a^- can only fire once b^- has fired. However, now we notice that this added dependency on transition a^- has introduced a new OP violation $a^- \stackrel{OP}{\prec} b^+$; that is, a^- can now be disabled by the firing of b^+.

This phenomenon can also be observed at the state graph level. In Figure 6.13(d), this introduced violation occurs at state 10, where both b^+ and a^- can occur. If b^+ occurs first, however, then a^- cannot occur at the next state 11, whereas it could originally in Figure 6.13(b) before we prioritised b^-.

This result is intuitive, as resolving violations involves the insertion of read arcs which can add more dependencies. Further work is therefore needed to investigate whether such phenomenon should be avoided or not, and what this could mean biologically.

Implicit resolution

We have so far considered the explicit resolution of OP violations by enforcing some priority into the STG. However, in some cases it is possible that the subsequent implicit information added to the STG from the priority enforcement can resolve other violations. For example, consider the STG in Figure 6.14(a) with three signals, where a and c are

outputs and b is an input. Initially, this STG has three OP violations: $a^+ \stackrel{OP}{\prec} b^+$; $c^+/1 \stackrel{OP}{\prec} b^+$; and $c^+ \stackrel{OP}{\prec} a^-$.

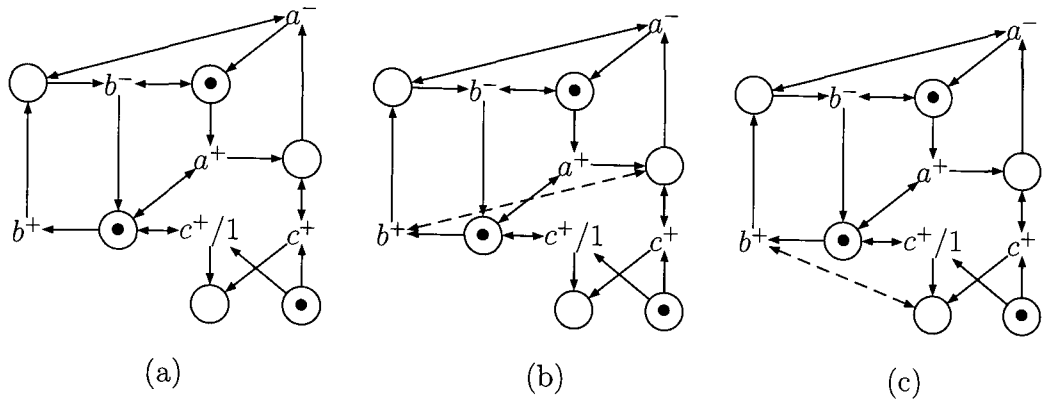


Figure 6.14: (a) STG with OP violations $a^+ \stackrel{OP}{\prec} b^+$, $c^+/1 \stackrel{OP}{\prec} b^+$ and $c^+ \stackrel{OP}{\prec} a^-$, (b) application of TGFOE to resolve $a^+ \stackrel{OP}{\prec} b^+$, resulting in implicit resolution of $c^+/1 \stackrel{OP}{\prec} b^+$, and (c) application of TGFOE to resolve $c^+/1 \stackrel{OP}{\prec} b^+$, resulting in implicit resolution of $c^+ \stackrel{OP}{\prec} a^-$.

Suppose we resolve $a^+ \stackrel{OP}{\prec} b^+$ using the priority assumption $a^+ \mapsto b^+$ under the TGFOE transformation. This assumption is enforced using the dashed arc in Figure 6.14(b), which ensures that b^+ must wait for a^+ to fire. However, it turns out that by doing this, we also solve violation $c^+/1 \stackrel{OP}{\prec} b^+$, leaving only $c^+ \stackrel{OP}{\prec} a^-$ remaining.

The reason that this occurs is that in the original STG in Figure 6.14(a), transitions $c^+/1$ and b^+ are enabled from the initial marking, and it is possible for b^+ to fire first, thus disabling $c^+/1$ and causing the violation (note this is possible when transition c^+ is not enabled). However, in the refined STG in Figure 6.14(b), this behaviour cannot occur, because now b^+ must wait until a^+ has fired, by which time c^+ is now also enabled. Thus, when b^+ does fire, it still disables $c^+/1$, but c^+ is now enabled and so this is not an OP violation.

A similar phenomenon can also occur by resolving $c^+/1 \stackrel{OP}{\prec} b^+$ in the original STG with assumption $c^+/1 \mapsto b^+$ under the TGFOE transformation. The resulting STG is shown in Figure 6.14(c), and now we find that violation $c^+ \stackrel{OP}{\prec} a^-$ has also been resolved, leaving only $a^+ \stackrel{OP}{\prec} b^+$ remaining. This is explained by noting that a^- has to wait in the original STG for b^+ to fire before it can fire. In the refined STG, however, there is also an additional dependency enforced between b^+ and $c^+/1$, in that b^+ must wait until $c^+/1$ has fired (or alternatively c^+). The result is in some sense an implied transitive priority assumption $c^+ \mapsto a^-$, because a^- can only fire after b^+, which itself can only fire after c has been raised.

Intuitively, these phenomena arise when a transformation eliminates so much behaviour that other violations are resolved as a result. Interestingly, though, the three violations in Figure 6.14(a) are in some sense linked, in that resolving $a^+ \stackrel{OP}{\prec} b^+$ implicitly resolves $c^+/1 \stackrel{OP}{\prec} b^+$, which itself implicitly resolves $c^+ \stackrel{OP}{\prec} b^-$. There is therefore a notion of a dependency relationship between violations which requires further investigation. In particular, the ability to detect such dependencies and causal relationships could prove

useful to the modeller. For instance, it may be desirable to resolve violations which do not implicitly resolve others, as this will prevent more behaviour than necessary from being removed. These interesting considerations are left as an area of future work.

6.4 Implementation issues

The circuit-STG approach from Definition 6.3 has been implemented as an extension to GNAPN (see Section 3.2), facilitating the automatic construction of STGs as well as PNs. Note currently only BNs can be translated into STGs, although the extension to cater for MVNs is a relatively straightforward area of future work. In addition, we have implemented a tool STGTOOL, which reads the STGs produced by GNAPN, automatically identifies all OP violations and allows the user to resolve them interactively via an intuitive GUI. The development of STGTOOL was motivated by the apparent lack of existing tools for providing detailed information on OP violations and for resolving them in an automated fashion.

Once an STG has been loaded into STGTOOL, all OP violations are displayed in a list to the user, who can then interactively resolve them by specifying which transition should be given priority. Solved OP violations are displayed in a separate list for documentation purposes, and any new violations are dynamically calculated. Furthermore, STGTOOL provides a useful feature for allowing the user to undo previously solved violations, thus allowing them to backtrack through the solve history. Note this is implemented as a stack, and therefore the user can only undo resolved violations in the order they were solved.

The combination of STGTOOL and GNAPN represent a key contribution to the biological community in supporting the incremental development of realistic models within the PN framework. Both tools are available for academic use at *bioinf.ncl.ac.uk/gnapn*.

6.5 Case study: lysis–lysogeny switch in phage λ

In this section, we illustrate the STG modelling techniques introduced by developing an SI STG model of the GRN responsible for the lysis–lysogeny switch in phage λ [153]. Using the Boolean model presented in [206] as a starting point, we construct and refine an STG model of this GRN, utilising both the support tool PETRIFY [54] and STGTOOL presented in the previous section. The model is refined by finding the points where it violates the SI conditions (in particular, OP violations) and then applying appropriate assumptions about the environment's behaviour and relative reaction rates to resolve the associated hazards. Since the lysis-lysogeny decision is a stochastic phenomenon, it is not resolved and remains in the final SI model (recall that this can still be handled in an SI way using arbitration [55]).

6.5.1 Model construction

The temperate bacteriophage λ [153] is a virus which infects the bacteria *E. coli*, and has been studied extensively in the literature. After infection of the host cell, a stochastic decision is made by λ based on environmental factors between two very different methods of reproduction, namely the *lytic* and *lysogenic* cycles [206]. In most cases, λ enters the lytic cycle, where it generates as many new viral particles as the host cell resources allow. Upon resource depletion, an enzyme is used to break down and lyse the cell wall, releasing the new phage into the environment. Alternatively, the λ DNA may integrate into the host DNA and enter the lysogenic cycle. Here, genes expressed in the λ DNA, now a prophage, synthesise a repressor which blocks expression of other phage λ genes including those involved in its own excision. As such, the host cell, now a lysogen, establishes an immunity to external infection from other λ phages, and the prophage is able to lie dormant, replicating with each subsequent cell division of the host. A high-level pictorial representation of this GRN is presented in Figure 6.15(a).

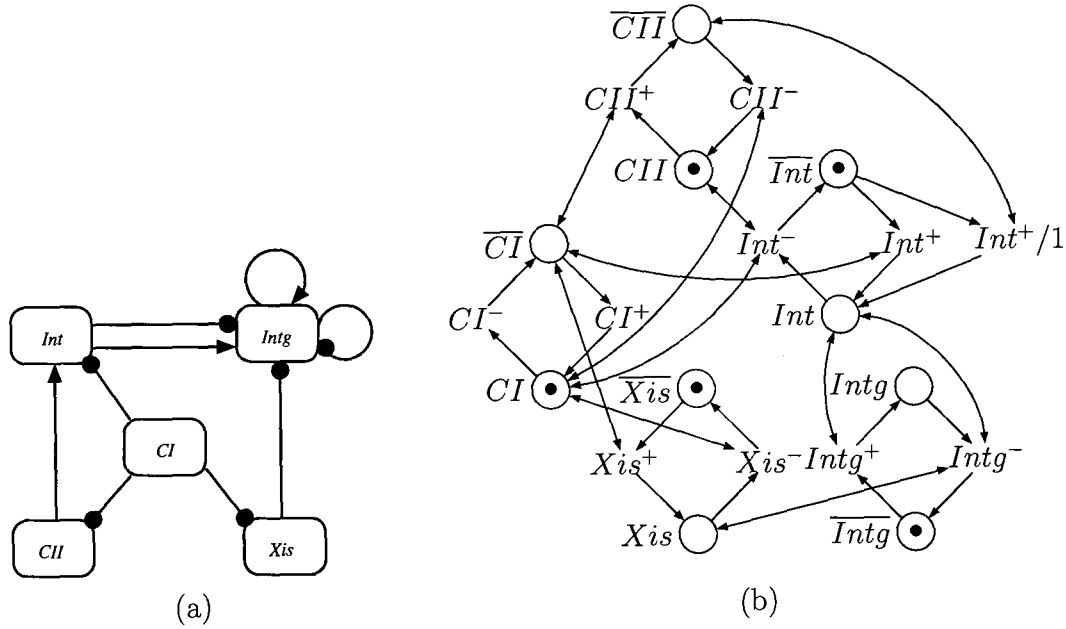


Figure 6.15: (a) A high-level representation of the GRN of the phage λ switch, and (b) STG automatically constructed by GNAPN from next-state equations using circuit-STG approach.

Integration of the λ DNA into the host DNA requires the presence of the integrase *Int*. Furthermore, the λ DNA remains integrated unless the excisionase *Xis* is also present. Thus, integration and excision occurs in both directions when both *Int* and *Xis* are present, and so the stochastic lysis-lysogeny choice is qualitatively modelled as a non-deterministic one [206]. The signal *Intg* is used as an output to indicate the status of this process, taking the value 1 if the λ DNA is integrated and 0 if it is not integrated or has been excised. Both *Int* and *Xis* are repressed by the λ repressor *CI*, which we regard as an input since it is regulated outside the scope of this model. However, *Int* is

also activated by CII, itself under negative control from CI. This additional control of Int therefore favours integration over excision [206].

The behaviour of each entity can be specified using the following set of Boolean next-state equations:

$$\begin{aligned} [CII] &= \overline{CI} \\ [Int] &= CII + \overline{CI} \\ [Xis] &= \overline{CI} \\ [Intg] &= \overline{Intg}\ Int + Intg\ (\overline{Int} + \overline{Xis}) \end{aligned}$$

From these equations, we can construct an STG describing the behaviour of the λ circuit based on the circuit-STG construction approach. We define CI as an input signal from the environment (and it is thus able to oscillate freely to model the most general one), $Intg$ as the output signal produced by the circuit, and CII, Int and Xis as internal signals which are invisible to the environment. Furthermore, we choose the initial state in which the values of all signals except CI are low. The construction of this STG is fully automated by our PN construction tool GNAPN, and is shown in Figure 6.15(b).

As explained previously, STGs constructed from the circuit-STG process are bounded (in fact, safe), consistent, deadlock-free and have CSC (in fact, USC), and these properties are preserved by the subsequent TGFOE transformations.

6.5.2 Model analysis and refinement

We now analyse our STG model with respect to the properties introduced in Section 6.2. We begin by running the model through PETRIFY, which shows, as predicted by our theory, that the STG satisfies boundedness, consistency, CSC and deadlock-freeness properties. However, it does not satisfy OP, resulting in non-deterministic behaviour which suggests that some of it may not be realisable in practice (note PETRIFY refers to violations of OP as *non-semimodularity*, but the two can be regarded as the same):

```
The STG has CSC.
Warning (non-semimodularity): signal CII disabled by signal CI
Warning (non-semimodularity): signal Int disabled by signal CI
Warning (non-semimodularity): signal Int disabled by signal CII
Warning (non-semimodularity): signal Xis disabled by signal CI
Warning (non-semimodularity): signal Intg disabled by signal Int
Warning (non-semimodularity): signal Intg disabled by signal Xis
***********************************************
* Warning: the STG is not output-semimodular     *
* Unable to generate a speed-independent circuit *
***********************************************
```

In order to identify precisely the transitions involved in the OP violations above, we then apply our support tool STGTOOL, which identifies the following:

(1) $Xis^+ \stackrel{OP}{\prec} CI^+$ (2) $Xis^- \stackrel{OP}{\prec} CI^-$ (3) $Int^+ \stackrel{OP}{\prec} CI^+$ (4) $Int^- \stackrel{OP}{\prec} CI^-$
(5) $CII^+ \stackrel{OP}{\prec} CI^+$ (6) $CII^- \stackrel{OP}{\prec} CI^-$ (7) $Intg^- \stackrel{OP}{\prec} Int^-$ (8) $Intg^- \stackrel{OP}{\prec} Xis^-$
(9) $Intg^+ \stackrel{OP}{\prec} Int^-$ (10) $Int^+/1 \stackrel{OP}{\prec} CII^-$

These violations of OP indicate the areas of the STG which require refinement with additional information about the environment's behaviour or relative reaction rates. We therefore proceed by considering OP violations (1)-(6) which involve choices between input and local transitions. Such choices can often be resolved by assuming that the environment is slow enough to allow the circuit to stabilise, and so we apply the following TGFOE transformations:

$$Xis^+ \mapsto CI^+,\ Xis^- \mapsto CI^-,\ Int^+ \mapsto CI^+,\ Int^- \mapsto CI^-,\ CII^+ \mapsto CI^+,\ CII^- \mapsto CI^-,$$

and these are shown by dashed arcs in Figure 6.16(a).

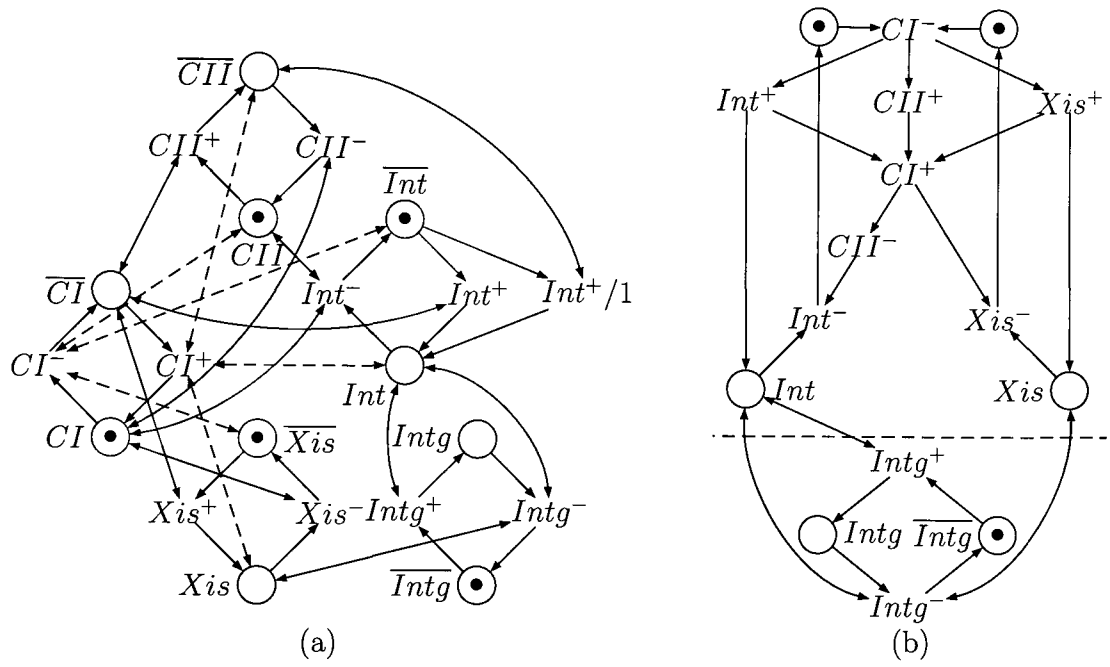


Figure 6.16: (a) STG from Figure 6.15(b) after TGFOE transformations (dashed arcs) expressing relative slowness of environment have been applied, and (b) SI STG from part (a) of this figure after simplification with PETRIFY.

Interestingly, it turns out that violation (10) is also resolved as a result of the above TGFOE transformations (see Section 6.3.4), leaving only violations (7)-(9) in the new model. Violations (7) and (8) show that excision (represented by the firing of $Intg^-$) when Int and Xis are high can be preempted if Int^- or Xis^- fires first, whilst violation (9) shows that integration (represented by the firing of transition $Intg^+$) can be preempted if Int^- fires first. These remaining OP violations represent the heart of the lysis-lysogeny switch in λ for our high-level model (which is a stochastic phenomenon in practice [206]),

and so are not resolved. The resulting STG is shown in Figure 6.16(b) after simplification with PETRIFY.

The new STG in Figure 6.16(b) is much less cluttered than the original one, as the unrealisable behaviour under the TGFOE transformations listed above has been stripped away, making it significantly simpler to interpret and analyse using e.g. model checking [51]. Moreover, this simplified STG clearly separates into two components, which capture the crucial mechanisms governing the lysis-lysogeny switch:

- Component 1 (top) involves the input signal *CI* and the internal signals *CII*, *Int* and *Xis*. From the initial stable state, it waits for the environment to lower signal *CI* indicating the absence of immunity, after which CII^+, Int^+ and Xis^+ can fire in any order. This component then waits for the environment to raise signal *CI*, resulting in the firing of transitions Xis^- and CII^- (in any order), with the latter followed by Int^-, which returns the component to its initial state.

- Component 2 (bottom) is a simple flip-flop for signal *Intg*, which is controlled by the values of the signals *Int* and *Xis* in the first component. Note that the only connections between the two components are read arcs between places of the former component and transitions of the latter one, i.e. the latter component accesses the former one in the read-only fashion and hence does not affect its behaviour.

After Component 1 has raised *Int*, transition $Intg^+$ is able to fire representing the integration of the λ DNA into the host cell. Once Component 1 has raised both *Int* and *Xis*, *Intg* can freely oscillate, i.e. there are no stable states in the absence of immunity [206]. Similarly, once the environment has raised *CI*, Component 1 executes Xis^- concurrently with CII^- followed by Int^-; the outcomes of the arbitrations between $Intg^+$ and Int^- and between $Intg^-$ and Int^- or Xis^- determine the stable state of signal *Intg* in the presence of immunity. These arbitrations exactly correspond to the OP violations (7)-(9) still remaining in the STG in Figure 6.16(b) and involve only local transitions.

Note that CII^- 'delays' Int^-, modelling that the presence of *CII* causes lambda to favour integration over excision; however, the latter is not a qualitative effect, and cannot in fact be formally derived neither from this STG nor from the equations in Figure 6.15(b) due to the arbitrary gate delays. In fact, one can see that *CII* can be removed from the model without affecting its qualitative behaviour, as its only role is to change the probabilities involved in the stochastic choice made by λ. Thus, it is no longer required once this stochastic choice has been qualitatively modelled by a non-deterministic one.

Finally, the output signal *Intg* in Figure 6.16(b) is self-triggering (note that the corresponding next-state equation is binate in *Intg*), and there is a divergency involving *Intg* (when *Int* and *Xis* are 1). This indicates that some auxiliary signal is missing from the model (which is not surprising due to its high level of abstraction), and so can provide a useful means of identifying areas which require careful documentation and further refinement in light of additional knowledge.

6.6 Discussion

6.6.1 Conclusions

In this chapter, we addressed the problem of unrealisable behaviour in asynchronous models by proposing a novel approach for model refinement using PN techniques based on STGs and SI circuits [55]. STGs are established in asynchronous circuit design for the specification of SI circuits, and are supported by a wide range of techniques and tools which do not appear to have been explored in biological modelling. In particular, the foundation of this work was the methodological assumption that biological systems can be qualitatively modelled by SI circuits, and that if this is not the case, then it suggests missing or erroneous information in the model.

We considered modelling asynchronous BNs using STGs, and used the properties of SI circuits to identify the key areas that required refining with additional information. A crucial advantage of STGs is that they enable *both* the behaviour of the environment and circuit to be captured naturally; we showed that this was essential for the derivation of SI circuits and that this could not be modelled by the circuit alone. In particular, we focused on the OP property for guaranteeing the robustness of the circuit under any gate delay, and showed how its biological interpretation could shed light on missing information regarding reaction rates. In general, few reaction rates affect the qualitative behaviour of a system, but these can nevertheless be difficult to identify by simple inspection of the circuit. This property is therefore of key importance to the biologist, as it represents precisely the points in the model which require further investigation. Specifically, the biologist can use violations of OP to identify only those rates which affect the qualitative behaviour of the model, and then conduct focused experimentation to shed light on these.

Existing studies in this area (for example, see [65, 70]) seem to have relied heavily on the manual construction of realistic asynchronous models. In contrast, we have presented a systematic formal approach which is amenable to practical models, by developing techniques for identifying and resolving OP violations based on model checking and priority transformations. These techniques have strengthened those available for STGs, and have represented a key contribution of our work. Indeed, the proposed extension to our PN construction tool GNAPN and development of STGTOOL have formed an insightful new modelling resource for the biological community, and offer an important link for exploiting established circuit design techniques and tools such as PETRIFY [54].

We demonstrated our approach to a model of the phage λ decision circuit [206]. We automatically constructed an STG model of this system, and this immediately made it amenable to SI analysis. PETRIFY was used to automatically check well-formedness properties such as boundedness, consistency and CSC, and then STGTOOL was utilised to find all the points in the model where OP was violated. As demonstrated by our theory, we were able to derive a more realistic STG which still captured the fundamental stochastic switch as a non–deterministic choice. In particular, this simpler STG split into two distinct components which tightly corresponded with our knowledge of the switching mechanism [206]. Interestingly, we were also able to observe some deeper insights regarding the fact that signal CII appeared to be redundant. Such an observation was not derivable from the next-state equations nor the STG due to arbitrary gate delays,

and thus represented a powerful type of simplification technique to complement analysis. This initial investigation has therefore demonstrated much promise for the application of STGs to realistic model development, and forms an interesting bridge between the fields of computing science, biology and circuit design.

6.6.2 Future work

A number of important developments are required to take this work forward. Firstly, we are interested in generalising our STG approach to handle MVNs. This is a relatively straightforward extension, but careful consideration will need to be paid to the consistency property.

In addition, we are interested at the prospect of utilising structural PN analysis techniques based on T-invariants to derive relative reaction rates. A study of this was reported by Popova-Zeugmann *et al.* [161] for bridging the gap between standard and timed PNs, but we see this as a promising complementary approach for our OP resolution techniques.

Further research into the dependencies between OP violations highlighted in Section 6.3.4 could provide many useful insights. In particular, we are interested in exploring methods for detecting such dependencies, for investigating their biological implications and for exploiting them during the resolution process.

Finally, we wish to investigate the application of these circuit design techniques to the area of *synthetic biology* [13,43,88]. Synthetic biology concerns the design of artificial biological systems which perform a certain task, and we see the application of STGs as a highly suitable approach for this. STGs could be utilised for designing complex biological systems in a compositional manner, where each sub-circuit is interfaced to its environment with input and output signals. In particular, the techniques presented could be used to develop a mechanism for testing each sub-circuit before it is "deployed" into its environment by identifying violations of SI. Furthermore, interfaces to databases such as *BioBricks* (see *http://bbf.openwetware.org*) could prove useful to facilitate the automatic construction, deployment and testing of such circuits using community-developed components.

6.6.3 Sources

The application of techniques from asynchronous circuit design was proposed by Victor Khomenko, along with the biological interpretation of properties for SI circuits and initial ideas for the FOE transformation. This work was presented at the 2nd International Meeting on Membrane Computing and Biologically Inspired Process Calculi in September 2008 [19].

Chapter 7

Concluding Remarks

7.1 Conclusions

In this thesis, we have set out to investigate, develop and evaluate the application of PNs for qualitatively modelling, analysing and refining GRNs. The focus on qualitative techniques has reflected the fact that quantitative approaches appear to be incommensurate with the quality and completeness of data currently available to the biological community. The PN formalism has formed the core of this work, owing to its numerous modelling advantages, including: (i) an *intuitive* graphical representation coupled with a well-studied formal mathematical semantics; (ii) an ability to capture *both* structure and behaviour in a concise and executable format; (iii) straightforward *extensibility*, by enabling extra information (such as time delays or probability) to be added whilst preserving the structure of the underlying net; (iv) modelling and analysis of systems at *multiple levels of detail*, from qualitative through to quantitative; and (v) a *mature* and well-developed base of theory, techniques and tool support.

We have systematically explored these key benefits with a specific application to GRNs. In particular, this research has resulted in the following key contributions:

(1) a new systematic PN approach for qualitatively modelling GRNs;

(2) a range of detailed case studies to demonstrate and validate our modelling approach;

(3) an investigation into the formal relationship between MVNs at different levels of abstraction;

(4) an approach for refining asynchronous Boolean models using PN techniques from asynchronous circuit design [55];

(5) a suite of integrated tool support to make these qualitative modelling and refinement techniques amenable to the biological community.

We will now discuss each of these key contributions in more detail.

(1) Systematic PN framework for qualitatively modelling GRNs

PNs have been applied to a wide range of qualitative and quantitative biological studies (see [215] for a bibliography), but these have generally required models to be manually constructed for the specific GRN under investigation, resulting in substantial modelling effort [45,155,160]. To improve the applicability and accessibility of PNs, well-developed systematic construction approaches were required, and this was addressed in Chapters 3, 4 and 6. We took a PN approach for modelling BNs proposed by Steggles *et al.* [194–196], and generalised it to cater for MVNs [146, 206]. In particular, this work extended similar approaches (for example, see [45,46]) in a number of important ways: (i) a range of PN representations for MVNs; (ii) the application of efficient *logic minimisation techniques* [146, 169] for compactness and scalability; (iii) modelling both synchronous and asynchronous MVNs in the PN environment; and (iv) coping with partial models in a useful way using non–determinism. Importantly, this work has contributed much-needed tool support for the biological community, thus forming an important link to powerful PN analysis techniques.

Our approach catered for three PN representations: safe, high-level and STG. Chapters 3 and 4 first considered a novel and compact safe PN framework which was then extended to exploit the expressive power of HLPNs. The key result was a visually-compact HLPN framework for MVNs which, interestingly, was also shown to be more efficient to analyse due to efficient unfolding techniques [111]. STGs formed a key focus and enabling technology for the development of new asynchronous model refinement techniques in Chapter 6, and were motivated by a crucial requirement for capturing both environmental and system behaviour. Indeed, this contribution offers unique and strengthened analysis prospects for asynchronous MVNs, by bridging the gap between the biological and circuit design communities.

In particular, this work has addressed three important considerations for MVN modelling in the PN framework:

- Scalability issues associated with realistic sized MVNs were addressed by exploiting efficient *logic minimisation techniques* [146,169] made available by the MVSIS tool [42]. Indeed, this formed an integral part of our model construction process, resulting in significantly more compact PNs which still captured the same behaviour, and which are more efficient to analyse. Although the potential for such techniques has been noted in the literature [46], no systematic applications appeared to have been reported for PN construction, and so this has formed an interesting contribution to the biological community.

- Both synchronous and asynchronous MVNs were catered for to reflect their wide interest in the literature [6, 83, 206, 208]. This proved to be challenging since PNs are asynchronous, but we addressed this in a compact and extensible way using novel control logic (in the safe PN case) and transition guard conjunction (in the HLPN case). This has therefore opened up the study of synchronous MVNs to PN techniques and tools; something which did not appear to have been addressed.

- Partial models of GRNs often occur in biology, and this is a key motivation for modelling them. Existing PN approaches in the literature appeared to have favoured

deterministic systems, but we showed how one of the most fundamental features of PNs – their non–deterministic firing semantics – could offer a useful solution. As such, our framework enables partial models to be documented, executed and refined in a meaningful way.

Tool support was crucial for making these techniques practical to realistic models. Our PN support tool GNAPN has made PN modelling of GRNs readily accessible to the biological community, and has strengthened the critical link between MVNs and the PN techniques and tools required to interpret them. As such, GNAPN formed an integral enabling technology for the case studies reported throughout this thesis, as well as some external ones (for example, see [178]).

(2) Application and validation of qualitative techniques using case studies

A number of detailed case studies have been presented throughout this thesis to demonstrate and validate our modelling techniques. In particular, Chapter 3 considered Boolean models for *E. coli* [167] and *B. subtilis* [99], Chapter 4 considered a multi-valued model for *E. coli* and Chapter 6 considered a Boolean model of the phage λ decision circuit [206]. All systems were well documented in the literature, thus representing good example GRNs to demonstrate and validate our PN techniques with.

GNAPN played an important link between the data and the PN tool community, and facilitated the study of practical models which would otherwise have been difficult to construct manually. In addition, PNs were shown to offer powerful analysis prospects for MVNs, as they are supported by advanced unfolding techniques [111] for coping with large state spaces. Combining GNAPN and these powerful techniques therefore formed an invaluable duo for gaining important insights *in silico*. One particular feature of GNAPN that we exploited in Chapters 3 and 4 was its initial marking capabilities, which enabled model checking to be performed over a constrained portion of the state space. This provided an extremely powerful proof mechanism for exploring whether global properties held, and formed the basis for mutant analysis experiments allowing us to investigate the robustness of our models under adverse conditions. Indeed, mutant analysis in the multi-valued *E. coli* study was the catalyst for Chapter 5, by identifying key behavioural differences with the Boolean representation.

The study on phage λ using STG techniques in Chapter 6 opened up a new avenue for PN modelling and analysis by linking biological studies with highly efficient circuit-theoretical techniques. This study made use of both GNAPN and STGTOOL as well as the circuit tool PETRIFY to incrementally develop a realistic model (based on the data from [206]). The key result from this case study was the identification and resolution of the missing relative reaction rates. This gave us a simpler STG which clearly separated into two components: the internal stochastic switch mechanism modelled as a non–deterministic choice; and the oscillating output based on this. Some deeper insights were also gleaned; in particular, the redundant role of *CII* in this switch. Such insights were not derivable from the STG nor the next-state equations due to arbitrary gate delays, and thus demonstrated a potentially powerful type of simplification technique for practical model development. STG techniques have therefore been shown as an invaluable tool for systematic model development, analysis and verification in the biological community.

(3) Relationship between models at different levels of abstraction

We noted some interesting behavioural differences between the Boolean and multi-valued *E. coli* models from Chapters 3 and 4, and this led us to investigate their formal relationship. In particular, it raised more general concerns about the scope and limitations of Boolean modelling, and the situations which required multi-valued expressiveness. Chapter 5 explored this by proposing a refinement theory which captured an assumption about this relationship. A number of insights were gleaned as a result. Notably, we showed an example motivating multi-valued over Boolean modelling in Theorem 5.2, where a single state for an entity performed two different interactions. Although valid in practical modelling, it nevertheless hinted at situations in which subtle behaviours could not be adequately represented by higher levels of abstraction. Indeed, more work is now needed to investigate problematic motifs of behaviour.

Furthermore, we identified the scope and limitations of analysing refinements derived from our theory. In particular, the results of Theorem 5.5 and Corollary 5.2 indicated that one could deduce a reasonable amount about an MVN using its refinements. However, it also showed that further developments were required for this work to address its major application area: circumventing state space explosion by deriving the simplest model which can answer the particular question of interest. For this, it will be necessary to explore different refinement theory notions to assess whether they provide a 'better' relationship for analysis, but 'better' needs to be biologically justifiable.

Finally, interesting insights into the relationship between the Boolean and multi-valued *E. coli* models were obtained. In particular, we deduced that the Boolean model was not a valid representation according to our theory, thus providing a possible explanation for the behavioural differences noted. Indeed, due to limitations with our algorithms, we did not answer the other key question concerning the existence of a valid alternative Boolean model, but this is clearly an important area of future work.

On one hand, this work has therefore developed what appears to be the first systematic framework for reasoning about the behaviour of MVNs at different levels of abstraction, and on the other it has provided the foundation for promising future developments in combatting state space explosion for practical modelling.

(4) Asynchronous qualitative model refinement techniques

Asynchronous BNs can be argued to be more biologically realistic than their synchronous counterparts [86]. However, such models have the problem of capturing too much information – some of which is unrealisable in practice - thus hampering analysis and interpretation. Chapter 6 addressed this by proposing a novel approach for model refinement using PN techniques based on STGs and SI circuits [55]. In particular, the foundation of this work was the methodological assumption that biological systems can be qualitatively modelled by SI circuits, and that if this is not the case, then it suggests missing or erroneous information in the model.

STGs are established in asynchronous circuit design for the specification of SI circuits, and are supported by a wide range of techniques and tools which did not appear to have been explored in biological modelling [54]. Furthermore, they enable *both* the behaviour

of the environment and circuit to be captured naturally; we showed that this was essential for the derivation of SI circuits and that this could not be modelled by the circuit alone. Existing studies in this area (for example, see [65, 70]) seemed to rely heavily on the manual construction of realistic asynchronous models. In contrast, we presented a systematic formal approach for identifying and resolving unrealisable behaviour based on output-persistency (OP) [55] violations.

We showed how the biological interpretation of such violations could shed light on missing information or stochasticity regarding reaction rates. In general, few reaction rates affect the qualitative behaviour of a system, but these can nevertheless be difficult to identify by simple inspection of the circuit. OP is therefore of key importance to the biologist, as it represents precisely the points in the model which require further investigation. Specifically, the biologist can use violations of OP to identify only those rates which affect the qualitative behaviour of the model, and then conduct focused experimentation to shed light on these. Our identification and resolution approach has therefore provided an insightful and novel technique for documenting this knowledge using PNs.

STGTool represents a crucial enabling step to make these techniques practical. It completely automates the identifcation of OP violations in STGs produced by GNAPN, and guides the user through model development by requesting only the required information. This has therefore strengthened links between the three established disciplines of computing science, biology and electronic engineering, and has opened up new analysis potential for biological networks. In particular, it has demonstrated the applicability of highly efficient and well-founded circuit theory, and now calls for further development to more general asynchronous models.

(5) Integrated tool support

Tool support has been a key contribution throughout this thesis to make the qualitative techniques presented applicable and available to the biological community. In particular, three tools have been developed to address the modelling, analysis and refinement of MVNs using PNs

- a PN construction tool GNAPN to automate the techniques of Chapters 3 and 4 ;
- a prototype tool Refiner to automate the techniques of Chapter 5;
- an incremental model development and documentation tool STGTool to automate the techniques of Chapter 6.

These have been tightly integrated to form a suite of tools which offers much flexibility and application potential to practical systems.

GNAPN supports the three key PN modelling approaches developed: safe PNs, HLPNs and STGs. Efficient logic minimisation techniques using the auxiliary tool MVSIS [42] enable compact PN to be rapidly constructed modelling MVNs under both synchronous and asynchronous semantics. A particularly insightful feature for biologists, and one which we explored in our case studies, was the rapid development of mutant models. Here, the biologist is able to knockout and overexpress any number of entities,

thus enabling an exploration of robustness under adverse conditions from within the PN environment. For powerful model checking of global properties, GNAPN supports special initial marking configurations, and we used these in the case studies of Chapters 3 and 4. Constructed PNs can be exported to a number of interchange formats, opening them up to mature and insightful analysis techniques and tools. Furthermore, GNAPN can produce STGs from BNs for use in STGTOOL, thus forming an important link in the incremental model development process. Overall, GNAPN has played an integral part throughout all case studies in this thesis, as well as others (e.g. see [178])

REFINER provides a straightforward interface to the systematic refinement and verification algorithms proposed in Chapter 5. REFINER takes as its input the MVN model files used by GNAPN, thus allowing simplified MVN to be analysed using PN techniques. REFINER has been a key factor for the investigations in Chapter 5, and was used to formally reason about the relationship between two models of the *E. coli* carbon starvation network.

STGTOOL automates the STG approach for incremental model development proposed in Chapter 6, and thus contributes new tool support to the biological community. In particular, its straightforward interface lists all OP violations and allows the biologist to interactively resolve and document them using relative rates based on priorities. A solve history provides an easy way to record previously resolved violations, and the ability to backtrack through this history at any point encourages the biologist to experiment with new hypotheses and dynamically observe their implications. We used STGTOOL in a case study to develop a realistic BN model of the phage λ decision circuit, resulting in a simpler model which still captured the fundamental switch.

All three tools are freely available for academic use from *bioinf.ncl.ac.uk/gnapn*, and work is now ongoing to fully integrate these into a single qualitative workbench for modelling, analysing and refining GRN models using PNs.

At a more abstract level, this work balanced a number of important considerations. The first is that meaningful models have been constructed to reflect the quality of the *input data*; something which we have argued is not currently in balance with quantitative techniques. The *effort* and *scalability* of these techniques has also been addressed, especially with the tool support developed. Finally, the *output* produced has been commensurate with the inputs – in fact, a key point has been that PNs can often add value, by providing important insights which can then *feedback* to improve this input. As a result, we have motivated PNs as a self-contained environment for modelling, analysing and refining GRNs.

As the quality of data improves, however, quantitative techniques may become increasingly utilised in the biological community to facilitate ever more comprehensive studies. One possible way to achieve this switch is through hybrid approaches, which amalgamate the two under one formalism [132, 135]. The attraction of these is that they allow qualitative techniques to do the "filtering" and quantitative techniques to do the "focusing". Complex models could therefore be incrementally developed by allowing quantitative components to replace qualitative ones as more information is acquired.

Ultimately, such approaches may change the very way in which biological study is

conducted. Indeed, the advances in post-genomic technology of the ninetees [33] changed what was a model-driven conceptual view to a data-driven one. However, this paradigm of study could shift back to being model-driven – with highly efficient and cost-effective experimentation conducted purely *in silico* – and if PNs continue to mature in the same way they have been, they could play an integral part in realising this goal.

7.2 Future work

This thesis has investigated the development of a wide range of qualitative techniques for GRNs based on PNs. We now see a number of interesting areas of future research to take this work forward:

(1) We intend to increase the range of import and export formats for our PN modelling techniques. In particular, SBML [179] would greatly improve the exchange and study of these models.

(2) We intend to extend the PN modelling framework from Chapters 3 and 4 to encompass GRNs, as well as metabolic and signaling pathways in a unifying way, to take into account stochastic affects and time delays, and to remain amenable to proper analytical techniques, such as advanced model checking based on temporal logics [41]. In particular, we see *compositional* model construction and analysis [36, 52, 218] crucial for achieving this in a scalable and useful way.

(3) Analysis and interpretation of practical biological models can be hampered by state space explosion. Taking the techniques of Chapter 5 as a starting point, we intend to investigate approaches for deriving the simplest model which captures the necessary behaviour for a particular analysis task.

(4) We indend to address efficiency issues with an existing distributed SPN simulator called NASTY [178, 180] for missing parameter estimation. In particular, we will investigate how OP violations in STGs can reduce this search space by identifying the key reaction rates that require estimating.

(5) We are interested in the development of systematic techniques for deriving qualitative reaction rate information in STGs. Currently, these must be specified by the user, but a possible solution could be to consider the use of T-invariants, as discussed by Popova-Zeugmann *et al.* [161].

(6) The qualitative techniques presented, especially STGs, appear to offer a promising technique for *synthetic biology* [13,43]. Here, synthetic biological circuits are developed to perform a given task, and we see STGs as a potentially invaluable method for achieving this.

Work is now ongoing to develop these ideas into an integrated workbench for comprehensively modelling and analysing biological systems within the PN framework. In particular, the field of synthetic biology is attracting much interest, and we see the application of these techniques as a promising avenue into this potentially lucrative area.

Bibliography

[1] Petri net world. *Petri Net World, www.informatik.uni-hamburg.de/TGI/PetriNets.*

[2] Mvsis: Logic synthesis and verification. *MVSIS*, 2008.

[3] R Agarwal and M Tanniru. A petri-net based approach for verifying the integrity of production systems. *International Journal of Man-Machine Studies*, 36(3):447–468, 1992.

[4] T Akutsu, S Kuhara, O Maruyama, and S Miyano. Identification of gene regulatory networks by strategic gene disruptions and gene overexpressions. *Theoretical Computer Science*, 298(1):235–251, 1998.

[5] T Akutsu, S Kuhara, O Maruyama, and S Miyano. A system for identifying genetic networks from gene expression patterns produced by gene disruptions. *Genome Informatics*, 9:151–160, 1998.

[6] T Akutsu, S Miyano, and S Kuhara. Identification of genetic networks from a small number of gene expression patterns under the boolean network model. *Pacific Symposium on Biocomputing.*, pages 17–28, 1999.

[7] T Akutsu, S Miyano, and S Kuhara. *Algorithms for Identifying Boolean Networks and Related Biological Networks based on Matrix Multiplication and Fingerprint Function.* 2000.

[8] T Akutsu, S Miyano, and S Kuhara. Algorithms for inferring qualitative models of biological networks. *Pac Symp Biocomput*, pages 8–14, 2000.

[9] T Akutsu, S Miyano, and S Kuhara. Inferring qualitative relations in genetic networks and metabolic pathways. *Bioinformatics*, 16(8):727–734, 2000.

[10] Tatsuya Akutsu, Morihiro Hayashida, Wai-Ki Ching, and Michael K Ng. Control of boolean networks: hardness results and algorithms for tree structured networks. *Journal of Theoretical Biology*, 244(4):670–9, 2007.

[11] R Albert and H Othmer. The topology of the regulatory interactions predicts the expression pattern of the segment polarity. *Journal of Theoretical Biology*, 223(1):1–18, 2003.

[12] A Alberts, D Bray, J Lewis, M Raff, K Roberts, and J.D Watson. Molecular biology of the cell. *New York, Garland*, 1983.

[13] E Andrianantoandro, S Basu, D Karig, and R Weiss. Synthetic biology: new engineering rules for an emerging discipline. *Mol Syst Biol*, 2, 2006.

[14] R Bagley and L Glass. Counting and classifying attractors in high dimensional dynamical systems. *Journal of Theoretical Biology*, 183(3):269–284, 1996.

[15] G Balbo. On the success of stochastic petri nets. *Petri Nets and Performance Models*, pages 2–9, 1995.

[16] G Balbo. Introduction to stochastic petri nets. *Lectures on Formal Methods and Performance Analysis*, 2090:84–155, 2001.

[17] P Baldi and A Long. A bayesian framework for the analysis of microarray expression data: regularized t-test and statistical inferences of ene changes. *Bioinformatics*, 17(6):509–519, 2001.

[18] V L Balke and J D Gralla. Changes in the linking number of supercoiled dna accompany growth transitions in escherichia coli. *J Bacteriol*, 169:4499–4506, 1987.

[19] R Banks, V Khomenko, and L J Steggles. A case for using signal transition graphs for analysing and refining genetic networks. *In 2nd International Meeting on Membrane Computing and Biologically Inspired Process Calculi (MeCBIC)*, 2008.

[20] R Banks and L J Steggles. A high-level petri net framework for genetic regulatory networks. *Journal of Integrative Bioinfomatics*, 4(3), 2007.

[21] G Batt, D Bergamini, H De Jong, H Garavel, and R Mateescu. Model checking genetic regulatory networks using gna and cadp. *Lecture Notes in Computer Science*, 2989:158–163, 2004.

[22] G Batt, H De Jong, J Geiselmann, and M Page. Analysis of genetic regulatory networks: A model-checking approach. *Working Notes of the Seventeenth International Workshop on Qualitative Reasoning*, pages 31–38, 2003.

[23] G Batt, D Ropers, H de Jong, J Geiselmann, R Mateescu, M Page, and D Schneider. Validation of qualitative models of genetic regulatory networks by model checking: analysis of the nutritional stress response in escherichia coli. *Bioinformatics*, 21(1):19–28, 2005.

[24] G Batt, D Ropers, H De Jong, M Page, and J Geiselmann. Symbolic reachability analysis of genetic regulatory networks using qualitative abstractions. *Automatica*, 44(4):982–989, 2007.

[25] G Batt, B Yordanov, R Weiss, and C Belta. Robustness analysis and tuning of synthetic gene networks, 2007.

[26] A Ben-Dor, R Shamir, and Z Yakhini. Clustering gene expression patterns. *Journal of Computational Biology*, 6, 1999.

[27] G Bernot, J Comet, A Richard, and J Guespin. Application of formal methods to biological regulatory networks: extending thomas' asynchronous logical approach with temporal logic. *Journal of Theoretical Biology*, 229(3):339–347, 2004.

[28] E Best, H Fleischhack, W Fraczak, R P Hopkins, H Klaudel, and E Pelz. A class of composable high level petri nets. *Application and Theory of Petri Nets*, 935:103–120, 1995.

[29] R Blossey, L Cardelli, and A Phillips. Compositionality, stochasticity, and cooperativity in dynamic models of gene regulation. *HFSP Journal*, 2(1):17–28, 2008.

[30] E Boros, T Ibaraki, and K Makino. Error-free and best-fit extensions of partially defined boolean functions. *Information and Computation*, 140:254–283, 2000.

[31] J Bower and H Bolouri. Computational modelling of genetic and biochemical networks. *MIT Press*, 2001.

[32] K Breeding. *Digital Design Fundamentals.* 1992.

[33] P O Brown and D Botstein. Exploring the new world of the genome with dna microarrays. *Nat Genet*, 21(1 Suppl):33–37, 1999.

[34] R Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, 35:677–691, 1986.

[35] R Bryant. Symbolic boolean manipulation with ordered binary-decision diagrams. *ACM Computing Surveys*, 24:293–318, 1992.

[36] T Bultan, J Fischer, and R Gerber. Compositional verification by model checking for counter-examples. *International Symposium on Software Testing and Analysis*, 21:224–238, 1996.

[37] C Burges. A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery*, 2:121–167, 1998.

[38] L Calzone, F Fages, and S Soliman. Biocham: an environment for modeling biological systems and formalizing experimental knowledge. *Bioinformatics*, 22(14):1805–1807, 2006.

[39] E Cardoza, R Lipton, and A Meyer. Exponential space complete problems for petri nets and commutative semigroups (preliminary report). *STOC '76: Proceedings of the eighth annual ACM symposium on Theory of computing*, pages 50–54, 1976.

[40] R Casey, H De Jong, and J L Gouze. Piecewise-linear models of genetic regulatory networks: Equilibria and their stability. *Journal of Mathematical Biology*, 52:27–56, 2006.

[41] N Chabrier-Rivier, M Chiaverini, V Danos, F Fages, and V Schchter. Modeling and querying biomolecular interaction networks, 2004.

[42] D Chai, J Jiang, Y Jiang, Y Li, and A Mishchenko. Mvsis 2.0 user's manual. *Department of Electrical Engineering and Computer Sciences.*

[43] K Channon, E Bromley, and D Woolfson. Synthetic biology through biomolecular design and engineering. *Current Opinion in Structural Biology*, 18(4):491–498, 2008.

[44] C Chaouiya. Petri net modelling of biological networks. *Bioinformatics*, 8(4):210–219, 2007.

[45] C Chaouiya, E Remy, P Ruet, and D Thieffry. Qualitative modelling of genetic networks: From logical regulatory graphs to standard petri nets. *Lecture Notes in Computer Science*, 3099:137–156, 2004.

[46] C Chaouiya, E Remy, and D Thieffry. Petri net modelling of biological regulatory networks, 2008.

[47] M Chen and R Hofestadt. Quantitative petri net model of gene regulated metabolic networks in the cell. *In Silico Biology*, 3:347–365, 2003.

[48] S Christensen and L Petrucci. Towards a modular analysis of coloured petri nets. *Lecture Notes in Computer Science; 13th International Conference on Application and Theory of Petri Nets*, 616:113–133, 1992.

[49] S Christensen and L Petrucci. Modular state space analysis of coloured petri nets. *Proc. 16th Int. Conf. Application and Theory of Petri Nets (ICATPN'95)*, 935:201–217, 1995.

[50] S Christensen and L Petrucci. Modular analysis of petri nets. *The Computer Journal*, 43:224–242, 2000.

[51] E Clarke, O Grumberg, and D Peled. Model checking. *Springer*, 1999.

[52] E M Clarke, D E Long, and K L McMillan. Compositional model checking. *Logic in Computer Science*, pages 353–362, 1989.

[53] J.-P Comet, H Klaudel, and S Liauzu. Modeling multi-valued genetic regulatory networks using high-level petri nets. *LNCS*, 3536:208–227, 2005.

[54] J Cortadella, M Kishinevsky, and A Kondratyev. Petrify: A tool for manipulating concurrent specifications and synthesis of asynchronous controllers. *IEICE TRANSACTIONS on Information and Systems*, 1997.

[55] J Cortadella, M Kishinevsky, A Kondratyev, L Lavagno, and A Yakovlev. Logic synthesis for asynchronous controllers and interfaces. *Springer-Verlag*, 2001.

[56] F Dellaert and R Béer. Toward an evolvable model of development for autonomous agent synthesis. *Artificial Life IV, Proceedings of the Fourth International Workshop on the Synthesis and Simulation of Living Systems*, 1994.

[57] P D'haeseleer, S Liang, and R Somogyi. Genetic network inference: from co-expression clustering to reverse engineering. *Bioinformatics*, 16:707–726, 2000.

[58] D Dill, M Knapp, P Gage, C Talcott, and K Laderoute. The pathalyzer: A tool for analysis of signal transduction pathways. *Lecture Notes in Computer Science*, 2007.

[59] A Doi. Constructing biological pathway models with hybrid functional petri nets. *In Silico Biology*, 4(3):271, Jan 2004.

[60] A Doi, M Nagasaki, H Matsuno, and S Miyano. Simulation-based validation of the p53 transcriptional activity with hybrid functional petri net. *In Silico Biology*, 2006.

[61] B Drossel, T Mihaljev, and F Greil. Number and length of attractors in a critical kauffman model with connectivity one. *Physical Review Letters*, 2005.

[62] R Durrett. Stochastic calculus: A practical introduction. 1996.

[63] J Esparza. Model checking using net unfoldings. *Science of Computer Programming*, 23:151–195, 1994.

[64] A Fauré, C Chaouiya, A Ciliberto, and D Thieffry. Logical modelling and analysis of the budding yeast cell cycle. *feedback*, 8(8), 2007.

[65] A Faure, A Naldi, C Chaouiya, and D Thieffry. Dynamical analysis of a generic boolean model for the control of the mammalian cell cycle. *Bioinformatics*, 24(14):124–131, 2006.

[66] R.P Feynman. Atoms in motion. *in T. Ferris (ed.), The world treasury of physics, astronomy and mathematics, Boston*, pages 3–17, 1991.

[67] N Friedman, M Linial, I Nachman, and D Peer. Using bayesian networks to analyze expression data. volume 7, pages 601–620, 2000.

[68] N Friedman, D Peer, and I Nachman. Learning bayesian network structure from massive datasets: The sparse candidate algorithm. *Proc. Fifteenth Conf. on Uncertainty in Artificial Intelligence*, pages 206–215, 1999.

[69] A Gambin, S Lasota, and M Rutkowski. Analyzing stationary states of gene regulatory network using petri nets. *In Silico Biology*, 6, 2006.

[70] A Garg, A Di Cara, I Xenarios, L Mendoza, and G De Micheli. Synchronous versus asynchronous modeling of gene regulatory networks. *Bioinformatics*, 24(17):1917–1925, 2008.

[71] J Gebert, N Radde, and G Weber. Modeling gene regulatory networks with piecewise linear differential equations. *European Journal of Operational Research*, 181:1148–1165, 2007.

[72] H Genrich, R Kuffner, and K Voss. Executable petri net models for the analysis of metabolic pathways. *International Journal on Software Tools for Technology Transfer*, 3:394–404, 2001.

[73] C Gershenson. Introduction to random boolean networks. *Workshop and Tutorial Proceedings, Ninth International Conference on the Simulation and Synthesis of Living Systems (ALife IX)*, pages 160–173, 2004.

[74] D Gilbert and M Heiner. From petri nets to differential equations–an integrative approach for biochemical network analysis. *Petri Nets and Other Models of Concurrency (ICATPN)*, 4024:181–200, 2006.

[75] D Gilbert, M Heiner, and S Lehrack. A unifying framework for modelling and analysing biochemical pathways using petri nets. *Lecture Notes in Computer Science*, 4695:200, 2007.

[76] D.T Gillespie. Exact stochastic simulation of coupled chemical reactions. *The Journal of Physical Chemistry*, 81(25):2340–2361, 1977.

[77] L Glass and S Kauffman. The logical analysis of continuous, non-linear biochemical control networks. *J Theor Biol*, 39(1):103–129, 1973.

[78] A Gonzalez, A Naldi, L Sánchez, and D Thieffry. Ginsim: A software suite for the qualitative modelling, simulation and analysis of regulatory networks. *BioSystems*, 84(2):91–100, 2006.

[79] P J E Goss and Jean Peccoud. Quantitative modelling of stochastic systems in molecular biology by using stochastic petri nets. *Proceedings of the National Academy of Sciences*, 95:6750–6755, 1998.

[80] E Grafahrend-Belau, F Schreiber, and M Heiner. Modularization of biochemical networks based on classification of petri net t-invariants. *BMC Bioinformatics*, 9(90), 2008.

[81] B Grahlmann. The pep tool. *Computer Aided Verification, LNCS, Springer*, 1254:440–443, 1997.

[82] F Greil, B Drossel, and J Sattler. Critical kauffman networks under deterministic asynchronous update. *New Journal of Physics*, 9(373), 2007.

[83] Simone Gupta, Siddharth S Bisht, Ritushree Kukreti, Sanjeev Jain, and Samir K Brahmachari. Boolean network analysis of a neurotransmitter signaling pathway. *Journal of Theoretical Biology*, 244(3):463–9, Feb 2007.

[84] S Hardy and P Robillard. Modeling and simulation of molecular biology systems using petri nets: modeling goals of various approaches. *Journal of bioinformatics and computational biology.*, 2:595–613, 2004.

[85] S Hardy and P Robillard. Petri net-based method for the analysis of the dynamics of signal propagation in signaling pathways. *Bioinformatics*, 2008.

[86] I Harvey and T Bossomaier. *Proceedings of the Fourth European Conference on Artificial Life (ECAL97.*

[87] J Hasty, D McMillen, F Isaacs, and J Collins.... Computational studies of gene regulatory networks: in numero molecular biology. *Nat. Rev. Genet*, 2:268–279, 2001.

[88] M Heinemann and S Panke. Synthetic biology–putting engineering into biology. *Bioinformatics*, 22(22):2790–2799, 2006.

[89] M Heiner and I Koch. Petri net based model validation in systems biology. *ICATPN LNCS*, 3099:216–237, 2004.

[90] M Heiner, I Koch, and J Will. Model validation of biological pathways using petri nets—demonstrated for apoptosis. *BioSystems*, 2004.

[91] Monika Heiner, Ina Koch, and Klaus Voss. Analysis and simulation of steady states in metabolic pathways with petri nets. pages 15–34, 2001.

[92] R Hengge-Aronis. The general stress response in escherichia coli. *Bacterial Stress Responses, American Society for Microbiology Press*, pages 161–178, 2000.

[93] S Huang. Gene expression profiling, genetic networks, and cellular states: An integrating concept for tumorigenesis and drug discovery. *Journal of Molecular Medicine*, 77:469–480, 1999.

[94] S Huang. Genomics, complexity and drug discovery: insights from boolean network models of cellular regulation. *Pharmacogenomics*, 2(3):203, 2001.

[95] S Huang and D Ingber. Shape-dependent control of cell growth, differentiation, and apoptosis: Switching between attractors in cell regulatory networks. *Experimental Cell Research*, 261(1):91–103, 2000.

[96] I Ivanov and E Dougherty. Modeling genetic regulatory networks: continuous or discrete? *Journal of Biological Systems*, 671:307–340, 2006.

[97] K Jensen. Coloured petri nets - basic concepts, analysis methods and practical use. *EATCS Monographs on Theoretical Computer Science, Springer-Verlag*, 1, 1992.

[98] H De Jong. Modelling and simulation of genetic regulatory systems: A literature review. *Journal of Computational Biology*, 9:67–103, 2002.

[99] H De Jong, J Geiselmann, G Batt, C Hernandez, and M Page. Qualitative simulation of the initiation of sporulation in b. subtilis. *Bull. Math. Biol.*, 66(2):261–299, 2002.

[100] G Karlebach and R Shamir. Modelling and analysis of gene regulatory networks. *Nature Reviews: Molecular Cell Biology*, 9, 2008.

[101] S Kauffman. Homeostasis and differentiation in random genetic control networks. *Nature*, 224:177–178, 1969.

[102] S Kauffman. Metabolic stability and epigenesis in randomly constructed genetic nets. *J Theor Biol*, 2(3):437–467, 1969.

[103] S Kauffman. Gene regulation networks: a theory for their global structure and behaviors. *Curr Top Dev Biol*, 6(6):145–82, 1971.

[104] S Kauffman. Antichaos and adaptation. *Scientific American*, 265:78–84, Jan 1991.

[105] S Kauffman. The origins of order: Self-organization and selection in evolution. *Oxford University Press, New York*, Jan 1993.

[106] S Kauffman. At home in the universe: The search for laws of self-organization and complexity. *Oxford University Press, USA*, Jan 1995.

[107] S Kauffman, C Peterson, B Samuelsson, and C Troein. Random boolean network models and the yeast transcriptional network. *Proceedings of the National Academy of Sciences of the USA*, 100(25):14796–14799, 2003.

[108] M Kearns and U Vazirani. An introduction to computational learning theory. *MIT Press*, 1994.

[109] V Khomenko. *Model Checking Based on Prefixes of Petri Net Unfoldings.* PhD thesis, 2003.

[110] V Khomenko, A Kondratyev, M Koutny, and W Vogler. Merged processes — a new condensed representation of petri net behaviour. *Lecture Notes in Computer Science*, 3653:338–352, 2005.

[111] V Khomenko and M Koutny. Branching processes of high-level petri nets. *Tools and Algorithms for the Construction and Analysis of Systems: 9th International Conference, TACAS 2003, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2003, Warsaw, Poland, Lecture Notes in Computer Science,*, 2619:458–472, 2003.

[112] H Kitano. Computational systems biology. *Nature*, Jan 2002.

[113] K Klemm and S Bornholdt. Stable and unstable attractors in boolean networks. *Physical Review E*, Jan 2005.

[114] I Koch, G Junker, and M Heiner. Applications of petri net theory for modelling and validation of the sucrose breakdown pathway in the potato tuber. *Bioinformatics*, 21:1219–1226, 2005.

[115] I Koch, S Schuster, and M Heiner. Using time-dependent petri nets for the analysis of metabolic networks. *In Hofest adt, R., Lautenbach, K., Lange, M. (eds): DFG-Workshop: Informatikmethoden zur Analyse und Interpretation groYer genomischer Datenmengen, Magdeburg*, pages 15–21, 2000.

[116] G Koh, H Teong, M Clement, and D Hsu. A decompositional approach to parameter estimation in pathway modeling: a case study of the akt and mapk pathways and their crosstalk. *Bioinformatics*, 22(14):271–280, 2006.

[117] M Kwiatkowska, G Norman, and D Parker. Prism: Probabilistic symbolic model checker. *Lecture Notes in Computer Science*, pages 200–204, Jan 2002.

[118] Yung-Keun Kwon and Kwang-Hyun Cho. Boolean dynamics of biological networks with multiple coupled feedback loops. *Biophys J*, 92(8):2975–81, 2007.

[119] H Lähdesmäki, S Hautaniemi, I Shmulevich, and O Yli-Harja. Relationships between probabilistic boolean networks and dynamic bayesian networks as models of gene regulatory networks. *Signal Processing*, 86(4):814–834, 2006.

[120] H Lahdesmki, I Shmulevich, and O Yli-Harja. On learning gene regulatory networks under the boolean network model. *Machine Learning*, 52:147–167, 2003.

[121] T Latvala and M Makela. Ltl model checking for modular petri nets. *Applications and Theory of Petri Nets*, 3099:298–311, 2004.

[122] D Lee, R Zimmer, S Lee, and S Park. Colored petri net modeling and simulation of signal transduction pathways. *Metabolic Engineering*, 2006.

[123] S Lehrack. Three petri net approaches for biochemical network analysis. *Technical Report I-01*, page 33, Jan 2006.

[124] Peng Li, Chaoyang Zhang, Edward J Perkins, Ping Gong, and Youping Deng. Comparison of probabilistic boolean network and dynamic bayesian network approaches for inferring gene regulatory networks. *BMC Bioinformatics*, 8 Suppl 7:S13, 2007.

[125] S Liang, S Fuhrman, and R Somogyi. Reveal: a general reverse engineering algorithm for inference of genetic network architectures. *In Pacific Symposium Biocomputing 1998*, pages 18–29, 1998.

[126] M Makela. Model checking safety properties in modular high-level nets. *ICATPN 2003, LNCS*, 2679:201–220, 2003.

[127] M Mano. Digital design. *Prentice Hall*, 1984.

[128] M Marsan. Stochastic petri nets: An elementary introduction. *Advances in Petri Nets*, Jan 1989.

[129] S Martin, G Davidson, E May, J Faulon, and M Werner-Washburne. Inferring genetic networks from microarray data. *IEEE Computational Systems Bioinformatics Conference (CSB?04)*, pages 566–569, 2004.

[130] S Martin, Z Zhang, A Martino, and J Faulon. Boolean dynamics of genetic regulatory networks inferred from microarray time series data. *Bioinformatics*, 23(7):866–874, 2007.

[131] H Matsuno, A Doi, R Drath, and S Miyano. Genomic object net: Object oriented representation of biological systems. *GENOME INFORMATICS SERIES*, 2000.

[132] H Matsuno, A Doi, M Nagasaki, and S Miyano. Hybrid petri net representation of gene regulatory networks. *Pacific Symposium on Biocomputing*, 5:338–349, 2000.

[133] H Matsuno, S Inouye, Y Okitsu, Y Fujii, and S Miyano. A new regulatory interaction suggest by simulations for circadian genetic control mechanisms in mammals. *Journal of Bioinformatics and Computational Biology*, 4(1):139–153, 2006.

[134] H Matsuno, C Li, , and Miyano. Petri net based descriptions for systematic understanding of biological pathways. *Transactions on Fundamentals of Electronics, Communications, and Computer Sciences*, E89-A:3166–3174, 2006.

[135] H Matsuno, Y Tanaka, H Aoshima, A Doi, and M Matsui. Biopathways representation and simulation on hybrid functional petri net. *In Silico Biology*, 3, 2003.

[136] E Mayr. An algorithm for the general petri net reachability problem. *Proceedings of the thirteenth annual ACM symposium on Theory of computing*, pages 238–246, 1981.

[137] H McAdams and A Arkin. Simulation of prokaryotic genetic circuits. *Annual Review of Biophysics and Biomolecular Structure*, 27:199–224, Jan 1998.

[138] H McAdams and A Arkin. It's a noisy business! genetic regulation at the nanomolar scale. *Trends in Genetics*, 1999.

[139] H McAdams and L Shapiro. Circuit simulation of genetic networks. *Science*, 269:650–656, 1995.

[140] P McGeer, J Sanghavi, R Brayton, and S Vincentelli. Espresso-signature: A new exact minimizer for logic functions. *IEEE Transactions on VLSI*, 1:432–440, 1993.

[141] L Mendoza, D Thieffry, and E R.Alvarez-Buylla. Genetic control of flower morphogenesis in arabidopsis thaliana: a logical analysis. *Bioinformatics*, 15(7):593–606, 1999.

[142] T Mestl, C Lemay, and L Glass. Chaos in high-dimensional neural and gene networks. *Physica D: Nonlinear Phenomena*, 98:33–52, 1996.

[143] T Mestl, E Plahte, and S Omholt. A mathematical framework for describing and analysing gene regulatory networks. *Journal of Theoretical Biology*, Jan 1995.

[144] R Milner, J Parrow, and D Walker. A calculus of mobile processes-part i. *topps.diku.dk*, 100(1):1–40, 1990.

[145] A Miner and G Ciardo. Efficient reachability set generation and storage using decision diagrams. *Application and Theory of Petri Nets*, Jan 1999.

[146] A Mishchenko and R Brayton. Simplification of non-deterministic multi-valued networks. volume 2, pages 557–562, 2002.

[147] T Murata. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, 77:541–580, 1989.

[148] K Murphy and S Mian. Modelling gene expression data using dynamic bayesian networks. *University of California*, 1999.

[149] M Nagasaki, A Doi, H Matsuno, and S Miyano. Genomic object net: I. a platform for modelling and simulating biopathways. *Appl Bioinformatics*, 2(3):181–184, 2003.

[150] M Nagasaki, A Doi, H Matsuno, and S Miyano. A versatile petri net based architecture for modeling and simulation of complex biological processes. *Genome Informatics*, 15(1):180–197, 2004.

[151] D Nazareth. Investigating the applicability of petri nets for rule-based system verification. *IEEE Transactions on Knowledge and Data Engineering*, 5(3):402, 1993.

[152] A Ohta and K Tsuji. On some analysis properties of coloured petri nets using the underlying net. *47th IEEE International Midwest Symposium on Circuits and Systems*, 3:395–398, 2004.

[153] A Oppenheim, O Kobiler, and J Stavans. Switches in bacteriophage lambda development. *Annu Rev Genet*, 39:409–429, 2005.

[154] M Peleg, I Yeh, and R Altman. Modelling biological processes using workflow and petri net models. *Bioinformatics*, 18(6):825–837, 2002.

[155] Mor Peleg, Daniel Rubin, and Russ Altman. Using petri net tools to study properties and dynamics of biological systems. *Journal of the American Medical Informatics Association : JAMIA*, 12:181–199, 2005.

[156] C Petri. Kommunikation mit automaten. *PhD thesis, Bonn*, 1962.

[157] C Petri. Interpretations of net theory. *GMD, Interner Bericht*, 1976.

[158] A Phillips and L Cardelli. A graphical representation for biological processes in the stochastic pi-calculus. *Transactions in Computational Systems Biology*, 4230:123–152, 2006.

[159] A Phillips, L Cardelli, R Blossey, and L Goldstein. The stochastic pi machine (spim). ... *online at http://research. microsoft. com/ aphillip/spim*, Jan 2007.

[160] J Pinney, D Westhead, and G McConkey. Petri net representations in systems biology. *Biochem. Soc. Trans*, 31(6):1513–1515, 2003.

[161] L Popova-Zeugmann, M Heiner, and I Koch. Time petri nets for modelling and analysis of biochemical networks. *Fundamenta Informaticae*, 67:149–162, 2005.

[162] C Priami. Stochastic -calculus. *The Computer Journal.*

[163] V Reddy, M Liebman, , and Mavrovouniotis. Qualitative analysis of biochemical reaction systems. *Computers in Biology and Medicine*, 26:9–24, 1996.

[164] V N Reddy, M L Mavrovouniotis, and M N Liebman. Petri net representations in metabolic pathways. *International Conference on Intelligent Systems for Molecular Biology.*, 1:328–336, 1993.

[165] E Remy, P Ruet, L Mendoza, D Thieffry, and C Chaouiya. From logical regulatory graphs to standard petri nets: Dynamical roles and functionality of feedback circuits. *Concurrent Models in Molecular Biology*, 4230:52–72, 2004.

[166] O Roig, J Cortadella, and E Pastor. Verification of asynchronous circuits by bdd-based model checking of petri nets. *16th Int. Conf. on Application and Theory of Petri Nets*, 935:374–391, 1995.

[167] Delphine Ropers, Hidde De Jong, Michel Page, Dominique Schneider, , and Geiselmann. Qualitative simulation of the carbon starvation response in escherichia coli. *Bio Systems.*, 84(2):124–152, 2006. 10.1016/j.biosystems.2005.10.005.

[168] L Rosenblum and A Yakovlev. Signal graphs: From self-timed to timed ones. *International Workshop on Timed Petri Nets table of contents*, pages 199–206, Jan 1985.

[169] R Rudell and A Sangiovanni-Vincentelli. Multiple-valued minimization for pla optimization. *IEEE Transactions on Computer-Aided Design*, CAD-6, 1987.

[170] A Sackmann, M Heiner, and I Koch. Application of petri net based analysis techniques to signal transduction pathways. *BMC Bioinformatics*, 7:482–482, 2006.

[171] B Samuelsson and C Troein. Superpolynomial growth in the number of attractors in kauffman networks. *Physical Review Letters*, 90(9), 2003.

[172] M Schaub, T Henzinger, and J Fisher. Qualitative networks: A symbolic approach to analyze bio-logical signaling networks. *feedback*, 1(4), 2008.

[173] S Schuster, D Fell, and T Dandekar. A general definition of metabolic pathways useful for systematic organization and analysis of complex metabolic networks. *Nat Biotechnol*, 18(3):326–332, 2000.

[174] S Schuster, T Pfeiffer, F Moldenhauer, and I Koch. Exploring the pathway structure of metabolism: decomposition into subnetworks and application to mycoplasma pneumoniae. *Bioinformatics*, 18(2):351–361, 2002.

[175] R Shamir and R Sharan. Algorithmic approaches to clustering gene expression data. *Current Topics in Computational Biology*, pages 269–300, 2001.

[176] C Shannon. Prediction and entropy of printed english. *Bell System Technical Journal*, Jan 1951.

[177] S Shatz, S Tu, T Murata, and S Duri. An application of petri net reduction for ada tasking deadlockanalysis. *Parallel and Distributed Systems*, 7(12):1307–1322, 1996.

[178] O Shaw. Modelling bacterial regulatory networks with petri nets. *PhD Thesis, Newcastle University*, page 262, Sep 2006.

[179] O Shaw, A Koelmans, J Steggles, and A Wipat. Applying petri nets to systems biology using xml technologies. *ATPN 2004*, page 39, 2004.

[180] O Shaw, L Steggles, and A Wipat. Automatic parameterisation of stochastic petri net models of biological networks. *Electronic Notes in Theoretical Computer Science*, 151:111–129, 2006.

[181] O J Shaw, C Harwood, L J Steggles, and A Wipat. Sarge: a tool for creation of putative genetic networks. *Bioinformatics*, 20(18):3638–40, Dec 2004.

[182] I Shmulevich, E Dougherty, S Kim, and W Zhang. Probabilistic boolean networks: a rule-based uncertainty model for gene regulatory networks. *Bioinformatics*, 18(2):261–274, 2002.

[183] I Shmulevich, E Dougherty, and W Zhang. From boolean to probabilistic boolean networks as models of genetic regulatory networks. *Proceedings of the IEEE*, 90:1778–1792, 2002.

[184] I Shmulevich and S Kauffman. Activities and sensitivities in boolean network models. *Physical Review Letters*, 93(4), Jan 2004.

[185] I Shmulevich, O Yli-Harja, and J Astola. Inference of genetic regulatory networks under the best-fit extension paradigm. *In Proceedings of the IEEE—EURASIP Workshop on Nonlinear Signal and Image Processing (NSIP-01)*, pages 3–6, 2001.

[186] I Shmulevich and W Zhang. Binary analysis of optimization-based normalization of gene expression data. *Bioinformatics*, 18:555–565, 2002.

[187] E Simao, E Remy, D Thieffry, and C Chaouiya. Qualitative modelling of regulated metabolic pathways application to the tryptophan biosynthesis in e.coli. *Bioinformatics*, 21(2):190–196, 2005.

[188] P Smolen, D Baxter, and J Byrne. Modeling transcriptional control in gene networks—methods, recent results, and future directions. *Bulletin of Mathematical Biology*, 62:247–292, Jan 2000.

[189] J Socolar and S Kauffman. Scaling in ordered and critical random boolean networks. *Physical Review Letters*, 90(6), Jan 2003.

[190] E Sontag, A Veliz-Cuba, R Laubenbacher, and A Jarrah. The effect of negative feedback loops on the dynamics of boolean networks. *Biophys J*, Mar 2008.

[191] J Sparso. Asynchronous circuit design. page 182, Apr 2006.

[192] R Srivastava, M S Peterson, and W E Bentley. Stochastic kinetic analysis of the escherichia coli stress circuit using sigma-32 targeted antisense. *Biotechnology and Bioengineering*, 75:120–129, 2001.

[193] R Srivastava, L You, J Summers, and J Yin. Stochastic versus deterministic modelling of intracellular viral kinetics. *Latest Issue of Journal of Theoretical Biology*, 218:309–321, 2002.

[194] L J Steggles, R Banks, O Shaw, and A Wipat. Qualitatively modelling and analysing genetic regulatory networks: a petri net approach. *Bioinformatics*, Nov 2006.

[195] L J Steggles, R Banks, and A Wipat. Modelling and analysing genetic networks: From boolean networks to petri nets. pages 127–141, 2006.

[196] L J Steggles, R Banks, and A Wipat. Modelling and analysing genetic networks: From boolean networks to petri nets. *CS-TR 962, Newcastle University, UK*, 2006.

[197] P Stragier and R Losick. Molecular genetics of sporulation in bacillus subtilis. *Annual review of genetics*, 30:297–241, 1996. 10.1146/annurev.genet.30.1.297.

[198] M Sugita. Functional analysis of chemical systems in vivo using a logical circuit equivalent. *J Theor Biol*, 1:415–430, 1961.

[199] M Sugita and M Fukuda. Functional analysis of chemical systems in vivo using a logical circuit equivalent. 3. analysis using a digital circuit combined with an analogue computer. *J Theor Biol*, 3:412–425, 1963.

[200] Z Szallasi and S Liang. Modeling the normal and neoplastic cell cycle with "realistic boolean genetic networks": Their application for understanding carcinogenesis and assessing therapeutic strategies. *Pacific Symposium on Biocomputing*, 3:66–76, 1998.

[201] C Taubner, B Mathiak, A Kupfer, and N Fleischer. Modelling and simulation of the tlr4 pathway with coloured petri nets. *Engineering in Medicine and Biology Society*, Jan 2006.

[202] D Thain, T Tannenbaum, and M Livny. Distributed computing in practice: the condor experience. *Concurrency and Computation: Practice and Experience*, 17(2-4):323–356, 2005.

[203] R Thomas. Boolean formalization of genetic control circuits. *Journal of theoretical biology.*, 42:563–585, Dec 1973.

[204] R Thomas. Kinetic logic: A boolean approach to the analysis of complex regulatory systems. *Springer-Verlag*, 1979.

[205] R Thomas. Regulatory networks seen as asynchronous automata: A logical description. *Theor. Biol.*, 153:1–23, 1991.

[206] R Thomas and R D'Ari. *Biological Feedback.* 1990.

[207] R Thomas, A Gathoye, and L Lambert. A complex control circuit. regulation of immunity in temperate bacteriophages. *FEBS Journal*, 71:211–227, Jan 1976.

[208] R Thomas, D Thieffry, and M Kaufman. Dynamical behaviour of biological regulatory networks - i. biological role of feedback loops and practical use of the concept of loop-characteristic state. *Bulletin of mathematical biology.*, 57:247–276, Mar 1995.

[209] R Valk. Self-modifying nets, a natural extension of petri nets. *Lecture Notes in Computer Science*, pages 464–476, Jan 1978.

[210] W van der Aalst. Workflow verification: Finding control-flow errors using petri-net-based techniques. *Business Process Management: Models*, 2000.

[211] W van der Aalst. Pi calculus versus petri nets: Let us eat "humble pie" rather than further inflate the "pi hype". *BPTrends*, Jan 2005.

[212] E van Someren, L Wessels, E Backer, and M Reinders. Genetic network modeling. *Pharmacogenomics*, 3(4):507–525, 2002.

[213] K Voss, M Heiner, and I Koch. Steady state analysis of metabolic pathways using petri nets. *In Silico Biology*, 3:367–387, 2003.

[214] X Wen, S Fuhrman, G Michaels, D Carr, and S Smith. Large-scale temporal gene expression mapping of central nervous system development. *Proceedings of the National Academy of Sciences*, 95:334–339, Jan 1998.

[215] J Will and M Heiner. Petri nets in biology, chemistry, and medicine-bibliography. *Computer*, pages 1–36, Jan 2002.

[216] A Wuensche. Genomic regulation modeled as a network with basins of attraction. *Pacific Symposium on Biocomputing*, 3:89–102, 1998.

[217] A Wuensche and M Lesser. The global dynamics of cellular automata. *delta.cs.cinvestav.mx*, 1, 1992.

[218] W J Yeh and M Young. Compositional reachability analysis using process algebra. *TAV4: Proceedings of the symposium on Testing, analysis and verification*, pages 49–59, 1991.

[219] C Yuh, H Bolouri, and E Davidson. Genomic cis-regulatory logic: Experimental and computational analysis of a sea urchin gene. *Science*, 279(5358):1896–1902, 1998.

[220] Z Yun and K Keong. Dynamic algorithm for inferring qualitative models of gene regulatory networks. *Computational Systems Bioinformatics Conference*, Jan 2004.

[221] Z Yun and K Keong. Reconstructing boolean networks from noisy gene expression data. *Control, Automation, Robotics and Vision Conference*, 2:1049–1054, Jan 2004.

[222] I Zevedei-Oancea and S Schuster. Topological analysis of metabolic networks based on petri net theory. *In Silico Biology*, 3(3):323–345, 2003.

[223] S Zhang, W Ching, M Ng, and T Akutsu. Simulation study in probabilistic boolean network models for genetic regulatory networks. *International Journal of Data Mining and Bioinformatics*, 1(3):217–240, 2007.

[224] Shu-Qin Zhang, Morihiro Hayashida, Tatsuya Akutsu, Wai-Ki Ching, and Michael K Ng. Algorithms for finding small attractors in boolean networks. *EURASIP journal on bioinformatics & systems biology*, page 20180, 2007.

[225] M Zhou and F DiCesare. Petri net synthesis for discrete event control of manufacturing systems. *Springer-Verlag*, 1993.

[226] R Zurawski and M Zhou. Petri nets and industrial applications: A tutorial. *IEEE Transactions on Industrial Electronics*, 41:567–583, 1994.

Index