Extending boolean regulatory network models with Answer Set Programming

Timur Fayruzov*, Jeroen Janssen†, Chris Cornelis*, Dirk Vermeir† and Martine De Cock*
*Dept. of Applied Mathematics and Computer Science, Ghent University, Krijgslaan 281 (S9), 9000 Ghent, Belgium
Email: {timur.fayruzov, martine.decock, chris.cornelis}@ugent.be
†Department of Computer Science, Vrije Universiteit Brussel, Pleinlaan 2, 1050 Brussels, Belgium
Email: dvermeir@tinf.vub.ac.be, jeroen.janssen@vub.ac.be

***Abstract*—Because of their simplicity, boolean networks are a popular formalism to model gene regulatory networks. However, they have their limitations, including their inability to formally and unambiguously define network behaviour, and their lack of the possibility to model meta interactions, i.e., interactions that target other interactions. In this paper we develop an answer set programming (ASP) framework that supports threshold boolean network semantics and extends it with the capability to model meta interactions. The framework is easy to use but sufficiently flexible to express intricate interactions that go beyond threshold network semantics as we illustrate with an example of a Mammalian cell cycle network. Moreover, readily available answer set solvers can be used to find the steady states of the network.**

I. Introduction

Boolean networks are a very simple yet popular formalism to model gene regulatory networks. Two recent examples applying these formalisms are [2] and [3], where they are used to model cell cycle processes of fission yeast, resp. mammals.

Although they are popular, the current boolean networks formalisms have several drawbacks. First, although the aforementioned applications are presented using boolean networks, a detailed analysis shows that they use different formalisms. In [3] the authors use standard boolean networks, while in [2] the authors use threshold boolean networks with different semantics. This difference is not evident from the surface, but makes a crucial difference in the way the networks behave.

Another issue is that the boolean network in [3] makes some assumptions about the 'default' state and behaviour of the nodes in the network, i.e., only the node activation rules are given, and it is implicitly assumed that if an activation rule is not satisfied then the node should be inhibited. This assumption is only stated in the plain text in [3].

Yet another problem is that the boolean formalism does not allow some facts about the network in [3] to be stated explicitly. In particular, statements of the form 'a inhibits the binding of b to c' that express a meta reaction with a binding interaction as its target, cannot be expressed directly.

Jeroen Janssen is funded by a research project of the Research Foundation – Flanders.
Chris Cornelis is a postdoctoral fellow of the Research Foundation – Flanders.

In this paper we propose a new approach to model regulatory networks that solves these problems. In particular, we propose to represent gene and protein regulatory networks by answer set programs, as an extension of our previous work presented in [4]. We extend this framework with new functionalities and provide a more efficient implementation that allows to

- express boolean networks in a uniform way
- formally and unambiguously define network behaviour
- describe meta interactions that are not allowed in traditional boolean networks, which allows to express regulation networks more intuitively.

To study the applicability of our framework we provide a case study of the boolean model of mammalian cell cycle first described in [3].

The paper is structured as follows. First, we present the necessary background on answer set programming and boolean networks in Section II, then we proceed with the framework description in Section III. Next, we provide a case study of our framework on the mammalian cell cycle model in Section IV and we conclude in Section V.

II. Preliminaries

A. Answer Set Programming

Answer set programming [6] is a declarative formalism that allows to express relations propositions with rules of the form

$$L_0 \quad \leftarrow \quad L_1, \ldots, L_m, not\ L_{m+1}, \ldots, not\ L_n$$

in which $L_0, L_1, \ldots, L_n$ are called *literals*. The left-hand (resp. right-hand) side of a rule is called the *head* (resp. *body*). A rule with an empty head (resp. body) is called a *constraint* (resp. *fact*). A rule intuitively states that whenever the literals in the body hold true, the head should be true as well. A literal can be negated; then it is preceded by the symbol $\neg$ and is called a negative literal. Another form of negation that represents a special feature of ASP, is negation-as-failure (naf) denoted by *not*.

An answer set program is a set of rules. The set of all literals of a program P is denoted by Lit_P. An interpretation of P is any consistent[1] subset $S \subseteq Lit_P$. S is said to

[1]A set of literals is said to be consistent if it does not contain a literal l and its negated literal $\neg l$ simultaneously

satisfy a rule with a nonempty head, if $\{L_1, \ldots, L_m\} \subseteq S$ and $\{L_{m+1}, \ldots, L_n\} \cap S = \emptyset$ implies that $L_0 \in S$. When the head is empty, then S is said to satisfy the rule if $\{L_1, \ldots, L_m\} \not\subseteq S$ or $\{L_{m+1}, \ldots, L_n\} \cap S \neq \emptyset$.

An interpretation that satisfies all rules of a program P is called a model of P. Answer sets are special kinds of models. First of all, for a program P without naf, an answer set of P is a minimal model of P, i.e., S is called an answer set of P iff S is a model of P and there is no model K such that $K \subset S$.

Example 1 The models of the program that contains only rule $a \leftarrow b$ are $\{a, b\}$, $\{a\}$ and $\emptyset$. The minimal model is $\emptyset$.

The concept of an answer set is extended for a program P containing negation-as-failure as described below. Suppose that S is a model of P, and our hypothesis is that S is an answer set of P. In order to check if S is an answer set of P, we build a reduct program P' by 1) removing from P all rules that contain a naf-literal $not\ L$, with $L \in S$; 2) removing all the naf-literals from the bodies of the remaining rules. If the minimal model S' of the naf-free program P' coincides with S, then S is an answer set of the original program P.

Example 2 A program with negation-as-failure can have more than one answer set. Suppose that we have one seat and two persons a and b, and we want to assign the seat to one of them. We can model this by the following program

$$\begin{array}{lcl} seat(a) & \leftarrow & not\ seat(b) \\ seat(b) & \leftarrow & not\ seat(a) \end{array}$$

It has two answer sets $\{seat(a)\}$ and $\{seat(b)\}$.

B. Boolean networks

A boolean network captures interactions between genes and proteins (further referred to as 'compounds') in the form of a directed graph $G = (V, E)$ with V a set of nodes and E a set of edges. The nodes represent compounds while the edges represent the influence of one compound on another. At any time, a node is in one of two states: either it is active (1), or it is not active (0) (hence, the name boolean networks). The state of a gene regulation network at any given time is defined in terms of the states of its nodes.

Definition 1 (Network state): Let $G = (V, E)$ be a graph representing a gene regulation network. Then a mapping $S : V \rightarrow \{0, 1\}$, that maps every node in V to a state in $\{0, 1\}$, is called a network state.

Every node has *input nodes* that are determined by inbound edges, and *output nodes* that are determined by the outbound edges of the node. For every node in the network a deterministic transition function (TF) can be defined that determines the next state of the node depending on the node's inputs. The network can switch from one state to another by applying the TF to its nodes. The TF can be represented as a boolean function although different representations are also possible as discussed further. For the purpose of this work we consider a network TF that is a combination of TFs on the nodes.

Definition 2 (Transition function): A function f that maps a network state S to another network state $f(S) = S'$ is called a transition function.

In threshold boolean networks the TF for a node is defined as the sum of input signals for the node. To define these networks we introduce the notion of network marking.

Definition 3 (Network marking): Let $G = (V, E)$ be a graph representing a gene regulation network. Then a mapping $M : V^2 \rightarrow \{-1, 0, 1\}$, that maps every pair of nodes to $\{-1, 0, 1\}$, is called a network marking.

Intuitively, given the pair of nodes $\langle v_1, v_2 \rangle$, a negative marking denotes the existence of the suppression edge between v_1 and v_2, a positive marking denotes an activation edge between v_1 and v_2 and 0 denotes the absence of an edge. The TF for node v_i can then be defined as follows

$$S_{t+1}(v_i) = \begin{cases} 1 & \text{if } \sum_{j=1}^{|V|} M(v_j, v_i) S_t(v_j) + h > 0, \\ 0 & \text{if } \sum_{j=1}^{|V|} M(v_j, v_i) S_t(v_j) + h < 0, \\ S_t(v_i) & \text{if } \sum_{j=1}^{|V|} M(v_j, v_i) S_t(v_j) + h = 0, \end{cases} \quad (1)$$

where $S_{t+1}(v_i)$ and $S_t(v_j)$ are states of node v_i at time point $t+1$, and of node v_j at time point t correspondingly, and h is a threshold parameter. By setting $h = 0$ we obtain an intuitive interpretation of a regulatory network where a gene is activated if it receives a positive input and inhibited if it receives a negative input.

For dynamics analysis of (threshold) boolean networks, discrete time is usually considered, i.e., there is an external 'clock' that iterates over the values $1, 2, \ldots$ At every time step, a TF is applied to the network which causes a change of the network state at the next time step. The network evolution over time is called a trajectory.

Definition 4 (Trajectory): A sequence T of network states $S, f(S), f(f(S)) \ldots$ is called a *trajectory* of the network.

Due to the deterministic nature of the network, after at most $2^{|V|}$ steps the network will visit a previously visited state and either will stay in this state, or start to loop through a set of visited states. Such states are called *steady states* and are defined below.

Definition 5 (Steady state, steady cycle): A state S of a network is called *a steady state* if $f(S) = S$. A subsequence $S_m \ldots S_n (m < n)$ of a trajectory is called a *steady cycle* if $f(S_n) = S_m$. The set of trajectories that lead to a given steady state or cycle is called *a basin of attraction*.

Example 3 Assume that in Figure 1a protein a is active and b is inhibited, then the initial network state is $\langle 1, 0 \rangle$. The TF is defined as in (1) with $h = 0$. By applying this function to the initial state, we can go to the next network state $\langle 1, 1 \rangle$. If we apply the TF once again, we move to

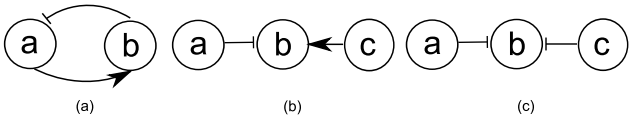

Figure 1. Examples of regulation networks. Arrows denote activation and blunt edges denote inhibition.

$$
\begin{array}{lll}
G1: & \text{time}(0..T). & \\
G2: & \text{interaction}(\text{activates}(X,Y)) & \leftarrow \text{activates}(X,Y), \text{protein}(X), \text{entity}(Y). \\
G3: & \text{interaction}(\text{inhibits}(X,Y)) & \leftarrow \text{inhibits}(X,Y), \text{protein}(X), \text{entity}(Y). \\
G4: & \text{entity}(X) & \leftarrow \text{interaction}(X). \\
G5: & \text{entity}(X) & \leftarrow \text{protein}(X).
\end{array}
$$

Figure 2. Basic framework rules

the state $\langle 0, 1\rangle$, and after that the state does not change any more, no matter how many times we apply the TF. This means that the network has reached a steady state. The corresponding trajectory is $\langle 1, 0\rangle, \langle 1, 1\rangle, \langle 0, 1\rangle$.

III. ASP FRAMEWORK

Several researchers applied ASP to model different aspects of biological systems ([1], [5]). In this section we provide an ASP framework that models the behaviour of threshold boolean networks, and on top of that allows to deal with meta interactions. The network model is presented as an ASP program that consists of two parts: the framework and the structure description. The framework contains the rules that describe how the network works (further referred to as G-rules). These rules are independent of any specific network. The structure description part contains the rules that describe the structure of a particular network that we want to model (further referred to as S-rules).

A. Structure description

We start building the regulation network by describing the S-rules. The set of S-rules for the network in Figure 1a consists of the following facts

$$
\begin{array}{ll}
\text{protein}(a). & \text{protein}(b). \\
\text{activates}(a,b). & \text{inhibits}(b,a).
\end{array}
$$

Here in the first line we declare that we have two nodes a and b, while the second line describes the interactions between them. Note that we define the type of a as *protein*. For the sake of the example, any type of biological entity can be defined in the model (gene, enzyme, etc.). By themselves these rules do not model anything; although they define the connection between genes and proteins, they do not describe the influence of these connections on the proteins at the different time steps – this is the task of the G-rules as described below.

B. Framework essentials

The rules that set up the environment for our framework are shown in Figure 2. Rule G1 is merely a shorthand for the facts $time(0)., \ldots, time(T)$. where T is a constant that defines how many time steps in our modelling process we would like to consider. Rules G2 and G3 declare that the inhibition and activation reactions that we define as S-rules (described above) are of type *interaction*. Rules G4 and G5 declare that both *proteins* and *interactions* are of type *entity*.

Next, we define what it means for one *protein* to activate or inhibit another *entity*. The threshold boolean network semantics defined in Section II-B can be captured by the rule

$$G6: \quad \text{int}(0..\#\_of\_ent).$$

and by the rules G7, G8 described in Figure 3. In these rules $\text{int}(A)$ and $\text{int}(I)$ are defined to be integers from 0 to the number of entities defined in the model as stated in rule G6. Indeed, the number of incoming edges to any given node cannot be more than the number of nodes in the network, thus this is a reasonable limitation. The rules G7 and G8 implement the idea behind the threshold network: we count the number of activation and inhibition links for every instance and make the decision based on this count. The counting happens in literals $\#\_\,\text{act}(Y, A, T-1)$ and $\#\_\,\text{inh}(Y, I, T-1)$ which store the count in variables A and I correspondingly. Note that when the number of incoming activation and inhibition links is equal, none of the rules G7 or G8 are applicable, because neither $A - I > 0$ nor $I - A > 0$ is satisfied, thus the entity Y should remain in the same state as before. This is implemented with the following inertia rules

$$
\begin{array}{lll}
G9: & \text{act}(X,T) & \leftarrow \text{act}(X,T-1), \mathit{not}\ \text{inh}(X,T), T > 0. \\
G10: & \text{inh}(X,T) & \leftarrow \text{inh}(X,T-1), \mathit{not}\ \text{act}(X,T), T > 0.
\end{array}
$$

Intuitively rule G9 says that that if X was active at $T-1$ and there is no evidence that it is inhibited at T then it remains active. A similar reasoning holds for G10.

Returning back to rules G7 and G8, how do we define which activation and inhibition links affect the state of Y, in other words, how do we define counting predicates $\#\_act$ and $\#\_inh$? The answer to this question is straightforward in the case of threshold network semantics: count the links that have active triggers.

Example 4 In the network in Figure 1b a is active and c is not. In this case only the inhibition link to b will be counted, as the activation link has no effect in the given network state.

However, the threshold boolean semantics does not take into account the fact that interactions themselves can be influenced by other entities (meta interactions). Examples of such interactions are presented in Figure 4a-d. Before

$$
\begin{array}{rlcl}
G7: & \mathrm{act}(Y,T) & \leftarrow & \#\_\mathrm{act}(Y,A,T-1), \#\_\mathrm{inh}(Y,I,T-1), A-I>0, T>0, \mathrm{int}(A), \mathrm{int}(I).\\
G8: & \mathrm{inh}(Y,T) & \leftarrow & \#\_\mathrm{act}(Y,A,T-1), \#\_\mathrm{inh}(Y,I,T-1), I-A>0, T>0, \mathrm{int}(A), \mathrm{int}(I).\\
G7^1: & \mathrm{act}(Y,T) & \leftarrow & \#\_\mathrm{act}(Y,A,T-1), \#\_\mathrm{inh}(Y,I,T-1), \mathrm{act\_th}(Y,Th), A-I>Th, T>0, \mathrm{int}(A), \mathrm{int}(I).\\
G8^1: & \mathrm{inh}(Y,T) & \leftarrow & \#\_\mathrm{act}(Y,A,T-1), \#\_\mathrm{inh}(Y,I,T-1), \mathrm{inh\_th}(Y,Th), I-A>Th, T>0, \mathrm{int}(A), \mathrm{int}(I).\\
G7^2: & \mathrm{act}(Y,T) & \leftarrow & \#\_\mathrm{act}(Y,A,T-1), \#\_\mathrm{inh}(Y,I,T-1), \mathrm{act\_th}(Y,Th), not\ \mathrm{abn}(Y,T-1), A-I>Th, T>0, \mathrm{int}(A), \mathrm{int}(I).\\
G8^2: & \mathrm{inh}(Y,T) & \leftarrow & \#\_\mathrm{act}(Y,A,T-1), \#\_\mathrm{inh}(Y,I,T-1), \mathrm{inh\_th}(Y,Th), not\ \mathrm{abn}(Y,T-1), I-A>Th, T>0, \mathrm{int}(A), \mathrm{int}(I).
\end{array}
$$

Figure 3. The evolution of the definition of rules G7 and G8.

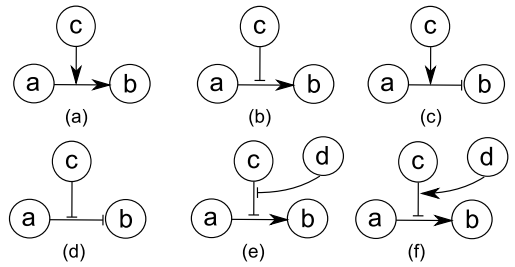

Figure 4. Examples of meta interactions

explaining how this is implemented in our framework, let us discuss the semantics of these interactions.

In the cases when c inhibits the interaction (activation/inhibition) between a and b, as in Figure 4b,d, we want to disregard the interaction between a and b when c is active, and consider it as usual when c is inactive. In the cases when c activates the interaction (activation/inhibition) between a and b, as in Figure 4a,c, the situation is slightly different. When we say that c activates the interaction between a and b we implicitly assume that this interaction is not functional without c, because if it were, there is no point to add c in the model. Thus, according to this reasoning the interaction between a and b in this case is taken into account only when c is active.

Keeping these semantics in mind we can start crafting the rules that define the predicates $\#\_act$ and $\#\_inh$ as presented in Figure 5. The following explanation will be focused on rule G11, but it applies equally to G12 as well. In G11 $\#\_potential\_act$ is the number of activation links that enter node Y. This is a structural property of the model, thus time is not an argument of this predicate. However, not all of these links are active at a given network state at time T, because interaction triggers may be inhibited or because the interaction itself may be inhibited as we discussed above. This is captured by the construct $I\{inh\_\,\mathrm{act}(X,Y,T) : \mathrm{activates}(X,Y) : \mathrm{protein}(X)\}I$ for activation links. We do not go into the technical details of this construct due to space restrictions, but the intuition is that it takes all activation links $\mathrm{activates}(X,Y)$ where X is a *protein* and Y is an *entity*, counts only those for which the inh_act predicate holds and stores this count in I. In other words, I counts the number of inactive activation links that enter Y. Thus, $A-I$ represents the actual number of activation links that influence node Y in the current network state.

Rules G11.1-3 and G12.1-3 from Figure 5 help to count these numbers. Before explaining them in more detail, let us define the cases when an interaction can be deactivated. An interaction does not influence its target if

1) the interaction trigger is not active
2) there is an interaction that inhibits this interaction and its trigger is active (Figure 4b,d where c is active)
3) there is an interaction that activates this interaction and its trigger is not active (Figure 4a,c where c is not active)

The first case is handled by rules G11.1 and G12.1.

The second case is handled by rules G11.2 and G12.2. In G11.2 we have an interaction between X and Y, but also we have Z that inhibits this interaction. The interaction will be inhibited if Z is active. Note also the recursive condition not inh_inh$(Z, \mathrm{activates}(X,Y), T)$ that says that the inhibition interaction that is triggered by Z should not be inhibited itself (as in Figure 4e). Rule G12.2 acts in a similar way.

The third case is handled by rules G11.3 and G12.3. Rule G11.3 makes use of the previously defined rules G11.1 and G11.2. The interaction between X and Y will be considered inactive if Z is not active, or, even if Z is active, if there is some other interaction that restricts the influence of Z (as in Figure 4f).

C. Sensitivity thresholds

Some features still cannot be expressed in this framework. For example, in reality proteins can become active when their inhibitors are not active, even without an external activation input. Another example is that some proteins can have a certain 'tolerance' to an inhibition/activation influence. For example, a protein can become inhibited only if two or more proteins that suppress it are active, otherwise it is not affected. To address these issues we introduce the notion of inhibition and activation thresholds. This can be implemented in the system by introducing inhibition/activation thresholds as shown in Figure 6.

We update rules G7 and G8 by G7^1 and G8^1 so that now they take into account the possible presence of a threshold. Rules G13 and G14 set the activation and inhibition threshold of every protein to 0 in case it was not set explicitly by a special predicate mod_act_th or mod_inh_th, which is checked by rules G13.1 and G14.1.

$G11:$ #_act$(Y, A-I, T) \leftarrow$ #_potential_act$(Y, A), I\{$inh_act$(X, Y, T) : \text{activates}(X, Y) : \text{protein}(X)\}I, A >= I, \text{int}(A), \text{int}(I).$
$G12:$ #_inh$(Y, A-I, T) \leftarrow$ #_potential_inh$(Y, A), I\{$inh_inh$(X, Y, T) : \text{inhibits}(X, Y) : \text{protein}(X)\}I, A >= I, \text{int}(A), \text{int}(I).$
$G11.1:$ inh_act(Y,A) $\leftarrow$ *not* act$(X, T), \text{activates}(X, Y).$
$G12.1:$ inh_inh$(Y, I) \leftarrow$ *not* act$(X, T), \text{inhibits}(X, Y).$
$G11.2:$ inh_act$(X, Y, T) \leftarrow \text{act}(Z, T), \text{protein}(Z), \text{inhibits}(Z, \text{activates}(X, Y)),$ *not* inh_inh$(Z, \text{activates}(X, Y), T).$
$G12.2:$ inh_inh$(X, Y, T) \leftarrow \text{act}(Z, T), \text{protein}(Z), \text{inhibits}(Z, \text{inhibits}(X, Y)),$ *not* inh_inh$(Z, \text{inhibits}(X, Y), T).$
$G11.3:$ inh_act$(X, Y, T) \leftarrow \text{activates}(Z, \text{activates}(X, Y)), inh_act(Z, \text{activates}(X, Y), T), \text{protein}(Z).$
$G12.3:$ inh_inh$(X, Y, T) \leftarrow \text{activates}(Z, \text{inhibits}(X, Y)), inh_act(Z, \text{inhibits}(X, Y), T), \text{protein}(Z).$

Figure 5. A set of rules that define the way interactions are counted. Predicates $\text{protein}(X)$ and $\text{entity}(Y)$ are omitted from every rule for clarity

$G13:$ act_th$(X, 0) \leftarrow$ *not* mod_act_th$(X).$
$G13.1:$ mod_act_th$(X) \leftarrow$ act_th$(X, Th), h \neq 0.$
$G14:$ inh_th$(X, 0) \leftarrow$ *not* mod_inh_th$(X).$
$G14.1:$ mod_inh_th$(X) \leftarrow$ inh_th$(X, Th), h \neq 0.$

Figure 6. Sensitivity rules

Having both inhibiting and activating thresholds instead of one threshold is not redundant, since these thresholds characterize not the 'on/off' level of the protein, but rather an effort that is needed to change its state. Positive values make the protein more tolerant and negative ones make it less tolerant. Example 5 describes how thresholds can be used in a network.

D. Exception handling

It may be the case that certain interactions in a biological network model do not conform with the threshold network assumptions and thus cannot be represented within the framework semantics we have provided above. In order to allow for modelling arbitrary behaviour in the framework we introduce the notion of 'abnormal situation', or exception. This notion can be introduced by using rules G7^2 and G8^2 instead of G7^1 and G8^1 from Figure 3.

The modification allows to include the exceptional behaviour in the framework by means of predicate $abn/2$. Now the state of a gene or protein Y tagged with predicate $abn/2$ will not be governed by the framework semantics, and can be redefined according to the user needs. The example below illustrates the use of the exception mechanism.

Example 5 Let us construct the answer set program P consisting of general rules G1 (with upper limit $T = 2$), G2, G3, G4, G5, G6, G7^2, G8^2, G9, G10, G11, G11.1, G11.2, G11.3, G12, G12.1, G12.2, G12.3, G13, G13.1, G14, G14.1 and the specific rules

$S1:$ protein(a). $S5:$ act$(b, 0)$.
$S2:$ protein(b). $S6:$ act$(c, 0)$.
$S3:$ protein(c). $S7:$ inhibits(a, b).
$S4:$ act$(a, 0)$. $S8:$ activates(c, b).

The activation and inhibition thresholds of a and b are not explicitly defined; hence they are automatically set to the default value. The answer set of this program is $\{\text{act}(a, 0), \text{act}(b, 0), \text{act}(c, 0)\, \text{act}(a, 1), \text{act}(b, 1), \text{act}(c, 1), \text{act}(a, 2), \text{act}(b, 2), \text{act}(c, 2)\}$. The state of protein b does not change over time since its inhibiting and activating inputs are equal, and its thresholds for activation and inhibition are both 0. From the answer set we retrieve that the steady state is $\{\text{act}(a), \text{act}(b), \text{act}(c)\}$.

To illustrate the use of the thresholds let us set the inhibition threshold of b to -1 to indicate that this protein is susceptible to inhibition by adding the rule $inh\_th(b, -1)$. The answer set of this program is $\{\text{act}(a, 0), \text{act}(b, 0), \text{act}(c, 0)\, \text{act}(a, 1), \text{inh}(b, 1), \text{act}(c, 1), \text{act}(a, 2), \text{inh}(b, 2), \text{act}(c, 2)\}$. The steady state in this case is $\{\text{act}(a), \text{inh}(b), \text{act}(c)\}$.

To illustrate the use of exceptions let us use the same framework P and model the network presented in Figure 1c. Moreover, let us define the semantics of this network as follows: b is inhibited only when both a and c are active, and is not affected by these nodes otherwise. This behaviour is beyond the threshold boolean network semantics, thus we may use exceptions to model it. The set of S-rules is then presented as follows

$S1:$ protein(a). $S4:$ act$(a, 0)$.
$S2:$ protein(b). $S5:$ act$(b, 0)$.
$S3:$ protein(c). $S6:$ act$(c, 0)$.
$S7:$ abn$(b, T) \leftarrow$ both_act(a, c, T).
$S8:$ inh$(b, T) \leftarrow$ both_act$(a, c, T-1), T > 0$.
$S9:$ both_act$(a, c, T) \leftarrow \text{act}(a, T), \text{act}(c, T), T > 0$.

Rule S7 says that b should be processed in an exceptional way in case both a and c are active (as defined in rule S9). Rule S8 defines how b should be processed, i.e., that it should be inhibited in this case. Note that since the facts $\text{inhibits}(a, b, T)$ and $\text{inhibits}(c, b, T)$ are not present in the program, nothing happens with b when only one of the agents a or c is active.

IV. Case study: mammalian cell cycle

In [3] a boolean model of the mammalian cell cycle network is presented. We argue that this model suffers from the constraints imposed by the boolean network formalism, which leads to some unintuitive modelling choices.

Let us first focus on the relationship between a cyclin-dependent kinase inhibitor *p27/Kip1* (denoted as *p27* in the model) and *cdk2/Cyclin A* (denoted as *CycA*). When both *p27* and *CycA* are active, *p27* forms a complex with *CycA* and blocks activity of *CycA*. However, the cyclin remains present, and to model this fact, rather than drawing an inhibiting edge from *p27* to *CycA*, the blocking effect of

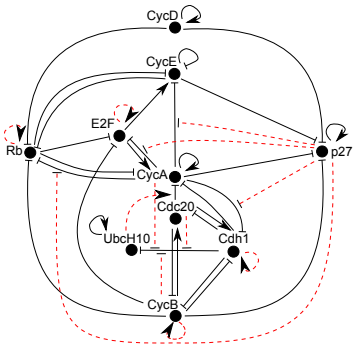

Figure 7. The mammalian cell cycle network model represented with the updated framework. Updated edges are shown dashed.

p27 is presented in [3] with edges from *p27* to the targets of *CycA*, but with opposite signs, i.e., if *CycA* activates *E2F* then there is also an inhibiting edge between *p27* and *E2F*. Apparently, this is a workaround because boolean networks cannot express an interaction between a node and an edge. This problem arises in modelling interactions of *p27* with *CycA* and *CycE* but also in modelling the activity of Anaphase Promoting Complex (APC).

APC is responsible for the progression and proper finishing of mitosis (the separation of the cell into two cells) and is presented in the model by proteins *Cdh1* and *Cdc20*. *Cdh1* is known to be an inhibitor of *CycA*, however, during the transition from the growth phase G2 to mitosis, while *Cdh1* is active, *CycA* reaches a concentration high enough to inactivate *Cdh1*, and it was long unclear why this happens. Recent research has revealed the role of the *E2 ubiquitin conjugating enzyme UbcH10* in this process by showing that *Cdh1*-dependent degradation of *CycA* can happen only in the presence of *UbcH10* [7]. Moreover, *Cdh1* triggers *UbcH10* ubiquitination, but only in the case when *Cdh1* targets are not active, or in terms of the logical model, when *CycA, CycB* and *Cdc20* are inactive. These observations were formalized in [3], but the relationships between the nodes in this model do not directly reflect them.

After the establishment of the new framework we can address the aforementioned issues and create a more understandable model of the mammalian cell cycle shown in Figure 7. This model has the same steady cycle as described in [3] as depicted in Table I, with the only exception that *UbcH10* is expressed one step earlier (denoted with the bold font face in the table). To determine the significance of this change for the model we need more feedback from biologists.

V. Conclusions

In this paper we have developed a framework to model regulatory networks as answer set programs. ASP is an area of logic programming that allows to model systems that exhibit non-monotone behaviour using negation-as-failure.

Table I
Mammalian cell cycle execution flow

node	1	2	3	4	5	6	7	8
CycD	1	1	1	1	1	1	1	1
Rb	0	0	0	0	0	0	0	0
E2F	0	1	1	1	0	0	0	0
CycE	0	0	1	1	1	0	0	0
CycA	0	0	0	1	1	1	1	0
p27	0	0	0	0	0	0	0	0
Cdc20	1	0	0	0	0	0	1	1
Cdh1	1	1	1	1	0	0	0	1
UbcH10	1	1	0	0	**1**	1	1	1
CycB	0	0	0	0	0	1	1	0

These models, represented as programs, can be executed to produce the set of steady states of a regulation network.

We have implemented a framework that covers threshold boolean network semantics and extended this framework with the possibility to model meta interactions. Furthermore our ASP framework is more formal compared to boolean networks, since it requires that all implicit assumptions are explicitly described in the body of the program, while in boolean networks this knowledge can be hidden in the non-formal description. However, the approach remains straightforward to apply; it does not require any formal logics knowledge from the biologist, who can operate with ready-to-apply blocks to build a model. At the same time the approach is very flexible due to the fact that any specific case which does not fit in the general picture can be incorporated with a minimal effort. Moreover, readily available answer set solvers can be used to find the steady states of a network.

References

[1] C. Baral, K. Chancellor, N. Tran, N. Tran, A.M. Joy and M.E. Berens. A knowledge based approach for representing and reasoning about signaling networks. *Bioinformatics* 20(1):15–22, 2004.

[2] M. I. Davidich and S. Bornholdt. Boolean network model predicts cell cycle sequence of fission yeast. *PLoS ONE*, 3(2), 2008.

[3] A. Faure, A. Naldi, C. Chaouiya, and D. Thieffry. Dynamical analysis of a generic Boolean model for the control of the mammalian cell cycle. *Bioinformatics*, 22(14):e124–131, 2006.

[4] T. Fayruzov, M. De Cock, C. Cornelis, and D. Vermeir. Modeling protein interaction networks with answer set programming. In *BIBM09*, pages 99–104, 2009.

[5] M. Gebser, C. Guziolowski, M. Ivanchev, T. Schaub, A. Siegel, S. Thiele and P. Veber. Repair and Prediction (under Inconsistency) in Large Biological Networks with Answer Set Programming. In *KR 2010*, pages 497–507, 2010.

[6] M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In *ICLP/SLP*, pages 1070–1080, 1988.

[7] M. Rape and M. W. Kirschner. Autonomous regulation of the anaphase-promoting complex couples mitosis to s-phase entry. *Nature*, 432:588 – 595, 2004.