Int. J. Data Mining and Bioinformatics, Vol

Modelling gene and protein regulatory networks with Answer Set Programming

Timur Fayruzov

Department of Applied Mathematics and Computer Science,
Ghent University,
Krijgslaan 281 (S9), 9000 Ghent, Belgium
E-mail: timur.fayruzov@ugent.be

Jeroen Janssen and Dirk Vermeir

Department of Computer Science,
Vrije Universiteit Brussel,
Pleinlaan 2, 1050 Brussels, Belgium
E-mail: jeroen.janssen@vub.ac.be
E-mail: dvermeir@tinf.vub.ac.be

Chris Cornelis

Department of Applied Mathematics and Computer Science,
Ghent University,
Krijgslaan 281 (S9), 9000 Ghent, Belgium
E-mail: chris.cornelis@ugent.be

Martine De Cock*

Department of Applied Mathematics and Computer Science,
Ghent University,
Krijgslaan 281 (S9), 9000 Ghent, Belgium

and

University of Washington,
1900 Commerce St., Tacoma, WA-98402, USA
E-mail: mdecock@u.washington.edu
*Corresponding author

Abstract: Recently, many approaches to model regulatory networks have been proposed in the systems biology domain. However, the task is far from being solved. In this paper, we propose an Answer Set Programming (ASP)-based approach to model interaction networks. We build a general ASP framework that describes the network semantics and allows modelling specific networks with little effort. ASP provides a rich and flexible toolbox that allows expanding the framework with desired features. In this paper, we tune our framework to mimic Boolean network behaviour and apply it to model the Budding Yeast and Fission

Yeast cell cycle networks. The obtained steady states of these networks correspond to those of the Boolean networks.

Keywords: systems biology; ASP; answer set programming; network modelling; budding yeast; fission yeast; cell cycle; steady state; steady cycle.

Reference to this paper should be made as follows: Fayruzov, T., Janssen, J., Vermeir, D., Cornelis, C. and De Cock, M. (2011) 'Modelling gene and protein regulatory networks with Answer Set Programming', *Int. J. Data Mining and Bioinformatics*, Vol.

Biographical notes: Timur Fayruzov holds an MSc in Computer Science from USATU (Russia). Currently, he is a PhD Student at Ghent University (Belgium), supported by a grant from the University's Special Research Fund. He worked as a Visiting Scholar at the University of Washington, Tacoma (USA). His research interests include text mining, knowledge representation and their application to the domain of molecular biology.

Jeroen Janssen holds an MSc in Computer Science from Ghent University. Currently, he is a PhD Student at the Vrije Universiteit Brussel (Belgium), supported by a project from the Research Foundation-Flanders. His research interests include declarative programming and fuzzy logic.

Dirk Vermeir is a Professor at the Department of Computer Science of the Vrije Universiteit Brussel. His current research interests include answer set programming semantics and applications.

Chris Cornelis is a Postdoctoral Fellow of the Research Foundation-Flanders at Ghent University. His research interests include fuzzy sets and rough sets, machine learning and recommender systems. He holds an MSc and a PhD in Computer Science from Ghent University. He worked as a Visiting Scholar at the Academic College of Tel-Aviv/Yafo (Israel), the University of Technology Sydney (Australia) and the University of Granada (Spain).

Martine De Cock is an Associate Professor at the Department of Applied Mathematics and Computer Science of Ghent University (on leave) and a Visiting Associate Professor at the Institute of Technology of the University of Washington, Tacoma. Her research interests are in computational web intelligence. She holds an MSc and a PhD in Computer Science from Ghent University. She worked as a Visiting Scholar in the BISC group at the University of California, Berkeley (USA) and the Knowledge Systems Laboratory at Stanford University (USA).

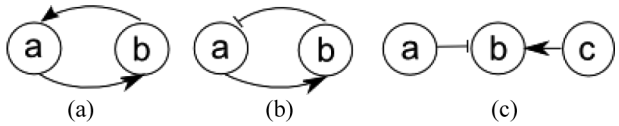
1 Introduction

Recent advances in molecular biology have led to a vast increase in experimental biological information. Integrating this information into a coherent model is an important task of systems biology. Nowadays, mathematical and computer science formalisms are widely adopted in this research domain to facilitate the modelling process.

The existing approaches (see, e.g., de Jong, 2002; Fisher and Henzinger, 2007, for good overviews) can roughly be divided into two groups: quantitative and qualitative ones. Typical quantitative models are built using differential equations. Such models require specific mathematical skills and a lot of experimental data to build, such as the concentration of different proteins over time, making their construction costly and time-consuming.

Qualitative models are used to analyse the system dynamics when there is a lack of experimental data, even though this has an influence on the model precision. However, it turns out that significant simplifications made in qualitative models, such as discrete timing and the absence of concentration dynamics, still allow one to model the behaviour of a system correctly. Discrete models are one of the cornerstones of computer science, thus, many attempts were made to adopt already developed techniques to the bioinformatics domain. Among them, discrete dynamical networks based on Boolean networks are one of the best-established qualitative modelling methods that are widely used by biologists to model protein regulatory networks (see, e.g., Albert, 2004; Davidich and Bornholdt, 2008; Mendoza et al., 1999). The nodes of a Boolean network represent protein molecules and the directed edges represent interactions. Edges can be typed to represent different kinds of interactions, such as inhibition and activation. For example, in Figure 1(a), proteins a and b activate each other. When at least one of the proteins is active at the initial state, the network settles in the state $\{act(a), act(b)\}$, i.e., both proteins will eventually become active.

Figure 1 Examples of Boolean networks



While liked for their simplicity, dynamical networks have the disadvantage of not being self-descriptive, i.e., they are built under some background assumptions that are not explicitly stated in the network itself. Furthermore, as will be explained in Section 2.2, their use requires the development of specific algorithms to retrieve the steady states of the network (see e.g., Garg et al., 2008), and different algorithms can cause different execution flows.

Moreover, dynamical networks provide little support for reasoning about network behaviour. As it was argued in Tran (2006), reasoning can leverage a biologist's experience and simplify tasks of model analysis and observation assimilation.

In this paper, we propose to represent gene and protein regulatory networks by answer set programs. Example 1 introduces our idea and illustrates the notation we will use throughout the paper.

Example 1: Program P_1, consisting of the rules G1–G6 and S1–S6, models the network in Figure 1(a). Rule labels, preceding the rules and separated from them by a colon ':', are introduced to refer to a particular rule as well as to distinguish general rules from specific ones. The specific rules or S-rules describe the structure of a particular network, in this case S1–S4, and initial conditions, in this case S5 and S6. The general rules or G-rules describe the semantics of the network, i.e., what 'activates' or 'inhibits' means in the context of our network. We refer to Section 3 for more details.

$$
\begin{array}{lll}
G1 : time(0..2). & & S1 : protein(a). \\
G2 : act(Y, T+1) & \leftarrow act(X,T), activates(X,Y,T). & S2 : protein(b). \\
G3 : inh(Y, T+1) & \leftarrow act(X,T), inhibits(X,Y,T). & S3 : activates(a,b,T). \\
G4 : & \leftarrow act(X,T), inh(X,T). & S4 : activates(b,a,T). \\
G5 : act(X, T+1) & \leftarrow act(X,T), not\ inh(X,T+1). & S5 : act(a,0). \\
G6 : inh(X, T+1) & \leftarrow inh(X,T), not\ act(X,T+1). & S6 : inh(b,0).
\end{array}
$$

As we will explain, the answer set of P_1 is $\{act(a,0), inh(b,0), act(a,1), act(b,1), act(a,2), act(b,2), act(a,3), act(b,3)\}$. Here, the predicate $act(a,0)$ means that protein a is active at time 0, and correspondingly, $inh(b,0)$ means that protein b is inhibited at time 0. In this answer set, there is no difference between the states at time steps 1 and 2, as both proteins remain active. Hence, we conclude that, under the given initial conditions, the steady state of the system is $\{act(a), act(b)\}$. As we will see later, the actual answer set of the program includes more information that is not relevant to our task. In our answer set representation, we omit this information for the sake of conciseness.

Note that in principle only network, specific rules such as rules S1–S6 need to be redefined for a given regulatory network, whereas the other rules model general biological properties. This makes the representation of such networks as answer set programs intuitively simple, while at the same time the ASP machinery becomes available to analyse and predict the behaviour of the described network at hand.

One of the main advantages of representing gene and protein regulatory networks by answer set programs is that all background information can be expressed explicitly in the program itself. This allows normalising differently expressed networks with one standard representation, thus avoiding the ambiguity of different descriptions. Furthermore, the use of ASP eliminates the need for specific network execution algorithms to retrieve the steady states of the networks. In fact, another main advantage of using ASP is that all supporting tools such as solvers and grounders are readily available. For the results described in this paper we used the *clingo* solver described in Gebser et al. (2007).

Our work is not the first attempt to use ASP to model biological networks. In Baral et al. (2004), Dworschak et al. (2008) and Tran (2006), the authors propose to use ASP-based action languages to model, query or plan the execution of biological systems. Our approach, however, is different from theirs in several

aspects. First of all, we do not use action languages; instead we propose a framework that models Boolean network semantics. Second, as we show in Section 6, our approach is less verbose because only the structure of the network needs to be described, and the semantics of the interactions is already defined in the framework itself.

This work is an extension of the study stated in Fayruzov et al. (2009) in which we proposed to model regulatory networks as answer set programs for the first time. Here, we provide a formal description of the approach we have developed and prove that this approach obtains correct results. Furthermore, we provide more detailed explanations of the framework along with additional examples. Moreover, we add an extra protein regulatory network example to showcase that our approach is easily scalable to other networks and present a modelling software tool that facilitates the modelling process for a biologist.

The remainder of this paper is structured as follows. We begin by recalling the necessary preliminaries about ASP and Boolean networks in Section 2. In Section 3, we explain in detail how to describe a regulatory network as an answer set program: we develop a framework of general rules, the so-called G-rules, that describe general Boolean network semantics, and we give examples of specific rules, the so-called S-rules, that allow the biologist to describe a specific network under study. Furthermore, in Section 4 we explain an efficient algorithm to solve the resulting answer set program, i.e., to find the steady states of the regulatory network while in Section 5 we describe a tool that we developed to facilitate the modelling process. In Section 6, we explain the relationship of our approach to the existing work and finally we conclude in Section 7.

2 Preliminaries

2.1 Answer Set Programming

Answer Set Programming (Gelfond and Lifschitz, 1988) is a declarative formalism that allows expressing relations between truth values of propositions with rules of the form $\alpha \leftarrow \beta$. Such a rule intuitively states that whenever β is true, proposition α should be true as well. The basic building blocks of answer set programs are constants, denoted by lower-case strings (e.g., a, b), that represent the entities; variables, denoted by upper-case strings (e.g., X, Y) that are substituted by constants during the program grounding stage and predicates (e.g., *protein(a), activates(a, X)*) that represent properties of, or relations between entities.

ASP allows two types of negation: classical negation denoted by $\neg$, and negation-as-failure (naf) denoted by not. For instance, rule G5 from Example 1 uses naf, stating that if protein X is active at time T and there is no evidence that X becomes inhibited at time $T+1$, then X should still be active at time $T+1$. On the other hand, if we would change naf by classical negation in G5 the rule will state that if protein X is active at time T and there is an evidence that X does not become inhibited at time $T+1$, then X should still be active at time $T+1$.

In general, a rule is of the form

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

in which $L_0, L_1 \ldots L_n$ are literals. A *literal* is a constant, variable or predicate that can be preceded with $\neg$. An expression of the form $not\ L$ is referred to as a naf-literal, whereas an expression of the form L is referred to as a positive literal. Expression of the form $pred/3$ denotes the arity of a predicate without an explicit reference to its arguments. Given a rule r of the form (1):

- The left-hand side of the rule is called the head, and defined as $head(r) = \{L_0\}$
- The right-hand side of the rule is called the body, and defined as $body(r) = \{L_1, \ldots, L_m, not\ L_{m+1}, \ldots, not\ L_n\}$
- The set of all positive literals in the body of the rule is defined as $pos(r) = \{L_1, \ldots, L_m\}$
- The set of all literals in the body of the rule preceded with not is defined as $neg(r) = \{L_{m+1}, \ldots, L_n\}$
- The set of all literals is defined as $lit(r) = head(r) \cup pos(r) \cup neg(r)$.

There exist two special kinds of rules: constraints and facts. A constraint is a rule with an empty head, such as rule G4 from Example 1. A rule with an empty body is called a fact. We usually write these as 'α.' instead of '$\alpha \leftarrow$.'. Rules S1–S6 are examples of facts.

Definition 1 (Answer set program)**:** A program P composed of rules of the form (1) is called an answer set program. Similar to the definitions on the rule level, the following shorthand notations are defined for a program:

$$pos(P) = \cup_{r \in P} pos(r) \quad lit(P) = \cup_{r \in P} lit(r)$$
$$neg(P) = \cup_{r \in P} neg(r) \quad head(P) = \cup_{r \in P} head(r)$$

At solving time, all variables in a rule r are instantiated with all possible constants. This process is called grounding. An answer set program that contains only grounded rules is called a grounded program.

Example 2: Consider the following program

$$somePred(p).$$
$$protein(a).$$
$$protein(b).$$
$$act(Y) \leftarrow activates(X, Y), protein(X), protein(Y).$$

The grounding of this program will be as follows

$$somePred(p).$$
$$protein(a).$$
$$protein(b).$$
$$act(a) \leftarrow activates(a, a), protein(a), protein(a).$$
$$act(a) \leftarrow activates(b, a), protein(b), protein(a).$$
$$act(b) \leftarrow activates(a, b), protein(a), protein(b).$$
$$act(b) \leftarrow activates(b, b), protein(b), protein(b).$$

Note that constant p was not associated with variables X and Y, because we put a restriction on the variables that they should occur as an argument of the $protein$ predicate. This predicate is called a domain predicate, and variables X and Y are said to be domain restricted.

To define what it means for a program to be solved, we recall the concepts of consistency, interpretation and satisfiability. A set of positive grounded literals S is said to be consistent if it does not contain the literals a and $\neg a$ together. An interpretation of a grounded program P is any consistent subset $I \subseteq pos(P)$.

Definition 2 (Satisfiability)**:** An interpretation I is said to satisfy a grounded rule with a non-empty head like (1), if $\{L_1 \dots L_m\} \subseteq I$ and $\{L_{m+1} \dots L_n\} \cap I = \emptyset$ imply that $L_0 \in I$. When the head is empty, interpretation I is said to satisfy the rule if $\{L_1 \dots L_m\} \not\subseteq I$ or $\{L_{m+1} \dots L_n\} \cap I \neq \emptyset$.

In other words, to satisfy a rule with a non-empty head, an interpretation I should contain the head when all positive literals and none of the literals with naf-prefix in the rule body are contained in I. Otherwise, if the head is empty, the interpretation should break the body conditions, i.e., either it does not contain some of the positive literals or it contains literals that have a naf-prefix in the body of the rule.

Definition 3 (Minimal model)**:** Any interpretation I that satisfies all rules of a program P is called a model of P. A model I is called a minimal model of P iff there is no model K such that $K \subset I$.

On the basis of this notion of a model, we can define answer sets. First, we consider the case of naf-free programs and then we proceed to programs containing negation-as-failure. First of all, for a program P without negation-as-failure, an answer set of P is a minimal model of P.

Example 3: Consider the program consisting only of a rule $p \leftarrow q$. The possible models of this program are $\{p, q\}$, $\{p\}$ and $\emptyset$. Indeed, if q is in the model, then p should be in the model because of the satisfiability definition; if there is no information on q, then we can make any conclusion about p, which means that p can be present or absent. The minimal model of the program is $\emptyset$, hence the unique answer set of this program is $\emptyset$.

The concept of an answer set is extended for programs containing negation-as-failure as follows. Suppose that an interpretation I is a model of a program P (with negation-as-failure), and our hypothesis is that I is an answer set of P. Then we first transform P into a naf-free program P^I with respect to the hypothesis I and solve this program as explained earlier. More formally,

Definition 4 (Gelfond-Lifschitz transformation) (Gelfond and Lifschitz, 1988)**:**
Let P be a ground answer set program. For an interpretation I, let P^I be the program called a reduct program obtained from P by deleting (1) all rules that contain a naf-literal $not\ L$ with $L \in I$ and (2) all the naf-literals from the bodies of the remaining rules.

Definition 5 (Answer set)**:** Given a program P, any interpretation I that is a minimal model of the reduct program P^I built from P is an answer set of the original program P.

Example 4: A program with negation-as-failure can have more than one answer set. Suppose that we have one seat and two persons p and q, and we want to assign the seat to a person. We can model this by the following program P

$$seat(p) \leftarrow not\ seat(q).$$
$$seat(q) \leftarrow not\ seat(p).$$

It has two answer sets $S_1 = \{seat(p)\}$ and $S_2 = \{seat(q)\}$. Indeed, by applying the Gelfond-Lifschitz transformation we can obtain the reduct program P^{S_1} consisting of the only rule $seat(p) \leftarrow$. The second rule is removed in the first step of the transformation, and the body of the first rule is removed in the second step. P^{S_1} is a naf-free program, and has a unique answer set $\{seat(p)\}$. This answer set coincides with our hypothesis S_1, which means that it is an answer set of the initial program P. One can verify in a similar way that S_2 is an answer set too.

We denote the set of all answer sets of a program *P* as *AS(P)*. Two programs P_1 and P_2 are said to be equivalent if they produce the same answer sets, i.e., if $AS(P_1) = AS(P_2)$.

2.2 Dynamical networks

A dynamical network of protein interactions, such as a Boolean network, captures interactions between proteins 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 proteins whereas the edges represent interactions between proteins. Pointed edges are used to represent activation and blunt edges are used to represent inhibition. At any given time, a protein or node is in one of the two states: either it is active, denoted by 1, or it is inhibited, denoted by 0. The state of a protein interaction network at any given time is defined in terms of the states of its nodes.

Definition 6 (Network state)**:** Let $G = (V, E)$ be a graph representing a protein interaction network. Then, a mapping $S : V \rightarrow \{0, 1\}$, which maps every protein in V to a protein state in $\{0, 1\}$, is called a network state. We use $\mathbb{S}_G$ to denote the set of all possible network states of a protein interaction network represented by a graph G.

Every node has *input nodes* that are determined by the inbound edges, and *output nodes* that are determined by the outbound edges of the node. For example in Figure 1(a), node *b* is at the same time an input and output node for node *a*. For every node in the network, a deterministic transition function 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 such update functions on its nodes. The update can occur *synchronously* (all elements are updated simultaneously) or *asynchronously* (one or several nodes are updated at

once). In this paper, we only consider synchronous updates, which allows us to consider a transition function acting on the network as a whole.

Definition 7 (Transition function)**:** Let $G = (V, E)$ be a graph representing a protein interaction network. Then, a mapping $f : \mathbb{S}_G \to \mathbb{S}_G$, which maps every network state to a network state is called a transition function of G.

For notational convenience, we use $f^{(k)}(S)$ to denote the state that is obtained after applying transition function f to an initial state $S \in \mathbb{S}_G$ k times, e.g. $f^{(0)}(S) = S$, $f^{(1)}(S) = f(S)$, $f^{(2)}(S) = f(f(S))$, etc.

Definition 8 (Trajectory)**:** Let $G = (V, E)$ be a graph representing a protein interaction network, f be a transition function of G, and $k \geq 1$, then a sequence $[S, f(S), \ldots, f^{(k)}(S)]$ with $S \in \mathbb{S}_G$ is called a trajectory of the network.

Definition 9 (Steady state, steady cycle)**:** A state S of a network is called *a steady state* w.r.t. a transition function f iff $S = f(S)$. A trajectory $[f^{(m)}(S), \ldots, f^{(n)}(S)]$ with $m < n$ is called a steady cycle iff $f^{(m)}(S) = f^{(n)}(S)$.

Note that once a steady state or cycle has been reached there is no point to calculate the trajectory further, because no new states can be obtained due to the deterministic nature of the transition function.

Example 5: Let us consider the network in Figure 1(a). The mapping for the network states is given in Table 1(a), and the transition function is defined in Table 1(b). Let us define the initial state of the network as S_1, then the next state is $f(S_1) = S_3$. To obtain the following state, we apply the transition function once again: $f(f(S_1)) = f(S_3) = S_3$. The state does not change on this step, which means that we encountered a steady state. The trajectory we have computed is $\mathcal{T} = [S_1, S_3, S_3]$.

Table 1 The network states and the transition function for the network in Figure 1(a)

S	a	b
S_0	0	0
S_1	0	1
S_2	1	0
S_3	1	1

(a)

S	$f(S)$
S_0	S_0
S_1	S_3
S_2	S_3
S_3	S_3

(b)

A set of trajectories that reach the same steady state or cycle is called a *basin of attraction*. For example, let us consider trajectory $\mathcal{T}' = [S_2, S_3, S_3]$ of the network from Figure 1(a). The set of trajectories $\{\mathcal{T}, \mathcal{T}'\}$ is a basin of attraction for the state S_3 of the network in Figure 1(a).

3 Building a network model in ASP

In this section, we set up the framework for describing gene and protein regulatory networks as answer set programs. We begin with a detailed explanation of the S-rules and G-rules of P_1 in Example 1. Next, we deal with issues such as conflicts and self-degradation that do not occur in the network of Figure 1(a) but might manifest themselves in other interaction networks.

3.1 Describing entities and their influences

The first step in describing a protein network is to introduce the network structure, cfr. rules S1–S6 in Example 1. Rules S1 and S2 define the proteins in the network, rules S5 and S6 define the initial state of these proteins, while rules S3 and S4 describe activation interactions between proteins. We add the extra time parameter T in these predicates to be able to model the dynamical network structure. Some interactions can be affected by external factors and be present or absent at different time points.

By themselves, these rules (facts) do not model anything yet; although they define the connection between proteins, they do not describe the influence of these connections on the proteins at different time steps. To this end, we introduce the G-rules. First of all, rule G1 is merely a shorthand for

$$time(0).\ time(1).\ time(2)$$

to introduce time steps into the program. Rules G2 and G3 define the actual semantics of the activation and inhibition concepts. Rule G4 of program P_1 is a constraint that expresses that a protein cannot be active and inhibited at the same time. Indeed, recall that to satisfy a constraint, at least one of the body conditions needs to be broken. Rules G5 and G6 are inertia rules that express what happens to a protein when there is no environmental influence: at the next time step, a protein stays in the same state unless its state was changed.

Once we have described the problem using ASP, the grounder is used to substitute variables with all possible constants. The activation rule G2, for example, says that protein Y will be active at time step $T+1$ if protein X is active and there is an activating connection between X and Y at the previous time step. When grounded, this rule will result in the following rules:

$$act(b,1) \leftarrow act(a,0), activates(a,b,0).$$
$$act(b,2) \leftarrow act(a,1), activates(a,b,1).$$
$$act(b,3) \leftarrow act(a,2), activates(a,b,2).$$
$$act(b,1) \leftarrow act(b,0), activates(b,b,0).$$
$$\ldots$$

In the programs presented in this paper, we omit domain predicates for the sake of brevity. In a real program, every rule that contains variable T would additionally contain the predicate $time(T)$, and every rule that contains at least one of the variables X, Y and Z would contain a corresponding predicate $protein(X)$, $protein(Y)$ and $protein(Z)$ in its body.

After the grounding has been done, the task of an ASP solver is to find an answer set of the ground program. In our application scenario, an answer set contains a sequence of protein states for each time point (see e.g., Example 1). Therefore, we can retrieve the steady state of the network w.r.t the transition function implicitly defined by G-rules by looking at the protein states in an answer set at each time step. When the protein states in two consequent time steps do not change, a steady state has been reached. In Example 1, we reach the steady state at time point 1, because the protein states do not change after this point. Note that, even though the last time step in rule G1 in Example 1 is 2, the network evolution is computed up to time step 3 because the heads of the rules of program P_1 contain $T + 1$.

Example 6: If we change rule S4 in program P_1 to $inhibits(b, a, T)$., we obtain the network from Figure 1(b). The answer set of the resulting program P_2 is $\{act(a, 0), inh(b, 0), act(a, 1), act(b, 1),\ inh(a, 2), act(b, 2), inh(a, 3), act(b, 3)\}$ (here and in the rest of the paper we omit from answer sets the predicates that are not essential for trajectory, e.g., $protein(a)$, $activates(a, b, 0)$, etc). Intuitively, this answer set can be explained as follows: all facts are in the answer set by definition, thus the predicates from rules S1–S6 are in the answer set; rules S3 and S5 trigger a ground version of rule G2 that causes the presence of $act(b, 1)$; rule G5 causes the presence of predicate $act(a, 1)$; the fact $act(b, 1)$ that is already in the answer set together with the new version of rule S4 triggers a ground version of rule G3, resulting in $inh(a, 2)$, and rules G5 and G6 result in the other predicates that are in the answer set.

From the above, we conclude that the steady state of this network is $\{inh(a), act(b)\}$.

3.2 Resolving conflicts

Rules G2–G4 might fail to work for more complex regulatory networks. Here we explain why they should be replaced with more refined rules, as well as supplemented by supporting rules.

Example 7: Let us consider the network in Figure 1(c). This network can be presented as follows

$$\begin{array}{lll} S1 : protein(a). & S4 : activates(c, b, T). & S6 : act(a, 0). \\ S2 : protein(b). & S5 : inhibits(a, b, T). & S7 : act(b, 0). \\ S3 : protein(c). & & S8 : act(c, 0). \end{array}$$

The program consisting of these facts S1–S8 together with the rules G1–G6 from Example 1 does not have an answer set under initial conditions S6–S8. Indeed, S5 together with S6 trigger rule G3, thus forcing *inh(b,1)* to be in the answer set. On the other hand, S4 together with S8 trigger rule G2, thus pushing *act(b,1)* to the answer set. However, due to constraint G4 both these predicates cannot be in the same answer set, thus the program does not have an answer set at all.

To resolve this conflict, we adopt a solution used in Davidich and Bornholdt (2008): if there are more incoming activation links than inhibition links, then the protein is active; if there are more inhibition links, then the protein is inhibited; if their number is equal, then the protein keeps the previous state. To implement this, we need to adjust the constraint as well as the activation and inhibition rules. The superscript in the rule labels here denotes the version of the rules; the compound numeration denotes the supporting rules for the main rule.

$$
\begin{aligned}
G2^1 : act(Y, T+1) &\leftarrow act(X, T), activates(X, Y, T), not\ conflict(Y, T).\\
G2.1 : act(Y, T+1) &\leftarrow conflict(Y, T), \#\_act(Y, A, T), \#\_inh(Y, I, T),\\
&\quad A - I > 0.\\
G3^1 : inh(Y, T+1) &\leftarrow act(X, T), inhibits(X, Y, T), not\ conflict(Y, T).\\
G3.1 : inh(Y, T+1) &\leftarrow conflict(Y, T), \#\_act(Y, A, T), \#\_inh(Y, I, T),\\
&\quad I - A > 0.\\
G4^1 : conflict(Y, T) &\leftarrow activates(X, Y, T), inhibits(Z, Y, T),\\
&\quad act(X, T), act(Z, T).
\end{aligned}
$$

The rules $G2^1$ and $G3^1$ say that if there is no conflict, then the old definitions work, but if there is a conflict (the body of rule $G4^1$ is satisfied), then we count the number of activation and inhibition links for the conflicting instance and make the decision based on this count (rules G2.1 and G3.1). The integrity constraint G4 we had before is now transformed to the definition of conflict (rule $G4^1$). It fires only if there are $inh/3$ and $act/3$ links on the protein and both can be executed at the current time point. The definition of the $\#\_act/3$ and $\#\_inh/3$ predicates is omitted here, but can be found in Fayruzov (2009).

This set-up already allows modelling fairly complex interaction networks, such as the Budding Yeast network described in Li et al. (2004). We describe the model of this network in Fayruzov (2009).

3.3 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. Let us return to Figure 1(c). Under the current definitions, protein b does not change its state when both a and c are active, i.e., if b is active it remains active. Imagine now that we want to modify the behaviour of b to change its sensitivity to the activating or inhibiting influence such that it requires less effort (less activation/inhibition inputs) to change the state of the protein. This requirement can be implemented in the system by introducing inhibition/activation thresholds.

$$G2^2 : act(Y,T+1) \leftarrow act(X,T), activates(X,Y,T), not\ conflict(Y,T), not\ mod\_act\_th(Y).$$
$$G2.1^1 : act(Y,T+1) \leftarrow conflict(Y,T), act\_th(Y,Th), \#\_act(Y,A,T), \#\_inh(Y,I,T), A-I>Th.$$
$$G2.2 : act(Y,T+1) \leftarrow act\_th(Y,Th), Th \neq 0, \#\_act(Y,A,T), \#\_inh(Y,I,T), A-I>Th.$$
$$G3^2 : inh(Y,T+1) \leftarrow act(X,T), inhibits(X,Y,T), not\ conflict(Y,T), not\ mod\_inh\_th(Y).$$
$$G3.1^1 : inh(Y,T+1) \leftarrow conflict(Y,T), inh\_th(Y,Th), \#\_act(Y,A,T), \#\_inh(Y,I,T), I-A>Th.$$
$$G3.2 : inh(Y,T+1) \leftarrow inh\_th(Y,Th), Th \neq 0, \#\_act(Y,A,T), \#\_inh(Y,I,T), I-A>Th.$$
$$G7 : act\_th(X,0) \leftarrow not\ mod\_act\_th(X).$$
$$G7.1 : mod\_act\_th(X) \leftarrow act\_th(X,Th), Th \neq 0.$$
$$G8 : inh\_th(X,0) \leftarrow not\ mod\_inh\_th(X).$$
$$G8.1 : mod\_inh\_th(X) \leftarrow inh\_th(X,Th), Th \neq 0.$$

Rules G4^1, G5 and G6 are omitted because they do not change. We replace G2^1 and G3^1 by G2^2 and G3^2, respectively, so that now they take into account the possible presence of a threshold. Conflict resolving rules G2.1 and G3.1 are changed in the same manner, and a comparison of the inhibition and activation influence is made against the threshold now. If there is no conflict, but an activation or inhibition threshold is imposed, we follow rules G2.2 and G3.2. Rules G7 and G8 set the activation and inhibition threshold of every protein to 0 in case it was not set explicitly by a special S-rule (G7.1 and G8.1). If the threshold values are not modified, the G-rules described earlier will lead to the same answer sets as ones in Section 3.2. Having both inhibiting and activating thresholds instead of one threshold is not redundant, since these thresholds characterise not the 'on/off' level of the protein, but rather an effort that is needed to change its state. The thresholds can be viewed as tolerance degrees of a protein to a corresponding input. Positive values make the protein more tolerant and negative ones make it less tolerant. The default value can be altered by a specific rule as illustrated in Example 9.

Example 8: Let P_3 be the answer set program consisting of general rules G1, G2^2, G2.1^1, G2.2, G3^2, G3.1^1, G3.2, G4^1, G5, G6, G7, G7.1, G8, G8.1 and the specific rules from Example 7. The activation and inhibition thresholds of these proteins are not explicitly defined; hence, they are automatically set to the default value. The answer set of this program is $\{act(a,0), act(b,0), act(c,0) act(a,1), act(b,1), act(c,1), act(a,2), act(b,2), act(c,2), act(a,3), act(b,3), act(c,3)\}$. 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 $\{act(a), act(b), act(c)\}$.

Example 9: For the network in Figure 1(c), let us explicitly set the inhibition threshold of b to -1 to indicate that this protein is susceptible to inhibition. In other words, let P_4 be the answer set program containing all the rules from P_3 as well as the additional *S9: inh_th(b,-1)*. The answer set of this program is $\{act(a,0), act(b,0), act(c,0), act(a,1), inh(b,1), act(c,1), act(a,2), inh(b,2), act(c,2), act(a,3), inh(b,3), act(c,3)\}$. The steady state in this case is $\{act(a), inh(b), act(c)\}$.

The phenomena of self-activation and self-degradation can also be modelled by adjusting activation and inhibition thresholds. Self-activation/degradation means that a protein is able to change its state when no external influence is applied. Let us consider the following example:

Example 10: Let P_4 be the answer set program containing all the rules from the program from Example 9, but we replace rules S6–S8 with the following:

$$S6 : inh(a,0).$$
$$S7 : inh(b,0).$$
$$S8 : inh(c,0).$$

to indicate that all proteins in Figure 1(c) are initially inhibited. Furthermore, we add the additional rule *S9: act_th(b,-1).* to indicate that b is susceptible to activation. According to rule G2.2, in this case protein b activates itself when no inhibition influence is applied, i.e., self-activation takes place. The answer set of program P_4 is $\{inh(a,0), inh(b,0), inh(c,0), inh(a,1), act(b,1), inh(c,1), inh(a,2), act(b,2), inh(c,2), inh(a,3), act(b,3), inh(c,3)\}$. The steady state in this case is $\{inh(a), act(b), inh(c)\}$.

3.4 Starting conditions

By writing S-rules, a user can model various networks and observe their behaviour under certain initial conditions. This requires the user to consequently set various initial protein activation combinations and analyse the results of each execution. On large networks with tens of proteins, the number of different possible combinations is very high, which makes this task very tiresome. To automate this process, we introduce two additional general rules that deal with different initial condition combinations:

$$G9 : act(X,0) \leftarrow not\ inh(X,0).$$
$$G10 : inh(X,0) \leftarrow not\ act(X,0).$$

These rules force the solver to make a choice for each protein: either it is active at the initial time point, or inhibited. In this way, different answer sets are automatically generated for each possible combination of active and inhibited proteins, which decreases the need for manual input of the user drastically.

4 Efficient network modelling

Each network has 2^N possible states, where N is the number of proteins in the network. As one can see for instance in Example 6, in the answer set representation, the state of a protein corresponds to a predicate in the set, and a network state corresponds to a subset that contains all protein states for one timepoint. For example, $\{act(a,1), act(b,1)\}$ is a part of the answer set for the program in Example 6 that describes the state of the network at timepoint 1. There are two problems associated with the approach as proposed in Section 3. First of all, when computing a trajectory for a given state it is impossible to estimate how many time steps are needed (the upper time bound). Trajectories within too short time intervals may not reach the steady state, whereas too long intervals increase computational expenses. For instance, if we limit the upper time bound in Example 6 by 1 (set rule $G1$ as $time(0..1)$.) the steady state will not be present in the answer set. On the other hand, if we set the upper time bound to, e.g., 5, we will find a steady state but at the same time we will compute 3 extra states that do not contain any additional information about the network behaviour.

Another problem, which follows from the first one, is that we cannot compute attraction basins efficiently. We can iterate over all possible initial states, but for every initial state we need to adjust the time interval every time, which, again, makes the whole process extremely computationally inefficient.

In Fayruzov et al. (2010), we introduced a method to efficiently solve time-dependent answer set programs like the ones resulting from the approach proposed in Section 3. To this end, we introduced the notion of *Markovian programs* and provided a *temporal algorithm* to solve these programs efficiently without a reference to a specific upper time bound.

A *time-dependent predicate* is a predicate whose last argument represents time. Examples of time-dependent predicates in the framework proposed in Section 3 are *activates, inhibits, active, inhibited, conflict, #_act* and *#_inh*. A *Markovian program* is an answer set program that satisfies the following condition: every rule with a time-dependent predicate in its head that depends on time T contains only time-dependent predicates that depend on T or $T-1$ in its body. In other words, the next state of the model depends only on the previous state or is determined with respect to other components in the current state of the model. Note that there are no specific constraints on rules that do not contain time-dependent predicates. It is easy to see that the framework we have built in Section 3 conforms to the definition of a Markovian program, which means that we can use the temporal algorithm described in Fayruzov et al. (2010) to analyse the behaviour of the models built in our framework.

The main idea behind the temporal algorithm is that instead of solving the answer set program for some long interval $\{0,\ldots,t_{\max}\}$ we consecutively solve smaller programs for intervals $\{0,1\}$, $\{1,2\},\ldots,\{t_{\max}-1,t_{\max}\}$, which can be done more efficiently. In Fayruzov et al. (2010), we have shown that by doing so we obtain the same answer sets as by solving the initial program for interval $\{0,\ldots,t_{\max}\}$. Moreover, we can stop the solving process at every moment, as soon as we encounter a steady state or cycle. For more details of how these smaller programs are defined, and for a more complete description of the algorithm, we refer to Fayruzov et al. (2010).

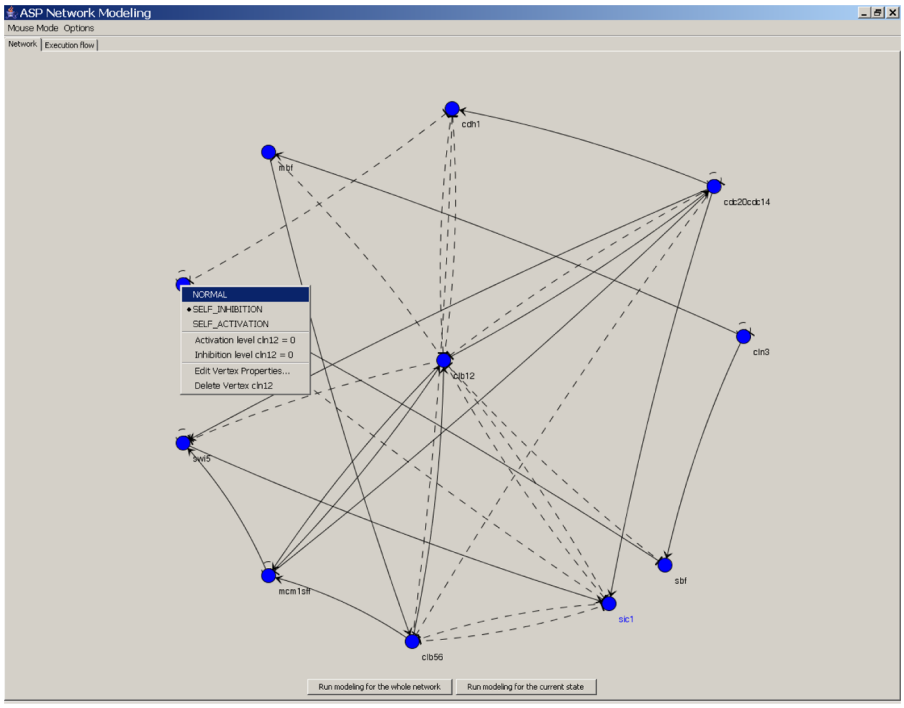
5 Modelling software

In Section 3, we explained how to model a regulatory network with ASP. However, the more practical question is: how can a biologist benefit from this framework? Does she or he need to know anything about ASP in order to use it?

To facilitate the use of the framework by biologists, we have developed a Java-based GUI tool. This tool provides an interactive interface and performs translations to ASP and back automatically. No intervention from the user's side is required in this translation process. This means that the tool can be used by a biologist with no background knowledge in ASP. In fact, it is possible to use the tool while remaining completely unaware of the underlying ASP model, i.e., while operating only at a conceptual level of genes, proteins and interactions.

The interactive interface consists of two tabs. The first tab, presented in Figure 2, is used to build graphical network models and to set node properties. This figure shows the Budding Yeast network, with nodes representing proteins and edges representing interactions between them. There are two possible ways to build a network. The first option is interactive editing in which a user can add or delete nodes and edges, indicate self-activation and self-inhibition, edit node names and define activation and inhibition thresholds. The initial state of the network can also be set by switching the nodes on/off. The second option is to write a network in ASP and load it from a file; it can still be edited interactively after that.

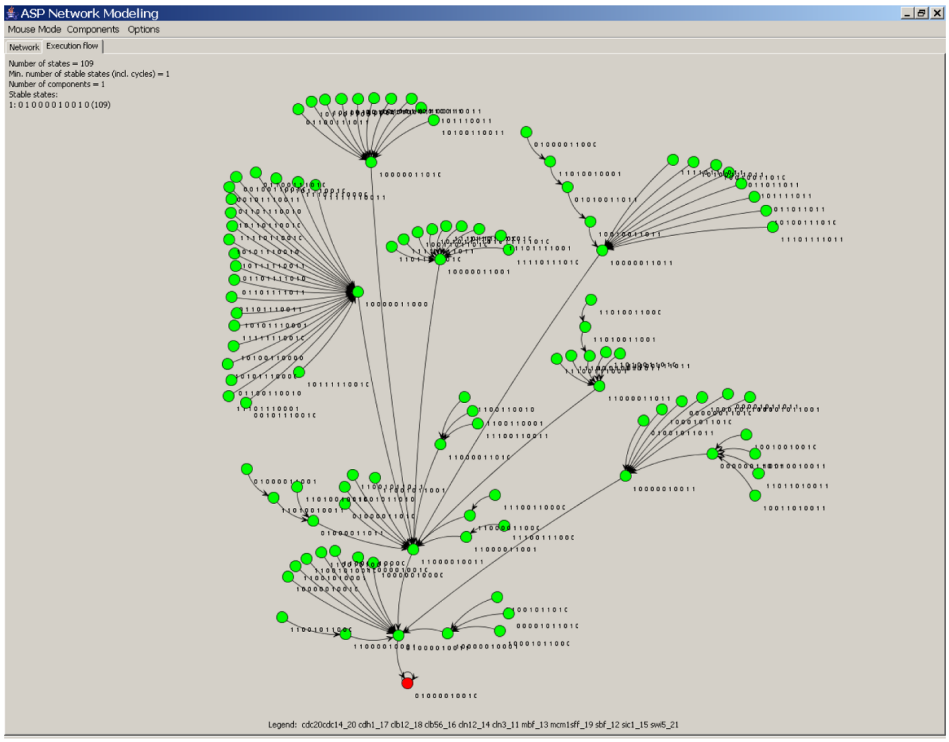
Figure 2 Graphical user interface with the Budding Yeast network model: (a) positive feedback loop; (b) negative feedback loop and (c) conflict (see online version for colours)



After building a network, a user can start the modelling process, which again can be performed in two scenarios. The first scenario is to build complete network state transition charts at once. Here, every possible state of the network is analysed and its steady states or cycles are computed. Another scenario is when a user wants to know the trajectory of the system from one particular state. In this case, the user can change the default state of the network by switching certain proteins on/off. Then, only the attractor for this state is computed.

During this phase, our application translates the graph built by the user to an ASP program, and uses *clingo* to ground and solve the program. The answer sets are translated back into a graphical representation as shown in Figure 3. This tab represents network trajectories, and in the case of the first scenario it can show all attraction basins together or separately. The graph shown in the figure was computed within the first scenario and represents one attraction basin of the Budding Yeast network. Here, a node represents a network state, and an edge between nodes represents a transition of the network from one state to another. The steady state of the network (marked by a colour node on the graph, the bottom one in this case) and the size of the attraction basin is shown in the top-left corner. Each node has a corresponding network state printed next to it.

Figure 3 An attraction basin of the Budding Yeast networks (see online version for colours)



We have applied our tool to model the Budding Yeast network presented in Li et al. (2004) and the Fission Yeast network described in Davidich and

Bornholdt (2008). The details and complete answer set programs can be found in the technical report (Fayruzov, 2009). More information on the application of our approach to the Fission Yeast network is also available in Fayruzov et al. (2009).

6 Related work

A detailed analysis of related approaches to model biological networks is provided in Tran (2006). Many of the approaches proposed to model regulatory networks such as Peleg et al. (2002), Ciocchetta and Hillston (2008) and Regev et al. (2001) follow a simulation–perturbation strategy, i.e. to analyse the behaviour of a system, the biologist changes the input and observes the changes in the output. Even though the structure of the system is known, it still works as a blackbox in the sense that the solution cannot be explained. As a result, many computationally expensive experiments may be needed to explain some properties of the model. ASP-based methods aim to overcome this limitation as they provide methods for system analysis such as prediction and planning as it is argued in Tran (2006).

To the best of our knowledge, all existing approaches that adopt ASP to model biological network behaviour are based on the concept of action languages. An action language is a high-level language with a simple intuitive syntax that allows describing the states of the world and effects of actions on these states (Gelfond and Lifschitz, 1992). The statements and queries in this language are then translated to standard ASP syntax and executed to find the answer sets.

The first attempt to model biological processes by means of ASP was made by Baral et al. (2004) where they present a BioSigNet-RR system that allows representing and reasoning about signal networks. In this paper, they define an action language that allows expressing the structure of a signal network (they use the $NF\kappa B$ network as an example) as well as the means to query the network (does protein a bind to b?), plan the execution (what sequence of actions should be taken to make a active?) and explain the results (given that a is active at a certain point, what was the initial state?). This work was further extended by Dworschak et al. (2008) who introduced several extra concepts such as *allowance* and *noconcurrency*, as well as a special language for biological modelling called C_{TAID}.

It is important to emphasise the difference between the above-described action-language-based approaches and the approach we propose here. The former requires that for each model the whole program is built from scratch and every interaction is defined as its own rule. In other words, the framework describes only the description language, and the biologist has to describe every interaction separately. In our setting, we go one step further, and provide the biologist with a background theory based on a Boolean network model semantics. For example, let us consider an example where protein b is activated by a and c and inhibited by d. In the default Boolean network semantics, if the number of activating links is higher than the number of inhibition links, then protein b would be activated; if it is lower, it would be inhibited; if it is equal, the state remains unchanged. To express

this with action languages, we need 2 actions: *activating_prot_b*, *inhibiting_prot_b*, and the following set of statements:

$$activating\_prot\_b \ causes \ b$$
$$inhibiting\_prot\_b \ causes \neg b$$
$$a, \neg d \quad causes \ activation\_prot\_b$$
$$c, \neg d \quad causes \ activation\_prot\_b$$
$$a, c \quad causes \ activation\_prot\_b$$
$$d, \neg a, \neg c \quad causes \ inhibiting\_prot\_b$$

The number of required rules grows combinatorially with the number of interactions.

In the framework that we propose, only factual input is needed from the biologist: *activates(a,b), activates(c,b)* and *inhibits(d,b)*. The Boolean network semantics implemented in the framework will do all other necessary inference. Moreover, as ASP offers non-monotonic inference, the existing semantics can be extended to handle exceptions when needed, making the framework scalable.

Another interesting application of ASP in the biological domain was recently proposed in Schaub and Thiele (2009). Here, the authors use ASP to expand metabolic networks and check if the network can be completed to reach a certain state. In contrast to the previously described approaches, this method does not aim to study system evolution over time.

Finally, in Section 4 we proposed a method to efficiently solve ASP programs built to model network behaviour. However, this is not the only way to deal with the problem. Gebser et al. (2008) described another theoretical formalism, which is called incremental program solving, and provided a description of a specially constructed solver *iclingo* that allows solving incremental programs. Although this approach is proposed in a general setting, it fits well to the network modelling problem. While duplicating the functionality of our framework, the incremental program approach has an important limitation. A network can potentially converge to different steady states from one starting state, while an incremental program would terminate after finding the first steady state and disregard the other ones. The temporal algorithm, on the other hand, allows us to compute all possible steady states.

7 Conclusions and future work

In this paper, we have proposed a modelling and simulation tool for gene and protein regulatory networks based on ASP. In the user interface, a biologist can build a network of proteins and specify the properties of the elements in a straightforward and intuitive way. Use of the tool does not require any formal logic knowledge from the biologist, who can operate with predefined concepts to build a model. At the same time, this approach has the advantage of being more expressive compared with Boolean networks, since all implicit assumptions and background knowledge can explicitly be described in the body of the program. Moreover, due to the non-monotonic nature of ASP, the framework is scalable,

i.e., it can be extended with non-typical cases that are not considered in the current version.

The constructed network is then automatically translated to an answer set program that can be solved with any of the off-the-shelf solvers to produce steady states of the network. In this paper we proposed a theoretically justified algorithm that allows a more efficient network computation.

One of the limitations, inherited from Boolean networks, is that every action should happen within one time step. In other words, it is impossible to model slow and fast interaction; everything is aligned to the same speed. Another problem is that in the current version of the system only synchronous execution is supported. We plan to address these issues in the near future.

References

Albert, R. (2004) 'Boolean modeling of genetic regulatory networks', *Complex Networks*, Springer, pp.459–481.

Baral, C., Chancellor, K., Tran, N., Tran, N., Joy, A. and Berens, M. (2004) 'A knowledge based approach for representing and reasoning about signaling networks', *Bioinformatics*, Vol. 20, No. 1, pp.15–22.

Ciocchetta, F. and Hillston, J. (2008) 'Process algebras in systems biology', *Formal Methods for Computational Systems Biology*, Springer, pp.313–365.

Davidich, M.I. and Bornholdt, S. (2008) 'Boolean network model predicts cell cycle sequence of fission yeast', *PLoS ONE*, Vol. 3, No. 2, p.e1672.

de Jong, H. (2002) 'Modeling and simulation of genetic regulatory systems: A literature review', *Journal of Computational Biology*, Vol. 9, No. 1, pp.67–103.

Dworschak, S., Grell, S., Nikiforova, V.J., Schaub, T. and Selbig, J. (2008) 'Modeling biological networks by action languages via answer set programming', *Constraints*, Vol. 13, Nos. 1–2, pp.21–65.

Fayruzov, T. (2009) *Technical Report 2009-01*, Technical Report, Ghent University. http://www.cwi.ugent.be

Fayruzov, T., De Cock, M., Vermeir, D. and Cornelis, C. (2009) 'Modeling protein interaction networks with answer set programming', *2009 IEEE International Conference on Bioinformatics and Biomedicine*, pp.99–105.

Fayruzov, T., Janssen, J., Vermeir, D., Cornelis, C. and De Cock, M. (2010) 'Efficient solving of time-dependent answer set programs', in Hermenegildo, M. and Schaub, T. (Eds.): *Technical Communications of the 26th Int'l. Conference on Logic Programming*, July, Dagstuhl, Germany, pp.64–73.

Fisher, J. and Henzinger, T.A. (2007) 'Executable cell biology', *Nature Biotechnology*, Vol. 25, No. 11, pp.1239–1249.

Garg, A., Cara, A.D., Xenarios, I., Mendoza, L. and Micheli, G.D. (2008) 'Synchronous versus asynchronous modeling of gene regulatory networks', *Bioinformatics*, Vol. 24, No. 17, pp.1917–1925.

Gebser, M., Kaminski, R., Kaufmann, B., Ostrowski, M., Schaub, T. and Thiele, S. (2008) 'Engineering an incremental asp solver', *ICLP'08: Proceedings of the 24th International Conference on Logic Programming*, pp.190–205.

Gebser, M., Kaufmann, B., Neumann, A. and Schaub, T. (2007) 'Clasp: a conict-driven answer set solver', *LPNMR*, pp.260–265.

Gelfond, M. and Lifschitz, V. (1988) 'The stable model semantics for logic programming', *ICLP/SLP 1988*, pp.1070–1080.

Gelfond, M. and Lifschitz, V. (1992) 'Representing actions in extended logic programming', *ICLP/SLP 1992*, pp.559–573.

Li, F., Long, T., Lu, Y., Ouyang, Q. and Tang, C. (2004) 'The yeast cell-cycle network is robustly designed', *PNAS*, Vol. 101, pp.4781–4786.

Mendoza, L., Thieffry, D. and Alvarez-Buylla, E.R. (1999) 'Genetic control of flower morphogenesis in arabidopsis thaliana: a logical analysis', *Bioinformatics*, Vol. 15, No. 7, pp.593–606.

Peleg, M., Yeh, I. and Altman, R.B. (2002) 'Modelling biological processes using workflow and petri net models', *Bioinformatics*, Vol. 18, No. 6, pp.825–837.

Regev, A., Silverman, W. and Shapiro, E.Y. (2001) 'Representation and simulation of biochemical processes using the pi-calculus process algebra', *Pacific Symposium on Biocomputing 2001*, pp.459–470.

Schaub, T. and Thiele, S. (2009) 'Metabolic network expansion with answer set programming', *ICLP'09: Proceedings of the 25th International Conference on Logic Programming*, Pasadena, California, USA, pp.312–326.

Tran, N. (2006) *Reasoning and Hypothesising about Signaling Networks*, PhD Thesis, Arizona State University, Phoenix, Arizona, USA.