Available online at www.sciencedirect.com
ScienceDirect
Electronic Notes in Theoretical Computer Science
Electronic Notes in Theoretical Computer Science 227 (2009) 3–19
www.elsevier.com/locate/entcs

A Case for Using Signal Transition Graphs for Analysing and Refining Genetic Networks

Richard Banks, Victor Khomenko, and L. Jason Steggles

School of Computing Science, Newcastle University, UK

Abstract

In order to understand and analyse *genetic regulatory networks (GRNs)*, the complex control structures which regulate cellular systems, well supported qualitative formal modelling techniques are required. In this paper, we make a case that biological systems can be qualitatively modelled by speed-independent circuits. We apply techniques from asynchronous circuit design, based on *Signal Transition Graphs (STGs)*, to modelling, visualising and analysing GRNs. STGs are a Petri net based model that has been extensively used in asynchronous circuit design. We investigate how the sufficient conditions ensuring that an STG can be implemented by a speed-independent circuit can be interpreted in the context of GRNs. We observe that these properties provide important insights into a model and highlight areas which need to be refined. Thus, STGs provide a well supported formal framework for GRNs that allows realistic models to be incrementally developed and analysed. We demonstrate the proposed STG approach with a case study of constructing and analysing a speed-independent circuit specification for the lysis-lysogeny switch in phage λ.

Keywords: Genetic regulatory networks, Signal Transition Graphs, Petri nets, network analysis

1 Introduction

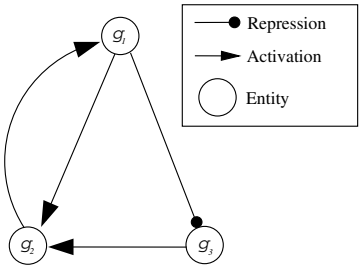
Biological systems are controlled by *genetic regulatory networks (GRNs)* [2] which comprise complex control structures of interacting entities including genes, proteins and metabolites. In order to be able to understand and investigate the complex behaviour of GRNs, various formal modelling techniques have been proposed, ranging from simple qualitative approaches, such as Boolean networks, to detailed quantitative approaches based on differential equations or stochastic techniques (see [2,8] for an overview). Given the lack of quantitative data concerning exact reaction rates and the noise associated with such data, qualitative modelling techniques have emerged as an important first approach to understanding GRNs [2].

Boolean networks [1,2] are a qualitative modelling technique that has received much attention in the literature. A Boolean network consists of a set of regulatory entities $\{g_1, \ldots, g_n\}$ which can be in one of two possible states, either 1 representing the entity is active (e.g., a gene is expressed or a protein is present) or 0 representing the entity is inactive (e.g., a gene is not expressed or a protein is absent). The state of a Boolean network is therefore a Boolean vector consisting of each entity's current state, and this results in a state space containing 2^n states for n entities. The behaviour of each entity g_i is described by a Boolean *next-state function,* which,

1571-0661/
doi:10.1016/j.entcs.2008.12.101

given the current states of the entities that affect it (referred to as its *neighbourhood*), returns the next state for g_i. A Boolean network can be interpreted in two distinct ways [2]: either *synchronously,* where all entities update their states together, or *asynchronously*, where entities update their states independently. Owing to the clear parallels between Boolean networks and digital circuits, we use the terms 'Boolean network' and 'circuit' interchangeably in this paper, and will often refer to the nodes of a Boolean network as *(logic) gates;* each gate computes the next-state function of the corresponding entity.

As an example, consider the Boolean network in Fig. 1(a) [1], which contains three entities, g_1, g_2 and g_3. The next state $[g_i]$ of each entity g_i is defined by the truth table given in Fig. 1(b) which corresponds to the equations in Fig. 1(c); the notation $\overline{x}$, $x + y$ and $x\,y$ is used to represent the Boolean operators *not*, *or* and *and*, respectively.



(a)

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

(b)

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

(c)

Fig. 1. A Boolean network (a); the truth table for the next-state functions (b); and the equations obtained from the truth table by Boolean minimisation (c).

While Boolean networks have proved successful in modelling GRNs [9,16], their application in practice is hindered by a number of shortcomings. Historically, the synchronous semantics has been favoured, since they are easier to work with due to their deterministic behaviour. However, the assumption of synchronous updates can be argued to be biologically unrealistic [17], which leads to reservations about the results obtained from such models.

Hence the asynchronous semantics seems to be more realistic. However, asynchronous networks also have shortcomings. In particular, they tend to have too rich behaviour, not all of which is realisable in practice. This behaviour also tends to be highly non-deterministic, i.e., (non-converging) *choices* are common when choosing the next state.

In practice, many such choices are resolved either by assuming that the environment of the biological system is slow (i.e., the system always has enough time to react to its changes), or by relative speeds of chemical reactions; that is, the behaviour in fact has much less non-determinism than such models suggest. (This may explain why synchronous networks, which are always deterministic, were often favoured over asynchronous ones, in spite of synchronous updates being biologically unrealistic.)

These considerations motivate us to consider modelling biological systems using

speed-independent (SI) circuits [7], which are a subclass of asynchronous circuits that work correctly (i.e., according to their specification) regardless of the delays associated with logic gates. We follow the classical Muller's approach [10] which regards each logic gate as an atomic evaluator of a Boolean function, with a delay element associated with its output (the wires are assumed to have negligible delays). In the SI framework, no assumptions are made about the gate delays (except that they are positive), i.e., individual gates can be arbitrarily slow/fast and even have variable unbounded delays.

SI circuits tend to be deterministic, though they can handle certain kinds of non-determinism using *arbiters* [7] — special devices deciding which of two inputs arrives first (this proves to be important from a biological perspective as illustrated in Section 5). Hence we make the following important methodological assumption:

Biological systems can be modelled by speed-independent circuits.

That is, if a biological system cannot be qualitatively modelled by an SI circuit then its model is either incorrect or misses some important information. We will discuss this issue later in the paper.

It turns out that whether a circuit is SI or not almost always depends on its environment, i.e., a circuit can be SI in one environment and non-SI in another one. That is, *whether the circuit is SI or not cannot be deduced solely from the structure of the circuit!* This suggests that *traditional asynchronous Boolean networks lack some important information (viz. the behaviour of the environment).*

In this paper, we make a case for using another formalism, viz. *Signal Transition Graphs (STGs)* [5,14], which allows one to capture in a natural way the behaviour of both the circuit and its environment. STGs are Petri nets in which transitions are labelled with the rising and falling edges of circuit signals. They have been used extensively for the design of asynchronous control circuits.

We investigate how the sufficient conditions ensuring that an STG can be implemented by an SI circuit [7] can be interpreted in the context of 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)* condition [7] indicates the presence of choices that either require further information to resolve or indicate some stochastic effects in the system that have to 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 models to be incrementally developed and analysed.

We illustrate our proposed STG framework 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 [12]. We begin by constructing an STG model based on the Boolean network presented in [17]. We then refine this by finding the points where this STG violates the SI conditions and appropriately resolving the problems. In particular, we see how some violations of OP highlight timing assumptions about the environment's behaviour, and how the arbitration represents the stochastic choice between lysogeny and lysis modes in phage λ. The case study

makes use of the STG support tool PETRIFY [7] and demonstrates its practical role in model development.

The paper is organised as follows. In Section 2, we briefly introduce STGs and consider how they can be used to model a GRN. Then, in Section 3, we consider how STG techniques from electronic circuit design can be applied to refine, visualise and analyse models of GRNs. In Section 4, we consider more formally the properties required for an STG to be implementable as an SI circuit and relate these to the biological setting. In Section 5, we present a detailed case study which illustrates how the techniques introduced are applied in practice. Finally, in Section 6, we conclude by summarising our results and discussing future work.

2 Signal Transition Graphs

The theory of Petri nets [13,11] provides a graphical notation with a formal mathematical semantics for modelling and reasoning about concurrent distributed systems. A Petri net [11] is a directed bipartite graph consisting of: *places*, denoted by circles, which represent resources or conditions; *transitions*, denoted by rectangles, which represent actions or events; and *arcs*, denoted by arrows, which connect places to transitions or transitions to places. A simple example of a Petri net is given in Fig. 2.

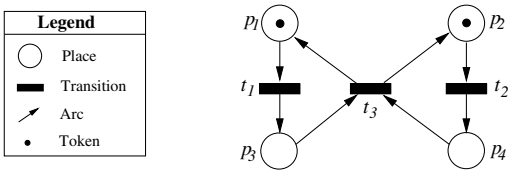


Fig. 2. A simple example of a Petri net.

The places, transitions and arcs describe the static structure of the Petri net; its state is given by the distribution of *tokens* (depicted as black dots) on its places, referred to as a *marking*. The dynamic properties of the system are modelled by transitions which can *fire,* changing the distribution of tokens on places in a Petri net. A transition is said to be *enabled* if each of its input places contains at least one token. An enabled transition can *fire* by consuming one token from each of its input places and then depositing one token on each of its output places. Often, more than one transition is enabled at any one time, and in such a case any enabled transition can fire. For example, in Fig. 2 both transitions t_1 and t_2 are enabled. Firing t_1 would result in a token being taken from place p_1 and a new token being deposited on place p_3. An important advantage of Petri nets is that they are supported by a wide range of theoretically well-founded analysis techniques and tools [18].

Signal Transition Graphs (STGs) [7] is a particular type of labelled Petri nets developed specifically for modelling asynchronous digital circuits. The idea is to associate a set of Boolean variables, referred to as *signals*, with a Petri net to represent the state of the actual digital signals (i.e., wires) within a circuit. The Petri net's transitions are then labelled to represent changes in the state of these

signals; a transition label either has the form a^+ to indicate a signal a goes from 0 to 1, or a^- to indicate the signal goes from 1 to 0. Thus, the underlying Petri net specifies the causal relationship between signals and is intended to capture the behaviour of a circuit. Clearly, for an STG to correctly represent a circuit one has to ensure that the labels a^+ and a^- are correctly alternated between for each signal. This *consistency* condition for STGs is discussed in more detail in Section 4. In general, several transitions can have the same label, e.g., a^+; in such a case, these transitions are named a^+, $a^+/1$, $a^+/2$, etc.

Since the behaviour of an STG is based on its underlying Petri net behaviour, the concepts of enabling and firing of transitions introduced above still hold. STGs are therefore amenable to general Petri net analysis tools, but are also supported by a range of specific tools, such as PETRIFY [7], which are able to analyse and optimise STGs, as well as synthesise digital circuits from them. An STG can be represented graphically simply as a labelled Petri net. However, a short-hand notation is often used, in which transitions are simply represented by their labels, and non-marked places with only one input and one output transition are contracted (see Fig. 3(a)).

The signals of an STG are partitioned into *input*, *output* and *internal* signals; the output and internal signals are collectively referred to as *local* signals. The inputs are controlled by the environment of the STG (in the context of biological systems, this could be either the actual environment of the organism, or the other systems within the organism, whose outputs affect the behaviour of the system), and the outputs are controlled by the system itself and are observable by the environment (e.g., they can be inputs of other systems within the organism). Internal signals represent some auxiliary entities needed to produce outputs; like outputs, they are controlled by the system, but they are not observable by the environment. The partitioning of signals is an important part of the modelling process and represents key design decisions when developing an STG. We discuss this further in the biological context in Section 3.

Intuitively, an STG represents a contract between the system and its environment, and is interpreted in the following way. If an input signal transition is enabled, then the environment is allowed (but is not obliged) to send this input, and vice versa, the environment is not allowed to send inputs which are not enabled. If a local transition is enabled, then the system is obliged eventually to produce this signal (or it is eventually disabled by another transition, in which case the *output-persistency* (discussed later) is violated), and vice versa, it is not allowed to produce outputs which are not enabled. That is, an STG specifies the behaviour of a system in the sense that the system must provide *all and only* the specified outputs, and that it must allow *at least* the specified inputs (in fact, it could optionally allow more inputs, which means that it could work in a more demanding environment).

For example, consider the STG in Fig. 3(a). It models a system with two inputs, a and b, and one output, c, and the initial value of each signal is 0. The system waits until the environment raises (in any order) the inputs a and b (transitions a^+ and b^+), and then raises the output c (transition c^+). (Observe that 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 reset inputs until c^- fires.)

3 Relationship between STGs and Circuits

In this section, we describe the relationship between asynchronous Boolean networks (or circuits) and STGs. We show that a circuit can be translated into an STG, and the latter can be semi-automatically refined into an SI model. To gain an initial insight into the proposed method, we start off informally, by considering an example; then we formalise our approach.

The behaviour described by the STG in Fig. 3(a) can be implemented by the circuit $[c] = ab + c(a + b)$, which is SI in the intended environment (as specified in Fig. 3(a)). However, *just by looking at this circuit equation it is impossible to say what were the assumptions about the environment;* in particular, there are environments where the behaviour of this circuit becomes non-SI, e.g., if the environment, after raising a and b, resets either of them before c^+ fires. This illustrates that *having an STG can be much more useful for analysing the system than simply having a circuit definition.*

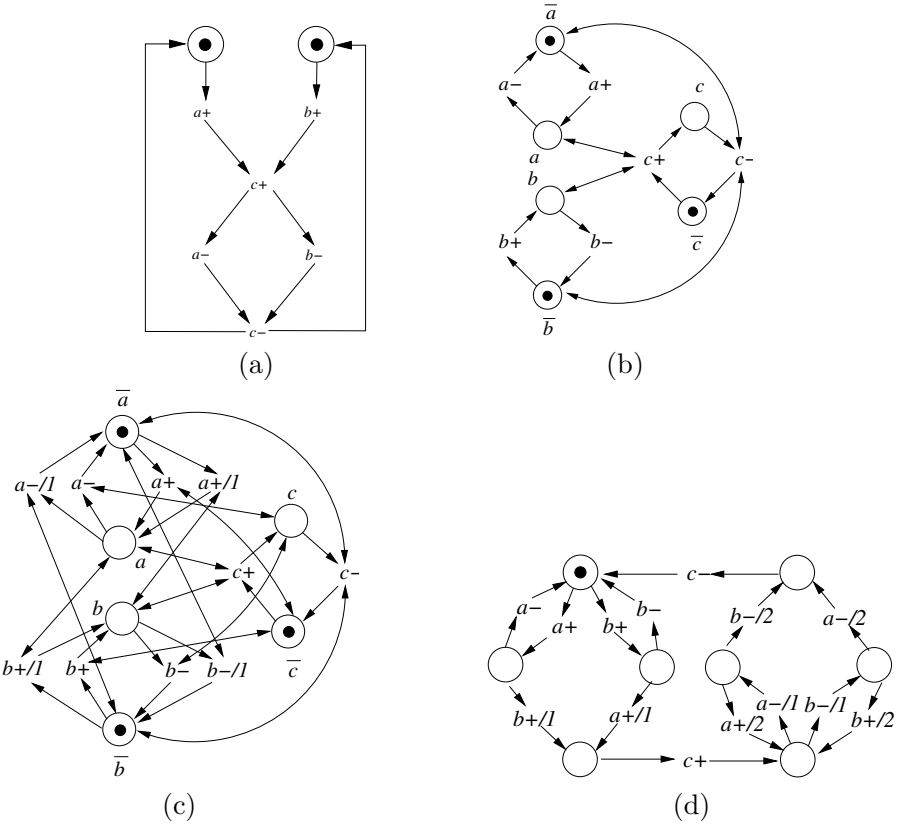


Fig. 3. An example STG (a); the circuit-STG for the circuit $[c] = ab + c(a + b)$ (b); a way to resolve choices in it by assuming a slow environment (c); and the STG simplified using PETRIFY (d).

Any digital circuit can be converted into an STG using the well-known translation based on complementary places [13, 15]. Fig. 3(b) illustrates this construction for the circuit $[c] = ab + c(a + b)$.

The circuit-STG construction

- Each signal (i.e., regulatory entity) g_i is represented by two places, g_i and $\overline{g_i}$, indicating 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 to g_i) and g_i^- (consuming a token from g_i and depositing a token to $\overline{g_i}$).
- For each local signal g_i the circuit computes the next-state value $[g_i]$ of g_i using the given Boolean equation $[g_i] = E_i$, see e.g., Fig. 1(c). (Note that such circuit equations can be straightforwardly extracted from a truth table definition using Boolean minimisation [15].) For each term (i.e., prime implicant) m_j in the minimised disjunctive normal form (DNF) 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 g_i on. We add an arc from place $\overline{g_i}$ to g_i^+/j and an arc from g_i^+/j to place g_i. 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 on a place without consuming it). We use a similar process to define the transitions g_i^-/j which reset g_i based on $E_i|_{g_i=1}$.

Note that the behaviour of the resulting STG shown in Fig. 3(b) strictly includes the behaviour of the initial model in part (a) of this figure, since the information about the behaviour of the environment could not be retrieved from the circuit, and the most general environment was modelled. PETRIFY automatically detects that the resulting STG is not SI in this environment, as an output c^+ can be disabled by a^- or b^-, and similarly, c^- can be disabled by a^+ or b^+.

If the circuit $[c] = ab + c(a + b)$ was used to model a system that is perceived to be deterministic, then some of this STG's behaviour is not realisable in practice. Hence the STG should be refined, so that it captures only the realistic behaviour. The candidate points where the changes should be made are where the speed-independence is violated, e.g., due to the choices involving a local transition. Such choices (unless they represent some truly stochastic phenomenon) have to be resolved either by making assumptions about the environment, or by looking at reaction rates. Methodologically, the points where the speed-independence is violated can be found automatically, but the resolution of choices requires interaction with the user.

Formally, $t \rightharpoonup t'$ means that a transition t can be disabled by firing a transition t', where t and t' have different labels and t is labelled by a local signal. One can see that for the STG in Fig. 3(b), $c^+ \rightharpoonup a^-$, $c^+ \rightharpoonup b^-$, $c^- \rightharpoonup a^+$ and $c^- \rightharpoonup b^+$

hold. This information is given to the user, who now can suggest a way to resolve this conflict. In this particular case, the user might know that the environment is relatively slow, i.e., if, say, a^- and c^+ are enabled simultaneously then c^+ will fire first. Alternatively, the relative rates of chemical reactions might determine which transition fires first. Of course, such rates must be provided by the user, since there is no way a tool can work them out from the STG or circuit. In practice, measuring reaction rates is a very effort-consuming task, but our method addresses this problem by giving information about what rates have to be measured (in practice, few rates affect the qualitative behaviour of the circuit), and by requiring only relative rates (i.e., it is enough to know that one reaction is faster than the other, rather than the absolute rates).

We use the following notation for the user-provided assumptions: we write $t \mapsto t'$ to denote that whenever transitions t and t' are enabled simultaneously then priority is given to t. (We assume that t and t' have different labels, at least one of these transitions is labelled by a local signal, t and t' share some pre-places, and not all of these shared places are accessed by t and t' in read-only fashion, i.e., by pairs of arcs going in opposite directions.) In our example, the slowness of the environment can be expressed as $c^+ \mapsto a^-$, $c^+ \mapsto b^-$, $c^- \mapsto a^+$, $c^- \mapsto b^+$.

Such priority assumptions $t \mapsto t'$ can be applied to the STG, resulting in a transformed model which captures this information. The idea is to replicate the transition with lower priority t' to capture each situation in which t is not enabled and t' can safely fire. We define this transformation more formally as follows.

The firing order enforcement (FOE) transformation

Suppose $t \mapsto t'$ has been assumed and let $p_1, \ldots, p_k$ be the pre-places of t which are not pre-places of t'. If $k = 0$ then t is enabled whenever t' is, and so t' can be simply eliminated from the STG, together with all the incident arcs, as in such a case it can never fire due to the assumption $t \mapsto t'$. Otherwise, t' is replicated $k-1$ times, so that there are k copies (denoted by $t'_1 = t', t'_2, \ldots, t'_k$) of t' altogether. All these replicas are labelled by the same signal as t', and have exactly the same connections. Furthermore, a pair of arcs going in the opposite directions is added between t'_i and $\overline{p_i}$ for each $i = 1, \ldots, k$, 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 $t'_1, \ldots, t'_k$ is 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 $t'_1, \ldots, t'_k$ is enabled in the modified STG. That is, the choice is resolved to favour t.

Our method allows for automatic application of user-given assumptions about the environment and relative reaction rates to the STG, in order to refine its behaviour. In particular, it transforms the STG in Fig. 3(b) into the one in part (c) of this figure, which, after simplification by PETRIFY, becomes the STG in part (d) of this figure. The latter STG has less behaviour than the STG in Fig. 3(b), and is SI. Somewhat unexpectedly, it has more behaviour than the initial model in Fig. 3(a). This is explained by the fact that it poses fewer constraints on the environment

(i.e., the system can actually cope with a more demanding environment than the one it was intended for).

4 Genetic Regulatory Networks as Circuits

In this section we discuss in more detail our methodological assumption that biological systems can be qualitatively modelled by speed-independent (SI) circuits. We present the properties necessary for an STG to be implementable as an SI circuit [7] and discuss their biological relevance. In particular, we consider the output-persistency condition and how a violation of this condition indicates the presence of choices which need further investigation.

For an STG to be implementable as an SI circuit (and hence, due to our methodological assumption, as a biological system), it must satisfy the following properties [7]:

Boundedness An STG has finitely many reachable states iff it is *bounded,* i.e., the number of tokens in each place can never exceed some bound k. Since a digital circuit (or a Boolean network) can have only finitely many reachable states, boundedness is taken as an implementability requirement. Note that the STGs produced from circuits by the circuit-STG construction are always bounded (in fact, *safe*, i.e., the respective bound is 1). Moreover, both boundedness and safeness are preserved by the FOE transformation, as it can only reduce the set of reachable markings.

Consistency *Consistency* is a basic well-formedness property, stating that the reachable signal values must be binary. That is, in every trace of the STG the transition labels for each signal a must alternate between a^+ and a^-, always beginning with the same sign. Note that the STGs produced from circuits by the circuit-STG construction are always consistent. Moreover, consistency is preserved by the FOE transformation, as it can only reduce the set of reachable markings.

Output-persistency *Output-persistency (OP)* property requires that if some local signal becomes enabled, it cannot be disabled by firing some other transition, i.e., there should be no choices involving local transitions. The rationale for this is that once a signal becomes enabled, its voltage starts, e.g., to rise from 0 to 1. If the signal is disabled during this process, the voltage is pulled down, resulting in a glitch. This glitch can be interpreted in different ways by the logic gates listening to this signal, depending on whether the voltage has crossed the threshold between 0 and 1 or not. Hence the behaviour of the circuit becomes non-deterministic. Such a situation can be interpreted in biological terms as well, with the voltage replaced by, e.g., the concentration of some protein.

Visually, if OP is violated then there are two transitions with different labels in the STG with at least one of them marked by a local signal, which share some pre-places and can be enabled simultaneously (unless both transitions are connected to these shared pre-places in the read-only way, i.e., by pairs of arcs going in opposite directions).

Note that a choice involving only inputs is not regarded as a violation of OP, and simply models a non-deterministic choice in the environment. (For example,

the environment might non-deterministically decide either to rise the temperature above normal, or to reduce it below normal.) Since this choice does not have to be implemented by the system, SI circuits can be synthesised for such STGs (provided that all the other conditions necessary for SI are met).

A choice involving only local transitions can still be implemented in a speed-independent way (in spite of the violation of OP) using an *arbiter* — a special component that can handle the meta-stable behaviour associated with such a choice. In such a case the behaviour of the circuit becomes non-deterministic.

When designing an SI circuit, the OP condition can always be imposed due to the modelling technique of factoring out the arbiter into the environment, converting thus the choice between local transitions into one between inputs (which is not a violation of OP). When modelling a biological system, violations of OP can be left in the model; however, any such violation should be looked at by the model designer and documented.

Note that arbitration should be used only for representing truly stochastic phenomena, like the choice between lysogeny and lysis modes in phage λ. Other violations of OP indicate that some important information is missing in the model, e.g., some assumptions about the environment's behaviour should be made, or the reaction rates can be used to resolve the choice. Methodologically, violations of OP are detected automatically, and if there are any, the user should either document the associated stochastic choice or refine the model, as we illustrated by an example in Section 3.

Complete State Coding (CSC) If the STG has two reachable states in which the values of all the signals coincide but the values of the next-state function for some local signal are different, then these two states are said to be in a *Complete State Coding (CSC)* conflict. The STG satisfies the *CSC property* if no two of its reachable states are in a CSC conflict.

An STG not satisfying the CSC property cannot be directly implemented as an SI circuit. Intuitively, during its execution the system 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. (One has to take care to preserve the consistency and other SI properties when inserting new signals.) 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 the internal workings of the system. 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. An STG with CSC conflicts might be useful in some cases as a high-level view of the system (in such a case all the internal signals can be *hidden* by PETRIFY in order to simplify the model), but if a detailed description of the system is needed, the STG should

satisfy the CSC property.

Note that STGs produced from circuits by the circuit-STG construction always have CSC. In fact, they satisfy a stronger property, called the *Universal State Coding (USC),* meaning that no two different states have the same values of all the signals, as in STGs derived from circuits using the described circuit-STG construction there is a one-to-one correspondence between the reachable markings and encodings. Furthermore, the FOE transformation preserves USC, as it can only eliminate reachable states, and never adds new ones. Though the FOE transformation does not in general preserve CSC (it can turn a USC conflict that is not a CSC conflict into a CSC conflict), the fact that USC implies CSC mean that all the STGs constructed during the proposed refinement procedure have CSC, if the initial STG was built from a circuit. Of course, if the initial STG has some other origin (e.g., it was constructed directly by the user) then CSC has to be separately checked.

The following properties are not directly required for SI, but their violation is nevertheless suspicious and might indicate a serious error in the model. At least, any violations of these properties should be documented by the model designer.

No self-triggering A signal is called *self-triggering* if firing one transition of this signal, e.g., a^+, can enable another transition of this signal, e.g., a^-.

Similarly to the violation of OP, self-triggering indicates that the corresponding signal might be pulled down (resp. up) before reaching its maximal (resp. minimal) value, and can also be interpreted in biological terms (see below). Self-triggering may also cause a CSC conflict, as the states before firing the first transition and after firing the second one have the same values of all the signals. It also manifests itself in the equation $[g_i] = E_i$ for the corresponding signal, as E_i is *binate* in g_i, i.e., both g_i and $\overline{g_i}$ occur in the minimised DNF of E_i.

In an STG modelling a biological system, self-triggering can sometimes be interpreted as missing auxiliary internal entities whose transitions would separate the pair of transitions involved in self-triggering.

Deadlock-freeness A reachable state is called a *deadlock* if no transition is enabled at it. It indicates that the system can stop functioning, which is probably not an intended behaviour in most realistic systems. [1] Note that the STGs produced from circuits by the circuit-STG construction are deadlock-free if there is at least one input, since inputs are allowed to oscillate freely. Moreover, the FOE transformation does not introduce new deadlocks, as it can disable only some (but never all) transitions enabled at any reachable state. [2] Of course, if the initial STG has not been generated from a circuit, but has some other origin (e.g., it was constructed directly by the user), then deadlock-freeness has to be separately checked.

Divergency-freeness An STG has *divergency* if, starting from some reachable state, it can execute infinitely many internal transitions. It indicates some infinite unproductive activity in the system, which nevertheless consumes resources.

[1] In some rare cases a deadlock-free circuit can be synthesised from an STG with deadlocks, but we do not elaborate such a case in this paper.

[2] If contradictory assumptions are simultaneously applied to the STG, a deadlock can be introduced, but such situations can be easily avoided.

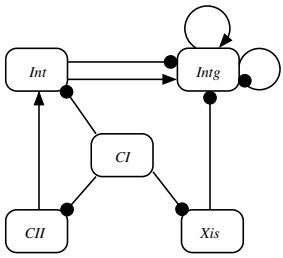
Checking the properties discussed above is automated by the STG support tool PETRIFY [7], and in the next section we show how to apply the developed theory to a biological system.

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 lysogeny-lysis switch in λ phage [12]. Using the Boolean model presented in [17] as a starting point, we construct and refine an STG model of this GRN, utilising the support tool PETRIFY [7]. 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.

Model Construction

The temperate bacteriophage λ [12] is a virus which infects the bacteria *Escherichia 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 [17]. 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 genes including those involved in its own excision. As such, the host cell, now a lysogen, establishes an immunity to external infection from other phages, and the prophage is able to lie dormant, replicating with each subsequent cell division of the host.



(a)

$$
\begin{aligned}
[CII] &= CI \\
[Int] &= CII + CI \\
[Xis] &= CI \\
[Intg] &= Intg\ Int + Intg\ (Int + Xis)
\end{aligned}
$$

(b)

Fig. 4. A high-level representation of the GRN of the phage λ switch (a); and the corresponding Boolean next-state equations (b).

A high-level pictorial representation of this GRN is presented in Fig. 4, along with the corresponding Boolean next-state equations describing the qualitative be-

haviour of each network entity [17]. 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 [17]. 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 [17].

From the Boolean network shown in Fig. 4, we are able to construct an STG describing the behaviour of the λ circuit using the circuit-STG construction.[3] We define *CI* as an input signal from the environment, *Intg* as the output signal produced by the circuit, and *CII*, *Int* and *Xis* as internal signals which are invisible to the environment. (As discussed earlier, this partitioning of signals is a decision which must be made by the modeller.) Furthermore, we choose the initial state[4] in which the values of all signals except *CI* are 0. Note that we allow *CI* to oscillate freely to represent the most general environment.

The resulting STG model is presented in Fig. 5(a). As explained in the previous section, STGs derived from circuits are bounded (in fact, safe), consistent, deadlock-free and have CSC, and these properties are preserved by the subsequent transformations.

Model Analysis and Refinement

We now analyse our STG model with respect to the properties introduced in Section 4. 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, there are a number of OP violations (resulting in non-deterministic behaviour) which suggests that some behaviour may not be realisable in practice:

(1) $Xis^+ \rightharpoonup CI^+$ (2) $Xis^- \rightharpoonup CI^-$ (3) $Int^+ \rightharpoonup CI^+$ (4) $Int^- \rightharpoonup CI^-$

(5) $CII^+ \rightharpoonup CI^+$ (6) $CII^- \rightharpoonup CI^-$ (7) $Intg^- \rightharpoonup Int^-$ (8) $Intg^- \rightharpoonup Xis^-$

(9) $Intg^+ \rightharpoonup Int^-$ (10) $Int^+/1 \rightharpoonup CII^-$

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 proceed by considering OP violations (1)-(6) which involve conflicts between input and local transitions. Such conflicts can often be resolved by assuming

[3] This model construction process from a Boolean network to an STG is fully automated by our prototype tool GNAPN, which is freely available for academic use at bioinf.ncl.ac.uk/gnapn.

[4] Choosing a meaningful initial state is outside the scope of this paper; we just remark that typically a biological system has cyclic behaviour, and any state on this cycle can be taken.

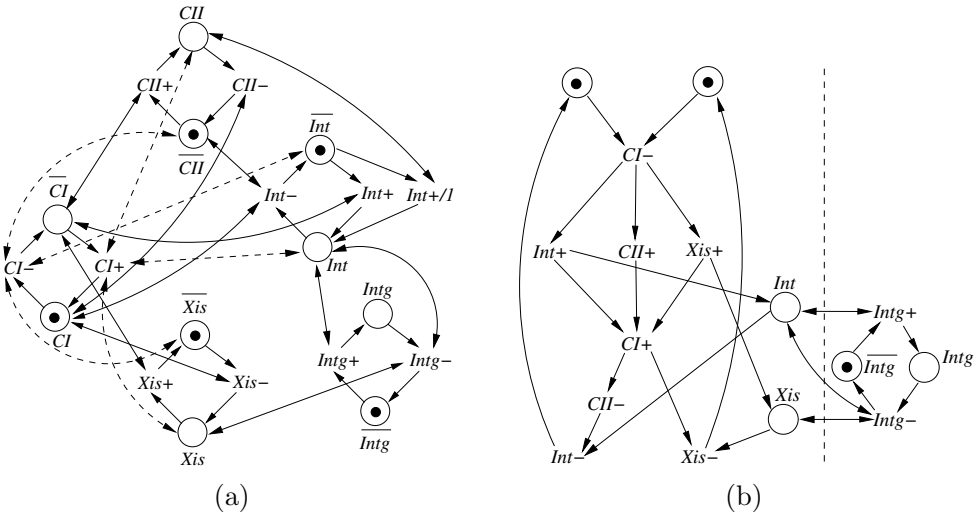


Fig. 5. STG representation for λ using circuit-STG construction, with the dashed arcs showing the FOE transformations expressing the relative slowness of the environment (a); and the STG simplified by PETRIFY (b).

that the environment is slow enough to allow the circuit to stabilise. We therefore apply the following FOE transformations to the model to resolve these violations:

$$Xis^+ \mapsto CI^+,\ Xis^- \mapsto CI^-,\ Int^+ \mapsto CI^+,\ Int^- \mapsto CI^-,\ CII^+ \mapsto CI^+,\ CII^- \mapsto CI^-,$$

which are also shown by dashed arcs in Fig. 5(a).

Interestingly, applying the above FOE transformations resolves also violation (10), 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 1 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 are at the heart of the lysis-lysogeny switch in λ (which is a stochastic phenomenon in practice [17]), and so are not resolved. The resulting STG is shown in Fig. 5(b) after simplification with PETRIFY.

The new STG in Fig. 5(b) is much less cluttered than the original one[5], as the unrealisable behaviour under the FOE transformations listed above has been stripped away, making it significantly simpler to interpret and analyse using e.g., model checking [6]. Moreover, this simplified STG clearly separates into two components, which capture the crucial mechanisms governing the lysis-lysogeny switch:

- Component 1 (left) 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.

[5] This is very typical, as the original STG contained a lot of (rather random) behaviour which is not realisable in practice, and hence was messy.

- Component 2 (right) 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 the pairs of arcs going in opposite directions 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 [17]. 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 Fig. 5(b) and involving 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 Fig. 4(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; indeed, its only role is to change the probabilities involved in the stochastic choice made by λ, and so it is no longer required once this stochastic choice has been qualitatively modelled by a non-deterministic one.

Finally, the output signal *Intg* in Fig. 5(b) is self-triggering (note that the corresponding next-state function 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 be used to identify areas which require careful documentation and further refinement in light of additional knowledge.

6 Conclusions

In this paper we have applied techniques and tools from asynchronous circuit design based on STGs [7] to modelling, visualising and analysing GRNs. Central to this has been the methodological assumption that biological systems can be modelled by speed-independent (SI) circuits [5, 7], and we investigated how the sufficient conditions required to ensure that an STG is implementable as an SI circuit can be interpreted in the biological setting. In particular, we have seen how violations of OP can be used to provide important insights into a model, by highlighting stochastic choices or areas that require refining.

The above framework was illustrated with a detailed case study, in which a refined SI STG model of the GRN for the lysis-lysogeny switch in phage λ [12,17] was developed using the support tool Petrify [7]. This process used STG techniques to remove unrealistic behaviour, making it easier to visually interpret the model

and, importantly, making it more amenable to automated analysis techniques, e.g., model checking [6]. Thus, STGs can be seen as providing a well supported formal framework for GRNs that allows realistic qualitative models to be developed and incrementally refined. We note that while the application of Petri nets to modelling biological systems has been widely considered (see for example [4]), our approach based on STGs and asynchronous circuit techniques appears to be new.

Further work is now needed to build on the initial ideas presented in this paper and to provide further tools to support the biologist applying these techniques. One particular interesting area currently being investigated is the application of STG techniques to *synthetic biology* [3]. Given that STGs were developed to support the compositional construction of asynchronous circuits, they appear to be ideally suited to designing artificial genetic control systems. Finally, we note that our approach can be extended to *multi-valued networks* [17] (i.e. where the Boolean state of signals is enlarged to a set of discrete values) in a number of ways, such as using several Boolean variables to represent a signal's state or reformulating the consistency rule on signal labels. Work is currently underway to investigate these approaches.

Acknowledgement

We would like to thank the EPSRC for supporting R. Banks and the BBSRC for their support via the Centre for Integrated Systems Biology of Ageing and Nutrition (CISBAN). This research was also supported by the Royal Academy of Engineering/ EPSRC post-doctoral research fellowship EP/C53400X/1 (DAVAC) and the Newcastle Systems Biology Resource Centre. Finally, we would like to thank the anonymous referees for their useful comments and suggestions.

References

[1] Akutsu, T., S. Miyano and S. Kuhara, *Identification of genetic networks from small number of gene expression patterns under the Boolean network model*, Proc. of Pac. Symp. on Biocomp. **4** (1999), 17–28.

[2] Bower, J.M., and H. Bolouri, *Computational Modelling of Genetic and Biochemical Networks*, MIT Press (2001).

[3] Bhutkar,A., *Synthetic Biology: Navigating the Challenges Ahead*, J. BioLaw and Bus. **8** (2005), 19–29.

[4] Chaouiya, C., *Petri net modelling of biological networks*, Briefings in Bioinformatics **8** (2007), 210–219.

[5] Chu, T.A., *Synthesis of Self-Timed VLSI Circuits from Graph-Theoretic Specifications*, MIT/LCS/TR-393 (1987), Lab. for Comp. Sci., MIT.

[6] Clarke, E.M., Grumberg, O., and Peled, D., *Model Checking*, MIT Press (1999).

[7] Cortadella, J., M. Kishinevsky, A. Kondratyev, L. Lavagno, and A. Yakovlev, *Logic Synthesis of Asynchronous Controllers and Interfaces*, Springer Series in Advanced Microelectronics **8** (2002), Springer.

[8] de Jong, H., *Modeling and simulation of genetic regulatory systems: a literature review*, J. of Comp. Bio. **9** (2002), 67–103.

[9] Huang, S., *Gene expression profiling, genetic networks, and cellular states: an integrating concept for tumorigenesis and drug discovery*, J. of Mol. Med. **77** (1999), 469–480.

[10] Muller, D. and W. Bartky, *A Theory of Asynchronous Circuits*, In Proc. Int. Symp. of the Theory of Switching (1959), 204–243.

[11] Murata, T., *Petri nets: properties, analysis and applications*, Proc. of the IEEE **77** (1989), 541–580.

[12] Oppenheim, A.B., O. Kobiler, J. Stavans, D. L. Court, and S. L. Adhya, *Switches in bacteriophage λ development*, Annual Review of Genetics **39** (2005), 4470–4475.

[13] Reisig, W., *Petri Nets, An Introduction*, EATCS Monographs on Theoretical Computer Science, W.Brauer *et al* (Eds.), Springer–Verlag, Berlin, (1985).

[14] Rosenblum, L., and A. Yakovlev, *Signal Graphs: from Self-Timed to Timed Ones*, Proc. of the Int. Workshop on Timed Petri Nets, IEEE Comp. Soc. Press (1985), 199–206.

[15] Steggles, L.J., R. Banks, O. J. Shaw, and A. Wipat, *Qualitatively modelling and analysing genetic regulatory networks: a Petri net approach*, Bioinformatics **23** (2007), Oxford University Press, 336–343.

[16] Szallasi, Z., and S. Liang, *Modeling the Normal and Neoplastic Cell Cycle with "Realistic Boolean Genetic Networks": Their Application for Understanding Carcinogenesis and Assessing Therapeutic Strategies*, Pac. Symp. on Biocomp. **3** (1998), 66–76.

[17] Thomas, R., and R. D'Ari, *Biological Feedback*, CRC Press (1990).

[18] *Petri nets World*, http://www.informatik.uni-hamburg.de/TGI/PetriNets/, (2008).