ASP for Construction and Validation of Regulatory Biological Networks

Alexandre Rocca, Nicolas Mobilia, Eric Fanchon, Tony Ribeiro, Laurent Trilling, Katsumi Inoue

ASP for Construction and Validation of Regulatory Biological Networks

Alexandre ROCCA, Nicolas MOBILIA, Éric FANCHON, TonyRIBEIRO, Laurent TRILLING and Katsumi INOUE

In this chapter, we present a declarative approach for analyzing and building genetic regulatory networks (GRNs). A declarative approach is not restricted to check properties of a fixed network, but aims at logically specifying networks that satisfy a given set of constraints. Here, we cover two aspects of this approach. First, we propose an implementation of a model checker for linear temporal logic (LTL) and computational tree logic (CTL) formulas in answer set programming (ASP), a logic programming language based on the stable model semantic. CTL formulas are well suited for specifying constraints implied by the dynamic behaviors of GRNs. Second, we present a specification in ASP of Thomas GRNs that are both a generalization of Boolean networks and an approximation of piecewise linear differential equations. At the same time, we also show how to express biological data like interaction characteristics in our framework. Then, we propose a methodology for analyzing networks with a declarative approach, including consistency repairing and learning of properties from a set of consistent models. Our aim is not only to exhibit the importance of logic programming for our purposes, but also to point up the advantages of ASP, coming from non-monotonicity, minimality of models, expressiveness and practical performances. Finally, the provided functionalities and the proposed methodology are evaluated over three real biological applications.

5.1. Introduction

As biological knowledge is expanding very rapidly, computer methods and tools become essential to organize and take advantage of this knowledge. It is particularly

the case in systems biology, i.e. the study of interactions between the components of biological systems. Most of the studied systems are GRNs, which had been mainly modeled by either Boolean network representation [KAU 69], or its more recent extension: Thomas' networks [THO 01].

The study of biological models mainly follows two phases. The first one is the construction and validation phase. This is done by gathering knowledge on the biological system and by formulating hypotheses. Traditionally, this leads to a first (and unique) model. Then, this model is validated with respect to experimental data. When inconsistencies occur the model is revised (manually). Once a model consistent with the experimental data has been finally obtained, the second phase may begin: the analysis and "prediction". The idea is to either design new experiments that could refine the model, or use the model to simulate the real system and make predictions.

A precise formalism is needed for expressing experimental data in the model construction phase. Regarding biological behaviors, temporal logics are considered as an excellent way to describe them (Chapter 7 and [BER 04]). Among these logics, LTL [QUE 82] and CTL [CLA 82] appear especially suitable. They have been proposed originally for specifying computer program behaviors and they gave rise to model-checking algorithms that verify properties of computer programs expressed as temporal logical formulas.

In this chapter, we explore the interest of a non-monotonic logic programming paradigm, namely ASP [BAR 03], both for modeling GNRs and for expressing LTL and CTL formulas. Our aim is to avoid the iterative *generate and test* phase during the model construction described above by adopting a *declarative* approach. Instead of instantiating (manually) the successive biological models, all possible knowledge on the network structure and its dynamics (interactions between species, behaviors like stationary states or response to environmental perturbations) is expressed in terms of logical formulas or *constraints*. In cases where these constraints are satisfied, a set of models is obtained instead of one model, composed of those models that are solutions of the constraints. Then, building on these constraints, we can explore the properties of this set of consistent models, for example in order to exhibit good candidates for further experimentations.

If biological behaviors are expressed by temporal logical formulas, it is necessary within this new approach that these formulas should be used to assert behaviors as opposed to only verify behaviors within a model-checking perspective (like in Chapter 7). Translating LTL and CTL formulas into ASP formulas satisfy this issue. We will show that the specificity of ASP as a logic programming language contributes importantly to this purpose. This specificity comes from the semantics of an ASP program that is defined in terms of the so-called logical *stable models* or *answer sets* (ASs) that are, by definition, *minimal* in that sense that removing any component from such a logical model cannot provide a logical model. The implementation of the CTL

in particular is new, general and non-monotone based, if we compare with the previous ASP implementation [HEY 05].

For formalizing and analyzing Thomas networks, the advantages of ASP that we illustrate include consideration of partial knowledge, non-monotonic reasoning facilitating knowledge addition and the expression of default rules, automatic consistency repairing, inference and learning capabilities resulting from the minimality of the accepted models and computer performances similar to those of SAT solvers without the restricted linguistic power. We make the most of this last point by describing Thomas networks in ASP directly, thus providing both a logical and an executable specification. Note that a good and classical introduction to Thomas networks is given in this book (Chapter 7).

This work is the merging of two different works [ROC 13, COR 10], and thus will be divided in two main parts. In section 5.2, we will give some formal preliminaries about Boolean networks, Thomas networks and ASP. In section 5.3, we will give a quite general implementation of both LTL and CTL, followed by an example of Boolean network computation in ASP. In section 5.4.1, we will focus on the definition of Thomas networks in ASP. Then, in section 5.4.2, we will describe, in the first part, methods to model efficiently biological data in ASP and temporal logics, and how to express mutants. In the second part, in section 5.4.3, we will develop how ASP inference and learning capabilities allows us to construct models. Finally, in section 5.4.4, three real applications will be described. Sections 5.3 and 5.4 have their own discussion sections (sections 5.3.4 and 5.4.5), but a general conclusion is given in section 5.5. An appendix gives a refined and more efficient specification of Thomas networks that distinguishes well the two minimization ways (logical versus paralogical), which can be used to select models.

5.2. Preliminaries: ASP and biological logical networks

5.2.1. *Answer set programming*

ASP [BAR 03] is a computational language which appeared at the end of the 1990s. It is based on a non-monotonic logic defined with *stable models*. Here is a short presentation based on [GEB 10], which proposes the *gringo* language, also presented and used in Chapter 2.

A logical ASP program is a finite set of rules:

$$a_0 \leftarrow a_1, \ldots, a_m, not\ a_{m+1}, \ldots, not\ a_n.$$

where $0 \leq m \leq n$ and $\forall i \mid 0 \leq i \leq n,\ a_i$ is an atom. For any rule r, $head(r) = a_0$ is the head of the rule and $body(r) = \{a_1, \ldots, a_m, not\ a_{m+1}, \ldots, not\ a_n\}$ is the body

of the rule. If $head(r)$ is empty, r is called *integrity constraint*. If $body(r)$ is empty, r is a *fact*.

Let A be the set of atoms, $body^+(r) = \{a{\in}A \mid a \in body(r)\}$ and $body^-(r) = \{a{\in}A \mid not\ a \in body(r)\}$. A set $X \subseteq A$ is an *AS* or *stable model* of a program P if X is the minimal model of the *reduct* $P^X = \{head(r){\leftarrow}body^+(r) | r{\in}P, body^-(r) \cap X = \varnothing\}$.

Example 1: Let E be the following ASP program where $\leftarrow$ is represented by "`:-`":

```
a :- not b,c.
b :- not a.
c.
```

Let $X = \{\texttt{a},\texttt{c}\}$. The minimal model of the reduct $E^X = \{\texttt{c},\ \texttt{a} \leftarrow \texttt{c}\}$ is $\{\texttt{a},\texttt{c}\}$. X is a stable model of E.

Let $X' = \{\texttt{a},\texttt{b},\texttt{c}\}$. The corresponding reduct is $E^{X'} = \{\texttt{c}\}$ and the minimal model of the reduct is $\{\texttt{c}\}$. X' is not a stable model of E.

Example 2: Let E' be the following program:

```
a :- not b.
b :- not a.
```

E' has two stable models $\{\texttt{a}\}$ and $\{\texttt{b}\}$. To this program, if we add the integrity constraint `:- a.`, we remove the model $\{\texttt{a}\}$. If we add the integrity constraint `:- not a.`, we remove the model $\{\texttt{b}\}$ because it does not contain `a`.

The *gringo* language provides logical variables and functional terms in a limited way (so that the program can be transformed in an equivalent finite propositional one). It provides also cardinality constraints on the number of true literals. If we impose the constraint $u\{\texttt{l\_1},\texttt{...},\texttt{l\_n}\}v$, we obtain only models such that the number of true literals is bigger than u (0 by default) and smaller than v (`n` by default). Moreover, this formalism allows the expression of enumeration through the symbol "`:`". In the following program:

```
dom(0). dom(1).
all_true :- p(X):dom(X).
at_least_one_true :- 1{p(X):dom(X)}.
```

the second line expresses that if `p(0)` and `p(1)` hold, then `all_true` is deduced. The second line is, so, equivalent to the rule `all_true :- p(0), p(1).`: in the left part od a rule such an enumeration would be a disjunction of theses litterals.

The third line expresses that `at_least_one_true` is deduced if a least one among `p(0)` and `p(1)` holds. Finally a pooling facility allows, for example, to write `p(0;1)` instead of `p(0),p(1)`.

The *gringo* language also provides a paralogic operator `#maximize` (respectively `#minimize`) to maximize (respectively minimize) the number of atoms true among a specified list of atoms. For example, if we impose `#maximize{f_1,...,f_n}`, we obtain only models with the highest number of `f_i` true.

The solver [GEB 10] we use proceeds in two steps to compute the ASs of a program P. First, a "grounder" substitutes the variables of the program by terms without free variables, and consequently produces a propositional program $\mathbf{P}$ corresponding to P. In the second step, a solver computes the ASs of $\mathbf{P}$. This motivates the programmer to reduce as far as possible the number of resulting Boolean variables and rules subject to a big expansion.

5.2.2. *Boolean networks and Thomas networks*

A Boolean network is a simple discrete representation widely used in bioinformatics [KAU 69, KLA 06, LAH 03]. Initially introduced to represent gene regulatory networks [KAU 69], Boolean networks have also been used in many research fields to represent other Boolean interaction system such as robot design [ROL 11] or social interaction models [GRE 07]. A Boolean network [KAU 69] is a pair (N,F) with N = {n_1, ... , n_k} a finite set of nodes (or variables) and F = {f_1, ... , f_k} a corresponding set of Boolean functions; $n_i(t)$ represents the value of n_i at time step t and n_i takes either 1 (expressed) or 0 (not expressed). A vector (or *state*) $s(t)$ = ($n_1(t)$, ... , $n_k(t)$) is the expression of the nodes of N at time step *t*. There are 2^k possible distinct states for each time step. The state of a node n_i at the next time step $t+1$ is determined by $n_i(t+1)$=$f_i(n_{i_1}(t)$, ... ,$n_{i_p}(t))$, with n_{i_1}, ... ,n_{i_p} the nodes directly influencing n_i, and also called regulation nodes of n_i. Boolean networks are mainly represented in three different ways: the *interaction graph* (see Figure 5.1, top left), the *state transition graph* (see Figure 5.1, bottom), which represents the transitions between $n_i(t)$ and $n_i(t+1)$, and the Boolean functions (see Figure 5.1, top right). It can also be represented as a truth table, which is simply the transition graph. In the case of a gene regulatory network, nodes represent genes and Boolean functions represent their interaction.

EXAMPLE 5.1.– Figure 5.1 shows the interaction graph and the state transition graph of a Boolean network N_1 composed of the following two variables: {n_1,n_2}. The Boolean functions of N_1 are f_{n_1}, f_{n_2}, respectively, the Boolean function of n_1, n_2 and:

$$f_{n_1} = \neg n_2$$

$$f_{n_2} = \neg n_1$$

Let us consider that the Boolean network N_1 in Figure 5.1 is a gene regulatory network so that n_1, n_2 are genes. According to the interaction graph of N_1, n_1 is an inhibitor of n_2 and n_2 is an inhibitor of n_1. According to the Boolean functions of N_1 in example 5.1, the presence of n_2 at time step t will prevent the expression of n_1 at $t+1$. The absence of n_1 is enough to activate the expression of n_2 at $t+1$. The complete state transition graph is generated from the rules in Figure 5.1. The choice between the full lines and the dot lines depends on the updating scheme of the Boolean network. Regulatory networks are represented by interaction graph; however, those graphs are not enough to describe the Boolean network without ambiguity.

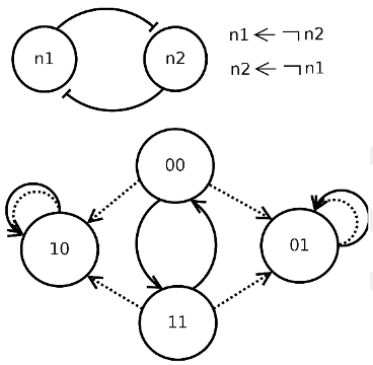


Figure 5.1. *The Boolean network N_1(top left), its Boolean function (top right), and its states transitions graph (bottom)*

From the Boolean functions, it is straightforward to generate the state transition graph (Figure 5.1). In a Boolean network, the value of nodes can be updated synchronously or asynchronously. In a *synchronous Boolean network*, all nodes are updated at the same time. The successive sequence of states during an execution, called *trajectory* of a Boolean network, is deterministic in a synchronous Boolean network. The behavior of the synchronous updating scheme can be seen on the full line graph in Figure 5.1. In an *asynchronous Boolean network*, a node may not be updated at given time (one node is updated at most in the following), so that its state transitions can be non-deterministic. In Figure 5.1, it corresponds to the dot line graph. The non-determinism can be seen on the multiple arrows starting from (00) or (11). In the case of regulatory networks, we will study asynchronous Boolean networks and the adapted temporal logic, the CTL. However, there exists numerous other kind of Boolean networks adapted for other models, for example the random Boolean networks [GER 03].

In [THO 01], Thomas networks are described as an extension of asynchronous Boolean networks (as well as an approximation of models constructed from differential equations). Boolean networks are extended in two directions. First, several

discrete levels of concentration for a gene may be considered, instead of two for Boolean networks. Second, the definition of the successor of a state is expressed with the help of the so-called *focal equations* associated with a gene in a given state. These equations provide a discrete concentration value called *focal values* indicating the tendency of a gene in a given state, thus extending the Boolean functions. We make the choice of presenting these notions, in section 5.4.1, by using the logical programming constructions of ASP for two purposes. This illustrates the expression power of the mathematically well-defined ASP formalism and, by this way, the formal aspect of our declarative approach is emphasized: data are described as logical variables that could be instantiated or not. In Chapter 7, we can find a mathematical introduction to the Thomas networks. Their presentation is complementary to ours in that sense that we define these networks directly in a logical and executable way by using ASP. Another introduction to temporal logics can also be found in addition to the following one.

5.3. Temporal logics

Temporal logics are extension of the propositional logic and describe properties on the dynamic of a system. There are many temporal logics, but most of them are extensions of two main temporal logics. That is why we will focus on those two logics: the LTL and the CTL. The LTL and the CTL allow us to describe two different sets of properties, and can both be useful for biology. Those properties can be hypothesis or data we want to verify on a model or real observations from experiments that we will use as constraint to build or to complete a model.

In the following section, a brief introduction to the LTL and CTL will be given. Then, we will present in details (section 5.3.2.1) the CTL implementation in ASP, which allows us to easily describe property on non-deterministic executions. On deterministic models, it is easy to build an LTL implementation in ASP from the CTL. In section 5.3.2.2, we will give this LTL implementation. However, this description is not correct if it is applied on non-deterministic models. After this short description of the LTL implementation, the CTL will be the main temporal logic to discuss.

5.3.1. *Definition of LTL and CTL*

We use the Kripke semantics to define the temporal logics. A Kripke model is an n-uplets $M = (S, I, T, L)$, with S a set of states, $I \subseteq S$ the set of initial states, $T \subseteq S \times S$ the set of the transitions of the system and L: $S \rightarrow P(A)$ a labeling function with A the set of atomic properties, and $P(A)$ the power set of A. For each state $s \in S$, $L(s)$ is the set of atomic properties that are true for s. The behavior of M is defined by the execution paths. A path p of M is a succession of states (s_0, s_1,...), with $s_i \in S$ and $T(s_i, s_{i+1})$ true for all $i \geq 0$. The ith state of the path is written as $p(i)$.

5.3.1.1. *Linear temporal logic*

The LTL describes properties on linear execution paths from an initial state s_0. This logic is defined as it follows:

$$\varphi ::= \text{a}\in\text{A}|\neg\varphi|\varphi_1 \wedge \varphi_2|\varphi_1 \vee \varphi_2|G\varphi|\varphi_1 U \varphi_2|X\varphi|F\varphi|\varphi_1 R\varphi_2$$

The semantic of the LTL for a path p is defined as follow:

$p \models$a iff a$\in L(p(0))$ $\quad$ $p \models \neg\varphi$ iff $p \not\models \varphi$
$p \models \varphi_1 \wedge \varphi_2$ iff $p \models \varphi_1$ and $p \models \varphi_2$ $\quad$ $p \models \varphi_1 \vee \varphi_2$ iff $p \models \varphi_1$ or $p \models \varphi_2$
$p \models G\varphi$ iff $p(i) \models \varphi\ \forall i \geq 0$ $\quad$ $p \models \text{X}\varphi$ iff $p(1) \models \varphi$

$$p \models \varphi_1 \text{U} \varphi_2 \text{ iff } \exists i \geq 0 \mid p(i) \models \varphi_2 \text{ and } \forall k, 0 \leq k < i\ p(k) \models \varphi_1$$

From those formulas, it is possible to build all the LTL.

$p \models \varphi_1 \Rightarrow \varphi_2$ iff $p \models \neg(\varphi_1 \wedge \neg\varphi_2)$
$p \models \text{F}\varphi$ iff $p \models \top \text{U}\varphi$
$p \models \varphi_1 \text{R} \varphi_2$ iff $p \models \neg\varphi_1 \text{U} \neg\varphi_2$

We note that verifying a property on a path is verifying the property on the initial state of the path. Because of the linearity, a path has only one initial state. However, if in the deterministic and synchronous case there is only one path for an initial state, in the non-deterministic case there is many possible execution for one initial state. In this last case, the formulas $\varphi(init)$ is verified if *all* the linear paths starting from $init$ verify φ. Because of this, it is more common to use LTL on deterministic executions and CTL on non-deterministic executions.

These are some examples of LTL properties applied to the synchronous Boolean network in Figure 5.1:

EXAMPLE 5.2.– The property: *in the future of the state* (00)*, the node* n_1 *will take the value 1.* The LTL formula to express this property is: F(n_1=1)(00).

The synchronous model in Figure 5.1 verifies this property because the state (11) of the path (00)→(11) satisfies this property.

However, the asynchronous model does not verify this property because the path (00)→(01)→(01)→... will never satisfy the formula.

EXAMPLE 5.3.– The property: *in the future of the state* (00)*,* $n_1 = 1$ *or* $n_2 = 1$ *will always be true.* The LTL formula to express this property is: FG(n_1=1 $\vee$ n_2=1)(00).

The synchronous model in Figure 5.1 does not satisfy this property because of the loop between (00) and (11). The property ($n_1 = 1 \vee n_2 = 1$) is false every time the path reach (00), this does not follow the "always" condition.

However, the asynchronous model verifies this property because for the paths (00)→(01)→(01)..., and (00)→(10)→(10)... (n_1=1 $\vee$ n_2=1) is always true after the initial state (00).

5.3.1.2. *Computational tree logic*

The CTL describes properties on a branching execution of a system[1]. CTL formulas can be separated in two categories: the *global* formulas with the prefix *A* and the *existential* formula with the prefix *E*. The syntax of the CTL is the following:

$$\varphi ::= \text{a}\in\text{A}|\neg\varphi|\varphi_1 \wedge \varphi_2|\varphi_1 \vee \varphi_2|EG\varphi|$$
$$E\varphi_1 U\varphi_2|EX\varphi|EF\varphi|AG\varphi|A\varphi_1 U\varphi_2|AX\varphi|AF\varphi$$

(a,$\neg$,$\wedge$,$\vee$) operators are the same as in LTL. The semantic of CTL for a state s$\in$S and a model M is defined as follow:

(M,s) $\models$ EGφ iff $\exists$ a path $p \mid p(0)$=s and $\forall i$, 0$\leq i$ (M,s$_i$=$p(i)$) $\models \varphi$
(M,s) $\models$ Eφ_1Uφ_2 iff $\exists$ a path $p \mid p(0)$=s and $\exists i \geq 0 \mid$ (M,s$_i = p(i)$) $\models \varphi_2$
and $\forall k$, 0$\leq k < i$ (M,s$_k = p(k)$) $\models \varphi_1$
(M,s) $\models$ EXφ iff $\exists$ a path $p \mid p(0)$=s and (M,s$_1 = p(1)$) $\models \varphi$

As for the LTL, the other CTL formulas can be build from the previous definitions.

(M,s) $\models$ EFφ iff (M,s) $\models E\top$Uφ
(M,s) $\models$ AGφ iff (M,s) $\models \neg$EF$\neg\varphi$
(M,s) $\models$ Aφ_1Uφ_2 iff (M,s) $\models \neg(\text{E}(\neg\varphi_1\text{U}(\neg\varphi_1 \wedge \varphi_2)) \wedge \neg\text{EG}(\neg\varphi_2))$
(M,s) $\models$ AXφ iff (M,s) $\models$ (M,s) $\models \neg$EX$\neg\varphi$
(M,s) $\models$ AFφ iff (M,s) $\models$ (M,s) $\models \neg$EG$\neg\varphi$

These are some example of CTL formulas applied to the asynchronous Boolean network in Figure 5.1:

EXAMPLE 5.4.– The property: *there exists a future of the state* (00) *where the node* n_1 *will take the value 1.* The CTL formula to express this property is EF(n_1=1)(00).

This model satisfies this property because the path (00)→(10) contains the state (10) where n_1 values 1.

1 Although it seems that LTL is a subset of CTL, LTL and CTL are in fact two distinct sets of properties. The logic allowing to express both LTL and CTL is CTL*.

EXAMPLE 5.5.– The property: *there exists a future of the state* (00) *containing a state* $p(i)$ *such as* $p(i)$ *validates: all the paths starting from* $p(i)$ *will ensure that the property* $n_1 = 1$ *is* always *true.* The CTL formula to express this property is EF(AG(n_1=1))(00).

The model verifies this property because the path (00)→(10)→(10)... contains a *loop* on (10) that has the property $(n_1 = 1)$.

5.3.2. *ASP implementation of CTL and LTL*

Now that we defined the CTL and LTL formalism, we will develop the implementation in ASP. In the further sections (5.3.2.1 and 5.4.2.1), we focus on the CTL formulas, and methods, for analyzing Thomas networks that are an extension of the Boolean networks. However, LTL is still useful for expressing property on a single path. For this reason, a LTL implementation in ASP will be given, but as an adaptation of the CTL implementation on deterministic models. This description can be found in section 5.3.2.2, after the description of the CTL implementation. On the other hand, the CTL implementation will be given in details and followed by some examples for the model checking of Boolean networks. Finally, this model-checking approach, and its utility, will be discussed, and other approaches will be developed in the following sections (sections 5.3.4 and 5.4).

5.3.2.1. *CTL implementation*

While LTL focuses on linear executions, CTL expresses properties of a set of branching paths. In this implementation, we only need to express the transitions[2] of the system (generated for example as in section 5.3.3). This CTL implementation presentation will be divided in two parts: implementation of the "simple" existential properties such as EX or EF and the related global properties, and the implementation of AF that allows us to define AU and EG.

For an easy description of the implementation, we associate to each CTL formula φ a predicate whose name exhibits the subformulas of φ. For example, if $\varphi =$ EX(n_1=1), the associated predicate is `eXtrue_n1` with `true_n1(S)` true if $n_1 = 1$ for the state `S`.

Also for the sake of simplicity, we represent the states by using the predicate `state(s(`n_1`,...,`n_k`))`, where n_1,...,n_k are the values of the species and k is the number of species. In the example (Figure. 5.1), `state(s(0,0))` true means that there is a state where n_1 =0 and n_2 =0.

2 In section 5.3.4, we give a way to verify CTL properties without having to know explicitly all the transitions.

EX, EU and the related formulas:

We first define the formulas related to the classical logic:

With `phi`, `phi1` and `phi2` some CTL formulas:

```
phi1_and_phi2(S) :- phi1(S), phi2(S).
phi1_or_phi2(S) :- phi1(S).
phi1_or_phi2(S) :- phi2(S).
not_phi(S) :- not phi(S), state(S).
phi1_Imply_phi2(S) :- not phi1_and_notphi2(S),
state(S).
```

The predicate `transition(S,S')` means there is a transition between the state S and the state S'. The definitions of EX and EU follow with:

```
eXphi(S) :- phi(S'), transition(S,S').
ephi1_U_phi2(S) :- phi2(S).
ephi1_U_phi2(S) :- phi1(S), ephi1_U_phi2(S'),
transition(S,S').
```

The first rule defining `ephi1_U_phi2(S)` means the state S directly verifies φ_2.

The second rule means that a state `S` verifies Eφ_1Uφ_2 if it verifies φ_1 and if it is linked to a state that verifies Eφ_1Uφ_2. This is the transitivity rule: if no reachable state verifies φ_2, neither the first nor the second rule can be true.

The advantage of this implementation is to able to manage complex models with loops (like in the example given in section 5.3.3) without introducing a specific test to take them into account. In case of loop, if we are in the situation where `ephi1_U_phi2(S)` is true only if `ephi1_U_phi2(S)` is true, then, because ASP accepts only minimal models, the property `ephi1_U_phi2(S)` is false, as it should be (see also section 5.3.4).

From those first formulas, we can build the following CTL formulas:

```
aXphi(S) :- not eXnotphi(S).
eFphi(S) :- etrue_U_phi(S).
aGphi(S) :- not eFnotphi(S).
```

EXAMPLE 5.6.– The property: *there exists a future of the state* (00) *where the node* n_1 *will take the value 1.* The CTL formulas to express this property is EF(n_1=1)(00), with $s(0) = (00)$.

Definition of the atomic property n_1=1:

```
n1(s(1,_)).
```
[3]

Then, we can define `EF` in a more direct way by:

```
eFn1(S) :- n1(S).
eFn1(S) :- eFn1(S_), transition(S,S_).
```

The required property of the initial state (00) is expressed by the integrity constraint:

```
:- not eFn1(s(0,0)).
```

EXAMPLE 5.7.– The property: *there exists a future of the state* (00) *containing a state* $p(i)$ *such as* $p(i)$ *validates: all the paths starting from* $p(i)$ *will ensure that the property* $n_1 = 1$ *is always true*. The CTL formulas to express this property is EFAG(n_1=1)(00).

Definition of `aGn1`:

```
not_n1(S) :- not n1(S).
eFnot_n1(S) :- not_n1(S).
eFnot_n1(S) :- eFnot_n1(S_), transition(S,S_).
aGn1(S) :- not eFnot_n1(S).
```

Definition of `eFAGn1`:

```
eFAGn1(S) :- eGn1(S).
eFAGn1(S) :- eFAG(S_), transition(S,S_).
```

AF implementation: The `AF(`φ`)` property means that all the paths will reach a state verifying φ. This is a strong reachability property, very useful for expressing the inevitability of a state. For example, a state S is an attractor if it respects the property AX(AF(S))(S): all the paths will lead to itself, see example 5.8.

```
aFphi(S) :- phi(S)
aFphi(S) :- not phi(S), aFphi(S_) :
            transition(S,S_), state(S).
```

Like for EF, AF(φ)(S) is true if S verifies the property φ, this is given by the first rule. The second rule indicates that if $\varphi(S)$ is false, then every states following S must

3 The "_" symbol in the predicate means that any value is possible to satisfy the rule.

verify AF(φ): this is expressed by the `":"` operator that enables to enumerate all the successors of S. Again, because of the minimality of the ASs, loops do not need to be explicitly checked.

EXAMPLE 5.8.– The property: *Given an initial state $S0$, all the path starting from $S0$ must lead to $S0$, which is the definition of an attractor. This can also be written: for all the* next *states S' of $S0$, all the paths starting from S' will reach $S0$ in the future*. The CTL formulas to express this property applied to the state S0 = (0,0) is AX(AF(eqS0))(S0).

Definition of the property: being equal to S0 (eqS0):

```
eqS0(S(0,0)).
```

Definition of `aF_eqS0`:

```
aF_eqS0(S) :- eqS0(S).
aF_eqS0(S) :- not eqS0(S),
          aF_eqS0(S_) :  transition(S,S_),state(S).
```

Definition of `eXnotAF_eqS0`:

```
eXnotAF_eqS0(S) :- not aF_eqS0(S_), transition(S,S_).
```

Definition of `aXAF_eqS0`:

```
aXAF_eqS0(S) :- not eXnotAF_eqS0(S), state(S).
```

The required property is:

```
:- not aXAF_eqS0(S(0,0)).
```

The AF implementation also allows us to define two other CTL formulas: EG and AU. These definitions are not shown here.

5.3.2.2. *LTL implementation*

The LTL logic allows us to describe properties on linear execution paths, and it can be very useful to describe the behaviors of deterministic systems. In this last case, the behavior is constituted by an unique linear path, and a description of LTL can be done by using our previous CTL implementation.

If the model is deterministic, and if the set of transitions `transition(S,S')` describing a linear path is given, then we can make the following equivalence between

LTL and CTL: EU is equivalent to U, AG is equivalent to G and EX is equivalent to X[4]. For example, given the deterministic model in Figure 5.1, we express in ASP the property F(n_1=1)(00) with the following program.

We define the atomic property n_1=1 as in example 5.6.

Then, we can define `fn1(S)`:

```
fn1(S) :- n1(S).
fn1(S) :- fn1(S_), transition(S,S_).
```

and finally, the required property on the initial state (00):

```
:- not fn1(s(0,0)).
```

5.3.3. *Example of model checking of a Boolean network*

In this section, we give an example of a Boolean network implementation. Now that we have seen how to implement CTL properties in ASP, we will check temporal properties, such as a reachability problem EF and the verification of an attractor. This Boolean network has been artificially built to possess two attractors of different sizes, and a cycle which is not an attractor. If this section focuses on the model checking of a fully known Boolean network, in the next section, we will describe how to analyze partially known Thomas networks.

The input or initial knowledge is the Boolean functions of the Boolean network (Figure 5.2):

$$f_{n_1} = n_1 \vee (n_3 \wedge \neg n_2)$$
$$f_{n_2} = (n_2 \wedge \neg n_3) \vee (n_3 \wedge \neg n_2) \vee (n_2 \wedge n_1) \vee (n_1 \wedge \neg n_2)$$
$$f_{n_3} = (\neg n_1 \wedge \neg n_3) \vee (n_1 \wedge n_2 \wedge n_3) \vee (\neg n_2 \wedge \neg n_1)$$

The goal is to apply some CTL properties, and for this we need the transitions of the system. The predicate `transition(S,S')` is defined by the following ASP program:

The data of the system are:

```
node(n1;n2;n3).
bool(0;1).
```

4 Again, we insist it is true only in this particular case.

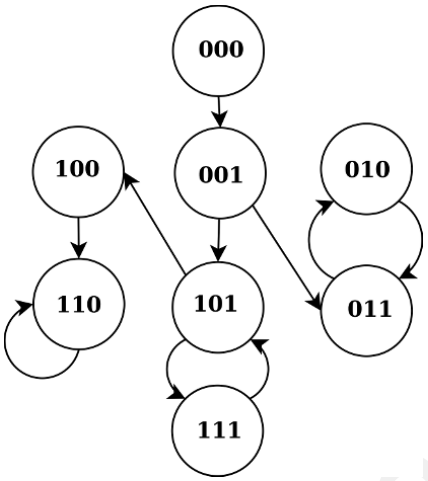


Figure 5.2. *State transition graph of a 3 nodes Boolean network*

The predicate `change(N,S,V)` means there is a successor S' of the state S such that S' is equal to S except for the value of N which is V. This give us this first rule:

```
    change(N,s(X,Y,Z),0) :- not change(N,s(X,Y,Z),1),
node(N), bool(X), bool(Y), bool(Z).
```

And then, we describe the Boolean functions:

```
    change(n1,s(X,0,1),1) :- bool(X).
    change(n1,s(1,X,Y),1) :- bool(X), bool(Y).
    change(n2,s(X,1,0),1) :- bool(X).
    change(n2,s(X,0,1),1) :- bool(X).
    change(n2,s(1,0,X),1) :- bool(X).
    change(n3, s(1, 1, 1), 1).
    change(n3, s(0, X, 0), 1) :- bool(X).
    change(n3, s(0, 0, X), 1) :- bool(X).
```

Finally, transitions are constructed when a state is subject to a `change`:

```
    transition(s(X, Y , Z), s(X_, Y , Z)) :- change(n1,
s(X, Y, Z), X_), X_ != X .
    transition(s(X, Y , Z), s(X, Y_ , Z)) :- change(n2,
s(X, Y, Z), Y_), Y_ != Y.
    transition(s(X, Y , Z), s(X, Y , Z_)) :- change(n3,
s(X, Y, Z), Z_), Z_ != Z.
```

```
    transition(s(X, Y , Z), s(X, Y , Z)) :- change(n1,
s(X, Y, Z), X), change(n2, s(X, Y, Z), Y), change(n3,
s(X, Y, Z), Z).
```

Then, we only need to add the rules defined in examples 5.6 and 5.7 to verify them on this model. However, we need to use the modified rules for AF defined in 5.3.4, if we want to apply the property of example 5.8 to this model, because `transition` atoms are not known after the grounding phase. This method allows us to verify all the CTL formulas on Boolean networks where the Boolean functions has been completely determined. However, as we will see later, the CTL formulas can be used for more than just verification, for example as constraints on the construction or the completion of a model, or as properties we can infer from an existing model.

5.3.4. *Discussion*

We can note again the important point that concerns the implementations of EF and AF formulas because they may seem straightforward at first sight. However, we could question on how loops in paths are tackled. For example, looking at example 5.6, one could find the case where `eFn1(s1)` is only defined by:

```
eFn1(s1) :- eFn1(s2), transition(s1, s2).
eFn1(s2) :- eFn1(s1), transition(s2, s1).
```

Thus, it is of crucial importance to recall that the stable models, those only accepted by ASP, are minimal [GEL 88]. This means that if `transition(s1, s2)` and `transition(s2, s1)` are true, the only accepted model is the one where `eFn1(s1)` and `eFn1(s2)` are false, whereas classical logic would consider also the model where they are true.

Another point concerns the status, fixed or unknown, of the atoms with predicate `transition`. The implementations that we have described suppose that these atoms are known. This simplifies significantly the grounding phase. In the last example, we analyze the state transition system of a Boolean network where these atoms are known after the grounding phase. Moreover, in most biological applications, we lack of complete models. For example, in the next section, we build the transitions and do not know them first. This notably affects the rules defining the AF property. For this property, we can deal with partial transition system, and infer properties because of the following implementation, where `hyp_AFphitrans(S,S')` replaces the unknown transitions:

```
    hyp_AFphitrans(S,S') :- aFphi(S'), transition(S,S'),
state(S), state(S').
    hyp_AFphitrans(S,S') :- not transition(S,S'),
```

```
state(S), state(S').
    aFphi(S) :- phi(S).
    aFphi(S) :- not phi(S), state(S),
hyp_AFphitrans(S,S') : state(S').
```

We can also make some remarks about the context of the use of some CTL properties and about the expression power of the ASP logic compared to CTL. First, in this work, we translated the whole CTL into ASP; however, in most of the analysis cases of biological networks, only the EF property is used. In fact, using global properties is often not relevant when we transcribe a biological experiment in CTL. Most of the data are in the form of state S reach state S'. If this result is given by only one, or few experiments, it must be translated as an existential property and not as a global one. In fact, the global property is too strong and will cut possibly true paths. However, if we change the context, and do not focus only on the analyzing of an existing model, but also on a synthetic biological approach, it becomes interesting to ensure global properties such as AF.

The second remark is that ASP is logically based and as such can express properties. It is not based on a temporal logic, but nevertheless can be used to represent temporal properties. In a sense, one can consider that it has a greater expression power than CTL, mostly because logical variables are available. It allows us, for example, to express properties such as "the model admits three different steady states", while this is not possible in CTL, see also section 5.4.2.1. CTL is a useful and compact way to express behavior of asynchronous models, but we gain to paired it with properties expressed directly in first order logic, as we will see in the next section.

5.4. ASP-based analysis of a GRN

As mentioned in section 5.1, this section is devoted successively to the definition of Thomas GRNs in ASP, biological data modeling, methodology for building models and finally to the description of three real biological applications.

5.4.1. *ASP Thomas networks specification*

We explain, in section 5.2.2, how a Thomas GRN is an extension of a Boolean network and why its specification is given here in terms of ASP logical constructions. We recall that a nice classical description of these GRNs is given in Chapter 7. Our notations are similar with a slight difference concerning kinetic parameters (see section 5.4.1.3). Also the network in Figure 5.3 is very close to the mucus production of the *Pseudomonas* network presented in Chapter 7.

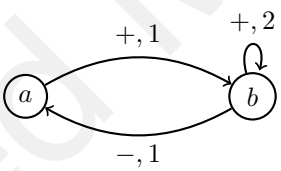
First, we present the interaction and transition graphs associated, respectively, with static and dynamic views of Thomas GRNs. Then, we introduce the notion of

focal state and *path* in a transition graph. Paths are well adapted to express observed behaviors (see section 5.4.2.1) as CTL like formulas of the form $EF\varphi$. Such formulas imply the existence of a path with a state having a property φ (see section 5.3.4 for a discussion on the adequacy of AF and EF formulas for representing biological behaviors).

5.4.1.1. *Interaction graph*

In the framework of Thomas GRNs, the concentration range of a gene is split into intervals possibly separated by several thresholds, instead of one for Boolean GRNs. Then, in interaction graphs, an edge (j, i) is labeled with the rank of the threshold concentration (in comparison to other outgoing edges from gene j), at which the effect of the protein produced by gene j on the expression of gene i changes.

This kind of graph is easily specified in ASP by using the predicates `node(N)` (`N` is an identifier representing a gene), `edge(N1,N2,Ne)` (there is an edge from node `N1` to `N2` whose index, introduced for modeling multiple edges with the same source and target, is `Ne`) and `threshold(N1,N2,Ne,T)` (`T` is the threshold of the edge `(N1,N2,Ne)`). The interaction graph of the example shown (Figure 5.3) would be specified in a program with the following atomic formulas: `node(a). node(b). edge(a,b,1). edge(b,a,1). edge(b,b,1). threshold(a,b,1,1). threshold(b,a,1,1).threshold(b,b,1,2).`



(a)

$$
\begin{aligned}
X_a &= K_a * s^-(x_b, \theta_b^1) + \\
&\quad K_a^b * s^+(x_b, \theta_b^1) \\
X_b &= K_b * s^-(x_a, \theta_a^1)\ s^-(x_b, \theta_b^2) + \\
&\quad K_b^a * s^+(x_a, \theta_a^1)\ s^-(x_b, \theta_b^2) + \\
&\quad K_b^b * s^-(x_a, \theta_a^1)\ s^+(x_b, \theta_b^2) + \\
&\quad K_b^{ab} * s^+(x_a, \theta_a^1)\ s^+(x_b, \theta_b^2)
\end{aligned}
$$

(b)

Figure 5.3. *a) Interaction graph corresponding to a GRN of two genes. The protein product of gene a stimulates the expression of gene b when above its first threshold θ_a^1, while the protein product of gene b inhibits the expression of gene a when above its first threshold θ_b^1. In addition, b activates its own expression when above its second threshold θ_b^2. (b) Focal equations relating a state characterized by the vector of protein concentrations $[x_a, x_b]$ and its focal state $[X_a, X_b]$. If $x \geq \theta$, then the value of $s^+(x, \theta)$ is 1 and else 0. Note that $s^-(x, \theta) = 1 - s^+(x, \theta)$.*

5.4.1.2. *Transition graph*

As for Boolean GRNs, the dynamic behavior of a Thomas GRN is represented in terms of a state transition graph, where each node represents a specific state (a vector of the expression level values of each gene) and the edges represent transitions between these states (see Figure 5.4). The gene expression levels take discrete values, each one representing an interval between two consecutive thresholds. In Figure 5.3, the gene b can take the values 0, 1 or 2.

The domain of values `V` of a gene `N` is specified because of the predicate `val(N,V)` and the rule:

```
val(N,0..D) :- node(N), out_degree(N,D).
```

where `D` is the outside degree of `N` in the interaction graph. Then, for each edge, the domain of a threshold and its unicity can be specified by the following rule:

```
1{threshold(N1,N2,Ne,V):val(N1,V):V>0}1 :-
                                  edge(N1,N2,Ne).
```

The compactness of the thresholds of a gene N is specified by the following integrity constraint:

```
:- threshold(N1,N2,Ne,T), T>1,
   not threshold(N1,N3,Ne3,T-1):edge(N1,N3,Ne3).
```

expressing that if `T`, with `T>1`, is a threshold value, then a threshold value `T-1` exists.

5.4.1.3. *Focal state*

A focal state is a specific attractor associated with each state, represented by a vector of the focal values of each genes. A focal value for a gene in a state expresses the level toward which this gene tends to evolve given the presence, or lack thereof, of activators and/or inhibitors on that gene in this state. In the example (Figure 5.3), the focal value of gene a depends on the expression level of gene b, that is whether the concentration of b is above or below its first threshold. A focal value is given by a discrete parameter called a *kinetic parameter* associated with a particular *cellular context*. A cellular context is any set of states that are equivalent with respect to the presence, or lack thereof, of activators and inhibitors acting on a particular gene. In the example, for the gene b (Figure 5.3), there are four cellular contexts depending on whether a is above its threshold and whether b is above its second threshold.

We denote the focal value of gene i in a cellular context c_i by $K_i^{act(c_i)}$, where $act(c_i)$ is called the *cellular context identifier* of c_i. The set $act(c_i)$, composed of all the genes influencing i whose value are above their threshold in the cellular context

c_i, is a representation of c_i. In the example (Figure 5.3), the focal value of b for all the states belonging to the cellular context of b where a is above its first threshold and b is above its second threshold is K_b^{ab}. In Chapter 7, the definition of $act(c_i)$ is different as it is based on the signs of the edges targeting i, which in our approach could be unknown.

To specify the focal value of a gene N in a state (defined as being the `I`th step in a path `P`, see section 5.4.1.4), we introduce the following predicates:

– `param(Ik): Ik = k(N,CC)` represents a kinetic parameter $K_N^{act(c_N)}$ and `CC = cc(N1,Ne1,cc(N2,Ne2,...,cc(Np,Nep,nil)))` represents $act\,(c_N)$. `CC` characterizes the set of states where the concentrations of all `Ni` are, respectively, above the threshold of the edges `(N1,N,Ne1),...,(Np,N,Nep)` and where the concentrations of the source genes of all other edges for which `N` is a target are under their edge's threshold. The definition of this predicate, which we do not detail here, is directly built from the specification of the interaction graph (see section 5.4.1.1).

– `kparam(K,Ik) : K` is the value of the parameter `Ik`. The following definition ensures that parameters values are unique and in the right range[5]:

```
1{kparam(K,k(N,CC)):val(N,K)}1 :- param(k(N,CC)).
```

– `cell_context(N,CC,I,P)` : the state at step `I` of the path `P` belongs to the cellular context `CC` of `N`. We do not give here the recursive definition of this predicate.

– `focal(N,K,I,P)`, representing the focal equation of `N`, i.e. `K` is the focal value of `N` of the state at the step `I` of `P`:

```
focal(N,K,I,P) :- kparam(K,k(N,CC)),
     cell_context(N,CC,I,P).
```

5.4.1.4. *Paths*

In our framework, we are interested in examining the behavior of a network using a succession of states that comprise a path. A successor of a state in the transition graph is deduced by comparing the current expression level of each gene with that of its focal state. The transition of a state to one of its successor states is asynchronous, in the sense that at most one gene can change expression level between states. This component value of the state is increased (respectively, decreased) by 1 if its focal value is greater (respectively, lower) than the gene expression level. If no component is updated, then the state is equal to its focal state and to its successor: it is said a *steady* (or stationary) state.

5 In the annex, we give a more refined definition.

For specifying the expression level and evolution of a gene as a component of a state, we introduce the predicate `species(N,V,I,P)` (`V` is the expression level of the gene `N` at step `I` of the path `P`), which is defined using the following predicates: `path(P)` (`P` is a path), `length(L,P)` (`L` is the length of `P`), `step(I,P)` (`I` is a step of `P`) and rules:

```
species(N,V+1,I+1,P) :- diff(N,I,P), val(N,V),
                  focal(N,K,I,P), step(I+1,P),
                      species(N,V,I,P), K > V.
species(N,V-1,I+1,P) :- diff(N,I,P), val(N,V),
                  focal(N,K,I,P), step(I+1,P),
                      species(N,V,I,P), K < V.
species(N,V,I+1,P) :- not diff(N,I,P),
      val(N,V), step(I+1,P), species(N,V,I,P).
```

The predicate `diff(N,I,P)` (`N` is the unique component to be updated at step `I` of `P`) ensures the asynchrony of transitions. Its definition implies that at most one atomic formula `diff(N,I,P)` is true for a given step `I`, so at most one component may change between states:

```
0{diff(N,I,P): node(N}1 :- step(I; I+1, P).
```

Finally, an integrity constraint is required to assert equality between a steady state (a step `I` of `P` where no atomic formula `diff(N,I,P)` is true) and its focal state (`foceg(N,I,P)` is true if `N` is equal to its focal value at step `I` of `P`):

```
:- 1{not foceg(N,I,P):node(N)},
     0{diff(N,I,P): node(N)}0, step(I; I+1,P).
```

5.4.2. *Biological data modeling*

Biological data are frequently qualitative and incomplete. In its current form, our implementation in ASP can analyze and model three types of biological data: behaviors of the network, interactions between genes and mutant networks whereby genetic engineering has altered the network.

5.4.2.1. *Behaviors*

Experimental behavioral data can generally be expressed using constraints on paths. This is the case for modeling observed steady states, cycles or repairing behaviors due to stress. The declarative approach presents a decisive advantage as information on these behaviors is usually incomplete; for example, there could exist a cycle for which only some concentrations of proteins are known throughout the cycle.

Despite the lack of information, our approach may provide biologically meaningful properties regarding the kinetic parameters.

Expressing the existence of a steady state requires a predicate `statpath(P)` (the two states of the path `P` of length `2` are equal) defined by:

```
statpath(P) :- path(P), length(2,P),
                    succeg1(N,P):node(N).
```

where `succeg1(N,P)` is true if at the first two steps of the path `P`, the concentrations of the species `N` are equal. The existence of a steady state `ss` can then be easily asserted with the two facts and the integrity constraint that follow:

```
path(ss). length(2,ss). :- not statpath(ss).
```

We can note that no concentrations were known or given to assert the existence of a stationary state.

This expressive power provides significant benefits over well-known temporal logics such as CTL [CLA 82] (see also section 5.3.4), which have been proposed to check instantiations of Thomas networks [BER 04, CHA 03]. For example, a query asking whether a model admits three different steady states, easy to formulate as an extension of the above rules, cannot be expressed in CTL.

Nevertheless, CTL is useful to express biological observations, typically with EF formulas like "there exists at least one path with states that have such properties". In our declarative framework, we can easily assert such formulas. For example, asserting the existence of a path for the network shown in Figure 5.3 following the CTL formula $(a = 0 \wedge b = 0) \wedge EF(a = 0 \wedge b = 2)$ (meaning that there exists a path beginning with a state where $a = 0$ and $b = 0$ and reaching a state where $a = 0$ and $b = 2$) is achieved with the following rules:

```
path(p). length(5,p).
:- not species(a,0,1,p).
:- not species(b,0,1,p).
exist_path :- species(a,0,I,p),
species(b,2,I,p), step(I,p).
:- not exist_path.
```

The only models satisfying this formula in Figure 5.4 are G_4 and G_6. Note that we set to 5 the length of a path because it is the maximal length of a non-looping path for this example.

Asserting universal CTL properties, such as "all paths originating from such states have such properties", are not so easily handled. But, as mentioned earlier (see

section 5.3.4), these formulas are not appropriate for analysis purposes. Representing biological observations by AF or AG formulas could lead to reject some networks unduly.

5.4.2.2. *Interaction signs*

An edge of an interaction graph is often labeled with a "+" or "−" sign. Intuitively, a "+" (respectively "−") sign means that the protein of the source of the edge activates (respectively inhibits) production of the targeted gene. However, these signs may be loosely interpreted in the literature. In this framework, we have to give them a precise and comprehensible definition in the form of conditions called *observability constraints* (they must not be confused with the integrity constraints which are the ASP constraints). A "+" (respectively "−") sign on an edge targeting a gene is understood as implying the existence of a couple of states $(s1, s2)$, with $s1$ just below the edge threshold, such that (1) $s2$ differs from $s1$ only by a $+1$ change in the value of the source gene and (2) $s2$ has a greater (respectively lower) focal value for the target gene than $s1$.

We may see why the transition graph G_4 (Figure 5.4) follows the "+" label associated with the edge $a \rightarrow b$ (Figure 5.3). A state $[0, 1]$ exists in G_4, such that the value of the source node a is lower than the threshold θ_a^1 of this edge. This state has a neighboring state $[1, 1]$, which differs only in the value of a by a change of $+1$. Furthermore, this neighbor shows a positive tendency ($K_b^a = 2$) for b, indicating a future growth in expression level, while the state $[0, 1]$ shows a negative tendency ($K_b = 0$).

By abstracting states by cellular contexts, one can note that the existence of a such a couple $(s1, s2)$ is equivalent to the existence of a couple $(c1, c2)$ of cellular contexts of the target node having the following extended properties for a "+" (respectively, "−") sign: all states in $c1$ below the edge threshold and (1) $c2$ differs from $c1$ only by value of the source gene greater or equal than the edge threshold and (2) the focal value of the target gene in the context $c2$ has a greater (respectively, lower) value than in context $c1$.

In the transition graph G_4, considering again the positive interaction $a \rightarrow b$, such a couple of cellular contexts of b could be for $c1$ the cellular context where $a < \theta_a^1 \wedge b < \theta_b^2$ and for $c2$ the one where $a \geq \theta_a^1 \wedge b < \theta_b^2$.

An observability constraint is modeled with the predicate `obs(S,N1,N,Ne1)` (in the case of `S=p` (respectively, `S=m`) then the edge `(N1,N,Ne1)` is an activation (respectively, inhibition)) having the following definition:

```
sign(p).sign(m).
obs(S,N1,N,Ne1) :- sign(S), ineq_K(S,N,K,K_r),
```

```
                neighbors(N1,N,Ne1,K,K_r),
                  auto_inter(S,N1,N,Ne1,K).
```

where the predicate `neighbors(N1,N,Ne1,K,K_r)` is true if there exists a cellular context identifier of `N` containing the edge `(N1,N,Ne1)` with a "neighbor" (deduced from it by deleting this edge) whose respective parameter values are `K` and `K_r`. The predicate `ineq_K(S,N,K,K_r)` then ensures that these parameter values stay in the right order according to `S`. The literal `auto_inter(S,N1,N,Ne1,K)` is devoted to the more informed autointeraction case that we do not describe here. For example, the observation of an activation in the interaction $a \rightarrow b$ (Figure 5.3) can be formalized with the integrity constraint `:- not obs(p,a,b,1).` The couples of cellular contexts of b that are involved are represented in ASP by `nil` and `cc(a, 1, nil), cc(b, 1, nil)` and `cc(a, 1, cc(b, 1, nil))`. The resulting observability constraints are expressed in terms of the focals of b by $(K_b < K_b^a) \vee (K_b^b < K_b^{ab})$.

Additionally, *additivity constraints* are considered to indicate that generally no inhibition (respectively, activation) can exist in case of a positive (respectively, negative) interaction. These additivity constraints are expressed with the help of the predicate `addit(S,N1,N,Ne)` defined by the following "default" [BES 89] rule:

```
addit(S,N1,N,Ne) :- obs(S,N1,N,Ne),
                    opposite_sign(S,Sp),
                    not obs(Sp,N1,N,Ne).
```

and also by considering the predicate `-obs(Sp,N1,N,Ne)`, which is the negation of `obs(S,N1,N,Ne)` (see [GEB 10]). For an interaction, consider the positive interaction $a \rightarrow b$ in Figure 5.3, an additivity constraint is asserted with an integrity constraint:

```
:- addit(p, a, b, 1), not -obs(m, a, b, 1).
```

This means that in the general case where `obs(p, a, b, 1)` holds but where `obs(m, a, b, 1)` (e.g. $(K_b > K_b^a) \vee (K_b^b > K_b^{ab})$) does not, then `-obs(m, a, b, 1)` (e.g. $(K_b \leq K_b^a) \wedge (K_b^b \leq K_b^{ab})$) holds. However, this does not mean that the exceptional case where both `obs(p, a, b, 1)` and `obs(m, a, b, 1)` hold is eliminated as being inconsistent. In this case, the above integrity constraint remains satisfied because `addit(p, a, b, 1)` does not hold.

5.4.2.3. *Mutants*

In the study of genetic networks, biologists frequently suppress the expression of a gene or over-express it using genetic engineering. The resulting networks are labeled *mutant*, in contrast with the unaltered *wild* networks. In this section, "model" refers to both wild and mutant networks. It is important to define a mutant network from its

comparable wild one and express different properties on each network while ensuring that they share the same kinetic parameters.

We proceed here with an extension of our previous work: we introduce the predicates `model(M)` (`M` is a model), `mutant(N,M,V)` (in the model `M`, the gene `N` is mutated and its expression value is `V`) and `mutant(N,M)` (in the model `M`, the gene `N` is mutated). Moreover, we extend the predicate `path(P)` to `path(P,M)` (`P` is a path in the model `M`) in order to distinguish between paths in different models. Depending on the model, the value `V` of gene `N` in the first step of a path may be fixed or left ambiguous.

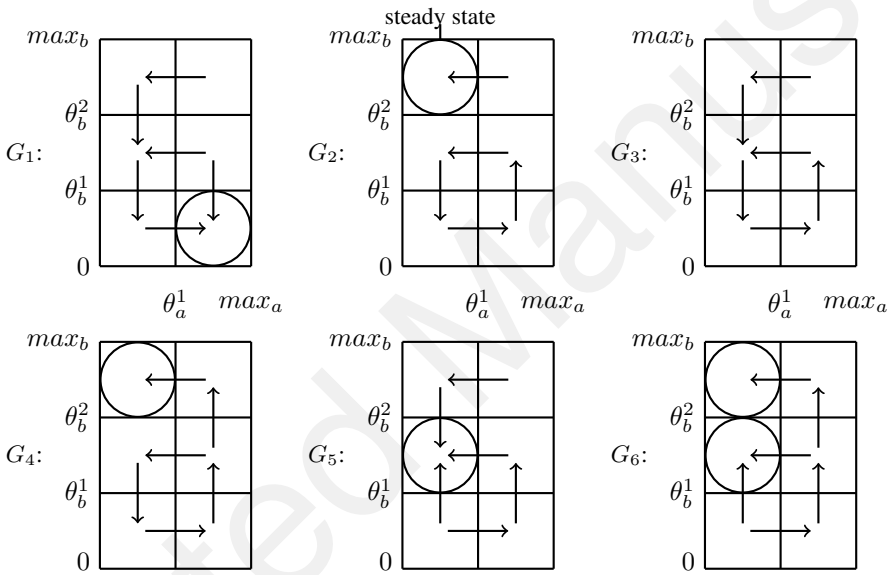


Figure 5.4. *Transition graphs G_1, ..., G_6 satisfying all the observability and additivity constraints associated with the example (Figure 5.3). Arrows represent possible transitions between states represented by boxes. Each graph corresponds to a specific set of instantiated kinetic parameters. For example, the graph G_4 corresponds to the following instantiation: $K_a = 1, K_a^b = 0, K_b = 0, K_b^b = 2, K_b^a = 2, K_b^{ab} = 2$*

The non-monotonicity of ASP allows for the same expression of focal equations for either the wild model or mutant models. To obtain this expression, we add the literals `not mutant(N,M), path(P,M)` to the body of the rule defining the focal equations (see section 5.4.1.3). This prevents that these rules reset the focal value of a mutant gene. In case `N` is a mutant gene in the model `M`, its focal value `V` is defined by the rule:

```
focal(N,V,I,P) :- mutant(N,M,V),
                  step(I; I+1,P), path(P,M).
```

5.4.3. *Methodology for building models*

The formulation of interesting queries in a methodical way is critical for the analysis of the network functioning. To deal with this problem, we proposed a simple, four-step method [COR 09]: (1) construct an initial set of constraints integrating as many biological observations and hypotheses as possible; (2) check the consistency of this set and in case of inconsistency remove as few non-mandatory constraints as possible from the resulting consistent set; (3) predict the meaningful properties verified by all consistent models by means of dedicated languages expressing properties about Thomas GRN depending on results and new experiments; (4) continue by removing or adding hypotheses and returning back to step 2. In the following, we focus on steps 2 (inconsistency repairing) and 3 (inference of properties) and also on minimization facilities that are often asked for.

5.4.3.1. *Inconsistency repairing*

In case the initial set of constraints is inconsistent, the constraints supported by strong biological observations, which cannot be removed, must be separated from those which are weakly supported. We then face a well-identified issue in AI: a maximization problem, which in this case, is finding the maximum number of acceptable weak biological observations to produce a consistent model. It necessitates a paralogic process to be tackled. In our framework, this problem can be solved using the *gringo* optimization statement `#maximize` in the following two steps:

– Determining the maximal number of acceptable constraints. Let `cv_1,...,cv_n` be the literals representing constraints that can be removed. By applying the statement `#maximize{cv_1,...,cv_n}`, we get the maximum number `Max_cv` of these literals that can be true in any one model.

– Inserting the cardinality constraint `Max_cv{cv_1,...,cv_n}Max_cv`. Each resulting AS then contains the maximum number of literals possible among `cv_1,...,cv_n`.

Different sets of constraints could possibly be removed *a priori*, all of them should be considered, with the exception of those that are not biologically plausible (see [COR 09] for such a case).

It is important to recall that the non-monotonic framework of ASP makes possible to escape from such a paralogic process, provided that defaults have been identified. As discussed earlier in section 5.4.2.2, a default rule allows us to infer consequences that could be deleted without provoking inconsistency if the default is not respected. In the example (Figure 5.3), the additivity hypothesis implies that $K_b \leq K_b^a$ because

there is a positive interaction associated with $a \rightarrow b$. Considering this hypothesis as biologically strongly supported would lead to a inconsistency in the case where the contrary $K_b > K_b^a$ holds. Because of the default expression of this hypothesis, this apparent contradiction would be admitted in this case. The consequence $K_b \leq K_b^a$ would no longer hold and different models than those shown in Figure 5.4 would be proposed.[6]

5.4.3.2. *Inference of properties*

Then, from a coherent set of constraints, we are naturally conduced to search for *predictions*. Within our approach, predictions can be seen as properties that hold in all models consistent with these constraints. To be worthwhile, this kind of automatic learning should be *non-supervised*, i.e. not predicting a priori fixed properties but, instead, non-fixed properties but belonging to a language. The design of languages expressing properties is a problem per se, which has to be discussed with biologists. Ideally, for predictions to be a guide for further experiments, biologically relevant properties should be experimentally verifiable. So we do not address this issue and will just give an insight into this question.

Such a language could be composed with logical clauses [BOS 85]. Inferring properties consequently consists of inferring clauses. It is then critical to pay attention to the choice of the set of atomic formulas. In [COR 09], the authors exhibit two languages where these atomic formulas are inequalities between kinetic parameters belonging to the same focal equation. For example (Figure 5.4), the formula $K_b < K_b^a \vee \neg(K_b < K_b^{ab})$ belongs to such a language and is actually true in all models.

Automatic deduction of common characteristics across all models is simple with the `--cautious` option of the solver [GEB 10]. This option provides all atomic formulas true across all models. By specifying the property languages using appropriate predicates, one can easily obtain all the properties consistent across all models.

ASP is specifically interesting when used in the inference of properties: as models are minimal, the number of deduced properties is at least as high as the number of properties deduced using classical logic. For example, `a :- not b.`

6 Another issue, connected to the above and important for property inference (see section 5.4.3.2), have to be discussed: it is the definition of the models that are desirable to retain regarding additivity constraints. A first and radical solution consists of retaining only the networks presenting the maximal number of additivity constraints, i.e. with the maximal number of `addit` literals. Within this modeling, it is appropriate to rely on the operator `#maximize` for that purpose. In the appendix (section 5.7), where we present a refined modeling, we discuss this notion of desirable models and the relative roles of the two possible "minimizing" ways: the (non-monotonic) logical way and the paralogical way.

`b :- not a.` has only two minimal models: {`a`} and {`b`}, while the model {`a`, `b`} is not minimal. We can, thus, infer the exclusion of `a`∧`b`, which would not be possible using classical logic.

5.4.3.3. *Minimization*

It can happen that the set of constraints accepts a very large number of consistent models, typically if the knowledge about the biological network is not very large. In these cases, a frequent request concerns the existence of specific models. For defining such a specificity, optimization criteria are commonly proposed. For example, it may be interesting from a biological point of view to focus on models for which the total number of thresholds is minimal. Of course, nothing prevents *a priori* that a gene has as many different thresholds as its output degree in the interaction graph. It may even be possible that a Boolean model (with only one threshold for each gene) should be consistent.

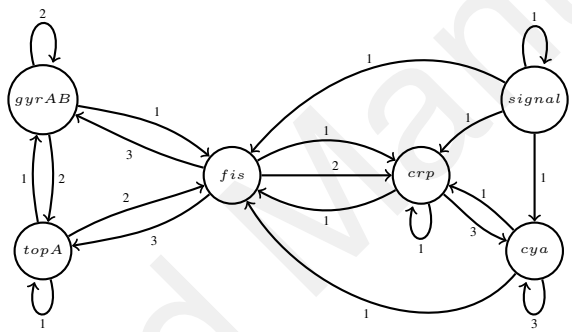


Figure 5.5. *Interaction graph of the regulation of the carbon starvation response in Escherichia coli*

A search for the minimum number of required thresholds in the network is possible using the `#minimize` operator, dual of `#maximize` (see section 5.4.3.1). By defining the predicate `threshold_max(N,T)` (`T` is the number of threshold of `N`), we ask for this minimum with the rule:

```
#minimize [threshold_max(N,T):val(N,T):node(N)=T].
```

Every atom `threshold_max(N,T)` that is true is associated with a weight `T`, and the rule automatically minimizes the sum of the weights.

5.4.4. *Applications*

The three applications that are presented below illustrate the advantage of the approach and the methodology discussed above.

5.4.4.1. *Carbon starvation response in Escherichia coli*

Our declarative approach has been applied to the re-examination of a piecewise-linear (PL) differential equation model of the regulation network of the carbon starvation in *E. coli* [ROP 06]. This PL model was developed with the generate-and-test approach classically used for constructing GNR models. This led to a unique, instantiated and inconsistent model. A declarative approach can address at least the question of the existence of alternative models[7].

As long as environmental conditions are favorable, a population of *E. coli* bacteria grows quickly. The bacteria are in a state called exponential phase. Upon a nutritional stress due to carbon starvation, the bacteria are no longer able to maintain a fast growth rate. They enter in a state called stationary state. Their response can be reversed as soon as the environmental conditions become favorable again. The network (Figure 5.4.3.3) and several biological observations on interactions, paths (stationary states and paths leading from the exponential phase to the stationary phase and vice versa) and even constraints on the shape of the DNA (*supercoiling*) are given in [ROP 06]. A declarative analysis of this network has been presented in [COR 09] (based on a constraint logical program cooperating with a SAT solver). We resumed this analysis with our ASP implementation and we illustrate here the repairing of inconsistency.

Applying step 2 of the proposed method (section 5.4.3.1) led to an inconsistency that rigorously showed the non-existence of alternative models, i.e. with a reasoning not based on the inconsistency of only one particular instantiated model. Then, for repairing inconsistency, we choose the additivity constraints as non-mandatory to the extent that were not supported experimentally. The repairing process proposed two solutions, that is to remove one constraint among $\{K^{fis}_{gyrAB} \leq K_{gyrAB}, K_{topA} \leq K^{fis}_{topA}\}$. After biological investigations, it appeared that the first one should not be removed, but that the second could be removed, as it can be considered as not biologically plausible.

Computer performances stay very acceptable for solving such requests that require numerous recombination computations. For example, it is for determining the removable constraints that [COR 09] reports the highest computer time (around $25'$), with CLP and SAT solvers cooperating. It was in the case where all additivity constraints were removed in advance. This result was understandable because of the size of the solution space in this case. The same issue takes $4''$ when solved by our ASP implementation (with a Core 2 Duo 3 GHz processor with 4 GB of RAM).

5.4.4.2. *Drosophila embryo gap genes network*

This approach has been applied in [COR 12] to the regulatory network controlling the earliest steps of *Drosophila* embryo segmentation, i.e. the gap genes and their

7 There is no difficulty to translate a PL model into a Thomas model.

cross-regulations, under the additional control of maternal gene products [SÁN 01, JAE 04, ALV 06]. Three kinds of data were considered:

– Published molecular genetic studies enable the identification of the main actors (seven genes), as well as the establishment or the suggestion of cross-regulatory interactions.

– Qualitative information on the spatiotemporal expression profiles of the main genes involved in the process, giving seven regions with different stable states.

– Data available on the gap gene expression profiles for seven loss-of-function mutations, affecting maternal or gap genes.

On the basis of this combination of interaction and expression constraints, the challenge was to identify the minimal complying model(s), i.e. the model(s) involving all established regulatory edges, along with a minimal set of potential ones, while minimizing the number of distinct thresholds. In a first step, the consistency of the data (i.e. the existence of at least one consistent model) was proved in 3,338″, using a Linux PC with an Intel Core 2 Duo 2.4 GHz of processor and 2.9 GB of memory. Then, a unique minimal regulatory network was obtained in 1,016″, which included only two potential interactions (on 11). Surprisingly, from this network, there was a unique instantiation of the thresholds minimizing the number of threshold values per component (obtained in 368″). Finally, some properties concerning the kinetic parameters were deduced: 52 parameters fixed (over 72), 12 inequalities connecting a threshold and a parameter and 36 connecting two parameters.

5.4.4.3. *In vivo benchmarking of reverse engineering and modeling approaches interaction network*

The *in vivo* benchmarking of Reverse engineering and Modeling Approaches (IRMA) network [CAN 09] comprises five genes: Swi5, Ash1, Cbf1, Gal4 and Gal80, as well as one input (gal) and eight interactions (see Figure 5.6). These genes were chosen for the synthesis of the network so that different types of interactions were included, including transcription regulation and protein–protein interaction, thereby capturing the behavior of eukaryotic GRNs. Cantone *et al*. [CAN 09] explored the dynamics of the IRMA network by measuring each gene's expression level in response to two different perturbations using qRT-PCR. In the first set of experiments, they shifted yeast cells from a glucose to galactose medium ("switch-on" experiments) and in the second set of experiments, they shifted the cells from a galactose to glucose medium ("switch-off" experiments). The presence of galactose allows for increased transcription of Swi5 and is thus "switch-on", while the opposite is true for the "switch-off" experiments. From these data, two temporal series, composed of averaged gene expressions over five "switch-on" and four "switch-off" independent experiments, have been extracted.

Finding possible models of the IRMA network respecting these time series is a challenge proposed in [BAT 10]. The network is given in such terms that the order between the kinetic parameters is known. So the issue is to find a consistent order between thresholds and these parameters and between the thresholds themselves. Time series are formalized by CTL formulas of the form $EF(prop_1 \wedge EF(...EF(prop_n)...)$ where $n = 12$ for the switch-off experiment and $n = 10$ for the switch-on experiment. A condition $prop_i$ relates to the values of the components of a state and also to the derivative signs of these components. Batt *et al.* [BAT 10] propose a new modeling leading to more states and that takes into account *singular states* (states admitting for a component a threshold value), together with the use of the model checking tool NuSMV. They claim, when comparing their work, that they provide more precise results and efficient coding.

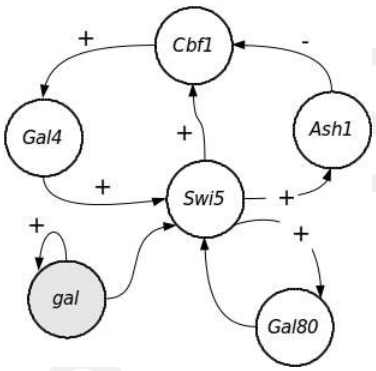


Figure 5.6. *Interaction graph of the IRMA network*

When applying the ASP declarative approach to this problem (not yet published work with Samuel Chorlton, Hamilton U.), we designed the appropriate constraints for expressing that a path satisfies a time series, while remaining in the Thomas framework, i.e. without additional states and without singular states. The same number of parametrizations (64) were exhibited[8] in $139''$ (compared to $885''$ on the same computer).

5.4.5. *Discussion*

Three topics will be discussed about the new way for modeling GRNs that is presented above: declarative versus functional approach, logic programming versus model checking tools and ASP assets.

8 Parameterizations were found identical, except for two of them that were erroneous in [BAT 10].

The application described in section 5.4.4.1 shows effectively the advantage of the declarative approach for building models. Instead of verifying conditions on an instantiated model and detecting the inconsistency of this specific model, these conditions are enforced on any models and this leads to demonstrate the absence of any consistent model. From that, an automatic consistency repairing is applied that allows a pertinent biological analysis. Also, we should note that the biologically oriented challenge mentioned in section 5.4.4.2 simply could not be defined outside of a declarative approach.

Model checking is based, for expressing conditions, on logic such as CTL. Logic programming also rely on some logics. So they could be compared from this point of view. Clearly, the example in section 5.4.2.1 (enforcing the existence of three different, a priori unknown, steady states) shows the weakness of the expression power of CTL, due to its lack of logical variables. But the difference between model checking tools and logic programming is best explained by their origins. The first ones were designed toward the verification of state transition systems (such as merged parallel programs), and the second ones toward (logic-based) programming. So if models checking tools are appropriate to verify properties (expressed typically in CTL) of a state-transition system, they are not to enforce properties. In a seminal paper on the topic [BER 04], exhibiting Thomas GRN consistent parametrizations requires an external process that enumerates all possible instantiated parametrizations. Along this line, the work reported in [BAT 10] needs an external process for exploiting counter examples provided by the used model checking tool. Also, biological properties such as activation\inhibition (see section 5.4.2.2) or supercoiling (section 5.4.4.1) are difficult to express with such software. Finally, implementing functionalities such as consistency repairing (see section 5.4.3.1) or inference of properties (see section 5.4.3.2) would require as well external processes. Concerning computer performance comparison, the result mentioned in section 5.4.4.3 is encouraging for the ASP software *gringo-clasp* [GEB 10], but more tests are necessary to get a significant opinion. Anyway, one should note that the solvers at the heart of both types of software have the same basis (SAT solver).

A first advantage of ASP concerns the non-monotonicity of the logic on which it is based, which distinguishes ASP from other logic programming technology. Benefits of this characteristic appears especially for expressing additivity constraints (see section 5.4.2.2), mutants (see section 5.4.2.3) and for avoiding inconsistency (see section 5.4.3.1) when adding knowledge that could provoke it with monotonic classical logics (see the annex for a clarification between the use of defaults and the paralogic maximization operators). Another advantage is the expressive power of the *gringo* language that is illustrated by the specification of Thomas GRNs (see section 5.4.1) in this language. We should point out also the efficiency of the *gringo-clasp* software when solving the ambitious challenge mentioned in section 5.4.4.2. It remains to discuss the delicate aspect of fixing the length of a path when representing a biological behavior. Ideally, it should be the maximal diameter of the possible transition graph,

which could be costly to compute. Practically, we fixed it empirically (for example 30 states in the application reported in section 5.4.4.3). A reasonable approach to this issue is certainly to use *iclingo*, an extension of *gringo-clasp*, that allows us to augment incrementally such a length until getting a solution.

5.5. Conclusions

We exhibit a new operational specification of CTL formulas and a new declarative way to model GRNs by using the logical paradigm ASP. We show the usefulness of this paradigm, particularly for the translation of CTL formulas such as EF and AF by relying on minimal stable models (see section 5.3.4) and for the formalization of some "generally true" specifications of GRNs by using defaults (see section 5.4.5). Also, we illustrate the expressive power, the inference capabilities and the efficiency of the *gringo-clasp* ASP software [GEB 10]. Among the improvements that we consider, one is related to CTL formulas of the form AF, interesting in view of synthesizing network (see section 5.3.4). But in the case where the number of possible transitions is huge and consequently the predicate `transition` not possibly fixed by advance, the described implementation of Thomas GRNs could be very costly for ensuring AF formulas. This implementation is appropriate for ensuring EF formulas provided an adequate limit `L` for the length for a path is given (see section 5.4.5): the number of states is then limited to `L`. Such a limit `L` is not any more suitable for ensuring AF formulas because the number of states would augment exponentially with `L`. To address this issue, we design actually an implementation admitting a limit but this time based on the number of states. Also, we are considering to improve the efficiency of the described implementation by using a constraint answer set solver, based on the cooperation of an AS solver and a constraint solver (see *cligcon* in [GEB 10]). This would allow us, for example, to extend the range of values for a variable without multiplying the corresponding Boolean variables produced at the grounding phase: it could be beneficial for the application described in section 5.4.4.3, where the range of values of the kinetic parameters and the thresholds has to be extended to the total of these entities. The declarative approach has been already been applied to other kind of biological model, such as Hopfield networks [BEN 13]. In the long term, with this perspective, we are considering exploring several other kind of networks, including Thomas networks integrating time (with delays) and networks composed of homogeneous networks or heterogeneous other networks (for better efficiency and better understanding of resulting properties).

5.6. Acknowledgments

We thank M. Gebser and T. Schaub for their initial help and fruitful exchanges and Samuel Chorlton for a review of a preceding text. This work was supported by Microsoft Research through its PhD Scholarship Program to Nicolas Mobilia.

5.7. Appendix on an advanced modeling for taking into additive constraints

The aim of this appendix is to present a more efficient modeling for additivity constraints. The following two issues arise: (1) escaping from a possible inconsistency that would result if these constraints would be imposed and (2) getting only the "most general" networks that is, intuitively, those which accept as many as possible additivity constraints compatible with the biological data. These issues are overcome in sections 5.4.2.2 and 5.4.3.1 by considering all possible networks, reflected by the rule enumerating the `kparam` literals (see section 5.4.1.3) and by using the paralogical maximization operator for obtaining only ASs with the maximum number of `addit` literals. One can note that, because of the exhaustive enumeration of the networks, this solution does not take full advantage of the default definition of `addit`.

However, both enumerating too many literals and using paralogic operators are costly. The following refined modeling reduces these costs as far as possible by taking advantage of the non-monotonicity of ASP. Also, it improves the previous modeling by associating additivity constraints even to edges that would not be labeled by any sign in the interaction graph but that would support, nonetheless, observability constraints as a result of the given behaviors.

5.7.1. *Lowering the enumeration of literals*

The `kparam` literals could come from three origins, which must be revisited: observability constraints due to the interaction graph, additivity constraints and biological behaviors.

Observability constraints coming from the interaction graph are now modeled by the following:

```
kparam(K, Ik) :- couple_kpr(K, Ik, _, _).
kparam(K_r, Ik_r) :- couple_kpr(_, _, K_r, Ik_r).
```

with

```
1{couple_kpr(K, Ik, K_r, Ik_r)
: obs_cond(S, N1, N, Ne1, K, K_r, Ik, Ik_r)}1
:- sign(S, N1, N, Ne1).
```

where `sign(S, N1, N, Ne1)`, provided by the modeler, means that `S` is the sign of the edge `N1, N, Ne1` and `obs_cond(S, N1, N, Ne1, K, K_r, Ik, Ik_r)` that this edge separates two cellular contexts of `N` identified by `Ik` and `Ik_r` such that their possible values `K` and `K_r` stay in the right order according to `S` (see below). Note that only one literal

`couple_kpr(K, Ik, K_r, Ik_r)` is necessary (makes the disjunction true in the left part) for ensuring the observability via the literals `kparam(K, Ik)` and `kparam(K_r, Ik_r)` that it implies. The curly brackets could even be suppressed, but we will not discuss this point here. The proper definition of `obs_cond` is:

```
obs_cond(S, N1, N, Ne1, K, K_r, Ik, Ik_r) :-
neighboring_cell_cont(N1, N, Ne1, Ik, Ik_r),
 param_obs(S, N1, N, Ne1, K, K_r).
```

where `neighboring_cell_cont(N1, N, Ne1, Ik, Ik_r)` ensures the existence of the two cellular contexts separated by the edge and `param_obs(S, N1, N, Ne1, K, K_r)` the right order of the parameters regarding the sign.

For modeling additivity constraints, the definition of `obs(S, N1, N, Ne1)` is slightly modified to take into account the new way of introducing the `kparam` literals and to accept edges with observability constraints non-necessarily coming from the interaction graph:

```
obs(S, N1, N, Ne1) :-
obs_cond(S, N1, N, Ne1, K, K_r, Ik, Ik_r),
kparam(K, Ik), kparam(K_r, Ik_r).
```

The `couple_kpr` literals due to additivity constraints are introduced by the rule:

```
1{couple_kpr(K, Ik, K_r, Ik_r)
: -param_obs(Sp, N1, N, Ne1, K, K_r)}1
:- neighboring_cell_cont(N1, N, Ne1, Ik, Ik_r),
obs(S, N1, N, Ne1),
opposite_sign(S, Sp),
not obs(Sp, N1, N, Ne1)
```

where `-param_obs(Sp, N1, N, Ne1, K, K_r)` is the negation of `param_obs(Sp, N1, N, Ne1, K, K_r)` and where one finds in the body the definition of `addit(S, N1, N, Ne1)` given in section 5.4.2.2. Note that this time, expressing the logical conjunction representing an additivity constraint requires every `couple_kpr` literal associated with a couple of cellular contexts separated by the edge `N1, N, Ne1`.

What remains is to remodel the rules defining the successor of a state. For the case where the species `N` changes its value at step `I + 1`, we get:

```
species(N, V, I, P) :- couple_ks(N, V, I, P, _, _).
kparam(K, k(N, CC)) :- couple_ks(N, _, _, _, K, CC).
```

with:

```
1{couple_ks(N, V_s, I+1, P, K, CC) :
   ineq_K(S, N, K, V) :  inc1(S, N, V, V_s)}1
:- diff(N, I, P), species(N, V, I, P),
step(I ; I+1, P), val(N, V), cell_context(N, CC, I, P).
```

where `ineq_K(S, N, K, V)` ensures that `K` and `V` are values of `N` ordered according to `S` and `inc1(S, N, V, V_s)` that `V_s` is `V+1` (respectively, `V-1`) if `S = p` (respectively, `m`). The parameter value `K` of the cellular context `CC` of `N` is fixed according to the transition between the two states.

For the case where the species `N` does not change its value at step `I + 1`, only a `species` literal has to be implied:

```
species(N, V, I+1 , P)
:- not diff(N, I, P), species(N, V, I, P),
step(I ; I+1, P).
```

and for the case where no species at all change, only `kparam` literal has to be implied:

```
kparam(V, k(N, CC))
:- not_any_diff(I, P), species(N, V, I, P),
step(I ; I+1, P),
cell_context(N, CC, I, P).
```

where `not_any_diff(I, P)` means that no species change.

5.7.2. *Conjunction of defaults and appropriate use of the paralogical maximization operator*

First, it is necessary to specify what is intended by the "most general" networks regarding additivity constraints. For this, we may raise two different questions. The first question is "for a set of parameters satisfying observability constraints and behaviors (e.g. paths), what are the ASs to be retained?", and the second question is "among the ASs that are answers to the first question, what are the desired ones?" As we will see below, the answer to the first question can be given in terms of (non-monotonic) logic, but the answer to the second question requires paralogical means.

For a set of a parameters satisfying observability constraints and behaviors, it appears natural to ask for keeping only ASs having additivity constraints for all edges of all species, if such an AS exists. If not, we would like to keep only the ASs having additivity constraints for all edges of the species for which it is possible.

For example, for the network (Figure 5.3) with a behavior implying only $K_b^{ab} = 2$, there are eight ASs (actually represented by the graphs $G_1, ..., G_6$) with additivity constraints for all edges of all species. But there are also other possible networks, for example with one edge of b with no additivity constraints. Unfortunately, the above modeling provides such undesirable networks due to possible additivity constraints for one edge that implies the non-additivity for some other edges. This is the case when insuring additivity constraints for the edge $a \rightarrow b$ with the additional parameter values $K_b = 1$, $K_b^a = 1$ and $K_b^b = 0$. These parameter values forbid additivity constraints for the edge $b \rightarrow b$.

A simple program would help in illustrating this last point and exhibiting a methodology to solve it. Let us consider the two following default rules that mimic the influences between the edges:

```
p2 | u :- not p1.
p1 | v :- not p2.
```

where `|` is the disjunction operator. They have the three ASs `{u, v}`, `{p2}` and `{p1}`. The challenge is to transform these rules so that we only get `{u, v}` when `p1` and `p2` are both unknown or false and `{p1}` if `p1` is true. First, the methodology consists of introducing the rules `c :- p1. c :- p2.` so that `not c` represents the case where both `p1` and `p2` are unknown or false, and second in completing the body of each of the original rules with a tautological term provided with a default impact power:

```
p2 | u :- not p1, 1{c, not c}1.
p1 | v :- not p2, 1{c, not c}1.
```

It has to be realized that when `not c` is true, then it is impossible to imply `p1` or `p2` and that only the AS `{u, v}` is obtained. If the rule `p1.` is added, we get the AS `{p1, c}`.

Applied to our case, this methodology simply asks for introducing in the body of the rule producing the additivity constraints the following terms:

```
1{not one_no_addit(N), one_no_addit(N)}1,
1{not one_no_addit, one_no_addit}1
```

where `one_no_addit(N)` means that one edge leading to the species `N` is not additive and `one_no_addit` means that one species does not have all its edges being additive. From a theoretical point of view, it should be noted that, by definition, from a network given by an interaction graph with labeled edges and without any additional behavior, this new modeling provides only ASs with additivity constraints for all edges of all species.

Meanwhile, there remain cases that need to be addressed. For example, for the network (Figure 5.3) with at least two stationary states, this new modeling provides, nonetheless, three ASs: one with the two edges of b being additive (graph G_6) and providing the stationary states $(0,1)$ and $(0,2)$, and the two others, respectively, with one and not any of these edges being additive and providing the stationary states $(0,1)$ and $(1,0)$. The parameters values of these last ASs come from the stationary states ($K_b = 1$ and $K_b^a = 0$) and the observability constraints ($K_b^{ab} = 2$, $K_b^b = 0$ or $K_b^b = 1$). So, they are acceptable from the "logical" point of view developed above, and these ASs are minimal. Consequently, discriminating some ASs among these three ASs requires definitively paralogic standards like the one presented in section 5.4.3.1, i.e. the winners are those having in the whole the greatest number of additive edges.

In summary, three points deserve to be retained: the minimal definition of the disjunction representing observability constraints, the methodology for building conjunctions of defaults and the distinctness regarding usage between the two ways to minimize the number of the resulting ASs.

5.8. Bibliography

[ALV 06] ALVES F., DILAO R., "Modeling segmental patterning in drosophila: maternal and gap genes", *Journal of Theoretical Biology*, vol. 241, pp. 342–359, 2006.

[BAR 03] BARAL C., *Knowledge Representation, Reasoning, and Declarative Problem Solving*, Cambridge University Press, New York, NY, 2003.

[BAT 10] BATT G., PAGE M., CANTONE I., *et al.*, "Efficient parameter search for qualitative models of regulatory networks using symbolic model checking", *Bioinformatics*, vol. 26, no. 18, pp. i603–i610, 2010.

[BEN 13] BEN AMOR H., CORBLIN F., FANCHON E., *et al.*, "Formal methods for Hopfield-like networks", *Acta Biotheoretica*, vol. 61, no. 1, pp. 21–39, March 2013.

[BER 04] BERNOT G., COMET J.-P., RICHARD A., *et al.*, "Application of formal methods to biological regulatory networks: extending Thomas' asynchronous logical approach with temporal logic", *Journal of Theoretical Biology*, vol. 229, no. 3, pp. 339–347, 2004.

[BES 89] BESNARD P., *An Introduction to Default Logic*, Springer, 1989.

[BOS 85] BOSSU G., SIEGEL P., "Saturation, nonmonotonic reasoning and the closed-world assumption", *Artificial Intelligence*, vol. 25, pp. 13–63, January 1985.

[CAN 09] CANTONE I., MARUCCI L., IORIO F., *et al.*, "A yeast synthetic network for in vivo assessment of reverse-engineering and modeling approaches", *Cell*, vol. 137, no. 1, pp. 172–181, 2009.

[CHA 03] CHABRIER N., FAGES F., "Symbolic model checking of biochemical networks", *Computational Methods in Systems Biology*, LNCS, Springer Berlin/Heidelberg, vol. 2602, pp. 149–162, 2003.

[CLA 82] CLARKE E.M., EMERSON E.A., "Design and synthesis of synchronization skeletons using branching time temporal logic", *Lecture Notes in Computer Science*, vol. 131, pp. 52–71, 1982.

[COR 09] CORBLIN F., TRIPODI S., FANCHON É., *et al.*, "A declarative constraint-based method for analyzing discrete genetic regulatory networks", *Biosystems*, vol. 98, pp. 91–104, 2009.

[COR 10] CORBLIN F., FANCHON É., TRILLING L., "Applications of a formal approach to decipher discrete genetic networks", *BMC Bioinformatics*, vol. 11, no. 1, pp. 1471–2105, 2010.

[COR 12] CORBLIN F., FANCHON E., TRILLING L., *et al.*, "Automatic inference of regulatory and dynamical properties from incomplete gene interaction and expression data", *Proceedings of the 9th International Conference on Information Processing in Cells and Tissues*, *IPCAT '12*, Springer-Verlag, Berlin, Heidelberg, pp. 25–30, 2012.

[GEB 10] GEBSER M., KAMINSKI R., KAUFMANN B., *et al.*, A user's guide to gringo, clasp, clingo, and iclingo (version 3.x), October 2010.

[GEL 88] GELFOND M., LIFSCHITZ V., *The Stable Model Semantics For Logic Programming*, MIT Press, pp. 1070–1080, 1988.

[GER 03] GERSHENSON C., "Classification of random boolean networks", *Artificial Life*, vol. 8, pp. 1–8, 2003.

[GRE 07] GREEN D., LEISHMAN T., SADEDIN S., "The emergence of social consensus in Boolean networks", *IEEE Symposium on Artificial Life, 2007, ALIFE '07,* pp. 402–408, 2007.

[HEY 05] HEYMANS S., NIEUWENBORGH D.V., VERMEIR D., "Synthesis from temporal specifications using preferred answer set programming", *ICTCS'05 (Italian Conference on Theoretical Computer Science)*, pp. 280–294, 2005.

[JAE 04] JAEGER J., BLAGOV M., KOSMAN D., *et al.*, "Dynamical analysis of regulatory interactions in the gap gene system of Drosophila melanogaster", *Genetics*, vol. 167, pp. 1721–1737, 2004.

[KAU 69] KAUFFMAN S.A., "Metabolic stability and epigenesis in randomly constructed genetic nets", *Journal of Theoretical Biology*, vol. 22, no. 3, pp. 437–467, 1969.

[KLA 06] KLAMT S., SAEZ-RODRIGUEZ J., LINDQUIST J.A., *et al.*, "A methodology for the structural and functional analysis of signaling and regulatory networks", *BMC Bioinformatics*, vol. 7, no. 15, p. 56, 2006.

[LAH 03] LÄHDESMÄKI H., SHMULEVICH I., YLI-HARJA O., "On learning gene regulatory networks under the Boolean network model", *Machine Learning*, vol. 52, nos. 1–2, pp. 147–167, 2003.

[QUE 82] QUEILLE J.-P., SIFAKIS J., "Specification and verification of concurrent systems in CESAR", *Symposium on Programming*, pp. 337–351, 1982.

[ROC 13] ROCCA A., RIBEIRO T., INOUE K., "Inference and learning of Boolean networks using answer set programming", *LNMR 2013*, p. 27, 2013.

[ROL 11] ROLI A., MANFRONI M., PINCIROLI C., *et al.*, "On the design of Boolean network robots", *Applications of Evolutionary Computation*, Springer, pp. 43–52, 2011.

[ROP 06] ROPERS D., DE JONG H., PAGE M., *et al.*, "Qualitative simulation of the carbon starvation response in *Escherichia coli*", *Biosystems*, vol. 84, no. 2, pp. 124–152, 2006.

[SÁN 01] SÁNCHEZ L., THIEFFRY D., "A logical analysis of the *Drosophila* gap-gene system", *Journal of Theoretical Biology*, vol. 211, pp. 115–141, 2001.

[THO 01] THOMAS R., KAUFMAN M., "Multistationarity, the basis of cell differentiation and memory. II. Logical analysis of regulatory networks in terms of feedback circuits", *CHAOS*, vol. 11, no. 1, pp. 180–195, 2001.