

# Abstracting Asynchronous Multi-Valued Networks

Jason STEGGLES<sup>1</sup>

## Abstract

*Multi-valued networks (MVNs)* provide a simple yet expressive qualitative state based modelling approach for biological systems. In this paper we develop an abstraction theory for asynchronous MVNs that allows the state space of a model to be reduced while preserving key properties. The abstraction theory therefore provides a mechanism for coping with the state space explosion problem and supports the analysis and comparison of MVNs. We take as our starting point the abstraction theory for synchronous MVNs which uses the under-approximation approach of trace set inclusion. We show this definition of asynchronous abstraction allows the sound inference of analysis properties and preserves other interesting model properties. One problem that arises in the asynchronous case is that the trace set of an MVN can be infinite making a simple trace set inclusion check infeasible. To address this we develop a decision procedure for checking asynchronous abstractions based on using the finite state graph of an asynchronous MVN to reason about its trace semantics and formally show that this decision procedure is correct. We illustrate the abstraction techniques developed by considering two detailed case studies in which asynchronous abstractions are identified and validated for existing asynchronous MVN models taken from the literature.

**Keywords:** Multi-valued networks, abstraction techniques, biological modelling, qualitative modelling.

## 1 Introduction

*Multi-valued networks (MVNs)* [25, 34, 35] are an expressive qualitative modelling approach for biological systems (for example, see [35, 7, 28, 3]).

---

<sup>1</sup>School of Computing Science, Newcastle University, United Kingdom.  
Email: Jason.Steggles@ncl.ac.uk

They extend the well-known *Boolean network* [17, 18] approach by allowing the state of each regulatory entity to be within a range of discrete values instead of just *true* or *false*. The state of each regulatory entity is influenced by other regulatory entities in the MVN and entities update their state using either a *synchronous update strategy* [18, 39] where all entities simultaneously update their state, or an *asynchronous update strategy* [33, 15, 36] where entities update their state independently using a non-deterministic approach.

While MVNs have shown their usefulness for modelling and understanding biological systems further work is still needed to strengthen the techniques and tools available for MVNs. One interesting area that needs developing is a theory for abstracting MVNs. Abstraction techniques allow a simpler model to be identified which can then be used to provide insight into the more complex original model. Such techniques are well-known in the formal verification community as a means of coping with the complexity of formal models (see for example [9, 6, 10, 13]). The main motivation behind developing such a theory for MVNs can be summarised as follows:

- (1) The analysis of MVNs is limited by the well-known problem of state space explosion. Using abstraction allows analysis results about a model to be inferred from a simpler approximate model and so provides a means of coping with the state space explosion problem.
- (2) Often several MVNs are defined at different levels of abstraction when modelling a system. It is therefore important to be able to formally relate these models using an appropriate theory.
- (3) An abstraction theory would provide a basis for the step-wise refinement of MVNs.
- (4) Identifying an abstraction for a complex MVN provides a means of better visualising and understanding the behaviour an MVN, giving greater insight into the system being modelled.

The abstraction theory we present for asynchronous MVNs is based on extending the synchronous abstraction theory presented in [5]. We formulate a notion of what it means for an asynchronous MVN to be correctly abstracted by a simpler MVN with the same network structure but smaller state space. The idea is to use an abstraction mapping to relate the reduced state space of an abstraction to the original MVN. An abstraction is then said to be correct if its set of asynchronous traces is within the abstracted

traces of the original MVN. This definition of abstraction represents an *under-approximation* [9, 24] approach since not all of the behaviour of the original MVN is guaranteed to have been captured within the abstraction. We show that this approach allows sound analysis inferences about positive reachability properties in the sense that any reachability result shown on an abstraction must hold on the original model. An important result of this is that it therefore follows that all attractors of an asynchronous abstraction correspond to attractors in the original MVN. Note that an alternative approach commonly used in abstraction is to use an *over-approximation* [9, 24, 10] in which false positives may occur for reachability results. However, such an approach appears to be problematic for MVNs and we discuss this further in Section 3.

The non-deterministic nature of asynchronous MVNs mean that we encounter additional complications compared to the synchronous case; an asynchronous MVN can have an infinite set of traces which means that directly testing trace inclusion to check a proposed abstraction is infeasible. We overcome these difficulties by constructing a decision procedure for checking asynchronous abstractions that is based on the underlying finite state graph of an MVN. We introduce the idea of *step terms* which are used to denote possible ways to use sets of concrete states to represent abstract states. The decision procedure starts with the set of all possible step terms and then iteratively prunes the set until either a consistent abstract representation has been found or the set of remaining step terms is too small to make it feasible to continue. We provide a detailed proof that shows the decision procedure correctly identifies asynchronous abstractions and discuss the complexity of the decision procedure.

We illustrate the abstraction theory we develop by considering two detailed case studies in which asynchronous abstractions are identified for biological MVN models taken from the literature. The first case study considers an MVN model of the regulatory network that controls the biosynthesis of tryptophan by the bacteria *Escherichia coli* [29, 27]. Tryptophan is essential for the development of *E. coli* and its resource intensive synthesis is carefully controlled to ensure its production only occurs when an external source is not available. We investigate identifying asynchronous abstractions for an existing MVN model of this regulatory mechanism which was developed in [30]. The second case study considers a model for the genetic regulatory network controlling the lysis-lysogeny switch in the bacteriophage  $\lambda$  [32, 7]. Bacteriophage  $\lambda$  [34, 23] is a virus which after infecting the bacteria *E. coli*

makes a decision to switch to one of two possible reproductive phases: in the *lytic cycle* the virus generates as many new viral particles as the infected cell's resources allow and then lyse the cell wall to release the new phage; in the *lysogenic cycle* the  $\lambda$  DNA integrates into the host DNA providing it with immunity from other phages and allowing it to be replicated with each cell division. We consider identifying asynchronous abstractions for a four entity MVN model [32] of the lysis-lysogeny switching mechanism.

The paper is organized as follows. In Section 2 we provide a brief overview of the MVN framework and present a simple illustrative example. In Section 3 we formulate a notion of abstraction for asynchronous MVNs and consider the analysis properties that can be inferred from an abstraction. In Section 4 we present a decision procedure for checking asynchronous abstractions and provide a detailed proof of correctness for this procedure. In Section 5 we illustrate the theory and techniques developed by presenting two case studies that consider identifying asynchronous abstractions for MVN models taken from the biological modelling literature. Finally, in Section 6 we present some concluding remarks and discuss related work.

## 2 Multi-valued Network Models

In this section, we introduce *multi-valued networks* (MVNs) [25, 34, 35], a qualitative modelling approach which extends the well-known *Boolean network* [17, 18] approach by allowing the state of each regulatory entity to be within a range of discrete values. MVNs can therefore discriminate between the strengths of different activated interactions, something which Boolean networks are unable to capture. MVNs have been extensively studied in circuit design (for example, see [25, 20]) and successfully applied to modelling biological systems (for example, see [35, 7, 28, 3]).

An MVN consists of a set of logically linked entities  $G = \{g_1, \dots, g_k\}$  which regulate each other in a positive or negative way. Each entity  $g_i$  in an MVN has an associated set of discrete states  $Y(g_i) = \{0, \dots, m_i\}$ , for some  $m_i \geq 1$ , from which its current state is taken. Note that a Boolean network is therefore simply an MVN in which each entity  $g_i$  has a Boolean set of states  $Y(g_i) = \{0, 1\}$ . Each entity  $g_i$  also has a neighbourhood  $N(g_i) = \{g_{i_1}, \dots, g_{i_{l(i)}}\}$  which is the set of all entities that can directly affect its state. A given entity  $g_i$  may or may not be a member of  $N(g_i)$  and any entity in which  $N(g_i) = \{\}$  is taken to be an input entity whose regulation is outside the current model. The behaviour of each entity  $g_i$  based on

these neighbourhood interactions is formally defined by a logical next-state function  $f_{g_i}$  which calculates the next-state of  $g_i$  given the current states of the entities in its neighbourhood.

We can define an MVN more formally as follows.

**Definition 1.** An MVN  $MV$  is a four-tuple  $MV = (G, Y, N, F)$  where:

- i)  $G = \{g_1, \dots, g_k\}$  is a non-empty, finite set of entities;
- ii)  $Y = (Y(g_1), \dots, Y(g_k))$  is a tuple of state sets, where each  $Y(g_i) = \{0, \dots, m_i\}$ , for some  $m_i \geq 1$ , is the state space for entity  $g_i$ ;
- iii)  $N = (N(g_1), \dots, N(g_k))$  is a tuple of neighbourhoods, such that  $N(g_i) \subseteq G$  is the neighbourhood of  $g_i$ ; and
- iv)  $F = (f_{g_1}, \dots, f_{g_k})$  is a tuple of next-state multi-valued functions, such that if  $N(g_i) = \{g_{i_1}, \dots, g_{i_n}\}$  then the function  $f_{g_i} : Y(g_{i_1}) \times \dots \times Y(g_{i_n}) \rightarrow Y(g_i)$  defines the next state of  $g_i$ .  $\square$

Consider the following simple example  $PL2$  of an MVN defined in Figure 1 which models the core regulatory mechanism for the *lysis-lysogeny switch* [34, 23] in the bacteriophage  $\lambda$  (this model is taken from [32]). It consists of two entities  $CI$  and  $Cro$ , defined with neighbourhoods  $N(CI) = \{CI, Cro\}$  and  $N(Cro) = \{CI, Cro\}$ , and state spaces  $Y(CI) = \{0, 1\}$  and  $Y(Cro) = \{0, 1, 2\}$ . The next-state functions for each entity are defined using the state transition tables presented in Figure 1.(b) (where  $[g_i]$  is used to denote the next state of entity  $g_i$ ). We can summarise the interactions as follows: entity  $Cro$  inhibits the expression of  $CI$  and at higher levels of expression, also inhibits itself; entity  $CI$  inhibits the expression of  $Cro$  while

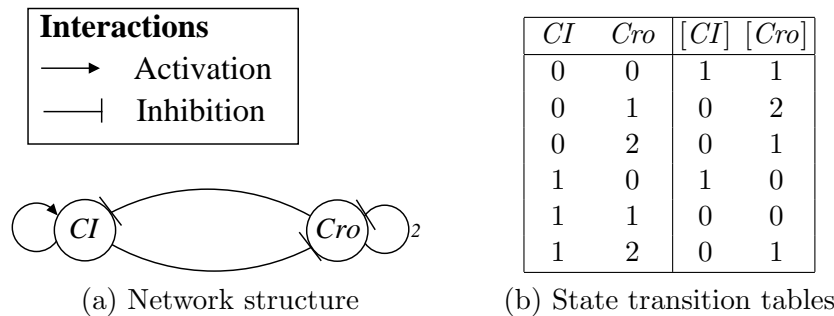


Figure 1: The MVN model  $PL2$  of the core regulatory mechanism for the lysis-lysogeny switch in bacteriophage  $\lambda$  (taken from [32]).

promoting its own expression. (Note we say  $A$  *inhibits*  $B$  if  $A$  acts to reduce the state of  $B$ , and  $A$  *promotes*  $B$  if  $A$  acts to increase the state of  $B$ .)

An MVN is said to satisfy the *unit state step assumption* [34] if every entity increases/decreases its state in unit steps of  $\pm 1$ . We note that the assumption of unit state steps is often made in the literature when using MVNs (for example, see [8]). We avoid making this explicit assumption in Definition 1 since it is not required for the theoretical results we develop. However, in Section 3 we show that our notion of abstraction preserves the unit state step assumption (see Theorem 12).

In the sequel, let  $MV = (G, Y, N, F)$  be an arbitrary MVN. In a slight abuse of notation we let  $g_i \in MV$  represent that  $g_i \in G$  is an entity in  $MV$ .

A *global state* of an MVN  $MV$  with  $k$  entities is represented by a tuple of states  $(s_1, \dots, s_k)$ , where  $s_i \in Y(g_i)$  represents the state of entity  $g_i \in MV$ . As a notational convenience we often use  $s_1 \dots s_k$  to represent a global state  $(s_1, \dots, s_k)$ . When the current state of an MVN is clear from the context we let  $g_i$  denote both the name of an entity and its corresponding current state. The *global state space* of an MVN  $MV$ , denoted  $S_{MV}$ , is the set of all possible global states  $S_{MV} = Y(g_1) \times \dots \times Y(g_k)$ .

The state of an MVN can be updated either *synchronously* (see [18, 39]), where the state of all entities is updated simultaneously in a single update step, or *asynchronously*<sup>2</sup> (see [33, 15]), where entities update their state independently. We define these update strategies more formally as follows:

**Definition 2.**

1) *Synchronous Update*: Given two states  $S_1, S_2 \in S_{MV}$ , we let  $S_1 \xrightarrow{Syn} S_2$  represent a *synchronous update step* such that  $S_2$  is the state that results from simultaneously updating the state of each entity  $g_i$  using its next-state function  $f_{g_i}$  and the appropriate states from  $S_1$  as indicated by the neighbourhood  $N(g_i)$ .

2) *Asynchronous Update*: For any  $g_i \in MV$  and any state  $S \in S_{MV}$  we let  $[S]^{g_i}$  denote the global state that results by updating the state of  $g_i$  in  $S$  using  $f_{g_i}$ . Define the global state function  $next^{MV} : S_{MV} \rightarrow \mathcal{P}(S_{MV})$  on any state  $S \in S_{MV}$  by

$$next^{MV}(S) = \{[S]^{g_i} \mid g_i \in MV \text{ and } [S]^{g_i} \neq S\}$$

<sup>2</sup>Note that different variations of the asynchronous semantics have been considered in the literature (see for example [26]) but that we focus on the one most commonly used for MVNs.

Given a state  $S_1 \in S_{MV}$  and  $S_2 \in next^{MV}(S_1)$ , we let  $S_1 \xrightarrow{Asy} S_2$  represent an *asynchronous update step*.  $\square$

Note that given the above definition, only asynchronous update steps that result in a change in the current state are considered (see [15]).

Continuing with our example, consider the global state 12 for *PL2* (see Figure 1) in which *CI* has state 1 and *Cro* has state 2. Then  $12 \xrightarrow{Syn} 01$  is a single synchronous update step on this state resulting in the new state 11. Considering an asynchronous update, we have  $next^{MV}(12) = \{02, 11\}$  and  $12 \xrightarrow{Asy} 02$  and  $12 \xrightarrow{Asy} 11$  are valid asynchronous update steps.

The sequence of update steps from an initial global state through  $S_{MV}$  is called a *trace*. In the case of the synchronous update semantics such traces are deterministic and infinite. Given that the global state space is finite, this implies that a synchronous trace must eventually enter a cycle, known formally as an *attractor cycle* [18, 35].

**Definition 3.** A *synchronous trace*  $\sigma$  is a list of global states  $\sigma = \langle S_0, S_1, S_2, \dots \rangle$ , where  $S_i \xrightarrow{Syn} S_{i+1}$ , for  $i \geq 0$ .  $\square$

The set of all synchronous traces, denoted  $Tr^S(MV)$ , therefore completely characterizes the behaviour of an MVN model under the synchronous semantics and is referred to as the *synchronous trace semantics* of *MV*. Note that we have one synchronous trace for each possible initial state and so the set of synchronous traces is always finite (see [18, 39]).

In the asynchronous case, traces are non-deterministic and can be finite or infinite. A single initial state can have an infinite number of possible asynchronous traces starting from it and thus in the asynchronous case there can be infinite number of traces.

**Definition 4.** An *asynchronous trace*  $\sigma$  is either:

- i) a finite sequence of global states  $\sigma = \langle S_0, S_1, \dots, S_n \rangle$ , where  $S_i \xrightarrow{Asy} S_{i+1}$ , for  $i = 0, \dots, n-1$ , and  $next^{MV}(S_n) = \{\}$ .
- ii) an infinite sequence of global states  $\sigma = \langle S_0, S_1, S_2, \dots \rangle$ , where  $S_i \xrightarrow{Asy} S_{i+1}$ , for  $i \geq 0$ .  $\square$

The set of all asynchronous traces, denoted  $Tr^A(MV)$ , therefore completely characterizes the behaviour of an MVN model under the asynchronous

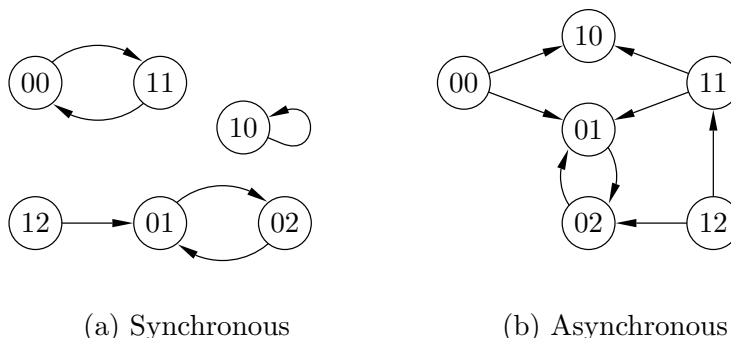


Figure 2: The (a) synchronous and (b) asynchronous state graphs for  $PL2$ .

semantics and is referred to as the *asynchronous trace semantics* of  $MV$ . Any state  $S \in S_{MV}$  which cannot be asynchronously updated, i.e.  $next^{MV}(S) = \{\}$ , is referred to as a *point attractor* [34].

In our running example,  $PL2$  has a state space of size  $|S_{PL2}| = 6$  and has the following (finite in this case) set of asynchronous traces:

$$\begin{array}{ll}
 \langle 00, 01, 02, 01, 02, \dots \rangle & \langle 11, 01, 02, 01, 02, \dots \rangle \\
 \langle 00, 10 \rangle & \langle 11, 10 \rangle \\
 \langle 01, 02, 01, 02, \dots \rangle & \langle 12, 02, 01, 02, 01, \dots \rangle \\
 \langle 02, 01, 02, 01, \dots \rangle & \langle 12, 11, 01, 02, 01, \dots \rangle \\
 \langle 10 \rangle & \langle 12, 11, 10 \rangle
 \end{array}$$

From the above traces it is clear that state 10 is a point attractor for  $PL2$ .

The behaviour of an MVN under the synchronous or asynchronous trace semantics can be represented by a state graph (for example, see [36]) in which the nodes are the global states and the edges are precisely the update steps allowed. We let  $SG^S(MV) = (S_{MV}, \xrightarrow{Syn})$  and  $SG^A(MV) = (S_{MV}, \xrightarrow{Asy})$  denote the corresponding state graphs under the synchronous and asynchronous trace semantics.

The synchronous and asynchronous state graphs for  $PL2$  are presented in Figure 2.

When analysing the behaviour of an MVN it is important to consider its attractors which can represent important biological phenomena, such as different cellular types like proliferation, apoptosis and differentiation [16]. In the synchronous case all traces are infinite and so must lead to a cyclic sequence of states which are taken as an *attractor* [18, 35, 39]. As



an example, consider  $PL2$  (see Figure 2.(a)) which has the point attractor  $10 \rightarrow 10$ ; and attractors  $00 \xrightarrow{Syn} 11 \xrightarrow{Syn} 00$  and  $01 \xrightarrow{Syn} 02 \xrightarrow{Syn} 01$  of period 2. In the asynchronous case we have *point attractors* which are states that cannot be updated and also the *strongly connected components*<sup>3</sup> in an MVN's asynchronous state graph are considered to be *attractors* [36]. Again, considering  $PL2$  (see Figure 2.(b)) we can see that in the asynchronous case it has a point attractor  $10$  and an attractor  $01 \xrightarrow{Asy} 02 \xrightarrow{Asy} 01$ .

### 3 Asynchronous Abstractions

In this section we consider developing a notion of abstraction for asynchronous MVNs. The idea is to formulate what it means for an MVN to be correctly abstracted by a simpler MVN with the same network structure but smaller state space. We take as our starting point the abstraction techniques developed for synchronous MVNs [5] and investigate extending these to the asynchronous case. We show that our approach allows sound analysis inferences about positive reachability properties and that all attractors of an asynchronous abstraction correspond to attractors in the original MVN.

We begin by recalling the notion of a state mapping and abstraction mapping [5] used to reduce an entity's state space.

**Definition 5.** Let  $MV$  be an MVN and let  $g_i \in MV$  be an entity such that  $Y(g_i) = \{0, \dots, m\}$  for some  $m > 1$ . Then a *state mapping*  $\phi(g_i)$  for entity  $g_i$  is a surjective mapping  $\phi(g_i) : \{0, \dots, m\} \rightarrow \{0, \dots, n\}$ , where  $0 < n < m$ .  $\square$

The state mapping must be surjective to ensure that all states in the new reduced state space are used. From a biological viewpoint it may also be reasonable to further restrict the state mappings considered, for example, only considering those mappings which are order-preserving. Note we only consider state mappings with a codomain larger than one, since a singular state entity does not appear to be of biological interest.

As an example, consider entity  $Cro \in PL2$  (see Figure 1) which has the state space  $Y(Cro) = \{0, 1, 2\}$ . It is only meaningful to simplify  $Cro$  to a Boolean entity and so there are two possible state mappings of interest to

<sup>3</sup>A *strongly connected component* of a graph is a maximal set of nodes such that every node is reachable from each other in the set [36].

achieve this:

$$\phi(Cro) = \{0 \mapsto 0, 1 \mapsto 1, 2 \mapsto 1\}, \quad \phi'(Cro) = \{0 \mapsto 0, 1 \mapsto 0, 2 \mapsto 1\}.$$

In the examples that follow we choose to use the first state map  $\phi(Cro)$  which maps state 0 to 0 and merges states 1 and 2 into a single state 1.

In order to be able to simplify several entities at the same time during the abstraction process we introduce the notion of a family of state mappings.

**Definition 6.** Let  $MV = (G, Y, N, F)$  be an MVN with entities  $G = \{g_1, \dots, g_k\}$ . Then an *abstraction mapping*  $\phi = \langle \phi(g_1), \dots, \phi(g_k) \rangle$  for  $MV$  is a family of mappings such that for each  $1 \leq i \leq k$  we have  $\phi(g_i)$  is either a state mapping for entity  $g_i$  or is the identity mapping  $I_{g_i} : Y(g_i) \rightarrow Y(g_i)$  where  $I_{g_i}(s) = s$ , for all  $s \in Y(g_i)$ . Furthermore, for  $\phi$  to be useful we normally insist that at least one of the mappings  $\phi(g_i)$  is a state mapping.  $\square$

Note in the sequel given a state mapping  $\phi(g_i)$  we let it denote both itself and the corresponding abstraction mapping containing only the single state mapping  $\phi(g_i)$ .

An abstraction mapping  $\phi$  can be used to abstract an asynchronous trace (see Definition 4) using a similar approach to that detailed for synchronous traces [5]. We begin by defining how an abstraction mapping can be lifted to a global state.

**Definition 7.** Let  $\phi = \langle \phi(g_1) \dots \phi(g_k) \rangle$  be an abstraction mapping for  $MV$ . Then  $\phi$  can be used to abstract a global state  $s_1 \dots s_k \in S_{MV}$  by applying it pointwise, i.e.  $\phi(s_1 \dots s_k) = \phi(g_1)(s_1) \dots \phi(g_k)(s_k)$ .  $\square$

We can apply an abstraction mapping  $\phi$  to an asynchronous trace  $\sigma \in Tr^A(MV)$  by applying  $\phi$  to each global state in the trace in the obvious way and then merging consecutive identical states. Note that removing consecutive identical states is needed to ensure abstracted traces are well-defined since by the definition of an asynchronous trace (see Definition 4) each asynchronous update rule must result in a new global state (i.e. the state of an entity has to change in order for a state transition to occur).

**Definition 8.** Let  $\phi = \langle \phi(g_1) \dots \phi(g_k) \rangle$  be an abstraction mapping for

$MV$  and let  $\sigma \in Tr^A(MV)$  be either a finite  $\sigma = \langle S_0, S_1, \dots, S_n \rangle$  or infinite  $\sigma = \langle S_0, S_1, S_2, \dots \rangle$  asynchronous trace. Then  $\phi(\sigma)$  is the abstracted trace that results by

- i) First apply the abstraction mapping to each state in  $\sigma$ , i.e. in the finite case  $\langle \phi(S_0), \phi(S_1), \dots, \phi(S_n) \rangle$  or in the infinite case  $\langle \phi(S_0), \phi(S_1), \phi(S_2), \dots \rangle$ .
- ii) Next merge consecutive identical global states in the trace into a single global state to ensure that no two consecutive states are identical in the resulting abstracted trace, i.e. suppose the result is an infinite trace  $\langle \phi(S_0), \phi(S_1), \phi(S_2), \dots \rangle$  then we know  $\phi(S_i) \neq \phi(S_{i+1})$ , for all  $i \in \mathbf{N}$ .  $\square$

We let  $\phi(Tr^A(MV)) = \{\phi(\sigma) \mid \sigma \in Tr^A(MV)\}$  denote the set of abstracted traces.

As an example, consider applying the abstraction mapping  $\phi(Cro) = \{0 \mapsto 0, 1 \mapsto 1, 2 \mapsto 1\}$  to the  $PL2$  asynchronous trace  $\langle 00, 01, 02, 01, 02, \dots \rangle$ . Part i) of Definition 8 above results in the trace  $\langle 00, 01, 01, 01, 01, \dots \rangle$ ; we now merge identical consecutive states to derive the abstracted trace  $\langle 00, 01 \rangle$ . It is interesting to note that abstracting an infinite trace can result in a finite abstracted trace, as above. The intuition here is that a cyclic set of states have been abstracted to a single point. The complete set of abstracted asynchronous traces of  $PL2$  using  $\phi(Cro)$  are given below:

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

The definition of an asynchronous abstraction is based on its trace semantics and follows along similar lines to that for the synchronous case [5]. We say an asynchronous abstraction is correct if its set of traces is within the abstracted traces of the original MVN. This definition of abstraction represents an *under-approximation* [9, 24] approach since not all of the behaviour of the original MVN is guaranteed to have been captured within the abstraction (we discuss the implications of this below).

**Definition 9.** Let  $MV_1 = (G_1, Y_1, N_1, F_1)$  and  $MV_2 = (G_2, Y_2, N_2, F_2)$  be two MVNs with the same structure, i.e.  $G_1 = G_2$  and  $N_1(g_i) = N_2(g_i)$ , for all  $g_i \in MV_1$ . Let  $\phi$  be an abstraction mapping from  $MV_2$  to  $MV_1$ . Then we say that  $MV_1$  *asynchronously abstracts*  $MV_2$  under  $\phi$ , denoted  $MV_1 \triangleleft_A^\phi MV_2$ , if, and only if,  $Tr^A(MV_1) \subseteq \phi(Tr^A(MV_2))$ .  $\square$

As an abstraction example, consider the MVN *APL2* defined in Figure 3 which has the same structure as *PL2* (see Figure 1) but is a Boolean model. Then given the abstraction mapping  $\phi(Cro) = \{0 \mapsto 0, 1 \mapsto 1, 2 \mapsto 1\}$  we

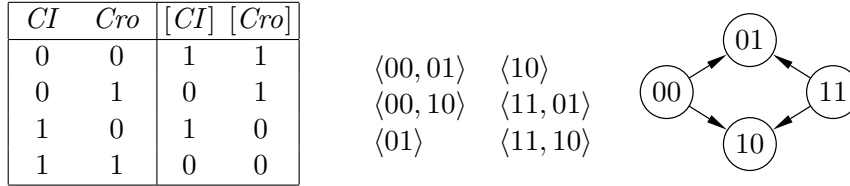


Figure 3: State transition tables defining *APL2*, the associated asynchronous trace semantics  $Tr^A(APL2)$  and asynchronous state graph.

can see that  $Tr^A(APL2) \subseteq \phi(Cro)(Tr^A(PL2))$  holds and so *APL2* is an abstraction of *PL2*, i.e.  $APL2 \triangleleft_A^{\phi(Cro)} PL2$  holds. Note that *APL2* has two point attractors: 01 and 10 which correspond to the two attractors associated with *PL2* (see Figure 2.(b)) and thus, *APL2* can be seen to be a good approximation of the behaviour of *PL2*.

Recall that one of the original motivations for developing an abstraction theory was to aid the analysis of complex MVNs. It is therefore important to consider what properties of an asynchronous MVN can be inferred from an abstraction MVN. We consider reachability and the existence of attractors since these are the main properties that are considered when analysing an MVN.

**Theorem 10.** Let  $MV_1 \triangleleft_A^\phi MV_2$  and let  $S_1, S_2 \in S_{MV_1}$ . If  $S_2$  is reachable from  $S_1$  in  $MV_1$  then there must exist states  $S'_1, S'_2 \in S_{MV_2}$  such that  $\phi(S'_1) = S_1$ ,  $\phi(S'_2) = S_2$ , and  $S'_2$  is reachable from  $S'_1$  in  $MV_2$ .

**Proof.** Since  $S_2$  is reachable from  $S_1$  there must exist a trace  $\sigma \in Tr^A(MV_1)$  which begins with state  $S_1$  and which contains state  $S_2$ . From Definition

9, we know that  $Tr^A(MV_1) \subseteq \phi(Tr^A(MV_2))$  must hold. Therefore there must exist a trace  $\sigma' \in Tr^A(MV_2)$  such that  $\phi(\sigma') = \sigma$ . From this it is straightforward to see that there must exist the required states  $S'_1$  and  $S'_2$  in  $\sigma'$  such that  $\phi(S'_1) = S_1$ ,  $\phi(S'_2) = S_2$ , and  $S'_2$  is reachable from  $S'_1$ .  $\square$

The above theorem indicates that inferring reachability properties from an abstraction is sound but not complete [13]. The implications of this can be summarised as follows: (i) If one state is reachable from another in an abstraction then a corresponding reachability property must hold in the original model; (ii) However, if one state is not reachable from another in an abstraction then a corresponding reachability property in the original MVN may or may not hold and more analysis will be required. This relates to the fact that our notion of abstraction represents an *under-approximation* [9, 24] of the original model. The alternative approach would be to use an *over-approximation* abstraction model [9, 24, 10] in which false positives can arise and need to be dealt with. It turns out that an over-approximation approach is not well suited to our goal of finding an abstraction model of an MVN that is itself a well-defined MVN. To illustrate the potential problems, consider what happens if a point attractor is identified to a non-attractor state by an abstraction mapping. In this case no over-approximation abstraction can exist since such an MVN would need to contain a state that was both a point attractor and also had a successor state. Thus the approach taken here of using an under-approximation appears to be the appropriate approach to use.

A consequence of the above reachability result is that all attractors in an abstraction must have corresponding attractors in the original MVN.

**Corollary 11.** If  $MV_1 \triangleleft_A^\phi MV_2$  then all attractors of  $MV_1$  must represent attractors in  $MV_2$ .

**Proof.** Follows directly from the definition of an asynchronous attractor and Theorem 10.  $\square$

It is interesting to consider what other properties of MVNs are preserved by abstractions. The result below shows that our notion of asynchronous abstraction preserves the unit state step assumption that is often placed on MVNs [34, 8].

**Theorem 12.** Let  $MV_1$  and  $MV_2$  be two MVNs with the same structure and let  $MV_2$  satisfy the *unit state step assumption*. Let  $\phi$  be an abstraction mapping from  $MV_2$  to  $MV_1$  which is *order preserving* (i.e. for any  $g_i \in MV_2$  and any states  $s_1, s_2 \in Y(g_i)$  (in  $MV_2$ ), if  $s_1 \leq s_2$  then  $\phi(g_i)(s_1) \leq \phi(g_i)(s_2)$ ). If  $MV_1 \triangleleft_A^\phi MV_2$  then  $MV_1$  must satisfy the unit state step assumption.

**Proof.** If  $MV_2$  satisfies the unit state step assumption then it is clear the state steps in every trace  $\sigma \in Tr^A(MV_2)$  must also satisfy this property. Since  $\phi$  is order preserving it follows by Definition 8 that every abstracted trace  $\phi(\sigma) \in \phi(Tr^A(MV_2))$  must satisfy this property. Now by assumption and Definition 9 we know  $Tr^A(MV_1) \subseteq \phi(Tr^A(MV_2))$ . Thus,  $Tr^A(MV_1)$  and therefore  $MV_1$  must satisfy the unit state step assumption.  $\square$

We conclude this section by discussing the problem of searching for possible abstractions for an MVN. Searching for abstractions is problematic due to the large number of potential models that need to be considered. For example, consider trying to find an abstraction with  $k$  entities each of which has  $n$  states. If we did this using a brute force approach then we would have to consider  $(n^{n^k})^k$  possible candidate models. In [5] an approach for limiting this search space for synchronous abstractions was proposed and we show that this approach can also be applied in the asynchronous case.

The key idea is to apply an abstraction mapping to the original MVN to produce a set of potential abstraction models [5].

**Definition 13.** Let  $\phi = \langle \phi(g_1), \dots, \phi(g_k) \rangle$  be an abstraction mapping for an MVN  $MV$ . For each entity  $g_i \in MV$  we can abstract the next-state function  $f_{g_i} : Y(g_{i_1}) \times \dots \times Y(g_{i_n}) \rightarrow Y(g_i)$  to a (possibly) non-deterministic next-state function

$$\phi(f_{g_i}) : \phi(g_{i_1})(Y(g_{i_1})) \times \dots \times \phi(g_{i_n})(Y(g_{i_n})) \rightarrow \phi(g_i)(Y(g_i))$$

by applying  $\phi$  to its definition, i. e. let  $s_j \in \phi(g_{i_j})(Y(g_{i_j}))$ , for  $j = 1, \dots, n$ , then we define  $\phi(f_{g_i})(s_1, \dots, s_n)$  by the set

$$\{\phi(f_{g_i}(s'_1, \dots, s'_n)) \mid s'_j \in Y(g_{i_j}), \phi(g_{i_j})(s'_j) = s_j, \text{ for } j = 1, \dots, n\}$$

We say that the MVN  $MV^A$  results from applying  $\phi$  to  $MV$  iff:

- (1)  $MV^A$  has the same entities and neighbourhood structure as  $MV$ ;
- (2) The state space of each entity  $g_i \in MV^A$  is the set  $\phi(g_i)(Y(g_i))$ ;

(3) For each  $g_i \in MV^A$  its next-state function

$$f_{g_i}^{MV^A} : \phi(g_{i_1})(Y(g_{i_1})) \times \cdots \times \phi(g_{i_n})(Y(g_{i_n})) \rightarrow \phi(g_i)(Y(g_i))$$

is a deterministic restriction of  $\phi(f_{g_i})$ .

We define  $\phi(MV)$  to be the set of all such MVNs, i.e.

$$\phi(MV) = \{MV^A \mid MV^A \text{ results from applying } \phi \text{ to } MV\} \quad \square$$

We can show that any asynchronous abstraction for an MVN  $MV$  under  $\phi$  must be a member of the set of potential abstractions  $\phi(MV)$ .

**Theorem 14.** If  $MV_1 \triangleleft_A^\phi MV_2$  then  $MV_1 \in \phi(MV_2)$ .

**Proof.** Suppose for a contradiction that  $MV_1 \notin \phi(MV_2)$ . Then there must exist at least one entity  $g_i \in MV_2$  such that the next-state function  $f_{g_i}^{MV_1} : Y(g_{i_1}) \times \cdots \times Y(g_{i_n}) \rightarrow Y(g_i)$  for  $g_i$  in  $MV_1$  is inconsistent with  $\phi(f_{g_i}^{MV_2})$ , i. e. for some  $s_1 \in Y(g_{i_1}), \dots, s_n \in Y(g_{i_n})$  we have

$$f_{g_i}^{MV_1}(s_1, \dots, s_n) \notin \phi(f_{g_i}^{MV_2})(s_1, \dots, s_n)$$

Given the above, it is clear there must exist a trace  $\sigma \in Tr^A(MV_1)$  which contains a state step based on the update  $f_{g_i}^{MV_1}(s_1, \dots, s_n)$  and therefore  $\sigma \notin \phi(Tr^A(MV_2))$ . However, this contradicts the assumption  $MV_1 \triangleleft_A^\phi MV_2$  since by Definition 9 we have  $Tr^A(MV_1) \subseteq \phi(Tr^A(MV_2))$ .  $\square$

## 4 A Decision Procedure for Abstractions

Having formulated a definition of an asynchronous abstraction in the previous section we are now interested in defining a procedure for checking whether a proposed abstraction  $MV_1$  is an asynchronous abstraction of an MVN  $MV_2$ . In the synchronous case the approach taken was to simply check that each trace  $\sigma \in Tr^S(MV_1)$  was contained within the set of abstracted traces  $\phi(Tr^S(MV_2))$ . However, in the asynchronous case both sets of traces  $Tr^A(MV_1)$  and  $\phi(Tr^A(MV_2))$  may be infinite and so such a simple set inclusion check is not feasible. Instead we propose a decision procedure based on using the finite state graph that summarise the behaviour of an asynchronous MVN. The idea is to consider all sets of states and associated

edges that can be used to model an abstract state. We then iterate through these removing those state sets which can not be represented given the current allowable state sets. If at any point we no longer have any state sets remaining for a particular abstract state then we have shown the abstraction is not valid and we terminate the decision procedure. If, on the other hand, we reach a point at which no more state sets can be removed then we know the abstraction must be valid and we can again terminate the procedure.

In the sequel let  $MV_1$  and  $MV_2$  be MVNs with the same structure and let  $\phi$  be an abstraction mapping from  $MV_2$  to  $MV_1$ .

In order to define a decision procedure  $\text{checkAsynAbs}(MV_1, MV_2, \phi)$  for checking if  $MV_1$  is an asynchronous abstraction under  $\phi$  of  $MV_2$  we begin by formulating some preliminary concepts.

i) *Representing abstract states:* Let  $S \in S_{MV_1}$  then we define

$$\phi^{-1}(S) = \{S' \mid S' \in S_{MV_2}, \phi(S') = S\}$$

to be the set of all states in  $MV_2$  that can represent the abstract state  $S$ .

ii) *Set of identical consecutive states:* For any state  $S' \in S_{MV_2}$  we define the set  $[S']^\phi$  of all *consecutive reachable states* from  $S'$  that have the same abstract state  $\phi(S')$ . Define  $[S']^\phi = \bigcup_{i \in \mathbf{N}} [S']_i^\phi$ , where  $[S']_i^\phi$  is defined recursively:  $[S']_0^\phi = \{S'\}$  and

$$[S']_{i+1}^\phi = \{S'_2 \mid S'_1 \in [S']_i^\phi, S'_2 \in \text{next}^{MV_2}(S'_1), \phi(S'_1) = \phi(S'_2)\}.$$

We now define the notion of a *step term*, an expression which is used to represent one possible way to model an abstract state and its next state connections using a set of concrete states. A step term has the form  $[S : \Gamma : D(S_1), \dots, D(S_m)]$  where  $S$  is the abstract state being modelled and  $\Gamma$  is a set of concrete states abstracting to  $S$  which are being used by the step term to model  $S$ . For each abstract next state  $S_i$  reachable from  $S$  the set of concrete states  $D(S_i)$  is included in the step term; it contains all those concrete next states reachable from  $\Gamma$  that abstract to  $S_i$ . (For an illustrative example of a step term see Figure 6.) Such step terms will form the basis of our decision procedure and are more formally defined as follows.

**Definition 15.** Let  $S \in S_{MV_1}$  and suppose  $\text{next}^{MV_1}(S) = \{S_1, \dots, S_m\}$ . Then for each non-empty set of states  $\Gamma \subseteq \phi^{-1}(S)$  we define the *step term*



$st(S, \Gamma)$  by

$$st(S, \Gamma) = [S : \Gamma : D(S_1), \dots, D(S_m)],$$

where  $D(S_i) = \{S'_2 \mid S'_1 \in \Gamma, S'_2 \in next^{MV_2}([S'_1]^\phi), \phi(S'_2) = S_i\}$ , and  $next^{MV_2}$  has been lifted from taking a single state as input to taking a set of states in the obvious way. Note that the use of  $[S'_i]^\phi$  is needed in the above definition to take account of the merging of consecutive identical states that occurs in abstracted traces (see part ii) in Definition 8).

We say a step term  $[S : \Gamma : D(S_1), \dots, D(S_m)]$  is *valid* iff:

- i) the states  $\Gamma$  used in a step term have the appropriate connections, i.e.  $D(S_i) \neq \{\}$ , for  $i = 1, \dots, m$ ; and
- ii) if  $next^{MV_1}(S) = \{\}$  (i.e.  $S$  is a point attractor in  $SG^A(MV_1)$ ) then for each  $S' \in \Gamma$  either: a) there must exist  $S'' \in [S']^\phi$  such that  $next^{MV_2}(S'') = \{\}$ ; or b) there exists states in  $[S']^\phi$  which form a cycle in  $SG^A(MV_2)$ .  $\square$

For any  $S \in S_{MV_1}$  we let  $Step(S)$  denote the set of all valid step terms

$$Step(S) = \{st(S, \Gamma) \mid \Gamma \subseteq \phi^{-1}(S), st(S, \Gamma) \text{ is valid}\}.$$

Observe that each valid step term  $st(S, \Gamma) \in Step(S)$  must correctly model in  $MV_2$  the connections between  $S \in S_{MV_1}$  and its corresponding next states  $next^{MV_1}(S)$  in  $MV_1$ .

The proposed decision procedure is presented in Figure 4. It works by creating a family  $C = \langle C(S) \subseteq Step(S) \mid S \in S_{MV_1} \rangle$  of sets of all valid step terms. It then repeatedly looks at each set of step terms  $C(S)$ , for each abstract state  $S \in S_{MV_1}$ , removing those that have next states that are not currently represented in the remaining stored step terms of  $C$ . The Boolean function  $notRepresented(S_i, D(S_i), C)$  used below is defined to be true if, and only if, there does not exist a step term  $st(S_i, \Gamma_i) \in C(S_i)$  such that  $\Gamma_i \subseteq D(S_i)$ .

To illustrate how the above decision procedure works we consider applying it to a simple example where we check whether a proposed abstraction  $MV_1$  does correctly abstract an MVN  $MV_2$ . For simplicity, we use an abstract description of the behaviour of  $MV_1$  and  $MV_2$  which is defined by the state graphs depicted in Figure 5, where  $A1, A2, \dots$  represent concrete global states and  $A, B, \dots$  represent abstract global states. Suppose we have an abstraction mapping  $\phi$  that maps global states in  $MV_2$  to abstract global states in  $MV_1$  in the obvious way, i.e.  $\phi(Ai) = A$ ,  $\phi(Bi) = B$ ,  $\phi(C1) = C$ , and  $\phi(D1) = D$ . Applying the initialization phase of the decision procedure results in the set of step terms given in the left column of the table in Figure

```

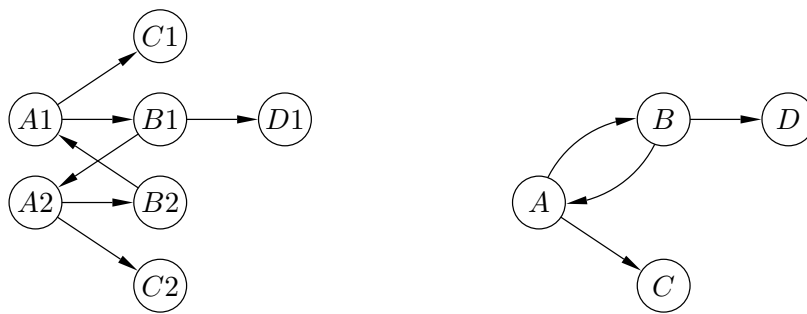
Algorithm checkAsynAbs( $MV_1, MV_2, \phi$ ):

/** Initialise valid step terms */
for each  $S \in S_{MV_1}$  do  $C(S) := Step(S)$ 
/** Iteratively check sets of step terms */
repeat
  done:=true
  for each  $S \in S_{MV_1}$  do
    for each  $[S : \Gamma : D(S_1), \dots, D(S_m)] \in C(S)$  do
      for  $i := 1$  to  $m$  do
        if  $notRepresented(S_i, D(S_i), C)$  then
           $C(S) := C(S) - \{[S : \Gamma : D(S_1), \dots, D(S_m)]\}$ 
          done:=false
      if  $C(S) = \{\}$  then return false
until (done)
return true

```

Figure 4: Decision procedure for checking asynchronous abstractions  $MV_1 \triangleleft_A^\phi MV_2$ .

6 (where the step terms for global states  $C$  and  $D$  have been omitted since they are point attractors and therefore cannot be removed). The decision procedure then executes the main outer loop three times, as depicted in Fig-



(a) State graph for  $MV_2$

(b) State graph for abstraction  $MV_1$

Figure 5: Abstract state graphs used to illustrate the decision procedure.

ure 6, each time removing those step terms that have next states that are not represented in the current set of step terms. At the end of these three iterations two consistent step terms remain and so the decision procedure terminates, returning true to indicate that  $MV_1$  is a correct asynchronous abstraction of  $MV_2$ .

Step Terms	Iter 1	Iter 2	Iter 3
$[A : \{A1\} : \{B1\}, \{C1\}]$	Okay	Remove	
$[A : \{A2\} : \{B2\}, \{C2\}]$	Remove		
$[A : \{A1, A2\} : \{B1, B2\}, \{C1, C2\}]$	Okay	Okay	Okay
$[B : \{B1\} : \{A2\}, \{D1\}]$	Remove		
$[B : \{B1, B2\} : \{A1, A2\}, \{D1\}]$	Okay	Okay	Okay

Figure 6: Table illustrating how the decision procedure for asynchronous abstractions would process the state graphs given in Figure 5.

The decision procedure given in Figure 4 is clearly inefficient and we now investigate this further by deriving a worst case performance bound. Assume  $MV_1$  is a Boolean model which has  $n$  entities (the size of its state space is therefore  $2^n$ ). The worst case occurs when  $MV_1$  is not an asynchronous abstraction of  $MV_2$ . We let  $k$  denote the upper bound on the number of states in  $MV_2$  that can be abstracted to a single state in  $MV_1$ , i.e.  $k \geq |\phi^{-1}(S)|$ , for all  $S \in S_{MV_1}$ . This value is important as it bounds the number of step terms, i.e. each abstract state can have at most  $2^k$  step terms and the total number of step terms possible is  $2^n \times 2^k = 2^{n+k}$ . The value  $k$  is calculated from the abstraction mapping being used and cannot be directly calculated from the size of concrete and abstract state spaces (to see this, note that we can increase the size of the concrete state space without affecting  $k$ ). However, multiplying the size of the abstract state space by  $k$  clearly gives an upper bound on the size of the concrete state space.

The decision procedure begins with an initialisation phase in which all the valid step terms are constructed. This has performance  $O(2^{n+k} \times nk)$ : we have  $2^{n+k}$  potential step terms to consider; and for each one, we have to calculate the connection sets  $D(S_i)$  and check its validity, all of which requires  $O(nk)$  work.

The main part of the decision procedure consists of three nested for loops which have a worst case performance of  $O(2^n \times 2^k \times n)$ , where:  $2^n$  is the size of the abstract state space;  $2^k$  is an upper bound on the number of

potential step terms for an abstract state; and  $n$  represents the maximum number of states that can be connected to a given state. For each of these iterations we apply the function *notRepresented* which has performance  $O(2^k \times n)$ , giving an overall performance of  $O(2^{n+2k} \times n^2)$ . Finally, we have the outer repeat until loop which iterates until one of the step term sets is empty. It therefore has a worst case performance of  $O(2^{n+k})$ . Combining this with the performance given above for the three nested loops gives an overall worst case performance of  $O(2^{2n+3k} \times n^2)$  for the decision procedure.

The above worst case performance is clearly very poor and means that in the worst case, the cost of checking an abstraction could be more than the cost of directly model checking the concrete state space. However, it is important to remember that the above is the worst case performance. In practice the decision procedure should perform much better than this, removing many more than one step term during each main iteration and terminating much sooner if the check is successful. The above has so far proved to be the case for the practical examples considered (e.g. the decision procedure worked well for both positive and negative checks made for the abstraction examples considered in Section 5). It is also important to remember that facilitating model checking was only one of the motivations for developing an abstraction theory. The other motivations discussed in the introduction (see Section 1) mean that having a decision procedure, however inefficient, is still worthwhile.

It remains to show formally that the above decision procedure correctly identifies asynchronous abstractions. We begin by showing that the decision procedure always terminate.

**Theorem 16.** The decision procedure  $\text{checkAsynAbs}(MV_1, MV_2, \phi)$  always terminates.

**Proof.** This follows from that fact we can only ever begin with a finite family of finite sets of step terms, that no step terms can ever be added, and that we must remove at least one step term in order to continue to the next iteration. Therefore the algorithm either terminates when no step terms are removed or continues to iterate until we reach a point where one set  $C(S)$  of step terms is empty, again resulting in termination of the algorithm.  $\square$

Let  $[S : \Gamma : D(S_1), \dots, D(S_m)]$  be a valid step term, let  $\alpha_1 \in \Gamma$  and

$\alpha_2 \in D(S_i)$ , for some  $1 \leq i \leq m$ . Then note that due to the way consecutive identical states are treated it may not directly hold that  $\alpha_1 \xrightarrow{Asy} \alpha_2$  since  $\alpha_2 \in next^{MV_2}([\alpha_1]^\phi)$ . We let  $\bar{\alpha}_1 = \alpha_1 \xrightarrow{Asy} \alpha_1^1 \xrightarrow{Asy} \dots \xrightarrow{Asy} \alpha_1^r$ , for  $\alpha_1^j \in [\alpha_1]^\phi$ , for  $1 \leq j \leq r$  represent the sequence of identical abstracted states needed such that  $\bar{\alpha}_1 \xrightarrow{Asy} \alpha_2$  does hold in  $MV_2$ .

The following lemma considers how step terms can be chained together and it is needed to prove the main correctness result below.

**Lemma 17.** Let  $C = \langle C(S) \subseteq Step(S) \mid S \in S_{MV_1} \rangle$  be a family of sets of valid step terms such that:

- i) For each  $S \in S_{MV_1}$  we have  $C(S) \neq \{\}$ ;
- ii) The family  $C$  is closed under step terms, i.e. for each  $S \in S_{MV_1}$  and  $[S : \Gamma : D(S_1), \dots, D(S_m)] \in C(S)$  there exists, for each  $1 \leq i \leq m$ , a step term  $st(S_i, \Gamma_i) \in C(S_i)$  such that  $\Gamma_i \subseteq D(S_1)$ .

Then every path<sup>4</sup>  $\gamma = \gamma_1 \xrightarrow{Asy} \dots \xrightarrow{Asy} \gamma_p$  in the abstraction state graph  $SG^A(MV_1)$  must have a corresponding path  $\alpha = \alpha_1 \xrightarrow{Asy} \dots \xrightarrow{Asy} \alpha_r$ ,  $r \geq p$ , in the original state graph  $SG^A(MV_2)$  such that  $\phi(\alpha) = \gamma$ .

**Proof.**

Let  $\gamma = \gamma_1 \xrightarrow{Asy} \dots \xrightarrow{Asy} \gamma_p$  be a path in the state graph  $SG^A(MV_1)$ . Then by assumptions i) and ii) it is straightforward to see there must exist a (not necessarily unique) chain of step terms

$$[\gamma_i : \Gamma_i : \dots, D(\gamma_{i+1}), \dots] \in C(\gamma_i), \quad st(\gamma_p, \Gamma_p) \in C(\gamma_p)$$

for  $1 \leq i < p$ , such that for  $j = 2, \dots, p$  we have  $\Gamma_j \subseteq D(\gamma_j)$ .

We now prove that for any  $\alpha_p \in \Gamma_p$  there must exist  $\alpha_i \in \Gamma_i$ , for  $1 \leq i < p$ , such that  $\alpha = \bar{\alpha}_1 \xrightarrow{Asy} \dots \xrightarrow{Asy} \bar{\alpha}_{p-1} \xrightarrow{Asy} \alpha_p$  is a path in  $SG^A(MV_2)$  with  $\phi(\alpha) = \gamma$ . We prove this using induction on  $p \in \mathbf{N}$ ,  $p \geq 2$  as follows.

- 1) *Induction Base.* Let  $p = 2$  and suppose we have a path  $\gamma_1 \xrightarrow{Asy} \gamma_2$ . Then we know there must exist step terms  $[\gamma_1 : \Gamma_1 : \dots, D(\gamma_2), \dots] \in C(\gamma_1)$  and  $st(\gamma_2, \Gamma_2) \in C(\gamma_2)$  such that  $\Gamma_2 \subseteq D(\gamma_2)$  (as explained above). Clearly

<sup>4</sup>We note that a path differs from a trace in that a trace represents a complete run of an MVN whereas a path is simply a walk through an MVN's state graph.

by the definition of step terms we know that for any  $\alpha_2 \in \Gamma_2$  there must exist  $\alpha_1 \in \Gamma_1$  such that  $\overline{\alpha_1} \xrightarrow{Asy} \alpha_2$  and

$$\phi(\overline{\alpha_1} \xrightarrow{Asy} \alpha_2) = \gamma_1 \xrightarrow{Asy} \gamma_2.$$

2) *Induction Step.* Let  $p = q + 1$ , for some  $q \in \mathbf{N}$ ,  $q \geq 2$ . Suppose we have a path  $\gamma = \gamma_1 \xrightarrow{Asy} \dots \xrightarrow{Asy} \gamma_q \xrightarrow{Asy} \gamma_{q+1}$ . Then we know there must exist step terms

$$[\gamma_i : \Gamma_i : \dots, D(\gamma_{i+1}), \dots] \in C(\gamma_i), \quad st(\gamma_{q+1}, \Gamma_{q+1}) \in C(\gamma_{q+1}),$$

for  $i = 1, \dots, q$  such that  $\Gamma_j \subseteq D(\gamma_j)$ , for  $j = 1, \dots, q + 1$  (as explained above). Then by the induction hypothesis we know for each  $\alpha_q \in \Gamma_q$  there must exist  $\alpha_i \in \Gamma_i$ , for  $1 \leq i < q$ , such that  $\overline{\alpha_1} \xrightarrow{Asy} \dots \xrightarrow{Asy} \overline{\alpha_{q-1}} \xrightarrow{Asy} \alpha_q$  is a path in  $SG^A(MV_2)$  with  $\phi(\overline{\alpha_1} \xrightarrow{Asy} \dots \xrightarrow{Asy} \overline{\alpha_{q-1}} \xrightarrow{Asy} \alpha_q) = \gamma_1 \xrightarrow{Asy} \dots \xrightarrow{Asy} \gamma_q$ . By the definition of step terms it follows that for any  $\alpha_{q+1} \in \Gamma_{q+1}$  there must exist  $\alpha_q \in \Gamma_q$  such that  $\overline{\alpha_q} \xrightarrow{Asy} \alpha_{q+1}$ . Combining this with the induction hypothesis given above shows the existence of the required path in  $SG^A(MV_2)$ .  $\square$

It now remains to show that  $\text{checkAsynAbs}(MV_1, MV_2, \phi)$  correctly acts as a decision procedure for asynchronous abstractions.

**Theorem 18.**  $\text{checkAsynAbs}(MV_1, MV_2, \phi)$  returns *true* if, and only if,  $MV_1 \triangleleft_A^\phi MV_2$ .

**Proof.**

*Part 1)*  $\Rightarrow$  Suppose  $\text{checkAsynAbs}(MV_1, MV_2, \phi)$  returns *true*. By inspecting the decision procedure we can see this means that a family  $\{C(S) \subseteq \text{Step}(S) \mid S \in \mathcal{S}_{MV_1}\}$  of non-empty sets of valid step terms must have been found which is closed under step terms. Consider any abstract trace  $\sigma \in \text{Tr}^A(MV_1)$ ; then by Lemma 17 and since any trace can be interpreted as a path in  $SG^A(MV_1)$  we have that there must exist a path  $\alpha$  in  $SG^A(MV_2)$  such that  $\phi(\alpha) = \sigma$ . It is straightforward to see that  $\alpha$  must be a well-defined trace for  $MV_2$ , i.e.  $\alpha \in \text{Tr}^A(MV_2)$ , by the definition of *valid* step term. This shows that  $\text{Tr}^A(MV_1) \subseteq \phi(\text{Tr}^A(MV_2))$  and so by Definition 9 we have  $MV_1 \triangleleft_A^\phi MV_2$ .

Part 2)  $\Leftarrow$  Suppose  $MV_1 \triangleleft_A^\phi MV_2$  then by Definition 9 we know

$$Tr^A(MV_1) \subseteq \phi(Tr^A(MV_2)) \quad (1)$$

Then we show that there must exist a non-empty family of sets of valid step terms which are closed under step term inclusion and thus that the decision procedure  $\text{checkAsynAbs}(MV_1, MV_2, \phi)$  must terminate returning *true*.

Let  $X \subseteq Tr^A(MV_2)$  be the set of traces that abstractly correspond to  $Tr^A(MV_1)$ :

$$X = \{\sigma' \mid \sigma' \in Tr^A(MV_2), \exists \sigma \in Tr^A(MV_1). \phi(\sigma') = \sigma\}$$

For each  $S \in S_{MV_1}$ , let  $X\langle S \rangle$  denote the set of all states that abstract to  $S$  which occur at the start of a trace in  $X$ :

$$X\langle S \rangle = \{\sigma'(1) \mid \sigma' \in X, \phi(\sigma'(1)) = S\}$$

where  $\sigma'(1)$  represents the first state of trace  $\sigma'$ .

Let  $\text{next}^{MV_1}(S) = \{S_1, \dots, S_m\}$ , then using Definition 15 we can define the step term

$$\text{st}(S, X\langle S \rangle) = [S : X\langle S \rangle : D(S_1), \dots, D(S_m)]$$

Clearly,  $\text{st}(S, X\langle S \rangle)$  must be valid by (1) above. We can now recursively define a set of step terms closed under step term inclusion from  $\text{st}(S, X\langle S \rangle)$  as follows. Define  $H(X\langle S \rangle) = \bigcup_{i \in \mathbf{N}} H(X\langle S \rangle)_i$ , where  $H(X\langle S \rangle)_i$  is defined recursively:  $H(X\langle S \rangle)_0 = \{\text{st}(S, X\langle S \rangle)\}$  and

$$H(X\langle S \rangle)_{i+1} = \{\text{st}(V_j, D(V_j) \cap X\langle V_j \rangle) \mid$$

$$[V : \Gamma : D(V_1), \dots, D(V_r)] \in H(X\langle S \rangle)_i, V_j \in \{V_1, \dots, V_r\}\}.$$

Clearly, the set  $H(X\langle S \rangle)$  is closed under step term inclusion by construction. It can also be shown that  $H(X\langle S \rangle)$  only contains valid step terms (the proof makes use of fact 1) above and is based on using induction on  $i \in \mathbf{N}$  to show all step terms in  $H(X\langle S \rangle)_i$  are valid). It therefore follows that for each  $S \in S_{MV_1}$  we know that each step term  $\text{st}(S_i, \Gamma) \in H(X\langle S \rangle)$  must occur in the initial family  $C$  of sets of step terms used in the decision procedure, i.e.  $\text{st}(S_i, \Gamma) \in C(S_i)$ . Since none of these step terms can be removed from  $C$  by the closure property it follows that the decision procedure  $\text{checkAsynAbs}(MV_1, MV_2, \phi)$  must terminate returning *true*.  $\square$

## 5 Case Studies

In this section we illustrate the techniques we have developed by presenting two detailed case studies. Each case study takes an existing biologically inspired MVN from the literature and considers identifying asynchronous abstractions using the decision procedure presented in Section 4.

### 5.1 Regulation of Tryptophan Biosynthesis in *E. coli*

Tryptophan is an amino acid which is essential for the development of *E. coli*. However, the synthesis of tryptophan is resource intensive and for this reason is carefully controlled to ensure it is only synthesised when no external source is available. The regulatory network that controls the biosynthesis of tryptophan by *E. coli* has been extensively studied (see for example [29, 27]).

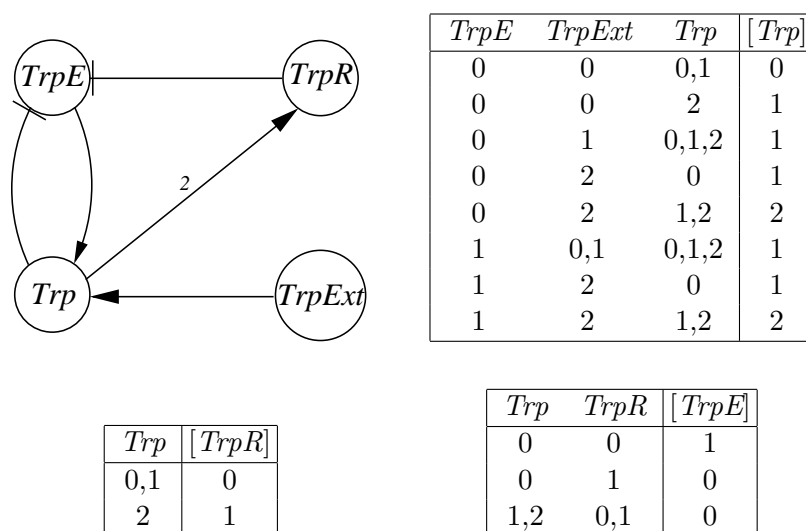


Figure 7: An MVN model *MTRP* of the regulatory mechanism for the biosynthesis of tryptophan in *E. coli* (from [30]). The state transition table for *TrpExt* has been omitted as this is a simple input entity. Note that the state transition tables use a shorthand notation where an entity is allowed to be in any of the states listed for it in a particular row.

Consider the MVN model *MTRP* for tryptophan biosynthesis presented in Figure 7 which is taken from [30]. It consists of four regulatory entities:



$TrpE$  – a Boolean input entity indicating the presence of the activated enzyme required for synthesising tryptophan;  $TrpR$  – a Boolean entity indicating if the repressor gene for tryptophan production is active;  $TrpExt$  – a ternary entity indicating the level of tryptophan in the external medium; and  $Trp$  – a ternary entity indicating the level of tryptophan within the bacteria. Note the above entity order is used when displaying global states for  $MTRP$ . We can see from the model that the presence of tryptophan in the external medium  $TrpExt$  directly affects the level of tryptophan within the bacteria  $Trp$  and that the activated enzyme  $TrpE$  is required to synthesise tryptophan. The presence of tryptophan within the bacteria deactivates the enzyme  $TrpE$  and at higher-levels also activates the repressor  $TrpR$  which then acts to inhibit the production of the enzyme  $TrpE$ .

The state space for  $MTRP$  consists of 36 global states and for this reason we do not reproduce its state graph here. Instead we simply note that the asynchronous state graph for  $MTRP$  comprises three disjoint graphs based on the following three attractors:  $0000 \xrightarrow{Asy} 1000 \xrightarrow{Asy} 1001 \xrightarrow{Asy} 0001 \xrightarrow{Asy} 0000$ ;  $0011$ ; and  $0122$ . To identify abstractions for  $MTRP$  we begin by defining appropriate state mappings for the non-Boolean entities  $TrpExt$  and  $Trp$  as follows:

$$\phi(Trp) = \{0 \mapsto 0, 1 \mapsto 1, 2 \mapsto 1\}, \quad \phi(TrpExt) = \{0 \mapsto 0, 1 \mapsto 1, 2 \mapsto 1\}.$$

These can then be combined into an abstraction mapping

$$\phi = \langle I_{TrpE}, I_{TrpR}, \phi(TrpExt), \phi(Trp) \rangle.$$

Following the approach discussed in Section 3 (see Theorem 14) we first apply this abstraction mapping to  $MTRP$  to produce a set  $\phi(MTRP)$  of candidate abstraction models. By analysing  $\phi(MTRP)$  we are able to establish that there are 8 possible candidate abstraction models (we have 4 choices for next-state of  $TrpR$  and 2 choices for  $Trp$ ). After investigating these candidate models we were able to identify one valid asynchronous abstraction  $ATRP$  (which is presented in Figure 8) for  $MTRP$  under  $\phi$  using the decision procedure  $\text{checkAsynAbs}(ATRP, MTRP, \phi)$ . Note that since  $Tr^A(ATRP)$  and  $\phi(Tr^A(MTRP))$  are finite trace sets in this case we were able to verify the result  $ATRP \triangleleft_A^\phi MTRP$ , by checking that  $Tr^A(ATRP) \subseteq \phi(Tr^A(MTRP))$  holds.

$Trp$	$[TrpR]$
0,1	0

$TrpE$	$TrpExt$	$Trp$	$[Trp]$
0	0	0,1	0
0	1	0,1	1
1	0,1	0,1	1

$Trp$	$TrpR$	$[TrpE]$
0	0	1
0	1	0
1	0,1	0

Figure 8: The asynchronous abstraction  $ATRP$  identified for  $MTRP$  under the state mappings  $\phi(Trp) = \{0 \mapsto 0, 1 \mapsto 1, 2 \mapsto 1\}$  and  $\phi(TrpExt) = \{0 \mapsto 0, 1 \mapsto 1, 2 \mapsto 1\}$ .

The state graph for  $ATRP$  consists of two disjoint graphs and has two attractors:  $0000 \xrightarrow{Asy} 1000 \xrightarrow{Asy} 1001 \xrightarrow{Asy} 0001 \xrightarrow{Asy} 0000$ ; and  $0011$ . It therefore successfully captures two of the three attractors present in  $MTRP$ .

## 5.2 The Lysis–Lysogeny Switch in Bacteriophage $\lambda$

The temperate bacteriophage  $\lambda$  is a virus which infects the bacteria *E. coli* [34, 23]. Once the bacteriophage  $\lambda$  has infected a cell it chooses between two very different methods of reproduction, known as the *lytic* and *lysogenic* cycles [34]. In most cases,  $\lambda$  enters the *lytic cycle*; it generates as many new viral particles as the host cell resources allow before producing an enzyme to lyse the cell wall and release the new phage into the environment. However,  $\lambda$  may choose to integrate its DNA into the host DNA and enter the *lysogenic cycle*. If this occurs, then the genes expressed in the  $\lambda$  DNA synthesize a repressor which blocks expression of other phage genes including those involved in its own excision, thus providing the cell with immunity from further infection. The phage  $\lambda$  is able to lie dormant, replicating with each subsequent cell division of the host but may at a later date enter the lytic cycle.

The core regulatory model  $PL2$  presented in Figure 1 (taken from [32]) was based on the cross-regulation between two regulatory genes,  $CI$  (the repressor gene) and  $Cro$ . This was extended in [32] by adding two further regulatory genes,  $CII$  and  $N$ . The result is a four entity MVN model  $PL4$ , presented in Figure 9, which contains two entities with non-Boolean state spaces, namely  $CI$  with states  $\{0, \dots, 2\}$  and  $Cro$  with states  $\{0, \dots, 3\}$ . The MVN  $PL4$  consists of 48 global states and its asynchronous behaviour

results in an infinite set of traces. It has one point attractor 2000 (where a global state has the order  $CI\ Cro\ CII\ N$ ) which corresponds to the lysogenic cycle and a range of cyclic attractors, the most important one of which is  $0200 \xrightarrow{Asy} 0300$ , corresponding to the lytic cycle.

We begin by looking to abstract the non-Boolean entities  $CI$  and  $Cro$  by defining appropriate state mappings. After considering the model, we define the following state mappings

$$\phi(CI) = \{0 \mapsto 0, 1 \mapsto 1, 2 \mapsto 1\}, \quad \phi(Cro) = \{0 \mapsto 0, 1 \mapsto 1, 2 \mapsto 1, 3 \mapsto 1\},$$

which form the abstraction mapping  $\phi = \langle \phi(CI), \phi(Cro), I_{CII}, I_N \rangle$ . Again,

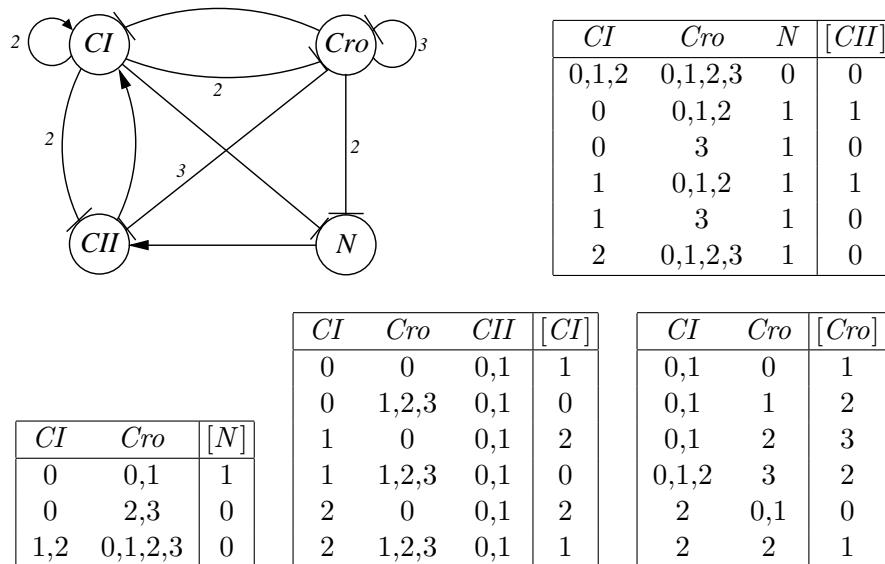


Figure 9: An extended MVN model  $PL_4$  of the control mechanism for the lysis-lysogeny switch in bacteriophage  $\lambda$  (taken from [32]).

we begin by restricting our search space for abstractions by first applying this abstraction mapping to  $PL_4$ . This results in the set  $\phi(PL_4)$  which contains 256 possible candidate abstraction models (we have 4 choices for  $CI$ , 4 choices for  $Cro$ , 8 choices for  $CII$ , and 2 choices for  $N$ ). By investigating these candidate models and applying the decision procedure `checkAsynAbs` we were able to identify two abstractions for  $PL_4$  under  $\phi$ . The abstractions,  $APL_{4_1} \triangleleft_A^\phi PL_4$  and  $APL_{4_2} \triangleleft_A^\phi PL_4$ , are presented in Figure 10. These abstractions have been carefully validated by hand, as have the other candidate

models considered that were correctly identified as not being abstractions. Interestingly, both abstractions appear to capture the key behaviour of  $PL_4$  in the sense that both contain the point attractors 1000 and 0100 which correspond to the two attractors presented above for  $PL_4$ .

$CI$	$Cro$	$[N]$
0	0	1
0	1	0
1	0,1	0

$CI$	$Cro$	$[Cro]$
0	0,1	1
1	0	0
1	1	1

$CI$	$Cro$	$CII$	$[CI]$
0	0	0,1	1
0	1	0,1	0
1	0	0,1	1
1	1	0,1	0

$CI$	$Cro$	$N$	$[CII]$
0,1	0,1	0	0
0	0	1	1
<b>0</b>	<b>1</b>	<b>1</b>	<b>1</b> or <b>0</b>
1	0	1	0
1	1	1	1

Figure 10: The transition tables for the two abstractions  $APL_{4_1}$  and  $APL_{4_2}$  identified for  $PL_4$  under  $\phi$ , where all the transition tables are the same except for  $CII$  where  $011 \rightarrow 1$  for abstraction  $APL_{4_1}$  but  $011 \rightarrow 0$  for abstraction  $APL_{4_2}$ .

## 6 Conclusions

In this paper we have developed an abstraction theory for asynchronous MVNs by extending the ideas developed for synchronous MVNs [5] based on trace set inclusion. In particular, we defined what it means for an MVN to be correctly abstracted by a simpler MVN with the same network structure but smaller state space. The abstraction approach used is based on an *under-approximation* approach [9, 24] in which an abstraction captures a subset of the behaviour of the original MVN. We showed that this approach allows positive reachability properties of an MVN to be inferred from a corresponding asynchronous abstraction and that all attractors of an asynchronous abstraction correspond to attractors in the original MVN. An alternative approach would be to use an *over-approximation* approach [9, 24, 10] in which false positive reachability results can arise. However, the construction of an abstraction model which over-approximates an MVN's behaviour appears to be problematic if we wish to remain within the MVN

framework (see Section 3 for a discussion of this).

Directly checking asynchronous abstractions turned out to be problematic since an asynchronous MVN may have an infinite set of traces making it infeasible to carry out a simple trace inclusion check. To address this we developed a decision procedure for checking asynchronous abstractions based on using the finite state graph of an asynchronous MVN. We introduced the notion of *step terms* to denote possible ways to use sets of concrete states to represent abstract states. The decision procedure worked by iteratively pruning the set of step terms until either a consistent abstract representation has been found or the set of remaining step terms is too small to make it feasible to continue. Importantly, we provided a detailed proof of correctness for the decision procedure.

Note that as it stands, the decision procedure is inefficient and we derived a worst case complexity of  $O(2^{2n+3k} \times n^2)$ . This poor worst case behaviour means that the cost of checking an abstraction could be more than the cost of directly model checking the concrete state space. However, it is important to remember that in practice the decision procedure should perform much better than this worst case. Indeed, practical experience so far has shown this to be the case; for example, the decision procedure worked well for all the positive and negative checks made for the abstraction examples considered in Section 5. It is also important to remember that facilitating model checking was only one of the motivations for developing an asynchronous abstraction theory. Other important motivations (see Section 1) included: formally relating MVN models at different levels of abstraction; aiding the visualisation and comprehension of an MVN; and providing the basis for a refinement theory. The above motivations mean that having a formally proved correct decision procedure, however inefficient, is still a substantial step forward. Work is now on going to refine the decision procedure and to efficiently implement it as part of a tool for MVN abstraction checking. Such a tool will provide the support needed to carry out more complex case studies, for example supporting the work currently underway to investigate abstractions for the relatively complex MVN model of the carbon starvation response in *E. coli* presented in [3].

We illustrated the abstraction theory and techniques developed by considering two detailed case studies based on identifying Boolean abstractions for existing asynchronous MVN models of biological systems. The abstractions found in both cases proved to faithfully represent the behaviour of the original MVNs and in particular, captured key attractors known to exist in

the original MVNs. The case studies illustrate the potential for the abstraction theory presented and in particular, how it allows the balance between the level of abstraction used and the tractability of analysis to be explored.

An alternative approach for abstracting MVNs is to reduce the number of regulatory entities in an MVN while ensuring the preservation of key properties (see [21, 37, 22]). This approach seems to be complimentary to the one developed here and we are currently investigating combining these ideas. Another possible abstraction approach would be to make use of results on modelling MVNs using Petri nets [11, 3, 4, 8] and to then apply Petri net abstraction techniques (see for example [31, 19, 38]). Such an approach appears promising from an analysis point of view but problematic in that the resulting Petri net abstraction may not be interpretable as an MVN and so force the modeller to explicitly use a different modelling formalism.

One interesting area for future work is to investigate automatically constructing abstractions for a given MVN and abstraction mapping. Some initial work on restricting the search space for such abstractions was presented at the end of Section 3 but more work is needed here. One idea is to consider developing refinement techniques similar to those of *CEGAR* (*Counterexample Guided Abstraction Refinement*) [10] and other abstraction refinement techniques [24]. Closely linked to this idea is the notion of a maximal abstraction, that is an abstraction which captures the largest possible behaviour of the original MVN with respect to all other possible abstractions for the given abstraction mapping. In future work we intend to investigate developing such a notion and in particular, consider how to automate the construction of such maximal abstractions.

**Acknowledgments.** We would like to thank Richard Banks and Maciej Koutny for their advice and support during the preparation of this paper. We would also like to thank the anonymous referees for their very helpful comments and suggestions.

## References

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

- 
- [2] R. Banks. *Qualitatively Modelling Genetic Regulatory Networks: Petri Net Techniques and Tools*. Ph. D. Dissertation, School of Computing Science, University of Newcastle upon Tyne, 2009.
  - [3] R. Banks and L. J. Steggles. A High-Level Petri Net Framework for Multi-Valued Genetic Regulatory Networks. *Journal of Integrative Bioinformatics*, 4(3):60, 2007.
  - [4] R. Banks, V. Khomenko, and L. J. Steggles. Modelling Genetic Regulatory Networks. In: I. Koch, W. Reisig and R. Schreiber (Eds), *Modelling in Systems Biology: the Petri Net Approach*, pages 73-100, Computational Biology Series, Springer Verlag, 2010.
  - [5] R. Banks and L. J. Steggles. An Abstraction Theory for Qualitative Models of Biological Systems. *Electronic Proceedings in Theoretical Computer Science*, 40:23-38, 2010.
  - [6] S. Bensalem, Y. Lakhnech, and S. Owre. Computing Abstractions of Infinite State Systems Compositionally and Automatically. In: *Proc. of the 10th Int. Conference on Computer Aided Verification*, Lecture Notes In Computer Science 1427, pages 319–331, Springer-Verlag, 1998.
  - [7] C. Chaouiya, E. Remy, and D. Thiéffry. Petri Net Modelling of Biological Regulatory Networks. *Journal of Discrete Algorithms*, 6(2):165–177, 2008.
  - [8] C. Chaouiya, A. Naldi, E. Remy, and Thiéffry. Petri net representation of multi-valued logical regulatory graphs. *Natural Computing*, 10(2):727–750, 2011.
  - [9] E. M. Clarke, O. Grumberg, and D. E. Long. Model Checking and Abstractions. *ACM Transactions on Programming Languages and Systems*, 16(5):1512 - 1542, 1994.
  - [10] E. M. Clarke, O. Grumberg, S. Jha, Y. Lu and H. Veith. Counterexample-guided abstraction refinement for symbolic model checking. *Journal of the ACM*, 50(5):752–794, 2003.
  - [11] J. Comet, H. Klaudel, and S. Liazu. Modeling Multi-valued Genetic Regulatory Networks Using High-Level Petri Nets. Lecture Notes in Computer Science, vol. 3536, page 208–227, Springer–Verlag, 2005.

- [12] B. Drossel, T. Mihaljev, and F. Greil. Number and length of attractors in a critical Kauffman model with connectivity one. *Physical Review Letters*, 94(8), 2005.
- [13] V. DSilva, D. Kroening, and G. Weissenbacher. A Survey of Automated Techniques for Formal Software Verification. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 27(7):1165–1178, 2008.
- [14] A. Esmaceli and C. Jacob. Evolutionary exploration of Boolean networks *Proceedings of the IEEE Congress on Evolutionary Computation*, pages 3396 - 3403, 2008.
- [15] I. Harvey and T. Bossomaier. Time Out of Joint: Attractors in Asynchronous Random Boolean Networks. In: P. Husbands and I. Harvey (eds.), *Proc. of ECAL97*, pages 67–75, MIT Press 1997.
- [16] S. Huang and D. Ingber. Shape-dependent control of cell growth, differentiation, and apoptosis: Switching between attractors in cell regulatory networks. *Experimental Cell Research*, 261(1):91–103, 2000.
- [17] S. A. Kauffman. Metabolic stability and epigenesis in randomly constructed genetic nets. *Journal of Theoretical Biology*, 22(3):437–467, 1969.
- [18] S. A. Kauffman. *The origins of order: Self-organization and selection in evolution*. Oxford University Press, New York, January 1993.
- [19] P. Küngas. Petri Net Reachability Checking Is Polynomial with Optimal Abstraction Hierarchies. *Lecture Notes in Computer Science*, vol. 3607, pages 149–164, Springer-Verlag, 2005. Copyright: 2005
- [20] A. Mishchenko and R. Brayton. Simplification of non-deterministic multi-valued networks. In: *ICCAD '02: Proc. of the 2002 IEEE/ACM Int. Conference on Computer-aided design*, pages 557–562, 2002.
- [21] A. Naldi, E. Remy, D. Thieffry, and C. Chaouiya. A Reduction of Logical Regulatory Graphs Preserving Essential Dynamical Properties. In: *Proc. of CMSB '09*, *Lecture Notes in Bioinformatics* 5688, pages 266 - 280, Springer-Verlag, 2009.



- 
- [22] A. Naldi, E. Remy, D. Thieffry, and C. Chaouiya. Dynamically consistent reduction of logical regulatory graphs. *Theoretical Computer Science*, 412(21):2207-2218, 2011.
- [23] A.B. Oppenheim, O. Kobiler, J. Stavans, D. L. Court, and S. L. Adhya. Switches in bacteriophage  $\lambda$  development. *Annual Review of Genetics*, 39:4470-4475, 2005.
- [24] R. Pelánek. *Reduction and Abstraction Techniques for Model Checking*. PhD thesis, Masaryk University Brno, 2006.
- [25] R. Rudell and A. Sangiovanni-Vincentelli. Multiple-Valued Minimization for PLA Optimization. *IEEE Transactions on Computer-Aided Design*, CAD-6, 1987.
- [26] A. Saadatpour, I. Albert, and R. Albert. Attractor analysis of asynchronous Boolean models of signal transduction networks. *Journal of Theoretical Biology*, 266:641-656, 2010.
- [27] M. Santillán and M. C. Mackey. Dynamic regulation of the tryptophan operon: A modeling study and comparison with experimental data *PNAS*, 98(4): 1364-1369, 2001.
- [28] M. Schaub, T. Henzinger, and J. Fisher. Qualitative networks: A symbolic approach to analyze bio-logical signaling networks. *BMC Systems Biology*, 1:4, 2007.
- [29] A. K. Sen and W. Liu. Dynamic analysis of genetic control and regulation of amino acid synthesis: The tryptophan operon in *Escherichia coli*. *Biotechnology and Bioengineering*, 35(2):185-194, 1990.
- [30] E. Simão, E. Remy, D. Thieffry and C. Chaouiya. Qualitative modelling of regulated metabolic pathways: application to the tryptophan biosynthesis in *E. Coli*. *Bioinformatics*, 21: ii190-196, 2005.
- [31] I. Suzuki and T. Murata. A method for stepwise refinement and abstraction of petri nets. *Journal of Computer and System Sciences*, 27:517-6, 1983.
- [32] D. Thieffry and R. Thomas. Dynamical behaviour of biological regulatory networks - II. Immunity control in bacteriophage lambda. *Bulletin of Mathematical Biology*, 57:277-295, 1995.

- [33] R. Thomas. Boolean formalization of genetic control circuits. *Journal of Theoretical Biology*, 42:563–585, 1990.
- [34] R. Thomas and R. D’Ari. *Biological Feedback*, CRC Press, 1990.
- [35] R. Thomas, D. Thieffry and M. Kaufman. Dynamical Behaviour of Biological Regulatory Networks - I. Biological Role of Feedback Loops and Practical use of the Concept of Loop-Characteristic State. *Bulletin of Mathematical Biology*, 57:247–276, 1995.
- [36] L. Tournier and M. Chaves. Uncovering operational interactions in genetic networks using asynchronous Boolean dynamics. *Journal of Theoretical Biology*, 260:196–209, 2009.
- [37] A. Veliz–Cuba. Reduction of Boolean Networks.  
<http://arxiv.org/abs/0907.0285>, submitted 2009. (Visited Dec 2010)
- [38] H. Wimmel and K. Wolf, Karsten. Applying CEGAR to the petri net state equation. In: *Proc. of TACAS’11/ETAPS’11*, Lecture Notes in Computer Science, vol. 6605, pages 224–238, Springer–Verlag, 2011.
- [39] A. Wuensch. Basins of Attraction in Network Dynamics: A Conceptual Framework for Biomolecular Networks, In: G.Schlosser and G.P.Wagner (Eds), *Modularity in Development and Evolution*, pages 288–311, Chicago University Press, 2002.