Modeling Multi-Valued Biological Interaction Networks using Fuzzy Answer Set Programming

Mushthofa Mushthofa[*,†,¶] Steven Schockaert[‡], Ling-Hong Hung[§], Kathleen Marchal[‖] and Martine De Cock[*,§]
[*]Department of Applied Mathematics, Statistics and Informatics, Ghent University, Ghent, Belgium,
Email: {Mushthofa.Mushthofa, Martine.DeCock}@UGent.be [†]Department of Computer Science, Bogor Agricultural University, Bogor, Indonesia,
Email: mush@ipb.ac.id [‡]School of Computer Science & Informatics, Cardiff University, UK,
Email: SchockaertS1@cardiff.ac.uk [§]Institute of Technology, University of Washington Tacoma, USA,
Email: {lhhung, mdecock}@uw.edu [¶]Bioinformatics Institute Ghent, Ghent University, 9052 Ghent, Belgium
[‖]Department of Information Technology (INTEC, iMINDS), Ghent University, 9052 Ghent, Belgium

***Abstract*—Fuzzy Answer Set Programming (FASP) is an extension of the popular Answer Set Programming (ASP) paradigm that allows for modeling and solving combinatorial search problems in continuous domains. The recent development of practical solvers for FASP has enabled its applicability to real-world problems. In this paper, we investigate the application of FASP in modeling the dynamics of Gene Regulatory Networks (GRNs). A commonly used simplifying assumption to model the dynamics of GRNs is to assume only Boolean levels of activation of each node. Our work extends this Boolean network formalism by allowing multi-valued activation levels. We show how FASP can be used to model the dynamics of such networks. We experimentally assess the efficiency of our method using real biological networks found in the literature, as well as on randomly-generated synthetic networks. The experiments demonstrate the applicability and usefulness of our proposed method to find network attractors.**

I. Introduction

In biological systems, genes are known to interact with each other in a complex and dynamic way. Briefly, each gene's activation state can influence the activation states of other genes, either positively or negatively. These interactions can be modelled using a graph structure, which is usually called a Gene Regulatory Network (GRN). It determines the patterns of activation states of the genes, which in turn affects the phenotypic behavior of the system.

One of the most important concepts in modeling the dynamics of GRNs are the so-called *attractors*, which are the sets of states to which the system converges. An attractor usually corresponds to the observed characteristics/phenotypes of the biological system [1]. For example, the attractors of a GRN usually correspond to the expression patterns of the genes in the network for specific types of cells [2], [3]. In studying the dynamics of such networks it is therefore of importance to be able to identify their attractors.

In systems biology, one of the most popular approaches to formalise a GRN is to use a so-called Boolean Network (BN) [4]–[6]. Boolean networks represent genes as nodes that can take on Boolean values (intuitively representing the activation levels of the genes), while interactions between the genes are represented as Boolean functions that determine the value of each node at a certain time, depending on the current values of the other genes. The state transitions of a GRN and their attractors can be readily represented using such a formalism.

There have been numerous works about computational tools to simulate the dynamics of Boolean networks and to compute their attractors, mostly using logic-based techniques such as Binary Decision Diagrams (BDDs) or Boolean SAT solvers [7]–[12]. More recently, Answer Set Programming (ASP) has become a particularly interesting framework for modeling GRNs and Boolean networks [13]–[16].

ASP is a popular declarative programming paradigm which allows for an easy and intuitive encoding of many combinatorial search and optimisation problems [17], [18]. The availability of fast and efficient solvers for ASP, such as clasp [19] and DLV [20], allows for the application of ASP in various fields [21], [22]. Despite its flexibility and expressive power, however, ASP lacks the ability to directly encode problems in continuous domains.

Having only two levels of activation is sometimes not always enough to fully understand the dynamics of real biological systems. For example, in [3], [23]–[26], examples of systems are given whose dynamics can only be modelled by considering more than two activation levels. One classic example is the *lac operon* regulatory system, which is a set of genes that controls the production of the proteins needed to metabolise lactose in enteric bacterias, such as *Escherichia coli* (see e.g., [27]). In this case, it has been shown that one of the key attractors cannot be characterized using a Boolean encoding (because of the so-called "leaky-expression"). Despite the importance of multi-valued activation levels for modeling gene regulatory networks, only limited progress has been made on developing simulation tools that can support them. To the best of our knowledge, only one tool has been developed that supports multi-valued activation levels [24].

In this paper, we propose the use of Fuzzy Answer Set Programming (FASP) [28] as a computational framework to simulate the dynamics of multi-valued regulatory networks.

FASP is a form of declarative programming that extends ASP by allowing graded truth values in atomic propositions and using fuzzy logic connectives to aggregate these truth values. Recent work on the implementation of a FASP solver, such as [29]–[33], has opened the door to the application of FASP for solving real-world applications. Other frameworks dealing with the extension of ASP, or more generally, logic programming into the fuzzy domains have been proposed in the literature, e.g., [34]–[39]. While we have specifically chosen to use the FASP framework and the corresponding solver from [32], other multi-valued extensions of ASP might also be suitable for the purpose of modeling the dynamics of multi-valued regulatory networks.

Here, we propose an encoding of the dynamics of multi-valued biological interaction networks that can be executed/solved using the FASP solver proposed in [32], and we prove the correctness of this encoding. We then perform an extensive benchmark test using synthetic networks as well as real biological networks found in the literature to show the efficiency and applicability of this method. The results indicate that the method is efficient for the size of the networks typically used in the Boolean/discrete modeling of regulatory networks (up to around a few dozen genes in the network).

This paper extends our previous work [40] with the following contributions: (1) we provide complete formal definitions of multi-valued networks and their dynamics, (2) we provide detailed proofs of the correctness of the encoding, (3) we extend the framework to address the problems with the encoding of cyclic attractors, in particular, in the case of asynchronous updates, (4) we describe a method to perform automatic encoding of the network structure into fuzzy propositions and the implementation of a tool to perform this (FASPG), and (5) we extend the experiments to include synthetic networks and show the performance of our methods for increasingly large networks, including the computation of cyclic attractors of synthetic networks under different schemes of updates.

The remainder of this paper is structured as follows: we first describe related work in Section II, and present the preliminaries on Boolean networks and the theoretical background on (F)ASP in Section III . We then formally define the multi-valued networks and present our FASP-based encoding in Section IV. Section V describes the FASPG tool that implements the proposed method, as well as providing an automatic encoding for the network. Section VI contains the experiments we conducted to test the feasibility and efficiency of the proposed method, while Section VII provides a conclusion.

II. Preliminaries

A. Boolean networks

A Boolean network consists of a set of nodes (representing genes/proteins that interact with each other) and a set of edges, representing any interaction between the nodes. Formally, a Boolean network [1] is a pair $G = \langle X, F\rangle$, where $X = \langle x_1, \ldots, x_n\rangle$ is a tuple of Boolean variables representing the nodes of the network, while $F = \langle f_1, \ldots, f_n\rangle$ is a tuple of Boolean functions encoding the incoming edges for each node, as well as their interactions. An assignment

Fig. 1. A Boolean network model with two genes. Edges with arrowed tips are activating interactions and edges with blunt tips are repressing (inhibiting) links.

$v \equiv \langle v(x_1), \ldots, v(x_n)\rangle$, where each $v(x_i) \in \{0, 1\}$, is called a *network state*. The set of all $2^{|X|}$ network states is called the *state space* of the Boolean network, denoted by S. Each function f_i is a Boolean expression involving standard Boolean connectives over the constants 0 and 1 and the set of variables in X. The value of the expression f_i, given the assignment v for the variables in X is denoted by $f_i(v)$. The tuple F of functions defines the state mapping function $f : S \to S$ as follows: the state $f(v)$ for a state v is the state $w \equiv \langle f_1(v), \ldots, f_n(v)\rangle$.

Example 1. *Consider the Boolean network $G_1 = \langle\{x, y\}, F\rangle$, with two nodes, as depicted in Figure 1. The Boolean functions F describing the interaction between the nodes in the network are given by*

$$f_1(x, y) = \neg x \vee y$$
$$f_2(x, y) = x \vee \neg y$$

The state space S is a set of 4 states $\{\langle 0, 0\rangle, \langle 0, 1\rangle, \langle 1, 0\rangle, \langle 1, 1\rangle\}$. In this case, if $v = \langle 0, 0\rangle$, then we have $f(v) = \langle 1, 1\rangle$.

The dynamics of a Boolean network are defined by the transitions between the network states as determined by the given *update scheme* used in the network. Earlier models of biological networks (e.g, in [1], [41]) assumed that at each time step, all the nodes are updated (using their corresponding function f_i) in a synchronous manner. This simplifying assumption allows for an easy computation of the transitions between the states of the network. However, this assumption does not necessarily hold in practice, for example due to the difference in speed of one chemical reaction compared to the others [42]. Thus, a more realistic setting for modelling biological networks would be to not assume any synchronicity between the update on each node. Instead, in this asynchronous setting, we assume that at each time step, a single node is non-deterministically chosen to be updated [42], [43]. Thus, rather than having one possible next state, each state can potentially have n possible next states (where n is the number of nodes in the network).

To formally explain the concept of update scheme, we first define the following notions. The Hamming distance function over pairs of valuations/states, $\Delta : S \times S \to \{1, \ldots, n\}$ is defined as the number of nodes for which the states have a different value, i.e.

$$\Delta(v, w) = |\{x \in X \mid v(x) \neq w(x)\}| \tag{1}$$

with $v, w \in S$. The dynamics of a Boolean network are modelled using a directed graph $\langle S, \hookrightarrow\rangle$, called the State Transition Graph (STG), where the edge relation $\hookrightarrow$ is determined by the considered update scheme, as follows:

(i) For the synchronous update scheme: $v \hookrightarrow w$ iff $f(v) = w$.

(ii) For the asynchronous update scheme: $v \hookrightarrow w$ iff either $v = w$ and $f(v) = v$, or $\Delta(v, w) = 1$ and $\Delta(w, f(v)) < \Delta(v, f(v))$

Intuitively, with the synchronous update scheme, the network transitions from a state to another state by applying *all* of the update functions to all of the nodes. In contrast, with the asynchronous state, the transition from a state to another is done by applying the update functions to only one node. The condition $\Delta(w, f(v)) < \Delta(v, f(v))$ intuitively means that after applying the update functions to one node, the new state w should be closer to $f(v)$ than v, since the updated node in w should have the same values as in $f(v)$. We also say that in the relation $v \hookrightarrow w$, w is a successor state of v in the STG. The following definition defines the concept of an attractor [7], [42].

Definition 1. *An attractor of a Boolean network G is a minimal set of states (w.r.t. set inclusion) A such that:*

- *For $A' = \{a' \mid a \hookrightarrow a', a \in A\}$, it holds that $A' = A$.*
- *For any $a \in A$, if the state a is visited in a transition, then the probability of visiting a again after a finite number of transitions is equal to one.*

The size of an attractor A is defined to be $|A|$.

An attractor consisting of only one state, i.e. an attactor of size 1, is called a single state attractor. The state that makes up a single state attractor is called a steady state. An attractor of size 2 or larger is usually called a *cyclic* attractor. Note that, in general, the attractors of a Boolean network under different update schemes are also different. Steady states, however, do not depend on the particular choice of update scheme [44]. This is due to the fact that for a steady state x, $f(x) = x$, which means that $\{x\}$ is an attractor w.r.t. both update schemes.

For the asynchronous state, note that even though in general, a state s can have multiple successor states (i.e., there can be two different states t_1 and t_2 such that $s \hookrightarrow t_1$ and $s \hookrightarrow t_2$), such a state s cannot be part of an attractor, for the following reason. Suppose that s were indeed part of an attractor A. Then we also have $t_1 \in A$ and $t_2 \in A$, by Definition 1. This means that s should be reachable from both t_1 and t_2 (since $s \in A$). Now, it cannot be the case that both any path from t_1 to s always contains t_2 and any path from t_2 always contains t_1. Suppose that there is a path from t_1 to s that does not go through t_2; the case where there is a path from t_2 to s that does not go through t_1 is analogous. Then there must be a loop $s \hookrightarrow t_1 \hookrightarrow \ldots \hookrightarrow s$ that does not contain t_2. Hence, once we visit either s or t_1, the probability of visiting t_2 within a finite number of steps cannot be equal to 1. This contradicts the assumption that t_2 was also in the attractor A. Thus, only states which have a unique successor can be included in an attractor (both in the synchronous and asynchronous case).

Example 2. *Consider the Boolean network given in Example 1. The dynamics of the network under the synchronous update scheme can be described using the STG given in Figure 2. For example, starting from the state $\langle 0, 1\rangle$, we move to the state $\langle 1, 0\rangle$, i.e., $f(\langle 0, 1\rangle) = \langle 1, 0\rangle$. From the figure, we can see that the Boolean network has 2 attractors: the steady state $\langle 1, 1\rangle$ and the cyclic attractor of size 2: $\{\langle 1, 0\rangle, \langle 0, 1\rangle\}$.*

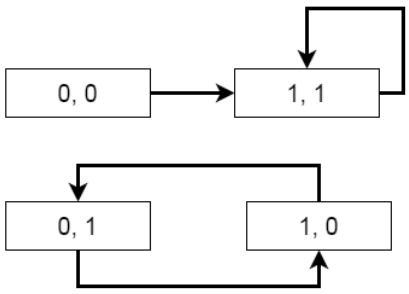

Fig. 2. State Transition Graph of the Boolean network G_1 under the synchronous update scheme

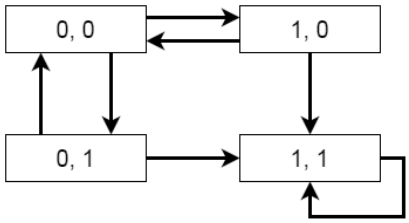

Fig. 3. State Transition Graph of the example Boolean network under the asynchronous update scheme. Note that due to the fact that each of the states $\langle 0, 0\rangle$, $\langle 0, 1\rangle$, $\langle 1, 0\rangle$ has multiple outgoing edges, there is no guarantee that the dynamics of the network will be confined to any possible subset of these states, and hence there is no attractor associated with these states. Consequently, the only attractor of this network is the steady state $\langle 1, 1\rangle$

An asynchronous update implies that the dynamics of the Boolean network may no longer be deterministic, since from a particular state, there can be more than one outgoing edge in the state transition graph. Figure 3 depicts the STG of the example Boolean network when the asynchronous update scheme is used. In this case, we only have one attractor, which is the steady-state $\langle 1, 1\rangle$.

B. Fuzzy answer set programming

Fuzzy Answer Set Programming (FASP) [28] is an extension of the well-known ASP paradigm into the fuzzy domain, where atomic propositions can take a graded truth value and rules are defined using fuzzy logic connectives. Assume that atomic propositions are drawn from a set of universal symbols $\mathcal{B}$. An interpretation is defined as a function $I : \mathcal{B} \to [0, 1]$. In this paper, we use the popular Łukasiewicz connectives [31], [45], [46], defined as follows:

- $I(\alpha \otimes \beta) = \max(I(\alpha) + I(\beta) - 1, 0)$.
- $I(\alpha \oplus \beta) = \min(I(\alpha) + I(\beta), 1)$.
- $I(\alpha \veebar \beta) = \max(I(\alpha), I(\beta))$.
- $I(\alpha \barwedge \beta) = \min(I(\alpha), I(\beta))$.
- $I(\textbf{not } \alpha) = 1 - I(\alpha)$.
- $I(\beta \to \alpha) = \min(1 - I(\beta) + I(\alpha), 1)$.

In a FASP program, a head expression is an expression of the form $a_1 \oplus a_2 \oplus \ldots \oplus a_n$, where each a_i's is a literal, while a body expression is an expression defined recursively as follows:

- A constant term $\overline{c}$ where $c \in [0,1]$, a positive literal a and a negative literal **not** a are body expressions.
- If a and b are body expressions, then so are $a \oplus b$, $a \otimes b$, $a \veebar b$ and $a \barwedge b$.

A FASP program consists of rules of the form

$$\alpha \leftarrow \beta$$

where α is a head expression and β is a body expression. We sometimes write *Head(r)* and *Body(r)* to denote the head and body expressions of the rule r, respectively. A FASP rule is said to be positive iff it contains no applications of the **not** operator. A FASP program is positive iff it only contains positive rules.

An interpretation I is a model of a rule r iff $I(r) \equiv I(Body(r) \rightarrow Head(r)) = 1$, and I is a model of a program P iff I is a model of every rule $r \in P$. We write $I \leq J$ for two interpretations I and J iff $I(a) \leq J(a)$ for every $a \in \mathcal{B}$. Furthermore, we define $I = J$ as $I \leq J$ and $J \leq I$, while $I < J$ is defined as $I \leq J$ but $I \neq J$. A model I of a positive program P is an answer set of P iff there is no model J of P s.t. $J < I$. For a non-positive program P, a generalization of the so-called Gelfond-Lifschitz reduct is defined in [47] as follows: the reduct of a rule r w.r.t. an interpretation I is the positive rule r^I obtained by replacing each occurrence of **not** a by the constant $\overline{I(\textbf{not } a)}$. The reduct of a FASP program P w.r.t. an interpretation I is then defined as the positive program $P^I = \{r^I \mid r \in P\}$. A model I of P is called an answer set of P iff I is an answer set of P^I.

Following [31], we consider the finite-valued answer sets of a FASP program P, by restricting the values of the interpretation function I to the set $\mathbb{Q}_k = \{0, \frac{1}{k}, \ldots, \frac{k-1}{k}, 1\}$. Any answer set derived by using this restriction is called a k-answer set of the program. Formally, we call an interpretation of a program P a k-interpretation, iff $I(a) \in \mathbb{Q}_k$ for every proposition a. Consequently, a k-interpretation is a k-model of a program P iff it satisfies every rule of P. For a positive program P, a k-model of P is a k-answer set of P iff there is no k-model J of P such that $J < I$. For a non-positive program P, a k-model P is a k-answer set of P iff it is a k-answer set of P^I. If we consider only rational-valued answer sets, then every answer set of a FASP program is necessarily a k-answer set of the program for some finite k. However, the converse is generally not true: a k-answer set of a program may not be an answer set of that program [31], [48].

Example 3. *Consider the FASP program P_1 having the following rules:*

$$open \leftarrow \textbf{\textit{not}}\ close$$
$$closed \leftarrow \textbf{\textit{not}}\ open$$

This program has infinitely many answer sets I_x having $I_x(open) = x$ and $I_x(closed) = 1 - x$, where $x \in [0,1] \cap \mathbb{Q}$. Furthermore, the program has exactly $k+1$ k-answer sets for each positive integer k, where each answer set I_x is of the form $I_x(open) = x$ and $I_x(closed) = 1 - x$, with $x \in \mathbb{Q}_k$.

Example 4. *Consider the FASP program having the single rule $a \oplus a \leftarrow \overline{1}$. One can see that the interpretation $\{(a,1)\}$ is a $1-$answer set. However, it is not an answer set of the program, because it is not minimal. The answer set of this program is $\{(a,0.5)\}$ instead.*

III. Multi-valued Networks

A. Modeling multi-valued networks using FASP

Models of multi-valued biological interaction networks are typically specified through a set of input-output relationships for each node, detailing the values each node takes, given the combinations of the values of the regulators, i.e. nodes that affect it. We formalize this idea, using the concept of a multi-valued network defined as follows.

Definition 2 (Multi-valued network, network state)**.** *A multi-valued network is a tuple $G = \langle X, F, k\rangle$ where $X = \langle x_1, \ldots, x_n\rangle$ is a tuple of multi-valued variables denoting the nodes of the network, $F = \langle f_1, \ldots, f_n\rangle$ is a tuple of update functions, and $k \geq 1$ is a parameter describing the number of activation levels for all the nodes. Specifically, for each node $x \in X$[1], we allow $k+1$ activation levels, i.e., the value for each x is taken from the set $\mathbb{Q}_k = \{0, \frac{1}{k}, \ldots, \frac{k-1}{k}, 1\}$. A network state is then defined as an assignment $V : X \rightarrow \mathbb{Q}_k$. Furthermore, each function $f_i \in F$ satisfies $f_i : \mathbb{Q}_k^n \rightarrow \mathbb{Q}_k$ and is defined using the Łukasiewicz connectives $\otimes, \oplus, \veebar, \barwedge$, and $\neg$, instead of the Boolean connectives.*

From this, we naturally extend the definitions of state transition, update scheme as well as attractor. Note that the definition of the Hamming distance function Δ in (1) can also be applied to the multi-valued network states.

Definition 3 (State transition)**.** *The tuple F of functions defines the state mapping function $f : S \rightarrow S$ as follows: the state $f(v)$ for a state v is the state $w \equiv \langle f_1(v), \ldots, f_n(v)\rangle$. The state transition of a multi-valued network is a relation $\hookrightarrow$: $S \rightarrow S$ whose definition is determined by the type of the update scheme that the network has. The notion of synchronous and asynchronous update scheme in multi-valued networks is defined similarly to the one in Boolean networks.*

In the literature (e.g., [3], [49]), the values each node can take are usually given as integers, ranging from $0, 1, \ldots, k$. Due to the fact that our model is expressed in fuzzy logic, we need to map these values into the [0, 1] range, which can simply be done by mapping each value v to $\frac{v}{k}$. Furthermore, the ranges of possible values often differ from node to node (e.g., the network in [49] has one node with two levels, and one node with three levels). For such cases, we choose k based on the node with the largest range of values, and we map the values of any node having $l < k$ possible values into the values of an l-sized subset of $\mathbb{Q}_k$ (while preserving order), as illustrated in Example 5. This does not affect the behaviour of the modeled system. In fact, in real biological networks encountered in the literature, we mostly see the situation where some nodes have exactly k levels, whereas the rest have only two possible values. In such a case, we can map the values of the two-valued nodes to the set $\{0, 1\}$.

[1]When it is more convenient, we will abuse the notation for X and treat it as a set.

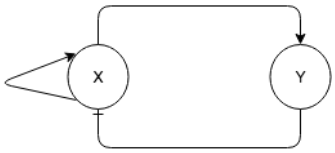

Fig. 4. Diagram for the network of *P. aeruginosa*

.

TABLE I
REGULATORY RELATIONSHIP IN THE *P. aeruginosa* MUCUS DEVELOPMENT NETWORK

No.	x(t)	y(t)	x(t+1)	y(t+1)
1	0	0	$\frac{1}{2}$	0
2	0	1	0	0
3	$\frac{1}{2}$	0	$\frac{1}{2}$	1
4	$\frac{1}{2}$	1	0	1
5	1	0	1	1
6	1	1	1	1

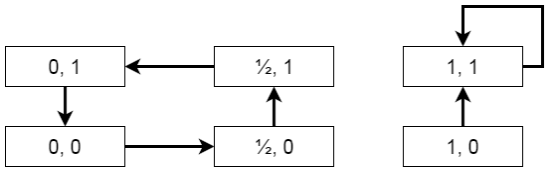

Fig. 5. State transition graph for the network of *P. aeruginosa* using the synchronous update

Example 5. *As a running example, we take the network describing the production of mucus in Pseudomonas aeruginosa described in [49]. There are two nodes in the network, namely x and y, with x having three possible values: 0, 1 or 2, and y having only two values: 0 or 1. Therefore, to model the network in our fuzzy logical representation, we set $k = 2$, and map the values of x into $\{0, \frac{1}{2}, 1\}$, while keeping the values of y as they are. The node x is negatively-regulated by node y and positively by itself, while y is positively-regulated by x. The network structure is shown in Figure 4. The input-output relationships between the two nodes, as given in [49], are shown in Table I. Based on the regulatory relationships between the nodes, the state transition graph of this network is as shown in Figure 5. From the state transition graph, we can clearly see that the network has two attractors: one is a steady state, namely $\langle 1, 1\rangle$, and the other is a cyclic attractor of size 4.*

Below, we extend the idea of using ASP to model the dynamics of biological networks as used in [50] and [14] by allowing a multi-valued activation level in each node. However, instead of using ASP in a *meta-level* approach to describe the dynamics of the network, as in [14], [15], we propose to directly encode the interaction between nodes using FASP rules, which allows for a simpler and more efficient implementation. As shown in [16], a direct encoding of the interaction between nodes in a Boolean network is enough to characterize fixed-size attractors. The same holds for multi-valued networks with FASP under an appropriate many-valued logic semantics.

B. Finding steady states

We first tackle the problem of finding the single state attractors – also called steady states – of a multi-valued network. Recall that the steady states are identical for the synchronous and asynchronous update schemes.

Let $G = \langle X, F, k\rangle$ be a multi-valued network. First, for every node $x \in X$ in the network, we consider two fuzzy propositional atoms p_x and n_x, and write the following FASP rules:

$$p_x \oplus n_x \leftarrow \overline{1}$$
$$\overline{0} \leftarrow p_x \otimes n_x$$

Intuitively, these two rules generate "guesses" for the values of p_x and n_x such that $p_x + n_x = 1$. Define *GS(G)* as the set of all such rules. If x is a node that only takes Boolean values, we can add the following rule (usually called the *saturation rule*):

$$p_x \leftarrow p_x \oplus p_x$$

This rule forces the atom p_x to take only Boolean values in any answer set of the program.

We then encode the interaction between nodes by creating a rule for every node x_i, where the head of the rule is a propositional atom p'_{x_i} associated with the node, while the body corresponds to the direct translation of the fuzzy logic function for the update rule of x_i, replacing the occurrences of the negation symbol $\neg$ with FASP's default negation **not** . Formally, let f_i be the update function of a node x_i of the network. The corresponding FASP *node update rule* of that node, denoted by $NU(f_i)$ is a FASP rule defined as follows:

$$p'_{x_i} \leftarrow BU(f_i)$$

where $BU(f_i)$ is the *body of the node update rule*, which is a FASP expression defined recursively as follows:

- $BU(f_i) = val$ if $f_i \equiv val$ and $val \in [0, 1]$
- $BU(f_i) = p_{x_i}$ if $f_i \equiv x_i$ for a node x_i
- $BU(f_i) = BU(exp_1) \circ BU(exp_2)$ if $f_i \equiv exp_1 \circ exp_2$ for some expressions exp_1, exp_2 and $\circ \in \{\oplus, \otimes, \veebar, \overline{\wedge}\}$
- $BU(f_i) =$ **not** p_x if $f_i \equiv \neg x$

Define $NU(G)$ as the set of rules created in this step, i.e., $NU(G) = \{NU(f_i) \mid 1 \leq i \leq n\}$. Intuitively, the atom p'_{x_i} holds the activation value of the node x_i after the update function has been applied. To drive the FASP program to find a steady-state, we enforce the condition that the activation level of each node is the same after the update. This can be done by using the following rules $CS(i)$ for each node x_i

$$\overline{0} \leftarrow p_{x_i} \otimes \textbf{not } p'_{x_i}$$
$$\overline{0} \leftarrow p'_{x_i} \otimes \textbf{not } p_{x_i}$$

Define $CS(G)$ as the set of all *constraining* rules, i.e. $CS(G) = \{CS(i) \,|\, 1 \leq i \leq n\}$. The example below illustrates the construction process of the FASP program $P(G) = GU(G) \cup NU(G) \cup CS(G)$ for the multi-valued network introduced in Example 5.

Example 6. *Consider the network of P. aeruginosa given in Example 5. Since the network consists of two nodes, x and y, the initial guessing rules for the nodes' values can be written as*

$$\begin{aligned} x \oplus n\_x &\leftarrow \overline{1} \\ \overline{0} &\leftarrow x \otimes n\_x \\ x \oplus n\_y &\leftarrow \overline{1} \\ \overline{0} &\leftarrow y \otimes n\_y \end{aligned}$$

Since we need y to be Boolean, we add the following rule:

$$y \leftarrow y \oplus y$$

The regulatory relationships between the nodes x and y in the network (as given by Table I) can be captured by the following update functions expressed in Łukasiewicz formulas:

$$\begin{aligned} f_1(x,y) &= ((x \veebar \tfrac{1}{2}) \otimes \neg y) \oplus z \\ z &= (x \otimes \tfrac{1}{2}) \oplus (x \otimes \tfrac{1}{2}) \\ f_2(x,y) &= x \oplus x \end{aligned}$$

where z is an auxiliary variable.[2] *We thus construct the following FASP rules to represent the update on each node.*

$$\begin{aligned} x' &\leftarrow ((x \veebar \tfrac{1}{2}) \otimes \textbf{\textit{not}}\ y) \oplus z \\ z &\leftarrow (x \otimes \tfrac{1}{2}) \oplus (x \otimes \tfrac{1}{2}) \\ y' &\leftarrow x \oplus x \end{aligned}$$

Finally, we add the following constraints to find only steady-states:

$$\begin{aligned} \overline{0} &\leftarrow x' \otimes \textbf{\textit{not}}\ x \\ \overline{0} &\leftarrow x \otimes \textbf{\textit{not}}\ x' \\ \overline{0} &\leftarrow y' \otimes \textbf{\textit{not}}\ y \\ \overline{0} &\leftarrow y \otimes \textbf{\textit{not}}\ y' \end{aligned}$$

It can be verified that the resulting program has exactly one 2-answer set, which contains $\{(x,1),(y,1)\}$, corresponding to the only steady state $\langle 1,1\rangle$ of the network.

The previous example also illustrates the fact that we need to translate the regulatory relationships between multi-valued activation levels into FASP rules. In practice, it may not always be easy to perform this translation manually. As we will explain in Section V, in practice this step can be performed automatically using the tool we wrote.

Next we show that the correspondence between steady states of the multi-valued network G and k-answer sets of the FASP program $P(G) = GU(G) \cup NU(G) \cup CS(G)$ holds in general.

Proposition 1. *The program $P(G) = GU(G) \cup NU(G) \cup CS(G)$ captures all the steady states of the multi-valued network G, i.e., for every k-answer set I of $P(G)$, the state S s.t. $S(x) = I(p_x)$ for every $x \in X$ is a steady state of G, and for every steady state S of G, there is a corresponding k-answer set I of G s.t. $S(x) = I(p_x)$ for every $x \in X$.*

[2]The variable z is an auxiliary variable only intended to allow us to present a more concise expression here.

Proof. First, it can be easily seen that in any answer set I of $P(G)$, we have that $I(p'_x) = I(p_x)$, due the rules in $CS(G)$. Suppose that S is a steady-state of the multi-valued network G. By definition, we have that $f_i(X) = S(x_i)$ for every $x_i \in X$. We will show that the interpretation I s.t. $I(p_x) = S(x)$ and $I(n_x) = 1 - S(x)$ for every $x \in X$ is a k-answer set of the program $P(G)$. First, by the definition of $GU(G)$, it is clear that I is a model of $GU(G)$. For every rule r in $NU(G)$ corresponding to the update function f_i, from the fact that $I(p_x) = I(p'_x) = S(x)$ for every $x \in X$, it can be shown that the recursive definition of $BU(f_i)$ entails that $I(Body(r)) = f_i(X)$. Since we have $f_i(X) = S(x_i)$, we also have that $I(Body(r)) = S(x_i)$. This means that $I(Head(r)) = I(p'_{x_i}) = S(x_i) = I(Body(r))$, which means that I is also a model of the rule r. Consequently, I is a model of $NU(G)$, and thus also of $P(G) = GU(G) \cup NU(G)$. It is easy to see that I is a minimal k-model of $GU(G)$, since any k-model $J < I$ will violate at least one rule in $GU(G)$.

Conversely, if we have a k-answer set I of $P(G)$, we can show that the state S s.t. $S(x) = I(p_x)$ for every $x \in X$ is a steady state of the network. It is sufficient to show that $f_i(X) = S(x_i) = I(p_x) = I(p'_x)$ for every $x_i \in X$. Since I is a model of the rule $NU(f_i)$, we have that $I(p'_{x_i}) \geq I(Body(NU(f_i))) = I(BU(f_i))$. From the definition of $BU(f_i)$ it can be shown that $I(BU(f_i)) = f_i(X)$. Hence we have that $I(p'_{x_i}) \geq f_i(X)$. Suppose that $I(p'_{x_i}) > f_i(X)$, for some $x_i \in X$. Consider the k-interpretation J such that $J(p'_a) = I(p'_a)$ for every $a \in X$ s.t. $a \neq x_i$, and $J(p'_{x_i}) = f_i(X)$. We have that $J < I$, and it can be seen that J is also a k-model of $P(G)$ (since it satisfies all the rules in $P(G)$), contradicting the minimality of I. Hence, we must have that $I(p'_{x_i}) = f_i(X)$ for every $x_i \in X$. □

C. Finding fixed-size cyclic attractors

It is clear that the approach from Section IV.B is not suitable for finding cyclic attractors, since the proposed encoding does not represent different values of each node at different update times. Recall that we can have either the synchronous or the asynchronous update schemes for our networks, and that using different update schemes on the same network can result in different sets of attractors. We need to explicitly take into account the time dimension to distinguish between different update schemes, and thus compute the appropriate sets of attractors.

Taking into account the time dimension can be achieved by adding a parameter t, representing time, to each of the fuzzy propositional atoms p_x and n_x. This time parameter can be limited up to a certain maximum value, say s, if we are interested in only finding cyclic attractors of size up to s. This can be done simply by adding facts that assert the truth of a predicate called $time(t)$ for $t = 0, 1, \ldots, s$.

The initial guessing rules $GU_0(G)$ are now written as

$$\begin{aligned} p_x(0) \oplus n_x(0) &\leftarrow \overline{1} \\ \overline{0} &\leftarrow p_x(0) \otimes n_x(0) \end{aligned}$$

where the parameter 0 encodes the fact that we are guessing at the initial time point $t = 0$. We then define a new encoding of

the node update rule that incorporates a time parameter t. In order to do this, we first introduce the so called *time-dependent body of a node update rule*, defined as follows:

- $TBU(f_i, t) = val$ if $f_i(x_i) \equiv val$ and $val \in [0, 1]$
- $TBU(f_i, t) = p_x(t)$ if $f_i(x_i) \equiv x$ for a node x
- $TBU(f_i, t) = TBU(exp_1, t) \circ TBU(exp_2, t)$ if $f_i(x_i) \equiv exp_1 \circ exp_2$ for some expressions exp_1, exp_2 and $\circ \in \{\oplus, \otimes, \veebar, \barwedge\}$
- $TBU(f_i, t) = \textbf{not}\ p_x(t)$ if $f_i(x_i) \equiv \neg x$

We then define the *time-dependent node update rules TNU*, that perform the update to the values in each node, as follows:

$$p_{x_i}(t+1) \leftarrow time(t) \otimes TBU(f_i, t)$$

For the synchronous case, this is enough to encode the fact that at each time step t, each node's value is updated using the update function defined for the node.

For the asynchronous update scheme, recall that even though a state can have multiple successor states (due to the non-deterministic choice of which node is updated), only states that have a single possible successor can be part of an attractor. Thus, at any time step, we need to ensure that there is only one possible successor state of the current state. This can be done by checking that there is exactly one node that gets a new value during the updates, since if no nodes get a new value, then the state would be a steady state, while if more than one node gets updated, then there will be multiple successors to the current state.

This can be done by first adding the following set of rules for each node $x \in X$:

$$\begin{aligned} d_x &\leftarrow p_x(t+1) \otimes \textbf{not}\ p_x(t) \\ d_x &\leftarrow p_x(t) \otimes \textbf{not}\ p_x(t+1) \\ d_x &\leftarrow d_x \oplus d_x \end{aligned}$$

which intuitively derives the atom d_x if the node x gets a new value during the update. We then add a constraint

$$\overline{0} \leftarrow d_x \otimes d_y$$

for every pair of nodes x and y. This forces that there is at most one node having a new value during the update. Finally, using

$$\begin{aligned} at\_least\_one &\leftarrow (d_{x_i} \veebar \ldots \veebar d_{x_n}) \\ \overline{0} &\leftarrow \textbf{not}\ at\_least\_one \end{aligned}$$

ensures that there is exactly one node that receives a new value during the update.

We can now define the required condition to find cyclic attractors, independent of the update scheme. The following set of rules and constraints can be used to find cyclic attractors up to size s. First, add the following rules for all k, $1 \leq k \leq s$ and all $x_i \in X$:

$$\begin{aligned} a_k &\leftarrow p_{x_i}(0) \otimes \textbf{not}\ p_{x_i}(k) \\ a_k &\leftarrow p_{x_i}(k) \otimes \textbf{not}\ p_{x_i}(0) \end{aligned}$$

These rules ensure that a_k is false iff the value of $p_x(0)$ equals to $p_x(k)$ for all $x \in X$, which means that there is a cyclic attractor of size k (or of size an integer divisor of k). Then add the following constraint:

$$\overline{0} \leftarrow a_1 \barwedge a_2 \ldots \barwedge a_s$$

which forces at least one of the a_k's to be false, say a_l, which means that there is a cyclic attractor of size l (or a divisor of l). The example below illustrates the FASP program construction process for the network from Example 5.

Example 7. *Consider again the network in Example 5, and consider the task of finding the cyclic attractors of size* 4 *under the synchronous update. Denote this network as G. The initial guessing rules $GU_0(G)$ are:*

$$\begin{aligned} x(0) \oplus n\_x(0) &\leftarrow \overline{1} \\ y(0) \oplus n\_y(0) &\leftarrow \overline{1} \\ \overline{0} &\leftarrow x(0) \otimes n\_x(0) \\ \overline{0} &\leftarrow y(0) \otimes n\_y(0) \end{aligned}$$

Furthermore, since we need to allow node y to be 0 or 1 only, we add a constraint:

$$y(T) \leftarrow y(T) \oplus y(T)$$

The node updates TNU(G) can be represented using the following rules

$$\begin{aligned} x(T+1) &\leftarrow time(T) \otimes ((x(T) \veebar \tfrac{1}{2}) \otimes \textbf{\textit{not}}\ y(T)) \oplus z(T)) \\ z(T) &\leftarrow (x(T) \otimes \tfrac{1}{2}) \oplus (x(T) \otimes \tfrac{1}{2}) \\ y(T+1) &\leftarrow time(T) \otimes (x(T) \oplus x(T)) \end{aligned}$$

To find synchronous cyclic attractors up to size 4, we add the following for all $i = 1, \ldots, 4$:

$$\begin{aligned} a_i &\leftarrow x(0) \otimes \textbf{\textit{not}}\ x(i) \\ a_i &\leftarrow x(i) \otimes \textbf{\textit{not}}\ x(0) \\ a_i &\leftarrow y(0) \otimes \textbf{\textit{not}}\ y(i) \\ a_i &\leftarrow y(i) \otimes \textbf{\textit{not}}\ y(0) \\ \overline{0} &\leftarrow a_1 \barwedge a_2 \barwedge a_3 \barwedge a_4 \end{aligned}$$

One can check that the resulting program has exactly five 2-answer sets. One of these answer sets encodes the static transitions of the steady-state $\langle 1, 1 \rangle$, by having the same values for $x(0), \ldots x(4)$ and $y(0), \ldots y(4)$. The other four answer sets encode the cyclic attractor $\langle 0, 0 \rangle \hookrightarrow \langle \frac{1}{2}, 0 \rangle \hookrightarrow \langle \frac{1}{2}, 1 \rangle \hookrightarrow \langle 0, 1 \rangle \hookrightarrow \langle 0, 0 \rangle$, with each answer set encoding the different initial conditions.

Recall that the example network does not have any cyclic attractor of size > 1 (as explained in Figure 3) for the asyn-

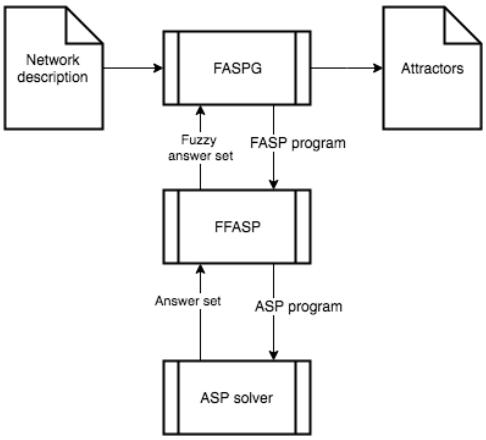

Fig. 6. FASPG work flow

chronous update. In this case, we need to add the following rules and constraints:

$$\begin{aligned} d_x &\leftarrow p_x(t+1) \otimes \textbf{\textit{not}}\ p_x(t) \\ d_x &\leftarrow p_x(t) \otimes \textbf{\textit{not}}\ p_x(t+1) \\ d_x &\leftarrow d_x \oplus d_x \\ d_y &\leftarrow p_y(t+1) \otimes \textbf{\textit{not}}\ p_y(t) \\ d_y &\leftarrow p_y(t) \otimes \textbf{\textit{not}}\ p_y(t+1) \\ d_y &\leftarrow d_y \oplus d_y \\ \overline{0} &\leftarrow d_x \otimes d_y \\ at\_least\_one &\leftarrow d_x \veebar d_y \\ \overline{0} &\leftarrow \textbf{\textit{not}}\ at\_least\_one \end{aligned}$$

We can see that the states $\langle 0,0\rangle$, $\langle 0,1\rangle$, and $\langle 1,0\rangle$ will be eliminated from the search immediately, since they have multiple successor states, as shown in Figure 3.

IV. Automatic encoding of network descriptions

Biological networks with multiple activation levels are often specified in terms of the regulatory relationships between their nodes (e.g., in [3], [49]). Such relationships are basically a set of input-output specification for every node, consisting of every possible combination of values of every node regulating it. To generate the required FASP program for computing the attractors, we need to represent these relationships in the form of fuzzy logic formulas under Łukasiewicz semantics, such as the ones given in Example 6. It is not always straightforward for a human expert to find a suitable formula that fits a certain input-output relationship specification. We therefore provide a tool, called FASPG[3], that performs this task automatically, and then invokes a FASP solver to compute the attractors of the GRN. Figure 6 shows the work flow of FASPG.

The input for FASPG is the description of a network, consisting of:

- The number of nodes, n
- The number of activation levels each node has, k
- An input-output specification for every node (described below)

An input-output specification of a node is a set of assignments for that node, given all possible combinations of the nodes regulating it. For example, consider a node x regulated by m nodes, $y_1, \ldots, y_m$. Then, the input-output specification for x is a table of k^m rows, each row consisting of a possible combination of the values of the y_i's and a corresponding value for x.

Given such an input-output specification for a node, FASPG automatically constructs a correct set of Łukasiewicz logic formulas that evaluates to the required value for the node, following the construction process outlined in Proposition 2.

Proposition 2. *Suppose we are given that x has value v whenever each y_i has the value v_i, $i = 1, \ldots, m$. Consider the program $F(x, v)$ consisting, for each i, of the following rules:*

$$\begin{aligned} p_i &\leftarrow y_i \otimes \overline{1 - v_i} \\ p_i &\leftarrow p_i \oplus p_i \\ q_i &\leftarrow \textbf{\textit{not}}\ y_i \otimes \overline{v_i} \\ q_i &\leftarrow q_i \oplus q_i \\ c_i &\leftarrow \textbf{\textit{not}}\ p_i \otimes \textbf{\textit{not}}\ q_i \end{aligned}$$

and the single rule

$$x \leftarrow c_1 \otimes \ldots \otimes c_m \otimes \overline{v}$$

It holds that in any answer set I of $F(x, v)$, $I(x) = v$ whenever $I(y_i) = v_i$ for every $i = 1, \ldots, m$.

Proof. Intuitively, the atoms p_i and q_i are Boolean atoms signifying the condition of whether the value of y_i is $> v_i$ and $< v_i$, respectively. Therefore, the atom c_i, which is only true when both p_i and q_i are false, encodes the condition when the value of y_i is exactly v_i. The last rule of $F(x, v)$ then assigns the value of v to x, given that all c_i's are true. □

Such an encoding is applied to every row in the input-output relationship table, and then used in the program encoding for the computation of the attractor. Note that this encoding is not the only possible one we can come up with, nor is it necessarily the most efficient one, but as the experiments below will show, it is efficient enough for real-world networks.

After obtaining the encoding for the regulatory relationships, FASPG writes the remaining program encoding for the appropriate problem, and then submits it to the FASP solver FFASP[4] [31], [32], which in turn performs the translation to ASP and calls the ASP solver CLINGO [19]. The attractors are then deduced from the resulting answer sets by FASPG.

V. Benchmark and Experiments

In the literature, little work has been done so far on computing attractors of multi-valued networks obtained from biological knowledge, due to the lack of appropriate tools to perform analysis on multi-valued networks. Our work is aimed to address this issue. In order to show the applicability of our approach, we collected several multi-valued networks obtained from the known biological networks in the literature. We

[3]FASPG is available at http://github.com/mushthofa/faspg

[4]FFASP is available at http://github.com/mushthofa/ffasp

run our approach on these networks and verify the expected results. Furthermore, to test the scalability of our approach, we also applied it to randomly generated synthetic networks and measure the time and memory requirements. All experiments were run on a machine with an 2.5GHz Intel Xeon CPU and a maximum of 15 GB of allowed memory consumption.

A. Experiments on real networks

To evaluate the correctness and efficiency of our method, we have tested it on a number of biological network models obtained from the literature. Table II represents the summary of the data collected. In each of these networks, each node is either Boolean-valued, or three-valued (represented as either the values 0, 1 and 2 or 'low', 'medium' and 'high' in the papers originally describing these multi-valued networks), except for the *D. melanogaster* segmentation network which uses a four-valued logical model. In encoding the regulatory relationships between the nodes in the network, we assign values from $\mathbb{Q}_k$ to any k-valued nodes. Consequently, in these network models, we only consider attractors reached from the set of states where the Boolean-valued nodes are assigned either 0 or 1, and 3-valued and 4-valued nodes are assigned values from $\mathbb{Q}_3$ and $\mathbb{Q}_4$, respectively. To generate all the possible relevant states, we add a saturation rule as described in Section IV to each Boolean node x. For each of these models, the steady-states are computed, and compared to the ones reported in their respective reference(s).

For the *A. thaliana* flowering network, the network update functions are listed in [3] as name-values pairs indicating the input-output pairs of the update function on each node. For the Th cell regulatory network, [52] proposed different versions of the network. For our purpose, we use the logical rules presented in the Equation 2 in that paper, and evaluate them as 3-valued Łukasiewicz functions (i.e., treating $\vee$ and $\wedge$ as $\oplus$ and $\otimes$, respectively), which is equivalent to the 1-hot encoding used in [2]. For the *D. melanogaster* segmentation network, the network update functions are represented using the notation used in [43]. By ignoring the time-delay parameter of this representation and using the assumption of the basal-expression levels of the genes to be 0 (as also done in [25]), we can faithfully represent each of the update functions given using Łukasiewicz logic formulas.

Table II shows, for each network, the number of nodes (n) , the number of Boolean nodes, the number of possible activation levels (k), the number of steady-states found, and the computation time using our method. We can see that for the largest network ($n = 23$), the computation time is still very manageable (< 5 seconds). Except for the *A. thaliana* flowering network, we have taken advantage of the fact that the source literature already represented the update function as a logical function that can be directly translated into Łukasiewicz logic formulas. This might not always be the case, as shown in the *A. thaliana* network, where the interaction network was given just in the form of input-output pairs between the regulating nodes and the regulated node. In such cases, FASPG relies on the construction process from Proposition 2 to automatically generate the update function. These automatically-generated formulas, despite being correct, might cause the computations to take more time compared to manually crafted ones. The following subsection details an experiment on applying our method to synthetic networks to gain a more realistic picture of the computational requirements when we use FASPG to assist in the encoding of the interaction network.

B. Experiments on synthetic networks

Due to the limited availability of results about them in the literature, experiments on real biological networks can only paint a small picture on the efficiency of the application of the proposed method. Furthermore, the benchmark test on real networks that we presented in the previous subsection was limited to only the computation of steady states, due to the non-availability of cyclic attractor data for any of the networks. Additionally, we would like to see the effects of using FASPG's automatic encoding of the interaction network. Below we therefore apply the method on randomly generated networks. These additional experiments are intended to asses the computational resources (in terms of time and space) needed to run the method, given increasing values of n and k. To this end, we generated 5 random networks for each combination of n and k, ranging from $n = 5$ to $n = 50$ with a step of 5, and $k = 1$ to $k = 6$. To generate realistic network topologies, we follow the procedure for generating random scale-free networks as given in [53]. Briefly, during the random network generation, each node is added one by one. At each step, the probability that an existing node is connected to a new node is proportional to its current degree. The directionality of the interactions are then chosen randomly. Furthermore, to limit the computational burden, we restrict the number of incoming regulatory interactions for a node to be within the range of 1 to 5. In each of these regulatory relationships, a set of random input-output relationships are generated (which covers every possible combination of values for the regulators).

For each of these random networks, we solve the following tasks using FASPG:

- Find all steady states of the network.
- Find at least one cyclic attractor with size < 5 using either synchronous or asynchronous updates (or report that there are none).

In each of our runs, we record the running time and the maximum memory usage. We set a time-out of 20 minutes per computation. For every combination of n and k, we run the method on 5 different randomly-generated networks, and we report the average of the running times and memory usages on the 5 networks, unless we observe a time-out or a memory-out in any of the 5 networks, in which case we report it as a failure.

Figure 7 and Figure 8 show the computation time and memory usage of the algorithm in finding all steady states, respectively. Overall, we notice that the method performs quite well in computing steady states for lower values of k, with the largest instance ($n = 50$) requiring less than 5 minutes, on average, to complete. However, we can clearly see that the

TABLE II
BENCHMARK RESULTS.

No.	network (and references)	# of nodes	# of Boolean nodes	k	# of steady-states	time (seconds)
1	*P. aeruginosa* mucus development network [49], [51]	2	1	3	2	0.03
2	*A. thaliana* flowering network [3]	15	7	3	10	2.87
3	Th cell regulatory network [2], [52]	23	9	3	4	4.3
4	*D. melanogaster* segmentation network [25]	7	4	4	4	4.2

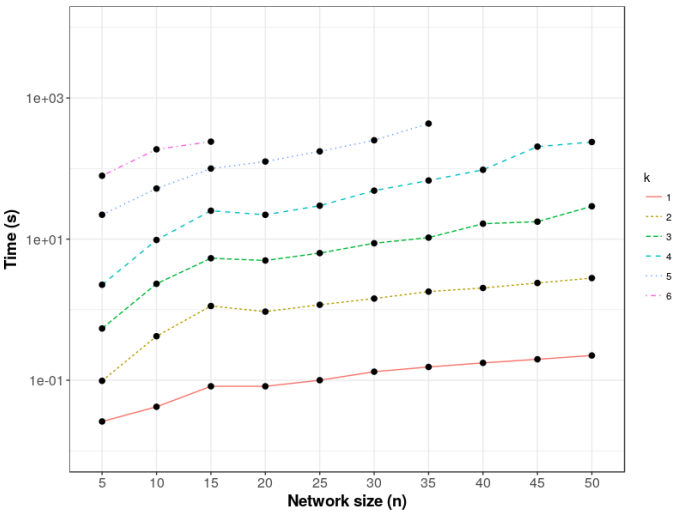

Fig. 7. Running time for computing steady states. Missing nodes indicate failure due to time-outs/memory-outs.

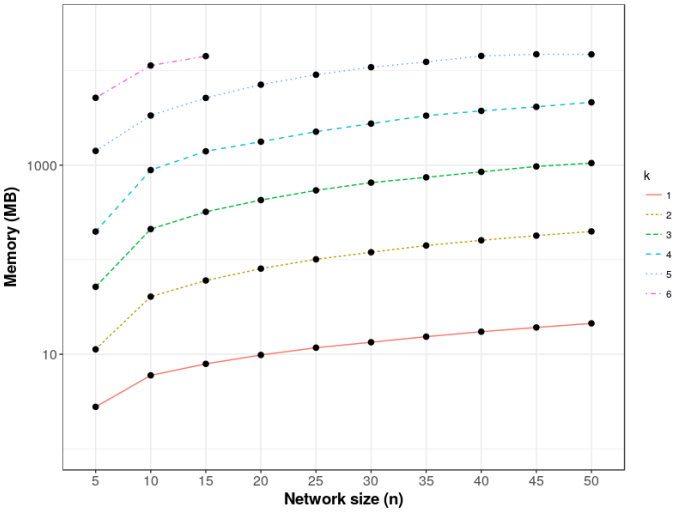

Fig. 8. Memory usage for computing steady states. Missing nodes indicate failure due to time-outs/memory-outs.

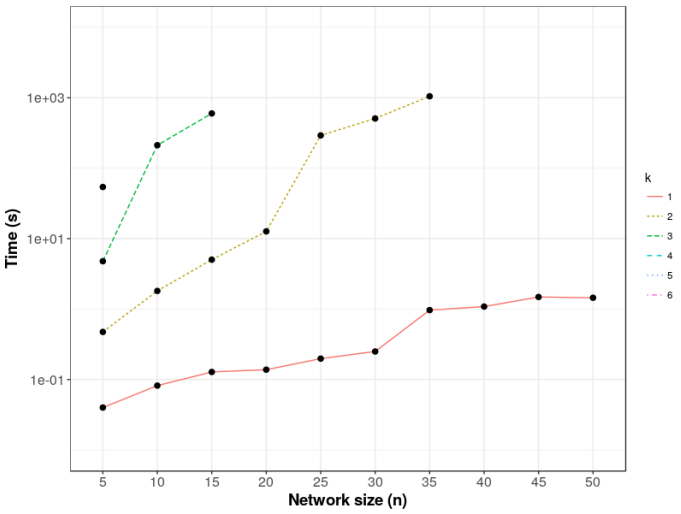

Fig. 9. Running time for computing synchronous cyclic attractors. Missing nodes indicate failure due to time-outs/memory-outs. The singleton node represents a value for $k = 4$. All instances with $k > 4$ failed due to time-outs/memory-outs, and are thus not shown.

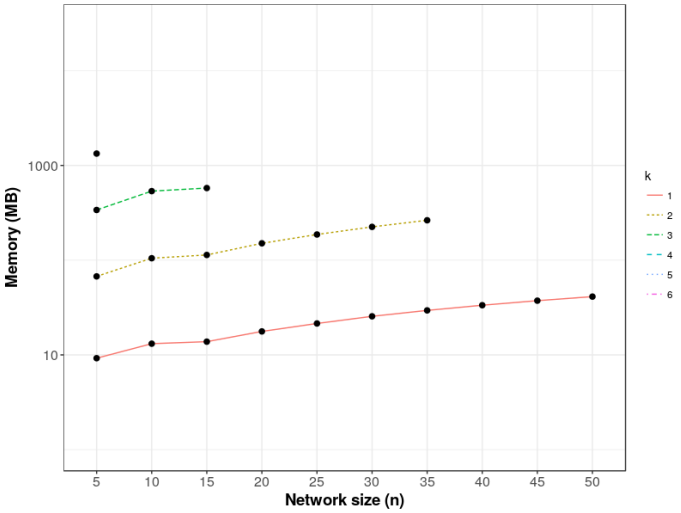

Fig. 10. Memory usage for computing synchronous cyclic attractors. Missing nodes indicate failure due to time-outs/memory-outs.The singleton node represents a value for $k = 4$. All instances with $k > 4$ failed due to time-outs/memory-outs, and are thus not shown.

bottleneck is in k, and for $k \geq 5$, computation time as well as memory usage increase drastically with larger values of n.

Figure 9 and Figure 10 show the computation time and memory usage for finding cyclic attractors using synchronous updates. Overall, we see that finding cyclic attractors generally takes more time and memory than finding steady states. The overall trend that k seems to be the bottleneck can still be observed, with even more time-outs. For $k = 1$, no time-outs are observed for the network sizes considered. For larger k, we start to observe more and more time-outs, with $k = 4$ having time-outs for $n > 5$.

Figure 11 and Figure 12 show the computation time and memory usage for finding cyclic attractors using asynchronous updates. Here, we notice that the time requirement for finding asynchronous cyclic attractors is, in general, lower than in

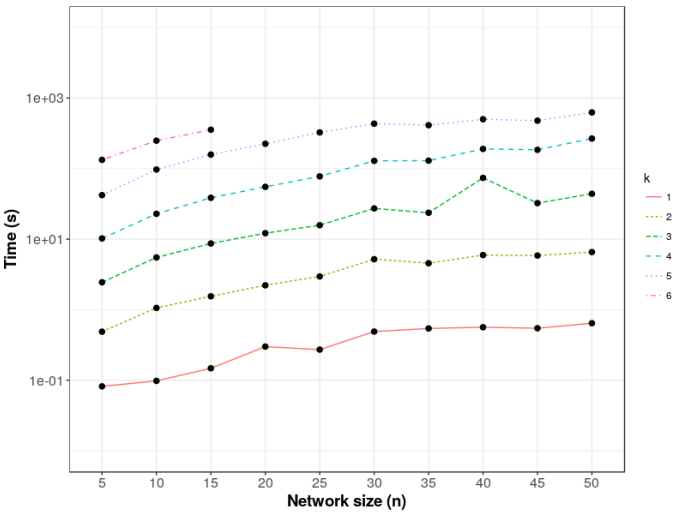

Fig. 11. Running time for computing asynchronous cyclic attractors. Missing nodes indicate failure due to time-outs/memory-outs.

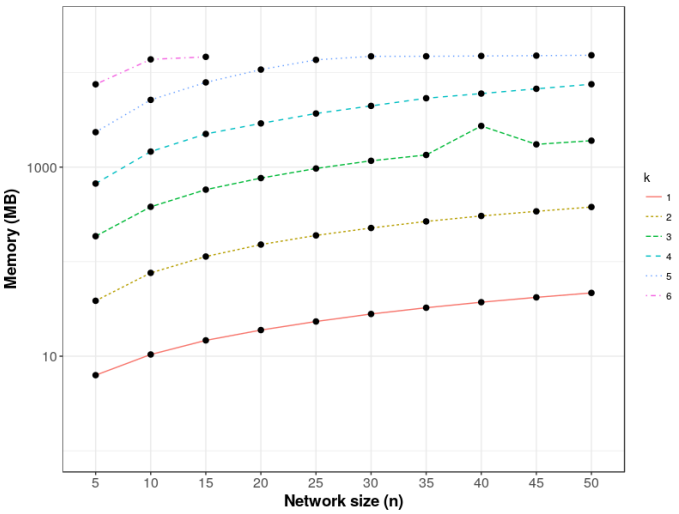

Fig. 12. Memory usage for computing asynchronous cyclic attractors. Missing nodes indicate failure due to time-outs/memory-outs.

the synchronous case. This is probably due to the more stringent criteria applied to the dynamics (in which only one possible successor state is allowed) which can be exploited by the solver. No time-outs are observed for the network sizes considered. However, we see a larger memory-usage than for either the steady-states and synchronous cyclic attractors, with larger instances having memory outs.

In conclusion, we observe, as expected, that time and memory requirements generally increase exponentially w.r.t the size of the network (n), while the number of possible values in the activation level of the genes (k) serves as an exponential factor. In addition, we observe that computing steady states generally has a lower computational requirements than computing synchronous and asynchronous attractors. Synchronous attractor computation generally requires more computation time than the other two, while memory consumption is generally the biggest bottleneck in asynchronous attractor computation.

VI. Related Work

Since they were introduced by Kauffman [4], Boolean networks have gained considerable popularity as a simple but powerful modeling technique in systems biology. Boolean networks have been used to describe the dynamics of regulatory networks in cases where we have reasonably good knowledge about the regulatory relationship between the genes, and where the activation levels of genes can be simply represented as "on" and "off". In such cases, the dynamics of the network, and especially the attractors, usually correspond to some biologically relevant phenotype, e.g. a cell type. For instance, in [54], [55] and more recently, [3] and [56], the flower development in *Arabidopsis thaliana* was modeled using a Boolean network, in which the network attractors corresponded to stable gene expression levels during the different stages of flower development. In [57], Li et al. used a Boolean network model and its steady states to describe the different stages of the yeast cell cycle, where the stages of the cycle correspond to the strong attractors of the network. Kaufman et al. [5] explained the various states of the immune system with Boolean network models. Similarly, the regulatory networks involved in the various parts of the development of *Drosophila melanogaster* were studied using Boolean networks in [58], [59] and [60].

Although Boolean networks provide a useful simplification to study the dynamics of gene regulatory networks, using only two values to represent the activation may cause one to miss important characterizations of GRNs that have attractors containing "intermediate" levels of expressions of the genes (see e.g., [3], [23]–[25], [43]). In [24], an extension of Boolean networks into multi-valued networks in which each node is allowed to have k levels of activation (where $k \geq 2$) is considered. Using the so-called 1-hot encoding, these multi-valued networks are reduced into a representation which allows techniques already used in Boolean networks, such as Binary Decision Diagrams (BDD), to be applied. However, the use of an encoding scheme such as 1-hot encoding can make the representation quite cumbersome, especially for large values of k, since it requires us to explicitly encode the logical operators for all combinations of truth values. As we will show, the use of FASP can overcome this problem by using fuzzy logic connectives.

Several computational tools have been developed to compute attractors in Boolean network models. In [7], Garg et al. developed **genYsis**, which uses techniques involving BDDs to compute attractors. Ay et al. [10] used state-space pruning and randomized state-space traversal methods to improve the scalability of the attractor computation. Dubrova et al. [11] used a Boolean Satisfiability (SAT) solver, which was shown to be more efficient, both in terms of computation time and space requirements, compared to the BDD-based approach. Zheng et al. [12] developed **geneFatt** based on the Reduced Order BDD (ROBDD) data structure, which further improves the efficiency of the attractor computation. Berntenis et al. [9] considered the enumeration of attractors of larger networks by restricting the enumeration of possible states to only the relevant subsets. More recently, [14] used Answer Set Programming (ASP) to model the computation of attractors in a Boolean network.

However, these methods were designed to compute the dynamics of Boolean networks, i.e., where the nodes can take only two possible values. In this paper, we extend the work in [14], by using Fuzzy Answer Set Programming (FASP) to allow the computation of the dynamics of multi-valued networks.

ASP has been successfully applied to model the dynamics of gene regulatory networks in the Boolean setting; see e.g. [14], [15]. In these works, the encoding of the update function is restricted to two specific types (denoted as r^* and r^+ in [14]), due to the particular way that the encoding of the dynamics is written (i.e., encoding the update function at a *meta-level*). In [16], it was suggested that each of the node's update functions of a Boolean network can be directly encoded as a rule in ASP. This allows for a more generic encoding of the network update function. Furthermore, it was shown that the steady states of the network are directly obtainable using the semantics of ASP. To obtain the cyclic attractors, [16] proposes an extension of the ASP semantics which allows to capture cyclic attractors "naturally" as answer sets of the program. Such an extension is not obvious nor easy to develop and implement, however, since it requires the redefinition of the basics of ASP, as well as the reimplementation of currently available solvers. In addition, this method is only geared towards Boolean networks, instead of multi-valued networks.

In this paper, we proposed a new method to encode the dynamics of multi-valued networks using FASP which incorporates two distinguishing characteristics:

- It allows graded activation levels in the nodes of the networks instead of only "on" and "off", and
- It allows a more flexible definition of the network update function by encoding the dynamics of the network using a *time* argument. In contrast to the approach used in [14], [50], the use of the time argument and the direct encoding of the network update function allows for a more general relationships between interacting nodes. Additionally, this alleviates the requirement to extend/redefine the theoretical notion of answer sets in logic programming, as is required by the approach used in [16] for encoding the computation of cyclic attractors.

VII. Conclusion

Boolean networks have traditionally been used as one of the most popular methods for modeling and analyzing the dynamics of GRNs. Using Boolean networks, we can capture the steady states/attractors of the network, which is useful to understand the biological function of such networks. Many tools, including the ones based on ASP, have been devised to model such dynamics. However, few developments have looked at characterizing attractors based on degrees of activation. In this paper, we have suggested the use of FASP, an extension of ASP in the continuous domain, as a convenient language for encoding the dynamics of multi-valued networks. To the best of our knowledge, this is the first real-world application of FASP that goes beyond small toy examples. We showed the correctness of our encoding, and we evaluated its efficiency for computing the steady-states of real biological networks found in the literature. The experimental result shows that the proposed method works quite efficiently, especially for finding the most biologically-relevant type of attractors, which are the steady-states and small-sized attractors.

References

[1] S. A. Kauffman, *The origins of order: Self-organization and selection in evolution*. Oxford university press, 1993.

[2] L. Mendoza, "A network model for the control of the differentiation process in Th cells," *Biosystems*, vol. 84, no. 2, pp. 101–114, 2006.

[3] C. Espinosa-Soto, P. Padilla-Longoria, and E. R. Alvarez-Buylla, "A gene regulatory network model for cell-fate determination during arabidopsis thaliana flower development that is robust and recovers experimental gene expression profiles," *The Plant Cell Online*, vol. 16, no. 11, pp. 2923–2939, 2004.

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

[5] M. Kaufman, J. Urbain, and R. Thomas, "Towards a logical analysis of the immune response," *Journal of Theoretical Biology*, vol. 114, no. 4, pp. 527–561, 1985.

[6] H. De Jong, "Modeling and simulation of genetic regulatory systems: a literature review," *Journal of computational biology*, vol. 9, no. 1, pp. 67–103, 2002.

[7] A. Garg, I. Xenarios, L. Mendoza, and G. DeMicheli, "An efficient method for dynamic analysis of gene regulatory networks and in silico gene perturbation experiments," in *Research in Computational Molecular Biology*. Springer, 2007, pp. 62–76.

[8] G. Arellano, J. Argil, E. Azpeitia, M. Benitez, M. Carrillo, P. Gongora, D. Rosenblueth, and E. Alvarez-Buylla, ""antelope": a hybrid-logic model checker for branching-time boolean grn analysis," *BMC Bioinformatics*, vol. 12, no. 1, p. 490, 2011.

[9] N. Berntenis and M. Ebeling, "Detection of attractors of large boolean networks via exhaustive enumeration of appropriate subspaces of the state space," *BMC Bioinformatics*, vol. 14, no. 1, pp. 1–10, 2013.

[10] F. Ay, F. Xu, and T. Kahveci, "Scalable steady state analysis of boolean biological regulatory networks," *PLoS ONE*, vol. 4, no. 12, p. e7992, 12 2009.

[11] E. Dubrova and M. Teslenko, "A SAT-based algorithm for finding attractors in synchronous boolean networks," *IEEE/ACM Trans. on Computational Biology and Bioinformatics*, vol. 8, no. 5, pp. 1393–1399, 2011.

[12] D. Zheng, G. Yang, X. Li, Z. Wang, F. Liu, and L. He, "An efficient algorithm for computing attractors of synchronous and asynchronous boolean networks," *PloS ONE*, vol. 8, no. 4, p. e60593, 2013.

[13] S. Dworschak, S. Grell, V. J. Nikiforova, T. Schaub, and J. Selbig, "Modeling biological networks by action languages via answer set programming," *Constraints*, vol. 13, no. 1-2, pp. 21–65, 2008.

[14] M. Mushthofa, G. Torres, Y. Van de Peer, K. Marchal, and M. De Cock, "ASP-G: an ASP-based method for finding attractors in genetic regulatory networks," *Bioinformatics*, vol. 30, no. 21, p. 3086, 2014.

[15] T. Fayruzov, M. De Cock, C. Cornelis, and D. Vermeir, "Modeling protein interaction networks with answer set programming," in *Proceedings of the IEEE International Conference on Bioinformatics and Biomedicine, 2009*, 2009, pp. 99–104.

[16] K. Inoue, "Logic programming for boolean networks," in *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence (IJCAI 2011)*, 2011, pp. 924–930.

[17] V. Lifschitz, "What is answer set programming?." in *Proceedings of the 23rd AAAI Conference in Artificial Intelligence*, vol. 8, 2008, pp. 1594–1597.

[18] C. Baral, *Knowledge representation, reasoning and declarative problem solving*. Cambridge University Press, 2003.

[19] M. Gebser, B. Kaufmann, R. Kaminski, M. Ostrowski, T. Schaub, and M. Schneider, "Potassco: The Potsdam answer set solving collection," *AI Communications*, vol. 24, no. 2, pp. 107–124, 2011.

[20] N. Leone, G. Pfeifer, W. Faber, T. Eiter, G. Gottlob, S. Perri, and F. Scarcello, "The DLV system for knowledge representation and reasoning," *ACM Trans. on Computational Logic*, vol. 7, no. 3, pp. 499–562, 2006.

[21] T. Eiter, G. Ianni, and T. Krennwallner, "Answer set programming: A primer," in *Reasoning Web. Semantic Technologies for Information Systems*, ser. Lecture Notes in Computer Science, S. Tessaris, E. Franconi, T. Eiter, C. Gutierrez, S. Handschuh, M.-C. Rousset, and R. Schmidt, Eds. Springer Berlin Heidelberg, 2009, vol. 5689, pp. 40–110.

[22] E. Erdem, "Theory and applications of answer set programming," Ph.D. dissertation, The University of Texas at Austin, 2002.

[23] G. Didier, E. Remy, and C. Chaouiya, "Mapping multivalued onto boolean dynamics," *Journal of Theoretical Biology*, vol. 270, no. 1, pp. 177 – 184, 2011.

[24] A. Garg, L. Mendoza, I. Xenarios, and G. DeMicheli, "Modeling of multiple valued gene regulatory networks," in *Proceedings of the 29th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBS 2007)*. IEEE, 2007, pp. 1398–1404.

[25] L. Sanchez and D. Thieffry, "Segmenting the fly embryo:: a logical analysis of the pair-rule cross-regulatory module," *Journal of Theoretical Biology*, vol. 224, no. 4, pp. 517–537, 2003.

[26] A. Bockmayr and H. Siebert, *Programming Logics: Essays in Memory of Harald Ganzinger*. Springer Berlin Heidelberg, 2013, ch. Bio-Logics: Logical Analysis of Bioregulatory Networks, pp. 19–34.

[27] H. Lodish, A. Berk, S. L. Zipursky, P. Matsudaira, D. Baltimore, J. Darnell *et al.*, *Molecular cell biology*. WH Freeman New York, 2000, vol. 5.

[28] D. Van Nieuwenborgh, M. De Cock, and D. Vermeir, "Fuzzy answer set programming," in *Proceedings of the 10th European Conference on Logics in Artificial Intelligence*, 2006, pp. 359–372.

[29] M. Blondeel, S. Schockaert, D. Vermeir, and M. De Cock, "Complexity of fuzzy answer set programming under Łukasiewicz semantics," *International Journal of Approximate Reasoning*, vol. 55, no. 9, pp. 1971–2003, 2014.

[30] M. Alviano and R. Peñaloza, "Fuzzy answer sets approximations," *Theory and Practice of Logic Programming*, vol. 13, no. 4-5, pp. 753–767, 2013.

[31] M. Mushthofa, S. Schockaert, and M. De Cock, "A finite-valued solver for disjunctive fuzzy answer set programs," in *Proceedings of European Conference in Artificial Intelligence 2014*, 2014, pp. 645–650.

[32] ——, "Solving disjunctive fuzzy answer set programs," in *Proceedings of the 13th International Conference on Logic Programming and Nonmonotonic Reasoning*, 2015, pp. 453–466.

[33] M. Alviano and R. Peñaloza, "Fuzzy answer set computation via satisfiability modulo theories," *Theory and Practice of Logic Programming*, vol. 15, pp. 588–603, 7 2015.

[34] P. Vojtáš, "Fuzzy logic programming," *Fuzzy sets and systems*, vol. 124, no. 3, pp. 361–370, 2001.

[35] J. Lee and Y. Wang, "Stable models of fuzzy propositional formulas," in *Proceedings of the 14th European Conference on Logics in Artificial Intelligence, JELIA 2014*, 2014, p. 326.

[36] N. Madrid and M. Ojeda-Aciego, "Towards a fuzzy answer set semantics for residuated logic programs." in *Web Intelligence/IAT Workshops*, 2008, pp. 260–264.

[37] C. V. Damásio and L. M. Pereira, "Antitonic logic programs," in *Proceedings of the 6th International Conference on Logic Programming and Nonmonotonic Reasoning*, 2001, pp. 379–392.

[38] U. Straccia, "Annotated answer set programming," in *In: Proceedings of the 11th International Conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems (IPMU-06*, 2006.

[39] ——, "Managing uncertainty and vagueness in description logics, logic programs and description logic programs," in *Reasoning Web, 4th International Summer School, Tutorial Lectures*, ser. Lecture Notes in Computer Science, vol. 5224. Springer Verlag, 2008, pp. 54–103.

[40] M. Mushthofa, S. Schockaert, and M. De Cock, "Computing attractors of multi-valued gene regulatory networks using fuzzy answer set programming," in *Proceedings of the 2016 IEEE World Congress on Computational Intelligence*, 2016.

[41] R. Thomas, "Boolean formalization of genetic control circuits," *Journal of Theoretical Biology*, vol. 42, no. 3, pp. 563–585, 1973.

[42] I. Harvey and T. Bossomaier, "Time out of joint: Attractors in asynchronous random boolean networks," in *Proceedings of the Fourth European Conference on Artificial Life*, 1997, pp. 67–75.

[43] R. Thomas, "Regulatory networks seen as asynchronous automata: a logical description," *Journal of Theoretical Biology*, vol. 153, no. 1, pp. 1–23, 1991.

[44] H. Klarner, A. Bockmayr, and H. Siebert, "Computing maximal and minimal trap spaces of boolean networks," *Natural Computing*, vol. 14, no. 4, pp. 535–544, 2015.

[45] M. Blondeel, S. Schockaert, D. Vermeir, and M. De Cock, "Complexity of fuzzy answer set programming under Łukasiewicz semantics," *International Journal of Approximate Reasoning*, vol. 55, no. 9, pp. 1971–2003, 2014.

[46] S. Schockaert, J. Janssen, and D. Vermeir, "Fuzzy equilibrium logic: Declarative problem solving in continuous domains," *ACM Trans. on Computational Logic*, vol. 13, no. 4, pp. 33:1–33:39, 2012.

[47] T. Lukasiewicz and U. Straccia, "Tightly integrated fuzzy description logic programs under the answer set semantics for the semantic web," in *Proceedings of the 1st International Conference on Web Reasoning and Rule Systems*, 2007, pp. 289–298.

[48] S. Aguzzoli and A. Ciabattoni, "Finiteness in infinite-valued Łukasiewicz logic," *Journal of Logic, Language and Information*, vol. 9, no. 1, pp. 5–29, 2000.

[49] J. Guespin-Michel and M. Kaufman, "Positive feedback circuits and adaptive regulations in bacteria," *Acta Biotheoretica*, vol. 49, no. 4, pp. 207–218, 2001.

[50] T. Fayruzov, M. De Cock, C. Cornelis, and D. Vermeir, "Modeling protein interaction networks with answer set programming," in *IEEE Internat. Conf. on Bioinformatics and Biomedicine*, 2009, pp. 99–104.

[51] S. Peres and J.-P. Comet, "Contribution of computational tree logic to biological regulatory networks: Example from pseudomonas aeruginosa," in *Proceedings of the First International Workshop on Computational Methods in Systems Biology (CMSB 2003)*, 2003, pp. 47–56.

[52] L. Mendoza and I. Xenarios, "A method for the generation of standardized qualitative dynamical systems of regulatory networks," *Theoretical Biology and Medical Modelling*, vol. 3, no. 1, p. 13, 2006.

[53] R. Albert and A.-L. Barabási, "Statistical mechanics of complex networks," *Reviews of modern physics*, vol. 74, no. 1, p. 47, 2002.

[54] L. Mendoza and E. R. Alvarez-Buylla, "Dynamics of the genetic regulatory network for arabidopsis thaliana flower morphogenesis," *Journal of Theoretical Biology*, vol. 193, no. 2, pp. 307–319, 1998.

[55] L. Mendoza, D. Thieffry, and E. R. Alvarez-Buylla, "Genetic control of flower morphogenesis in arabidopsis thaliana: a logical analysis." *Bioinformatics*, vol. 15, no. 7, pp. 593–606, 1999.

[56] Y.-E. Sanchez-Corrales, E. R. Alvarez-Buylla, and L. Mendoza, "The arabidopsis thaliana flower organ specification gene regulatory network determines a robust differentiation process," *Journal of Theoretical Biology*, vol. 264, no. 3, pp. 971–983, 2010.

[57] F. Li, T. Long, Y. Lu, Q. Ouyang, and C. Tang, "The yeast cell-cycle network is robustly designed," *Proceedings of the National Academy of Sciences of the USA*, vol. 101, no. 14, pp. 4781–4786, 2004.

[58] L. Sánchez and D. Thieffry, "A logical analysis of the drosophila gap-gene system," *Journal of Theoretical Biology*, vol. 211, no. 2, pp. 115–141, 2001.

[59] R. Albert and H. G. Othmer, "The topology of the regulatory interactions predicts the expression pattern of the segment polarity genes in drosophila melanogaster," *Journal of theoretical biology*, vol. 223, no. 1, pp. 1–18, 2003.

[60] A. González, C. Chaouiya, and D. Thieffry, "Logical modelling of the role of the hh pathway in the patterning of the drosophila wing disc," *Bioinformatics*, vol. 24, no. 16, pp. i234–i240, 2008.