Fundamenta Informaticae XXI (2001) 1001–1026
DOI 10.3233/FI-2015-0000
IOS Press

Modelling and Analysing Qualitative Biological Models using Rewriting Logic

Abdullah Alhumaidan
School of Computing Science,
University of Newcastle, U. K.
A.Alhumaidan@ncl.ac.uk

Jason Steggles
School of Computing Science,
University of Newcastle, U. K.
Jason.Steggles@ncl.ac.uk

Abstract. Qualitative logical modelling techniques play an important role in biology and are seen as crucial for developing scalable methods for modelling and synthesizing biological systems. While a range of interesting work has been done in this area there still exists challenging issues that need to be addressed for the practical application of these modelling techniques. In this paper we present an algebraic framework for exploring these issues by developing techniques for modelling and analysing qualitative biological models using *Rewriting Logic (RL)*. The aim here is to develop a universal formal framework which is able to integrate models expressed in different formalisms (e.g. Boolean networks, Petri Nets and process algebra) and provide a basis for new work in this area (e.g. merging models based on different formalisms; compositional model construction and analysis; and tools for synthetic biology). We take as our starting point *Multi-valued networks (MVNs)*, a simple yet expressive qualitative state based modelling approach widely used in biology. We develop a semantic translation from MVNs to a corresponding RL model and formally show that this translation is correct. We consider both the asynchronous and synchronous update semantics, and investigate the use of rewriting strategies to enable synchronisation to be modelled. We illustrate the RL framework developed and the potential RL analysis possible by presenting two detailed case studies.

Keywords: rewriting logic, qualitative modelling, genetic regulatory networks, multi-valued networks

1. Introduction

Logical modelling techniques play an increasingly important role in biology where they are used to qualitatively model complex biological control systems (see [1] for a recent survey). They are seen as a crucial approach for developing scalable formal methods for modelling and synthesizing biological systems. *Multi-valued networks (MVNs)* [2, 3] are an expressive qualitative modelling approach for biological systems (for example, see [3, 4, 5]). They extend the well–known *Boolean network* [6] approach by allowing the state of each control entity to be within a given range of discrete values. In an MVN the entities interact to regulate their state using either *synchronous updates* [7], where the state of all entities is updated simultaneously, or *asynchronous updates* [8], where entities update their state independently. Both update semantics are important and used widely in the literature; the synchronous semantics leads to simpler dynamic behaviour which aids analysis (thus historically it was favoured) while the asynchronous semantics is seen as providing a more realistic model since entities are allowed to react independently. While a range of interesting work has been done to support qualitative modelling there still exists a number of important challenges for which further techniques and tools are required (for example, merging models based on different formalisms; compositional model construction and analysis; multi–scale modelling techniques; and tools for synthetic biology).

Rewriting Logic (RL) [9] is an algebraic formalism that extends standard algebraic specification techniques by allowing the dynamic behaviour of systems to be modelled using rewrite rules. The idea in RL is to define the static state based aspects of a system using an equational specification and to then use rewrite rules to specify the dynamic, non–deterministic state transition behaviour of the system. RL provides rewrite strategies which are able to control the application of rewrite rules and so allow an RL specification to capture subtle aspects of the behaviour of a dynamic system. RL has been successfully applied to modelling a wide range of different systems (for example, see [10, 11, 12, 13, 14, 15]). One important motivation for using RL is the powerful support tools available for simulating and analysing RL models, such as *Maude* [16]; *Elan* [17]; and *Tom* [18].

In this paper we consider using RL as a framework for developing and analysing both synchronous and asynchronous MVN models. The key motivation here is to introduce a new range of algebraic tools for MVNs and to provide an extendable framework which can be used as the basis for the future development of tools and techniques for MVNs. We begin by introducing the basic definitions for MVNs, and the synchronous and asynchronous update semantics. We then develop semantic translations for both asynchronous and synchronous MVNs into a corresponding RL model which is based on deriving a set of rewrite rules to represent the entity state updates that can occur in an MVN. The key idea is to represent the next–state functions associated with entities in an MVN as Boolean equations. These equations can then be simplified and used to directly derive the required rewrite rules for the RL model. We show that in the case of the synchronous update semantics it is necessary to make use of *rewriting strategies* [19] to be able to cope with the synchronization of updates. We formally show that the semantical translations developed are correct by proving that the derived RL models are *sound* (each state transition possible in the RL model has a counter part in the MVN) and *complete* (every global state update in an MVN is captured by a rewriting step in the RL model).

To illustrate the RL modelling techniques introduced we present two detailed case studies. In the first case study we consider modelling and analysing the genetic regulatory network controlling the biosynthesis of tryptophan in the bacteria *Escherichia coli* [20, 21] using an asynchronous MVN derived from an existing logical model in the literature [22]. The second case study illustrates the techniques developed for synchronous MVNs by considering an existing synchronous MVN model for the genetic regulatory network controlling the lysis–lysogeny switch in the bacteriophage λ [23, 4]. Importantly, both case studies illustrate the interesting range of analysis possible using the Maude rewriting tool [16].

One closely related modelling framework to this work is *Pathway Logic (PL)* [14, 15], an existing RL framework for symbolically modelling and analysing signal transduction and metabolic pathways. It represents each biomolecule involved in a biological pathway using a term containing three components:

the name of the biomolecule; a modifier which indicates the state of the biomolecule (such as whether it is phoshorylatd or not); and the location of the biomolecule. Using the standard RL approach, the state of a cellular pathway is represented as a multi-set of biomolecule terms. Rewrite rules are used to capture the local changes that occur to biomolecules during the signal transduction steps and these rules then form the so called *Rule Knowledge Base* which specifies how a pathway can be traversed. The resulting pathway model can then be executed and analysed in Maude using the tool's model checking capabilities. A range of work has been done to develop interesting techniques and tools for PL (see [24] for an overview) and the end result is a powerful framework for engineering biological pathways.

The RL framework presented here for MVNs has a different focus to PL which is primarily used for signal transduction pathways. In particular, we set out to develop a formally verified correct RL framework that was specifically tailored to MVNs and which naturally represented them in RL. While it is possible to model asynchronous MVNs using PL it would require substantial work along the lines presented here in terms of an appropriate mapping and correctness proof. It is also important to note that since PL has an inherently asynchronous semantics it is not suited to modelling synchronous MVNs which further motivates our work.

The paper is organized as follows. In Section 2 we provide a brief introduction to RL and in Section 3 we introduce MVNs. In Section 4 we develop a semantical translation of an asynchronous MVN into a corresponding RL model and formally show its correctness. We conclude this section with a case study to illustrate the techniques developed and the analysis capabilities of RL. In Section 5 we extend the semantical translation to apply to synchronous MVNs and again illustrate the techniques developed with a detailed case study. Then in Section 6 we evaluate the performance of the RL framework we have developed using an artificial, scalable test model. Finally, in Section 7 we present some concluding remarks and consider future work.

2. Rewriting Logic

Rewriting logic (RL) [9] is an algebraic specification framework which is capable of modelling and analysing the behaviour of dynamic, concurrent systems. RL has been successfully used to model a wide range of different formalisms and systems, such as process algebras [10, 11], Petri nets [12, 13], and biological systems [14, 15]. For a detailed introduction to RL we recommend [9, 16].

In RL the static states of a system are described using a standard equational specification. The dynamic behaviour of the system is then modelled using rewrite rules which are able to capture the non–deterministic state transitions that occur in such systems. RL also provides rewrite strategies which are able to control the application of rewrite rules and so allow an RL model to capture subtle aspects of the behaviour of a dynamic system.

As an example of using RL, consider modelling the following simple dynamic system in which system states are multi–sets consisting of the symbols A, B, and C. The system's dynamic behaviour occurs by transforming symbols and can be summarised as follows: symbol A can dynamically change to B; if symbol B is present then symbol C can change to an A; and two occurrences of symbol B can be replaced by symbol C. An RL specification for this dynamic system is given below (based on the syntax of the *Maude* tool [16]).

```
mod EX1 is
  sorts Symbol State .
  subsort Symbol < State .

  ops A B C : -> Symbol .
  op __ : Symbol Symbol -> State [assoc comm].

  rl [rule1] : A => B .
  rl [rule2] : B C => B A .
  rl [rule3] : B B => C .
endm
```

The above RL specification introduces the sorts *Symbol* and *State* to represent symbols and multi–sets of symbols. Note that sort *Symbol* is declared to be a subsort *State*, and so every symbol can be veiwed as a singleton multi–set. Three constants are declared to represent the symbols A, B and C in the system and an implicit union operator __ : *State State* $\rightarrow$ *State* (where _ is used to denote an infix argument location [16]) is declared which we define to be associative and commutative (this can be done by adding appropriate equations or using appropriate flags within a rewriting tool as is done here).

Let $A\ C$ be a multi–set representing the initial state of the system. Then the following rewrite trace represents one possible evolution of the system:

```
A C => B C => B A => B B => C
```

An important motivation for using RL is the powerful support tools available (e.g. [16, 17, 18]). We have chosen to use *Maude* [16] in this work due to its range of analysis tools, such as a Linear Temporal Logic model checker [25], and meta–programming capabilities. As an example, we consider using Maude's built–in search command `search S =>+ P`, which allows us to check if a pattern term `P` can be reached by rewriting an initial ground term `S`. Note that here `=>+` indicates one or more rewrites must occur (alternatives include only one rewrite `=>1`, and rewrite to termination `=>!`). We can use this search command to investigate whether we can reach a state containing `C C` from an initial state `A A B A A`:

```
search A A B A A =>+ C C s:State .
```

This search returns true and we can view a corresponding witness rewrite trace. We can check that all four `A`'s are needed by executing the search

```
search A A B A =>+ C C s:State .
```

which returns false. We consider Maude's analysis tools in more detail in Section 4.3.

The meta–programming capabilities offered by Maude are invaluable as they allow the construction of rewriting strategies which can control how rewrite rules are applied. As an example, suppose we want to prioritise the application of `rule1` over the other two rules. Then we can construct a corresponding rewrite strategy `oneFirst` in Maude to do this as shown below.

```
ceq oneFirst(T) = if Step? :: Result4Tuple then getTerm(Step?)
    else (if Step2? :: ResultPair then getTerm(Step4?) else T  fi)  fi
    if Step? := metaXapply(upModule('EXABC,false),T,'rule1,none,0,unbounded,0)
       /\  Step2? := metaRewrite(upModule('EXABC,false),T,1) .
```

This strategy is implemented using a conditional equation and makes use of two metalevel operations: `metaXapply` which allows a specific rule to be applied (in this case `rule1`) to a term `T` ; and `metaRewrite` which allows a term `T` to be rewritten a given number of times using all rules in a module. Note the use of type checking to see whether the metalevel operations have been successfully applied, e.g. `Step? :: Result4Tuple` is used to check if `rule1` has been successfully applied. For full details of the notation used here and further strategy examples see the Maude manual [26].

3. Multi-Valued Networks

Multi-valued networks (MVNs) [2, 3] provide a logical framework for qualitatively modelling and then analysing control systems. They have been successfully applied to biological systems [3, 4, 27, 5] and circuit design [2, 28]. In this section we introduce the basic definitions for MVNs and provide an illustrative example.

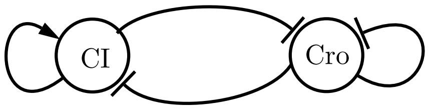
An MVN comprises a set of control entities each of which has a discrete state taken from a given set of states. The state of each entity is regulated by a subset of entities in the MVN and we refer to this subset as the neighbourhood of an entity (an entity may or may not be in its own neighbourhood). An entity updates its state by applying a logical next–state function to the current states of the entities in its neighbourhood. A formal definition of an MVN can be given as follows.

Definition 1. An MVN MV is a four-tuple $MV = (G, D, N, F)$ where:
i) $G = \{g_1, \ldots, g_n\}$ is a non-empty, finite set of entities;
ii) $D = (D(g_1), \ldots, D(g_n))$ is a tuple of state sets, where each $D(g_i) = \{0, \ldots, m_i\}$, for some $m_i \geq 1$, is the state space for entity g_i;
iii) $N = (N(g_1), \ldots, N(g_n))$ is a tuple of neighbourhoods, such that $N(g_i) \subseteq G$ is the neighbourhood of g_i; and
iv) $F = (f_{g_1}, \ldots, f_{g_n})$ is a tuple of next-state multi-valued functions, such that if $N(g_i) = \{g_{i_1}, \ldots, g_{i_n}\}$ then the function $f_{g_i} : D(g_{i_1}) \times \cdots \times D(g_{i_n}) \to D(g_i)$ defines the next state of g_i. □

To illustrate the above definition consider the example MVN *PL2* presented in Figure 1 which models the regulatory network underlying the *lysis–lysogeny switch* in the bacteriophage λ [3, 23, 29]. This MVN consists of two entities CI and Cro, defined with neighbourhoods $N(CI) = \{CI, Cro\}$ and $N(Cro) = \{CI, Cro\}$. These entities have the state spaces $D(CI) = \{0, 1\}$ and $D(Cro) = \{0, 1, 2\}$, and their next-state functions are defined by the state transition tables given in Figure 1.(b). In this MVN the entity Cro inhibits the expression of CI (i.e. acts to lower its state) and at higher levels of expression, also inhibits itself. Meanwhile, entity CI inhibits the expression of Cro and promotes its own expression (i.e. acts to increase its state).

In the sequel, let $MV = (G, D, N, F)$ be an arbitrary MVN. A *global state* of an MVN MV with n entities is represented by a tuple of states $(s_1, \ldots, s_n)$, where $s_i \in D(g_i)$ represents the state of entity g_i. The set of all global states, denoted S_{MV}, is then defined by $S_{MV} = D(g_1) \times \cdots \times D(g_n)$. As a notational convenience we normally write $s_1 \ldots s_n$ to represent a global state $(s_1, \ldots, s_n) \in S_{MV}$. 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 of an MVN can be updated *synchronously* [6, 7], where the state of all entities is

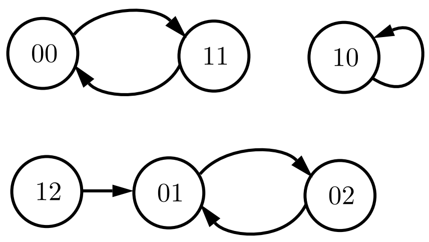


(a) Network structure

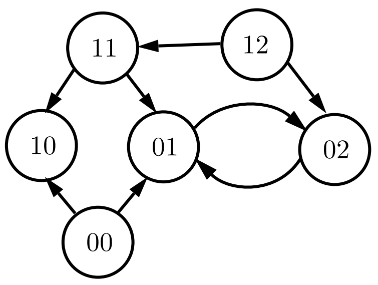
CI	*Cro*	*CI*
0	0	1
0	1	0
0	2	0
1	0	1
1	1	0
1	2	0

CI	*Cro*	*Cro*
0	0	1
0	1	2
0	2	1
1	0	0
1	1	0
1	2	1

(b) State transition tables



(c) Synchronous state graph



(d) Asynchronous state graph

Figure 1. An MVN model *PL2* of the regulatory network for the lysis-lysogeny switch in bacteriophage λ (based on [23]).

updated simultaneously in a single step, or *asynchronously* [30, 8], where entities update their state independently.

Definition 2. (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 S_1 by simultaneously updating the state of each entity g_i, for $i = 1, \ldots, n$, using its next-state function f_{g_i} and the appropriate states from S_1 as indicated by the neighbourhood $N(g_i)$. $\square$

Given a global state S_0 we can generate a *(synchronous) trace* by repeatedly applying the synchronous update step $S_0 \xrightarrow{Syn} S_1 \xrightarrow{Syn} S_2 \xrightarrow{Syn} \cdots$. Note in the sequel we will often represent such traces simply as a comma separated list of global states $S_0, S_1, S_2, \ldots$. Such traces are deterministic and so each possible initial state generates only a single trace. Since the set of global states is finite this means the set of synchronous traces is always finite. Each synchronous trace is itself infinite and will eventually enter an *attractor cycle* [31, 7]). For example, under the synchronous semantics *PL2* has three attractors: a point attractor $10, 10, \ldots$; and two cyclic attractors $00, 11, 00, \ldots$ and $01, 02, 01, \ldots$.

Definition 3. (Asynchronous Update) For any entity $g_i \in G$ 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 $next^{MV} : S_{MV} \to \mathcal{P}(S_{MV})$ on any state $S \in S_{MV}$ by

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

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

Note that given the above definition, only asynchronous update steps that result in a change in the current global state are considered (see [8]). 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 an infinite number of traces.

To illustrate the above, consider the global state 12 for *PL2* (see Figure 1) in which CI has state 1 and Cro has state 2. Then a single synchronous update step will result in the state 01. Applying the asynchronous update semantics, we have $next^{PL2}(12) = \{02, \; 11\}$ and so $12 \xrightarrow{Asy} 02$ and $12 \xrightarrow{Asy} 11$ are valid asynchronous update steps.

The state transition behaviour of an MVN can be represented by a *state graph* [32] in which the nodes of the graph are the global states and the edges are precisely the update steps allowed. The synchronous and asynchronous state graphs for *PL2* are presented in Figure 1.

Any state $S \in S_{MV}$ which cannot be asynchronously updated, i.e. $next^{MV}(S) = \{\}$, is a *point attractor*. The notion of a simple cyclic attractor is not valid in the asynchronous case and instead we consider *terminal connected components* in an MVN's asynchronous state graph to be *attractors* [32, 33]. For example, the MVN *PL2* has one point attractor 10 and one cyclic attractor $01, 02, 01, \ldots$.

In the example presented we defined the next–state function for each entity using a state transition table. An alternative approach is to specify the next–state function equationally [2, 28] by using Boolean terms called *literals* to represent when an entity g_i is in one of its given states. The literals have the form g_iS, for any $S \subseteq D(g_i)$, and evaluate to true when $g_i \in S$ and to false otherwise. The literals can be combined using conjunction resulting in *product terms* which represent possible states for a collection of entities. These can then be used to construct equations to represent the next–state function for an entity.

As an example, consider the entity CI in the MVN *PL2*. From the state transition table in Figure 1.(b) we can see that CI will have next state 1 when we are in state $CI = 0$, $Cro = 0$ or when we are in state $CI = 1$, $Cro = 0$. Therefore, the product term $CI\{0,1\}Cro\{0\}$ specifies when CI's next state will be 1. Using this approach we are able to completely specify equationally the next–state functions for the entities, as presented below:

$$\begin{array}{ll} CI\{0\} = CI\{0,1\}Cro\{1,2\} & Cro\{0\} = CI\{1\}Cro\{0,1\} \\ CI\{1\} = CI\{0,1\}Cro\{0\} & Cro\{1\} = CI\{0\}Cro\{0,2\} + CI\{1\}Cro\{2\} \\ & Cro\{2\} = CI\{0\}Cro\{1\} \end{array}$$

An important observation is that the Boolean terms derived above can normally be simplified using *multi-valued logic minimization techniques* [28] and we make use of this when modelling an MVN in the next section.

4. Modelling Asynchronous MVNs using RL

In this section we develop an RL model for MVNs based on their asynchronous update semantics. In particular, we present an approach for translating an MVN into an RL model and provide a formal correctness argument for this translation. We illustrate the techniques developed and the resulting analysis possible by presenting a detailed case study as the end of the section.

4.1. Constructing an RL Model for an Asynchronous MVN

Let $MV = (G, D, N, F)$ be an MVN and assume that we are using the asynchronous update semantics. Then we construct a corresponding RL model $RL_A(MV)$ as follows.

For each discrete state space $D(g) = \{0, \dots, m\}$ associated with some entity $g \in G$ define a corresponding sort D_m in $RL_A(MV)$ with constants $i : D_m$, for $i = 0, \dots, m$. We define the sort *Entity* in $RL_A(MV)$ for entities and represent each entity $g \in G$ in our MVN by a function $g : D_m \to$ *Entity*, where $D(g) = \{0, \dots, m\}$.

We represent global states in an MVN as non-empty multi–sets of entities. To do this we introduce the sort *GState*, defining *Entity* to be a subsort of *GState*, and then use the normal approach of adding an implicit union operator (see the example in Section 3). Note that given the above definitions not all terms of sort *GState* will correspond to well–defined global states of MV (they may omit an entity or contain multiple terms for an entity). For simplicity, we avoid restricting the generation of global state terms and instead introduce appropriate restrictions as part of our correctness argument (see Section 4.2 below).

As an example, consider the following signature generated for our running example *PL2* (see Figure 1) presented using the Maude syntax [16]:

```
mod PL2 is
  sorts Entity GState .
  sorts D1 D2 .
  subsort Entity < GState .

  ops 0 1 : -> D1 .
  ops 0 1 2 : -> D2 .
  op CI : D1 -> Entity .
  op Cro : D2 -> Entity .
  op __ : Entity Entity -> GState [assoc comm].
endm
```

Given the above definitions a global state in which CI has state 1 and Cro has state 2 would be represented by the term `CI(1) Cro(2)`.

The final stage in modelling an MVN is to represent within our RL model the next–state function associated with each entity and the associated asynchronous update rule. Note that in doing this we will implicitly capture the neighbourhood information associated with the MVN. In order to do this we begin by deriving a equational representation for the next–state functions as described in Section 3. We then simplify the resulting equations using *multi–valued logic minimization techniques* [28] (note that this process can be automated using tools such as *MVSIS* [34]). Each of the simplified equations will have the general form

$$g\{s\} = P_1 + \cdots + P_k$$

for some $k > 0$, $g \in G$ and $s \in D(g)$. The equation specifies that g will have next state s precisely when one or more of the product terms $P_1, \ldots, P_k$ is true. We generate the rewrite rules for asynchronously updating g to state s by deriving a set of rewrite rules for each product term P_i, $i = 1, \ldots, k$, as follows.

First we need to ensure that a term representing the current state of g occurs within P_i. To do this we check P_i and if no term of the form gS' is present, for some $S' \subseteq D(g)$, then we use conjunction to add the term $g(D(g) - \{s\})$ to P_i (in a slight abuse of notation we let P_i still denote this new term). Now suppose

$$P_i = g_{E_1}S_1 \ \ldots \ g_{E_m}S_m \ gS_{m+1}$$

for some $m \geq 0$, and state sets $S_j \subseteq D(g_{E_j})$, $1 \leq j \leq m$ and $S_{m+1} \subseteq D(g)$. Then we add the following rewrite rules to $RL_A(MV)$:

$$g_{E_1}(u_1) \ \ldots \ g_{E_m}(u_m) \ g(u_{m+1}) \Longrightarrow g_{E_1}(u_1) \ \ldots \ g_{E_m}(u_m) \ \ g(s)$$

for each $u_j \in S_j$, $1 \leq j \leq m+1$, such that $u_{m+1} \neq s$. Note the final restriction $u_{m+1} \neq s$ is needed here to ensure that only state transitions which change the current state of entity g are allowed (this is a fundamental part of the asynchronous update semantics).

To illustrate the above process consider applying it to our running example *PL2*. First we derive the equations for *PL2* (see Section 3). We then apply multi–valued logic minimization to simplify these to the following set of equations:

$$\begin{array}{lll}
\mathit{CI}\{0\} = \mathit{Cro}\{1,2\} & \mathit{Cro}\{0\} = \mathit{CI}\{1\}\mathit{Cro}\{0,1\} & \mathit{Cro}\{2\} = \mathit{CI}\{0\}\mathit{Cro}\{1\} \\
\mathit{CI}\{1\} = \mathit{Cro}\{0\} & \mathit{Cro}\{1\} = \mathit{CI}\{0\}\mathit{Cro}\{0\} + \mathit{Cro}\{2\} &
\end{array}$$

The final stage is to use the simplified equations to derive a set of rewrite rules to model the asynchronous update of *PL2*. To illustrate the approach defined above consider deriving the rewrite rules for *Cro* entering state 0. From the equations above we see that *Cro* has next state 0 whenever *CI* is in state 1 and *Cro* is in state 0 or 1. However, a state change occurs here only if *Cro* is currently in state 1 so we need only one rewrite rule to model this state transition:

$$\mathit{CI}(1) \ \mathit{Cro}(1) \Longrightarrow \mathit{CI}(1) \ \mathit{Cro}(0)$$

The full set of rules derived to model the asynchronous next–state functions for *PL2* is given below:

```
rl [CI0] : CI(1) Cro(1) => CI(0) Cro(1) .
rl [CI0] : CI(1) Cro(2) => CI(0) Cro(2) .
rl [CI1] : CI(0) Cro(0) => CI(1) Cro(0) .

rl [Cro0] : CI(1) Cro(1) => CI(1) Cro(0) .
rl [Cro1] : CI(0) Cro(0) => CI(0) Cro(1) .
rl [Cro1] : Cro(2) => Cro(1) .
rl [Cro2] : CI(0) Cro(1) => CI(0) Cro(2) .
```

A prototype tool has been developed that given an XML description of an MVN will automatically apply the above translation process (this tool is available from the authors on request).

4.2. RL Model Correctness

In this section we formally show that our translation from an asynchronous MVN to an RL model given in the previous section is correct. To do this we show that: 1) (*soundness*) each global state transition possible in our RL model represents a corresponding asynchronous update in the original MVN; and 2) (*completeness*) every asynchronous update possible in an MVN is specified in our RL model.

We begin by precisely defining which terms in our RL model represent well–defined global states. Let MV be an MVN with entities $G = \{g_1, \dots, g_n\}$, for some $n > 0$. Then we define the set $validGS(MV)$ of RL terms of sort *GState* representing well–defined global states in MV by

$$validGS(MV) = \{g_1(s_1)\ \dots\ g_n(s_n) \mid s_1 \in D(g_1), \dots, s_n \in D(g_n)\}$$

The following result shows that rewrite steps in our RL model preserve valid global state terms.

Theorem 4. For any $GS_1 \in validGS(MV)$, if $GS_1 \Longrightarrow GS_2$ in one rewrite step in $RL_A(MV)$ then $GS_2 \in validGS(MV)$.

Proof. Let $GS_1 = g_1(s_1)\ \dots\ g_n(s_n) \in validGS(MV)$, for some $s_i \in D(g_i)$, $1 \leq i \leq n$. Suppose that $GS_1 \Longrightarrow GS_2$ in one rewrite step in $RL_A(MV)$. Then by the definition of the rewrite rules in $RL_A(MV)$ we know that no term representing the state of an entity is added or removed from a global state term when a rewrite rule is applied. Noting that the multi–set union operator is associative and commutative, it therefore follows that GS_2 must have the form $GS_2 = g_1(s'_1)\ \dots\ g_n(s'_n)$, where for some $j \in \{1, \dots, n\}$ we have $s'_i = s_i$, for $i = 1, \dots, n$, $i \neq j$, and $s'_j \in D(g_j)$ such that $s'_j \neq s_j$. Then by definition we have GS_2 is a valid global state term (i.e. $GS_2 \in validGS(MV)$). □

In order to formally map global states of an MVN MV to corresponding global state terms in $RL_A(MV)$ we define a global state term mapping $\phi : D(g_1) \times \dots \times D(g_n) \to validGS(MV)$ by

$$\phi(s_1 \dots s_n) = g_1(s_1)\ \dots\ g_n(s_n),$$

for any states $s_i \in D(g_i)$, $1 \leq i \leq n$. Note that since ϕ can be shown to be a bijective mapping it has an inverse $\phi^{-1} : validGS(MV) \to (D(g_1) \times \dots \times D(g_n))$.

We can now show that $RL_A(MV)$ is *sound* and *complete* with respect to MV.

Theorem 5. (Soundness) Let $GS_1, GS_2 \in validGS(MV)$ be any valid global state terms such that $GS_1 \Longrightarrow GS_2$ in a single rewrite step in $RL_A(MV)$. Then $\phi^{-1}(GS_1) \xrightarrow{Asy} \phi^{-1}(GS_2)$.

Proof. Suppose that $GS_1 \Longrightarrow GS_2$ in a single rewrite step in $RL_A(MV)$. Then a rewrite rule with the following form must have been applied:

$$g_{E_1}(s_1)\ \dots\ g_{E_m}(s_m)\ g(s) \Longrightarrow g_{E_1}(s_1)\ \dots\ g_{E_m}(s_m)\ \ g(s')$$

for some $m \in \mathbf{N}$, distinct $g_{E_1}, \dots, g_{E_m}, g \in G$, $s_i \in D(g_{E_i})$, $i = 1, \dots, m$, and $s, s' \in D(g)$ such that $s \neq s'$. By the definition of $RL_A(MV)$ this rule was derived from an equation of the form $g\{s'\} = P_1 + \dots + P_k$, for some $k > 0$ and product terms P_i, $i = 1, \dots, k$. In particular, the rewrite rule was derived from one of the product terms, say P_i, for some $i \in \{1, \dots, k\}$. It is therefore clear that product term P_i must be true in global state $\phi^{-1}(GS_1)$ and so the state of entity g can be asynchronously updated to state s' (as specified by the equation). In other words, we have that $\phi^{-1}(GS_1) \xrightarrow{Asy} \phi^{-1}(GS_2)$ as required. □

Theorem 6. (Completeness) Let $S_1, S_2 \in D(g_1) \times \cdots \times D(g_n)$ be global states in MV such that $S_1 \xrightarrow{Asy} S_2$. Then $\phi(S_1) \Longrightarrow \phi(S_2)$ in a single rewrite step in $RL_A(MV)$.

Proof. Suppose that the asynchronous update $S_1 \xrightarrow{Asy} S_2$ can occur in MV. Then this means the global state S_2 is produced by updating the state of one entity in S_1 while keeping the states of all other entities the same. Suppose that the entity which has been updated is g, for some $g \in G$ and that its state has changed from s to s', for some states $s, s' \in D(g)$ such that $s \neq s'$. Then there must exist an equation of the form $g\{s'\} = P_1 + \cdots + P_k$, for some $k > 0$ and product terms P_i, $i = 1, \ldots, k$. Furthermore, at least one of the product terms P_i, for some $i \in \{1, \ldots, k\}$, must evaluate to true in the global state S_1. It follows by the definition of $RL_A(MV)$ that there must exist a rewrite rule derived from the above equation and in particular, derived from the product term P_i which has the form:

$$g_{E_1}(s_1) \ \ldots \ g_{E_m}(s_m) \ g(s) \Longrightarrow g_{E_1}(s_1) \ \ldots \ g_{E_m}(s_m) \ \ g(s')$$

for some $m \in \mathbf{N}$, distinct $g_{E_1}, \ldots, g_{E_m} \in G$, $s_i \in D(g_{E_i})$, $i = 1, \ldots, m$. Clearly, given the assumptions above we must be able to apply this rewrite rule to $\phi(S_1)$ and the resulting state term will correspond to $\phi(S_2)$. In other words, we have that $\phi(S_1) \Longrightarrow \phi(S_2)$ in a single rewrite step in $RL_A(MV)$ as required. □

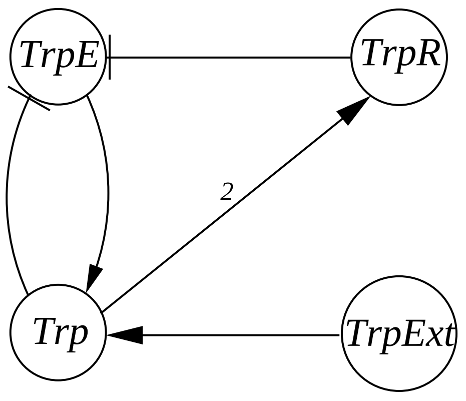
4.3. Case Study: Tryptophan Biosynthesis

We illustrate the RL framework developed above by presenting a case study based on modelling and analysing the genetic regulatory network for the synthesis of tryptophan in *E. coli* [20, 21]. This case study helps to motivate our RL framework by illustrating the analysis techniques and flexibility available when using RL and the tool *Maude* [16].

The biosynthesis of the amino acid tryptophan in *E. coli* is carefully regulated since it is essential for the growth of the bacteria but is also costly to produce. For this reason tryptophan biosynthesis only occurs when no external source of tryptophan is available [20, 21]. An MVN for the underlying genetic regulatory network for the biosynthesis of tryptophan in *E. coli* (based on [22]) is presented in Figure 2. The MVN consists of four regulatory entities:

TrpE – indicates the presence of the activated enzyme required for synthesising tryptophan, has neighbourhood $N(\mathit{TrpE}) = \{\mathit{TrpR}, \mathit{Trp}\}$ and state space $D(\mathit{TrpE}) = \{0, 1\}$;
TrpR – indicates if the repressor gene for tryptophan production is active, has neighbourhood $N(\mathit{TrpR}) = \{\mathit{Trp}\}$ and state space $D(\mathit{TrpR}) = \{0, 1\}$;
TrpExt – an input entity indicating the level of external tryptophan, has $D(\mathit{TrpExt}) = \{0, 1, 2\}$;
Trp – indicates the level of tryptophan within the bacteria, has neighbourhood $N(\mathit{Trp}) = \{\mathit{TrpExt}, \mathit{TrpE}\}$ and state space $D(\mathit{Trp}) = \{0, 1, 2\}$.

We use the above entity order when presenting global states for *MTRP*.



TrpE	*TrpExt*	*Trp*	[*Trp*]
0	0	0,1	0
0	0	2	1
0	1	0,1,2	1
0	2	0	1
0	2	1,2	2
1	0,1	0,1,2	1
1	2	0	1
1	2	1,2	2

Trp	[*TrpR*]
0,1	0
2	1

Trp	*TrpR*	[*TrpE*]
0	0	1
0	1	0
1,2	0,1	0

Figure 2. An MVN model *MTRP* of the regulatory mechanism for the biosynthesis of tryptophan in *E. coli* (based on [22]). The state transition table for *TrpExt* has been omitted as this is a simple input entity.

The regulatory network works as follows: the activated enzyme *TrpE* is required to synthesise tryptophan but this enzyme is deactivated by the presence of tryptothan within *E. coli* and at higher-levels of tryptophan the production of *TrpE* is inhibited by the activation of the repressor *TrpR*. The asynchronous state graph for *MTRP* consists of 36 global states and has the following three attractors: two point attractors 0011 and 0122 which occur in the presence of external tryptophan; and a cyclic attractor $0000, 1000, 1001, 0001, 0000$ representing tryptophan synthesis.

Following the approach defined in Section 4.1 we can construct an RL model $RL_A(MTRP)$ for *MTRP*. We begin by deriving equations for the next–state functions of *MTRP* from the state transition tables in Figure 2. Simplifying these results in the equations below:

$$\begin{array}{ll} TrpE\{0\} = TrpR\{1\} + Trp\{1,2\}TrpR\{0\}, & TrpR\{0\} = Trp\{0,1\}, \\ TrpE\{1\} = Trp\{0\}TrpR\{0\}, & TrpR\{1\} = Trp\{2\}, \\ Trp\{0\} = TrpE\{0\}TrpExt\{0\}Trp\{0,1\}, & Trp\{2\} = TrpExt\{2\}Trp\{1,2\} \end{array}$$
$$Trp\{1\} = TrpE\{0\}TrpExt\{0\}Trp\{2\} + TrpExt\{1\} + TrpExt\{2\}Trp\{0\} + TrpE\{1\}TrpExt\{0\},$$

These equations are then used to derive the set of rewrite rules for $RL_A(MTRP)$ to model the asynchronous behaviour of *MTRP* as described in Section 4.1. We present the rewrite rules derived for *Trp* below as an example (the remaining rules are omitted for brevity).

```
rl [Trp0] : TrpE(0) TrpExt(0) Trp(1) => TrpE(0) TrpExt(0) Trp(0) .
rl [Trp1] : TrpE(0) TrpExt(0) Trp(2) => TrpE(0) TrpExt(0) Trp(1) .
rl [Trp1] : TrpExt(1) Trp(0) => TrpExt(1) Trp(1) .
rl [Trp1] : TrpExt(1) Trp(2) => TrpExt(1) Trp(1) .
rl [Trp1] : TrpE(1) TrpExt(0) Trp(0) => TrpE(1) TrpExt(0) Trp(1) .
rl [Trp1] : TrpE(1) TrpExt(0) Trp(2) => TrpE(1) TrpExt(0) Trp(1) .
rl [Trp1] : TrpExt(2) Trp(0) => TrpExt(2) Trp(1) .
rl [Trp2] : TrpExt(2) Trp(1) => TrpExt(2) Trp(2) .
```

Once a model has been developed for a biological system then the next stage is to analyse its behaviour. The idea is to validate the model by checking that it has known biological properties and to produce important new insights that can then be experimentally investigated by biologists. We illustrate the wide range of analysis possible using our RL framework and the support tool Maude by providing a selection of analysis examples for $RL_A(\mathit{MTRP})$.

We begin by considering Maude's built-in command `search` [16] which provides interesting ways to search the states reachable from a given initial state. For example, the following search allows the attractors in our model to be investigated by checking whether an initial state leads to a point attractor (note the use of `=>!` here to ensure the rewriting terminates).

```
search TrpE(1) TrpR(0) TrpExt(2) Trp(2) =>! GS:GState .
```

This command confirms that 1022 leads to the point attractor 0122 in *MTRP* and we can view the associated trace for this behaviour. A similar search shows that the initial state 1102 does not lead to a point attractor. We can also check general invariant properties on the state space reachable from a given initial state as the example below illustrates.

```
search TrpE(1) TrpR(1) TrpExt(1) Trp(0) =>+ Trp(2) GS:GState .
```

This search confirms that *Trp* can not reach state 2 from the initial state 1110. A similar search shows that *Trp* can reach state 2 from initial state 0120 (in fact, there are four different traces that result in this behaviour and we can view these). As a final example, consider the following search which confirms that *TrpE* and *TrpR* are not mutually exclusive from the initial state 0120.

```
search TrpE(0) TrpR(1) TrpExt(2) Trp(0) =>+ TrpE(1) TrpR(1) GS:GState .
```

Maude indicates there is a single counter example trace here and we able to view it to gain insight into this behaviour. A similar search confirms that *TrpE* and *TrpR* are mutually exclusive from 0121.

Maude also provides a model checking tool for *Linear Temporal Logic (LTL)* [35] which allows dynamic properties not checkable with the search command to be analysed for finite state rewrite systems [25]. Since any MVN has a finite state space we can use this model checking tool to investigate a wide range of biologically relevant dynamic properties of an MVN. To use the model checker we define a range of atomic propositions to represent the key properties of interest. For example, for *MTRP* we might define an atomic proposition `atTrp:D2 -> Prop`, where `atTrp(s)` is true only if *Trp* is in state `s`. This can be done equationally as follows (where `[owise]` is a Maude shortcut which allows all remaining possibilities to be covered):

```
   eq Trp(s2) GS |= atTrp(s2) = true .
   eq GS |= atTrp(s2) = false [owise] .
```

We can then model check a range of interesting dynamic properties expressed using the temporal operators of LTL as illustrated by the examples below for *MTRP*.

The first property we consider is used to validate the model by showing that tryptophan is always eventually present in the bacteria. We represent this property in Maude using the LTL formula `[] <> (atTrp(1) \/ atTrp(2))`, where `[]` stands for the always operator and `<>` for eventually [35]. We can check this LTL formula for the initial state 0000 using the following Maude command:

```
red modelCheck(TrpE(0) TrpR(0) TrpExt(0) Trp(0), [] <> (atTrp(1) \/ atTrp(2)))
```

This returns true showing that the property holds and indeed it holds for all initial states considered.

Another property we might consider is whether from a given initial state an entity eventually becomes stable in a given state. For example, suppose we want to check that from initial state 0020 that *Trp* must eventually become fixed in state 2 (i.e. present at high–levels in the bacteria). We can confirm this property holds using the command:

```
red modelCheck(TrpE(0) TrpR(0) TrpExt(2) Trp(0), <> [] atTrp(2))
```

We can extend this property further to check whether a high–level of external tryptophan is enough to ensure a high–level of tryptophan in the bacteria. We represent this property using the formula `atExt(2) -> (<> [] atTrp(2))`, and model checking shows it holds for all initial states sampled.

As a final example, suppose we want to check whether an entity entering a specific state can trigger some important behaviour. For example, the following LTL formula `[] (atTrp(2) -> <> atR(1))`, captures the property that if *Trp* is ever fully expressed then eventually *TrpR* must become active. We can model check this LTL formula for some intial state, say 1002, using the following Maude command:

```
red modelCheck(TrpE(1) TrpR(0) TrpExt(0) Trp(2), [] (atTrp(2) -> <> atR(1)))
```

This returns false showing that the formula does not hold and provides a counter example trace which gives important insight into the result. Repeating the above check we are able to confirm that the property does hold for initial state 1020.

5. Modelling Synchronous MVNs using RL

In this section we build on the results of the previous section by developing an RL model for synchronous MVNs. The challenge here is to be able to coordinate update steps and we make use of Maude's metalevel capabilities to achieve this. We again show that the resulting model construction is formally correct and illustrate the developed RL framework for synchronous MVNs with a case study.

5.1. Constructing an RL Model for a Synchronous MVN

Modelling the synchronous update semantics for an MVN in RL follows along similar lines to the asynchronous approach in Section 4.1. However, it is more complex since we have to ensure that all entities update their state simultaneously when moving from one global state to another. For this reason we use a two phase update approach [36] for computing global next states: 1) compute and record the next state of each entity while preserving their current states; and 2) update the current states of all entities to reflect the recorded next states. Given an MVN $MV = (G, D, N, F)$ we let $RL_S(MV)$ represent the RL model resulting from the construction outlined below for the synchronous update semantics of MV.

To model the first phase of the update step we take the basic sort and function definitions given in Section 4.1 and adapt the term representation of entities so that they contain a second state component to represent the recorded next state. For each entity $g \in G$ in the MVN we define the function $g : D_m\ D_m \to$ *Entity*, where $D(g) = \{0, \ldots, m\}$. The idea is that $g(s_1, s_2)$ represents that g is currently in state s_1 and will have next state s_2. We will use the convention that before a synchronous update takes place each entity's current and next state are the same. For example, the global state 12 in the example MVN *PL2* (see Section 3) would be represented by the term `CI(1,1) Cro(2,2)`. Thus the current and next states of an entity will only differ when we are partly through the synchronous update step.

We formalize the next–state function associated with each entity using a similar approach to that detailed in Section 4.1 for the asynchronous model. The difference here is that we reformulate the resulting rewrite rules so that the current state is not changed at this stage and instead the next state of an entity is simply recorded. To illustrate this consider the rewrite rule

```
rl [CI0] : CI(1) Cro(1) => CI(0) Cro(1) .
```

derived previously to represent an asynchronous update of *CI* from state 1 to 0 in *PL2*. This rule is replaced by the following one in the synchronous case:

```
rl [CI0] : CI(1,1) Cro(1,s2) => CI(1,0) Cro(1,s2) .
```

The use of variable `s2` is needed since the next state of *Cro* may or may not have been updated at this stage. Note that this rule can only be applied if the current and next state of *CI* are the same. This is important as it ensures the rule can only be applied at most once during a given update step and links to our assumption above about the representation of global states.

The full set of rewrite rules used to model the synchronous case for *PL2* is presented below:

```
rl [CI0] : CI(1,1) Cro(1,s2) => CI(1,0) Cro(1,s2) .
rl [CI0] : CI(1,1) Cro(2,s2) => CI(1,0) Cro(2,s2) .
rl [CI1] : CI(0,0) Cro(0,s2) => CI(0,1) Cro(0,s2) .
rl [Cro0] : CI(1,s1) Cro(1,1) => CI(1,s1) Cro(1,0) .
rl [Cro1] : CI(0,s1) Cro(0,0) => CI(0,s1) Cro(0,1) .
rl [Cro1] : Cro(2,2) => Cro(2,1) .
rl [Cro2] : CI(0,s1) Cro(1,1) => CI(0,s1) Cro(1,2) .
```

The idea is that rewriting using the resulting set of rewrite rules will update the next states of entities and that when rewriting terminates the first phase of the two phase synchronous update will be complete. To model the second, synchronization phase of the update we need to replaces each current state by its recorded next state. In order to do this we define an update function `upDate:GState -> GState` recursively using the following equations (where `gs` and `gs2` are variables of sort `GState`):

```
eq upDate(CI(s1,s11)) = CI(s11,s11) .
eq upDate(Cro(s2,s22)) = Cro(s22,s22) .
eq upDate(gs gs2) = upDate(gs) upDate(gs2) .
```

We now have all the functionality required to implement the two phase synchronous state update and what is now needed is a way to combine them correctly. We do this by making use of Maude's metalevel capabilities [16] to define an operator to capture the required rewriting strategy. When working at the metalevel global states will be represented using Maude's meta–notation and are given the meta–type `Term`. For example, the metalevel term `'__['CI['0.D1,'0.D1],'Cro['1.D2,'1.D2]]` will be used to represent the global state term `CI(0,0) Cro(1,1)` (Note that Maude provides the operator `upTerm` to lift a term to the metalevel.)

The rewriting strategy we require for synchronous updates is defined in two parts. First, we define a metalevel operation `phaseOne : Term -> Term` which implements the first phase of the global state update by applying the synchronous rewrite rules (in this case assumed to be in module `EXPL2`) to a global state term `T` using `metaRewrite` (see Section 2).

```
ceq phaseOne(T) = if Step?::ResultPair then getTerm(Step?) else T   fi
      if Step? := metaRewrite(upModule('EXPL2, false), T, unbounded) .
```

We then build on this by defining a metalevel operation `next : Term -> Term` which applies `phaseOne` to a term and then resets the current state using the `upDate` function.

```
ceq next(T) = if Step?::ResultPair then getTerm(Step?) else T1 fi
      if T1 := phaseOne(T) /\
         Step? := metaReduce(upModule('EXPL2, false), 'upDate[T1]) .
```

Note that in the above we use the metalevel representation of `upDate` as indicated by the backquote and that `metaReduce` is used to apply its defining equations. We can now use the metalevel operator `next` to simulate synchronous update steps. For example, the synchronous step $12 \xrightarrow{Syn} 01$ in *PL2* can be reproduced by the following Maude command:

```
red next('__['CI['1.D1,'1.D1],'Cro['2.D2,'2.D2]]) .
```

which correctly returns the metalevel state term `'__['CI['0.D1,'0.D1],'Cro['1.D2,'1.D2]]`.

The above process is supported by a prototype tool (this tool is available from the authors on request).

5.2. RL Model Correctness

We now show that the RL model proposed above for an MVN using the synchronous update semantics is correct by following a similar approach to that used in Section 4.2 for the asynchronous case. We begin by precisely defining which terms in our RL model represent well–defined global states in the synchronous case:

$$validGS(MV) = \{g_1(s_1, s_1) \ \ldots \ g_n(s_n, s_n) \mid s_1 \in D(g_1), \ldots, s_n \in D(g_n)\}$$

The following result shows that the metalevel operator `next` preserves valid global state terms. Note that to apply `next` we need to move to and from the metalevel representation of state terms.

Theorem 7. For any $GS_1 \in validGS(MV)$, if $GS_1 \Longrightarrow GS_2$ in one application of `next` in $RL_S(MV)$ then $GS_2 \in validGS(MV)$.

Proof. Let $GS_1 = g_1(s_1, s_1)\ \ldots\ g_n(s_n, s_n) \in \mathit{validGS}(MV)$, for some $s_i \in D(g_i)$, $1 \leq i \leq n$. Suppose that $GS_1 \Longrightarrow GS_2$ in one application of `next` in $RL_S(MV)$. By the definition of `phaseOne` and the synchronous rewrite rules in $RL_S(MV)$ we know that no term representing the state of an entity is added or removed from a global state term. It follows that after applying `phaseOne` to GS_1 we must have a state term of the form $g_1(s_1, s'_1)\ \ldots\ g_n(s_n, s'_n)$, for some $s'_i \in D(g_i)$, $1 \leq i \leq n$. Applying the reset function `upDate` to this state term will result in the state term $GS_2 = g_1(s'_1, s'_1)\ \ldots\ g_n(s'_n, s'_n)$ which is clearly a valid global state term as defined above. □

We define a global state term mapping $\phi : D(g_1) \times \cdots \times D(g_n) \to \mathit{validGS}(MV)$ by

$$\phi(s_1 \ldots s_n) = g_1(s_1, s_1)\ \ldots\ g_n(s_n, s_n),$$

for any states $s_i \in D(g_i)$, $1 \leq i \leq n$.

The following results show that $RL_S(MV)$ is a correct (i.e. *sound* and *complete*) model of MV under the synchronous update semantics.

Theorem 8. (Soundness) Let $GS_1, GS_2 \in \mathit{validGS}(MV)$ be any valid global state terms such that $GS_1 \Longrightarrow GS_2$ in one application of `next` in $RL_S(MV)$. Then $\phi^{-1}(GS_1) \xrightarrow{Syn} \phi^{-1}(GS_2)$.

Proof. Let $GS_1 = g_1(s_1, s_1)\ \ldots\ g_n(s_n, s_n)$, $GS_2 = g_1(s'_1, s'_1)\ \ldots\ g_n(s'_n, s'_n) \in \mathit{validGS}(MV)$, for some $s_i, s'_i \in D(g_i)$, $1 \leq i \leq n$. Suppose that $GS_1 \Longrightarrow GS_2$ in one application of `next` in $RL_S(MV)$. Then by the definition of ϕ we need to show that $s_1 \ldots s_n \xrightarrow{Syn} s'_1 \ldots s'_n$.

For each $i = 1, \ldots, n$ there are two possible cases to consider:

Case 1: Suppose $s_i \neq s'_i$. Then a rewrite rule with the following form must have been applied:

$$g_{E_1}(s_{E_1}, v_1)\ \ldots\ g_{E_m}(s_{E_m}, v_m)\ g_i(s_i, s_i) \Longrightarrow g_{E_1}(s_{E_1}, v_1)\ \ldots\ g_{E_m}(s_{E_m}, v_m)\ \ g_i(s_i, s'_i)$$

for some $m \in \mathbf{N}$, distinct entities $g_{E_1}, \ldots, g_{E_m} \in G$ and state variables $v_1, \ldots, v_m$. By the definition of $RL_S(MV)$ this rule was derived from an equation of the form $g_i\{s'_i\} = P_1 + \cdots + P_k$, for some $k > 0$ and product terms P_j, $j = 1, \ldots, k$. It is therefore clear that the right hand side of the equation must be true in the global state $s_1 \ldots s_n$ and so by definition the state of entity g_i must be updated to s'_i after a synchronous update is applied.

Case 2: Suppose $s_i = s'_i$. Then the state of entity g_i is not changed meaning that none of the rewrite rules updating g_i were applicable in GS_1. Since by the definition the rewrite rules of $RL_S(MV)$ are derived from the next state equations of MV this implies that the only equation applicable for g_i was of the form $g_i\{s_i\} = P_1 + \cdots + P_k$. Therefore, it follows that the state of g_i remains unchanged after a synchronous update step is applied to $s_1 \ldots s_n$. □

Theorem 9. (Completeness) Let $S_1, S_2 \in D(g_1) \times \cdots \times D(g_n)$ be global states in MV such that $S_1 \xrightarrow{Syn} S_2$. Then $\phi(S_1) \Longrightarrow \phi(S_2)$ in one application of `next` in $RL_S(MV)$.

Proof. Let $S_1 = s_1 \ldots s_n$, $S_2 = s'_1 \ldots s'_n \in D(g_1) \times \cdots \times D(g_n)$ and suppose that the synchronous update step $s_1 \dots s_n \xrightarrow{Syn} s'_1 \dots s'_n$ can occur. Then for each $i = 1, \dots, n$ there are two cases to consider:

Case 1: Suppose $s_i \neq s'_i$. Then the state of g_i changes from s_i to s'_i during the synchronous step and so the next–state equation applicable for g_i must have been of the form $g\{s'_i\} = P_1 + \cdots + P_k$, for some $k > 0$ and product terms P_j, $j = 1, \dots, k$. In particular, at least one of the product terms, say P_j, for some $j \in \{1, \dots, k\}$, must be true. Then by definition of $RL_S(MV)$ there must be a rewrite rule

$$g_{E_1}(s_{E_1}, v_1) \ \dots \ g_{E_m}(s_{E_m}, v_m) \ g_i(s_i, s_i) \Longrightarrow g_{E_1}(s_{E_1}, v_1) \ \dots \ g_{E_m}(s_{E_m}, v_m) \ \ g_i(s_i, s'_i)$$

that was derived from the equation above based on P_j. Clearly, this rewrite rule will be applicable in the global state term $\phi(S_1)$ and so by definition of `next` the entity term $g_i(s_i, s_i)$ must be updated to $g_i(s'_i, s'_i)$ as required.

Case 2: Suppose $s_i = s'_i$. Then the state of entity g_i is not changed in the update step meaning that the only next–state equation applicable for g_i was of the form $g_i\{s_i\} = P_1 + \cdots + P_k$. Therefore, it follows by definition that no rewrite rule exists in $RL_S(MV)$ that can be applied to the global state term $\phi(S_1)$ and so the entity term $g_i(s_i, s_i)$ remains unchanged from $\phi(S_1)$ to $\phi(S_2)$ as required. □

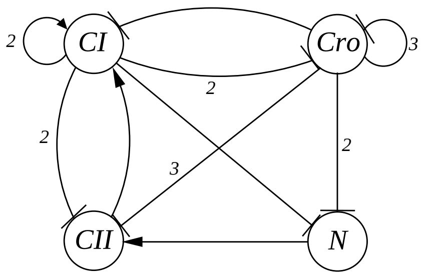
5.3. Case Study: Lysis–Lysogeny Switch

In this section we illustrate the RL framework developed for synchronous MVNs by considering an existing MVN model for the genetic regulatory network controlling the lysis–lysogeny switch in the bacteriophage λ [23, 4]. The bacteriophage λ is a virus which makes an interesting choice after infecting the bacteria *Escherichia coli* between two different ways of propagating itself [3, 29]. In most cases, λ enters the *lytic cycle*, where it generates as many new viral particles as the host cell resources allow before producing an enzyme to lyse the cell wall, releasing the new phage into the environment. However, it can decide to enter the *lysogenic cycle* where it integrates its DNA into the host's DNA and then lies dormant simply replicating with each subsequent cell division of the host. In this case the host cell becomes immune to external infection from phages since the genes expressed in the λ DNA synthesize a repressor blocking other phage genes (including its own excision genes).

An MVN *PL4* of the resulting regulatory network underlying the lysis–lysogeny switch is presented in Figure 3. This MVN extends *PL2* [23] and contains four entities: N, with $D(N) = \{0, 1\}$, which promotes *CII* expression; *CII*, with $D(\mathit{CII}) = \{0, 1\}$, which activates *CI*; *CI*, with $D(\mathit{CI}) = \{0, \dots, 2\}$, a repressor which is expressed in the lysogenic cycle; and *Cro*, with $D(\mathit{Cro}) = \{0, \dots, 3\}$, a repressor present in the lytic cycle. This MVN has a global state space consisting of 48 states and has two attractor cycles: $0003, 0002, 0003, \dots$ which corresponds to the lytic cycle; and $0020, 0020, \dots$ which corresponds to the lysogenic cycle.

Applying our RL translation approach detailed above we begin by deriving the following simplified equations for *PL4* from the truth tables given in Figure 3:

$$\begin{array}{ll}
[N\{0\}] = \mathit{CI}\{0\}\mathit{Cro}\{2,3\} \ + \ \mathit{CI}\{1,2\}, & [\mathit{CII}\{0\}] = N\{0\} + \mathit{Cro}\{3\} + \mathit{CI}\{2\}, \\
[N\{1\}] = \mathit{CI}\{0\}\mathit{Cro}\{0,1\}, & [\mathit{CII}\{1\}] = N\{1\}\mathit{CI}\{0,1\}\mathit{Cro}\{0,1,2\}, \\
[\mathit{CI}\{0\}] = \mathit{CII}\{0\}\mathit{CI}\{0,1\}\mathit{Cro}\{1,2,3\}, & [\mathit{CI}\{1\}] = \mathit{CII}\{1\}\mathit{CI}\{0\} \ + \ \mathit{CII}\{0\}\mathit{CI}\{0\}\mathit{Cro}\{0\}, \\
[\mathit{CI}\{2\}] = \mathit{CII}\{1\}\mathit{CI}\{1\} \ + \ \mathit{CII}\{0,1\}\mathit{CI}\{2\} \ + \ \mathit{CII}\{0\}\mathit{CI}\{1\}\mathit{Cro}\{0\}, &
\end{array}$$



CI	*Cro*	$[N]$
0	0,1	1
0	2,3	0
1,2	0,1,2,3	0

CI	*Cro*	*CII*	$[CI]$
0,1	1,2,3	0	0
0	0,1,2,3	1	1
0	0	0	1
1	0,1,2,3	1	2
2	0,1,2,3	0,1	2
1	0	0	2

CI	*Cro*	$[Cro]$
0,1	0	1
0,1	1	2
0,1	2	3
0,1,2	3	2
2	0,1	0
2	2	1

CI	*Cro*	*N*	$[CII]$
0,1,2	0,1,2,3	0	0
0,1	0,1,2	1	1
0	3	1	0
1	3	1	0
2	0,1,2,3	1	0

Figure 3. An extended MVN model *PL4* of the control mechanism for the lysis-lysogeny switch in bacteriophage λ (based on [23]).

$$[Cro\{0\}] = CI\{2\}Cro\{0,1\}, \qquad [Cro\{1\}] = CI\{0,1\}Cro\{0\} \; + \; CI\{2\}Cro\{2\},$$
$$[Cro\{2\}] = CI\{0,1\}Cro\{1\} \; + \; Cro\{3\}, \quad [Cro\{3\}] = CI\{0,1\}Cro\{2\}$$

From these equations we are then able to derive the rewrite rules required in $RL_S(PL4)$ to model the synchronous behaviour of *PL4*. To illustrate further this approach we present below the rewrite rules derived for CI (the remaining rules are omitted for brevity).

```
rl [CI0] : CII(0,s1) CI(1,1) Cro(1,S3) => CII(0,s1) CI(1,0) Cro(1,S3) .
rl [CI0] : CII(0,s1) CI(1,1) Cro(2,S3) => CII(0,s1) CI(1,0) Cro(2,S3) .
rl [CI0] : CII(0,s1) CI(1,1) Cro(3,S3) => CII(0,s1) CI(1,0) Cro(3,S3) .
rl [CI1] : CII(1,s1) CI(0,0) => CII(1,s1) CI(0,1) .
rl [CI1] : CII(0,s1) CI(0,0) Cro(0,s3) => CII(0,s1) CI(0,1) Cro(0,s3) .
rl [CI2] : CII(1,s1) CI(1,1) => CII(1,s1) CI(1,2) .
rl [CI2] : CII(0,s1) CI(1,1) Cro(0,s3) => CII(0,s1) CI(1,2) Cro(0,s3) .
```

As in the previous case study (see Section 4.3), we are now able to use the full range of Maude's analysis tools to investigate the behaviour of the *PL4* model. In order to do this we have to ensure the rewriting strategy `next` we developed is invoked when rewriting the model at the metalevel. This can be done by adding the following rewrite rule:

```
rl [step] : '__[T1,T2,T3,T4] => next('__[T1,T2,T3,T4]) .
```

This approach allows `next` to be introduced after each synchronous update step and is based on the fact that Maude always applies equations first before considering rewrite rules.

With the above in place we can then do searches at the metalevel similar to those illustrated in Section 4.3. For example, the following search checks whether *CI* and *CII* can be simultaneously in state 1 starting from initial state 0000:

```
search '__['N['0.D1,'0.D1],'CII['0.D1,'0.D1],'CI['0.D2,'0.D2],'Cro['0.D3,'0.D3]]
       =>+ '__['CI['1.D2,'1.D2],'CII['1.D1,'1.D1],T1:Term,T2:Term] .
```

Maude returns that this search has no solutions showing that the property doesn't hold. A similar search using initial state 1001 does hold and this indicates there is one solution that results in state 0113.

Identifying the point attractors for a model is an important part of the analysis. This was straightforward in the asynchronous case since point attractors represented deadlocked global states. However, in the synchronous case all traces are infinite and so additional work is needed. One way to address this is to introduce a Boolean operator `repState` at the metalevel that indicates if a global state remains unchanged after a synchronous update step. (Note that this again illustrates the expressiveness provided by Maude's metalevel capabilities.) This can be defined equationally as follows:

```
eq repState(T) = same(T,next(T)) .
```

where `same` is an equationally defined function that returns true only if two metalevel state terms contain the same entities in the same states. We can then use this together with the `search` command to find point attractors. For example, the following search shows that state 1010 will eventually enter the point attractor 0020, 0020, . . . (note the use of `such that` to place a condition on the result of the search).

```
search '__['N['1.D1,'1.D1],'CII['0.D1,'0.D1],'CI['1.D2,'1.D2],'Cro['0.D3,'0.D3]]
       =>+ T:Term such that repState(T) .
```

Further analysis of *PL4* can be done by again utilizing the LTL model checker provided by Maude. This can be applied in a similar way to that illustrated in Section 4.3 but in this case it will work at the metalevel. To illustrate this, consider checking the hypothesis that when *CI* becomes permanently inactive then *Cro* must continually be able to reach full activation. This can be checked using the following model checking instruction:

```
red modelCheck('__['CI['1.D2,'1.D2],'CII['1.D1,'1.D1],'Cro['2.D3,'2.D3],
               'N['0.D1,'0.D1]], <> [] atCI('0) -> []<> atCro('3)) .
```

This holds for the given initial state 0112, and further tests indicate this is a potential invariant.

We can further extend our analysis capabilities by developing our own metalevel operators which can be used to define interesting atomic propositions. As an example, consider using the metalevel Boolean operator `repState` to define an atomic proposition `rept : -> Prop` as follows:

```
eq  T |= rept = repState(T) .
```

where `T` is a variable of type $Term$. This atomic proposition can the be used to form interesting LTL formulas for model checking. For example, suppose we want to check whether N and *CII* becoming simultaneously active is a predictor for a point attractor. The following model checking instruction shows the property is true for initial state 1000.

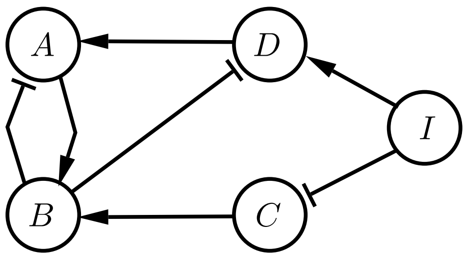
```
red modelCheck('__['CI['0.D2,'0.D2],'CII['0.D1,'0.D1],'Cro['0.D3,'0.D3],
               'N['0.D1,'0.D1]], (<> (atN('1) /\ atCII('1))) -> <> rept) .
```

Checking a further sample of initial states shows that the property is potentially an invariant of the model.

6. Evaluating the Scalability of the RL Framework

The case studies presented in Sections 4.3 and 5.3 provide a good illustration of the practical application of the RL techniques we have developed. However, they provide little indication of how the developed RL approach would scale when applied to larger MVN models and what impact the well–known state pace explosion problem would have. In this section we set out to address this by investigating how the our RL framework performs as the MVN size (i.e. number of entities) increases. The approach we take is to define an artificial, scalable test MVN and then use this to produce an incremental set of test MVNs. Note that the ease with which this scalable test model can be implemented in our RL framework illustrates how versatile it is.

The idea is to define a five entity MVN model *ABCDI* that can be used as a basic building block and to then compose instances of these basic blocks together to make models of size 10, 15, 20, 25, etc. The basic building block MVN is presented in Figure 4. To allow multiple instances of this model



$$\begin{aligned}
A\{0\} &= B\{0\}D\{0\} \;+\; B\{1,2\}\\
A\{1\} &= B\{0\}D\{1\}\\
B\{0\} &= A\{0\}C\{0\}\\
B\{1\} &= A\{0\}C\{1\} \;+\; A\{1\}C\{0\}\\
B\{2\} &= A\{1\}C\{1\}\\
C\{0\} &= I\{1\}\\
C\{1\} &= I\{0\}\\
D\{0\} &= I\{0\} \;+\; B\{2\}\\
D\{1\} &= B\{0,1\}I\{1\}
\end{aligned}$$

Figure 4. An MVN model *ABCDI* which is used as a basic building block to construct models of increasing size. Note that I can be viewed as simply an input entity that remains in its initial state.

to be easily composed we represent each entity as a family of entities. This is straightforward to do in RL by associating a natural number parameter to each entity as the following excerpt shows for the asynchronous case:

```
ops A C D I : Nat D1 -> Entity .
op  B : Nat D2 -> Entity .
```

We can then have general rewrite rules for the MVN as the following rules modelling the asynchronous behaviour of entity A illustrate:

```
rl [A0] :  A(i,1) B(i,0) D(i,0) => A(i,0) B(i,0) D(i,0) .
rl [A0] :  A(i,1) B(i,1) => A(i,0) B(i,1) .
rl [A0] :  A(i,1) B(i,2) => A(i,0) B(i,2) .
rl [A1] :  A(i,0) B(i,0) D(i,1) => A(i,1) B(i,0) D(i,1) .
```

To compose instances of *ABCDI* together we simply link them by allowing one instance to influence the state of entity I in the other instance. For example, to create a model of size 10 we could compose two instances, referred to as instance 1 and 2, by having entity A in instance 2 activating entity I in instance 1. This is straightforward to do by adding the following rules for I in instance 1:

```
rl [I10]:  I(1,1) A(2,0) => I(1,0) A(2,0) .
rl [I11]:  I(1,0) A(2,1) => I(1,1) A(2,1) .
```

We can now create initial states for the new MVN model of size 10 which can be analysed in the normal way using Maude. For example, the following search checks whether entity B in instance 1 can reach state 2 starting from the initial global state 0100101100:

```
search A(1,0) B(1,1) C(1,0) D(1,0) I(1,1) A(2,0) B(2,1) C(2,1) D(2,0) I(2,0)
       =>+ B(1,2) GS:GState .
```

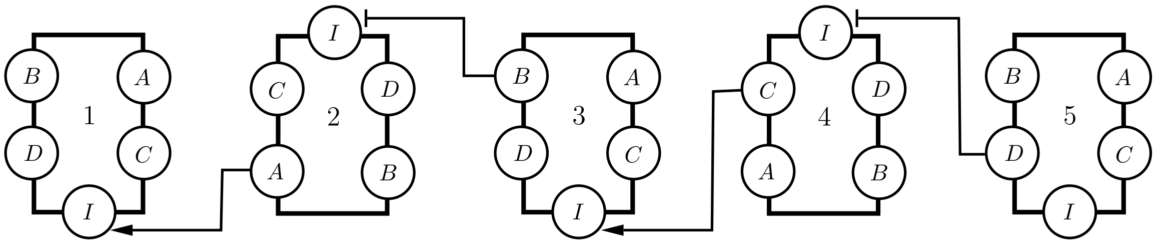


Figure 5. A pictorial representation of the five test MVN models constructed incrementally from the basic building block MVN given in Figure 4. Note that the arcs ending in arrows represent activation whereas the arcs ending in bars represent inhibition.

Using the test model generation approach presented above we were able to create a series of MVN test models in incremental steps of 5 entities as depicted in Figure 5. For each test model we used three simple types of test searches to see how the model performed: 1) searched for a point attractor; 2) searched for a state in which all input places become inactive; and 3) checked whether the entities could all go from being active to being inactive.

A summary of the test results produced is shown in Table 1 below. The tests were carried out using Maude on a computer with a 2.7 GHz Intel Core i5 processor and containing 16 GB 1333 MHz DDR3 memory. It can be seen that the largest test model that was able to produce results in the asynchronous case was the one containing 20 entities. The test model with 25 entities did not complete the searches even after an extended period. This highlights one of the key limitations of MVNs given the state space explosion problem. In order to allow larger models to be analysed new ways to compositional construct and analyse MVNs are needed and we discuss this further in the future work section at the end of the paper. In the synchronous case test results could be obtained for a much larger set of models (upto size 30 and beyond) and the resources needed to carry out the tests scaled well with respect to the MVN size. This is perhaps not surprising; while the synchronous RL framework is more complex due to the need to use meta–level strategies the actual dynamics of synchronous MVNs is far less complex (recall traces are deterministic) than their asynchronous counterparts. Finally, we note that while the performance testing in this section provides some insight into the scalability of the developed RL framework it is limited in its scope. Much further work is needed here and in particular, we aim to develop a set of benchmark scalable test models which can be used to evaluate MVN approaches in the literature.

Table 1. Summary of test results for performing three simple searches on the series of (A) asynchronous and (B) synchronous test MVNs.

(A) Asynchronous Test Results

Model Size	5	10	15	20	25
Rewrites	30	2472	150133	8374315	-
	33	2632	292228	12696152	-
	30	2472	150133	8374315	-
Time (ms)	0	17	1040	10.01mins	-
	0	17	2234	26mins	-
	0	17	1145	10.67mins	-

(B) Synchronous Test Results

Model Size	5	10	15	20	25	30
Rewrites	120	400	510	805	1080	1395
	182	477	752	807	1082	1397
	142	452	602	772	962	1307
Time (ms)	6	14	19	29	37	48
	8	19	28	28	37	48
	7	19	222	27	35	47

7. Conclusions

In this paper, we have developed an algebraic framework for modelling and analysing MVNs using RL. Our aim here has been to provide a well–supported, unifying algebraic framework which can be used as the basis for investigating the challenging issues that remain for biological qualitative modelling. In particular, this work is motivated by interesting interactions with the synthetic biology group at Newcastle[1] and their search for powerful, adaptable and scalable formal tools and techniques to support their work on engineering biological systems. The systematic translation presented from both synchronous and asynchronous MVNs to RL models with associated formal correctness proofs is therefore an important contribution to this work. Further, we have illustrated the potential application of our techniques with two detailed case studies[2] based on using the Maude tool to model and analyse existing genetic regulatory networks from the literature. In particular, these case studies have highlighted the interesting range of analysis possible for MVNs using our RL framework and the powerful analysis tools provided by Maude.

As noted in the introduction, *Pathway Logic (PL)* [14, 15] is a closely related existing RL framework for symbolically modelling and analysing signal transduction and metabolic pathways. While the two RL frameworks have different focuses they can be seen to be complementary and it would be interesting

[1]www.ncl.ac.uk/csbb/

[2]All the RL models used in this paper along with a prototype tool automating the translation process are available upon request.

to consider linking the two approaches (one of the clear advantages afforded by using RL).

Petri nets provide a natural alternative framework for modelling MVNs and a range of work in this area can be found in the literature (for example, see [4, 36, 5]). One notable tool framework is *GinSim* [37] which provides an impressive array of techniques for modelling and analysing asynchronous MVN models. The work reported here provides an alternative framework to Petri nets and provides an important extension to the formal foundations available for exploring the many challenging areas that remain for MVNs. It is also interesting to note that since Petri nets can be modelled using RL [38] our RL framework can be seen as as providing a foundation for integrating related models expressed in different formalisms.

The work presented in this paper provides the foundation for a wide range of interesting further work. For example, one key area is the development of compositional techniques for the construction and analysis of MVNs. Work in this area is currently underway with the aim of integrating compositional methods directly into the RL framework presented here. Another interesting area is the application of this work to support the scalable automation of engineering techniques in synthetic biology. The idea here is to investigate using the RL tools and techniques being developed by taking part in a joint project with the synthetic biology group at Newcastle. Finally, we note that one of the strengths of RL is its ability to integrate different modelling approaches (e.g. Petri nets, Pathway Logic, process algebra, etc.) within one formal framework. We intend to explore making use of this feature in future work by considering how a range of different models for a biological system based on different formalisms can be integrated and analysed using RL.

Acknowledgements. We would like to thank Prof Maciej Koutny and Prof Anil Wipat for their support and encouragement with this work. We would also like to take this opportunity to thank Prof Gabriel Ciobanu for many interesting and inspiring discussions.

References

[1] E. Bartocci and P. Lió. Computational modeling, formal analysis and tools for systems biology. *PLOS Computational Biology*, 12(1), pp. e1004591, 2016.

[2] R. Rudell and A. Sangiovanni-Vincentelli. Multiple-Valued Minimization for PLA Optimization. *IEEE Transactions on Computer-Aided Design*, CAD-6, 1987.

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

[4] C. Chaouiya, E. Remy, and D. Thieffry. Petri Net Modelling of Biological Regulatory Networks. *Journal of Discrete Algorithms*, 6(2):165–177, 2008.

[5] 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.

[6] S. A. Kauffman. Metabolic stability and epigenesis in randomly constructed genetic nets. *Journal of Theoretical Biology*, 22(3):437–467, 1969.

[7] 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.

[8] 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.

[9] J.Meseguer. Conditional rewriting logic as a unified model of concurrency. *Theoretical Computer Science*, 96(2):73–155, 1992.

[10] N.Martí-Oliet and J.Meseguer. Rewriting logic as a logical and semantic framework. In: D.M.Gabbay and F.Guenthner (eds), *Handbook of Philosophical Logic (Second Edition)*, Vol. 9, pages 1–87, Kluwer Academic Publishers, 2002.

[11] G. Ciobanu, M. Koutny, and L. J. Steggles. Strategy based semantics for mobility with time and access permissions. *Formal Aspects of Computing*, 27(3):525–549, 2014.

[12] M-O. Stehr, J. Meseguer, and P. C. Ölveczky Rewriting Logic As a Unifying Framework for Petri Nets. In: H. Ehrig, *et al.* (eds), Unifying Petri Nets: Advances in Petri Nets, LNCS 2128, pages 250–303, Springer Verlag, 2001.

[13] L. J. Steggles. Rewriting Logic and Elan: Prototyping Tools for Petri Nets with Time. Applications and Theory of Petri Nets 2001, LNCS 2075, pages 363-381, Springer Verlag, 2001.

[14] S. Eker, M. Knapp, K. Laderoute, P. Lincoln, J. Meseguer, K. Sonmez. Pathway Logic: Executable models of biological networks. In: F. Gadducci, U. Montanari (eds.), *Proc. of WRLA 2002*, *Electronic Notes in Theoretical Computer Science*, 71:144–161, 2004.

[15] V. Nigam, R. Donaldson, M. Knapp, T. McCarthy and C. Talcott. Inferring Executable Models from Formalized Experimental Evidence. In: Computational Methods in Systems Biology 9308, pages 90–103, Springer Verlag, 2015.

[16] M.Clavel, *et al.* Maude: Specification and Programming in Rewriting Logic. *Theoretical Computer Science*, 285(2):187–243, 2002.

[17] P.Borovanský, C.Kirchner, H.Kirchner, P.–E. Moreau and C.Ringeissen. An overview of ELAN. In: C. Kirchner and H. Kirchner (eds), Proc. of WRLA '98, *Electronic Notes in Theoretical Computer Science*, 15, 1998.

[18] E.Balland, P.Brauner, R.Kopetz, P.-E.Moreau, and A.Reilles. Tom: Piggybacking rewriting on java. In: RTA'07, LNCS 4533, pages 36–47, Springer Verlag, 2007.

[19] S. Eker, N. Martí-Oliet, J. Meseguer, A. Verdejo. Deduction, Strategies, and Rewriting. *Proc. of STRATEGIES 2006*, *Electronic Notes in Theoretical Computer Science*, 174(11):3–25, 2007.

[20] 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.

[21] 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.

[22] E. Simão, E. Remy, D. Thieffry and C. Chaouiya. Qualitative modelling of regulated metabolic pathways: application to the tryptophan biosynthesis in E. Coli. *Bioinformatics*, 21:190-196, 2005.

[23] 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.

[24] *Pathway Logic*, http://pl.csl.sri.com/. Accessed: 30/12/2016.

[25] S. Eker, J. Meseguer, A. Sridharanarayanan. The Maude LTL Model Checker. In: F. Gadducci, U. Montanari (eds.), *Proc. of WRLA 2002*, *Electronic Notes in Theoretical Computer Science*, 71:162–187, 2004.

[26] M.Clavel, *et al. Maude Manual (Version 2.7)* http://maude.lcc.uma.es/manual/maude-manual.html Accessed April 2016.

[27] M. Schaub, T. Henzinger, and J. Fisher. Qualitative networks: A symbolic approach to analyze biological signaling networks. *BMC Systems Biology*, 1:4, 2007.

[28] 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.

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

[30] R. Thomas. Boolean formalization of genetic control circuits. *Journal of Theoretical Biology*, 42:563–585, 1990.

[31] S. A. Kauffman. *The origins of order: Self-organization and selection in evolution.* Oxford University Press, New York, January 1993.

[32] L. Tournier and M. Chaves. Uncovering operational interactions in genetic networks using asynchronous Boolean dynamics. *Journal of Theoretical Biology*, 260:196–209, 2009.

[33] 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.

[34] *MVSIS Group. UC Berkeley*: https://embedded.eecs.berkeley.edu/mvsis/, Accessed: 30/12/2016.

[35] Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems – Specification.* Springer, 1992.

[36] L. J. Steggles, R. Banks, O. Shaw, A. Wipat. Qualitatively modelling and analysing genetic regulatory networks: a Petri net approach. *Bioinformatics*, 23(3):336-343, 2006.

[37] C. Chaouiya, A. Naldi, D. Thieffry. Logical Modelling of Gene Regulatory Networks with GINsim. Methods in molecular biology (Clifton, N.J.), 804:463-79, 2012.

[38] J. Meseguer and U. Montanari. Petri nets are monoids. *Information and Computation*, 88(2):105Ű155, 1990.