Revising Boolean Logical Models of Biological Regulatory Networks

Frederico Aleixo , Matthias Knorr , João Leite
NOVA LINCS, NOVA University Lisbon, Portugal
fp.aleixo@campus.fct.unl.pt,{mkn, jleite}@fct.unl.pt

Abstract

Boolean regulatory networks are used to represent complex biological processes, modelling the interactions of biological compounds, such as proteins or genes, with each other and with other substances in a cell. Creating and maintaining computational models of these networks is crucial for comprehending corresponding cellular processes, as they allow reproducing known behaviours and testing new hypotheses and predictions *in silico*. In this context, model revision focuses on validating and (if necessary) repairing existing models based on new experimental data. However, model revision is commonly performed manually, which is inefficient and prone to error, and the few existing automated solutions either only apply to simpler networks or are limited in their revision process, since they may not be able to produce a solution within a reasonable time frame or miss the optimal solution. In this paper, we develop a solution for revising logical models of Boolean regulatory networks, able to find repairs that are consistent with provided, possibly incomplete experimental data, and minimal w.r.t. the differences to the original network. We show that our solution can be used to revise different real-world Boolean logical models very efficiently, surpassing a previous solution in terms of solved instances and with a considerable margin w.r.t. processing time.

1 Introduction

The field of systems biology has flourished for the last two decades. It focuses on making the most of the principles of engineering, mathematics, physics, and computer science to model real-life biological systems, in an attempt to understand them through a holistic lens that allows us to gain new insight about these systems in ways that were not possible before (Siegel et al. 2006). One of the cornerstones of this field is the ability to accurately model real-world biological systems, with the ultimate goal of acquiring a better understanding of the complex processes that take place in cells, as doing so may lead to new discoveries and theories about living organisms. To this end, biological regulatory networks (BRNs) enable us to computationally generate models that allow for the emulation of patterns and behaviors of real-world systems, as well as the testing of hypotheses and the identification of predictions *in silico* (Jong 2002).

Biological regulatory networks are sets of biological compounds, such as for example proteins or genes, that interact with each other and with other substances in a cell. These networks can be classified into continuous models, single-molecule level models, and logical (qualitative) models (Karlebach and Shamir 2008). Among these, continuous and single-molecule level models use real variables to represent the concentration values of compounds as well as equations to model the changes of these values over time, requiring considerable amounts of detailed experimental data. On the other hand, logical models, first introduced by Glass and Kauffman (1973) and Thomas (1973), allow us to abstract from actual concentration values, considering concentration thresholds instead to represent whether a compound is active or inactive. This usually requires far less information than quantitative models, which means that logical models have the advantage that they can be used with incomplete, imprecise and noisy information regarding the biological system.

Among such logical models, Boolean logical models, also known as Boolean networks (BNs) have been extensively used, e.g., as models of gene regulation networks and other biological systems (Salinas, Gómez, and Aracena 2022). They utilize graphs to represent the topology of the network, i.e., the possible interactions between compounds, and regulatory functions for each compound to determine its state based on the (Boolean) state – active or inactive – of the compounds that can affect it, and whether they have an activating or inhibiting effect.

To be able to fully leverage BRNs requires the existence of models that are as accurate as possible w.r.t. the considered biologicial process. This necessitates revising models of BRNs as new experimental data is gathered, to assure that the models remain consistent with this new experimental data, for which inconsistencies must be identified and corrected. This is often a manual process carried out by domain experts, but it is a complicated, labor-intensive process, and prone to error, having to deal with huge combinatorics of possible revisions to match the observed behavior.

While the automatic creation of models of BRNs (Ostrowski et al. 2016; Réda and Wilczyński 2020; Mitsos et al. 2009; Schaub, Siegel, and Videla 2014; Videla et al. 2012) as well as the analysis of regulatory models (Melkman, Tamura, and Akutsu 2010; Tamiura and Akutsu 2009; Dubrova and Teslenko 2011; Khaled and Benhamou 2019; Baral et al. 2004) has already been extensively explored, automation in model revision, in particular for BNs, has received considerably less attention. Notably, existing work

on revision of BNs is limited in terms of possible revision operations on the regulatory functions (Merhej, Schockaert, and Cock 2017; Lemos, Lynce, and Monteiro 2019; Chai 2019) or focuses on simpler models for the interactions of compounds (Gebser et al. 2010; Mobilia et al. 2015). The sole exception is ModRev (Gouveia, Lynce, and Monteiro 2020b; Gouveia, Lynce, and Monteiro 2020b), a system for revising BNs that is very general in terms of applicable modification operations, able to handle different forms of dynamics and deal with incomplete experimental data, but may fail to find a solution in reasonable time, or return a suboptimal solution whenever the more efficient search strategy it implements is used. ModRev also implements a general search strategy that would in principle guarantee an optimal solution, but the number of instances where no solution is found within a reasonable time frame would only increase.

In this paper, we present a system for analysing and revising Boolean logical networks, dubbed ARBoLoM, that is as general as ModRev in terms of applicable modification operations and the forms of dynamics and experimental data it can handle, provides solutions that are as close as possible to the given network, following the principle of minimal change, and is elaboration-tolerant and flexible with respect to the criteria employed to determine such minimality. Developed in ASP and leveraging the state of the art system clingo (Gebser et al. 2019), ARBoLoM shows that both consistency checking and model repair for real networks can be done highly efficiently, surpassing ModRev in both number of solved instances and required time, by quite some margin.

The remainder of the paper is structured as follows. We first recall BNs and ASP in Sec. 2 and Sec. 3 resp.; then, we present our solution for revising BNs, in Sec. 4, and we evaluate our approach in Sec. 5, before we conclude in Sec. 6.

2 Boolean Regulatory Networks

A Boolean Regulatory Network, or Boolean Network (BN), is a set of biological compounds (be they proteins, genes, metabolites or other) that interact with each other and with other substances in a cell, representing the complex biological processes that take place in that environment. A typical way of modeling a BN makes use of a *regulatory graph*.

Definition 1. *A* regulatory graph *is a directed graph* $G = (V, E)$*, where* $V = \{v_1, ..., v_n\}$ *is the set of vertices (nodes) representing the regulatory compounds, and* $E = \{(u, v, s) : u, v \in V, s \in \{+, -\}\}$ *is the set of signed edges representing the* interactions *between compounds such that* E *does not contain* $(u, v, +)$ *and* $(u, v, -)$ *for any* $u, v \in V$*.*

An edge with $s = +$ is called *positive interaction* (or *activation*), representing that u activates v, while an edge with $s = -$ is called *negative interaction* (or *inhibition*), representing that u inhibits v.[1] A node with no incoming edges in G is called *input node*. Such nodes represent external stimuli, and their values do not change over time.

Example 1. *Fig. 1 (on the left) shows regulatory graph* $G = (V, E)$ *with* $V = \{v_1, v_2, v_3, v_4\}$ *and* $E = \{(v_1, v_2, -), (v_1, v_3, +), (v_2, v_1, +), (v_2, v_3, +), (v_4, v_2, +), (v_4, v_3, -)\}$.

[1] Sometimes, nodes can activate and inhibit another node under different circumstances, but we do not consider this case here.

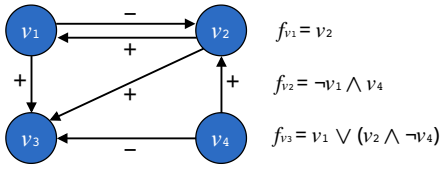

Figure 1: Example of a Boolean logical model.

Regulatory graphs do not clarify how different compounds that affect the same node interact with each other for that node's activation. To mend this, BNs make use of regulatory functions for each compound, thus specifying what combination of regulators produces an effect on a given compound. This gives rise to Boolean logical models.

Definition 2. *A* Boolean logical model M *of a regulatory network is defined as a tuple* (V, F) *where* $V = \{v_1, v_2, ..., v_n\}$ *is the set of variables representing the regulatory compounds of the network such that* v_i *is assigned to a value in* $\{0, 1\}$*, and* $F = \{f_1, f_2, ..., f_n\}$ *is the set of Boolean functions such that* f_i *defines the value of* v_i *and where* $f_i = v_i$ *if* v_i *is an input node.*

Regulatory functions of input nodes may be omitted (cf. Fig. 1), but in our solution we use the explicit representation, common when involving implementational aspects (Klarner, Bockmayr, and Siebert 2014).

Example 2. *Fig. 1 presents a boolean logical model with* G *from Ex. 1 and regulatory functions for* G *on the right.*

In line with (Lemos, Lynce, and Monteiro 2019; Gouveia, Lynce, and Monteiro 2020b), we adopt the Blake canonical form (BCF), a special case of the disjunctive normal form (DNF), to represent the regulatory functions of each biological compound. In the BCF, a function is represented by the disjunction of all its *prime implicants* (Blake 1937). The BCF is a canonical form, i.e., it is unique up to reordering (of disjuncts), which considerably facilitates the search for possible functions when looking for repairs. Recall, that a conjunction of literals is an implicant of a Boolean function if, whenever the conjunction takes the value 1, then so does the function, and that a *prime implicant* is an implicant that does not absorb any other implicant (Crama and Hammer 2011). As no compound can occur both positively and negatively in a specific regulatory function by definition of regulatory graphs, verification of prime implicants in such a function thus reduces to ensuring that no disjunct is contained in the other. E.g., f_{v_3} from Fig. 1 is in BCF and both disjuncts are prime implicants.

Dynamics The state of a BN changes over time as the various compounds that are a part of it interact with each other. The dynamics of a network refer to this process of interaction between compounds that alters the network's state.

When modeling a BRN, one of the main challenges is to ensure that the dynamics of the model correctly represent the dynamics one can observe in the real system. Network states are used to provide a representation of the activations of the network's compounds.

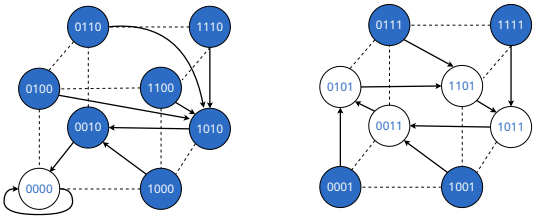

Figure 2: STG of the model in Ex. 2 for synchronous updating scheme, with v_4 inactive on the left, and active on the right (adapted from (Gouveia 2021)).

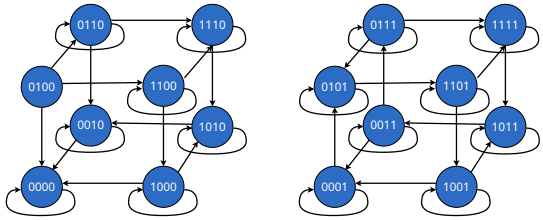

Figure 3: STG of the model in Ex. 2 for asynchronous updating scheme, with v_4 inactive on the left, and active on the right (adapted from (Gouveia 2021)).

Definition 3. *The* network state *of a BN with n compounds is a vector $S = [v_1, v_2, ...v_n]$ where v_i is the value of the variable representing the i-th compound of the network.*

Clearly, for Boolean logical models, the number of different states in a network is given by 2^n. E.g., if nodes 1 and 3 in Ex. 2 are active and the other two are not, then the state will be represented by 1010.

We can then use state transition graphs (Naldi et al. 2011) to describe how networks evolve over time.

Definition 4. *A State Transition Graph (STG) is a directed graph $G_{STG} = (S, T)$ where S is the set of vertices representing the different states of the network, and T is the set of edges representing the viable transitions between states.*

Two update schemes are employed to update the values of nodes in a BN: the synchronous and the asynchronous updating scheme (Faure et al. 2006; Garg et al. 2008).

In the synchronous updating scheme, at each time step, all compounds are updated simultaneously. Each network state has at most one successor (cf. Fig. 2), which is biologically less realistic and less accurate for analysing systems.

In the asynchronous updating scheme, at each time step, only one regulatory function may be applied. This is closer to what is observable in real systems, since these changes seldomly tend to take place simultaneously. With *n* compounds in a network, each state can have at most n possible state transitions (including a transition to itself - cf. Fig. 3).

Boolean networks may enter so-called *attractors* during updates, i.e., a set of network states that form a cycle. Such cycles are linked to many important cellular processes, such as phenotypes, cell cycle phases, cell growth, differentiation, and apoptosis (Hopfensitz et al. 2012). Among them, *stable states* are attractors that contain only a single state, whereas *cyclic attractors* contain several. An example for a cyclic attractor appears on the right of Fig. 2 with 0101, 1101, 1011 and 0011, whereas 0000 is a stable state on the left of Fig. 3. Hence, stable states represent a particular case of dynamics in BNs which can be checked for consistency as well.

3 Answer Set Programming

We briefly recall relevant notions and notation for logic programs under the answer set semantics.

Answer Set Programming (ASP) is a form of declarative programming oriented towards difficult, primarily NP-hard, search problems (Lifschitz 2008). It has become an attractive paradigm for knowledge representation and reasoning, thanks to its appealing combination of rich, yet simple modeling languages with powerful solving engines.

A logic program P is a finite set of rules r of the form:

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

where each a_i ($0 \leq i \leq n$, $0 \leq m \leq n$) is an *atom*. An atom takes the form $p(t_1, ..., t_k)$, where *p* is a predicate symbol of arity *k*, and $t_1, ..., t_k$ are terms, built from variables and constants of the (implicit) language of the program of *r*. Atoms to the left of the arrow ($\leftarrow$) are said to be the head of the rule, whereas (negated) atoms to the right of the arrow are the body of the rule. We define as literals both an atom a as well as its negation, $not\ a$. We say that the head of *r* is a_0 ($head(r) = a_0$), and the body of *r* is the set of all literals that occur in its body (also known as body literals) ($body(r) = \{a_1, ..., a_m, \ not\ a_{m+1}, ..., \ not\ a_n\}$). We call rules with $m = n = 0$ *facts* and admit that a_0 is $\perp$ to represent *constraints*, and we commonly omit $\leftarrow$ in the former case and $\perp$ in the latter. The ground instantiation of a program, $ground(P)$ is obtained by replacing all (first-order) variables with constants from the given language of the program in all possible ways. A set of ground atoms S is then an answer set of $ground(P)$ if S is the subset-minimal model of the reduct $ground(P)^S$ obtained as $\{a_0 \leftarrow a_1, ..., a_m \mid r$ of the form (1) in $ground(P)$ and $\{a_{m+1}, \ldots, a_n\} \cap S = \emptyset\}$. ASP then also admits a number of additional language constructs to express choices, constraints on these, aggregates, as well as optimization statements, that we will explain in the following sections.

4 Revision of Boolean Networks

Model revision of BNs consists of two parts: consistency checking and repairing. Consistency checking is about contrasting the experimental results, obtained from real-world biological systems, with the ones generated using computational models. If a computational model is able to replicate the experimental observations, then the model is consistent with them. Otherwise, it is inconsistent requiring a repair.

As shown in Fig. 4, our solution employs ASP encodings for consistency checking and repairing using clingo, while processing of input data and integration of the different components is realized in Python. Before we discuss in detail our solution to consistency checking and repairing, we first show how to represent the necessary information for this revision process. As our solution makes use of ASP encodings, this amounts to providing corresponding sets of

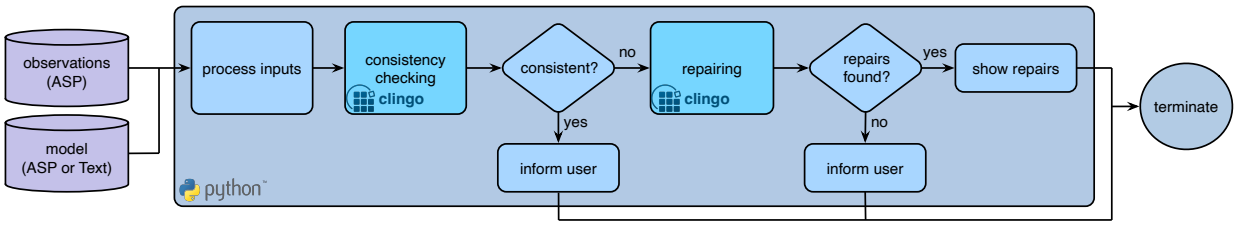

Figure 4: ARBoLoM – overview of the revision process of BNs

facts, similar in spirit to such representations in the literature (Gebser et al. 2010; Lemos, Lynce, and Monteiro 2019; Gouveia, Lynce, and Monteiro 2020b).

We first encode a regulatory graph $G = (V, E)$ as a set of facts using $compound/1$ to represent all elements of V and a set of facts using $regulates/3$ where $regulates(u, v, s)$ represents that compound u regulates v with the sign s (if s is 0, then u is an activator, otherwise an inhibitor). E.g., we indicate that v_1 inhibits v_2 by writing $\mathsf{regulates}(\mathsf{v1}, \mathsf{v2}, 1)$.

To represent regulatory functions of boolean logical models, we make use of the predicates $function/2$ and $term/3$ to represent the number of terms of a regulatory function and the constituting terms, respectively. I.e., $\mathsf{function}(\mathsf{v1}, 1)$ represents that v_1's regulatory function has one term, while $\mathsf{term}(\mathsf{v1}, 1, \mathsf{v2})$ represents that term 1 of v_1's regulatory function contains v_2. E.g., f_{v_3} from Ex. 2 can be represented by $\mathsf{term}(\mathsf{v3}, 1, \mathsf{v1})$, $\mathsf{term}(\mathsf{v3}, 2, \mathsf{v2})$, and $\mathsf{term}(\mathsf{v3}, 2, \mathsf{v4})$ together with the atoms using $regulates$ to indicate the sign.

Finally, we encode observations, i.e., the experiments used to assess and revise BNs, by means of two predicates, $experiment/1$ and $observation/4$. The predicate *experiment* is used to represent the id of an experiment, while $\mathsf{observation}(\mathsf{e}, \mathsf{t}, \mathsf{c}, \mathsf{s})$ represents the observed state s (0 for inactive, 1 for active) of compound c, at time step t of experiment e. Note that, for stable state observations where the network does not change its state over time, no argument t is required, and $oberservation/3$ is used instead. Also note that observations in Fig. 4 are assumed to be in ASP format already, but any adaptation to a different format can be easily realized either by adapting the processing of inputs or by employing an adequate pre-processing step.

4.1 Consistency Checking

For checking consistency, in our solution, we consider three different modes of interaction between compounds, namely stable states (for which no time component is considered), and the synchronous and asynchronous update scheme. Here, for the sake of readability, we spell out in detail the ASP encoding for checking consistency for stable states, even if it amounts to a rather routine encoding, and then we briefly explain the additions for the other schemes.[2]

Given the model representation of the BN and (possibly incomplete) observations for some experiments, the main idea of consistency checking is comparing, for each experiment, whether the observed states of the compounds match those that are determined based on the regulatory functions.

Listing 1 shows the encoding for stable state consistency which assumes as input the ASP specification of the network and the experimental data. We next explain this encoding.

First, observation data is converted into curated observations to be able to handle incomplete experimental data. Line 1 simply passes the observed data for the compounds where it exists, while Line 2, by means of a cardinality constraint, allows one to fix the observation of a compound C (active or inactive) when such observation is not available for it in experiment E. This may cause additional inconsistencies, but we will see in a moment how this can be avoided. Based on the curated observations, we can determine when compounds are active according to the model. Recall that regulatory functions are presented in BCF, i.e., a disjunction of conjunctions (its prime implicants). Now an implicant I_NO of C is inactive if one of its activating compounds R is inactive (Line 3) or one of its inhibiting compounds R is active (line 4). If, in the considered experiment E, at least one term I_NO of C is not inactive, then C is active, as one true term suffices to make the entire disjunction true (Line 5). In Line 6, we determine input nodes, namely by identifying those compounds C whose regulatory function has a single term that is a single activator of itself. Then, Line 7 is used to determine if these input nodes are active based on the experimental data alone. With this in place, inconsistency of compound C in experiment E can be verified if C is active according to the model, but observed to be inactive (Line 8), or if C is inactive according to the model, but observed to be active (Line 9). The third argument of $inconsistent/3$ is used to identify the type of inconsistency that was encountered. Finally, in Line 10, the optimization statement minimizes the number of inconsistencies found. The rationale is that the choices made on unseen experimental data may create additional inconsistencies. Here, our aim is to emulate the behavior of the real world system as closely as possible, so that, e.g., missing observations should lead to as few inconsistencies as possible, and the minimization ensures that. Using $\#show$ statements (omitted from Listing 1), information over predicates $inconsistent/3$, $experiment/1$, and $curated\_observation/3$ is extracted from the obtained model. This information indicates which compounds are inconsistent for which experiments and the corresponding curated observations, required for repairing the model later.

The encodings for synchronous and asynchronous time update schemes follow the same main idea presented for stable states, only a (discrete) time component is added. I.e., at

[2]The complete encodings can be found at https://github.com/fpaleixo/arbolom/tree/main/encodings/consistency.

Listing 1: Stable state consistency checking.

```
1   curated_observation(E,C,S) :- observation(E,C,S).
2   1{curated_observation(E,C,0);curated_observation(E,C,1)}1 :- not observation(E,C,_), experiment(E), compound(C).
3   implicant_inactive(E,C,I_NO) :- function(C,I), term(C,I_NO,R), regulates(R,C,0), curated_observation(E,R,0).
4   implicant_inactive(E,C,I_NO) :- function(C,I), term(C,I_NO,R), regulates(R,C,1), curated_observation(E,R,1).
5   active(E,C) :- function(C,I), not implicant_inactive(E,C,I_NO), experiment(E), term(C,I_NO,_).
6   input_compound(C) :- compound(C), function(C,1), regulates(C,C,0), #count{R : term(C,1,R)} = 1.
7   active(E,C) :- curated_observation(E,C,1),input_compound(C).
8   inconsistent(E,C,1) :- active(E,C), curated_observation(E,C,0).
9   inconsistent(E,C,0) :- not active(E,C), curated_observation(E,C,1).
10  #minimize{1,E,C : inconsistent(E,C,_)}.
```

time step T, curated observations are created based on the observations at T (recall that experiments now have such a temporal component) from which inactive implicants are determined at T. This, in turn, allows us to conclude which compounds are active at $T+1$, which provides the means to detect inconsistencies (at $T+1$) between the supposed result and the observed one. Note that we can only check for time series consistency for $T > 0$, since, for $T = 0$, we have no information regarding the previous timepoint.

For synchronous consistency checking, recall that all regulatory functions are applied at each time step. If each compound can replicate the observed value at each timepoint, given the value of its regulators in the previous timepoint, then the model is consistent. To account for missing obervations (and states of compounds) at timepoint 0, we guess the state of those compounds, determine corresponding curated observations, and minimize inconsistencies as before.

In asynchronous consistency checking, only one regulatory function may be applied at each time step. If the compound, whose regulatory function is applied, can replicate the observed value at the time step of its application given the values of its regulators in the previous time step, then the model is consistent. The corresponding encoding is quite similar to the synchronous case, but we need to ensure that only one compound changes at each time step. This is achieved by adding rules that determine compounds whose curated observations change from time T to $T+1$, and then validate with a constraint that only one compound changes at each time step. In addition, the inconsistency check is adjusted to only take into account differences between the determined state of a compound and its corresponding observation if that compound changed in the previous time step.

4.2 Model Repair

Repair of inconsistent models is realized by individually modifying each of the regulatory functions that have been identified as inconsistent so that they be consistent with the expected behavior from the experiments. This requires as input the ASP representation of the network together with the curated observations and the obtained inconsistencies.

The modifications we consider are adding and removing regulators, changing regulators from activator to inhibitor and vice-versa, and changing the function's format, i.e., changing the number of terms as well as their composition. Combining these allows for a large variety of possible repaired functions. In the spirit of minimal change, our objective is to find repairs that are as close as possible to the original functions, using the following order of priority:

1. Minimize the changes to the number of function terms;
2. Minimize the changes to the function's regulators;
3. Minimize the changes to the signs of regulators;
4. Minimize the changes to the format of each term.

While the order of criteria 2.–4. is easily adjustable (as we will see), the choice for criterion 1. turns out to be important for our developed solution, because the main search algorithm employs an iterative deepening search on the number of changes to the number of function terms. The central idea is to start searching for solutions with the same number of function terms as the function to be repaired, and if no solution is found, simultaneously search for solutions with one more or one less function term, picking the better solution according to the other criteria, if one exists, and iterate this process otherwise. This iteration is limited by the lower bound of 0 function terms and by an upper bound determined based on the number of positive observations. This is possible because, in the worst case, each positive observation is only covered by a single term in the function, so if we cannot find a function with at most as many terms as existing distinct observations (for the function to be repaired), then no such candidate can exist. This is further improved upon by not counting identical transitions of such observations.

Similar to consistency checking, we discuss with more detail our repair solution for stable state repairs (Listing 2), and then the main differences for time-series repairs.[3]

First (Lines 1–2), we define constants for the compound whose function we want to repair and for the number of nodes in the function (i.e., the number of disjuncts). Both these values can be set during the execution of the iterative deepening process. Then, we generate the possible candidates for the regulatory function. Since there is only one compound to be regulated, we use a more compact representation of the function. Here, $activator/1$ and $inhibitor/1$ indicate the kind of regulation for a compound w.r.t. that compound, and $node\_regulator/2$ allows us to represent that a certain compound occurs in a specific node (disjunct). Thus, Lines 3–9 determine for each compound whether it is

[3]The complete encodings can be found at https://github.com/fpaleixo/arbolom/tree/main/encodings/repairs.

Listing 2: Repair under stable state observations.

```
1  #const compound = c.
2  #const node_number = n.
3  fixed_regulator(C) :- fixed(C,compound), compound(C).
4  fixed_regulator(C) :- fixed(C,compound,_), compound(C).
5  fixed_activator(C) :- fixed(C,compound,0), compound(C).
6  fixed_inhibitor(C) :- fixed(C,compound,1), compound(C).
7  activator(C) :- fixed_activator(C).
8  inhibitor(C) :- fixed_inhibitor(C).
9  1 {activator(C); inhibitor(C)} 1 :- compound(C), not fixed_activator(C), not fixed_inhibitor(C).
10 available_node_ID(1..TERM_NO) :- function(compound, TERM_NO).
11 available_node_ID(TERM_NO + 1..node_number):- function(compound, TERM_NO), node_number > TERM_NO.
12 {node_regulator(N,C) : compound(C)} :- available_node_ID(N).
13 node_ID(N) :- node_regulator(N,_).
14 :- fixed_regulator(C), not node_regulator(_,C).
15 :- node_number != #count{N : node_regulator(N,R)}.
16 :- COMMON_VARNO = #count{C : node_regulator(N1,C), node_regulator(N2,C)}, node_ID(N1), node_ID(N2), N1 != N2,
       N1_VARNO = #count{ C : node_regulator(N1,C) }, COMMON_VARNO = N1_VARNO.
17 experiment_negative_node(E,N) :- node_regulator(N,C), curated_observation(E,C,0), activator(C).
18 experiment_negative_node(E,N) :- node_regulator(N,C), curated_observation(E,C,1), inhibitor(C).
19 experiment_positive_node(E,N) :- not experiment_negative_node(E,N), node_ID(N), curated_observation(E,_,_).
20 :- experiment_positive_node(E,N), curated_observation(E,compound,0).
21 :- not experiment_positive_node(E,_), curated_observation(E,compound,1).
22 original_regulator(C) :- regulates(C,compound,_).
23 present_regulator(C) :- node_regulator(N,C).
24 missing_regulator(C) :- original_regulator(C), not present_regulator(C).
25 extra_regulator(C) :- not original_regulator(C), present_regulator(C).
26 sign_changed(C) :- regulates(C,compound,0), inhibitor(C).
27 sign_changed(C) :- regulates(C,compound,1), activator(C).
28 missing_node_regulator(ID,R) :- term(compound, ID, R), node_ID(ID), not node_regulator(ID, R).
29 extra_node_regulator(ID,R) :- node_regulator(ID, R), term(compound, ID, _), not term(compound, ID, R).
30 #minimize{1@3,C : missing_regulator(C)}.
31 #minimize{1@3,C : extra_regulator(C)}.
32 #minimize{1@2,C : sign_changed(C)}.
33 #minimize{1@1,N,C : missing_node_regulator(N,C)}.
34 #minimize{1@1,N,C : extra_node_regulator(N,C)}.
```

an activator or an inhibitor taking also into account possible information from the user on certain compounds being fixed to be a regulator of the considered compound (or even of a specific kind). In Lines 10–11, available nodes are determined based on the number of disjuncts of the original function possibly adjusting if n in the iteration is greater than that previous number. This allows us to assign a number of compounds to each such node with a choice rule (Line 12) and also project on node IDs (Line 13). Then, a number of restrictions ensure that fixed regulators occur somewhere in the function (Line 14), the number of nodes (containing some compound) is correct (Line 15), and that the function be in BCF (Line 16) by verifying that no node can have the same number of compounds as it does have common compounds with another node. We determine nodes as negative if one of its activators is inactive (Line 17) or one of its inhibitors is inactive (Line 18), and as positive if it is not negative (Line 19). Then, Lines 20–21 discard any function where the observed result differs from those calculated by the function, which suffices to find all valid solutions.

To find an optimal one according to the criteria established earlier, we also determine in Lines 22–25 regulators that have been added/removed from the function, in Lines 26–27 changes from activator to inhibitor and vice-versa, as well as, in Lines 28–29, whether a certain compound is missing or has been added to a certain node in the repaired function. Then, Lines 30–34 encode the optimization criteria, where, e.g., $\#\mathsf{minimize}\{1@3, \mathsf{C} : \mathsf{missing\_regulator}(\mathsf{C})\}.$ represents that any missing regulator C is associated a weight 1 with priority 3 (statements with higher priority are optimized first). The info on activators and inhibitators as well as node regulators can then be extracted from the answer set to obtain the repaired function (if it exists). Subsequently, we can re-run this encoding with the next compound's function to be repaired.

The encodings of time series repairs is quite similar. In fact, the definition of the function (Lines 1–16) as well as the optimization criteria (Lines 22–34) from Listing 2 can be used as is. The only difference lies in time-dependent determination of negative and positive nodes and the subsequent elimination of undesired solutions given the time-series experiments. While synchronous repairs perform updates simultaneously, for asynchronous repairs, we slightly adjust to only consider the validation for the node that changed.

Abbr.	#C	#I	#SS	Avg. Reg.	Max. Reg.
MCC	10	35	1	3.500	6
FY	10	27	12	2.700	5
TCR	40	57	7	1.425	5
SP	19	57	7	3.000	8
Th	23	35	3	1.520	5

Table 1: Boolean models data with model's abbreviations (Abbr.), their number of compounds (#C), number of compound interactions (#I), number of stable states (#SS), average number of regulators (Avg.Reg.), and maximum number of regulators (Max.Reg.).

#	1	2	3	4	5	6	7	8	9	10	11	12
F	5	25	50	100	0	0	0	0	0	0	0	0
E	0	0	0	0	5	10	15	20	25	50	75	0
R	0	0	0	0	0	0	0	0	0	0	0	1
A	0	0	0	0	0	0	0	0	0	0	0	0

#	13	14	15	16	17	18	19	20	21	22	23	24
F	0	0	0	0	0	0	0	25	50	100	5	10
E	0	0	0	0	0	0	0	5	25	50	25	10
R	5	10	15	0	0	0	0	0	0	0	5	5
A	0	0	0	1	5	10	15	0	0	0	5	5

Table 2: Corruption configurations. Displayed is an ID for the configuration (#), and the probability (0%-100%) of applying each of the four different corruption operations: (F)unction change, (E)dge sign flip, (R)emove a regulator, and (A)dd a regulator.

	stable state		synchronous		asynchronous	
Model	**# Cons.**	**# Inc.**	**# Cons.**	**# Inc.**	**# Cons.**	**# Inc.**
MCC	1372	1028	543	1857	657	1743
FY	728	1672	602	1798	760	1640
TCR	545	1855	390	2010	423	1977
SP	413	1987	306	2094	378	2022
Th	514	1886	358	2042	449	1951

Table 3: Statistics on consistent (#Cons.) and inconsistent (#Inc.) created models for each of the three modes of interaction.

5 Evaluation

We have implemented ARBoLoM[4] as described in the previous section and we report here on its evaluation.

We first assess whether our system is able to perform revision of Boolean models efficiently, following the methodology described in (Gouveia, Lynce, and Monteiro 2020a). To this end, we have used five well-known biological models:

- The network that controls the Mammalian Cell Cycle (**MCC**) (Faure et al. 2006);
- The cell-cycle regulatory network of Fission Yeast (**FY**) (Davidich and Bornholdt 2008);
- The T-Cell Receptor signalling network (**TCR**) (Klamt et al. 2006);
- The Segment Polarity network (**SP**), which plays a role in the fly embryo segmentation (Sanchez, Chaouiya, and Thieffry 2008);
- And the regulatory network controlling T-helper cell differentiation (**Th**) (Mendoza and Xenarios 2006).

The precise characteristics for each of these models can be found in Table 1, showing the variety in terms of numbers of compounds, regulation interactions, and stable states, and the average and max. number of regulators per function.

For each of these models, we generated new (corrupted) versions using four operations (with varying probability X):

- Each regulatory function is altered with probability X (0.1 to 1). This alteration can change the number of terms in the function and/or the format of the terms;
- The sign of each edge (regulator) in the functions is changed with probability X (0.05 to 0.75).
- Each regulator in the functions can be removed with probability X (0.01 to 0.15).
- For each function, any regulator not in that function may be added as a regulator with probability X (0.01 to 0.15).

In our tests, we used 24 different configurations of corruption operations, varying on how many of these operations are used simultaneously and their probabilities. Table 2 shows the details of these configurations. For each configuration, 100 corrupted models were generated providing a total number of 2400 models for each of the five models and each of the three modes of interaction between compounds. The corresponding distribution into consistent and inconsistent models is summarized in Table 3 (based on the consistency analysis of our tool).

The tests were conducted on an Intel(R) Xeon 2.2GHz running Linux with a timeout of 3600s and 13GB of memory, although memory usage never exceeded 1.4GB.

For testing stable state observations, we used available information on the stable states of the five networks[5] to create the experimental data. In the experiments, all 12000 tested models were either identified as consistent or repaired in around 0.1 seconds on average, and even the most complicated cases were solved in under one second. Table 4 summarizes the results showing the times for the entire revision process, consistency checking, and repair, restricted, in the case of repair, to the corrupted models where repair was realized (which explains why the average of the total revision process is often smaller than the average on repairs).

For experiments with time-series observations, we created a number of experiments with a variable number of characteristics, resulting in 4 kinds of observations, from 1 experiment with 3 timepoints to 5 experiments with 20 timepoints. Table 4 also includes the results for synchronous and asynchronous observations for 5 experiments and 20 timepoints.

While processing times increase with time observations (slightly more for synchronous observations, likely because more changes happen at every time step that need to be validated), all instances are identified as consistent or repaired, and the overall revision process can still be done on average within 1 second, with a maximum of 5 seconds in a few cases. We note that among the five models, **TCR** has slightly higher processing times, likely due to the higher number of compounds.

[4]https://github.com/fpaleixo/arbolom

[5]https://filipegouveia.github.io/ModRev/

Model	Revision Proc.	stable state			synchronous			asynchronous		
		Avg.	Min.	Max.	Avg.	Min.	Max.	Avg.	Min.	Max.
MCC	Revision (s)	0.01	< 0.01	0.44	0.14	0.03	0.59	0.11	0.03	0.56
	Consistency (s)	< 0.01	< 0.01	0.44	0.04	0.03	0.47	0.03	0.03	0.45
	Repair (s)	0.02	< 0.01	0.07	0.13	0.03	0.53	0.10	0.03	0.50
FY	Revision (s)	0.04	< 0.01	0.45	0.14	0.03	0.51	0.09	0.03	0.53
	Consistency (s)	< 0.01	< 0.01	0.45	0.04	0.03	0.47	0.03	0.03	0.52
	Repair (s)	0.05	< 0.01	0.15	0.13	0.03	0.45	0.09	0.03	0.32
TCR	Revision (s)	0.11	0.01	0.81	0.76	0.11	5.02	0.54	0.09	3.08
	Consistency (s)	0.01	< 0.01	0.62	0.13	0.11	0.59	0.11	0.08	0.86
	Repair (s)	0.12	0.01	0.80	0.76	0.09	4.88	0.51	0.09	2.92
SP	Revision (s)	0.10	< 0.01	0.69	0.60	0.06	2.64	0.30	0.05	1.21
	Consistency (s)	< 0.01	< 0.01	0.51	0.07	0.05	0.47	0.05	0.04	0.40
	Repair (s)	0.11	< 0.01	0.68	0.45	0.05	2.52	0.30	0.05	1.14
Th	Revision (s)	0.07	< 0.01	0.47	0.41	0.06	2.21	0.28	0.05	1.20
	Consistency (s)	< 0.01	< 0.01	0.46	0.07	0.06	0.53	0.06	0.05	0.49
	Repair (s)	0.07	< 0.01	0.38	0.39	0.06	2.08	0.27	0.05	1.11

Table 4: Revision process times for each family of models, under stable state observations, and synchronous and asynchronous observations.

Average times	1 – 3	1 – 20	5 – 3	5 – 20
Revision (s)	0.06	0.11	0.13	0.41
Consistentency (s)	< 0.01	0.02	0.02	0.07
Repair (s)	0.07	0.11	0.13	0.39

Table 5: Average revision times for varying numbers of experiments X and numbers of timepoints Y – represented by X–Y – under synchronous observations for model **Th**.

We also examined how the size and number of the experiments influence the revision process. Table 5 presents the data gathered from the revision of the corrupted instances of **Th** under synchronous observations, showing that increasing the number of experiments and/or the number of timesteps used, increases both consistency checking and repair times. Similar observations are obtained for the other models and under asynchronous observations.

We also compared ARBoLoM with ModRev (Gouveia, Lynce, and Monteiro 2020a), which also uses ASP, but only to check for consistency, while repair is implemented as a search in C++[6], with the following order of optimization criteria:

1. Minimise the changes to the function's regulators;
2. Minimise the changes to the signs of regulators;
3. Minimise the number of function change operations.

The third item incorporates the minimization of changes to the number of function terms and to the format of each term.

ModRev's reported experimental evaluation uses the same set of real-world models, the same set of corruption configurations – creating 100 instances for each model and configuration – the same stable states and the same method for creating observations. However, the set of created models[7] appears to have been restricted to only contain instances with at least one modification (our test set may contain unchanged models, mainly for configurations where the probabilities of change were low), impacting on the frequency with which revision is required. For the sake of fairness, the following results compare ARBoLoM's performance on ModRev's test dataset, with the reported performance of ModRev.[8]

Table 6 presents the statistics regarding the number of successfully revised models and average revision times (for time-series observations with 5 experiments and 20 timesteps), both for ARBoLoM and ModRev. While ARBoLoM solves each of the 12000 instances in less than a second on average (apart from **TCR** in synchronous mode), ModRev fails to solve a considerable number of instances for all models in either mode – in particular the (vast) majority for some more complicated corruption configurations (e.g., 11, 18, 19, 23, 24 - cf. Table 2 and the detailed ModRev results). We can also observe that ARBoLoM commonly presents revision times better than ModRev by 1 to 3 orders of magnitude, which is even more significant given that the reported average times do not take into account the instances where ModRev timed out. ARBoLoM's better performance can be explained by a combination of our ASP encoding of the repair process, the efficiency of *clingo*, and possibly also by the selected optimization criteria. Even though the obtained repaired solution will commonly differ between the two approaches due to the different optimization criteria, the empirical guarantee to find a solution (very quickly) certainly is in favor of ARBoLoM. It is worth noting that ModRev's most efficient search algorithm – the one used in the reported experiments – may fail to find consistent functions, even though they may exist, and even if they are found, there is no guarantee that the function found is optimal (Gouveia 2021). Also, the optimization criteria adopted by ARBoLoM can very easily be changed, since it amounts to adjusting the preference order of the minimization criteria in the repair encoding (see Listing 2), leaving only fixed the

[6]In fact, (Gouveia 2021) also mentions an implementation that partially uses ASP in the revision process, which was nevertheless discarded for reasons of efficiency

[7]https://filipegouveia.github.io/ModRev/validation

[8]ModRev's experiments were conducted on an Intel(R) Xeon 2.1GHz running Linux, with the same timeout of 3600s and limited to 2GB of memory. The results are available at https://filipegouveia.github.io/ModRev/results-global-v1.3.1.pdf

	synchronous				asynchronous			
	ModRev		ARBoLoM		ModRev		ARBoLoM	
Model	Solved	Avg.	Solved	Avg.	Solved	Avg.	Solved	Avg.
MCC	53.04%	277.92	100%	0.37	77.87%	93.91	100%	0.28
FY	82.54%	382.02	100%	0.29	94.54%	437.54	100%	0.19
TCR	78.96%	104.57	100%	1.34	86.08%	179.56	100%	0.64
SP	53.37%	72.63	100%	0.89	49.25%	241.06	100%	0.68
Th	73.33%	28.78	100%	0.91	83.00%	50.38	100%	0.69

Table 6: Comparison of model revision with ModRev for time series observations with 5 experiments and 20 timesteps indicating the percentage of solved instances (out of 2400 in each case) and the average revision time (in s) of solved instances.

5 Exp. 20 steps	Avg.	Min.	Max.	Std. Dev.
Revision (s)	3.11	0.24	14.40	2.98
Consistentency (s)	0.27	0.22	0.78	0.08
Repair (s)	3.25	0.26	14.13	2.98

Table 7: Experimental results for revising models combining all 5 realistic models under asynchronous observations.

first criterion of number of changes on function terms.

Finally, to test whether our solution scales up to larger networks, we built an artificial boolean network composed of the five real networks we tested before. The composing parts are only loosely connected (one compound is shared between two of them), but the corruption may introduce connections when adding regulators, and the combinatorics in the revision process are considerably increasing compared to the individual networks due to using 101 compounds. We created 10 corrupted models for each of the same 24 corruption configurations using the most demanding experimental setup (5 experiments and 20 steps) under the arguably most realistic setting of asynchronous observations. Our results show that all 240 corrupted models are correctly identified as consistent (31) or inconsistent, and then repaired (209). Table 7 summarizes the experiments for this revision process, showing that, on average, revision is done within 3s, and consistency checking can still be done very efficiently. In some cases, the revision process takes up to 13 seconds, but mainly for configurations where corruptions occur with high probability. For example, the corrupted model with the highest revision time was corrupted under configuration #11 resulting in 33 corrupted regulatory functions within that model, whose individual repair time varied between 0.1 and 4s, requiring several changes to restore consistency in some cases. As the repair process is oblivious of the corruption process, it does not necessarily restore the original functions in such cases, but rather finds a function that is close to the corrupted one, but consistent with the observations. We believe that in realistic settings such profound changes to existing/established models are not that likely, hence reducing the cases where this happens. In any case, ARBoLoM is perfectly able to handle the revision process for a network integrating the 5 real networks with considerable modifications, and we believe that these results show that our solution is in fact capable of efficiently revising realistic Boolean networks.

6 Conclusions

Throughout this paper, we presented ARBoLoM, a tool for revising Boolean Networks that is able to check the consistency of a network given a set of experiments, and repair the network, whenever necessary, providing optimal solutions with respect to the developed optimization criteria. Our approach leverages ASP encodings and iterative deepening on the number of used terms in the revised regulatory function to deal with the high arising combinatorics. Our solution permits the usage of incomplete experimental data, allows the user to provide specific information for the revision process to acertain conditions that must hold for the revised function(s), and is flexible as to the order of the adopted optimization criteria. The evaluation of our tool shows that it is able to perform revision very efficiently for realistic BNs, outperforming ModRev – the only previously existing tool with the same scope – by a significant margin, in particular in terms of solved instances and processing times.

Other related work on revising Boolean models is restricted in terms of applicable repairs (no addition of regulators and changes to the formula refer to switching AND and OR) and considered functions (Lemos, Lynce, and Monteiro 2019), or only admits very specific changes and in part only to acyclic graphs (Chai 2019). Other work employs "rules of thumb" for model revision (Merhej, Schockaert, and Cock 2017), aiming to find least contradictory repairs, but it is limited to synchronous observations and uses a different evaluation of regulatory functions. Further work on model revision uses interaction graphs (Mobilia et al. 2015) for revising generic regulatory networks and ASP for revising sign consistency models (Gebser et al. 2010) respectively. Both formalisms differ from BNs, and the former approach was observed to not always provide results that accurately model the biological counterpart, whereas the latter formalism is arguably less expressive.

In the future, we may consider extending our tool to allow for mixed observations, i.e., synchronous and asynchronous, or admit additional optimization criteria that together with the existing ones can be picked by an expert user to adjust the revision process to their needs. Another possible improvement could be achieved by introducing explanations to the presented repairs, making the process more transparent and allowing the user to better understand why a certain modification was done. Finally, we can explore with synthesizing regulatory functions based on the developed approach that uses iterative deepening to find suitable candidates.

Acknowledgments

We would like to thank Filipe Gouveia for clarifying some aspects regarding the ModRev system, and the anonymous reviewers for their comments that helped improve the paper. This work is supported by NOVA LINCS (UIDB/04516/2020) with the financial support of FCT.IP.

References

Baral, C.; Chancellor, K.; Tran, N.; Tran, N.; Joy, A.; and Berens, M. 2004. A knowledge based approach for representing and reasoning about signaling networks. *Bioinformatics* 20(Suppl 1):i15–i22.

Blake, A. 1937. *Canonical expressions in Boolean algebra*. Ph.D. Dissertation, The University of Chicago.

Chai, X. 2019. *Reachability Analysis and Revision of Dynamics of Biological Regulatory Networks*. Ph.D. Dissertation, École centrale de Nantes.

Crama, Y., and Hammer, P. L. 2011. *Boolean functions: Theory, algorithms, and applications*. Cambridge University Press.

Davidich, M., and Bornholdt, S. 2008. Boolean network model predicts cell cycle sequence of fission yeast. *PloS one* 3:e1672.

Dubrova, E., and Teslenko, M. 2011. A SAT-based algorithm for finding attractors in synchronous boolean networks. *IEEE/ACM Transactions on Computational Biology and Bioinformatics* 8(5):1393–1399.

Faure, A.; Naldi, A.; Chaouiya, C.; and Thieffry, D. 2006. Dynamical analysis of a generic boolean model for the control of the mammalian cell cycle. *Bioinformatics* 22(14):e124–e131.

Garg, A.; Cara, A. D.; Xenarios, I.; Mendoza, L.; and Micheli, G. D. 2008. Synchronous versus asynchronous modeling of gene regulatory networks. *Bioinformatics* 24(17):1917–1925.

Gebser, M.; Guziolowski, C.; Ivanchev, M.; Schaub, T.; Siegel, A.; Thiele, S.; and Veber, P. 2010. Repair and prediction (under inconsistency) in large biological networks with answer set programming. *Principles of Knowledge Representation and Reasoning: Proceedings of the 12th International Conference, KR 2010*.

Gebser, M.; Kaminski, R.; Kaufmann, B.; and Schaub, T. 2019. Multi-shot ASP solving with clingo. *Theory and Practice of Logic Programming* 19(1):27–82.

Glass, L., and Kauffman, S. A. 1973. The logical analysis of continuous, non-linear biochemical control networks. *Journal of Theoretical Biology* 39(1):103–129.

Gouveia, F.; Lynce, I.; and Monteiro, P. T. 2020a. Modrev - model revision tool for boolean logical models of biological regulatory networks. In Abate, A.; Petrov, T.; and Wolf, V., eds., *Computational Methods in Systems Biology - 18th International Conference, CMSB 2020, Konstanz, Germany, September 23-25, 2020, Proceedings*, volume 12314 of *Lecture Notes in Computer Science*, 339–348. Springer.

Gouveia, F.; Lynce, I.; and Monteiro, P. T. 2020b. Revision of boolean models of regulatory networks using stable state observations. *J. Comput. Biol.* 27(2):144–155.

Gouveia, F. 2021. *Model Revision of Boolean Logical Models of Biological Regulatory Networks*. Ph.D. Dissertation, Instituto Superior Técnico, Universidade de Lisboa.

Hopfensitz, M.; Müssel, C.; Maucher, M.; and Kestler, H. A. 2012. Attractors in boolean networks: a tutorial. *Computational Statistics* 28(1):19–36.

Jong, H. 2002. Modeling and simulation of genetic regulatory systems: A literature review. *Journal of computational biology : a journal of computational molecular cell biology* 9:67–103.

Karlebach, G., and Shamir, R. 2008. Modelling and analysis of gene regulatory networks. *Nature reviews. Molecular cell biology* 9:770–80.

Khaled, T., and Benhamou, B. 2019. An asp-based approach for attractor enumeration in synchronous and asynchronous boolean networks. *Electronic Proceedings in Theoretical Computer Science* 306:295–301.

Klamt, S.; Saez-Rodriguez, J.; Lindquist, J. A.; Simeoni, L.; and Gilles, E. D. 2006. A methodology for the structural and functional analysis of signaling and regulatory networks. *BMC Bioinformatics* 7(1).

Klarner, H.; Bockmayr, A.; and Siebert, H. 2014. Computing symbolic steady states of boolean networks. In Wąs, J.; Sirakoulis, G. C.; and Bandini, S., eds., *Cellular Automata*, 561–570. Cham: Springer International Publishing.

Lemos, A.; Lynce, I.; and Monteiro, P. 2019. Repairing boolean logical models from time-series data using answer set programming. *Algorithms for Molecular Biology* 14:9.

Lifschitz, V. 2008. What is answer set programming? In *Proceedings of the 23rd National Conference on Artificial Intelligence - Volume 3*, AAAI'08, 1594–1597. AAAI Press.

Melkman, A. A.; Tamura, T.; and Akutsu, T. 2010. Determining a singleton attractor of an AND/OR boolean network in time. *Information Processing Letters* 110(14-15):565–569.

Mendoza, L., and Xenarios, I. 2006. A method for the generation of standardized qualitative dynamical systems of regulatory networks. *Theoretical Biology and Medical Modelling* 3(1).

Merhej, E.; Schockaert, S.; and Cock, M. D. 2017. Repairing inconsistent answer set programs using rules of thumb: A gene regulatory networks case study. *International Journal of Approximate Reasoning* 83:243–264.

Mitsos, A.; Melas, I. N.; Siminelakis, P.; Chairakaki, A. D.; Saez-Rodriguez, J.; and Alexopoulos, L. G. 2009. Identifying drug effects via pathway alterations using an integer linear programming optimization formulation on phosphoproteomic data. *PLoS Computational Biology* 5(12):e1000591.

Mobilia, N.; Rocca, A.; Chorlton, S.; Fanchon, E.; and Trilling, L. 2015. Logical modeling and analysis of regulatory genetic networks in a non monotonic framework. In *Bioinformatics and Biomedical Engineering*. Springer International Publishing. 599–612.

Naldi, A.; Remy, E.; Thieffry, D.; and Chaouiya, C. 2011. Dynamically consistent reduction of logical regulatory graphs. *Theoretical Computer Science* 412(21):2207–2218.

Ostrowski, M.; Paulevé, L.; Schaub, T.; Siegel, A.; and Guziolowski, C. 2016. Boolean network identification from perturbation time series data combining dynamics abstraction and logic programming. *Biosystems* 149:139–153. Selected papers from the Computational Methods in Systems Biology 2015 conference.

Réda, C., and Wilczyński, B. 2020. Automated inference of gene regulatory networks using explicit regulatory modules. *Journal of Theoretical Biology* 486:110091.

Salinas, L.; Gómez, L.; and Aracena, J. 2022. Existence and non existence of limit cycles in boolean networks. In *Automata and Complexity*, volume 42, 233–252. Springer.

Sanchez, L.; Chaouiya, C.; and Thieffry, D. 2008. Segmenting the fly embryo: logical analysis of the role of the segment polarity cross-regulatory module. *The International Journal of Developmental Biology* 52(8):1059–1075.

Schaub, T.; Siegel, A.; and Videla, S. 2014. *Reasoning on the Response of Logical Signaling Networks with ASP*. 49–92.

Siegel, A.; Radulescu, O.; Borgne, M. L.; Veber, P.; Ouy, J.; and Lagarrigue, S. 2006. Qualitative analysis of the relation between DNA microarray data and behavioral models of regulation networks. *Biosystems* 84(2):153–174.

Tamiura, T., and Akutsu, T. 2009. Detecting a singleton attractor in a boolean network utilizing SAT algorithms. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences* E92-A(2):493–501.

Thomas, R. 1973. Boolean formalization of genetic control circuits. *Journal of Theoretical Biology* 42(3):563–585.

Videla, S.; Guziolowski, C.; Eduati, F.; Thiele, S.; Grabe, N.; Saez-Rodriguez, J.; and Siegel, A. 2012. Revisiting the training of logic models of protein signaling networks with ASP. In *Computational Methods in Systems Biology*. Springer Berlin Heidelberg. 342–361.