Identification of Transition-Based Models of Biological Systems using Logic Programming

Ashwin Srinivasan[1] Michael Bain[2]

[1] Department of Computer Science, IIIT, New Delhi, India
[2] School of Computer Science & Engineering, UNSW, Sydney, Australia

Technical Report
UNSW-CSE-TR-201425
December 2014

THE UNIVERSITY OF
NEW SOUTH WALES

School of Computer Science and Engineering
The University of New South Wales
Sydney 2052, Australia

Abstract

Transition systems like Petri nets have been widely used to model networks and to capture the dynamics of system behaviour. To date, these models have mostly been specified using specialised languages and associated simulators. In this paper we adopt the representation of first-order logic to specify a very general class of transition systems. Logical Guarded Transition Systems (or LGTSs) are characterised by the use of transitions that combine the usual linear numerical constraints associated with Petri nets with logical constraints (the "guard") expressed as a first-order formula. Our interest here is that this class of transition systems allows a very flexible way of specifying complex systems, such as large-scale biological networks. Using LGTSs we define the system-identification task in terms of logical consequence-finding, given domain-specific background knowledge and data on the system's behaviour. Consequence-finding by a logic-programming system is used to determine if there exists a finite-state automaton (FSA) that accepts a sequence of observational data S, given the background knowledge B. The output symbols of the FSA specify an LGTS consistent with the data. This basic approach is adequate to handle a number of situations like the hierarchical construction of large networks, the use of domain-knowledge to constrain answers, and the hypothesis of missing states. We also describe how the deductive machinery can be augmented using abduction and induction to deal with deficiencies in the data and background knowledge. Using a number of classical networks from the literature, we demonstrate the identification of: pure Petri nets; extended Petri nets; networks that re-use sub-nets; network from data with missing values; and networks requiring new transitions. The results suggest that LGTSs and the logical formulation of system-identification can be used to obtain qualitative network models for a wide variety of biological systems, and is sufficiently general to apply to areas other than biology.

1 Introduction

Networks are ubiquitous in Biology. They are used to represent biological relationships ranging across all levels of organisation: for example, relationships between organisms, and between an organism and its environment; the flow of energy and matter in an ecosystem; the pathway of carbon atoms through an ecosystem from producers of organic compounds to consumers that release carbon by respiration; the nitrogen cycle that links the environment to proteins and compounds that form the bodies of living things; the stimulus-response mechanisms in constituting nervous pathways; the regulation and control of endocrine glands; the events related to the division and replication of cells; and intra- and inter-cellular interactions between chemicals.

Computationally, substantial research effort has been, and continues to be invested in developing models of biological networks (Junker and Schreiber, 2008). While much of this research has been directed at representation and reasoning, the field of Systems Biology (Ideker et al., 2001) has highlighted the need to extract automatically models of networks from experimental data. The requirement is for mathematical models that not only determine the underlying relationships amongst entities, but are also capable of simulating the dynamics of the system. The choice of models that naturally comes to mind first is that of differential equations (both ordinary, or ODEs, and partial, or PDEs). There is a tradition stretching back to Alan Turing that has attempted to understand biological systems by the use of PDEs, but not much has been done to extract such models automatically from data (however see, for example, Perkins et al. (2006)). Using a logical abstraction of differential equations, Inductive Logic Programming (ILP) systems have managed to identify qualitative differential equations, or QDEs (Srinivasan and King, 2008). The representation of QDEs provides a direct and simple abstraction of quantitative ODEs. However the representation does have limitations. First, simulation can produce spurious behaviour, arising from the ambiguities inherent in the qualitative approach. Second, issues of concurrency, which are prevalent in biological systems, are not well handled. Third, there appears to be no straightforward mechanism of introducing any form of quantitative information. Fourth, there is little room for accounting for stochastic aspects inherent in the system.

Most of these issues are largely absent in the long-established qualitative representation of Petri nets. Starting from a simple bipartite graph representation that is ideally suited for metabolic networks, Petri nets have been extended in a number of ways that are of interest for biological networks. This incorporates timing (timed Petri nets), concentrations (continuous Petri nets), stochasticity (stochastic Petri nets), multiple levels of organisation (hierarchical Petri nets), activation and inhibition relations needed for signalling networks (Petri nets with "read" and "inhibitor" arcs), and so on. For reasons of space, we do not elaborate on these here, but refer the reader to (David and Alla, 2010). Mathematically, the power of Petri nets ranges from simple qualitative producer-consumer models to that of quantitative ODEs. Computationally, the range is from above regular languages to Turing machines (Peterson, 1981). We note here that the basic Petri net structure and its extensions have found widespread use in representing networks in Biology: (Koch et al., 2011) provides an excellent summary of their use in representing metabolic, signalling and genetic networks. Most of this work has been concerned with hand-crafted Petri

net models, specified using a specialised representation language and associated simulators.

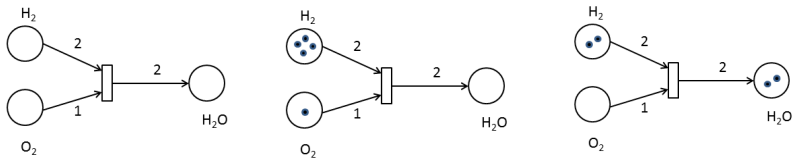
The work in this paper takes a different route. Using the language of logic programs, we describe a form of transition system called a Logical Guarded Transition System (LGTS) (Srinivasan and Bain, 2012). Both pure and extended Petri nets are special kinds of LGTSs. We formulate identification of such systems from data using the logical consequence relation. By adopting the language of logic programs, we are able to employ the standard computational machinery of refutation-based theorem proving to find solutions to the system-identification task. This provides a clear semantics to the enumerative techniques employed in the Petri net literature. Second, we are able to use well-established network models as background knowledge to identify structured network models representing complex biological systems. Third, the use of logical constraints enables us to go beyond learning pure and extended Petri nets, with tests on activation or inhibition states, but also to specify, for example, domain-dependent properties that should be true for any identified system. This allows us to identify not only metabolic networks but signalling networks also. Fourth, we are able to draw on work done in extending logic programs with abduction and induction to extend system-identification to model construction when there are significant deficiencies in background knowledge and the data. We demonstrate each of these advantages using some well-known biological networks. The principal contributions of this paper are as follows:

- To the literature on system identification we provide a logical formulation for identifying pure and extended Petri nets, along with even more general forms of transitions than activator and inhibitor arcs.
- To the logic-programming literature we present the use of Logic Programming and its extensions to the task of identifying large dynamic models of systems, using a combination of logic-programs, abduction and induction.
- To the biological literature we provide a mechanism of learning efficiently using a representation that is a reasonable mixture of qualitative and quantitative aspects, and which can be extended in either direction quite easily. We also provide a principled manner in which learning can scale with the size of the system being studied (that is, networks constructed previously become new transitions at the next level of abstraction).

The rest of the paper is organised as follows. In Section 2 we introduce LGTSs by way of some simple examples. Section 3 formulates LGTSs as a logic program. The system identification task is presented in Section 4, and extended to handle identification with incomplete information in Section 5. Section 6 discusses related work and Section 7 concludes the paper. Appendix A contains details on data used and networks identified, and Appendix B gives a formal description of Petri nets and their representation as Logical Guarded Transition Systems.

2 Transition Systems: An Example

Transition systems are a general formalism for representing the dynamic behaviour of a system. They are used to represent both sequential and concurrent



(a) (b) (c)

Figure 2.1: (a) A simple Petri net representing the reaction $2H_2 + O_2 \rightarrow 2H_2O$; (b) An "initial marking", in which molecules of hydrogen and oxygen are shown by tokens (small solid circles); (c) A "final marking", which results in two molecules of water, from the molecules of hydrogen and oxygen in (b).

events in areas ranging from programming language semantics, to business process models and network models in biology. Petri nets are a widely used type of transition system[1]. Fig. 2.1 shows a simple Petri net. Petri nets have two kinds of nodes. Conventionally, the circular nodes are called *places* and the rectangular nodes are called *transitions*. Edges can only exist between a place and a transition or *vice versa* (but never from one place to another, or from one transition to another: the structure is thus a bipartite graph), and each edge has a weight (or label: by convention, if a label is absent, then the weight is taken to be 1). A transition thus has a finite number of input places and a finite number of output places. Places can contain 0 or more tokens (usually shown as small black circles, as in Fig. 2.1(b)), and the dynamics of the system are described by the *firing* of transitions and the movement of tokens from one place to another. A transition is *enabled* if the number of tokens at each input place for the transition is at least equal to the weight of the arc from the place to the transition (a transition with no input places is always enabled). An enabled transition can *fire*, resulting in consuming tokens from an input place and depositing tokens in an output place: the numbers of tokens consumed and deposited being determined by the arc weights. The state of the Petri net at any point in time is the number of tokens at each place, and is called a *marking*.

The structure of a Petri net can be described using a matrix called its "incidence matrix". The column vectors of the incidence matrix of a Petri net represent the transitions. An entry (i, j) in the matrix denotes the net transfer of tokens at place p_i when transition t_j fires: see Fig. 2.2(a) for the simple Petri net we have just shown. Since we focus on systems biology in this paper we will often refer to the column vectors as reaction vectors.[2]

A change in state due to the firing of a single transition t can be written as a vector equation involving the vectors representing the marking before and after the transition fires, along with the reaction vector for the transition. In general, a change from one state to another may be due to the firing of a sequence of transitions. This can still be represented as a vector equation involving the state and reaction vectors involved: see Fig. 2.2(a)–(c). This expression does not uniquely determine the incidence matrix: it is possible for several incidence

[1]Formalisation of Petri nets, and their relation to LGTSs, is in Appendix B.

[2]In systems biology the incidence matrix represents the *stoichiometry* of the system, i.e., the relative quantities of all molecular species in each of the reactions in the system (Palsson, 2006). In this case tokens might represent concentration levels of a molecular species in a reaction, discretized in a suitable manner.

matrices (and hence, pure Petri nets) to express a difference in this manner (see Fig. 2.2(d). This makes the problem of identifying Petri nets from observed data an under-constrained one.

It is evident from Fig. 2.1 that this "token game" is ideally suited for chemical reactions in which reactants are consumed and products are produced. However not all reactions are that simple. In many cases, due to energy constraints, the reactants will not combine to produce the products. However the presence of a catalyst can greatly lower the energy requirements of a reaction, and allow it to proceed. For example, energy requirements prevent the spontaneous splitting of water into hydrogen and oxygen (as is evident from the world around us). However in the presence of a catalyst, and some additional energy, the reaction can happen. Fig. 2.3(a) shows this as an "extended" Petri net, in which the catalyst enables the reaction, without being used up itself (surface catalysis is intended here). That is, as long as a token in present at the place representing the catalyst, and adequate tokens exist at the input places, the transition will fire. That the catalyst is not used up is captured by the token at the catalyst-place not being consumed by the firing of the transition. The bi-directional arc is an example of the "read" arc mentioned earlier). Unfortunately, the presence of bidirectional arcs means that that incidence matrix representation is no longer possible, since entries in the matrix have to be either positive or negative but not both at once[3].

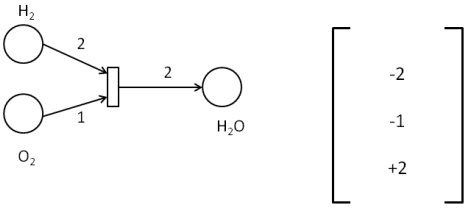
Suppose now that we want to enforce some additional constraints on the firing of the transition. For example, the catalyst for splitting water into hydrogen and oxygen may only function at some environmental conditions. With effort, this could be represented in some cases by a combination of read-arcs, and input places, but the representation becomes cumbersome. It is more natural to associate instead of transitions with logical constraints that can always be evaluated to be true or false. A transition that fails to satisfy its constraints does not fire: see Fig. 2.3(b). In this paper, we introduce transition-systems in which each transition has 3 kinds of constraints, representing pre- and post-conditions, and invariants of the transition. Such transition systems are called here Logical Guarded Transition Systems, or LGTSs.

It can be seen without too much effort that both pure and extended Petri nets are special kinds of LGTSs (this is formalised in Appendix B). For example:

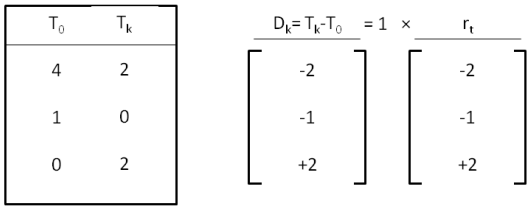
- A pure Petri net is an LGTS in which pre- and post-conditions and invariants are all trivially $TRUE$.
- An extended Petri net with an input-place that has a read arc is an LGTS in which the pre-conditions ensure that the input place has some tokens; and the invariant ensures that the number of tokens remains constant.

It is convenient to think of an LGTS as a pure Petri net, augmented with logical constraints. With an LGTS, there are now two kinds of constraints to be satisfied: the linear constraints associated with incidence matrix of the pure Petri net component, and the logical constraints associated with each transition.

[3]It may be possible in some cases to introduce intermediate places and transitions to rewrite the bi-directional arc. For example, with surface catalysis, we can think of the combined structure of the reactants and the catalyst as an intermediate entity. Transitions can be introduced to represent the formation and dis-association of this intermediate.



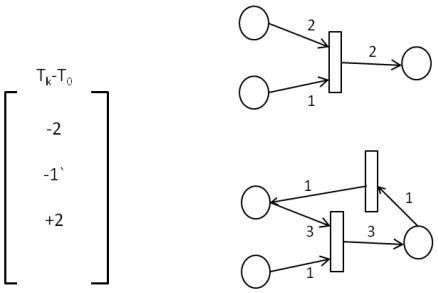
(a)



(b)

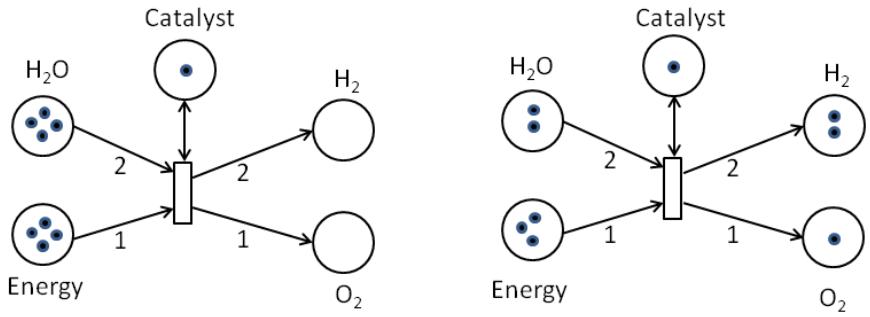
$$m' - m \;=\; \sum_{r \in M} \lambda_r r$$

(c)

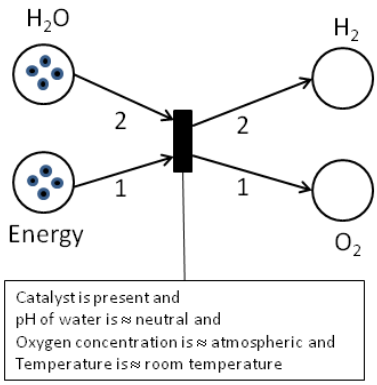


(d)

Figure 2.2: (a) Incidence matrix representing a Petri net; (b) The tokens (markings) at each place at times T_k and T_0. The difference in markings D_k can be represented as a vector equation involving the marking and reaction vectors; (c) In general, the (vector) difference between two markings m' and m can be represented a linear combination of reaction vectors in the incidence matrix; (d) A difference between two markings may be represented as the linear combination of reaction vectors of more than one Petri net (the incidence matrices of either net shown will satisfy this requirement).



(a)



(b)

Figure 2.3: (a) An extended Petri net representing the reaction $2H_2O + Energy \rightarrow 2H_2 + O_2$. The diagram on the left shows the initial marking—each token associated with Energy represents some minimum amount of energy needed for the catalysed reaction to proceed. The diagram on the right shows the final marking, which results in two molecules of hydrogen, and one of oxygen. (b) An LGTS representing the same reaction, with some logical constraints associated with the transition. If the formula in the "constraint box" evaluates to true, then the transition fires and tokens are transferred as usual. In general, the constraint box for a transition t will contain a formula $Guard_t(s_i, s_f) \equiv PreCondition_t(s_i) \wedge PostCondition_t(s_f) \wedge Invariant_t(s_i, s_f)$. Here s_i and s_f are states of the system before and after the transition fires. In this paper, we will show LGTS transitions as coloured rectangles.

Of course, we are not interested simply in syntactic sugar. The value of an LGTS is that allows the specification of more elaborate kinds of constraints. For example:

- A water-splitting catalysis reaction must use catalysts that are capable of working with pH neutral water, atmospheric oxygen, and room temperature.
- Reactions that break 3 or more chemical bonds require too much energy.
- All reactions must have no more than 1 or 2 reactants and 1 or 2 products.

These kinds of constraints, amongst others, can be encoded quite simply in an LGTS.

3 Transition Systems as Predicates

3.1 Specification

A pure Petri net is completely defined by the set of reaction vectors in its incidence matrix.

Definition 1. (Transition function) *Given the set of states S and reaction vectors R, we can represent a transition in a pure Petri net as the function $t : S \times S \times R \to \{TRUE, FALSE\}$, defined as:*

$$t(s1, s2, r) = \begin{cases} TRUE & \text{if}(s2 - s1 = r) \\ FALSE & \text{otherwise} \end{cases}$$

We can define a pure Petri net as a function, as follows.

Definition 2. (Pure Petri net) *We can represent a pure Petri net as the function $pn : S \times S \times 2^R \to \{TRUE, FALSE\}$, defined as:*

$$pn(s1, s2, m) = \begin{cases} TRUE & \text{if } (m = \{r_1, r_2, \ldots, r_n\}) \text{ and} \\ & \quad \exists m_0, m_1, \ldots, m_n \in S \text{ s.t.} \\ & \quad\quad ((m_0 = s1) \wedge (m_n = s2) \\ & \quad\quad\quad \bigwedge_1^n (t(m_{i-1}, m_i, r_i) = TRUE)) \\ FALSE & \text{otherwise} \end{cases}$$

While there is no immediate way of representing extended Petri nets with the incidence matrix representation, the definition of their transitions in the logical setting is a natural extension to the function of Definition 1.

Definition 3. (Extended transition function) *Given the set of states S and reaction vectors R, we can represent a transition in an extended Petri net as a transition t with an associated Boolean function f_t on states, defined as:*

$$t(s1, s2, r) = \begin{cases} TRUE & \text{if } (f_t(s1, s2) = TRUE) \wedge (s2 - s1 = r) \\ FALSE & \text{otherwise} \end{cases}$$

If t is a transition that contains a read arc from place p, then $f_t(s1, s2)$ is TRUE iff p has non-zero, identical values in states $s1$ and $s2$, and if t is a transition that contains an inhibitor arc from place p, then $f_t(s1, s2)$ is TRUE iff p has value zero in states $s1$ and $s2$. Otherwise $f_t(s1, s2)$ is TRUE.

The logical definition of a Petri net remains unchanged with these extensions. However, the change to a logical representation allows us to go much further than transitions with activator and inhibitor functions. We are, in fact, able to represent pure and extended Petri nets as special forms of logical guarded transition systems.

We now generalise the transition function of Definition 3 to a *logical guarded* transition function. First we define guards.

Definition 4. (Guard function) *Given the set of states S, the set of possible transitions T and predicates on pre-conditions, post-conditions and invariants of transitions, we can represent the guard on a transition as a function $g : T \times S \times S \to \{TRUE, FALSE\}$, defined as:*

$$g(t, s1, s2) = \begin{cases} TRUE & \text{iff } ((pre_t(s1) \wedge inv_t(s1, s2) \wedge post_t(s2)) = TRUE) \\ FALSE & \text{otherwise} \end{cases}$$

where pre_t represents the pre-condition for transition t, $post_t$ the post-condition for t, and inv_t the invariant.

With the concept of logical guards, or simply guards, we can now further generalize Definitions 1 and 3 for transition functions.

Definition 5. (Guarded transition function) *Given the set of states S, the set of reaction vectors R and the set of possible transitions T, we can represent a guarded transition as a function $gt : S \times S \times T \times R \to \{TRUE, FALSE\}$, defined as:*

$$gt(s1, s2, t, r) = \begin{cases} TRUE & \text{if}(g(t, s1, s2) = TRUE) \text{ and } (s2 - s1 = r) \\ FALSE & \text{otherwise} \end{cases}$$

Analogous to Definition 2 of Petri nets as a function, we can now define logical guarded transition systems.

Definition 6. (Logical guarded transition systems) *Given the set of states S, the set of reaction vectors R and the set of subsets of tuples of possible transitions T and reaction vectors R, we can represent a guarded transition system as the function $lgts : 2^{S \times S} \times 2^{T \times R} \to \{TRUE, FALSE\}$ defined as:*

$$lgts(s, t) = \begin{cases} TRUE & \text{if } (\forall (s_i, s_j) \in s \; gts(s_i, s_j, t_{i,j}) = TRUE) \text{ and} \\ & \qquad t = \bigcup t_{i,j}) \\ FALSE & \text{otherwise} \end{cases}$$

where:

$$gts(s1, s2, m) = \begin{cases} TRUE & \text{if } (m = \{(t_1, r_1), (t_2, r_2), \ldots, (t_n, r_n)\}) \text{ and} \\ & \quad \exists m_0, m_1, \ldots, m_n \in S \text{ s.t.} \\ & \quad\quad ((m_0 = s1) \wedge (m_n = s2) \\ & \quad\quad\quad \bigwedge_1^n (gt(m_{i-1}, m_i, t_i, r_i) = TRUE)) \\ FALSE & \text{otherwise} \end{cases}$$

Note that we will treat sets of transitions interchangeably with sequences of transitions in the remainder of the paper, for example in the definition of LGTSs as logic programs.

Transition-Reaction Pairs as Graphs

Suppose the system has places $p_1, p_2, \ldots, p_k$. In this paper, each transition t in a (t, r) pair is a (logical) term. Any transition t transforms the system from state s_i to state s_j only if $g(t, s_i, s_j) = pre_t(t, s_i) \wedge post_t(s_j) \wedge inv_t(s_i, s_j)$ is $TRUE$. A special term *anonymous* will be taken to denote a transition whose guard is trivially $TRUE$. Each reaction r in a (t, r) pair will be a k-dimensional vector of integers. With some abuse of notation, let $Input(r)) = \{p_i : x_i \in r \text{ s.t. } x_i < 0\}$ and $Output(r) = \{p_i : x_i \in r \text{ s.t. } x_i > 0\}$. Further, let $h = hash((t, r))$ be an injective hash function that returns a (unique) non-negative natural number for each distinct (t, r) pair.

Now, with each transition-reaction pair (t_i, r_i) we are able associate a bipartite digraph $G_i = (U_i, V_i, E_i)$, consisting of vertices $U_i = Input(r_i) \cup Output(r_i)$ and $V_i = \{hash((t_i, r_i))\}$. Let the set of ordered pairs $E_i = \{(v_i, t_i) : v_i \in Input(r_i)\} \cup \{(t_i, v_i) : v_i \in Output(r_i)\}$ denote the edges of the digraph. It is straightforward to extend this to digraphs with weighted edges, but we do not do so here.

A set $T = \{(t_1, r_1), (t_2, r_2), \ldots, (t_n, r_n)\}$ is then associated with the graph $Graph(T) = G_1 \cup G_2 \cup \cdots G_n$, where the union of graphs is defined in the usual manner (that is, union of vertices and edges).

Drawing Conventions

For a bipartite digraph $G = (U, V, E)$ we will show each vertex in U diagrammatically as a circular node labelled by the vertex. For each vertex $v \in V$ let $(t, r) = hash^{-1}(v)$. The following conventions are are adopted when drawing the vertex v:

- If $t = anonymous$ then the vertex v is shown diagrammatically as an unfilled rectangular node, denoting that the guard for v is trivially $TRUE$;
- Otherwise, if t represents a transition activated by some place p, usually denoted in this paper by $read(p)$, the vertex v is shown as a rectangular node connected by a doubly-directed edge to a circular node representing the place p;
- Otherwise, if t represents a transition inhibited by some place p, usually denoted in this paper by $inhibit(p)$, the vertex v is shown as a rectangular node connected by a hammer-head edge to a circular node representing the place p;

– Otherwise, the vertex v is shown by a filled rectangle, denoting that there are some logical conditions that need to be satisfied.

We emphasise that these are just drawing conventions—all transitions are treated the same way in practice, requiring their guards to be $TRUE$ before they can "fire".

3.2 Implementation

In this section we present an implementation of the specification of LGTSs, using the syntax of logic programs. It is useful to implement the *gts* function as a finite state automaton *fsm*(*start_state*, *final_state*, *transition_sequence*), that denotes that a finite-state automaton (FSM) starting in *initial_state* will terminate in *final_state* having generated *transition_sequence* as output. An acceptor variant is simply an automaton that answers "yes" for all triples in the relation and "no" otherwise. An equivalence between FSMs and Petri nets has been established in (Takahashi and Takahara, 1995).

We will see below that *transition_sequence* is a sequence of (t, r) pairs, where t denotes a transition and r a vector denoting a difference between a pair of states. In the final answer computed for the $lgts$ function, the ordering of transition-reaction pairs is unimportant.

An example of system behaviour will be provided by a sequence of states (in Petri net terminology, each state is a "marking"). Thus observations of a system's behaviour will be of the form:

Place	States					
	0	1	2	...	...	n
p_1	$s_{1,0}$	$s_{1,1}$	$s_{1,2}$	...	...	$s_{1,n}$
p_2	$s_{2,0}$	$s_{2,1}$	$s_{2,2}$	...	...	$s_{2,n}$
...	...	...	...	...	...	...
...	...	...	...	...	...	...
p_l	$s_{l,0}$	$s_{l,1}$	$s_{l,2}$	...	...	$s_{l,n}$

Any pair of adjacent states in the sequence have to be distinct, and the states need not necessarily be observed at equidistant time instants. An LGTS will be defined as a relation that holds over a pair of sets S and T. S is a a set of elements (s_i, s_{i+1}) where s_{i+1} is a successor state of s_i in the observations of system behaviour. T is a set of transition-reaction pairs, Definitions of the FSM and and an LGTS in a Prolog-like language[4] are as follows:

$fsm(S_i, S_f, [T_s]) :-$
  $S_i \neq S_f,$
  $reachable(S_i, S_f, T_s).$

$fsm(S_i, S_f, [T_s|TRs]) :-$
  $S_i \neq S_f,$
  $reachable(S_i, S_j, T_s),$
  $fsm(S_j, S_f, TRs).$

$lgts([], []).$

$lgts([(S1, S2)|States], T) :-$
  $fsm(S1, S2, TR_1),$
  $lgts(States, TR_2),$
  $conc(TR_1, TR_2, T).$

where [] denotes the empty list, and *conc* denotes the usual concatenation function over lists (we will henceforth use a list, set and sequence notation interchangeably unless explicitly stated otherwise). *reachable* is a relation that is

[4]Capitalised letters and words denote variables; predicates and function symbols are in lower-case; $:-$ denotes the $\leftarrow$ connective; and [...] denotes a list (sequence) of elements.

true for all state-pairs $S_{i,j}$ and a transition-reaction T_s such that it is possible to reach state S_j from state S_i using exactly the transition and reaction T_s:

$$\begin{array}{l} reachable(S_i, S_j, T_s) :- \\ \quad trans((T, R), S_i, S_j), \\ \quad S_i \neq S_j, \\ \quad T_s = (T, R). \end{array}$$

where *trans* is true for all transition-reaction tuples (t, r), that hold for a state pair (S_i, S_j). In the tuple (t, r), t is a term representing the transition and r is a (difference) vector. The transition-reaction tuples thus encode a combination of logical and arithmetic constraints on state-pairs:

$$\begin{array}{l} trans((T, R), S_i, S_j) :- \\ \quad transition(T), \\ \quad place\_vector(R), \\ \quad guard(T, S_i, S_j), \\ \quad diff(S_j, S_i, R). \end{array} \qquad \begin{array}{l} guard(T, S_i, S_j) :- \\ \quad pre(T, S_i), \\ \quad post(T, S_j), \\ \quad inv(T, S_i, S_j). \end{array}$$

We will assume that both *transition* and *place_vector* are able to act as generators of suitable ground terms for transitions and vectors, and that $diff(S_j, S_i, R)$ denotes the relation $R = S_j - S_i$, which can be solved, given any two legal arguments, to obtain the third legal argument (a linear constraint solver would do this automatically). With this generative machinery in place, there is no requirement for the observed sequence of states to be complete. For example, if instead of the sequence $S = (s_1, s_2, \ldots, s_{i-1}, s_i, s_{i+1}, \ldots, s_k)$, we are given instead $S' = (s_1, s_2, \ldots, s_{i-1}, s_{i+1}, \ldots, s_k)$. This follows from the definition of the FSM that checks to see if there is a path from a pair of states s_{i-1}, s_{i+1} through some intermediate state s_i

It will be useful to encode some domain-independent constraints relevant to system identification within the *trans* predicate. Specifically, following (Durzinsky et al., 2008), we will introduce the notion of terminal (sink) states, and that of a maximal set $R_{i,j}^{max}$ of vectors consistent with $S_j - S_i$. Both these can be incorporated by rewriting the *trans* definition:

$$\begin{array}{l} trans((T, R), S_i, S_j) :- \\ \quad transition(T), \\ \quad diff(S_j, S_i, D), \\ \quad in\_box(D, R), \\ \quad transition\_ok((T, R), S_i, S_j). \end{array}$$

$$\begin{array}{l} transition\_ok((T, R), S_i, S_j) :- \\ \quad not(changes\_terminal\_state((T, R)), \\ \quad guard(T, S_i, S_j). \\ \\ changes\_terminal_state((T, R)) :- \\ \quad terminal\_state(S), \\ \quad diff(S', S, R), \\ \quad guard(T, S, S'). \end{array}$$

Legal transitions therefore satisfy their guards, and also do not change any known terminal state of the system. *in_box* enumerates elements of the maximal set of feasible difference vectors given the observed difference between a pair of states $S_{i,j}$ (the maximal set is termed a "box" in (Durzinsky et al., 2008)). In effect, the *trans* predicate enumerates elements of the cross-product of the set of legal transitions and the maximal set of difference vectors.

Example 7. *For the water-splitting network suppose states are integer-valued vectors denoting the tokens corresponding to* H_2, O_2, H_2O, *Energy, and* Co_3O_4

as before. Suppose we observe a pair of states: $(s_0, s_1) = ([0, 0, 4, 4, 1], [2, 1, 2, 3, 1])$. *The difference vector* $d_{0,1} = s_1 - s_0$ *between this pair of states is* $[+2, +1, -2, -1, 0]$. *The algorithm in (Durzinsky et al., 2008) returns the set* $R_{0,1}^{max} = \{[+2, +1, -2, -1, 0]$, $[+1, +1, -1, -1, 0]$, $[+1, +1, -1, 0, 0]$, $[+2, 0, -2, -1, 0]$, $[0, +1, -2, -1, 0], ...\}$ *that contains all vectors such that* $d_{0,1} = \sum_i \lambda_i r_i$ *(*$r_i \in R_{0,1}^{max}$ *(some constraints are needed to ensure that this set is finite). Suppose we are allowed the set of transitions* $T = \{t_1, t_2, t_3\}$ *defined as follows:* t_1 *is a simple transition without constraints;* t_2 *requires the presence of the catalyst* Co_3O_4*; and* t_3 *requires the presence of the catalyst and environmnentally favourable conditions). Let us assume further that the guard functions for all 3 transitions are TRUE in both states* $[0, 0, 4, 4, 1], [2, 1, 2, 3, 1]$. *This denotes the set of legal transitions* $T_{0,1}$ *for the states* $s_{0,1}$. *Then acceptable transition-reaction pairs are elements of the cross-product* $T_{0,1} \times R_{0,1}^{max} = M_{0,1} = \{(t_1, [+2, +1, -2, -1, 0])$, $(t_2, [+2, +1, -2, -1, 0])$, $(t_3, [+2, +1, -2, -1, 0])$, $(t_1, [+1, +1, -1, -1, 0])$, $(t_2, [+1, +1, -1, -1, 0])$, $\ldots\}$.

Both pure and extended Petri nets can be expressed using these definitions. For pure Petri nets, pre- and post-conditions and invariants are trivially true for all transitions. Extended Petri nets follow from appropriate definitions of these predicates:

$$
\begin{array}{ll}
transition(read(P)) :- & transition(inhibit(P)) :- \\
\quad place(P). & \quad place(P). \\
\\
pre(read(P), S) :- & pre(inhibit(P), S) :- \\
\quad (P = X) \in S, & \quad (P = X) \in S, \\
\quad X > 0. & \quad X = 0. \\
\\
post(read(P), S) :- & post(inhibit(P), S) :- \\
\quad (P = X) \in S, & \quad (P = X) \in S, \\
\quad X > 0. & \quad X = 0. \\
\\
inv(read(P), S_i, S_j) :- & \\
\quad (P = X) \in S_i, & \\
\quad (P = X) \in S_j. &
\end{array}
$$

(A slightly more compact definition is possible just using *inv*). Other, more elaborate transitions will usually involve domain-knowledge. For example, here is a definition of a transition dealing with the addition of a phosphate group:

$$
\begin{array}{ll}
transition(pp(A, B, C)) :- & post(pp(P, Reactant, Product), S) :- \\
\quad place(A), & \quad (P = X) \in S, \\
\quad place(B), & \quad X > 0, \\
\quad place(C), & \quad (Product = P) \in S, \\
\quad can\_phosphorylate(A, B), & \quad P > 0. \\
\quad phosphorylated\_form(B, C). & \\
\\
pre(pp(P, Reactant, Product), S) :- & inv(pp(P, Reactant, Product), S_i, S_j) :- \\
\quad (P = X) \in S, & \quad (Reactant = R_i) \in S_i, \\
\quad X > 0, & \quad (Reactant = R_j) \in S_j, \\
\quad (Reactant = R) \in S & \quad R_i > R_j, \\
\quad R > 0. & \quad (Product = P_i) \in S_i, \\
 & \quad (Product = P_j) \in S_j, \\
 & \quad P_j > P_i.
\end{array}
$$

which requires domain-knowledge for appropriate definitions of *can_phosphorylate* and $phosphorylated\_form$. We are now able to state more clearly the basic system-identification problem we wish to address.

4 The System Identification Task

System Identification Task SI_1:

Given: (a) The observed behaviour of a system in the form of a state-sequence consisting of a set of state-pairs $S = \{(s_1, s_2), (s_2, s_3), \ldots, (s_{n-1}, s_n)\}$; and (b) Background-knowledge B in the form of definitions of guard, guarded-transition function and any other relevant predicates; and (c) The definition of $lgts$ (Definition 6)

Find: A set T of (t, r) pairs such that $B \models \exists\, T\ lgts(S, T)$

As long as B can be encoded as a logic program, the T's can be computed as answer-substitutions obtained using the usual refutation-based theorem proving machinery used by logic programming systems. The computational adequacy of the logic-programming formalism, and proofs of soundness and completeness of the proof strategy employed by logic programming systems have been established elsewhere (see Lloyd (1987)). Our interest here is to see if this can be used meaningfully in the identification of biological networks from data. We will call each T found as a solution an *explanation* for S, and consider the system-identification task as successful if the biologically correct network is amongst the explanations found. It is also desirable of course that there aren't too many spurious explanations into the bargain.[5]

We will use the following translation mechanism between transition-reaction tuples and a network representation.

4.1 SI_1 for Biological Systems

In this section we demonstrate that if the definition of B is complete,then refutation-based theorem proving can be used to identify an LGTS representations of reasonably complex biological networks. We concentrate on two kinds of networks, metabolic and signal transduction, and use as targets networks that demonstrate several features:

[5]Readers familiar with the data-mining literature will recognise these requirements as associating a higher cost with false negatives than with false positives. This is more a statement of what is desirable: the deductive engine has no knowledge of such costs.

Network	Type	Biological Feature(s)
Glycolysis	Metabolic	Long chain of reactions, in which a substrate is modified step-by-step into a product
Fructolysis	Metabolic	Substrate-product transformation achieved by sharing parts of the glycolysis pathway
MAPK cascade	Signalling	(a) Chain of phosphorylation reactions, each triggered by the presence of a protein earlier in the sequence; (b) Sequential signalling results in a cascade
Yeast pheromone	Signalling	(a) Signalling cycle involving transmembrane receptors activating a MAPK cascade; (b) Cycle shut down using a negative feedback loop; and (c) Concurrent reactions

In turn, these networks impose several requirements on a program for identification of models from data:

Network	PN Type	Modelling Requirements
Glycolysis	Pure	Models with lots of transitions and places (10 or more of each)
Fructolysis	Pure, hierarchical	Re-use of known models, or parts of known models
MAPK cascade	Extended	(a) Models with activators for signalling ("read" arcs); (b) Models with specific ordering of signals
Yeast pheromone	Extended, hierarchical	(a) Re-use of multiple models; (b) Models with feedback and (c) Deterministic behaviour from models with concurrent transitions

Problems

Glycolysis. The glycolysis pathway was the first metabolic pathway to be discovered. It is a classic case of a series of metabolic reactions in which products of one reaction form the substrates (reactants) for the next reaction. The glycolysis pathway is comprised of 10 such reactions. The reactions breakdown (metabolize) each molecule of glucose into two molecules of pyruvate. The sequence proceeds in three stages: primary (3 reactions), splitting (2 reactions) and phosphorylation (5 reactions). Altogether, 15 metabolites are involved. The pathway is one of the central metabolic pathways in living organisms: it provides an essential part of the energy required for the functioning of a cell, and is used in several metabolic processes.

Fructolysis. Fructose (fruit sugar) metabolism uses many of the same enzymes as glycolysis. Unlike glucose, which can be metabolized in many places in the human body, fructose is primarily broken down in the liver, mainly for replenishing the energy-storage molecule glycogen in that organ. Metabolism of fructose occurs in two parts. The first, consisting of 5 principal reactions, results in the breakdown of fructose into glyceraldehyde. This then enters the phosphorylation stage of glycolysis, which, with a further 5 reactions results in pyruvate. Here, we examine the re-use of the phosphorylation stage of glycolysis.

MAPK. The MAPK pathway is a protein-based sequence of events that translate a signal at the cell-surface to the nucleus. The pathway commences when a protein or a hormone binds to a receptor protein that is usually bound to the cell-membrane. This triggers a sequence of events that stops with the DNA expressing one or more genes that alter cell function. At any one step of the cascade, phosphor groups are attached to proteins. This phosphorylated form of the protein then forms a "switch" for commencing the next step. MAPK is a central signalling pathway that is used in all cell-tissues to communicate extra-cellular events to the cell nucleus. It is used to regulate a variety of responses, like hormone action, cell-cycle progression and cell-differentiation. It is also of immense clinical value, since a defect in the pathway often leads to uncontrolled growth. Proteins in the pathway are thus natural targets for anti-cancer drugs.

Yeast Pheromone. Receptors at the cell-surface receive an extra-cellular signal and transmit this signal, by use of a series of proteins and enzymes to the nucleus, in turn triggering the expression of a gene or a set of genes that cause the cell to respond to the external signal. The intra-cellular mechanisms by which yeast (*S. cerevisiae*) responds to an external mating signal (a pheromone) is one of the best understood signal transmission mechanisms in eukaryotes. Using a combination of genetics, biochemistry, and theoretical biology, the following are now known either completely, or substantially: the proteins, and enzymes involved; the order in which events take place; the protein-protein interactions, enzyme-catalyzed reactions and feedback links; and the rates at which many of the reactions occur. The resulting network is complex, but is one that nevertheless reuses a number of components that are found in signalling pathways of all eukaryotes. Specifically, many of the proteins found in the pathway have homologs in humans, and the G protein cycle and MAPK pathways are conserved in both yeast and humans.

Background knowledge

In addition to the predicates already described, for each problem, we provide the following information related to the system structure: (a) the places in the network; (b) a specification of each place as "input" (it has no incoming arcs), "output" (it has no outgoing arcs), or "free" (it could have zero or more incoming or outgoing arcs). This terminology follows (Wagler, 2011); (c) The labels allowed on arcs between nodes. These are the numbers of tokens that are transferred from place to transitions or *vice versa*, and are usually $+1$ or -1; and (d) A set of known terminal states of the system. The background knowledge specification can also include: (e) whether the data satisfy the monotone constraint in the sense described in (Durzinsky et al., 2011c). These definitions allow some of the computations (especially obtaining the "box" or the maximal-set of difference vectors) to be performed efficiently.

In addition, the following problem-specific background knowledge is provided:

Glycolysis. The only transitions allowed are un-named ("anonymous") transitions, with guards that are trivially satisfied. That is, pre and post-

Problem	Answers Obtained	Correct Structure?
Glycolysis	1	Yes
Fructolysis	1	Yes
MAPK	1	Yes
Pheromone	1	Yes

Figure 4.1: System identification using a refutation-based theorem prover. For each problem, the correct network structure is obtained as a logical consequence of the definitions in B.

conditions and invariants are all $TRUE$. This ensures that answer-substitions will only contain anonymous transitions (pure Petri nets).

Fructolysis. In addition to anonymous transitions, we now allow named transitions corresponding to the three different stages of glycolysis (primary, splitting, and phosphorylation). This requires providing definitions for pre- and post-conditions and invariants for these transitions. For example, the pre-condition for the primary stage of glycolysis is the following: there should be at least 2 molecules of ATP and at least 1 molecule of glucose. The post-condition is that the value of fructose biphosphate-1. This will allow answer-substitutions that may contain a mixture of named and un-named transitions (thus allowing the re-use of networks constructed earlier).

MAPK. In addition to anonymous transitions with trivial guards the system is now allowed to use transitions that can contain any of the(non-output) places as a potential activator. This allows the construction of extended Petri nets (that is, transitions with "read" arcs).

Yeast Pheromone. The pheromone response pathway uses some standard signalling building blocks: a G-protein cycle for transmission from the receptor; a MAPK cascade; G-protein formation from sub-units; and pathways for the formation of scaffolds that hold proteins in place. We provide named transitions in the background knowledge for the re-use of some known sub-nets, along with transitions for allowing read arcs, and anonymous transitions. This allows transitions to be unconditional (as in pure Petri nets), activated (as in extended Petri nets) and re-use known sub-nets.

4.2 Results

System behaviour provided as input and the transition-reaction sequences obtained as solutions for the system identification SI_1 for each of the problems are in Appendix A, and summarised in Fig. 4.1 below. It is evident that it is possible to obtain the correct structure of systems using a combination of background knowledge and the finite-state machine described earlier.

That the correct structure is identified exactly suggests also that both the background knowledge and the data observed are complete and correct. It is

nevertheless encouraging to find that under those conditions, it is possible to view system-identification as a form of logical consequence-finding. There are some interesting aspects of using background knowledge, which may not be immediately apparent from these result

Scale. It is impractical to consider constructing very large-scale networks without some method for decomposing the task into one of constructing sub-nets, and building larger structures that re-use these smaller ones. This form of hierarchical network construction requires a mechanism to re-use all or portions of networks that are already known, or have been identified. Guarded transitions in the background knowledge provide a natural way to represent such sub-nets. For example, consider the problem of modelling fructolysis which used guarded transitions that uses some portions of the glycolysis metabolic pathway, which is traditionally taken to occur in 3 stages (primary, splitting, and phosphorylation). One possible representation of these, using the predicates of the kind we have just seen is as follows:

```
transition(anonymous).
transition(glyc_primary).
transition(glyc_splitting).
transition(glyc_phos).

pre(glyc_primary,PreState):-
    val(atp,PreState,Atp),
    Atp >= 2,
    val(glu,PreState,Glucose),
    Glucose > 0.

post(glyc_primary,PostState):-
    val(f16bp,PostState,F16bp),
    F16bp > 0.

inv(glyc_primary,PreState,PostState).

...
(and so on)
```

Efficiency. Clearly, many different answer-substitutions T may be obtained for a sequence S of observed data. Each of these is a correct solution to the logical consequence relation $B \models \exists\, T\ lgts(S,T)$. One way to constrain the solutions is to enrich B. This can be done in a domain-independent, or a domain-dependent manner. As an example for the former, (Srinivasan and King, 2008) describe "feasible" chemical reactions as ones in which reactants and products are compatible (the usual mass-balance criterion) and the reaction does not break 3 or more bonds, which requires far more energy than is usually present. For the chemicals in glycolysis, here is a tabulation of the only reactions that are possible, once such a restriction is imposed:

Feasible reactions (break fewer than 3 bonds):

$G6P \rightarrow F6P$	$FBP \rightarrow DHAP + G3P$	$G3P + NAD \rightarrow 1,3BPG + NADH$
$3PG \rightarrow 2PG$	$1,3BPG + Glu \rightarrow 3PG + G6P$	$F6P + ATP \rightarrow FBP + ADP$
$2PG \rightarrow PEP$	$1,3BPG + ADP \rightarrow 3PG + ATP$	$G6P + Pv \rightarrow Glu + PEP$
$DHAP \rightarrow G3P$	$3PG + FBP \rightarrow 1,3BPG + F6P$	$FBP + Glu \rightarrow F6P + G6P$
$1,3BPG \rightarrow 3PG$	$Glu + ATP \rightarrow G6P + ADP$	$FBP + Glu \rightarrow F6P + PEP$
$FBP \rightarrow F6P$	$PEP + ADP \rightarrow Pv + ATP$	$FBP + Pv \rightarrow F6P + PEP$

These constraints can be incorporated directly into B by only allowing transitions that involve feasible reactions, thus greatly reducing the space of possible answers:

```
transition(reaction(LHS,RHS)):-
    feasible_reaction(LHS,RHS).

% reactions that break less than 3 bonds
feasible_reaction(['3pg'],['2pg]).
feasible_reaction(['2pg'],[pep]).
feasible_reaction([adp,pep],[atp,pv]).
    ...
    ...
(and so on)
```

4.3 Identification from Multiple System Behaviours

So far a single experiment has provided all the data needed to identify a system. Since system behaviour is simply taken as sets of (consecutive) state-pairs, there is no difficulty in principle in using the same implementation to obtain a solution for observations from multiple experiments. We demonstrate this by taking another look at the MAPK pathway.

Consider the MAPK cascade, in which data are available across three different experiments:

Data S_1:

Place	States	
	0	1
map4k	1	1
map3k	1	0
map3kp	0	1
map2k	0	
map2kp	0	0
map2kpp	0	0
mapk	0	0
mapkp	0	0
mapkpp	0	0

Data S_2:

Place	States		
	0	1	2
map4k	0	0	0
map3k	0	0	0
map3kp	1	1	1
map2k	1	0	0
map2kp	0	1	0
map2kpp	0	0	1
mapk	0	0	0
mapkp	0	0	0
mapkpp	0	0	0

Data S_3:

Place	States		
	0	1	2
map4k	0	0	0
map3k	0	0	0
map3kp	0	0	0
map2k	0	0	0
map2kp	0	0	0
map2kpp	1	1	1
mapk	1	0	0
mapkp	0	1	0
mapkpp	0	0	1

For simplicity, we have shown all experiments having data for all places (with 0-values for places that are not part of the experiment). We note that initial and final states of the complete system may not be present in all of the data. For example, S_1 is missing the final state of the MAPK cascade, S_2 is missing both initial and final states, and S_3 is missing the initial state. Assuming the same background as was used with the single-experiment dataset results in the following answer-substitutions (the entries of the reaction-vectors are to be read in order of the places shown in the data above):

T_1 $[(read(map4k), [0, -1, +1, 0, 0, 0, 0, 0, 0])]$

T_2 $[(read(map3kp), [0, 0, 0, -1, +1, 0, 0, 0, 0]), (read(map3kp), [0, 0, 0, 0, -1, +1, 0, 0, 0])]$

T_3 $[read(map2kpp), [0, 0, 0, 0, 0, -1, +1, 0, 0]), (read(map2kpp), [0, 0, 0, 0, 0, 0, 0, -1, +1])]$

The graphs associated with each transition-reaction are:

G_1 $(\{map3k, map3kp\}, \{t_1\}, \{e(map3k, t1), e(t1, map3kp)\})$

G_2 $(\{map2k, map2kp, map2kpp\}, \{t_2, t_3\}, \{e(map2k, t_2), e(t_2, map2kp), e(map2kp, t_3), e(t_3, map2kpp)\})$

G_3 $(\{mapk, mapkp, mapkpp\}, \{t_4, t_5\}, \{e(mapk, t_4), e(t_4, mapkp), e(mapkp, t_5), e(t_5, mapkpp)\})$

where:
$t_1 = hash((read(map4k), [0, -1, +1, 0, 0, 0, 0, 0, 0]))$,
$t_2 = hash((read(map3kp), [0, 0, 0, -1, +1, 0, 0, 0, 0]))$,
$t_3 = hash((read(map3kp), [0, 0, 0, 0, -1, +1, 0, 0, 0]))$
and $e(x, y)$ denotes a directed edge from vertex x to y. The graph of the conjunction of the T_i is the union of the G_i.

5 Identification with Incomplete System Information

First we discuss the case of missing data, then the situation of transitions missing from background knowledge.

5.1 Incomplete Data: Abduction

In the networks identified so far, we have assumed that data are available for all the relevant places in a state. The specification, in background knowledge of transitions representing sub-networks allows one way around this, since data on places in the sub-network are not required. But this may not always be possible. What is to be done when data on places are missing, something can still be done using a combination of guarded transitions and the logical notion of *abducibles*. Specifically, we can determine if there is a sequence of transitions that can be used to build a complete model, if a set of places—a subset of the set of "abducible" places—were assumed to have a sufficient number of tokens. In keeping with this kind of analysis, the set of abducibles (that is, the places whose values can be assumed) is pre-specified by the user (see Fig. 5.1) as part of the background knowledge. The introduction of abducibles can result in many possible hypothesized networks. In turn, these may suggest additional experiments that may need to be conducted to focus on the best model.

In implementation terms, abduction is enabled by a simple alteration to the definition of the pre-condition of a transition from:

```
pre(reaction(LHS,RHS),PreState):-
    present(LHS,PreState).

present([X|Rest],State):-
```

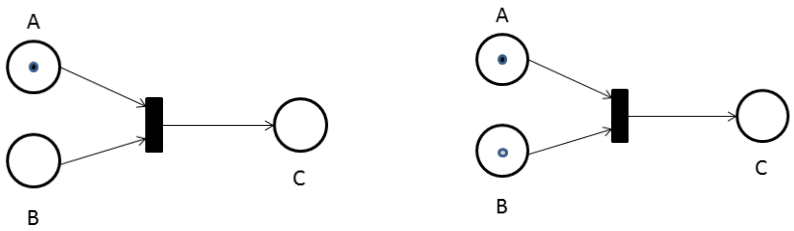


Figure 5.1: Enabling transitions when data are missing. On the left, the (guarded) transition is not enabled, since there is no token at B. If B is specified as an "abducible" place, then at least one token is assumed to be present (shown in a lighter shade on the right), which enables the transition (that is, satisfies the pre-condition of the transition). Experimental data would be needed to confirm (or refute) the reality of such a transition.

```
    tokens(X,State,N),
    N > 0,
    present(Rest,State).
...
(and so on)
```

to a definition like:

```
pre(reaction(LHS,RHS),PreState):-
    present_or_abducible(LHS,PreState).

present_or_abducible([X|Rest],State):-
    present([X],State), !,
    present_or_abducible(Rest,State).
present_or_abducible([X|Rest],State):-
    abducible(X),
    present_or_abducible(Rest,State).
...
(and so on)
```

(Similarly, the post-condition is either each output place value is non-zero, or that it is abducible.)

As a specific case, consider the fructolysis problem, this time leaving out any background knowledge about the different stages of the glycolysis pathway and no data are available on metabolites involved in glycolysis. The data available for system-identification are thus:

Place	States					
	0	1	2	3	4	5
fruc	1	0	0	0	0	0
atp	2	1	1	0	2	4
adp	4	5	5	6	4	2
f1p	0	1	0	0	0	0
ga	0	0	1	0	0	0
dhap	0	0	1	0	0	0
g3p	0	0	0	2	1	0
nad	2	2	2	2	1	0
nadh	0	0	0	0	1	2
pv	0	0	0	0	1	2
glu	?	?	?	?	?	?
f6p	?	?	?	?	?	?
f16bp	?	?	?	?	?	?
13bpg	?	?	?	?	?	?
3pg	?	?	?	?	?	?
2pg	?	?	?	?	?	?
pep	?	?	?	?	?	?

Even with background knowledge defining feasible chemical reactions in glycolysis, it is not possible to obtain an answer-substitition for the state-sequence shown above. However, allowing places in glycolysis to be abducible, and relaxing pre- and post-conditions in the manner just described, we are able to hypothesize the missing portion of the glycolysis sub-net corresponding to the phosphorylation stage (see Fig. A.5 in Appendix A).

Each of the features we have described so far (scaling-up, constraining answers, hypothesising values in a state) are all made feasible by definitions in the background knowledge B. In the next section we investigate extensions to the specification for system-identification that allow us address certain kinds of deficiencies in the background knowledge and in the data.

5.2 Incomplete Background Knowledge: Induction

We have assumed so far that the background knowledge B is sufficient, given an observed sequence of states S to identify all the T s.t. $B \models \exists\, T\ lgts(S,T)$. Suppose we find that no such T is possible. Ignoring, for the moment, any incompleteness in the proof procedure used, we consider adding definitions H automatically to B, i.e., learning new transitions. The revised system identification task we consider is thus:

System Identification Task SI_2:

Given: (a) The observed behaviour of a system in the form of a set of of consecutive state-pairs $S = \{(s_1, s_2), (s_2, s_3), \ldots, (s_{n-1}, s_n)\}$; and (b) Background-knowledge B in the form of definitions of guard, guarded-transition function and any other relevant predicates; and (c) The definition of $lgts$ (Definition 6); and (d) $B \not\models \exists\, T\ lgts(S,T)$

Find: A set of clauses H and a set T of (t,r) pairs such that $B \wedge H \models \exists\, T\ lgts(S,T)$

A natural way to address the problem of automatically augmenting the definitions of a logic program B is to use the approach of Inductive Logic Programming (ILP: (Raedt, 2008)). Given a logic program B and a non-empty set E^+

("positive" examples) and a set (possibly empty) E^- ("negative" examples), and ILP engine attempts to find a set of clauses H s.t. $B \wedge H \models E^+$ and $B \wedge H \wedge E^- \not\models \Box$. That is, given B, H "explains" E^+; and is consistent with E^-. We refer the reader to (Muggleton, 1995) for details of how acceptable H's are found to satisfy this requirement.

We consider here how an ILP engine could be used to find an H as required by SI_2. Suppose the observed set of state-pairs $S = \{(s_1, s_2), \ldots, (s_{n-1}), s_n)\}$. Recall that if $B \models \exists\ T\ lgts(S, T)$ then for every $(s_i, s_{i+1}) \in S$) there exists a $tr_i \subseteq T$ such that $gts(s_i, s_{i+1}, tr_i)$ is $TRUE$. So, if $B \not\models \exists\ T\ lgts(S, T)$, then there is at least one (s_i, s_{i+1}) for which this does not hold. It is not hard to see that a sufficient condition for this is that for some pair of states $s_{a,b}$ there is no single-step transition-reaction possible given the background B.[6]

Assuming that it is always possible to obtain a reaction vector r between a pair of states, this means the background knowledge is missing the appropriate definition of the transition t, along with its guard. Since the invariant of the guard has all the information available to the pre- and post-conditions, we focus on just the definition of invariants. That is, the problem of constructing and H in SI_2 translates to learning transition invariants for new transitions $new(t)$ between pairs of states $s_{a,b}$ for which no existing transition is applicable. These will be used in B as follows:

```
% allowed to use anonymous transitions
transition(anonymous)
% allowed to learn new transitions if anonymous ones do not work
transition(new(_)).

pre(anonymous,PreState).
post(anonymous,PostState).
inv(anonymous,PreState,PostState).

pre(new(_),PreState).
post(new(_),PostState).

% re-use a transition invariant that has been learnt
inv(new(T),PreState,PostState):-
    transition_invariant(new(T),PreState,PostState), !.
% otherwise learn a new transition invariant
inv(new(T),PreState,PostState):-
    learn_transition(new(T),PreState,PostState).

learn_transition(new(T),PreState,PostState):-
    new_symbol(T),
    create_positive_examples(PreState,PostState),
    create_negative_examples(PreState,PostState),
    <call ILP engine>
    <add new definitions for transition_invariant>
...
(and so on)
```

We still have to clarify the step creating positive and negative examples. The ILP engine is required to learn a definition for $transition\_invariant/3$. For a transition $new(t)$ between states $s_{a,b}$ the ILP engine is given the single positive example $transition\_invariant(new(t), s_a, s_b)$. Negative examples are less obvious. Suppose $r_{a,b}$ is any element of the maximal set $R^{max}_{a,b}$ consistent with $d_{a,b} = s_b - s_a$. That is $r_{a,b}$ denotes a feasible reaction vector given $s_{a,b}$. They consist of statements of the form $\neg transition\_invariant(new(t), s_t, s_x)$ where s_t is any terminal state of the system and s_x is any legal state of the system s.t. $r_{a,b} = s_x - s_t$. That is, negative examples are all legal states is

[6] $s_{a,b}$ need not be the same as $s_{i,i+1}$, but could be some intermediate states between them.

reachable from a terminal state with a feasible reaction vector. The ILP engine is thus forced to search for invariant definitions that do not hold for such pairs (ensuring automatically that the transition will not violate the terminal-state specification of the system).

We demonstrate this approach with the MAPK cascade, using the same data as before:

Place	States					
	0	1	2	3	4	5
map4k	1	1	1	1	1	1
map3k	1	0	0	0	0	0
map3kp	0	1	1	1	1	1
map2k	1	1	0	0	0	0
map2kp	0	0	1	0	0	0
map2kpp	0	0	0	1	1	1
mapk	1	1	1	1	0	0
mapkp	0	0	0	0	1	0
mapkpp	0	0	0	0	0	1

With only un-named (anonymous) transitions allowed in B, it is not possible to obtain a transition-reaction sequence for this data (all anonymous transitions fire on one or more terminal states). That is, $B \not\models \exists\ T\ lgts(S,T)$. We now consider using an ILP engine to construct an H s.t. $B \wedge H \models \exists\ T\ lgts(S,T)$. [7] We allow the ILP engine to use the following predicates when constructing H:

```
read(Place,Pre,Pos):-
    place(Place),
    val(Place,Pre,X),
    val(Place,Post,X),
    X > 0.

inhibit(Place,Pre,Post):-
    place(Place),
    val(Place,Pre,X),
    val(Place,Post,X),
    X = 0.
```

This allows the construction of new transitions that include combinations of "read" and "inhibit" arcs from more than one place (these combinations can occur in a single transition). The ILP engine constructs the following definition H:

```
transition_invariant(new(1),Pre,Post):-
    read(map4k,Pre,Post).
transition_invariant(new(2),Pre,Post):-
    read(map3kp,Pre,Post).
transition_invariant(new(3),Pre,Post):-
    read(map2kpp,Pre,Post).
```

The definition is constructed when the theorem-prover attempts to execute the definition of the invariant predicate for a new transition. In conjunction with the background knowledge B and sequence of states S above, we are now able to get the following answer T to the relation $B \wedge H \models \exists\ T\ lgts(S,T)$:

$$\begin{aligned} T \quad = \quad & [(new(1),[0,-1,+1,0,0,0,0,0,0]), \\ & (new(2),[0,0,0,-1,+1,0,0,0,0]),(new(2),[0,0,0,0,-1,+1,0,0,0]), \\ & (new(3),[0,0,0,0,0,-1,+1,0,0]),(new(3),[0,0,0,0,0,0,0,-1,+1])] \end{aligned}$$

[7] We use the Aleph ILP system (Srinivasan, 2007).

which is the correct definition of the MAPK cascade.

6 Related Work

Knowledge discovery in natural science is increasingly being seen as a process of combinatorial optimisation, in which computational approaches are of central importance (Kell, 2012). On one hand this may be addressed by using "big data" to reduce the space of possible solutions (Hey et al., 2009). On the other hand, data alone may be insufficient, but we may have access to considerable relevant domain knowledge. The latter is the situation addressed in this work.

Since the introduction of Petri nets (Petri, 1962) they have been used extensively in many areas, for example, in modelling concurrency (Nielsen et al., 1981). As noted above, Petri nets have also been quite widely used for modelling in systems biology (Koch et al., 2011). Typically though such models are hand-crafted from biological knowledge.

The problem of identification, or reverse engineering, in systems biology has been the subject of many studies; a recent review is in (Villaverde and Banga, 2014). However, the work of Durzinsky et al. (2008) appears to be the first published method for the reconstruction of Petri net models from time series data. In a number of papers this group developed a mathematical framework and algorithms for the problem they term automatic network reconstruction, i.e., to generate the set of all pure Petri nets from experimental discrete time-series data. Their method was subsequently broadened to handle extended Petri nets, with read and inhibitory arcs (Durzinsky et al., 2011b). For pure Petri nets their approach is a combinatorial method to generate all sets of reaction vectors such that each set is consistent with the data, by determining that it is conic integer combination of the difference vectors.

Answer Set Programming (ASP – see Baral (2003) for an overview) is an approach to logic programming that has a number of useful features for systems biology modelling. These include: true negation (as well as default negation or negation as failure (Gelfond, 2008)); efficient solvers; and a number of declarative built-in language constructs for choice and optimization. Durzinsky et al. (2011a) reformulated their Petri net reconstruction algorithm using ASP. They note some advantages of ASP for this work: that it allows a declarative reformulation of their previous implementation; the possibility of addition of declarative biological knowledge as constraints; and, since the ASP system used is based on a constraint solvers, the approach is as efficient as a previous special-purpose implementation. Essentially, the method searches for models that conform to a graph of system states, termed the experiment graph, where models are constrained by clauses specifying the network reconstruction algorithm. A version was also developed that allows the addition of a fixed number of molecular species to enable recovery of a network if no valid experimental graph can be constructed from the original data.

Inoue (2011) formulated the computation of the next state from the current state in a dynamic system in terms of the immediate consequence operator T_P of logic program P (Lloyd, 1987). Based on this, and using the simpler formalism of Boolean networks, Videla et al. (2014) give a method to identify models using ASP. This method requires as input a signed directed graph of genes showing causal dependencies of molecular species, together with a discretised dataset.

The identified system is restricted to only modelling the so-called immediate-early response, i.e., the Boolean functions determined for a directed hypergraph that model the resulting phosphorylation state of proteins, given a set of perturbations of the system. System identification is then an optimization problem, namely to minimize the model error with respect to the dataset. Importantly, the modelling approach is restricted to acyclic dependencies of genes represented by propositional formulas, meaning that feedback and hence complex dynamic behaviours are not captured.

Clearly, these are purely (constrained) deductive approaches. This accords with our present approach, which, as we said in our introduction, makes sense when there is relatively little data, and a reasonably large amount of background knowledge. An early example of this was a logical formulation of pathways and reactions for metabolism (Reiser et al., 2001) that enabled the deductive computation of the reachability of a system state; this became the foundation of the Robot Scientist project (King et al., 2004).

On the other hand, it is sometimes necessary to deal with missing data, or background knowledge; as in our approach above, this has been addressed in the logical setting using abductive or inductive algorithms (Tamaddoni-Nezhad et al., 2004), typically using ILP. Applications of ILP on learning in systems biology have ranged from addressing the incompleteness of rich background knowledge, such as (King et al., 2004) for network completion, to those focused on classification, such as (Fröhler and Kramer, 2008) for predicting phenotypic state. An important topic in ILP has been learning models of dynamical systems. Much of this work has essentially been based on some form of temporal logic, often motivated by robotics (e.g., (Moyle, 2002)). This has typically required an explicit handling of time in a physical sense, rather than within an event-based formalism. However, systems biology data usually has insufficient temporal resolution for this to be applicable.

As mentioned above, Inoue (2011) formulated Boolean networks as (normal) logic programs, and Inoue et al. (Inoue et al., 2014) showed how this semantics enabled a method of learning from state transition pairs, where each state is an interpretation. Interestingly, this in some sense is the inverse of our method; in both methods simulation is by logical consequence (computation of the successor states) but in the method of Inoue et al. (2014) identification is by a form of generalisation. More specifically, pairs of a state and its successor form the predicate for a target gene, and the learning proceeds by constructing positive and negative examples from which rules are generalised to define how the successor (Boolean) state of the target gene can be computed, given the current state of all genes that may regulate it. In contrast, in our framework, by treating state transitions as first-class objects we are able to find logical guards for transitions by deduction. Of course, as outlined above, our method can also abduce or induce missing state values or transitions if required.

We note that, contrary to what is claimed in (Inoue et al., 2014), in our method feedback loops *are* modelled (for example, in the yeast pheromone pathway reconstruction experiment – see Appendix A). This misconception may be due to the problem of learning the recursive definitions required for feedback relations in logic programs defined on *states*, as in the framework of (Inoue et al., 2014). However, this problem does not arise with logical definitions where *transitions* are treated as a first-class object, as in Petri nets and LGTSs. In such representations feedback is simply a repeated application of transitions

(see Fig. 2.2(c)).

In this paper our formalism (LGTS) is inspired in part by Guarded Horn Clauses (Ueda, 1988). However, a similarly named approach — Guarded Transition Systems (GTS) — has appeared in a recent paper by Rauzy (2008). In this work GTSs are proposed in order to overcome a number of limitations of modelling formalisms for reliability analyses in systems with loops, such as electrical networks. This kind of loop occurs when the state of a component A depends on the state of another component B and *vice versa*. Analysis of such systems requires, in general, the propagation of states through the model, given a set of initial states.

A GTS is defined similarly to a Petri net, except (1) for additional constraints on each transition, and (2) a global set of assertions. The relation to LGTS lies principally in the formalisation of transitions. In GTSs, a transition is defined as a triple $\langle G, e, P \rangle$, where G is a guard, e is an event (transition) and P a post-condition. Unlike in LGTS, the guard G is not completely specified in (Rauzy, 2008), but is stated to include arithmetic expressions or Boolean, i.e., propositional, expressions over variables. Variables are allowed to be of any type, but this is not given a detailed specification. The post-condition P is an *instruction*, meaning that the values of system variables can be updated following the transition. The key contribution of (Rauzy, 2008) is to add to the post-condition an *assertion* with a fixpoint semantics. The assertion is composed of a set of assignment "rules" for updating variables. Use of a fixpoint semantics enables repeated application of these rules until the system arrives in a terminal state. To avoid infinitely looping behaviour a finite bound is placed on the number of steps allowed to achieve this fixpoint, essentially turning the application of the assertion into an iterative process. This mechanism thus enables propagation of states through the model to allow verification of components in circuits with loops. Note that there is only a single assertion for each GTS; it is a global property of the model, applied to all transitions in the same way., rather than being a local property of a particular transition like guards and post-conditions.

It should be noted that, since extended Petri nets are Turing-equivalent, the GTS formalism does not increase expressive power. For example, the effect of the fixpoint could be achieved using an extended Petri net, but this could become very tedious to specify. Rather it enables a useful modelling framework, particularly where the reliability of components in a looping circuit or similar iterative system must be validated. The use of a fixpoint assertion method in the modelling language is something that may be investigated as a potential addition to our approach as part of future work.

It is notable that in most work on systems biology (Ideker et al., 2001) the concept of a "system" is not formally defined separately from the particular representation used (differential equations, Boolean networks, etc.). In this work we have been partly inspired by the logical approach of Takahashi and Takahara (1995), where the system modelling task is treated formally by fixing a representation language and axioms defining the possible structures.

7 Concluding Remarks

Model identification in systems biology is a difficult problem. It is typically thought that the more complex the model class, the more difficult the iden-

tification problem, and consequently considerable effort has been devoted to methods that build quantitative models (like ODEs), or qualitative network models using formalisms that capture some domain-features. Some of the difficulties that arise are these: (a) the numerical data available are often of variable quality, making it difficult to identify good quantitative models; (b) biology is a knowledge-rich domain, and system identification needs to incorporate as much of this as possible; (c) there is no clearly specified way of scaling-up the identification procedure to identify complex systems beyond the level of "decompose the system into sub-parts"; and (d) the models may not be easily comprehensible to biologists.

The representation of Logical Guarded Transition Systems we have proposed here allows a qualitative network representation of system structure that has pure and extended Petri nets as special cases. It has also allowed us to formulate the system-identification task in terms of the logical entailment relation. This opens the door to the use of logic-programming as the computational means of finding system models consistent with domain-specific constraints and requirements. We have also shown how the setting extends naturally through the use of Abductive Logic Programming (ALP) and Inductive Logic Programming (ILP).

In terms of the specific application to the identification of biological networks, we believe LGTS retain much of the attractiveness of Petri nets: they are able to model system dynamics, non-determinism and concurrency; and their graphical representation is appealing to biologists. The specific networks we have identified this paper show that:

1. Guarded transitions allow the straightforward inclusion of background knowledge, primarily in the form of transitions that are allowed, and in the definition of the guards.

2. We are able to identify reasonably large (over 10 transitions and 10 places) pure Petri nets from data by identifying transitions with trivial guards and their associated reaction vectors.

3. We are able to construct larger networks with more places straightforwardly by using guarded transitions that abstract over sub-nets.

4. We are able to identify extended Petri nets from data by identifying activators in guarded transitions along with their associated reaction vectors.

5. We are able to use techniques from ALP to construct models when data are missing.

6. We are able to augment existing background knowledge by learning new transitions using an ILP engine.

These results are sufficiently encouraging to believe that the area we have introduced is a promising direction to follow. There are many ways in which this work could be extended. More can always be done with attempting to learn other complex networks. We would also like to demonstrate an end-to-end system identification approach starting with experimental data, building LGTS models, converting them to quantitative models and performing simulations. Of these, the link between experimental data and the discretised data needed

for learning Petri net models will be particularly important. The representation used here is a step into the much larger area of process algebras. It remains to be seen whether the start made in this paper naturally leads on to the hitherto unxplored territory of the use of abduction, induction, and probabilities—used here in their philosophical sense—in pi-calculus representations of biological systems.

Acknowledgements

A.S. is a Ramanujan Fellow of the Government of India; a Visiting Professor at the Department of Computer Science, Oxford University; and a Visiting Professorial Fellow at the School of CSE, University of New South Wales, Sydney. M.B. acknowledges the support of the Australia-India Strategic Research Fund. The authors would like to thank Ross King for providing some of the domain-specific constraints used in this paper, and Mark Temple for his advice on the yeast pheromone pathway.

References

Baral, C. (2003). *Knowledge Representation, Reasoning and Declarative Problem Solving*. Cambridge University Press.

Bolouri, H. and Davidson, E. (2002). Modeling transcriptional regulatory networks. *BioEssays*, 24:1118–1129.

David, R. and Alla, H. (2010). *Discrete, Continuous, and Hybrid Petri Nets*. Springer, Berlin, Second edition.

Durzinsky, M., Marwan, W., Ostrowski, M., Schaub, T., and Wagler, A. (2011a). Automatic Network Reconstruction using ASP. *Theory and Practice of Logic Programming*, 11(4-5):749–766.

Durzinsky, M., Wagler, A., and Marwan, W. (2011b). Reconstruction of extended Petri nets from time series data and its application to signal transduction and to gene regulatory networks. *BMC Systems Biology*, 5:113.

Durzinsky, M., Wagler, A., and Weismantel, R. (2011c). An algorithmic framework for network reconstruction. *Theoretical Computer Science*, 412:2800–2815.

Durzinsky, M., Wagler, A., Weismantel, R., and Marwan, W. (2008). Automatic reconstruction of molecular and genetic networks from discrete time series data. *BioSystems*, 93:181–190.

Fröhler, S. and Kramer, S. (2008). Inductive logic programming for gene regulation prediction. *Machine Learning*, 70:225–240.

Gelfond, M. (2008). Answer Sets. In van Harmelen, F., Lifschitz, V., and Porter, B., editors, *Handbook of Knowledge Representation*, pages 285–316. Elsevier, Amsterdam.

Hey, T., Tansley, S., and Tolle, K. (2009). *The Fourth Paradigm: Data-Intensive Scientific Discovery*. Microsoft Reasearch.

Ideker, T., Galitski, T., and Hood, L. (2001). A new approach to decoding life: systems biology. *Ann. Review of Genomics and Human Genetics*, 2:343–372.

Inoue, K. (2011). Logic Programming for Boolean Networks. In *IJCAI 2011: Proc. 22nd Intl. Joint Conference on Artificial Intelligence*, pages 924–930.

Inoue, K., Ribeiro, T., and Sakama, C. (2014). Learning from interpretation transition. *Machine Learning*, 94(1):51–79.

Junker, B. H. and Schreiber, F. (2008). *Analysis of Biological Networks*. Wiley, NJ.

Kell, D. (2012). Scientific discovery as a combinatorial optimisation problem: How best to navigate the landscape of possible experiments? *Bioessays*, 34:236–244.

King, R., Whelan, K., Jones, F., Reiser, P., Bryant, C., Muggleton, S., Kell, D., and Oliver, S. (2004). Functional genomic hypothesis generation and experimentation by a robot scientist. *Nature*, 427:247–252.

Koch, I., Reisig, W., and Schreiber, F., editors (2011). *Modeling in Systems Biology: the Petri Net Approach*. Springer, Berlin.

Lloyd, J. W. (1987). *Logic Programming, 2nd Edition*. Springer-Verlag, Berlin.

Moyle, S. (2002). Using theory completion to learn robot navigation control programs. In *ILP 2002: Proc. Intl. Conference on Inductive Logic Programming*, volume 2583 of *LNAI*, pages 182–197, Berlin. Springer.

Muggleton, S. (1995). Inverse Entailment and Progol. *New Generation Computing*, 13:245–286.

Nielsen, M., Plotkin, G., and Winskel, G. (1981). Petri Nets, Event Structures and Domains, Part 1. *Theoretical Computer Science*, 13:85–108.

Palsson, B. (2006). *Systems Biology: Properties of Reconstructed Networks*. Cambridge University Press, Cambridge.

Perkins, T., Jaeger, J., Reinetz, J., and Glass, L. (2006). Reverse Engineering the Gap Gene Network of *Drosophila melanogaster*. *PLoS Computational Biology*, 2(5):e51.

Peterson, J. (1981). *Petri Net Theory and the Modeling of Systems*. Prentice-Hall, Englewood Cliffs, NJ.

Petri, C. (1962). *Kommunikation mit Automaten*. PhD thesis, Technical University Darmstadt, Darmstadt.

Raedt, L. D. (2008). *Logical and Relational Learning*. Springer, Berlin.

Rauzy, A. (2008). Guarded Transition Systems: a new States/Events Formalism for Reliability Studies. *Journal of Risk and Reliability*, 222(4):495–505.

Reiser, P., King, R., Kell, D., Muggleton, S., Bryant, C., and Oliver, S. (2001). Developing a Logical Model of Yeast Metabolism. *Electronic Transactions in Artificial Intelligence*, 5:223–244.

Srinivasan, A. (2007). *The Aleph manual*. University of Oxford, Oxford.

Srinivasan, A. and Bain, M. (2012). Knowledge-Guided Identification of Petri Net Models of Large Biological Systems. In Muggleton, S., Tamaddoni-Nezhad, A., and Lisi, F., editors, *Proc. 21st Intl. Conference on Inductive Logic Programming (ILP 2011; Revised Selected Papers)*, volume 7207 of *Lecture Notes in Computer Science*, pages 317–331, Berlin. Springer.

Srinivasan, A. and King, R. D. (2008). Incremental Identification of Qualitative Models of Biological Systems using Inductive Logic Programming. *Journal of Machine Learning Research*, 9:1475–1533.

Takahashi, S. and Takahara, Y. (1995). *Logical Approach to Systems Theory*. Springer, Berlin.

Tamaddoni-Nezhad, A., Kakas, A., Muggleton, S., and Pazos, F. (2004). Modelling inhibition in metabolic pathways through abduction and induction. In *ILP 2004: Proc. 14th Intl. Conference on Inductive Logic Programming*, Berlin. Springer.

Ueda, K. (1988). Guarded Horn Clauses: A Parallel Logic Programming Language with the Concept of a Guard. In Nivat, M. and Fuchi, K., editors, *Programming of Future Generation Computers*, pages 441–456. North-Holland.

Videla, S., Guziolowski, C., Eduati, F., Thiele, S., Gebser, M., Nicolas, J., Saez-Rodriguez, J. J., Schaub, T., and Siegel, A. (2014). Learning Boolean logic models of signaling networks with ASP. *Theoretical Computer Science*, (in press).

Villaverde, A. and Banga, J. (2014). Reverse engineering and identification in systems biology: strategies, perspectives and challenges. *J. R. Soc. Interface*, 11(20130505).

Wagler, A. (2011). Prediction of Network Structure. In Koch, I., Reisig, W., and Schreiber, F., editors, *Modeling in Systems Biology: the Petri Net Approach*, pages 307–336. Springer, Berlin.

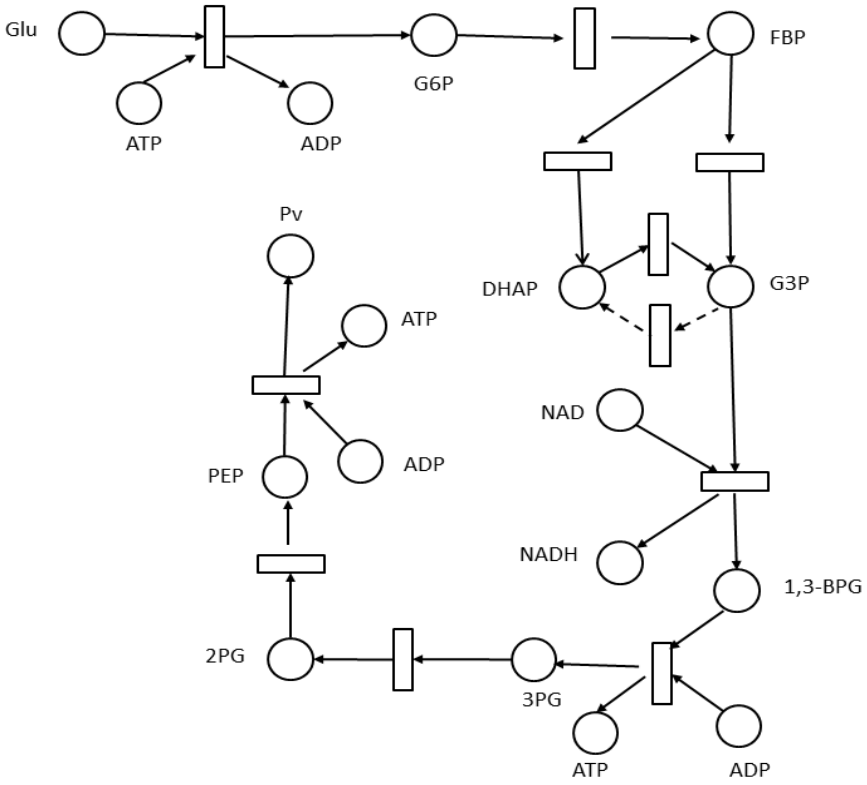


Figure A.1: Petri net model of the glycolysis pathway identified. The conversion of DHAP to G3P is taken to be in one-direction only (the reverse is shown by a dashed line, and not identified).

A Data Provided and Networks Identified

A.1 Pure Petri Nets (Glycolysis)

Place	States															
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
glu	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
atp	2	1	1	0	0	0	1	1	1	2	2	2	3	3	3	4
adp	4	5	5	6	6	6	5	5	5	4	4	4	3	3	3	2
g6p	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
f6p	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
f16bp	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
dhap	0	0	0	0	1	1	1	1	1	1	0	0	0	0	0	0
g3p	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0
nad	2	2	2	2	2	1	1	1	1	1	1	0	0	0	0	0
13bpg	0	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0
nadh	0	0	0	0	0	1	1	1	1	1	1	2	2	2	2	2
3pg	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0	0
2pg	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0
pep	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0
pv	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	2

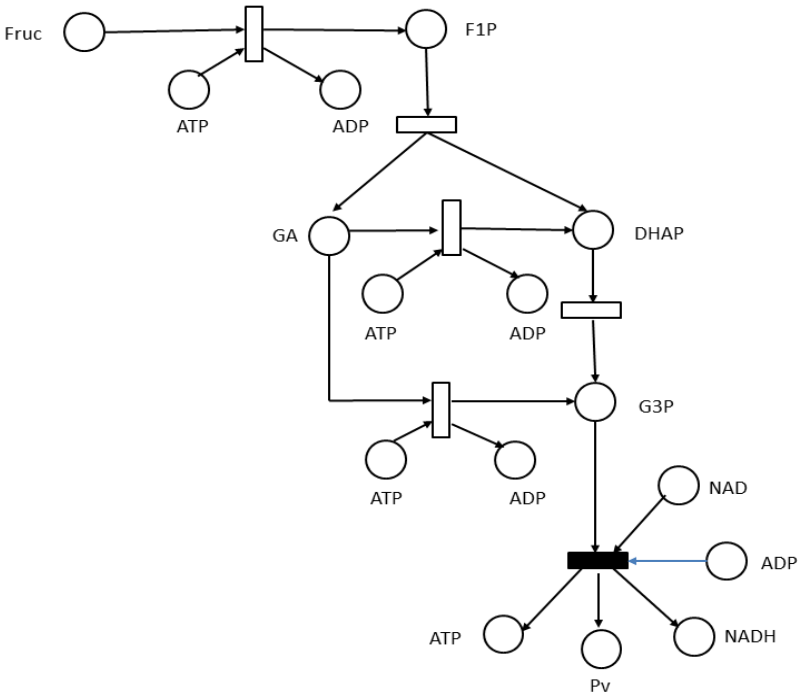


Figure A.2: Network model identified for fructose metabolism. The shaded transition is a guarded transition that stands for the sub-net corresponding to the phosphorylation stage of glycolysis. The pre-condition for using this transition is that G3P, NAD and ADP have to be present. The post-condition is that Pv, NADH and ATP are present. No invariants were specified, although the obvious one is that the values of all other metabolites are unchanged.

A.2 Pure Petri Nets with Sub-Nets (Fructolysis)

Place	States					
	0	1	2	3	4	5
fruc	1	0	0	0	0	0
atp	2	1	1	0	2	4
adp	4	5	5	6	4	2
f1p	0	1	0	0	0	0
ga	0	0	1	0	0	0
dhap	0	0	1	0	0	0
g3p	0	0	0	2	1	0
nad	2	2	2	2	1	0
nadh	0	0	0	0	1	2
pv	0	0	0	0	1	2

A.3 Extended Petri Nets (MAPK cascade)

In the data, MAP4K will stand for the extra-cellular signal; MAP3K, for the membrane-coupled receptor protein; MAP2K, for the protein involved in the second step; and MAPK, for the protein involved in the third step. We char-

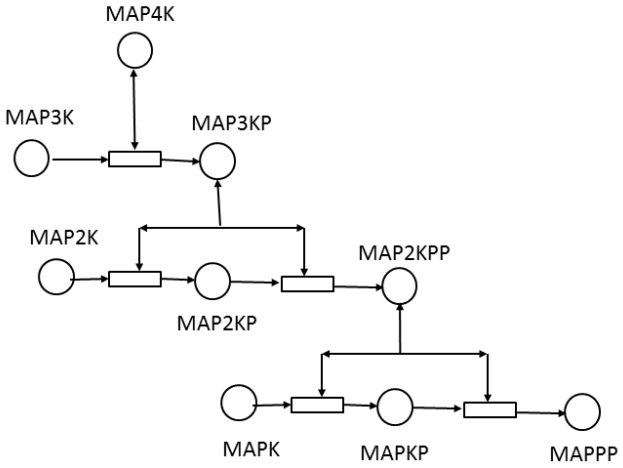


Figure A.3: Network identified from data of the MAPK cascade.

acterise phosphorylated forms by the addition of the letter 'P', and double phosphorylated forms by the letters 'PP'.

Place	States					
	0	1	2	3	4	5
map4k	1	1	1	1	1	1
map3k	1	0	0	0	0	0
map3kp	0	1	1	1	1	1
map2k	1	1	0	0	0	0
map2kp	0	0	1	0	0	0
map2kpp	0	0	0	1	1	1
mapk	1	1	1	1	0	0
mapkp	0	0	0	0	1	0
mapkpp	0	0	0	0	0	1

A.4 Extended Petri Nets with Sub-Nets (Yeast Pheromone)

Data are derived from the states shown below. The states simulate the known sequence of events in the order in which they are traditionally explained and understood in the biological literature. The external pheromone is α, which is eventually degraded by the extra-cellular secretion *bar*1 into an inactive form (here represented as α^-. The receptor is the G-protein coupled receptor (GPCR) *ste*2. A G-protein consists of essentially two sub-units G_α and $G_{\beta\gamma}$. Other proteins involved are either kinases that form part of a MAPK cascade or are used for scaffolding. We have only shown those chemicals involved largely in the feedback mechanism of the pheromone response: not represented here are the chemicals involved in the actual mating response. Both the expression of *bar*1 and the genes are triggered by the DNA transcription activator *ste*12.

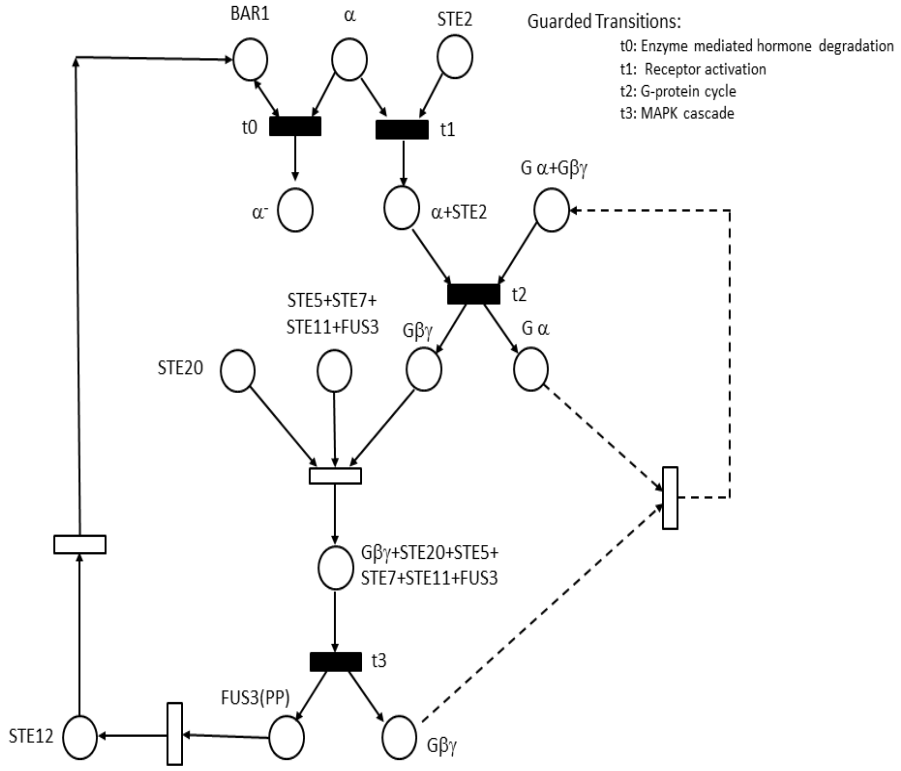


Figure A.4: Network identified for yeast pheromone response. The model uses a number of generic components of signalling pathways, encoded as guarded transitions. The pathway re-constituting the G-protein (shown dashed above) is identified when the state sequence provided is extended by further time steps.

Place	States								
	0	1	2	3	4	5	6	7	8
bar1	0	0	0	0	0	0	0	1	1
α	10	9	9	9	9	9	9	9	8
α^{-}	0	0	0	0	0	0	0	0	1
ste2	10	9	9	9	9	9	9	9	9
α+ste2	0	1	0	0	0	0	0	0	0
$G_{\alpha} + G_{\beta\gamma}$	10	10	9	9	9	9	9	9	9
G_{α}	0	0	1	1	1	1	1	1	1
$G_{\beta\gamma}$	0	0	1	0	0	1	1	1	1
ste5+ste11+ste7+fus3	10	10	10	9	9	9	9	9	9
ste20	10	10	10	9	9	9	9	9	9
ste20+$G_{\beta\gamma}$+ste5+ste11+ste7+fus3	0	0	0	1	0	0	0	0	0
fus3(pp)	0	0	0	0	0	1	0	0	0
ste20+$G_{\beta\gamma}$+ste5+ste11+ste7+fus3(pp)	0	0	0	0	1	0	0	0	0
ste12	0	0	0	0	0	0	1	0	0

A.5 Petri Nets with Missing Information (Fructolysis)

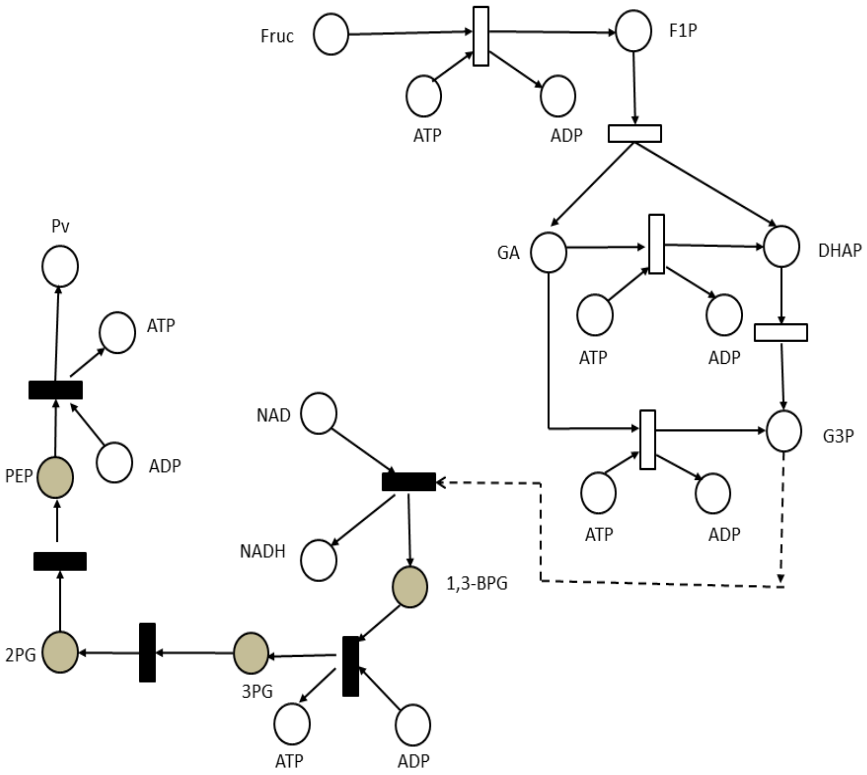


Figure A.5: Petri net model for fructolysis without guarded transitions for the glycolysis stages. The model uses a number of generic components of signalling pathways, encoded as guarded transitions. The system is able to identify the missing portion of the glycolysis model, with some help: (1) Places have to be specified as being "abducible" (that is, their values can be assumed). These are shown as shaded circles; (2) Guarded transitions have to be provided in background knowledge that specify feasible chemical reactions. These have been described elsewhere in the text for glycolysis; and (3) An upper-bound is specified on the size of the sub-net that should be hypothesized.

B Petri Nets and Logical Guarded Transition Systems

B.1 Formal Description of Petri Nets

Definition 8. (Petri net as a graph) *A Petri net (Peterson, 1981; David and Alla, 2010) is a bipartite directed labelled graph, with its vertex (node) set partitioned into two subsets: the set of places P and the set of transitions T. Directed edges (arcs) are either from a place to a transition (denoting consumption) or a transition to a place (denoting production). Both places and transitions may be labelled. Places are sometimes referred to as* conditions *and transitions as* events.

A place may be labelled by a non-negative integer number of "tokens" denoting its *state*, usually called a "marking" in the Petri net literature. The marking function is an assignment of tokens to places in the Petri net.

Definition 9. (Petri net marking function) *The marking function $m : P \to \mathbb{N}$ is a mapping from the set of places P to the set of natural numbers $\mathbb{N}$. We represent by $m(p)$ the marking of place $p \in P$, that is, the number of tokens assigned to place p. Additionally, we may represent by $m(u, p)$ the marking at time u of place $p \in P$.*

As noted by Peterson (Peterson, 1981), the marking function is a primitive of Petri nets. The state of the system modelled by a Petri net is given by its marking.

Definition 10. (Petri net marking) *The value of the marking function for every place $p_i \in P$ is called the* marking *of the net. This is represented as a $k-$vector $m = (n_1, n_2, \ldots, n_k)$, where $n_i = m(p_i)$, the marking of place p_i, and $k = |P|$ is the number of places.*

By Definition 10 the *state space* of a Petri net is the set of markings $\mathbb{N}^k$.

Petri net dynamics are based on a "firing" rule that determines state transitions, defined in terms of changes in the marking of a net, based on its flow relation.

Definition 11. (Flow relation) *The arc set of a Petri net is defined by the* flow *relation $F \subseteq (P \times T) \cup (T \times P)$.*

Definition 12. (Transition pre-set) *The set of places $p \in P$ on an arc to a particular transition $t \in T$ is the pre-set of t, denoted $Pre(t)$. In the case that $Pre(t) = \emptyset$ then t may be referred to as a source, or input, transition.*

Definition 13. (Transition post-set) *The set of places $p \in P$ on an arc from a particular transition $t \in T$ is the post-set of t, denoted $Post(t)$. In the case that $Post(t) = \emptyset$ then t may be referred to as a sink, or output, transition.*

We will sometimes represent the pre-set of transition t by ${}^\bullet t$, and the post-set by $t^\bullet$. A *self-loop* in a Petri net is a pair of directed arcs, (p, t) from a place p to a transition t, and (t, p) in the reverse direction.

Arcs (p, t), (t, p) may be labelled with a positive integer weight, denoting a number of tokens, indicating the "multiplicity" or enabling threshold of the arc.

Definition 14. (Transition weight) *Weights are assigned to arcs by a mapping w from the flow relation F to the set of natural numbers $\mathbb{N}$ (that is, $w : F \to \mathbb{N}$).*

Note that typically in diagrams of Petri nets only arc weights greater than one are shown.

A transition may become enabled at some time instant, dependent on the marking of places in its pre-set.

Definition 15. (Transition enabling) *A transition t is said to be enabled at time u if and only if $\forall p \in Pre(t).m(u, p) \geq w(p, t)$.*

An enabled transition may fire at some time instant, updating the marking of the places in its pre- and post-sets.

Definition 16. (Transition firing) *Given a transition t enabled at some time u that fires at some time v, $u < v$, the marking is updated as follows: $\forall p_{pre} \in Pre(t).m(v, p_{pre}) = m(u, p_{pre}) - w(p_{pre}, t)$, and $\forall p_{post} \in Post(t).m(v, p_{post}) = m(u, p_{post}) + w(t, p_{post})$.*

Definition 17. (Pure Petri net (David and Alla, 2010)) *A self-loop exists in a Petri net if there is a place p and a transition t such that $p \in Pre(t)$ and $p \in Post(t)$. A Petri net without self-loops is called* pure*, otherwise it is called* impure*.*

It is often useful to represent a pure Petri net by its equivalent *matrix* representation. In this paper we will represent a matrix using box brackets and write vectors in transposed form using parentheses, e.g., $(1, 0, -1) = [1, 0, -1]^{\mathsf{T}}$.

Definition 18. (Pure Petri net as a matrix) *Let P be a set of places and T a set of transitions. Let the number of places $k = |P|$ and the number of transitions $l = |T|$. A pure Petri net can be represented as a tuple $\langle P, T, M \rangle$. The $k \times l$ matrix M is the* incidence matrix *of the graph. The column vectors of M are the transition, or reaction, vectors of the Petri net. Let $1 \leq i \leq k$, $1 \leq j \leq l$. The incidence matrix M can be represented as the set of reaction vectors $\{r^{(t_1)}, r^{(t_2)}, \ldots, r^{(t_l)}\}$ where each $r^{(t_j)}$ is a column vector, representing transition t_j. An entry (i, j) in the matrix M denotes the net transfer of tokens at place p_i when transition t_j fires.*

Definition 19. (Reaction vector) *For a pure Petri net a transition t is denoted by the integer vector $r^{(t)} \in \mathbb{Z}^{|P|}$ with entries:*

$$r_p^{(t)} = \begin{cases} -w(p,t) & \text{if}(p,t) \in F \\ w(t,p) & \text{if}(t,p) \in F \\ 0 & \text{otherwise} \end{cases}$$

A change in state from marking m to m' due to the firing of a single transition t is then given by $m + r^{(t)} = m'$. In general, a change from m to m' may be due to the firing of a sequence of transitions. Then $m' - m = \sum_{r \in M} \lambda_r r$, where M is the incidence matrix of the Petri net and λ_r are non-negative integers. This is sometimes called the fundamental equation of Petri nets.

For a given set of states or markings, S, of a Petri net we can represent the change between any pair of states by a *difference* vector.

Definition 20. (Difference vector) *Let $k = |P|$, the number of places in a Petri net. Let d^v be a k-vector, the* difference *vector. Each component d^v_p of d^v denotes, for place $p \in P$, the state, or marking difference that occurs from time u to time v, $u < v$, defined as $d^v_p = m(v, p) - m(u, p)$.*

A bound may be placed on the number of tokens in any marking of a Petri net.

Definition 21. (h-boundedness (David and Alla, 2010)) *A place p is bounded for an initial marking m_0 if in any marking reachable from m_0 the number of tokens does not exceed a non-negative integer bound h (p is said to be h-bounded). A Petri net is bounded for an initial marking m_0 if all its places are bounded for m_0 (the Petri net is h-bounded if all its places are h-bounded).*

It is standard to ensure boundedness for any implementable Petri net.

Definition 22. (Empty Petri net) *An empty, or degenerate, Petri net is a Petri net with a (possibly empty) set of places and a (possibly empty) set of transitions and an empty set of arcs. Equivalently, it is a Petri net whose incidence matrix is zero.*

B.2 Extended Petri Nets

Interactions in biological networks such as genetic regulatory networks can be activating or repressing, and can be combined in regulatory functions where the effect of a number of input elements is integrated as a function of those inputs on the gene to be regulated (the target gene). For example, the regulatory function may be essentially Boolean combinational logic using NOT-, AND and OR- operators, leading to either activation or inhibition of the expression of the target gene (Bolouri and Davidson, 2002). However, the pure Petri net formalism is often not well suited to concise representation of such Boolean regulatory relations. To aid in this type of modelling two additional arc types are introduced.

An *read* or *activator* arc connects a place p to a transition t such that, although t is enabled by the marking of p in the usual way, the firing of t does not change the marking of p. Hence the activator arc acts only as a "test" or "on-switch" for the transition, "reading" the place marking.

Definition 23. (Transitions with activator arcs) *Place p may be connected to transition t by an activator arc. This is equivalent to a pair of arcs (p, t) and (t, p) with $w(p, t) = w(t, p)$. Transition t is not enabled unless the marking $m(p) \geq w(p, t)$. Since p is in both the pre- and post-sets of t, connected by arcs with equal weight, when t fires the net effect is to leave the marking of p unchanged.*

Read arcs are in fact syntactic sugar for self-loops, and do not change the modelling power of Petri nets. However, a Petri net with an *inhibitor* arc (p, t), where transition t cannot be enabled unless $m(p) < w(p, t)$, *does* extend modelling power, since there is no mechanism in a pure Petri net to check an unbounded place for zero. Petri nets with inhibitor arcs are called extended Petri nets (Peterson, 1981).

Definition 24. (Transitions with inhibitor arcs) *Place p may be connected to transition t by an inhibitor arc. Transition t is not enabled unless the marking $m(p) < w(p, t)$. Firing transition t does not change the marking of p.*

Definition 25. (Extended Petri net) *A Petri net with one or more inhibitor arcs is called an* extended *Petri net.*

It is typically assumed in extended Petri nets that the weight $w(p,t)$ on an inhibitor arc is 1 and the transition is enabled only when $m(p) = 0$, thus allowing the testing of places for zero tokens. Extended Petri nets are Turing-equivalent (Peterson, 1981).

B.3 Relation of Logical Guarded Transition Systems to Petri nets

We now prove some results showing the correspondence of pure and extended Petri nets to Logical Guarded Transition Systems. The basic idea is to show that both pure and extended Petri nets can be *represented* by equivalent Logical Guarded Transition Systems. By *equivalent* in these results we mean a form of *extensional* or behavioural equivalence, defined below, based on the formalism of equivalent sets of state transitions. In this sense we will show that: pure Petri nets are a special case of extended Petri nets; extended Petri nets are a special case of Logical Guarded Transition Systems; and hence pure Petri nets are also a special case of Logical Guarded Transition Systems. Although Logical Guarded Transition Systems cannot have greater expressive power than extended Petri nets, since the latter are Turing-equivalent, the former may have a more suitable representation for some modelling tasks.

In the following, let N be a pure Petri net, E be an extended Petri net, and G be a Logical Guarded Transition System. Without loss of generality, in this section for all Petri nets and Logical Guarded Transition Systems we fix: a set of places P, $k = |P|$, a set of transitions T, $l = |T|$, and a flow relation $F \subseteq (P \times T) \cup (T \times P)$.

From Definition 10, the state space S is the set of all k-vectors representing possible place markings. For the comparison of the different formalisms, we identify three types for places in the pre- and post-sets of each transition.

Definition 26. (Place types) *Each place $p \in P$ with an arc to a transition $t \in T$ may be assigned a* type *based on the nature of the arc connecting it to t. Let $P^{(t)} = {}^\bullet t \cup t^\bullet$, the set of places connected to t, be represented as a tuple of disjoint place types, $\langle P_P^{(t)}, P_A^{(t)}, P_I^{(t)} \rangle$, as follows: $P_P^{(t)} = \{p \mid (p,t) \in F$ or $(t,p) \in F$ is neither a read nor an inhibitor arc$\}$; $P_A^{(t)} = \{p \mid (p,t) \in F$ is a read arc$\}$; and $P_I^{(t)} = \{p \mid (p,t) \in F$ is an inhibitor arc$\}$. $P^{(t)} = P_P^{(t)} \cup P_A^{(t)} \cup P_I^{(t)}$.*

The letters 'P', 'A' and 'I' indicate 'pure', 'activator' and 'inhibitor'. We now introduce this structure into the state space, distinguishing states of places representing *objects* of the system that comprise the pure flow relation from *control* elements represented by places with an activatory or inhibitory role. This is referred to as an *extended state space.*

Definition 27. (Extended state space) *Let the k-vector $s \in \mathbb{N}^k$ be a state vector, or Petri net marking (Definition 10). With respect to a transition t, an* extended state *$s_X^{(t)}$ is a tuple $\langle s_P^{(t)}, s_A^{(t)}, s_I^{(t)} \rangle$, defined for place $p \in P^{(t)}$ as follows: $s_P^{(t)} = \{s_p \mid p \in P_P^{(t)}\}$; $s_A^{(t)} = \{s_p \mid p \in P_A^{(t)}\}$; and $s_I^{(t)} = \{s_p \mid p \in P_I^{(t)}\}$,*

where s_p is the p-th component of s. Let $S_X^{(t)}$ be the set of all extended states $s_X^{(t)}$ for transition t, and the extended state space S_X *be the union of extended states $S_X^{(t)}$ for all transitions t.*

Essentially, the extended state space is a restriction to the pre- and post-set places of a transition of the markings in the system state vector s, partitioned by place type. Note that this is purely a syntactic construct; it has no effect on how the model operates, but is useful for comparisons of formalisms in terms of their behaviours, i.e., their state transitions. Accordingly in the remainder of this section we will use the terms 'state' and 'extended state' interchangeably.

From Definitions 15, 16, 23 and 24 it is clear that the change in state due to the firing of a transition t is dependent only on the change in markings of the places in the pre- and post-sets of t. From this we obtain the *next-state* function, following (Peterson, 1981) but defined in terms of the logical setting of this section.

Definition 28. (State transition) *The next-state function $\delta : S_X \times S_X \rightarrow \{ TRUE, FALSE \}$ is defined for Petri nets and Logical Guarded Transition Systems in extended state $s1 \in S_X$ if there is an enabled transition (Definitions 15, 23 and 24) in $s1$, otherwise it is undefined. If δ is defined, the transition fires at some instant (Definitions 16, 23 and 24), the successor extended state $s2 \in S_X$ is obtained, and $\delta(s_i, s_j) = TRUE$. In a pure Petri net the transition is $t(s1, s2, r)$ as in Definition 1. In an extended Petri net the transition is $t(s1, s2, r)$ as in Definition 3. In a Logical Guarded Transition System the transition is $gt(s1, s2, t, r)$ as in Definition 5.*

It is now straightforward to define the state transition matrix.

Definition 29. (State transition matrix) *Let S_X be the set of all extended states. An $|S_X| \times |S_X|$ matrix S can be constructed for any Petri net, or Logical Guarded Transition System, representing the set of all state transitions δ. Let s_i, s_j be the i-th and j-th elements, respectively, of S_X, $1 \leq i, j \leq |S_X|$. Entry $S_{i,j}$ is 1 if and only if $\delta(s_i, s_j) = TRUE$, otherwise it is 0.*

Without loss of generality, we assume system *behaviours* can be viewed as sequences of transition firings and represented as sequences of states (Peterson, 1981).

Lemma 1. *The state transition matrix S is sufficient to represent the set of behaviours of a system modelled by a Petri net or Logical Guarded Transition System.*

Proof. *By induction, it is shown that any state generated as part of a sequence of states from some initial state by repeated application of the state transition matrix S will be a reachable state in a Petri net or Logical Guarded Transition System model. If in any initial state s_0 there is an enabled transition then, by Definitions 28 and 29, there will be an entry $S_{0,1} = 1$ in the state transition matrix for any successor state s_1 for which the state transition $\delta(s_0, s_1) = TRUE$. Hence each such s_1 is reachable in the model. Assume that s_i is a state reachable from s_0 in the model. From s_i let the set of successor states s_{i+1} be those with entries $S_{i,i+1} = 1$ in the state transition matrix. By Definitions 28 and 29 this is the set $\{s_{i+1} \mid \delta(s_i, s_{i+1}) = TRUE\}$, that is, those in the model for which a transition enabled in s_i may fire, and are therefore reachable. This completes the proof.*

The next result will be used in each of the main results comparing system models represented as pure or extended Petri nets and Logical Guarded Transition Systems.

Theorem 2. (System model equivalence) *System models M_A and M_B are equivalent if and only if their state transition matrices S_A and S_B are equivalent.*
Proof.
(only-if) *M_A is equivalent to M_B, therefore S_A is equivalent to S_B. Assume the opposite, that M_A and M_B are equivalent but S_A is not equivalent to S_B. Therefore, it follows from Definition 29 that there must be some entry $S_{A_{i,j}} \neq S_{B_{i,j}}$. That is, either (1) there is a state s that enables a transition in M_A that is not enabled by s in M_B, or (2) vice versa, state s enables a transition in M_B that is not enabled by s in M_A. In both cases (1) and (2) this contradicts the assumption, since any transition enabled by state s in M_A must be enabled in M_B, and vice versa, since they are equivalent.*
(if) *S_A is equivalent to S_B, therefore M_A is equivalent to M_B. By Lemma 1 the state transition matrix of a system model is a sufficient representation of it, therefore the claim is proved.*
This completes the proof.

We now relate Logical Guarded Transition Systems to pure and extended Petri nets by applying restrictions to their permitted transitions and showing that this makes them equivalent. The idea underlying the following results is to *reduce* all elements of the extended state space, and hence the state transition matrix, by eliminating one or more place types, leading to a simpler model class. It is well-known that extended Petri nets are a generalization of pure Petri nets — see, for example, (Peterson, 1981). However, for completeness, we first show that this is also the case in the logical setting.

Theorem 3. *For every pure Petri net N there is a corresponding extended Petri net E such that E is equivalent to N.*
Proof. *Given a pure Petri net N, construct an extended Petri net E, as follows. For each transition t_N in N, let there be a transition t_E in E such that $P_P^{(t_E)} = P_P^{(t_N)}$ and $S_P^{(t_E)} = S_P^{(t_N)}$. By construction, since for each transition t_N the associated transition function $t_N(s_i, s_j, r) = TRUE$, $s_i, s_j \in S_P^{(t_N)}$, from Definitions 1 and 3 we have that $s_j - s_i = r$ in N and hence also E, $s_i, s_j \in S_P^{(t_E)}$. A pure Petri net has no activator (Definition 17) or inhibitor arcs (Definition 25), so we set $P_A^{(t_E)} = P_A^{(t_N)} = P_I^{(t_E)} = P_I^{(t_N)} = \emptyset$. By Definition 3, each transition t_E in E has an associated transition function f_{t_E}. Since by construction there are no places in $P_A^{(t_E)}$ or $P_I^{(t_E)}$ for transition t_E, the component sets $s_A^{(t_E)}$ and $s_I^{(t_E)}$ of each extended state $s_X^{(t_E)}$ are empty. Therefore by Definition 3 f_{t_E} is trivially true. By this method an equivalent transition function can be constructed in E for every transition function in N. So for every state s_i for which a defined state transition $\delta_N(s_i, s_j) = TRUE$ in N, by the above construction there must be a state transition $\delta_E(s_i, s_j) = TRUE$ in E. From Definition 29 the corresponding state transition matrix entries $S_{N_{i,j}}$ (respectively, $S_{E_{i,j}}$) must be 1, otherwise they are 0. It follows by Theorem 2 that since the state transition matrices are equivalent, E is equivalent to N.*

Without loss of generality, we will assume that Logical Guarded Transition Systems have guard functions (Definition 4) represented only in terms of

pre-conditions, invariants and post-conditions defined on extended states (Definition 27).

Theorem 4. *For every extended Petri net E there is a corresponding Logical Guarded Transition System G such that G is equivalent to E.*
Proof. *Given an extended Petri net E, construct an equivalent Logical Guarded Transition System G, as follows. For each transition t_E in E, let there be a transition t_G in G such that $P^{(t_G)} = P^{(t_E)}$ and $S_X^{(t_G)} = S_X^{(t_E)}$. By Definition 6 each transition constraint (t_G, r) in G has an associated guarded transition function gt. By Definition 5 each gt has an associated guard function g and reaction vector r. Let the guard function g on transition t_G be represented as a set of pre-conditions, invariants and post-conditions on place types, as follows. For $P_P^{(t_G)}$ these define the transition enabling and transition firing rules of Definitions 15 and 16, and for $P_A^{(t_G)}$ and $P_I^{(t_G)}$ these define the transition enabling and transition firing rules of Definitions 23 and 24. Reaction vector r is the state difference $s_j - s_i$, $s_i, s_j \in S_X^{(t_G)}$. This construction of a guarded transition function clearly satisfies the requirements for an extended transition function in Definition 3, and since $S_X^{(t_G)} = S_X^{(t_E)}$, transition t_G in G becomes in this case equivalent to the corresponding transition t_E in E. The remainder of the proof proceeds as for Theorem 3.*

Relating Logical Guarded Transition Systems to pure Petri nets is now straightforward.

Corollary 5. *For every pure Petri net N there is a corresponding Logical Guarded Transition System G such that G is equivalent to N.*
Proof. *For any pure Petri net N, construct an extended Petri net E using the approach of Theorem 3. Then E is equivalent to N. Now construct a Logical Guarded Transition System G such that G is equivalent to E using the approach of Theorem 4. Since G is equivalent to E and E is equivalent to N, G is equivalent to N.*