Qualitatively modelling and analysing genetic regulatory networks: a Petri net approach

L. Jason Steggles*, Richard Banks, Oliver Shaw and Anil Wipat
School of Computing Science, University of Newcastle, Newcastle upon Tyne, UK

Received on June 16, 2006; revised on November 6, 2006; accepted on November 18, 2006
Advance Access publication November 22, 2006
Associate Editor: Satoru Miyano

1 INTRODUCTION

New developments in post-genomic technology now provide researchers with the data necessary to study regulatory processes in a holistic fashion at multiple levels of biological organization. One of the major challenges for the biologist is to integrate and interpret these vast data resources to gain a greater understanding of the structure and function of the molecular processes that mediate adaptive and cell cycle driven changes in gene expression. In order to achieve this biologists require new tools and techniques to allow pathway related data to be modelled and analysed as network structures, providing valuable insights which can then be validated and investigated in the laboratory.

RESULTS: We propose a new technique for constructing and analysing qualitative models of genetic regulatory networks based on the Petri net formalism. We take as our starting point the Boolean network approach of treating genes as binary switches and develop a new Petri net model which uses logic minimization to automate the construction of compact qualitative models. Our approach addresses the shortcomings of Boolean networks by providing access to the wide range of existing Petri net analysis techniques and by using non-determinism to cope with incomplete and inconsistent data. The ideas we present are illustrated by a case study in which the genetic regulatory network controlling sporulation in the bacterium Bacillus subtilis is modelled and analysed.

AVAILABILITY: The Petri net model construction tool and the data files for the B. subtilis sporulation case study are available at http://bioinf.ncl.ac.uk/gnapn.

Contact: L.J.Steggles@ncl.ac.uk

*To whom correspondence should be addressed.

Copyright © The Author 2006. Published by Oxford University Press. All rights reserved. For Permissions, please email: journals.permissions@oxfordjournals.org
2 PETRI NETS

The theory of Petri nets (Reisig, 1985; Murata, 1989) provides a graphical notation with a formal mathematical semantics for modelling and reasoning about concurrent, distributed systems. A Petri net is a directed bipartite graph and consists of four basic components: places, which are denoted by circles; transitions denoted by black rectangles; arcs denoted by arrows; and tokens denoted by black dots. A simple example of a Petri net is given in Figure 1.

The places, transitions and arcs describe the static structure of the Petri net. Each transition has a number of input places (places with an arc leading to the transition) and a number of output places (places with an arc leading to them from the transition). We normally view places as representing resources or conditions and transitions as representing actions or events (Murata, 1989). Note arcs that directly connect two transitions or two places are not allowed.

The state of a Petri net is given by the distribution of tokens on places within it, referred to as a marking. The state space of a Petri net is therefore the set of all possible markings. The dynamic properties of the system are modelled by transitions which can fire to move tokens around the places in a Petri net. Transitions are said to be enabled if each of their input places contain 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. For example, in Figure 1 both transitions $t_1$ and $t_2$ are enabled. Firing transition $t_1$ would result in a token being taken from place $p_2$ and a new token being deposited on place $p_3$. Note that the firing rule is normally generalized to allow arcs to contain weights indicating the number of tokens to be consumed or produced when a transition is fired (Murata, 1989). Since such arc weights are not required in the sequel we have omitted them from our introduction for simplicity.

At any given instance a number of transitions may be enabled to fire (as in the example in Fig. 1). In such a case, a transition is chosen non-deterministically to fire. A marking $m_2$ is said to be reachable from a marking $m_1$ if there is a sequence of transitions that can be fired starting from $m_1$ which results in the marking $m_2$. A Petri net is said to be $k$-bounded if in all reachable markings no place has more than $k$ tokens. A Petri net which is 1-bounded is said to be safe. safeness is an important property since any safe Petri net is well-suited to automatic analysis (Reisig and Rozenberg, 1998).

An important advantage of Petri nets is that they are supported by a wide range of theoretically well-founded techniques and tools for simulation and analysis. For example, Petri nets can be automatically checked for boundedness and the presence of deadlocks (markings in which no transitions are enabled to fire) (Reisig and Rozenberg, 1998). A Petri net can also be analysed by constructing its reachability graph (Murata, 1989) which captures the possible firing sequences that can occur from a given initial marking. A range of techniques based on model checking (Esparza, 1994; Khomenko, 2003) have been developed for analysing reactivity properties and these provide a means of coping with the potentially large state space of a Petri net model.

3 MODELLING APPROACH

In this section we present a new qualitative model for genetic regulatory networks based on Petri nets. The idea is to use an intuitive Petri net structure to represent the Boolean relationships that exist between regulatory entities. We start by defining each entity’s individual behaviour as a truth-table following the approach used in Boolean networks (Akutsu et al., 1999). We then extract from these truth-tables the fundamental relationships between entities as Boolean terms by applying logic minimization techniques (Grossman, 2002; Breeding, 1992). In particular, we automate this process using the Expression (Brayton et al., 1984) logic minimization tool. We then directly translate these Boolean terms into appropriate Petri net control structures to produce a compact Petri net model that correctly captures the original Boolean behaviour of a genetic regulatory network.

Both the synchronous and asynchronous semantic interpretation of Boolean networks (Gershenson, 2002) can be modelled using our approach. We choose to focus on the synchronous semantics here and develop a simple two phase commit protocol to allow synchronized state updates within the asynchronous Petri net framework. Once constructed, our Petri net models can be simulated and analysed using the wide range of existing Petri net tools. These tools provide a range of powerful, theoretically well-founded analysis techniques (Reisig and Rozenberg, 1998; Esparza, 1994) which allow researchers to investigate the structure and dynamic behaviour of a genetic regulatory network.

3.1 Boolean networks

In a Boolean network (Kauffman, 1969; Akutsu et al., 1999) the state of each regulatory entity is represented as a Boolean value, 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 genetic regulatory network containing $n$ entities is then naturally represented as a Boolean vector $[g_1, \ldots, g_n]$ and this gives us a state space containing $2^n$ states (Akutsu et al., 1999). The behaviour of each $g_i$ is described using a Boolean function $f_i$ which, given the current states of the entities in its neighbourhood (i.e. those entities which directly affect it), defines the next state for $g_i$. As an example consider the Boolean network in Figure 2a (Akutsu et al., 1999) which contains three entities $g_1, g_2$ and $g_3$. The next state $g'_i$ of each entity is defined by the truth-table in Figure 2b or equivalently by the following Boolean functions:

$$ g'_1 = g_2, \quad g'_2 = g_1 g_3, \quad g'_3 = \overline{g_1} $$

where the notation $\overline{x}$, $x + y$ and $xy$ is used to represent the Boolean operators not, or and (Grossman, 2002), respectively. The dynamic behaviour of a Boolean network can be semantically interpreted either asynchronously where genes update their state independently, or synchronously where all genes update their state together (see Gershenson, 2002). We focus on the synchronous semantics in this paper which appears to be widely used in the literature (Bower and Bolouri, 2001; Gershenson, 2002). The synchronous behaviour for our example Boolean network is shown in the...
Boolean term for gene of an entity. Continuing with our example, we derive the following normal form (Grossman, 2002) that defines the functional behaviour or state 010 for the states 010, 011, 110 and 111 (where (Breeding, 1992; Grossman, 2002). From a biological point of logic minimization plex and can normally be simplified using.

Note that this term completely defines the functional behaviour of each entity. The idea is to consider the truth-table for each entity and to list all the states which result in a next state in which the entity is active (i.e. in state 1). For example, in the truth-table given in Figure 2b the entity is active after the states 010, 011, 110 and 111 (where \(xyz\) denotes the state \(g_1 = x, g_2 = y\) and \(g_3 = z\)). We can represent each state as a product term, called a minterm (Grossman, 2002), using the and Boolean operator, where the variable \(g_i\) is used to represent that an entity \(g_i\) is in state 1, and the negated variable \(\neg g_i\) represents that an entity \(g_i\) is 0. So the state 010 for \(g_1\) is represented by the minterm \(\neg g_1g_2g_3\). Applying this approach and then summing the derived minterms using the or Boolean operator allows us to derive a Boolean term in disjunctive normal form (Grossman, 2002) that defines the functional behaviour of an entity. Continuing with our example, we derive the following Boolean term for gene \(g_1\):

\[
\neg g_1g_2g_3 + \neg g_1g_2g_3 + g_1g_2g_3 + g_1g_2g_3
\]

Note that this term completely defines the functional behaviour of \(g_1\), i.e. whenever the term above evaluates to 1 in a state we know \(g_1\) will be active in the next state, and whenever the term is 0 we know \(g_1\) will be inactive. Using this technique we can construct a Boolean network that completely specifies the functional behaviour of a genetic network.

The Boolean terms derived above are often unnecessarily complex and can normally be simplified using logic minimization (Breeding, 1992; Grossman, 2002). From a biological point of view, this simplification process is important as it helps to identify the underlying regulatory relationships that exist between entities in a genetic network. The idea behind logic minimization is to simplify Boolean terms by merging minterms that differ by only one variable. As an example, consider the term \(\neg g_1g_2g_3 + \neg g_1g_2g_3\) which contains two minterms. Note that these two minterms differ by only one variable \(g_3\) and this implies that the value of \(g_3\) is unimportant in determining the value of the term. We can therefore simplify the term by merging the two minterms to produce a simpler term \(\neg g_1g_2\) which is logically equivalent (Grossman, 2002). For brevity we omit the full details of Boolean logic minimization here (we refer the interested reader to Breeding, 1992) and instead illustrate the idea by simplifying the Boolean terms in our running example:

\[
(g'_1) \quad \neg g_1g_2g_3 + g_1g_2g_3 + g_1g_2\neg g_3 + g_1g_2g_3 \\
\Rightarrow \neg g_1g_2 + g_1g_2g_3 + g_1g_2g_3 \\
\Rightarrow \neg g_1g_2 + g_1g_2 \Rightarrow g_2,
\]

\[
(g'_1) \quad g_1\neg g_2g_3 + g_1g_2g_3 \Rightarrow g_1g_3.
\]

Note that the minimized Boolean terms presented above correctly correspond to the Boolean network definition in Section 3.1.

3.2 Deriving Boolean regulatory relationships

Given a set of truth-tables defining the Boolean behaviour of a genetic network we would like to extract a compact representation of the regulatory relationships between entities. We address this using well–known techniques from Boolean logic (Breeding, 1992; Grossman, 2002) which allow us to derive Boolean terms describing the functional behaviour of each entity. The idea is to consider the truth-table for each entity and to list all the states which result in a next state in which the entity is active (i.e. in state 1). For example, in the truth-table given in Figure 2b the entity is active after the states 010, 011, 110 and 111 (where \(xyz\) denotes the state \(g_1 = x, g_2 = y\) and \(g_3 = z\)). We can represent each state as a product term, called a minterm (Grossman, 2002), using the and Boolean operator, where the variable \(g_i\) is used to represent that an entity \(g_i\) is in state 1, and the negated variable \(\neg g_i\) represents that an entity \(g_i\) is 0. So the state 010 for \(g_1\) is represented by the minterm \(\neg g_1g_2g_3\). Applying this approach and then summing the derived minterms using the or Boolean operator allows us to derive a Boolean term in disjunctive normal form (Grossman, 2002) that defines the functional behaviour of an entity. Continuing with our example, we derive the following Boolean term for gene \(g_1\):

\[
\neg g_1g_2g_3 + \neg g_1g_2g_3 + g_1g_2g_3 + g_1g_2g_3
\]

Note that this term completely defines the functional behaviour of \(g_1\), i.e. whenever the term above evaluates to 1 in a state we know \(g_1\) will be active in the next state, and whenever the term is 0 we know \(g_1\) will be inactive. Using this technique we can construct a Boolean network that completely specifies the functional behaviour of a genetic network.

While the Boolean terms derived in Section 3.2 compactly capture the behaviour of a Boolean network they are not amenable to analysis in their current form. We address this by translating these terms into appropriate Petri net control structures. The approach we take is to represent the Boolean state of each entity \(g_i\) in a Petri net by the well–known approach (see for example Reisig, 1985; Chauviya et al., 2004) of using two complementary places \(P_i\) and \(\overline{P}_i\), where a token on place \(P_i\) indicates the entity is active, \(g_i = 1\), and a token on place \(\overline{P}_i\) that it is not, \(g_i = 0\). Note the total number of combined tokens on places \(P_i\) and \(\overline{P}_i\) will therefore always be equal to 1.

Since Petri nets fire transitions asynchronously it is straightforward to model the asynchronous behaviour of a Boolean network in this setting (see Chauviya et al., 2004 for a related approach). We focus on modelling the synchronous behaviour of a Boolean network here and make use of a two phase commit protocol to synchronise updates in our model. In the first phase of the protocol each entity \(g_i\) in the model decides whether it should be active or not in the next state. This decision is recorded using two places, \(P_i\_On\) and \(P_i\_Off\), where a token on \(P_i\_On\) indicates \(g_i\) is active in the next state and a token on \(P_i\_Off\) that it is not. When all the entities have made a decision about their next state the second phase of the protocol begins and the state of each entity is synchronously updated.

Phase one: update decisions. We begin by considering under what conditions each entity is active (i.e. in state 1) in its next state and use the process detailed in Section 3.2 to derive a minimized Boolean term which compactly captures these conditions. We then model the minimized Boolean term for each entity \(g_i\) in our Petri net by adding a separate transition to represent each minterm it contains. The idea is that each transition will be enabled to fire,
placing a token on $\Pi_{i,On}$, precisely when the corresponding minterm is true. As an example, suppose the following minimized Boolean term
\[ \overline{x}_3 \overline{g}_1 + \overline{x}_3 g_2 \]
has been derived for a gene $g_2$. Then the first minterm $\overline{x}_3 g_1$ tells us that gene $g_1$ should be expressed in the next state (i.e. $g_1 = 1$) when genes $g_1 = 0$ and $g_2 = 1$ in the current state. We model this minterm using the transition $t_1$ depicted in Figure 3. This transition fires when places $P_x$ and $P_y$ contain a token (i.e. when $g_x = 0$ and $g_y = 1$) and results in a token being placed on $P_{x,On}$ (indicating that $g_x$ is expressed in the next state). Note the use of read arcs (Murata, 1989) here, i.e. bidirectional arcs which do not consume tokens but just check they are present. This ensures the tokens on places $P_x$ and $P_y$ are not removed at this stage (doing so would corrupt the current state of the genes). The start place $P_{x,Start}$ is used as a control input to the transition to ensure only one decision is made for gene $g_1$ during a single protocol step (update transitions for $g_1$ can only fire if a token is present on $P_{x,Start}$). The place $P_{x,Syn}$ is used to indicate when an update decision has been made for gene $g_1$, information needed by the protocol to determine when the first phase is complete. This process is then repeated to model the complete Boolean term for a given entity. In our example, this results in transition $t_2$ (Fig. 3) being added to model the minterm $\overline{x}_3 g_2$.

It remains to model the complementary decision procedure for deciding when an entity is inactive in the next state, that is when $P_{i,Off}$ should be marked. To do this we simply apply the process detailed above again but this time consider the conditions under which each entity becomes inactive. We then model the resulting Boolean terms as explained above except that this time each transition we add will mark place $P_{i,Off}$ instead of $P_{i,On}$.

**Phase two: synchronous state update.** After all the entities have made their update decisions all the synchronisation places will be marked and this allows the control transition depicted in Figure 4a to fire, initiating the second phase of the protocol. This phase performs a synchronised update step in which the state of each entity $g_i$ is updated in turn by placing a token on $P_i$ if place $P_{i,On}$ is marked or on $\overline{P_i}$ if place $P_{i,Off}$ is marked. An example fragment of the Petri net structure used for this update is given in Figure 4b for an arbitrary entity $g_i$. The fragment contains four transitions which represent the four possible update situations that can occur: move token from place $P_i$ to $P_l$; leave token on $P_i$; move token from place $P_l$ to $P_{i,On}$; leave token on $\overline{P_l}$. Only one of these transitions will be enabled to fire. Once the gene $g_i$ has updated its state a token is placed on place $P_{i,Done}$ to indicate that the next entity can be updated. When the last entity $g_n$ has been updated place $P_{n,Done}$ will be marked and the control transition depicted in Figure 4c initiates a reset step which re-marks the start places, allowing the whole synchronisation protocol to begin again.

### 3.4 Inconsistent and incomplete data

So far we have assumed that we are always able to derive complete and consistent truth-tables which correctly capture the behaviour of each entity in a genetic network. However, in practice it is rarely the case that a genetic network is fully understood and indeed, this is one important reason for modelling such networks. The data provided may be incomplete in the sense that information is missing about what happens in certain states, or it may be inconsistent in that we have conflicting information. The result is that the behaviour of some entities under certain conditions may be unknown.

Such incomplete and/or inconsistent behavioural information is problematic for standard Boolean network models which are unable to represent the possibility of more than one next state. However, Petri nets are a non-deterministic modelling formalism (Reisig, 1985) which are able to represent conflicting choices and unknown behaviour by incorporating all possible next state transitions. The idea is to identify for each entity all the problematic states in which the next state is unknown and then to include these states when deriving Boolean terms using logic minimization for both the active and inactive next state behaviour. The result will be a model in which for any unknown state there will be two conflicting transitions enabled, representing the choice between an active or inactive next state for the associated entity. In this way we allow these problematic states to exhibit both possible behaviours and rely on the non-deterministic choice mechanisms of the Petri net model.

As an example, consider the truth-table defining the behaviour of an entity $g_2$ presented in Table 1. The question mark indicates that when $g_1 = 1$ and $g_3 = 0$ the next state for $g_2$ is unknown. Using our approach, we add the minterm $\overline{g}_1 \overline{g}_3$ representing the unknown state to the terms derived to define the active and inactive next state of $g_2$.

![Fig. 3. Transitions modelling the Boolean term $\overline{x}_3 g_1 + \overline{x}_3 g_2$ for gene $g_2$.](image)

![Fig. 4. Petri net fragments for controlling synchronous updates.](image)

### Table 1. Truth-table for entity $g_2$

<table>
<thead>
<tr>
<th>$g_1$</th>
<th>$g_3$</th>
<th>$g_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>?</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
resulting in the following defining equations:

\[ g_2 = g_1g_3 + g_1\overline{g}_3, \quad \overline{g}_2 = \overline{g}_1 + \overline{g}_1g_3 + g_1\overline{g}_3. \]

Applying logic minimization allows us to simplify these to \( g_2 = g_1 \) and \( \overline{g}_2 = \overline{g}_1 + \overline{g}_1g_3 \) which will then be modelled using three transitions as shown in Figure 5. Note that transitions \( t1 \) and \( t3 \) are in conflict since they are both enabled to fire in state \( g_1 = 1 \) and \( g_3 = 0 \) (Fig. 5) and this represents the non–deterministic choice between the two possible next states for \( g_2 \) in this unknown state.

We can still perform meaningful analysis on the resulting Petri net models since the Petri net tools are able to cope with such non–deterministic choices. As more data becomes available for a genetic network the Petri net model can be refined to reduce the amount of non-determinism it contains and thus Petri nets provide an interesting means of documenting the development of knowledge about a genetic network.

3.5 Tool support for model construction

To support the modelling approach presented above we have developed a prototype tool to automate the model construction process. The tool takes as input a series of truth-tables describing the behaviour of the entities in a genetic network. These truth-tables are allowed to contain inconsistent and incomplete data as discussed in Section 3.4 above. From these tables the tool is able to automatically extract the minimized Boolean terms using the Espresso logic minimization tool (Brayton et al., 1984). It then uses these terms to construct a Petri net model of the given genetic network which can be based on either the synchronous or asynchronous Boolean network semantics (Gershenson, 2002). The resulting Petri net model can then be exported as a PNML (Petri net markup language) (Billington et al., 2003) file to a wide range of existing Petri net tools to be simulated and analysed. This prototype Petri net construction tool is freely available for academic use from our project website http://bioinf.ncl.ac.uk/gnapn.

4 CASE STUDY: SPORULATION IN \textit{B.subtilis}

In this section we illustrate our approach by presenting a case study in which part of the genetic regulatory network for initiating sporulation in \textit{B.subtilis} (Stragier and Losick, 1996; Stephenson and Lewis, 2005) is modelled and analysed. Using the detailed data provided in de Jong et al. (2004) we define the Boolean behaviour of the key regulatory entities involved in initiating sporulation using truth-tables. We then apply our prototype tool to automatically construct a qualitative Petri net model capturing the behaviour of the sporulation regulatory network. This Petri net is then validated and analysed using PEP (Grahlmann, 1997) and in particular, model checking techniques (Esparza, 1994) are applied to gain a deeper understanding of the model’s behaviour.

4.1 Regulating sporulation in \textit{B.subtilis}

The soil bacterium \textit{B.subtilis} is able to survive extreme conditions by forming dormant spores which are resistant to adverse environmental conditions (Stragier and Losick, 1996). This process of sporulation is controlled by a complex regulatory network, a small part of which is shown abstractly in Figure 6 (adapted from de Jong et al., 2004). The presence or absence of adverse environmental conditions, such as nutritional starvation, is represented by a signalling entity Signal; when Signal is active it indicates that the bacteria is under nutritional stress and needs to consider initiating sporulation. At the centre of the regulatory network is the phosphorylation of the protein SpoOA, which in turn activates a cascade of sigma factors which direct the transcription of genes that initiate sporulation. This phosphorelay (Stragier and Losick, 1996) 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 phosphatase SpoOE is able to reverse the phosphorylation of SpoOA, thereby inactivating SpoOA 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 (Stragier and Losick, 1996; Stephenson and Lewis, 2005).

4.2 Constructing the Petri net model

The process of constructing a Petri net model for a genetic regulatory network begins by identifying the entities (i.e. genes, proteins or stimuli) we need to consider. For our case study, there
are 12 entities (de Jong et al., 2004): SigF, KinA, SpoOA, SpoOA→P, AbrB, SpoOE, SigH, Hpr, SinR, SinI, SigA and Signal. In our Boolean model we consider each entity as either being in an on state (i.e. sufficiently present to have an affect) or an off state (i.e. not sufficiently present to have an affect). The last two entities, SigA and Signal, are not regulated within the scope of our network and as such their state remains fixed once it has been initialized. For each of the remaining 10 entities we use the available literature to construct a truth-table to abstractly define how each entity’s state is affected by the state of the entities in its neighbourhood. Note it is important that the information sources used are carefully documented so that we are able to clearly see the evidence and assumptions our resulting model is based on.

As an example, consider the entity AbrB. Using de Jong et al. (2004) and Figure 6 we can derive that its neighbourhood consists of three entities, namely SigA, AbrB and SpoOA→P. We can then consider how the state of AbrB is affected by these entities (de Jong et al., 2004; Stragier and Losick, 1996; Stephenson and Lewis, 2003, 2004). We begin by running a simulation test in which the model’s initial state is set to represent negative growth but Signal is activated to represent the presence of nutritional stress. The sequence of states resulting from this simulation is presented in Table 3, where the first column of the table represents the model’s initial state and each subsequent column the next state observed after a synchronized update. The simulation shows that SpoOA is initialize our model so that Signal is inactive, representing the absence of nutritional stress, and simulate the model again. The results correctly show that sporulation does not occur, i.e. SpoOA→P is not produced from the phosphorylation when Signal is inactive and so sigF is not expressed.

Table 2. Truth-table for AbrB

<table>
<thead>
<tr>
<th>SigA</th>
<th>AbrB</th>
<th>SpoOA→P</th>
<th>AbrB</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 3. Simulation results in the presence of nutritional stress

<table>
<thead>
<tr>
<th>Entity</th>
<th>States</th>
</tr>
</thead>
<tbody>
<tr>
<td>SigF</td>
<td>0 0 0 0 0 0 1</td>
</tr>
<tr>
<td>KinA</td>
<td>0 0 0 1 0 0 0</td>
</tr>
<tr>
<td>SpoOA</td>
<td>1 1 1 1 1 1 0</td>
</tr>
<tr>
<td>SpoOA→P</td>
<td>0 0 0 0 0 1 0</td>
</tr>
<tr>
<td>AbrB</td>
<td>1 0 1 0 0 1 0</td>
</tr>
<tr>
<td>SpoOE</td>
<td>1 0 1 0 1 0 0</td>
</tr>
<tr>
<td>SigH</td>
<td>0 0 1 0 1 0 0</td>
</tr>
<tr>
<td>Hpr</td>
<td>1 1 0 1 0 0 0</td>
</tr>
<tr>
<td>SinR</td>
<td>1 0 0 0 0 0 0</td>
</tr>
<tr>
<td>SinI</td>
<td>0 0 0 0 0 0 0</td>
</tr>
</tbody>
</table>

4.3 Model validation and analysis

Once we have constructed our Petri net model we can then validate and analyse it using the wide range of tools available for Petri nets. In this case study we use the PEP tool (see Grahlmann, 1997; Khomenko, 2003) to confirm that basic properties of the genetic regulatory network are respected. For example, we know that in the absence of nutritional stress sporulation should never be initiated (de Jong et al., 2004). We can use model checking to show this holds in our model by proving that no reachable state exists with SigF = 1 starting from any initial state in which Signal = 0, SigF = 0 and SpoOA→P = 0. Using a similar approach we can prove that sporulation can not occur in the absence of SigA (de Jong et al., 2004).

4.3.2 Property analysis

Having gained some confidence in the correctness of our model from the validation process above we can now consider a more detailed analysis of the properties from simple validation tests to more indepth gene knock out and overexpression analysis.

4.3.1 Model validation

We begin the analysis process by validating our model to ensure it is a reasonable representation of the regulatory system in question. The idea is to perform a range of tests on the model to ensure that it satisfies the basic behavioural properties indicated by the experimental results in the literature. In the case of the sporulation regulatory network the main control structure is the phosphorelay (de Jong et al., 2004; Stragier and Losick, 1996), which is responsible for the phosphorylation of SpoOA under nutritional stress conditions to activate the expression of sigF and so initiate sporulation. To validate that this control structure has been correctly modelled we formulate a range of tests on our Petri net model and compare the results with those presented in de Jong et al. (2003, 2004). We begin by running a simulation test in which the model’s initial state is set to represent negative growth but Signal is activated to represent the presence of nutritional stress. The sequence of states resulting from this simulation is presented in Table 3, where the first column of the table represents the model’s initial state and each subsequent column the next state observed after a synchronized update. The simulation shows that SpoOA→P accumulation is allowed to occur and SigF production is correctly activated indicating that the bacteria is able to sporulate. We then initialize our model so that Signal is inactive, representing the absence of nutritional stress, and simulate the model again. The results correctly show that sporulation does not occur, i.e. SpoOA→P is not produced from the phosphorelay when Signal is inactive and so sigF is not expressed.

We can further validate the model using the model checking tools provided by PEP (Grahlmann, 1997; Khomenko, 2003) to confirm that basic properties of the genetic regulatory network are respected. For example, we know that in the absence of nutritional stress sporulation should never be initiated (de Jong et al., 2004).
of the model. The idea is to begin by performing a range of simulations on the model, starting from biologically plausible initial states, and to then examine the results using standard techniques (Gershenson, 2002) to gain insight into the model’s behaviour. As an example, consider the simulation shown in Table 4 which indicates there is a strong attractor cycle of period two (Wuensche, 2003) (last two columns in table) which correctly describes the physiological conditions for the exponential growth phase (de Jong et al., 2004).

Next we consider using our model to investigate experimental hypotheses which we formulate by examining the available experimental literature and using the insights gained from the simulations that have been performed. For example, in our case study the available literature indicates an important relationship between SigF and SpoOA~P; the phosphorylated protein SpoOA~P is reported to activate the production of SigF but also repress its own production (de Jong et al., 2004). Thus, it seems likely that the two entities should never be active at the same time, a property known as mutual exclusion (Murata, 1989). We verify this property by using model checking to show that no reachable state exists in our model in which SigF and SpoOA~P are both active, specified by the following constraints:

\[(\text{SigF} + \text{SpoOA} \sim P) = 2, \quad \text{SinI}_\text{Done} = 1\]

Note the condition SinI_Done = 1 is used here to ensure we only consider states reached after a complete pass of the two-phase-commit protocol, where SinI is the last entity to be updated.

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 Tables 3 and 4) it appears that SpoOE and AbrB always have the same state. We can attempt to verify this hypothesis by using model checking to see if a contradictory state can be reached, which we specify using the following constraints:

\[(\text{SpoOE} + \text{AbrB}) = 1, \quad \text{SinI}_\text{Done} = 1\]

Model checking shows that such a contradictory state is in fact reachable, disproving our hypothesis. The tool returns a transition firing sequence which leads to a contradictory state in which SpoOE and AbrB have different states as a counter example and this can be automatically simulated in PEP giving important insight into how this behaviour occurs.

### 4.3.3 Mutant analysis

We complete the analysis by investigating the affect of ‘fixing’ a gene in the model to either be permanently active or inactive. This corresponds to the experimental approach of creating mutants in which genes are knocked out or overexpressed and is a useful analysis technique which can provide important insight into a genetic regulatory network. The idea is to construct a new model for each entity to be fixed where the input data used by the support tool has had the truth-table for the entity in question removed. As such the chosen entity will be treated as an input entity, as SigA and Signal are, and so its state will remain fixed once initialized. We have applied this technique to a range of entities in the sporulation model and then investigated how this affects the sporulation process using simulations and model checking. The observed results are summarised in Table 5 and correspond well with the experimental results available in the literature (see de Jong et al., 2004). 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 (de Jong et al., 2004).

### 5 DISCUSSION

In this paper we have developed a new approach to qualitatively modelling genetic regulatory networks based on using the well-developed formal framework of Petri nets. The results we have presented significantly extend existing work on using Boolean models to analyse genetic regulatory networks (e.g. Chaouiya et al., 2004). In particular, the key contributions of this paper are as follows: (1) A new compact approach to qualitatively modelling genetic regulatory networks based on using logic minimization and Petri nets; (2) Both synchronous and asynchronous semantics of Boolean networks (Gershenson, 2002) are catered for; (3) Approach able to model incomplete and inconsistent data; (4) Provision of tool support to automate model construction; and (5) A detailed case study investigating the application of Petri net tools to analyse a genetic regulatory network.

The case study we have presented provides important insight into the practical application of Petri net analysis techniques for validating and investigating models of complex regulatory systems. It highlighted the important role that model checking techniques can play in investigating a range of interesting model properties such as mutual exclusion. More work is needed to extend the set of biologically relevant properties that can be tested and the use of temporal logic as a property description language appears promising here (Chabrier-River et al., 2004).

<table>
<thead>
<tr>
<th>Entity</th>
<th>States</th>
</tr>
</thead>
<tbody>
<tr>
<td>SigF</td>
<td>0 0 0 0</td>
</tr>
<tr>
<td>KinA</td>
<td>0 0 1 0</td>
</tr>
<tr>
<td>SpoOA</td>
<td>1 1 1 1</td>
</tr>
<tr>
<td>SpoOA~P</td>
<td>0 0 0 0</td>
</tr>
<tr>
<td>AbrB</td>
<td>0 0 0 0</td>
</tr>
<tr>
<td>SpoOE</td>
<td>0 1 0 1</td>
</tr>
<tr>
<td>SigH</td>
<td>0 1 0 0</td>
</tr>
<tr>
<td>Hpr</td>
<td>0 0 1 0</td>
</tr>
<tr>
<td>SinR</td>
<td>0 0 0 0</td>
</tr>
<tr>
<td>Signal</td>
<td>0 0 0 0</td>
</tr>
</tbody>
</table>

### Table 4. Simulation results starting from a state without nutritional stress

<table>
<thead>
<tr>
<th>Entity</th>
<th>Knock out</th>
<th>Overexpressed</th>
</tr>
</thead>
<tbody>
<tr>
<td>KinA</td>
<td>No sporulation</td>
<td>Normal</td>
</tr>
<tr>
<td>SpoOA</td>
<td>No sporulation</td>
<td>Normal</td>
</tr>
<tr>
<td>AbrB</td>
<td>No sporulation</td>
<td>No sporulation</td>
</tr>
<tr>
<td>SpoOE</td>
<td>Normal</td>
<td>No sporulation</td>
</tr>
<tr>
<td>SigH</td>
<td>No sporulation</td>
<td>Normal</td>
</tr>
<tr>
<td>SinR</td>
<td>Normal</td>
<td>No sporulation</td>
</tr>
</tbody>
</table>

### Table 5. Results of mutant analysis
An important area not considered in this paper is the automatic extraction of truth-table definitions for entities in a genetic network. We are working to address this by linking our techniques with an existing network inference tool SARGE (Shaw et al., 2004). One drawback of using a Boolean model is that the high-level of abstraction used means we can fail to capture important dynamic network behaviour. We aim to address this by extending our work to multi-valued networks (Mishchenko and Brayton, 2002), with a particular emphasis on ensuring our Petri net models remain tractable to analysis. Finally, we intend to integrate our qualitative modelling tools into related work on stochastic Petri net modelling (Shaw et al., 2006) and so provide much needed support in this area.

ACKNOWLEDGEMENTS

We are very grateful to M. Koutny and V. Khomenko for many useful discussions and to the anonymous referees for their helpful comments. We thank the EPSRC for supporting R. Banks and the BBBSRC for support via the Centre for Integrated Systems Biology of Ageing and Nutrition (CISBAN). Finally we acknowledge the support of the Newcastle Systems Biology Resource Centre.

Conflict of Interest: none declared.

REFERENCES


