MASARYK UNIVERSITY

FACULTY OF INFORMATICS

Satisfaction Degree for Computation Tree Logic over Multivalued Gene Regulatory Networks

Master's Thesis

ANDREJ ŠIMURKA

Brno, Spring 2025

MASARYK UNIVERSITY

FACULTY OF INFORMATICS

Satisfaction Degree for Computation Tree Logic over Multivalued Gene Regulatory Networks

Master's Thesis

ANDREJ ŠIMURKA

Advisor: doc. RNDr. David Šafránek, Ph.D.

Department of Machine Learning and Data Processing

Brno, Spring 2025


FACULTAS ARTIS INFORMATICÆ
UNIVERSITAS MASARYKIANA

Declaration

Hereby I declare that this paper is my original authorial work, which I have worked out on my own. All sources, references, and literature used or excerpted during elaboration of this work are properly cited and listed in complete reference to the due source. During the preparation of this thesis, I used several AI tools. These tools are Grammarly for grammar check and ChatGPT to improve my writing style. I declare that I used these tools in accordance with the principles of academic integrity. I checked the content and took full responsibility for it.

Andrej Šimurka

Advisor: doc. RNDr. David Šafránek, Ph.D.

Acknowledgements

First and foremost, I would like to thank my advisor, doc. RNDr. David Šafránek, Ph.D., for his guidance and advice and for providing valuable feedback during my writing. I want to express my gratitude to my family and girlfriend for their unconditional support. I am also very thankful to my friend and colleague, Bc. Oto Stanko, with whom we exchanged knowledge throughout the research.

Abstract

Standard temporal logic model-checking techniques provide a binary verdict indicating whether a property is satisfied or not. While sufficient for many applications, this evaluation fails to capture how significantly a property holds or how severely it is violated. In biological systems, such strict evaluation can fail to capture important details. A system may satisfy or not satisfy a property. Still, it could be very close to the opposite case.

This thesis proposes a novel algorithm for computing a real-valued satisfaction degree for Computation Tree Logic (CTL) formulas evaluated over Multivalued Gene Regulatory Networks (MvGRNs). It addresses this limitation by providing a more nuanced evaluation of temporal logic properties. By enriching CTL semantics with signed distance and its propagation, we enable a more expressive analysis that captures the degree of satisfaction or violation of a temporal logic property.

Our method builds on ideas from the existing robustness analyses in real-valued and discrete-time temporal logics. It adapts them to the context of branching-time semantics. We introduce a slightly restricted yet expressive syntactic fragment of CTL. This restriction allows us to systematically decompose formulas and propagate satisfaction degrees from atomic formulas through logical and temporal operators.

The thesis provides both a formal foundation and a prototype implementation. We evaluate the method on small biological models to demonstrate how the satisfaction degree can be used to compare different setups of the same model with respect to a given property. Such comparison allows for a more informative analysis of how changes in the model dynamic influence its ability to satisfy temporal logic specifications.

Keywords

Multivalued networks, algorithm, asynchronous semantics, CTL model checking, robustness

Contents

Introduction 1

1 Preliminaries 4
- 1.1 Multivalued Gene Regulatory Networks 4
  - 1.1.1 Definition of MvGRNs 4
  - 1.1.2 Regulatory Contexts 6
  - 1.1.3 Compact Context Notation 7
  - 1.1.4 Semantics of MvGRN 7
  - 1.1.5 Attractors of MvGRN 9
- 1.2 Computational Tree Logic 10
  - 1.2.1 Kripke Structure 10
  - 1.2.2 Syntax of CTL Formulas 11
  - 1.2.3 Semantics of CTL Operators 11
  - 1.2.4 Model-Checking Procedure 12

2 Robustness in Temporal Logics 14
- 2.1 Robustness Degree in STL 14
- 2.2 Robustness Degree in MTL 15
- 2.3 Constraint Solving for QFCTL over Metric Space 16

3 Problem Definition 17

4 Satisfaction Degree for Atomic Formulas 21
- 4.1 Atomic Formulas 21
- 4.2 Signed Distance Definition 22
- 4.3 Satisfaction Degree Definition 24

5 Propagation of Satisfaction Degree 26
- 5.1 Syntax Restriction 26
- 5.2 Syntax Tree Decomposition 27
- 5.3 Satisfaction Degree of Operators 27
- 5.4 Quantitative Labeling Function 28

6 Algorithm Design 32
- 6.1 The Main Procedure Overview 32

6.2 Atomic Formula Evaluation 34
6.2.1 Computing the Domain of Validity 36
6.2.2 Identification of Border States 36
6.2.3 Computation of Weighted Distance 37
6.2.4 Computation of Maximal Depth 38
6.3 Logical Operators 40
6.4 Next Operators 40
6.5 Future Operators 41
6.6 Globally Operators 44
6.7 Until Operators 46
6.8 Weak Until Operators 48
6.9 Complexity 50

7 Implementation **53**
7.1 Third-Party Libraries 53
7.2 Input Specification 54
7.2.1 Network Definition 54
7.2.2 Formula Specification 55
7.2.3 Initial State Constraints 55
7.3 Input Parsing 56
7.4 Priority Queues 57
7.5 Multivalued Network 57
7.6 CTL Formula Components Classes 58
7.7 Computation of Satisfaction Degree 59
7.7.1 Weighted Distance Implementation 59
7.7.2 Extreme Depth Implementation 60

8 Evaluation **61**
8.1 Predator-Prey Model 61
8.1.1 Model Specifications 62
8.1.2 Results 63
8.2 Incoherent Feed-Forward Loop with Negative Feedback 65
8.2.1 Model Specifications 66
8.2.2 Results for the First Formula 67
8.2.3 Results for the Second Formula 69
8.2.4 Results for the Third Formula 70
8.2.5 Results for the Fourth Formula 71
8.2.6 Analysis of Non-Terminal Cycles 72

8.3 Single Input Module 73
8.3.1 Model Specifications 73
8.3.2 Results 75

9 Conclusion **77**

Bibliography **79**

A Github Repository Content **82**

B Specification of Evaluated Models **83**
B.1 Predator-Prey Model 83
B.2 Incoherent Feed-Forward Loop with Negative Feedback 84
B.2.1 Model 1 84
B.2.2 Model 2 85
B.2.3 Model 3 86
B.2.4 Model 4 87

Introduction

The family of temporal logics provides a powerful formalism for specifying and verifying properties of dynamic systems. Traditionally, the evaluation of a temporal logic formula has a Boolean result, indicating whether a formula is satisfied or not. However, in many applications, such a rough evaluation can be insufficient, as it does not reflect how significantly a formula is satisfied or how severely it is violated. To address this, various recent works have proposed approaches that quantify the satisfaction or violation of temporal logic formulas.

Donzé et al. [1] introduced an algorithm for computing the robustness of Signal Temporal Logic formulas over dense real-valued signals. Faniekos et al. [2] developed a multivalued semantics for Metric Temporal Logic, where satisfaction is evaluated based on the distance from a formula's validity domain. Their technique relies on constructing a robust neighborhood (tube) around the signal to define a region where the signal satisfies the formula. Similarly, Fages and Rizk [3] proposed a method for computing the satisfaction degree of Quantifier-Free CTL formulas over discrete domains. They construct validity domains parametrized by placeholders and evaluate satisfaction based on the signed distance to these domains.

Computation Tree Logic (CTL) is one of the most widely used temporal logics, especially in verifying the branching-time properties of finite-state systems. Standard CTL model checking is often applied in combination with Boolean Networks (BNs) [4, 5, 6], where system variables are restricted to binary values. While this is often sufficient for qualitative analysis, it lacks the expressiveness needed to quantify the degree of satisfaction or violation of a property.

To address these limitations, we adopt Multivalued Gene Regulatory Networks (MvGRNs). This modeling framework generalizes Boolean networks by allowing discrete variables to take on an arbitrary number of values. The general concept was first introduced by Thomas [7], who emphasized that many biological processes cannot be accurately captured using only binary states. For instance, when a single protein regulates multiple genes with different kinetics or when a gene exhibits leaky expression in the presence of both activators

and repressors, intermediate levels of gene expression are required to model such behaviors.

Recently, several frameworks for modeling MvGRNs have been proposed [8, 9, 10], introducing concepts such as regulatory graphs, activation levels, and regulatory contexts. Tools like GINsim [11, 12, 13] offer qualitative analysis based on these ideas. GINsim defines, for each regulator, an operating subinterval of its values under which the regulation is active. This thesis extends this approach by introducing an MvGRN modeling framework that captures how interactions behave across the full range of the regulator's values.

In this thesis, we build on existing ideas aiming to develop an algorithm that computes a bounded real-valued satisfaction degree for a given CTL formula evaluated from individual states of a given MvGRN. Rather than providing a Boolean outcome, our approach offers a more informative evaluation. Specifically, the satisfaction degree for each state reflects both - the truth value of the property and the relative magnitude of its satisfaction or violation concerning other possible scenarios within the examined network. This scale's upper bound represents the most significant satisfaction, while the lower bound indicates the most severe violation. This enables a clear differentiation between marginal cases and those that clearly satisfy or violate the property. To achieve this, we introduce a novel approach based on signed distance, evaluating states according to their position relative to the validity domains of atomic formulas. Additionally, we propose a method that systematically propagates the satisfaction degrees of atomic formulas through the temporal and logical operators of the CTL formula, thereby enabling the evaluation of the entire property.

This thesis is structured as follows. Chapter 1 consists of two parts. The first part introduces the formal Multivalued Gene Regulatory Networks model that extends existing approaches, while the second part summarizes the basics of Computation Tree Logic, on which we later build. Chapter 2 reviews related work and highlights the ideas that inspired our approach. Chapter 3 defines the main problem we aim to address: computing a satisfaction degree for CTL formulas over MvGRNs. Chapters 4 and 5 present the core of our method. The first defines how to compute satisfaction degrees for atomic formulas using signed and weighted Manhattan distances. The second describes how these values are propagated through CTL operators to achieve the

resulting satisfaction degree. In Chapter 6, we introduce and comment on the design of a prototype algorithm based on the previously stated formal definitions. Chapter 7 presents implementation-specific details along with the external libraries used. Chapter 8 summarizes the evaluation results of our algorithm on small biological models, highlighting its advantages by comparing different specifications for the same model topology. Finally, Chapter 9 discusses current limitations and suggests future improvements, especially in handling larger state spaces more efficiently.

1 Preliminaries

Before presenting the main contribution, we first introduce the key concepts and formal structures that we later build on.

In the first section, we define an extended framework for MvGRNs, which serve as the underlying model for our analysis. We describe their structure, state space, and the way they capture regulatory interactions between genes.

In the second section, we summarize the essential principles of CTL model checking, providing an overview of its syntax, semantics, and evaluation principles, based on [14]. This serves as a basis for understanding how the satisfaction degree extension build upon traditional model-checking techniques.

1.1 Multivalued Gene Regulatory Networks

In this section, we formally define Multivalued Gene Regulatory Networks (MvGRNs). This model generalizes Boolean networks to multivalued variable domains by introducing regulatory contexts, which extend the notion of Boolean update functions to handle multiple discrete activity levels. This formalism serves as the foundation for the modeling framework introduced in this thesis. The framework builds on the concept proposed by Klarner et al. [8] and extends the capabilities of GINsim regulatory networks [11].

1.1.1 Definition of MvGRNs

We define a Multivalued Gene Regulatory Network of n^{th} dimension $\mathcal{R}^{(n)}$ as a tuple $(\mathcal{V}, \mu, \mathcal{E}, \rho, \mathcal{F})$. For simplicity, the dimension is often omitted if it is intuitively inferable. In the following, we define the components of the $\mathcal{R}^{(n)}$ together with other auxiliary constructs.

$\mathcal{V} = \{v_1, \ldots, v_n\}$ is a finite set of n discrete variables with an arbitrary and further constant order. Although variables can represent different biological substances like genes, mRNAs, proteins, or even cellular processes and phenotypes, they are often referred to simply as *genes*. Instantaneous values of the variables indicate the activity levels of their respective biological components during the evaluation [15].

$\mu : \mathcal{V} \to \mathbb{N}^+$ is a mapping that assigns to each discrete variable from $\mathcal{V}$ its *maximum level of activity*. In the following, we simplify the notation from $\mu(v_i)$ to μ_i. Each variable then has its *discrete* interval of activity referred to as *domain*, defined as $\mathcal{D}(v_i) = [0 \dots \mu_i]$. Here, we also simplify the notation to $\mathcal{D}_i$. Additionally, we distinguish a *positive domain* of i, denoted as $\mathcal{D}_i^+ = \mathcal{D}_i \setminus \{0\}$.

The *state space* of a MvGRN $\mathcal{R}^{(n)}$, denoted $\mathcal{S}(\mathcal{R}^{(n)})$, is a set of all possible configurations of values of discrete variables $\mathcal{V}$. Therefore, we define it in the following way:

$$\mathcal{S}(\mathcal{R}^{(n)}) = \prod_{i=1}^{n} \mathcal{D}_i$$

The *state* of MvGRN $s \in \mathcal{S}(\mathcal{R})$ denotes one specific configuration of values of variables in $\mathcal{V}$.

$\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ is a set of edges, also referred to as *regulations* that represent interactions between components of the network. Considering a regulation $(u, v) \in \mathcal{E}$, we refer to the first element of the regulation as the *regulator* and to the second as the *target*. Later, we use the simplified notation $uv \in \mathcal{E}$.

$reg : \mathcal{V} \to 2^{\mathcal{V}}$ is a mapping that defines a set of *regulators* of each variable, such that $reg(v) = \{u \in \mathcal{V} \mid uv \in \mathcal{E}\}$.

A total function ρ assigns a tuple of *activation thresholds* to each regulation $uv \in \mathcal{E}$. Each regulation may have different number of thresholds, denoted as k. The activation thresholds must lie within the positive domain of the regulator, $\mathcal{D}_u^+$, and follow an ascending order. Therefore, $\rho(uv) = (\, t_1, \dots, t_k \mid 1 \leq t_1 < \dots < t_k \leq \mu_u \,)$. Based on this, the function divides the domain into $k + 1$ *activity intervals*, denoted $I_1^{uv} \dots I_{k+1}^{uv}$ (collectively referred to as $\mathcal{I}^{uv}$), as follows:

$$\mathcal{I}^{uv} = I_1^{uv} \dots I_{k+1}^{uv} = [0; t_1 - 1], [t_1; t_2 - 1], \dots, [t_k; \mu_u]$$

1.1.2 Regulatory Contexts

Let us consider a target variable $v \in \mathcal{V}$ with a set of regulators $reg(v) = \{u_1, \ldots, u_k\}$. Each regulator u_i has an associated sequence of activity intervals $\mathcal{I}^{u_i v}$, which partition its domain based on activation thresholds. The set of all feasible regulatory contexts for the target v, denoted by $\mathcal{C}_v$, is defined as the Cartesian product of these intervals:

$$\mathcal{C}_v := \prod_{i=1}^{k} \mathcal{I}^{u_i v}$$

A regulatory context $c_v \in \mathcal{C}_v$ is thus a tuple specifying one activity interval for each regulator of v. Each context corresponds to the set of states in which the regulators of v take values within the specified intervals. To identify the active context for a given state s, we define a function $\omega_i : \mathcal{S} \to \mathcal{C}_i$, which returns the unique regulatory context satisfied by state s for the target variable i. Since the intervals are disjoint, this context is uniquely determined.

Then, we define a mapping $\tau_i : \mathcal{C}_i \to \mathcal{D}_i$ that assigns a *target activity value* to each regulatory context of variable i.

Given a state s and a target activity value $t \in \mathcal{D}_i$, we define the following update notation:

$$s[i \to t] = (s_1, \ldots, s_{i-1}, s_i + sgn(t - s_i), s_{i+1}, \ldots, s_n)$$

This represents a quasi-continuous transition of the i^{th} component of state s towards the target value t.

Finally, we define a set of update functions $\mathcal{F} = \{f_1, \ldots, f_n\}$, where each $f_i : \mathcal{S} \to \mathcal{S}$ corresponds to variable v_i and is defined as:

$$f_i(s) = s[i \to \tau_i(\omega_i(s))]$$

Informally, the update function f_i selects the context of variable v_i that is satisfied in s, and updates the variable towards the corresponding target activity value.

Variables without regulators, typically modeling external influences on the system, are called *input variables*. For these, the set of contexts $\mathcal{C}_i$ consists of a single empty context, and τ_i maps it to a constant target activity value. Thus, the update function f_i always moves the variable towards this fixed value, regardless of the system state.

1.1.3 Compact Context Notation

Following the preceding definitions, the regulatory context of a target variable v is an element of the Cartesian product of the activity intervals of all its regulators. Hence, to specify a particular context formally, one must define the activity intervals for each regulator of v. However, this leads to a combinatorial explosion when enumerating all possible contexts.

To address this, we allow contexts to be defined using only a subset of intervals. Any unspecified interval is treated as a wildcard, representing all admissible intervals for the corresponding omitted regulator. This *compact notation* enables a single context definition to represent multiple formal contexts.

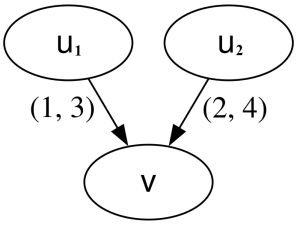
To maintain consistency with the formal semantics, context evaluation has to be performed in descending order concerning their size. A compact context captures all formal contexts that match its specified intervals unless a more specific, higher-priority context has already accounted for them.

For example, consider the simple regulatory motif illustrated in Figure 1.1, consisting of three variables and two regulations. Each regulation has two activation thresholds, dividing the activity domains into three intervals. Figure 1.2 demonstrates how the same regulatory behavior can be specified using either the full or the compact notation. Although their syntactic representations differ, both approaches are equivalent regarding the behavior they describe.

Since we consider defining the regulatory behavior using the full context notation to be not very user-friendly, we decided to enable the compact notation in our algorithm. To maintain equivalence with formal notation, it is necessary to pay attention to the order of evaluation of the compact contexts. We discuss this in more detail in Chapter 6.

1.1.4 Semantics of MvGRN

The semantics interprets the dynamics of an MvGRN. Specifically, semantics are defined as directed *state transition graphs* (STG) between states. There are two semantics commonly used in connection with regulatory networks - synchronous and asynchronous. They both describe different types of network behavior. Therefore, their STGs



$$\mathcal{V} = \{\, u_1, u_2, v \,\} \qquad \mathcal{E} = \{\, u_1 v, u_2 v \,\}$$
$$\mathcal{I}^{u_1 v} = I_1^{u_1 v}, I_2^{u_1 v}, I_3^{u_1 v} = [0;0], [1;2], [3;4]$$
$$\mathcal{I}^{u_2 v} = I_1^{u_2 v}, I_2^{u_2 v}, I_3^{u_2 v} = [0;1], [2;3], [4;4]$$

Figure 1.1: The definition and illustration of a simple regulatory motif with three variables u_1, u_2 and v, and two regulations $u_1 \rightarrow v$ and $u_2 \rightarrow v$. Both regulations have two activation thresholds, which divide their activities into the three intervals specified below.

use to differ significantly [16, 17]. In this thesis, we consider only asynchronous semantics. In *asynchronous semantics*, individual variables are updated independently. Therefore, the resulting STG is not deterministic. Formally $STG(\mathcal{R}^{(n)}) = (S, T)$ where $S = \mathcal{S}(\mathcal{R}^{(n)})$ and

$$T = \bigcup_{i=1}^{n} f_i$$

Generally, one state can have up to n successors. However, we introduce an additional restriction on the STG in our approach. Specifically, if a state has at least one successor different from itself yet also includes a self-loop, i.e., its own successor, the self-loop is removed. In other words, a state is allowed to have a self-loop only if it has no other successors. This restriction partially captures a fairness property by excluding paths that could transition to a different state but instead remain stationary. Although this does not guarantee complete fairness, such a trivial restriction significantly improves the expressive power of the proposed modeling framework. Formally:

$$\exists s' \in S : (s, s') \in T \wedge s \neq s' \iff (s, s) \notin T$$

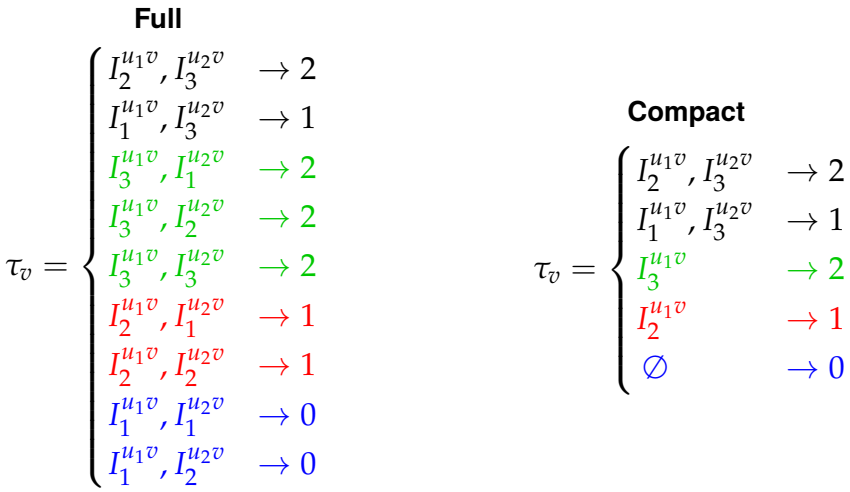


Figure 1.2: The comparison of the full and the compact context notation. The first two contexts (in black) are fully specified, i.e., they both capture one formal context. The third context (in green) has a specified interval only for the regulator u_1 and thus expands to three formal contexts. The fourth context (in red) has again specified interval only for u_1. However, this context expands to only two formal contexts because one instance has already been included in the higher-priority context. The last context (in blue) is empty and represents all possible contexts. However, 7 out of 9 have been defined before. Therefore, it expands only to the remaining two contexts.

1.1.5 Attractors of MvGRN

Since the size of the state space of MvGRN is finite, after a large enough number of transitions is performed, the network must return to a previously visited state. This implies the existence of repeating transition sequences [18]. Such sequences define a long-term behavior of the network and are referred to as *attractors*. The set of attractors corresponds to terminal strongly connected components of the STG.

According to multiple studies [19, 20, 21], asynchronous semantics give rise to three types of long-term behavior:

- *Steady-state* consists of a single state and indicates the system's stability. Steady-states are represented as self-loops in the STG. Therefore, further transitions do not result in any change of state.

- *Oscillation* is a cyclic attractor which consists of two or more states. Each state has exactly one successor in the STG, and any two adjacent states have a Manhattan distance of one.
- *Disorder* includes multiple interlinked oscillations. Each state can have more than one successor in the STG, and there is only a Manhattan distance of one between any two adjacent states. The system tends to behave unpredictably.

1.2 Computational Tree Logic

This section provides a concise yet comprehensive overview of aspects of standard Computation Tree Logic (CTL) model checking that are relevant to our work. We present the syntax of CTL formulas and the formal semantics of temporal and logical operators. Furthermore, we outline the standard model-checking procedure, which involves syntactic traversal of the formula's parse tree and iterative labeling of the states within a Kripke structure. Subsequently, we modify these standard approaches to adapt them to the computation of the satisfaction degree of CTL formulas, thereby introducing a quantitative perspective on the logical satisfaction of formulas over MvGRNs.

Throughout the whole section, we adopt [14] as our primary reference, applying minor changes in notation to keep consistency with the notation used later in this thesis.

1.2.1 Kripke Structure

A *Kripke structure* $\mathcal{K}$ is a tuple (S, T, I, AP, L), where:

- S is a finite set of states.
- $T \subseteq S \times S$ is a total transition relation between states.
- $I \subseteq S$ is the set of initial states from which the evaluation of CTL formulas is initiated.
- AP is a finite set of atomic propositions.
- $L : S \to 2^{AP}$ is a labeling function that assigns to each state a set of atomic propositions that hold in that state.

1.2.2 Syntax of CTL Formulas

Atomic propositions (APs) describe basic state properties within the system. Their role is further discussed in Section 4.1.

In standard CTL, formulas are divided into state formulas (Φ_s) and path formulas (Φ_p). A crucial syntactic rule in CTL is that every path formula must be preceded by a quantifier, meaning path formula alone does not form a complete CTL formula.

CTL formulas are constructed according to the following grammar:

$$\Phi_s ::= \text{true} \mid a \mid \Phi_s \wedge \Phi_s \mid \Phi_s \vee \Phi_s \mid \neg\Phi_s \mid \mathsf{E}\,\Phi_p \mid \mathsf{A}\,\Phi_p$$

$$\Phi_p ::= \mathsf{X}\,\Phi_s \mid \mathsf{F}\,\Phi_s \mid \mathsf{G}\,\Phi_s \mid \Phi_s\,\mathsf{U}\,\Phi_s \mid \Phi_s\,\mathsf{W}\,\Phi_s$$

where a is an AP, each Φ_s is a state formula and Φ_p is a path formula. We adopt a self-descriptive notation of CTL operators, namely: A (for all), E (exists), X (next), F (future), G (globally), U (until) and W (weak until).

1.2.3 Semantics of CTL Operators

Satisfaction of a CTL formula by the state $s \in S$ in Kripke structure $\mathcal{K}$ is defined for the standard CTL approach as follows:

$$\begin{aligned}
s \models a &\iff a \in L(s) \\
s \models \neg\Phi_s &\iff \neg(s \models \Phi_s) \\
s \models \Phi_{s_1} \wedge \Phi_{s_2} &\iff (s \models \Phi_{s_1}) \wedge (s \models \Phi_{s_2}) \\
s \models \Phi_{s_1} \vee \Phi_{s_2} &\iff (s \models \Phi_{s_1}) \vee (s \models \Phi_{s_2}) \\
s \models \mathsf{E}\,\Phi_p &\iff \pi \models \Phi_p \text{ for some } \pi \in \text{Paths}(s) \\
s \models \mathsf{A}\,\Phi_p &\iff \pi \models \Phi_p \text{ for all } \pi \in \text{Paths}(s)
\end{aligned}$$

For an infinite path $\pi = s_1, s_2, s_3, \ldots$ over finite number of states, the satisfaction relation $\models$ of path formula is defined as follows:

$$
\begin{aligned}
\pi \models \mathsf{X}\,\Phi_s &\iff s_1 \models \Phi_s \\
\pi \models \mathsf{F}\,\Phi_s &\iff s_j \models \Phi_s \text{ for some } j \geq 0 \\
\pi \models \mathsf{G}\,\Phi_s &\iff s_j \models \Phi_s \text{ for all } j \geq 0 \\
\pi \models \Phi_{s_1} \mathsf{U}\,\Phi_{s_2} &\iff \exists j \geq 0 : (s_j \models \Phi_{s_2} \wedge (\forall 0 \leq k < j : s_k \models \Phi_{s_1})) \\
\pi \models \Phi_{s_1} \mathsf{W}\,\Phi_{s_2} &\iff \big(\exists j \geq 0 : \big(s_j \models \Phi_{s_2} \wedge \forall\, 0 \leq k < j : s_k \models \Phi_{s_1}\big)\big) \\
&\qquad \vee\, (\forall\, k \geq 0 : s_k \models \Phi_{s_1})
\end{aligned}
$$

Finally, the CTL formula Φ_s is satisfied by the Kripke structure $\mathcal{K}(S, T, I, AP, L)$ if and only if $\forall s \in I : s \models \Phi_s$. In a special case when $I = S$ we call it *global model checking*.

1.2.4 Model-Checking Procedure

The core principle of CTL model checking is to determine, for each subformula of a given CTL state formula Φ_s, the set of states in which it is satisfied. To enable this, the set of APs is systematically extended with newly derived propositions corresponding to more complex subformulas of Φ_s. The labeling function L, which assigns to each state a set of APs that hold in it, is accordingly extended to include these new labels. As a result, each state is labeled not only with the original APs but also with labels indicating satisfaction of more complex subformulas. By recursively evaluating and labeling each subformula of Φ_s, the satisfaction of the entire formula can be efficiently determined.

Formally, let $Sub(\Phi_s)$ denote the set of all subformulas of Φ_s, and let a_φ be the fresh atomic proposition associated with each $\varphi \in Sub(\Phi_s)$. The labeling function is iteratively extended such that:

$$\forall \varphi \in Sub(\Phi) : \quad a_\varphi \in L(s) \iff s \in \mathrm{Sat}(\varphi)$$

where $\mathrm{Sat}(\varphi)$ denotes the satisfaction set of φ, defined as the set of all states in which φ holds, i.e., $\mathrm{Sat}(\varphi) = \{\, s \in S \mid s \models \varphi \,\}$.

Given a Kripke structure $\mathcal{K}(S, T, I, AP, L)$, the model-checking procedure proceeds by traversing the syntax tree of the formula Φ_s in a bottom-up fashion. Satisfaction sets are computed inductively: atomic propositions are satisfied by the states labeled with them, logical operators like conjunction, disjunction, and negation are evaluated using

set operations, and temporal operators such as G or U are handled using *fixed-point* computations.

The procedure terminates when the satisfaction set of the root formula Φ_s is determined. The CTL formula Φ_s is then said to hold in the system if all initial states I belong to $\mathrm{Sat}(\Phi_s)$, i.e., $I \subseteq \mathrm{Sat}(\Phi_s)$.

In the following sections, we build upon this standard procedure by redefining and extending it to allow for computing the continuous satisfaction degree of CTL formulas over MvGRNs.

2 Robustness in Temporal Logics

Temporal logic plays a crucial role in formal verification, enabling the specification and analysis of system behaviors over time. However, the traditional model-checking approach provides only Boolean results - whether a formula holds or not - without capturing how robustly the system satisfies the given specification or how severely it violates it. Robustness in temporal logic aims to quantify the degree of satisfaction or violation of a formula across the set of initial states, offering a more nuanced understanding of system behavior [22].

In this chapter, we provide an overview of existing approaches to robustness in temporal logic, examining how different frameworks extend classical Boolean semantics to incorporate robustness measures.

2.1 Robustness Degree in STL

Donzé et al. [1] address the challenge of monitoring real-time systems using Signal Temporal Logic (STL), a formalism designed for specifying and analyzing the behavior of dense-time real-valued signals in continuous and hybrid systems. Standard STL evaluation methods provide binary assessment. The authors introduce the quantitative semantics of STL, which assigns a robustness degree to each formula.

A key concept in this framework is the *validity domain* of a formula, which consists of all states where the formula holds. For a simple predicate such as $x < c$, the validity domain is $X_1 = \{x \mid x < c\}$, while its complement, $X_0 = \{x \mid x \geq c\}$, represents violation. The robustness degree quantifies how far a given state is from the boundary of the validity domain. In this example, the robustness of $x < c$ is given by $c - x$, where the sign indicates whether the formula is satisfied ($x < c$) or violated ($x \geq c$), and the magnitude measures the distance from the threshold.

The robustness framework extends to logical and temporal operators. Conjunction propagates robustness by taking the minimum value between subformulas, while disjunction takes the maximum. Negation inverts the robustness sign. For temporal operators, the robustness of globally is computed as the minimum robustness over an interval, while future takes the maximum. Until propagates ro-

bustness by taking the supremum over all valid time points, where the robustness is determined by the minimum of the robustness of the second property and the infimum of the robustness of the first property over the preceding interval.

This notion of robustness and validity domains serves as a crucial concept for our work, where we aim to adapt similar principles to a different logical framework.

2.2 Robustness Degree in MTL

Fainekos and Pappas [2] introduce *signed distance* as a quantitative measure for MTL robustness, similar to STL. Given a metric d, the distance of a point x from a set S is the shortest distance to S. In contrast, the *depth* of x in S measures proximity to its boundary. For $x \in X$ be a point, $S \subseteq X$ be a set, $cl(S)$ its closure, and d be a metric on X, they define distance from x to S as $\text{dist}_d(x, S) := \inf\{d(x, y) \mid y \in \text{cl}(S)\}$, depth of x in S is given by: $\text{depth}_d(x, S) := \text{dist}_d(x, X \setminus S)$ and signed distance from x to S is defined as:

$$\text{Dist}_d(x, S) := \begin{cases} -\text{dist}_d(x, S) & \text{if } x \notin S \\ \text{depth}_d(x, S) & \text{if } x \in S \end{cases}$$

The *signed distance function* unifies both: it is positive inside S, negative outside, and its magnitude reflects the closeness to the boundary. They define the robustness degree as the bound on the perturbation that a signal can tolerate without changing its Boolean truth value concerning a specification expressed in MTL.

This concept is crucial for our extension of CTL with quantitative semantics, as it allows us to define a robustness measure for states with respect to atomic properties. By adapting signed distance, we can quantify how strongly a system satisfies (or violates) an atomic formula rather than relying on a strict Boolean evaluation.

2.3 Constraint Solving for QFCTL over Metric Space

Fages and Rizk [3] extend classical model checking by reformulating it as a constraint-solving problem, enabling the inference of kinetic parameters from temporal logic specifications. Instead of the traditional Boolean evaluation of temporal properties, they introduce a continuous degree of violation, quantifying how severely a system deviates from a given specification. This generalization allows free variables in temporal formulas, making it possible to compute the validity domains of these variables.

Their work extends QFCTL constraint solving to metric spaces, where the computation domain is equipped with a distance function. This enables a continuous evaluation of QFCTL formulas on a scale from 0 to 1 rather than a strict Boolean result.

A key concept in their approach is a *pattern formula* $\psi(x_1, \ldots, x_k)$ for a given QFCTL formula φ, where some constants in the original formula φ are replaced by variables $\{x_1, \ldots, x_k\}$. The *violation degree* of φ is then defined using the distance between the validity domain of the variables $x_1, \ldots, x_k$ in ψ, denoted as D_y^ψ and the objective values $v = (v_1, \ldots, v_k)$ as follows:

$$vd(\varphi, \psi) = \min_{v' \in D_y^\psi} d(v', v)$$

In order to normalize this violation degree, the authors define a *satisfaction degree*, which provides a measure of how well a system satisfies a given temporal specification as follows:

$$sd(\varphi, \psi) = \frac{1}{1 + vd(\varphi, \psi)}$$

The only problem with this approach is that it does not express the robustness of the formula satisfaction. Suppose the computed validity domain contains the original objective values of the variables. In that case, the violation degree is zero, and the satisfaction degree is one. We want to extend this concept to reflect both the violation and satisfaction degree.

3 Problem Definition

In this chapter, we define the problem of satisfaction degree computation for Computation Tree Logic (CTL) over Multivalued Gene Regulatory Networks (MvGRNs). This thesis aims to design and implement an algorithm that extends standard CTL model checking by introducing a method for evaluating formula satisfaction on a continuous scale rather than producing only Boolean outcomes.

MvGRNs are well-suited for this problem, as they represent gene expression using multiple discrete activity levels. This multivalued nature enables a more nuanced evaluation of whether and how strongly a temporal property holds in individual model states. It allows us to distinguish between states that strongly satisfy the property and those that satisfy it only marginally, as well as between strong and weak violations using distances.

Several previous works have also addressed this problem. One notable approach, proposed by Fages et al. [3], introduces a violation degree that quantifies the extent to which a system fails to satisfy a formula. However, this method has a key limitation: it does not capture the degree of satisfaction. Once a state lies within the validity domain, it is assigned a satisfaction value of one, regardless of how robustly the formula holds. Our goal is to enhance this concept by incorporating a graded notion of satisfaction as well.

The inputs of the problem are a Multivalued GRN $\mathcal{R}^n$, a CTL state formula Φ_s specifying a temporal property of interest, and a set of initial states I. Using these inputs, we construct a modified Kripke structure $\mathcal{K}(S, T, I, \mathcal{Q})$ that captures the system's state transitions. The goal is to compute a quantitative labeling function $\mathcal{Q}$, which serves as an alternative to the labeling function L from the standard approach. This function defines a total mapping from each pair of a network state and a subformula of the property of interest to a real-valued satisfaction degree.

The function $\mathcal{Q}$ is computed recursively analogous to the computation of the labeling function L in the standard model checking. The process begins with the evaluation of atomic formulas and proceeds by propagating values through the syntactic structure of the CTL formula. However, unlike the standard approach, which assigns

labels corresponding to individual atomic propositions, our method computes real-valued satisfaction degree for the entire atomic formulas with respect to each state. It then systematically propagates these values through the logical and temporal operators of subformulas.

The output of the problem is a statistical summary of the satisfaction degrees across the initial states I of the Kripke structure. Instead of returning a single Boolean outcome, the result includes values such as the minimum, maximum (each accompanied by a corresponding state), and average satisfaction degree. These are obtained by aggregating the values of the function $\mathcal{Q}$ once computed.

The goal is to provide a complementary metric that quantifies the degree of satisfaction or violation. This measure is bounded, signed, and relative. Its sign shows whether the formula is satisfied or violated, and its magnitude expresses how strongly the model behavior exhibits this property compared to other possible behaviors the model allows. While such a metric may not preserve logical equivalence between formulas, we aim to prioritize practical utility over formal invariance. Specifically, we aim to provide a method for comparing different model specifications concerning the same temporal property, offering more informative insights than binary satisfaction can provide.

We use the following figures to illustrate the comparative ability of our approach, focusing on two of the most common CTL operators: F (future) and G (globally). Each figure presents a pair of trees, both depicting the *unfolding* of all possible trajectories starting from the same state (root). The trees correspond to two slightly different model specifications, allowing us to visually compare how the variation affects system behavior with respect to the examined property. Although unfoldings represent the branching structure of all possible evolutions from a single state and are conceptually infinite, we truncate them for clarity, assuming the omitted parts do not further affect the resulting satisfaction degree.

The colors of states encode their satisfaction degree with respect to a given atomic formula Φ_a: red indicates a severe violation, orange is a moderate violation, yellow is a marginal satisfaction, light green is a moderate satisfaction, and dark green is a strong satisfaction.

Suppose we are given two similar MvGRN model specifications and a CTL formula AF Φ_a to evaluate, with a single initial state - the root. The two models yield slightly different unfoldings starting from

this initial state, depicted in Figure 3.1. The evaluation using the standard CTL model checking would return *true* for both specifications. However, our approach not only confirms the satisfaction for both specifications but also quantifies the strength of the satisfaction, allowing for model comparison.

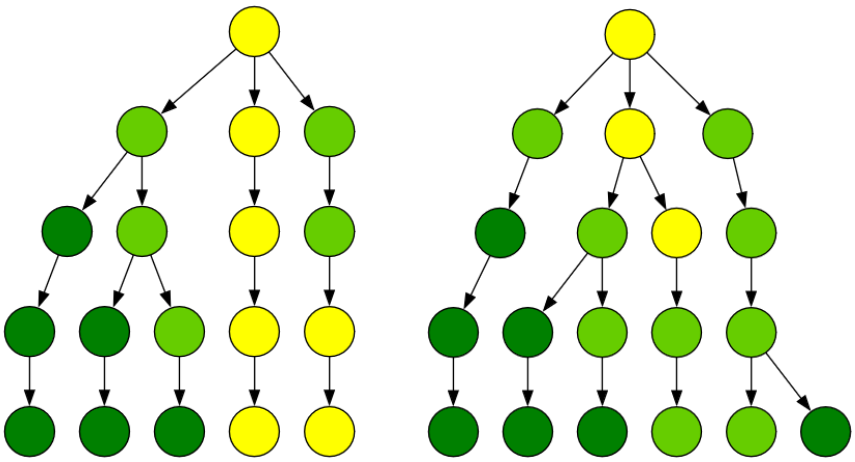


Figure 3.1: Comparison of two unfoldings originating from the same initial state for two distinct models. Each tree represents the unfolding of all possible model trajectories from the given initial state (root). Node colors indicate the degree of satisfaction of a given atomic formula Φ_a. Although both dynamics satisfy the formula AF Φ_a in the Boolean sense, our approach distinguishes the strength of satisfaction: The left unfolding leads to a weaker (yellow) satisfaction. In contrast, the right unfolding ensures stronger (light green) satisfaction.

The resulting satisfaction degree for AF Φ_a in the root state of the left unfolding corresponds to yellow, reflecting the existence of a trajectory where only a marginal satisfaction is achieved. In the slightly modified model specification, corresponding to the right unfolding, weaker trajectories are absent. Consequently, the satisfaction degree in the root state corresponds to light green, as every trajectory eventually passes through a light green-colored state. Thus, although both dynamics satisfy the formula in the classical Boolean sense, our approach distinguishes between them based on the strength of satisfaction, favoring the right model.

In a different situation, illustrated in Figure 3.2, we consider the evaluation of a CTL formula EG Φ_a. Both model specifications satisfy the formula in the classical Boolean sense, as each unfolding contains a trajectory along which Φ_a holds globally.

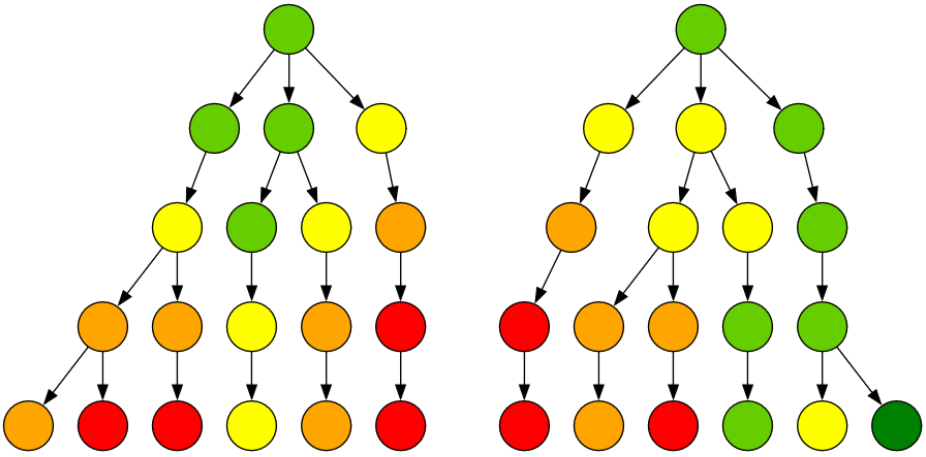


Figure 3.2: Comparison of two unfoldings originating from the same initial state for two distinct models. Each tree represents the unfolding of all possible model trajectories from the given initial state (root). Node colors indicate the degree of satisfaction of a given atomic formula Φ_a. Although both dynamics satisfy the formula EG Φ_a in the Boolean sense, our approach distinguishes the strength of satisfaction: The left unfolding leads to weaker (yellow) satisfaction. In contrast, the right unfolding ensures a stronger (light green) satisfaction.

In the left unfolding, the optimal trajectory maintains a satisfaction degree corresponding to yellow. In contrast, the right unfolding features a trajectory that sustains light green satisfaction throughout. This difference leads to a higher overall satisfaction degree for the right model. Therefore, although the standard model checking provides the same Boolean result for both dynamics, our approach is able to differentiate between them by quantifying the strength of satisfaction, favoring the right model.

4 Satisfaction Degree for Atomic Formulas

In this chapter, we define formalisms that express the satisfaction degree of the individual states of MvGRN with respect to a property defined by an atomic formula. As mentioned above, unlike standard model checking, we evaluate atomic formulas as a whole, not the individual atomic propositions separately.

4.1 Atomic Formulas

As already mentioned, atomic propositions (APs) intuitively express simple state properties of a system. Naturally, they can be used to constrain a subspace of states. In our case, we can use APs to constrain a subinterval of the activity domain for a specific variable. The set of all feasible APs is defined as follows:

$$AP := \{\, v_i \,\square\, k \mid v_i \in \mathcal{V},\ \square \in \{\geq, \leq\},\ k \in \mathcal{D}_i \,\}$$

Note that we limit ourselves to only two comparison operators since they have sufficient expressive power for our purposes. A given AP defines a subset of the state space, satisfying a property that intuitively follows from its notation. To achieve more complex constraints, individual APs can be chained using standard logic operators $\wedge$ and $\vee$. Such a chain of APs using logical operators is referred to as *atomic formula*, distinguished by a subscript a (e.g., Φ_a). Naturally, each atomic formula is also a valid state formula. Feasible atomic formulas are constructed according to the following grammar:

$$\Phi_a ::= ap \mid \Phi_{a_1} \wedge \Phi_{a_2} \mid \ \Phi_{a_1} \vee \Phi_{a_2}$$

where $ap \in AP$ and $\Phi_a, \Phi_{a_1}, \Phi_{a_2}$ are atomic formulas. We denote the set of all well-formed atomic formulas according to the declared grammar as $\mathcal{A}$. Since APs constrain sets of states, their combinations using logical operators are interpreted as set operations $\cap$ and $\cup$, respectively. We formally disallow negations of atomic propositions, as they do not increase expressive power in the context of a discrete state space. The negation of an atomic proposition can be achieved by constraining its complement within the entire state space. Although

the implementation supports negations of atomic formulas for user convenience, they are internally converted to positive formulas.

We adopt a pattern from the standard CTL model checking, in which the labeling function $L : \mathcal{S} \to 2^{AP}$ assigns a set of labels that hold in each state. During the bottom-up evaluation of a CTL formula's syntax tree, this set of labels is updated as subformulas are processed [14]. However, our approach does not evaluate the satisfaction degree of the individual APs separately. Instead, we assign a single real-valued satisfaction degree to each state with respect to the entire atomic formula.

As mentioned above, an atomic formula intuitively constrains a subset of states within the state space. This subset, where the atomic formula Φ_a holds, is referred to as its *domain of validity* (DoV), denoted as Ω_{Φ_a}. Notably, allowing the disjunction of APs causes the DoV not necessarily to be a convex set. Furthermore, we distinguish the DoV boundary, denoted as $\partial\Omega_{\Phi_a}$, which is defined as follows:

$$\partial\Omega_{\Phi_a} = \{\ s \in \Omega_{\Phi_a} \mid \exists s' \in \mathcal{N}_1(s) : s' \notin \Omega_{\Phi_a}\ \}$$

where $\mathcal{N}_1(s) = \{\ s' \mid \sum_{i=1}^{n} |s_i - s'_i| = 1\ \}$ stands for a one-step Manhattan neighborhood of state s.

We also refer to the complement of the DoV within the state space $\mathcal{S}$, denoted $\overline{\Omega}\Phi_a$, which corresponds to the set of states where $\neg\Phi_a$ holds. Similarly, its boundary is denoted as $\partial\overline{\Omega}_{\Phi_a}$.

4.2 Signed Distance Definition

In this section, we define a signed distance function $sd : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ that quantifies both the distance and relative position of a state with respect to the DoV of an atomic formula. The signed distance function follows these principles:

- If a state lies *inside* the DoV, its signed distance is *positive*, indicating how far the state is from violation. This distance, referred to as *depth*, is the shortest distance to any state outside the DoV.
- If a state is *outside* the DoV, its signed distance is *negative* and the absolute value reflects how far the state is from satisfaction. Namely, it is equal to the shortest distance to any state within the DoV.

Let us consider an MvGRN $\mathcal{R}^{(n)}$, where the variables may have different maximum activity levels μ_i. The state space of such a network forms an n-dimensional discrete space, with each dimension corresponding to a network variable $v_i \in \mathcal{V}$. The domain of each dimension is given by the activity domain $\mathcal{D}_i$ of the corresponding variable v_i.

Let $s \in \mathcal{S}$ be a state and Φ_a an atomic formula. The atomic propositions forming Φ_a impose constraints on a subset of the system variables but not necessarily on all n variables. Without loss of generality, we assume that Φ_a constrains exactly the first c variables, denoted $v_1, \ldots, v_c$. Consequently, the computation of the signed distance between s and Ω_{Φ_a} is based solely on the projection of the state space onto these first c dimensions. At the same time, the remaining unconstrained variables are omitted, as they do not affect the evaluation.

Since we operate in a discrete space, we measure distances between states using the Manhattan distance. This allows us to break down the calculation into contributions from the individual dimensions and sum them up. However, it is crucial to note that variables may have different sizes of their activity domains $\mathcal{D}_i$, meaning that variables can have different maximum activity values μ_i. This leads to an important consideration. The same absolute Manhattan distance has greater significance in a dimension with fewer activity levels. That is because the same absolute distance in a smaller domain represents a larger difference compared to a domain with more activity levels. To account for this, we weight the absolute Manhattan distances by the inverse of the domain size of each dimension.

Let us assume two states $s, s' \in \mathcal{S}$. The absolute Manhattan distance between s and s' along the i^{th} constraint dimension ($i \in \{1, 2 \ldots c\}$) denoted as d_i is defined as follows:

$$d_i(s, s') = |s_i - s'_i|$$

Accordingly, weighted distance along i^{th} dimension (weight is inversely proportional to the size of the dimension) is:

$$wd_i(s, s') = d_i(s, s') * \frac{1}{|\mathcal{D}_i|}$$

Thus, the weighted distance for two states is defined as the sum of weighted distances along all the constrained dimensions:

$$wd(s, s') = \sum_{i=1}^{c} wd_i(s, s')$$

Finally, we define a signed distance *sd* for a state s and atomic formula Φ_a in the following way:

$$sd(s, \Phi_a) = \begin{cases} wd_{min}(s, \overline{\Omega}_{\Phi_a}) & \text{if } s \in \Omega_{\Phi_a}, \\ -wd_{min}(s, \Omega_{\Phi_a}) & \text{if } s \notin \Omega_{\Phi_a}. \end{cases}$$

where wd_{min} is defined for an arbitrary domain Ω as follows:

$$wd_{min}(s, \Omega) = \min_{s' \in \partial\Omega} \; wd(s, s'),$$

Intuitively, suppose a state is inside the DoV of the formula. In that case, its signed distance corresponds to its weighted distance from the nearest state outside the DoV. Conversely, if a state lies outside the DoV, its signed distance equals the weighted distance to the nearest state inside the DoV. Obviously, the closest state of the complementary set is always one of its boundary states.

4.3 Satisfaction Degree Definition

In this section, we define the *satisfaction degree* $\delta^* : \mathcal{S} \times \mathcal{A} \to [-1, 1]$. This function quantifies the level of satisfaction of an atomic formula Φ_a in a given state s.

The satisfaction degree must satisfy the following properties:

- If a state lies inside the formula's DoV, its satisfaction degree is in the range $(0, 1]$.
- If a state lies outside the formula's DoV, its satisfaction degree is in the range $[-1, 0)$.
- Since we operate in discrete state space, the value of the satisfaction degree can never be equal to zero.
- The state that maximally violates the formula is assigned a satisfaction degree of -1. In contrast, the state that maximally satisfies the formula is assigned a 1.

Since the satisfaction degree must always fall within the range $[-1, 1]$, the absolute distance must be normalized relative to the maximum possible value for the given formula that defines the constraining conditions. Based on this requirement, we formally define the satisfaction degree function δ^* as follows:

$$\delta^*(s, \Phi_a) = \begin{cases} \frac{sd(s,\Phi_a)}{wd_{max}(\Omega_{\Phi_a})} & \text{if } s \in \Omega_{\Phi_a}, \\ \frac{sd(s,\Phi_a)}{wd_{max}(\overline{\Omega}_{\Phi_a})} & \text{if } s \notin \Omega_{\Phi_a}. \end{cases}$$

where wd_{max} is defined for an arbitrary domain Ω as follows:

$$wd_{max}(\Omega) = \max_{s \in \Omega} \; wd_{min}(s, \overline{\Omega})$$

The function wd_{max} for a given domain Ω returns the maximum depth among all states within the set. This is determined by computing the maximum of the minimal distances from individual states to the domain complement.

The satisfaction degree function δ^* for a state s with respect to the atomic formula Φ_a is defined as the ratio of the signed distance of the state and formula to the maximum weighted distance among the states in the same domain as s. If a state lies inside the DoV, the maximum is computed from the states inside the DoV. If a state lies outside the DoV, the maximum is calculated from the complement of the DoV.

In this chapter, we formally defined the satisfaction degree of a given state with respect to an atomic formula. In the next chapter, we explain how the satisfaction degree of an atomic formula is propagated through individual CTL operators.

5 Propagation of Satisfaction Degree

This chapter formally defines the propagation of the previously introduced satisfaction degree for atomic formulas through temporal and logical operators of CTL. The propagation is described using formal notation, while the subsequent chapter focuses on algorithmic realization, which addresses challenges not captured by the formalisms.

We introduce the restricted CTL syntax under consideration, followed by an explanation of formula decomposition and its systematic evaluation. Then, we define the modified semantics for the supported operators, adapted to propagate satisfaction degrees. Finally, we present the definition of the quantitative labeling function $\mathcal{Q}$, which assigns satisfaction degree values to individual states with respect to all the subformulas of the evaluated formula. We conclude with both a textual and visual interpretation of $\mathcal{Q}$.

5.1 Syntax Restriction

We impose a syntactic restriction on CTL formulas to support a structured evaluation. Specifically, we logically separate atomic formulas Φ_a, which are evaluated using a dedicated signed distance method, from the rest of the logical structure. This separation allows us to treat the satisfaction degree of atomic formulas independently and propagate their results through the rest of the formula in a uniform way.

Under this restriction, atomic subformulas must appear in a compact, left-aligned form within their local subformulas. The left alignment ensures compactness of the atomic part at each level of nesting, such that its satisfaction degree can be directly integrated into the evaluation of higher-level operators.

Formulas respecting this structure are generated by the following grammar, with the initial non-terminal Φ_s:

$$\Phi_s ::= \Phi_a \mid \Phi'_s \mid \Phi_a \wedge \Phi'_s \mid \Phi_a \vee \Phi'_s$$

$$\Phi'_s ::= \Phi'_s \wedge \Phi'_s \mid \Phi'_s \vee \Phi'_s \mid \mathsf{E}\,\Phi_p \mid \mathsf{A}\,\Phi_p$$

$$\Phi_p ::= \mathsf{X}\,\Phi_s \mid \mathsf{F}\,\Phi_s \mid \mathsf{G}\,\Phi_s \mid \Phi_s \mathsf{U}\,\Phi_s \mid \Phi_s \mathsf{W}\,\Phi_s$$

where $\Phi_a \in \mathcal{A}$. The language of all well-formed formulas generated by this grammar is denoted by $\mathcal{L}$. Clearly, $\mathcal{A}$, the set of all atomic formulas, is a subset of $\mathcal{L}$.

5.2 Syntax Tree Decomposition

The syntax tree construction rules follow those of standard CTL model checking, with one key modification: instead of individual APs, the leaves of the syntax tree correspond to entire atomic formulas. This is because atomic formulas define a single domain of validity and are not decomposed further for separate evaluation.

The inner nodes of the syntax tree represent subformulas of the formula being evaluated. Importantly, only state and atomic formulas are considered complete CTL formulas. Path formulas alone are incomplete in CTL. They must always be combined with quantifiers to form complete formulas.

5.3 Satisfaction Degree of Operators

In the following, we define a modified semantics for CTL operators, denoted by $[\![\cdot]\!]$, which propagates the satisfaction degree. As in standard CTL, we distinguish between state formulas and path formulas. We operate over a finite state space, where paths are infinite sequences of states. Hence, set operations such as infimum and supremum always yield values within the set. We use a notation $\mathsf{Paths}(s)$ to denote a set of all infinite paths originating from a given state s.

$$
\begin{aligned}
[\![\Phi_a]\!](s) &= \delta^*(s, \Phi_a) \\
[\![\Phi_s \wedge \Psi_s]\!](s) &= \min\left\{[\![\Phi_s]\!](s),\ [\![\Psi_s]\!](s)\right\} \\
[\![\Phi_s \vee \Psi_s]\!](s) &= \max\left\{[\![\Phi_s]\!](s),\ [\![\Psi_s]\!](s)\right\} \\
[\![\mathsf{E}\ \Phi_p]\!](s) &= \sup_{\pi \in \mathsf{Paths(s)}} [\![\Phi_p]\!](\pi) \\
[\![\mathsf{A}\ \Phi_p]\!](s) &= \inf_{\pi \in \mathsf{Paths(s)}} [\![\Phi_p]\!](\pi)
\end{aligned}
$$

For a path $\pi = s_1, s_2, \ldots$, the path formulas are defined as:

$$
\begin{aligned}
[\![\mathsf{X}\ \Phi_s]\!](\pi) &= [\![\Phi_s]\!](s_1) \\
[\![\mathsf{F}\ \Phi_s]\!](\pi) &= \sup_{j \geq 1}\ [\![\Phi_s]\!](s_j) \\
[\![\mathsf{G}\ \Phi_s]\!](\pi) &= \inf_{j \geq 1}\ [\![\Phi_s]\!](s_j) \\
[\![\Phi_s\ \mathsf{U}\ \Psi_s]\!](\pi) &= \sup_{j \geq 1}\ \min\left\{[\![\Psi_s]\!](s_j),\ \inf_{0 \leq i < j}\ [\![\Phi_s]\!](s_i)\right\} \\
[\![\Phi_s\ \mathsf{W}\ \Psi_s]\!](\pi) &= \max\left\{[\![\Phi_s\ \mathsf{U}\ \Psi_s]\!](\pi),\ [\![\mathsf{G}\ \Phi_s]\!](\pi)\right\}
\end{aligned}
$$

5.4 Quantitative Labeling Function

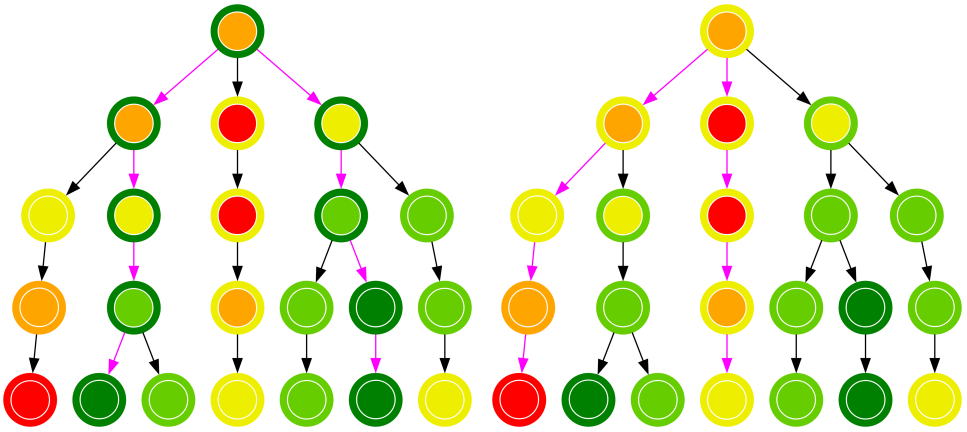
Finally, a total quantitative labeling function $\mathcal{Q} : \mathcal{S} \times \mathcal{L} \to [-1; 1]$ for each state and state formula is defined in the following way:

$$\mathcal{Q}(s, \Phi_s) = [\![\Phi_s]\!](s)$$

In the following, we discuss the interpretation of the quantitative labeling function $\mathcal{Q}$ for individual temporal operators at a given state s. For some operators, we also provide visual demonstrations to illustrate the interpretation:

- $\mathsf{EX}\ \Phi_s$: The satisfaction degree of Φ_s in the next state will be at most $\mathcal{Q}(s, \mathsf{EX}\ \Phi_s)$ regardless of which next state the model moves to.
- $\mathsf{AX}\ \Phi_s$: The satisfaction degree of Φ_s in the next state will be at least $\mathcal{Q}(s, \mathsf{AX}\ \Phi_s)$ regardless of which next state the model moves to.
- $\mathsf{EF}\ \Phi_s$: The satisfaction degree of Φ_s never exceeds $\mathcal{Q}(s, \mathsf{EF}\ \Phi_s)$ regardless of which trajectory from state s is executed. Alternatively, a state with satisfaction degree $\mathcal{Q}(s, \mathsf{EF}\ \Phi_s)$ is reachable from s.
- $\mathsf{AF}\ \Phi_s$: Regardless of the executed trajectory from s, it is certain that at some point the model reaches a state with satisfaction degree at least $\mathcal{Q}(s, \mathsf{AF}\ \Phi_s)$.

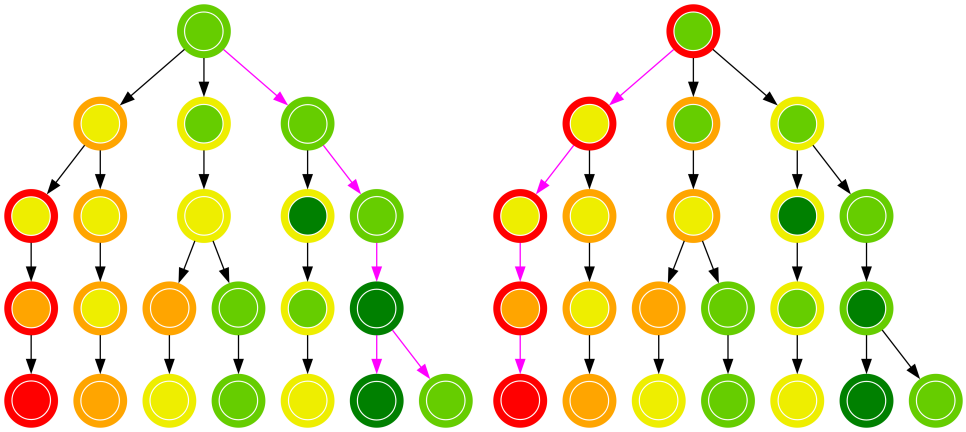
- EG Φ_s : Regardless of which trajectory from state s is executed, the satisfaction degree of Φ_s never permanently rises above $\mathcal{Q}(s, \mathsf{EG}\ \Phi_s)$. Alternatively, a trajectory maintaining the satisfaction degree of Φ_s at least $\mathcal{Q}(s, \mathsf{EG}\ \Phi_s)$ can be executed.
- AG Φ_s : The satisfaction degree of Φ_s is continuously maintained above $\mathcal{Q}(s, \mathsf{AG}\ \Phi_s)$ regardless of which trajectory from state s is executed. Alternatively, it is certain that the satisfaction degree of Φ_s never drops below $\mathcal{Q}(s, \mathsf{AG}\ \Phi_s)$.



(a) EF Φ_s **(b)** AF Φ_s

Figure 5.1: Unfolding of possible trajectories from the root state. Each node is divided into two parts: the inner color captures the satisfaction degree of Φ_s, and the outer ring shows the propagated satisfaction degree of the temporal formula. Colors range from dark green (high satisfaction) to red (low satisfaction), with yellow indicating marginal satisfaction. Magenta transitions highlight the trajectories that determine the final satisfaction degree in the root. **(a)** The root's satisfaction degree is dark green since the best trajectories eventually reach a dark green state. **(b)** The root's satisfaction degree is yellow, as the middle and left-most trajectories eventually reach only a yellow state.

Figure 5.3 illustrates how the satisfaction degree is evaluated and propagated for the until operator under both quantifiers, highlighting key differences in interpretation. To interpret an until formula with operands Φ_s and Ψ_s in a given state s, we examine all *prefixes* that start



(a) EG Φ_s **(b)** AG Φ_s

Figure 5.2: Unfolding of possible trajectories from the root state. Each node is divided into two parts: the inner color captures the satisfaction degree of Φ_s, and the outer ring shows the propagated satisfaction degree of the temporal formula. Colors range from dark green (high satisfaction) to red (low satisfaction), with yellow indicating marginal satisfaction. Magenta transitions highlight the trajectories that determine the final satisfaction degree in the root. **(a)** The root's satisfaction degree is light green since the rightmost trajectories maintain this level. **(b)** The root's satisfaction degree is red, as the left-most trajectory reaches a red state.

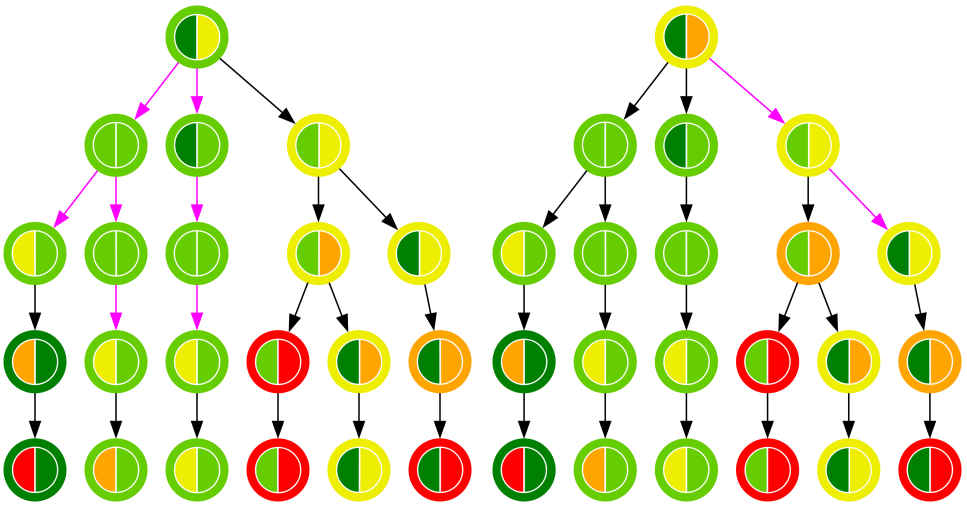
from s. A prefix is defined as any non-empty sequence of adjacent states in the STG. For a prefix of length n, the satisfaction degree with respect to the until formula is computed as the minimum of:

- the satisfaction degrees of Φ_s over the first $n-1$ states, and
- the satisfaction degree of Ψ_s in the n-th (last) state of the prefix.

Using this definition, the semantics of the until operators are interpreted as follows:

- $\mathsf{E}\,\Phi_s\,\mathsf{U}\,\Psi_s$: Regardless of which trajectory from state s the model executes, the satisfaction degree of any prefix along this trajectory maintains the satisfaction degree at most $\mathcal{Q}(s, \mathsf{E}\,\Phi_s\,\mathsf{U}\,\Psi_s)$. Alternatively, a trajectory having a prefix that maintains the satisfaction degree of $\mathcal{Q}(s, \mathsf{E}\,\Phi_s\,\mathsf{U}\,\Psi_s)$ can be executed.

- A Φ_s U Ψ_s : Regardless of which trajectory from state s is executed, some prefix of such trajectory maintains the satisfaction degree of $\mathcal{Q}(s, \mathsf{A}\ \Phi_s\ \mathsf{U}\ \Psi_s)$.



(a) E Φ_s U Ψ_s **(b)** A Φ_s U Ψ_s

Figure 5.3: Unfolding of possible trajectories from the root state. Each node is divided into three parts: the left color captures the satisfaction degree of Φ_s, and the right color the satisfaction degree of Ψ_s. The outer ring captures the propagated satisfaction degree of the temporal formula. Colors range from dark green (high satisfaction) to red (low satisfaction), with yellow indicating marginal satisfaction. Magenta transitions capture the best prefixes of all trajectories, based on which the resulting value is determined. **(a)** The root's satisfaction degree is light green since all the highlighted prefixes maintain a light-green value. **(b)** The root's satisfaction degree is yellow, as the best prefix (of length 1) of the rightmost branch maintains yellow satisfaction degree (the prefix of length 2 has only orange satisfaction degree for Ψ_s).

The weak until operators optimize the appropriate quantifier for each state individually between the global and until operators. Their interpretation is, therefore, based on the interpretation of the aforementioned operators.

6 Algorithm Design

In the previous chapters, we introduced the satisfaction degree function for atomic formulas. We defined a restricted syntax of CTL with modified semantics that allows the propagation of real-valued satisfaction through temporal and logical operators.

The formal definitions provide a theoretical foundation. However, their direct application can lead to computational challenges. In this chapter, we propose a prototype algorithm design to address these challenges. Since it is a prototype, some parts of the computation are optimized for efficiency, while others are implemented purely in a functional way. Rather than focusing on low-level implementation, we emphasize the logical structure of the algorithm, using abstract notation and pseudocode to explain the main ideas, workflow, and use of suitable data structures. We conclude the chapter with a discussion of the algorithm's complexity.

6.1 The Main Procedure Overview

In this section, we describe the workflow of the main procedure of the algorithm. The workflow is captured by the Algorithm 1. The algorithm takes three inputs: encoded MvGRN $\mathcal{R}^{(n)}$ that reflects the defined formal structure, CTL state formula Φ_s to be evaluated, and a set of initial states I. We discuss the correspondence between the formal structure defined in Section 1.1 and the input MvGRN encoding later in Section 7.2.

First, the formula Φ_s undergoes a transformation into an equivalent positive form, denoted Φ_s^+, such that all the negations (allowed only within atomic formulas) are systematically eliminated using De Morgan's laws and state space complements. Then, the set of subformulas of Φ_s^+ is determined, ensuring a bottom-up ordering that follows the structure of the syntax tree. Next, the STG corresponding to the input network is constructed using asynchronous semantics. This ensures that transitions reflect independent changes in gene activities. Additionally, the construction partially implements the fairness property by excluding transient self-loops. This procedure is described in Algorithm 3. Based on the STG, a corresponding Kripke structure $\mathcal{K}$

is instantiated, incorporating the defined initial state constraints. The core EvaluateSubformulae procedure then computes a quantitative labeling function $\mathcal{Q}$ that assigns satisfaction degrees to all state-subformula tuples. Finally, the algorithm outputs the minimum, maximum (along with a single corresponding state for each), and average satisfaction values, providing an overview of the formula satisfaction or violation across initial states.

Algorithm 1: The main procedure

Function Main($\mathcal{R}^{(n)}(\mathcal{V}, \mu, \mathcal{E}, \rho, \mathcal{F}), \Phi_s, I$)
- $\Phi_s^+ \leftarrow$ EliminateNegation(Φ_s)
- $Sub(\Phi_s^+) \leftarrow \left(\phi_1 \dots \phi_n \mid \forall i, j : \phi_i \in Sub(\phi_j) \implies i \leq j \right)$
- **create** state-transition graph $STG(\mathcal{R}^{(n)})$
- **create** Kripke Structure $\mathcal{K}(STG, I, \mathcal{Q})$
- $\mathcal{Q} \leftarrow$ EvaluateSubformulae($\mathcal{K}, Sub(\Phi_s^+)$)
- FormatResult($\mathcal{Q}, I, \Phi_s^+$)

Algorithm 2 describes a structured approach for the computation of quantitative labeling function $\mathcal{Q}$. The order of evaluation of subformulas guarantees that when a formula is evaluated, all its subformulas have already been processed, meaning their satisfaction degrees are already computed. Each subformula is evaluated over the Kripke structure $\mathcal{K}$, gradually building up satisfaction degrees for more complex formulas based on the results of their subformulas.

Algorithm 2: Evaluation of subformulas of Φ_s^+

Function EvaluateSubFormulae($\mathcal{K}(S, T, I, \mathcal{Q}), Sub(\Phi_s^+)$)
- **foreach** $\phi \in Sub(\Phi_s^+)$ **do**
  - Evaluate(ϕ) over $\mathcal{K}$
- **return** $\mathcal{Q}$

Algorithm 3 outlines the procedure for constructing the STG. The function iterates over all states and computes successors by applying each update function $\mathcal{F}$ individually to the variables, explicitly exclud-

ing the original state from the successor set. A self-loop is added only if no successor is found to ensure that the STG contains infinite paths.

The STG construction algorithm partially implements the fairness property by excluding transient self-loops, ensuring that states having transition to another state do not have self-loops. This restricts the graph on meaningful transitions, preventing the system from being stuck in transient states and not reaching attractors. Despite this restriction, multi-state transient cycles can still occur in STG. However, such cycles are often intentional and reflect meaningful system behavior.

Algorithm 3: Construction of STG from MvGRN $\mathcal{R}^{(n)}$

Function CreateSTG($\mathcal{R}^{(n)}(\mathcal{V}, \mu, \mathcal{E}, \rho, \mathcal{F})$)
- $S = \mathcal{S}(\mathcal{R}^{(n)}), T = \emptyset$
- **foreach** $s \in S$ **do**
  - $Succ(s) \leftarrow \{ f_i(s) \mid i \in 1 \dots n \} \setminus \{ s \}$
  - **if** $Succ(s) = \emptyset$ **then**
    - $T \leftarrow T \cup \{ ss \}$
  - **else**
    - $T \leftarrow T \cup \{ ss' \mid s' \in Succ(s) \}$
- **return** $STG(S, T)$

6.2 Atomic Formula Evaluation

As discussed before, atomic formulas are constructed using conjunctions and disjunctions of atomic propositions, with negations eliminated. The evaluation consists of the computation of each state's satisfaction degree function δ^*. The value is determined by the state's relative depth within (or distance from) the DoV of the formula. The evaluation procedure, outlined in Algorithm 4, follows a systematic approach to ensure efficient computation of minimal distances.

First, the DoV Ω_{Φ_a} is determined, identifying the subset of states in which the atomic formula holds. This is done purely in an enumerative way. Subsequently, the complement of this set $\overline{\Omega}_{\Phi_a}$ is computed. The

borders of both - the DoV and its complement are identified simultaneously. Namely, the state is considered a border state if it has some of its neighbors in the complementary set. These neighbors are likewise marked as border states of the complementary set.

Next, two reference values for normalization are computed: the maximum depth among states inside the DoV $wd_{max}(\Omega_{\Phi_a})$ measured with respect to the border of the complement, and the maximum distance among states outside the DoV $wd_{max}(\overline{\Omega}_{\Phi_a})$ measured with respect to the border of the DoV.

Following this, a weighted distance to the complementary border $wd_{min}(s, \partial\Omega)$ is calculated for each state. This value represents the minimal distance to any state in the complementary set, which approximates the ability of the system executed from the given state to escape from the state's starting set (i.e., to achieve the opposite value in a Boolean sense). Next, the signed distance sd is calculated, which enriches wd_{min} with the information of the state assignment in DoV.

Finally, the signed distance is normalized into satisfaction degree δ^* within the interval $[-1; 1]$, using the previously computed extreme values as bounds.

All the concepts mentioned are formally defined in Chapter 4.

Algorithm 4: Evaluation of Atomic Formula Φ_a over $\mathcal{K}(S, T, I, \mathcal{Q})$

```
1  Function Evaluate(Φ_a)
2      Ω_{Φ_a} ← ComputeDoV(Φ_a, μ)
3      Ω̄_{Φ_a} ← S \ Ω_{Φ_a}
4      ∂Ω_{Φ_a}, ∂Ω̄_{Φ_a} ← GetBorderStates(Ω_{Φ_a}, μ)
5      compute wd_max(Ω_{Φ_a}) and wd_max(Ω̄_{Φ_a})
6      foreach s ∈ S do
7          ∂Ω ← ∂Ω̄_{Φ_a} if s ∈ Ω_{Φ_a} else ∂Ω_{Φ_a}
8          compute wd_min(s, ∂Ω)
9          compute sd(s, Φ_a)
10         Q(s, Φ_a) ← δ*(s, Φ_a)
```

In the following, we describe the individual steps of the computation of satisfaction degree for atomic formulas.

6.2.1 Computing the Domain of Validity

The DoV is constructed by iterating over all states and determining whether they satisfy the atomic formula based on given constraints. This is done enumeratively by generating the set of satisfying states for atomic propositions, computing intersections for conjunctions, and unions for disjunctions, as described in Algorithm 5. Obviously, the maximum activity values μ help bound the state space.

Unlike intersections, unions can produce nonconvex sets, making it impossible to represent the DoV as a list of admissible values for individual domains. Storing multiple convex components is also impractical, as overlaps between them could disrupt the correct identification of border states. These facts prevent a more compact representation, which leads to DoVs being stored enumeratively. Sets are used to optimize merging operations, allowing for efficient membership checks and a combination of subspaces.

Algorithm 5: Computation of Domain of Validity of formula Φ_a

```
1  Function ComputeDoV(Φ_a, μ)
2      switch Φ_a do
3          case φ ∧ ψ do
4              return ComputeDoV(φ, μ) ∩ ComputeDoV(ψ, μ);
5          case φ ∨ ψ do
6              return ComputeDoV(φ, μ) ∪ ComputeDoV(ψ, μ);
7          case v_i ≤ k do
8              return { s | s_i ≤ k } ∩ ∏_{i=1}^{n} [0; μ_i]
9          case v_i ≥ k do
10             return { s | s_i ≥ k } ∩ ∏_{i=1}^{n} [0; μ_i]
```

6.2.2 Identification of Border States

The border of the DoV consists of states within the DoV that have at least one neighbor outside it. To identify these border states, each state is examined by checking its immediate neighborhood, denoted

as $\mathcal{N}_1$, based on Manhattan distance one. If any neighboring state lies outside the DoV, the current state is marked as a border state. To improve efficiency, the algorithm computes both borders at once. If a state is evaluated as a border state of Ω_{Φ_a} based on a nonempty set of neighbors N_{out}, these neighbors are symmetrically boundary states of $\overline{\Omega}_{\Phi_a}$. The resulting set $\partial\Omega_{\Phi_a}$ and $\partial\overline{\Omega}_{\Phi_a}$ explicitly list all identified border states. The described procedure is captured in Algorithm 6.

Algorithm 6: Identification of border states

```
1 Function GetBorderStates(Ω_{Φ_a}, μ)
2     ∂Ω_{Φ_a} ← ∅
3     ∂Ω̄_{Φ_a} ← ∅
4     foreach s ∈ Ω_{Φ_a} do
5         N_out ← { n | n ∈ N_1(s, μ) ∧ n ∉ Ω_{Φ_a} }
6         if N_out ≠ ∅ then
7             ∂Ω_{Φ_a} ← ∂Ω_{Φ_a} ∪ { n }
8             ∂Ω̄_{Φ_a} ← ∂Ω̄_{Φ_a} ∪ N_out
9     return ∂Ω_{Φ_a}, ∂Ω̄_{Φ_a}
```

6.2.3 Computation of Weighted Distance

The weighted distance between a state s and the complementary set border $\partial\Omega$ is computed using Dijkstra's algorithm. The procedure is outlined in Algorithm 7. The state space is treated as a weighted graph, where each state represents a vertex, and edges connect neighboring states in the state space. The cost of the edge between neighboring states is equal to the weight of a particular dimension w_i, which is calculated as an inversion of the corresponding dimension's size.

A min-heap priority queue is used to efficiently extract the state with the smallest accumulated distance. The algorithm initializes the queue with the starting state s at distance zero and iteratively expands the closest unvisited state. Each expansion considers all neighboring states within the Manhattan neighborhood $\mathcal{N}_1$, updating their distances accordingly. The neighbors are yielded in ascending order with respect to the offset from the currently explored state.

A *visited* set ensures that each state is processed only once, preventing redundant computations and improving efficiency. The algorithm terminates when the first border state is extracted from the queue. Since Dijkstra's algorithm guarantees that a state is dequeued only when its distance is finalized, this ensures that the first dequeued border state is the one that has the shortest weighted distance to the initial state s. If no border state is reachable, the function returns ∞. This should happen only if the DoV covers the entire state space.

Algorithm 7: Weighted distance between state s and border $\partial\Omega$

```
1  Function wd_min(s, ∂Ω)
2      w_i ← 1/(μ_i+1), ∀i ∈ {1...n}
3      q ← ∅
4      q.DecreasePriority(s, 0)
5      visited ← ∅
6      while q ≠ ∅ do
7          s_curr, d_curr ← q.ExtractMin()
8          if s_curr ∈ ∂Ω then
9              return d_curr
10         visited ← visited ∪ {s_curr}
11         foreach (s_n, Δ_n) ∈ N_1(s_curr, μ, w, visited) do
12             d_n ← d_curr + Δ_n
13             q.DecreasePriority(s_n, d_n)
14     return ∞
```

6.2.4 Computation of Maximal Depth

The maximal weighted depth of a state within set Ω with respect to the complementary set border $\partial\overline{\Omega}$ is computed using a multi-source Dijkstra's algorithm, as outlined in Algorithm 8. The state space is again modeled as a weighted graph, where states correspond to vertices, and edges connect neighboring states along the individual dimensions, weighted by inversion of dimensions' sizes.

The algorithm initializes the priority queue with all complementary border states $\partial\overline{\Omega}$ having a distance equal to zero. Distances to all

other states are initially set to ∞. The queue is then processed iteratively, always expanding the state with the smallest current distance. Each expansion considers neighbors within the Manhattan neighborhood $\mathcal{N}_1$, yielding them in an ascending order based on the offset from the currently explored state.

Only states within Ω are processed, ensuring the search remains confined to the valid region. The priority queue ensures that states are visited in increasing distance order, thereby accelerating convergence.

After all states in Ω are processed, meaning that the queue is empty, the algorithm determines the maximal weighted distance by selecting the largest computed distance. This value represents the distance of the deepest state from Ω from the complementary set border in terms of weighted distance.

Algorithm 8: Computation of maximal depth among states in Ω

```
1  Function wd_max(Ω)
2      w_i ← 1/(μ_i+1), ∀i ∈ {1...n}
3      D ← { s ↦ ∞ | s ∈ Ω }
4      q ← ∅
5      foreach s ∈ ∂Ω̄ do
6          D(s) ← 0
7          q.DecreasePriority(s, 0)
8      while q ≠ ∅ do
9          s_curr, d_curr ← q.ExtractMin()
10         foreach (s_n, Δ_n) ∈ N̂_1(s_curr, μ, w) do
11             if s_n ∉ Ω then
12                 continue
13             d_n ← d_curr + Δ_n
14             if D(s_n) > d_n then
15                 D(s_n) ← d_n
16                 q.DecreasePriority(s_n, d_n)
17     return max { D(s) | s ∈ Ω }
```

6.3 Logical Operators

Logical operators propagate the satisfaction degree based on the satisfaction degrees corresponding to their operands.

The conjunction $\wedge$ traditionally requires both subformulas to hold. In our approach, this is reflected by propagating the minimal satisfaction degree of its operands, as outlined in Algorithm 9.

Algorithm 9: Evaluate Conjunction over $\mathcal{K}(S, T, I, \mathcal{Q})$

```
1 Function Evaluate(Φ_s ∧ Ψ_s)
2     foreach s ∈ S do
3         Q(s, Φ_s ∧ Ψ_s) ← min{ Q(s, Φ_s), Q(s, Ψ_s) }
```

Conversely, the disjunction $\vee$ requires at least one subformula to hold, leading to the propagation of the maximum satisfaction degree of its operands, as described in Algorithm 10.

Algorithm 10: Evaluate Disjunction over $\mathcal{K}(S, T, I, \mathcal{Q})$

```
1 Function Evaluate(Φ_s ∨ Ψ_s)
2     foreach s ∈ S do
3         Q(s, Φ_s ∨ Ψ_s) ← max{ Q(s, Φ_s), Q(s, Ψ_s) }
```

6.4 Next Operators

The classical interpretation of the AX operator requires that the subformula Φ_s holds in all immediate successor states. In our adaptation (outlined in Algorithm 11), this is reflected by assigning the minimum satisfaction degree among its successors to each state, capturing the worst-case scenario in the next step and thus preserving the universal nature of AX.

Conversely, the EX operator in classical CTL requires that Φ_s holds in at least one successor. Our approach preserves this existential interpretation by assigning the maximum satisfaction degree among the

successors to each state (outlined in Algorithm 12), thereby identifying the most favorable outcome reachable in one step.

Algorithm 11: Evaluate AX over $\mathcal{K}(S, T, I, \mathcal{Q})$

```
1 Function Evaluate(AX Φ_s)
2     foreach s ∈ S do
3         Succ(s) ← { s' | (s, s') ∈ T }
4         Q(s, AX Φ_s) ← min { Q(s', Φ_s) | s' ∈ Succ(s) }
```

Algorithm 12: Evaluate EX over $\mathcal{K}(S, T, I, \mathcal{Q})$

```
1 Function Evaluate(EX Φ_s)
2     foreach s ∈ S do
3         Succ(s) ← { s' | (s, s') ∈ T }
4         Q(s, EX Φ) ← max { Q(s', Φ_s) | s' ∈ Succ(s) }
```

6.5 Future Operators

Both future operators, EF and AF, are evaluated in a given state s based on the satisfaction degrees of their operand observed along the individual paths starting from s. Specifically, each path is characterized by the maximum satisfaction degree of the operand among states encountered on the path. However, the operators differ in how they aggregate these path-level values: EF (outlined in Algorithm 13) reflects the best-case scenario by selecting the maximum of these best values across all paths. In contrast, AF (outlined in Algorithm 14) reflects the worst-case scenario by taking the minimum among them.

Both future operators' satisfaction degree in each state is initially copied from the precomputed satisfaction degree of their operand Φ_s, establishing the lower bound. This value can be only improved by propagating the satisfaction degree of states with higher values for Φ_s, but it cannot decrease further. The states are then enqueued with initial priorities determined by the satisfaction degree of their successors,

not their own satisfaction degree. This is important because, during propagation, a state's satisfaction degree is updated based on its successors' values. As a result, after the two initial cycles, each state's priority reflects the highest satisfaction degree among its successors. Since the priority queue uses a maximum heap structure, states are always extracted in descending order of priority. This ensures that paths with the most significant impact are processed first.

Algorithm 13: Evaluate EF over $\mathcal{K}(S, T, I, \mathcal{Q})$

```
1  Function Evaluate(EF Φ_s)
2      q ← ∅
3      foreach s ∈ S do
4          Q(s, EF Φ_s) ← Q(s, Φ_s)
5      foreach s ∈ S do
6          Succs(s) ← { s' | (s, s') ∈ T }
7          q.IncreasePriority(s, max_{s'∈Succs(s)} Q(s', EF Φ_s))
8      while q ≠ ∅ do
9          s, _ ← q.ExtractMax()
10         Succ(s) ← { s' | (s, s') ∈ T }
11         v_max ← max{ Q(s', Φ_s) | s' ∈ Succ(s)}
12         if v_max > Q(s, EF Φ_s) then
13             Q(s, EF Φ_s) ← v_max
14             Pred(s) ← { s' | (s', s) ∈ T }
15             foreach s' ∈ Pred(s) do
16                 q.IncreasePriority(s', v_max)
```

In the propagation phase, algorithms gradually dequeue states from the priority queue until it is empty. At this point, the behavior of EF and AF diverges. For EF, the algorithm attempts to update the dequeued state's satisfaction degree to reflect the strongest path starting from it. This means the state's value is updated if the maximum satisfaction degree among its successors is greater than its current value. In contrast, AF aggregates the minimum satisfaction degree of its successors and updates the state's value only if the aggregated value is higher than the current one. This reflects the best reachable

satisfaction degree along the worst path that can be executed from the given state.

To ensure that the satisfaction degree is propagated through the entire state space structure, all its predecessors are re-enqueued with the priority equal to the updated value whenever a given state's satisfaction degree is updated. However, due to the initial prioritization of the states, when the value for a given state is once updated, it should not be changed again. This causes each state to be re-enqueued at most once, which significantly improves the efficiency.

Algorithm 14: Evaluate AF over $\mathcal{K}(S, T, I, \mathcal{Q})$

```
1  Function Evaluate(AF Φ_s)
2      q ← ∅
3      foreach s ∈ S do
4          Q(s, AF Φ_s) ← Q(s, Φ_s)
5      foreach s ∈ S do
6          Succs(s) ← { s' | (s, s') ∈ T }
7          q.IncreasePriority(s, max_{s'∈Succs(s)} Q(s', AF Φ_s))
8      while q ≠ ∅ do
9          s, _ ← q.ExtractMax()
10         Succ(s) ← { s' | (s, s') ∈ T }
11         v_min ← min{ Q(s', Φ_s) | s' ∈ Succ(s)}
12         if v_min > Q(s, AF Φ_s) then
13             Q(s, AF Φ_s) ← v_min
14             Pred(s) ← { s' | (s', s) ∈ T }
15             foreach s' ∈ Pred(s) do
16                 q.IncreasePriority(s', v_min)
```

Thus, EF provides an upper bound, capturing the state with the best possible satisfaction degree to be reached from a given state while also indicating that no state with a higher satisfaction degree is reachable from the given state. On the other hand, AF provides a lower bound, ensuring that, no matter the path executed from a given state, a state with at least the computed satisfaction degree will eventually be reached.

6.6 Globally Operators

Similarly to future operators, both globally operators EG and AG are evaluated in a given state s based on the satisfaction degrees of their operand observed along the individual paths starting from s. However, each path is characterized by the minimum satisfaction degree of operand among states encountered on the path. Similarly, the two operators differ in how they aggregate these path-level values: EG (outlined in Algorithm 15) reflects the best-case scenario by selecting the maximum of these worst values across all paths. In contrast, AG (outlined in Algorithm 16) reflects the worst-case scenario by taking the minimum over them.

Both operators' satisfaction degree in each state is initially copied from the precomputed satisfaction degree of their operand Φ_s, establishing the upper bound. This value can only decrease by propagating the satisfaction degree of states with lower values for Φ_s, but it cannot increase further. The states are then enqueued with initial priorities determined by the satisfaction degree of their successors, not their own satisfaction degree. This is important because, during propagation, a state's satisfaction degree is updated based on its successors' values. As a result, after the two initial cycles, each state's priority reflects the lowest satisfaction degree among its successors. Since the priority queue uses a minimum heap structure, states are always extracted in ascending order of priority, ensuring that paths with the most significant impact (in this case, paths containing states with low satisfaction degrees for operand) are processed first.

In the propagation phase, algorithms gradually dequeue states from the priority queue until it is empty. At this point, the behavior of EG and AG diverges. For EG, the algorithm attempts to update a state's satisfaction degree to reflect the strongest path starting from it. This means the state's value is updated if the maximum satisfaction degree among its successors is lower than its current value. This reflects the worst reachable satisfaction degree along the best path that can be executed from the given state. In contrast, AG aggregates the minimum satisfaction degree of its successors and updates the state's value only if the value is lower than the current one. The resulting value in a given state reflects the very worst satisfaction degree of operand among all the reachable states from the given state.

Algorithm 15: Evaluate EG over $\mathcal{K}(S, T, I, \mathcal{Q})$

```
1  Function Evaluate(EG Φ_s)
2      q ← ∅
3      foreach s ∈ S do
4          Q(s, EG Φ_s) ← Q(s, Φ_s)
5      foreach s ∈ S do
6          Succs(s) ← { s' | (s, s') ∈ T }
7          q.DecreasePriority(s, min_{s'∈Succs(s)} Q(s', EG Φ_s))
8      while q ≠ ∅ do
9          s, _ ← q.ExtractMin()
10         Succ(s) ← { s' | (s, s') ∈ T }
11         v_max ← max{ Q(s', Φ_s) | s' ∈ Succ(s) }
12         if v_max < Q(s, EG Φ_s) then
13             Q(s, EG Φ_s) ← v_min
14             Pred(s) ← { s' | (s', s) ∈ T }
15             foreach s' ∈ Pred(s) do
16                 q.DecreasePriority(s', v_max)
```

Similarly, to ensure that the satisfaction degree is propagated through the entire state space structure, all of its predecessors are re-queued whenever a given state's satisfaction degree is updated. However, due to the initial prioritization of the states, when the value for a given state is once updated, it should not be changed again. This causes each state to be re-enqueued at most once, which significantly improves the efficiency.

Thus, EG provides an upper bound, ensuring that in the best possible scenario, the executed path will sustain the computed satisfaction degree throughout. Conversely, AG provides a lower bound, guaranteeing that, regardless of the path executed from a given state, states along it will always maintain at least the computed satisfaction degree for the operand.

Algorithm 16: Evaluate AG over $\mathcal{K}(S, T, I, \mathcal{Q})$

```
1  Function Evaluate(AG Φ_s)
2      q ← ∅
3      foreach s ∈ S do
4          Q(s, AG Φ_s) ← Q(s, Φ_s)
5      foreach s ∈ S do
6          Succs(s) ← { s' | (s, s') ∈ T }
7          q.DecreasePriority(s, min_{s'∈Succs(s)} Q(s', AG Φ_s))
8      while q ≠ ∅ do
9          s, _ ← q.ExtractMin()
10         Succ(s) ← { s' | (s, s') ∈ T }
11         v_min ← min{ Q(s', Φ_s) | s' ∈ Succ(s) }
12         if v_min < Q(s, AG Φ_s) then
13             Q(s, AG Φ_s) ← v_min
14             Pred(s) ← { s' | (s', s) ∈ T }
15             foreach s' ∈ Pred(s) do
16                 q.DecreasePriority(s', v_min)
```

6.7 Until Operators

Until operators EU and AU evaluate the satisfaction degree of a state based on the satisfaction degree of the best prefix along each path starting from such state. The satisfaction degree with respect to given until formula associated with a prefix having n states is equal to the minimum of the satisfaction degree of Φ_s among the first $n-1$ states and Ψ_s in the n^{th} state of the prefix. Similarly, the two operators differ in how they aggregate these path-level values: EU (Algorithm 17) reflects the best-case scenario by selecting the maximum of these worst values across all paths, while AU (Algorithm 18) reflects the worst-case scenario by taking the minimum over them.

Both operators' satisfaction degree in each state is initially copied from the precomputed satisfaction degree of their right operand Ψ_s, reflecting the prefix of size one, thus establishing the lower bound. This value can only increase by propagating the satisfaction degree

of longer prefixes. The states are then enqueued with initial priorities determined by the maximal satisfaction degree of Ψ_s among their successors. This is important because, during propagation, a state's satisfaction degree is also updated based on its successors' values. As a result, after the two initial cycles, each state's priority reflects the lowest satisfaction degree among its successors. Since the priority queue uses a maximum heap structure, states are always extracted in ascending order of priority, ensuring that paths with the most significant impact are processed first.

Algorithm 17: Evaluate EU over $\mathcal{K}(S, T, I, \mathcal{Q})$

```
1  Function Evaluate(E Φ_s U Ψ_s)
2      q ← ∅
3      foreach s ∈ S do
4          Q(s, E Φ_s U Ψ_s) ← Q(s, Ψ_s)
5      foreach s ∈ S do
6          Succs(s) ← { s' | (s, s') ∈ T }
7          q.IncreasePriority(s, max_{s'∈Succs(s)} Q(s', Ψ_s))
8      while q ≠ ∅ do
9          s, _ ← q.ExtractMax()
10         Succ(s) ← { s' | (s, s') ∈ T }
11         u_max ← max{ Q(s', E Φ_s U Ψ_s) | s' ∈ Succ(s) }
12         v_ext ← min(Q(s, Φ_s), u_max)
13         if v_ext > Q(s, E Φ_s U Ψ_s) then
14             Q(s, E Φ_s U Ψ_s) ← v_ext
15             Pred(s) ← { s' | (s', s) ∈ T }
16             foreach s' ∈ Pred(s) do
17                 q.IncreasePriority(s', v_ext)
```

Once the queue is initialized, the core propagation step follows a common structure across both AU and EU, differing only in the way satisfaction degrees are updated. For AU, each state's satisfaction degree is updated to reflect the weakest prefix among the best prefixes achieved so far for each path. This is done by taking the minimum between the state's own Φ_s value and the weakest satisfaction degree

among its successors. For EU, the objective is to reflect the strongest best prefix found so far originating from a given state.

Algorithm 18: Evaluate AU over $\mathcal{K}(S, T, I, \mathcal{Q})$

```
1  Function Evaluate(A Φ_s U Ψ_s)
2      q ← ∅
3      foreach s ∈ S do
4          Q(s, A Φ_s U Ψ_s) ← Q(s, Ψ_s)
5      foreach s ∈ S do
6          Succs(s) ← { s' | (s, s') ∈ T }
7          q.IncreasePriority(s, max_{s'∈Succs(s)} Q(s', Ψ_s))
8      while q ≠ ∅ do
9          s, _ ← q.ExtractMax()
10         Succ(s) ← { s' | (s, s') ∈ T }
11         u_min ← min{ Q(s', A Φ_s U Ψ_s) | s' ∈ Succ(s) }
12         v_ext ← min(Q(s, Φ_s), u_min)
13         if v_ext > Q(s, A Φ_s U Ψ_s) then
14             Q(s, A Φ_s U Ψ_s) ← v_ext
15             Pred(s) ← { s' | (s', s) ∈ T }
16             foreach s' ∈ Pred(s) do
17                 q.IncreasePriority(s', v_ext)
```

Thus, EU provides an upper bound, ensuring that in the best-case scenario, the executed path contains some prefix that maintains the computed satisfaction degree. In contrast, AU provides a lower bound, guaranteeing that all paths originating from s contain a prefix that maintains at least the computed satisfaction degree.

6.8 Weak Until Operators

The weak until operators EW and AW differ from classic until operators in that they do not necessarily require the prefix to end with a state where the satisfaction degree of Ψ_s is accounted for. Instead, higher satisfaction can also be achieved if Φ_s maintains a high satisfaction

degree indefinitely along the path. This distinction is reflected in the evaluation process, where the satisfaction degree is determined by considering both possibilities.

For $\mathsf{E}\ \Phi_s\ \mathsf{W}\ \Psi_s$, the algorithm first evaluates $\mathsf{E}\ \Phi_s\ \mathsf{U}\ \Psi_s$ to determine the best satisfaction degree in case of prefixes with accounting satisfaction degrees of Φ_s and finally accounting Ψ_s. Then, it evaluates $\mathsf{EG}\ \Phi_s$, which accounts for Φ_s globally, i.e., infinite prefixes accounting only satisfaction degrees of Φ_s. Similarly to the universal case, the final satisfaction degree is computed as the maximum of these two values, selecting the better option.

Algorithm 19: Evaluate EW over $\mathcal{K}(S, T, I, \mathcal{Q})$

```
1 Function Evaluate(E Φ_s W Ψ_s)
2     Evaluate(E Φ_s U Ψ_s)
3     Evaluate(EG Φ_s)
4     foreach s ∈ S do
5         Q(s, E Φ_s W Ψ_s) ← max{ Q(s, E Φ_s U Ψ_s), Q(s, EG Φ_s) }
```

A similar approach applies to $\mathsf{A}\ \Phi_s\ \mathsf{W}\ \Psi_s$. Here, the evaluation combines the results of two formulas. First, the algorithm computes $\mathsf{A}\ \Phi_s\ \mathsf{U}\ \Psi_s$, capturing the satisfaction degree for finite prefixes with accounting satisfaction degrees of Φ_s and finally accounting Ψ_s. Then, it evaluates $\mathsf{AG}\ \Phi_s$, which accounts for a satisfaction degree of Φ_s globally, i.e., infinite prefixes accounting only for satisfaction degrees of Φ_s. Since weak until optimizes between the stated scenarios, the final value is determined as the maximum of these two satisfaction degrees, ensuring that the strongest applicable condition is chosen.

Algorithm 20: Evaluate AW over $\mathcal{K}(S, T, I, \mathcal{Q})$

```
1 Function Evaluate(A Φ_s W Ψ_s)
2     Evaluate(A Φ_s U Ψ_s)
3     Evaluate(AG Φ_s)
4     foreach s ∈ S do
5         Q(s, A Φ_s W Ψ_s) ← max{ Q(s, A Φ_s U Ψ_s), Q(s, AG Φ_s) }
```

By combining these two evaluations, the weak until operators ensure that the satisfaction degree optimizes between both cases - accounting Φ_s globally and accounting finite prefix of Φ_s with finally accounting Ψ_s. This guarantees that, in the best case, the computed satisfaction degree will be maintained by the executed path.

6.9 Complexity

This section discusses the computational complexity of the algorithm described in this chapter.

We omit the complexity analysis of preprocessing steps, including the parsing of the MvGRN structure, the CTL formula, the set of initial states, and the elimination of formula negation. These components are not dominant contributors to the overall computation time. Therefore, the following analysis focuses solely on the algorithm's core computational steps.

For this discussion, we introduce the following notation:

- $|\Phi_s|$: the total number of subformulas contained in the verified formula Φ_s. Notably, the total number of atomic formulas is also bounded by $|\Phi_s|$.
- $|\mathcal{V}|$: number of variables in the MvGRN,
- $|\mathcal{S}| = \prod_{i=1}^{n}(\mu_i + 1)$: number of states of STG, which is exponential in the number of variables,

In the following, we discuss the complexity of the algorithm's key steps to finally state its overall complexity.

In the worst-case scenario of global model checking, where all states in the model are considered initial, the generation of the initial state set is linear in the number of states: $\mathcal{O}(|\mathcal{S}|)$.

Constructing the STG requires exploring all possible asynchronous updates. Each state can have up to $|\mathcal{V}|$ successors since one variable is updated at a time. In the worst-case scenario, we assume a fully connected model where each variable regulates all others. Additionally, we assume that each regulation is defined over as many activity intervals as the size of the regulator's domain. Under these assumptions,

the number of contexts that must be evaluated for each successor can grow as large as the number of states, $|\mathcal{S}|$.

As a result, the worst-case time complexity of STG construction is:

$$\mathcal{O}(|\mathcal{S}|(|\mathcal{V}| + |\mathcal{S}|)).$$

However, in practice, MvGRNs tend to be sparse. Therefore, the number of regulatory contexts is typically polynomial in $|\mathcal{V}|$.

The most complex step of the algorithm is the calculation and subsequent propagation of the satisfaction degree through the individual subformulas. The complexity of calculating the satisfaction degree for atomic formulas fundamentally differs from the complexity of propagation of the resulting satisfaction degree through operators. Therefore, we define their complexities separately.

Calculating the satisfaction degree for all states with respect to a given atomic formula consists of the following steps:

- Computation of the DoV is performed in an enumerative manner, with an overall complexity of $\mathcal{O}(|\mathcal{S}||\mathcal{V}|)$. This is because each atomic formula can, in the worst case, impose constraints on all $|\mathcal{V}|$ variables. Consequently, determining whether a state belongs to the DoV requires $\mathcal{O}(|\mathcal{V}|)$ checks.

- Computation of the maximal depth for both the DoV and its complement is performed using a multi-source Dijkstra's algorithm. The underlying state-space graph contains $|\mathcal{S}|$ vertices and $|\mathcal{S}||\mathcal{V}|$ edges since each state has two neighbors per dimension and edges are considered in only one direction. The algorithm utilizes a heap-based priority queue, with each insertion having a complexity of $\mathcal{O}(\log |\mathcal{S}|)$. As a result, the overall time complexity of this computation is $\mathcal{O}(|\mathcal{S}||\mathcal{V}| \log |\mathcal{S}|)$.

- Computation of the distance to the boundary of the complementary set is carried out by running Dijkstra's algorithm separately for each state. Since the algorithm is executed $|\mathcal{S}|$ times over a graph with $|\mathcal{S}|$ vertices and $|\mathcal{S}||\mathcal{V}|$ edges, the overall time complexity is $\mathcal{O}(|\mathcal{S}|^2|\mathcal{V}| \log |\mathcal{S}|)$.

Assuming that the number of atomic subformulas is bounded by $|\Phi_s|$, the total complexity of satisfaction degree computation is:

$$\mathcal{O}\big(|\Phi_s||\mathcal{S}|^2|\mathcal{V}|\log|\mathcal{S}|\big)$$

Consider the following to analyze the complexity of satisfaction degree propagation through temporal operators. Each state can have up to $|\mathcal{V}|$ successors in the STG. The satisfaction degrees of successors are used to determine insertion priorities in a heap-based priority queue, where each insertion has a cost of $\mathcal{O}(\log|\mathcal{S}|)$. Thanks to algorithmic optimizations that minimize redundant operations, each state is reinserted at most once during the propagation phase. We also assume that basic operations, such as lookup, insertion, and access to a state's successors or predecessors, have time complexity of $\mathcal{O}(1)$.

Assuming that the number of temporal subformulas is bounded by $|\Phi_s|$, the satisfaction degree must be propagated for each subformula across all $|\mathcal{S}|$ states. For each state, this process may involve examining up to $|\mathcal{V}|$ successors and performing at most one reinsertion into the heap. This results in an overall complexity of:

$$\mathcal{O}\big(|\Phi_s||\mathcal{S}|\big(|\mathcal{V}|+\log|\mathcal{S}|\big)\big).$$

Taking into account all the operations performed by the algorithm, the overall time complexity is:

$$\mathcal{O}\big(|\Phi_s||\mathcal{S}|^2|\mathcal{V}|\log|\mathcal{S}|\big)$$

The complexity discussion reveals that the majority of computation time is spent on calculating the satisfaction degrees for atomic formulas, while the subsequent propagation through temporal operators contributes negligibly. Therefore, in the evaluation chapter, we also focus on defining the size of input instances for which the algorithm remains practically feasible.

7 Implementation

We implemented the algorithm described in the previous chapter in Python. The implementation related to this thesis is freely available on `https://github.com/xsimurka/CTL_sat_degree/tree/master_thesis`. The `master_thesis` branch contains the version that aligns with the contents of this thesis. The content of the repository is discussed in detail in Appendix A. This chapter describes how the procedures formally introduced earlier have been realized in code.

The algorithm can be executed from the command line as follows:

```
python main.py path/to/input.json
```

where `path/to/input.json` is the path to the input JSON file containing the model definition, CTL formula to be verified, and initial states' conditions.

7.1 Third-Party Libraries

We used several third-party Python libraries to support the implementation. To define an abstract class hierarchy for CTL formula components, we relied on the `abc` module. The `itertools` module enabled efficient construction of Cartesian products needed during state space and DoV generation. For representing the STG as a directed graph, we used the `DiGraph` class from the `networkx` library, which provided convenient access to state successors and predecessors. The `heapdict` library formed the basis for implementing minimum and maximum priority queues, essential for efficient STG traversal during the propagation of satisfaction degrees through CTL operators. Additionally, the `lark` parser [23] was used to define a custom grammar and transformer for parsing CTL formulas into syntax trees. We also employed `pydot` to visualize STGs, which helped us set up the correct dynamics of evaluated models.

7.2 Input Specification

The input of the framework is a single JSON file having three mandatory fields: `network` encoding a Multivalued Gene Regulatory Network, `formula` defining a CTL formula to be checked, and `init_states` capturing the set of initial states of Kripke structure. The file must respect the following structure and contain all the mentioned fields. The following sections correspond to the three mandatory fields mentioned above.

7.2.1 Network Definition

The value of this field is an object that defines a multivalued GRN $\mathcal{R}^{(n)}(\mathcal{V}, \mu, \mathcal{E}, \rho, \mathcal{F})$. We explicitly state the correspondence between JSON fields and the formal definition in Section 1.1. In the following, we outline its expected structure:

`variables` An object mapping variable names to their maximum activity levels. Names must be unique, defining a set of variables $\mathcal{V}$. Values must be strictly positive integers, representing maximum activity levels μ.

`regulations` An array of regulations $\mathcal{E}$, defining how the individual genes influence each other. Each regulation is an object requiring the following fields:

- **`target`** A variable name that is the target of the regulation.
- **`regulators`** An array of regulator objects influencing `target`. Each element is an object with:
  - **`variable`** An existing name that regulates the target. Formally, $(\texttt{variable}, \texttt{target}) \in \mathcal{E}$.
  - **`thresholds`** An array of activity threshold values within the target's activity domain, defining activity intervals $I_1^{uv}, \ldots, I_{k+1}^{uv}$ for the regulation.
- **`contexts`** An array defining regulatory contexts. Each context represents a single compact context and includes:

`intervals` An array of indices $(1, \ldots, k+1)$ of activity intervals (defined by the `thresholds` attribute). Wildcards (`"*"`) may be used to compactly represent multiple formal contexts (see Section 1.1.3).

`target_value` A target activity value that must lie within the target variable's activity domain.

The input variables can be specified in the `regulations` field with an empty array of `regulators`. The corresponding `contexts` array must contain exactly one context having an empty array of `intervals`.

7.2.2 Formula Specification

This field specifies a CTL formula to be evaluated. The formula must adhere to the restricted syntax of CTL as defined in Section 5.1.

The syntax employs the following notation. Negation (`!`) is strictly limited to atomic formulas and cannot be applied to complex state formulas. Union (`|`) and intersection (`&`) operate exclusively on atomic formulas, whereas disjunction (`||`) and conjunction (`&&`) connect state formulas. Temporal operators Next (EX, AX), Future (EF, AF), and Globally (EG, AG) use prefix notation. Until (U) and weak until (W) are written in infix form, but the quantifier (A or E) remains in prefix notation.

7.2.3 Initial State Constraints

This attribute defines an array of constraints on initial states. Each constraint is an object containing fields following the format `"variable": [list of admissible values]`, where values must be within the activity domain of the respective variable. Only specified genes are constrained; missing genes remain unrestricted. Each constraint object defines a convex subset of states, allowing the list of constraints to easily capture any nontrivial subset of desired initial states.

7.3 Input Parsing

We implemented a `MvGRNParser` class to parse the input network, which processes and validates the JSON input representing a multi-valued gene regulatory network. The parser ensures that all required fields (variables and regulations) are present and correctly instantiated. Namely, variables must be a dictionary mapping each gene to a positive integer representing its maximum activity level. Regulations are validated to ensure that each element contains a valid target variable, a list of valid regulators with appropriate activity thresholds, and a list of contexts specifying valid indices of activity intervals and target values. The parser performs consistency checks, raising descriptive errors for structural or semantic issues in the input. Once validated, the parser constructs and returns a `MultivaluedGRN` object storing the parsed network attributes.

CTL formulas are parsed using a custom grammar defined for the `Lark` parser. This grammar enforces the syntactic restrictions outlined in Section 5.1. The grammar distinguishes between two non-terminals for state formulas to ensure that only syntactically valid formulas are accepted. The initial non-terminal, `state_formula`, allows only left-aligned atomic formulas. Then, it allows for a sequence of additional non-atomic state formulas through the expansion rules of the `state_formula_c` non-terminal.

Lark uses the LALR(1) parsing algorithm, which stands for *Look-Ahead Left-to-right Rightmost* derivation with one token of lookahead. It is a fast and deterministic method suitable for well-structured grammars. After parsing, a custom transformer is applied to convert the resulting syntax tree into an internal representation using defined formula classes, which are then used for further evaluation.

A limitation of the LALR(1) approach in this context is that it prioritizes logical operators over temporal ones. Since logical operators appear earlier in the grammar structure, they are parsed first and thus implicitly receive higher precedence. While Lark supports alternative parsing strategies that can better handle ambiguities or precedence issues, these do not support custom transformers yet, which is an essential feature in our implementation for constructing the syntax trees. To eliminate this problem, temporal formulas concatenated with logical operators must be closed in brackets to increase their priority.

7.4 Priority Queues

To optimize the exploration of the STG, we implemented custom minimum and maximum priority queues using the `heapdict` library, which supports efficient indexing and priority updates. The maximum queue is realized as a min-heap by inserting inverted (negative) priority values. A key motivation for our custom implementation was the need for conditional priority updates: priorities are only updated if the new value brings the element closer to the top of the heap (lower for min-heap, higher for max-heap). Otherwise, the original priority is retained. This selective update strategy, which we could not find in existing libraries, reduces unnecessary heap operations and ensures that the most promising paths are propagated first.

7.5 Multivalued Network

The `MultivaluedGRN` class serves as a data container that stores information about the variables (genes and their maximum activity values) and the regulatory rules. These rules include regulators and contexts. The data structure is then used by the `StateTransitionGraph` class to construct a model of the system's state transitions.

The `StateTransitionGraph` uses the `networkx` library to build a directed graph, where each node is a tuple representing a discrete state of the network, and edges represent directed state transitions based on the regulatory rules. All combinations of gene activity levels are generated using `itertools.product`. For each state, successors are computed by evaluating whether regulatory contexts are satisfied using interval-based logic. Successors for each state are computed by checking which regulatory contexts are satisfied for each regulated variable, resulting in at most $n - 1$ successors. This is done using interval-based logic with the `bisect.bisect_right` function. This function determines the index of activity interval where the current gene activity fits based on defined activity thresholds. The resulting index is compared to the one specified in the context. If the state fully matches the context, the corresponding target gene's activity is adjusted by one step toward the selected context's target activity value, and the updated state is added to successors of the original state.

As mentioned earlier, the resulting STG has the following important property: if there is no transition from a given state to another state, such a state receives only one successor - itself. However, if there is at least one transition to another state for a given state, such a state does not have itself as a successor. This key property is closely related to the fairness property, which assumes that every path of STG eventually reaches an attractor. The property we employed excludes self-loops on transient states and, therefore, partially implements fairness.

7.6 CTL Formula Components Classes

The classes representing individual elements of a CTL formula follow a clear hierarchy. All of them inherit from the abstract class `StateFormula`, which defines abstract methods for negation elimination, collecting subformulas, and evaluation. The `AtomicFormula` abstract class extends `StateFormula` and serves as a base for all atomic components. It defines an abstract method for computing the DoV, as each atomic formula propagates the DoV in its own way. Therefore, the actual implementation of DoV computation is moved to the individual subclasses, where each `AtomicProposition` yields its DoV and set operation classes `Conjunction` and `Disjunction` combines them. The abstract class also provides a default implementation of `get_subformulas()` by returning itself, as atomic formulas in our approach are no further divisible. Lastly, it implements a unified evaluation method that calculates the DoV, its complement, and both borders and assigns a satisfaction degree δ^* to each state accordingly.

The remaining classes represent CTL operators. We distinguish between unary operators, which contain a single `operand` attribute, and binary operators, which use `left` and `right` attributes to reference their subformulas. To encapsulate shared behavior, we introduce two abstract base classes—`UnaryOperator` and `BinaryOperator`. All operands are instances of `StateFormula`, collectively forming a syntax tree representing the entire formula. Each class also provides a custom textual representation, which serves both as the formula's printed output and as a unique identifier (label) for the corresponding subformula during the evaluation.

7.7 Computation of Satisfaction Degree

The model-checking procedure begins by retrieving a list of all subformulas of the verified formula and ordered according to its structure. This ensures that all its subformulas appear earlier in the list for each subformula, guaranteeing that operands are evaluated before the formula that depends on them. A `KripkeStructure` object that contains the STG, initial states, and the `quantitative_labeling` structure is initialized. This structure represents a quantification labeling function $\mathcal{Q}$ in the implementation. The structure is a two-level dictionary where the first-level keys are individual states. The corresponding values are dictionaries mapping subformula labels (i.e., their textual representations) to their satisfaction degrees, initially set to `None`.

Each component of the CTL formula then implements the `evaluate` method, taking the Kripke structure as a parameter. Each subclass has its own way of propagating the satisfaction degree. Since the implementation of satisfaction degree propagation closely mirrors the design presented in Chapter 6, there is no need to discuss it further here. In the following, we further discuss the implementation of weighted distance computation and the procedures for determining the extreme depth and distance.

7.7.1 Weighted Distance Implementation

The function `weighted_distance` computes the shortest weighted Manhattan distance from a given source state to the nearest border of the complementary set. If the source state lies inside the DoV, the border refers to the DoV complement; otherwise, it targets the DoV border itself. The function uses a `MinPriorityQueue` to implement Dijkstra's algorithm, where each edge cost corresponds to the weight of a specific dimension, inversely proportional to its maximum activity value. Starting from the given source state, the algorithm explores neighboring states generated by `get_manhattan_neighbors`, ordered by increasing cost, while tracking visited states. A key feature of the implementation is the use of the `decrease_priority()` method, which only updates a state's priority if a shorter path has been found. The algorithm terminates as soon as it dequeues the first border state. Since Manhattan neighbors are generated in increasing order of distance, the

priority queue always dequeues the state with the lowest accumulated cost, and edge costs are positive, the first border state encountered is guaranteed to be the closest to the source.

7.7.2 Extreme Depth Implementation

The `find_extreme_depth` function identifies the depth of the deepest state in a given subspace (DoV or DoV complement) in terms of minimal weighted distance from any border state of the complementary subspace. To achieve this, the function performs a multi-source Dijkstra's algorithm, initializing the distances of all co-border states to zero. Then, it propagates these distances throughout the subspace, maintaining the shortest discovered distance for each state in `distances` dictionary. A `MinPriorityQueue` ensures that states are processed in order of increasing distance. Once the minimum distance to the border has been computed for every state in the subspace, the function returns the maximum of these values, representing the deepest located state in terms of accessibility from the complementary subspace.

8 Evaluation

We present a series of case studies to demonstrate the effectiveness of our algorithm. These are designed to showcase the algorithm's ability to quantify the satisfaction of temporal properties, while also testing its scalability across diverse input sizes.

The evaluation comprises three case studies. The first highlights the comparative and exploratory potential of the algorithm. The second provides a more comprehensive analysis, in which the algorithm is executed on various model specifications, combining different initial states and CTL formulas, to explore and characterize the system's long-term behavior. In the third case study, we evaluate models with a gradually increasing state space and formula complexity, achieved by structurally adding more activity levels to certain variables and introducing additional atomic formulas and temporal operators, in order to examine the computational limits of the algorithm.

Each case study is presented in its own section. Beginning with an introductory overview and biological context, the section is then divided into two parts. The first, Model Specifications, defines the parameters of the models that are examined. The second, Results, describes the performed analysis and discusses its outcomes, including resulting satisfaction degrees, revealed behavioral patterns, or computational performance for the last case.

8.1 Predator-Prey Model

The predator-prey model is a minimal regulatory motif involving two components with asymmetric interactions: the prey promotes the growth of the predator, while the predator represses the prey population. Prey also benefits from self-activation (reproduction), while predators tend to self-degrade in the absence of prey. Such motifs are widely observed in ecological systems. They are also employed in synthetic biology to model competitive or cooperative interactions. The interaction graph is shown in the Figure 8.1.

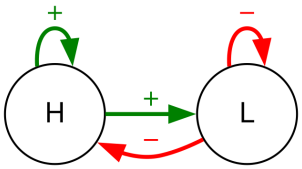


Figure 8.1: Interaction graph of the predator-prey model. The prey (H) exhibits self-activation and promotes the predator (L), while the predator self-represses and inhibits the prey. Green arrows indicate activation and red arrows indicate inhibition.

8.1.1 Model Specifications

We consider two distinct specifications for this model to examine the algorithm's ability to differentiate between models' behavior. In the first, the prey reproduces slowly, and the predator hunts effectively, leading to stronger regulatory control and a higher risk of prey extinction. In the second, the prey population reproduces rapidly while predators hunt less efficiently, resulting in weaker feedback but greater resilience to extinction. Although both configurations exhibit oscillatory behavior, they differ in the robustness of species survival and the individual states' satisfaction degrees.

The following formal parameters define the model topology shared by both examined specifications. We substitute the generic terms prey and predator with H (hares) and L (lynxes), respectively, to be able to distinguish the model variables.

$$\begin{array}{ll} \mathcal{V} = \{H, L\} & \mathcal{E} = \{HH, HL, LH, LL\} \\ \mu_H = 4 & \rho_{HH} = \rho_{HL} = \{1, 2, 3, 4\} \\ \mu_L = 3 & \rho_{LH} = \rho_{LL} = \{1, 2, 3\} \end{array}$$

The specifications of dynamics for both models are formally defined in Appendix B.1. By evaluating their satisfaction degrees, we aim to demonstrate the algorithm's ability to distinguish varying levels of robustness against extinction.

We evaluate the satisfaction of the formula $\mathsf{EG}\,(H \geq 1 \wedge L \geq 1)$, which expresses the existence of a persistent, non-extinction long-term behavior for both species. To focus on biologically meaningful scenarios, we restrict the analysis to initial states where both hares

(H) and lynxes (L) have values greater than zero, ensuring a viable starting population for both prey and predator.

8.1.2 Results

Figure 8.2 shows the comparison of STGs of the predator-prey model under two described specifications. The first component of a state corresponds to prey (H) and the second to predator (L). Despite differing dynamics, the STGs share several structural similarities. Namely, both contain two steady states (highlighted in blue) and a non-terminal strongly connected component, representing a dynamic balance between the populations of both species.

The first steady-state represents the extinction of both species due to the extinction of prey and the subsequent starvation of predators. The second steady state captures the scenario in which predators have gone extinct, allowing the prey population to recover and gradually reach its maximum value.

However, the transient behavior of these two systems differs. The left STG captures a system characterized by slow prey reproduction and efficient predation. Here, the balance between species is sustained only in states where both species' populations are greater or equal than two, which indicates a higher risk of extinction for one or both species. Once the prey population drops to one, the system tends to settle on trajectories leading toward extinction. Overall, transitions leading to terminal states are dominant, indicating lower robustness and a more fragile balance between both species.

In contrast, the right STG reflects dynamics shaped by faster prey reproduction and less effective predation. The system still maintains a balance between both species, but in a more relaxed form, allowing their survival even when the prey population drops to one while predator numbers remain high. Although the same two steady states are present, transitions supporting both species' continued coexistence are dominant. As a result, this configuration demonstrates greater robustness and more flexible species coexistence.

Exploring the output of the algorithm on both model specifications at initial states where $H \geq 1$ and $L \geq 1$, we can unambiguously distinguish the presented dynamics and support the hypothesis about robustness. The results are summarized in Table 8.1.

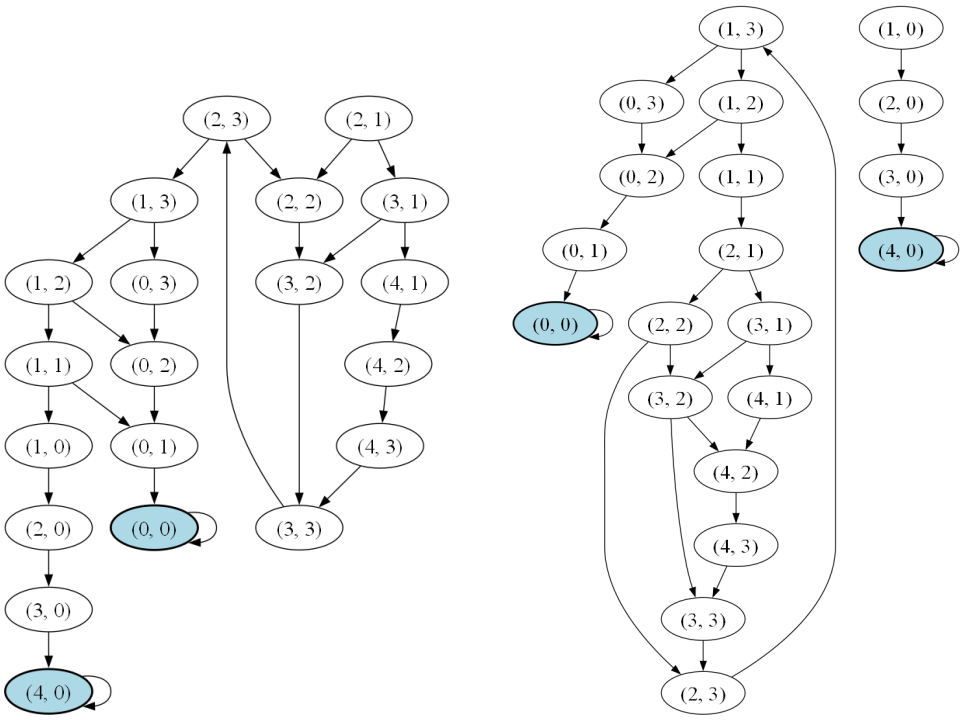


Figure 8.2: Comparison of STGs of the predator-prey model under different regulatory conditions. The first component of a state corresponds to prey (H) and the second to predator (L). **(a)** Slow prey reproduction and efficient predation result in tighter balance between species and an increased risk of prey extinction. **(b)** Rapid prey reproduction and reduced predation result in looser balance between species and greater robustness against extinction.

The results highlight a contrast between strong but less robust satisfaction in Model 1 and more robust but weaker satisfaction in Model 2. Model 1 achieves a higher maximal satisfaction degree (0.5), indicating that when the property is satisfied, it is satisfied strongly. However, it also exhibits a much lower minimal value (–0.5714) and a lower average (0.19047), suggesting that the system frequently falls into states that do not satisfy the formula, indicating lower robustness overall. On the other hand, Model 2 maintains a uniform satisfaction degree across initial states, with both minimal and maximal values at

0.25. While this lower value implies that the system operates closer to the boundary of violating the formula, it satisfies the formula more robustly, as reflected by the higher average satisfaction degree (0.25).

Table 8.1: Output of the algorithm for the two predator-prey model specifications with respect to the formula $\mathsf{EG}\,(H \geq 1 \wedge L \geq 1)$. It shows the minimal, maximal, and average satisfaction degrees among the initial states. For each model, the minimal and maximal values also include an example of a state where they occur.

	Model 1		**Model 2**	
	Value	**State**	**Value**	**State**
Minimal	-0.5714	(1,2)	0.25	(1,2)
Maximal	0.5	(4,3)	0.25	(1,2)
Average	0.19047		0.25	

8.2 Incoherent Feed-Forward Loop with Negative Feedback

The incoherent feed-forward loop is a regulatory motif where upstream node X activates both intermediate node Y and target Z while Y represses Z. We extend this by adding negative feedback from Z to X. This structure supports transient activation, adaptation, and oscillations. Such motifs appear in bacterial stress responses, developmental signaling, and transcriptional networks requiring precise timing and control. The interaction graph is depicted in the Figure 8.3.

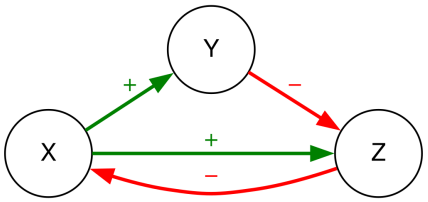


Figure 8.3: Interaction graph of the incoherent feed-forward model with negative feedback. X activates both Y and Z; Y performs delayed inhibition of Z, forming an incoherent feed-forward loop, while Z inhibits X, creating a negative feedback loop.

8.2.1 Model Specifications

In this case study, we focus on a more detailed analysis of the long-term behavior. We explore several similar model specifications using multiple formulas to analyze long-term behavior. The model parameters common to all the examined specifications are defined in the following:

$$\begin{array}{ll} \mathcal{V} = \{X, Y, Z\} & \mathcal{E} = \{XY, XZ, YZ, ZX\} \\ \mu_X = 4 & \rho_{XY} = \rho_{XZ} = \{1, 2, 3, 4\} \\ \mu_Y = 4 & \rho_{YZ} = \{1, 2, 3, 4\} \\ \mu_Z = 4 & \rho_{ZX} = \{1, 2, 3, 4\} \end{array}$$

To further explore the algorithm's ability to distinguish and characterize diverse system behaviors based on satisfaction degree and its robustness among the initial states, we examine the following four specifications of proposed model topology:

1. Regulations from X $\rightarrow$ Y and Z $\dashv$ X exhibit slower onset but develop stronger effect over time; regulation of Z having balanced effects of activation X $\rightarrow$ Z and inhibition Y $\dashv$ Z. This specification leads to a bifurcation, resulting in two steady-states.
2. Regulations from X $\rightarrow$ Y and Z $\dashv$ X similarly exhibit slower onset but develop stronger effect over time; the inhibition Y $\dashv$ Z is slightly favored over activation X $\rightarrow$ Z in the regulation of Z. This subtle shift destabilizes the system into a disordered attractor, where Z can intermittently drop to zero activity.
3. Both the activation X $\rightarrow$ Y and inhibition Z $\dashv$ X have linear effects on their targets; the activation X $\rightarrow$ Z is slightly favored over inhibition Y $\dashv$ Z in the regulation of Z. This creates a stable long-term behavior characterized by a single steady state at intermediate activity levels.
4. The inhibition Z $\dashv$ X exhibits slower onset. Still, it develops a stronger effect over time is combined with linear activation X $\rightarrow$ Y, the activation X $\rightarrow$ Z is slightly favored over inhibition Y $\dashv$ Z in the regulation of Z. This results in a disordered attractor where all components fluctuate. Yet, their activity levels stay above zero, avoiding complete extinction.

The specifications of dynamics for all four models are formally defined in Appendix B.2. Since the resulting STGs are too extensive, we do not include them in the thesis. However, they are stored in separate files in the attached repository.

As we can see, despite sharing the same regulatory logic and similar dynamics, these models exhibit four qualitatively distinct long-term behaviors. By evaluating their satisfaction degrees, we aim to demonstrate the algorithm's ability to distinguish varying levels of robustness against extinction.

To map the reachability of long-term behaviors and assess their robustness against variable extinction, we perform global model checking of the following four temporal formulas:

1. $\mathsf{EF}\,(\mathsf{EG}\,(X \geq 1 \wedge Y \geq 1 \wedge Z \geq 1))$
2. $\mathsf{EF}\,(\mathsf{AG}\,(X \geq 1 \wedge Y \geq 1 \wedge Z \geq 1))$
3. $\mathsf{AF}\,(\mathsf{EG}\,(X \geq 1 \wedge Y \geq 1 \wedge Z \geq 1))$
4. $\mathsf{AF}\,(\mathsf{AG}\,(X \geq 1 \wedge Y \geq 1 \wedge Z \geq 1))$

Each formula expresses different conditions on the long-term activity of all three variables, combining reachability and invariance operators to capture whether fully active states are eventually reachable and, if so, whether they persist.

We evaluated all formulas by performing global model checking to assess how robustly the system as a whole supports the sustained activity of all variables. Results are summarized in separate tables - one for each verified formula, reporting the minimal, maximal, and average satisfaction degrees for each formula, accompanied by an example of a state where they occur.

8.2.2 Results for the First Formula

The results for the formula $\mathsf{EF}\,(\mathsf{EG}\,(X \geq 1 \wedge Y \geq 1 \wedge Z \geq 1))$ are summarized in Table 8.2.

Model 1: Based on the minimum value, we can observe that the steady-state with no variable going extinct is reachable from each state of the model. At the same time, based on the average value, which is close to 0.5, we see that a steady-state (3,3,2), which has a higher

satisfaction degree, is achievable from most states. However, based on the minimum value state (2,0,3), we see that there is at least one transient state, from which the only reachable steady state is (2,1,3), having a weaker satisfaction degree.

Model 2: Based on the outputs, we can observe that long-term disordered behavior with no variable going extinct is reachable from all states. Since the model has a single disorder attractor containing states with zero component Z, it is evident that the disorder contains also an inner cycle with all non-zero components, which is reachable from all states of the model.

Model 3: The results confirm that there is reachable long-term behavior from all states of the model, which exhibits a higher satisfaction degree value. Namely, it is the steady-state (2,2,2).

Model 4: Verifying the current formula does not allow us to distinguish this specification from Model 2. However, the satisfaction degree reveals that no internal cycle maintains all variables strictly above one, as the highest observed satisfaction degree is only 0.25.

Table 8.2: Summary of the algorithm's output for the four specifications of the incoherent feed-forward loop with negative feedback, evaluated against the formula $\mathsf{EF}\,(\mathsf{EG}\,(X \geq 1 \wedge Y \geq 1 \wedge Z \geq 1))$ and performing global model checking. The table reports the minimal, maximal (with their corresponding example states), and average satisfaction degrees over all the initial states.

	Model 1		**Model 2**		**Model 3**		**Model 4**	
	Value	**State**	**Value**	**State**	**Value**	**State**	**Value**	**State**
Minimal	0.25	(2,0,3)	0.25	(0,0,0)	0.5	(0,0,0)	0.25	(0,0,0)
Maximal	0.5	(0,0,0)	0.25	(0,0,0)	0.5	(0,0,0)	0.25	(0,0,0)
Avgerage	0.496		0.25		0.5		0.25	

8.2.3 Results for the Second Formula

The results for the formula $\mathsf{EF}\,(\mathsf{AG}\,(X \geq 1 \wedge Y \geq 1 \wedge Z \geq 1))$ are summarized in Table 8.3.

Model 1: This model specification has two steady-state attractors, each capturing a stable long-term behavior. Therefore, we cannot distinguish this behavior by using the current formula versus using the first formula, and the results are identical to the first analysis.

Model 2: Here, we observe a difference compared to the evaluation of the first formula. In addition to the inner cycle, where values do not drop to 0, the model contains an outer cycle in which the value of variable Z temporarily drops to 0. The outputs indicate that this cycle is reachable from all states. The violation degree of -0.33 suggests that the violation involves only a single variable.

Model 3: Based on the given formula, the behavior is indistinguishable from the first formula analysis. As in Model 1, the steady state represents a single stable long-term behavior, so changing the quantifier of the globally operator does not affect the result.

Model 4: Using this formula does not provide any additional information compared to the analysis made by the first formula.

Table 8.3: Summary of the algorithm's output for the four specifications of the incoherent feed-forward loop with negative feedback, evaluated against the formula $\mathsf{EF}\,(\mathsf{AG}\,(X \geq 1 \wedge Y \geq 1 \wedge Z \geq 1))$ and performing global model checking. The table reports the minimal, maximal (with their corresponding example states), and average satisfaction degrees over all the initial states.

	Model 1		**Model 2**		**Model 3**		**Model 4**	
	Value	**State**	**Value**	**State**	**Value**	**State**	**Value**	**State**
Minimal	0.25	(2,0,3)	-0.33	(0,0,0)	0.5	(0,0,0)	0.25	(0,0,0)
Maximal	0.5	(0,0,0)	-0.33	(0,0,0)	0.5	(0,0,0)	0.25	(0,0,0)
Avgerage	0.496		-0.33		0.5		0.25	

8.2.4 Results for the Third Formula

The results for the formula $\mathsf{AF}\,(\mathsf{EG}\,(X \geq 1 \wedge Y \geq 1 \wedge Z \geq 1))$ are summarized in Table 8.4.

The negative minimum satisfaction degree observed across all models indicates the presence of long-term behavior where at least one variable consistently drops to zero. This suggests the existence of transient cycles accessible only from specific regions of the state space.

Model 1: The maximum value shows that certain states inevitably lead to a steady state without entering the transient cycle. The average suggests that roughly half of the states fall into this category.

Model 2: The maximal achieved satisfaction degree indicates that some states always reach a disorder attractor and avoid the transient cycle. Based on the average value, we assume that there are more than half of such states since they exhibit a lower satisfaction degree as the average value is greater than zero.

Model 3: The maximum value again points to the presence of states that consistently reach a steady state. The average again suggests that roughly half of the states fall into this category.

Model 4: As with Model 2, the results imply that some states reach disorder attractor. A higher average value, considering that they exhibit a lower satisfaction degree, indicates that these states are relatively frequent.

Table 8.4: Summary of the algorithm's output for the four specifications of the incoherent feed-forward loop with negative feedback, evaluated against the formula $\mathsf{AF}\,(\mathsf{EG}\,(X \geq 1 \wedge Y \geq 1 \wedge Z \geq 1))$. The table reports the minimal, maximal (with their corresponding states), and average satisfaction degrees over all the initial states.

	Model 1		**Model 2**		**Model 3**		**Model 4**	
	Value	**State**	**Value**	**State**	**Value**	**State**	**Value**	**State**
Minimal	-0.33	(0,0,0)	-0.33	(0,0,0)	-0.33	(0,0,0)	-0.33	(0,0,0)
Maximal	0.5	(2,2,2)	0.25	(1,1,1)	0.5	(2,2,0)	0.25	(1,1,0)
Avgerage	0.1039		0.0259		0.13		0.0446	

8.2.5 Results for the Fourth Formula

The results for the formula $\mathsf{AF}\,(\mathsf{AG}\,(X \geq 1 \wedge Y \geq 1 \wedge Z \geq 1))$ are summarized in Table 8.5.

Since the minimum value across all specifications is -1, we conclude that, in the worst-case scenario, there exists at least one state from which it is possible to reach a transient cycle that includes the state $(0,0,0)$. The consistently negative average values across all specifications suggest that the inner cycle is widely reachable, with most states eventually leading into it.

Nevertheless, in Models 1, 3, and 4, some states reach only attractors satisfying the specification without encountering the transient cycle. Model 2 displays interesting dynamics. The negative maximum value indicates that no state exclusively reaches an attractor. Yet, the average value of -0.9306 suggests that certain states also avoid the inner cycle where all variables vanish — pointing to a more complex transient landscape.

Given these observations across all four specifications, we proceed with a deeper analysis of the transient behavior present in the respective dynamics.

Table 8.5: Summary of the algorithm's output for the four specifications of the incoherent feed-forward loop with negative feedback, evaluated against the formula $\mathsf{AF}\,(\mathsf{AG}\,(X \geq 1 \wedge Y \geq 1 \wedge Z \geq 1))$. The table reports the minimal, maximal (with their corresponding states), and average satisfaction degrees over all the initial states.

	Model 1		**Model 2**		**Model 3**		**Model 4**	
	Value	**State**	**Value**	**State**	**Value**	**State**	**Value**	**State**
Minimal	-1.0	(0,0,0)	-1.0	(0,0,0)	-1.0	(0,0,0)	-1.0	(0,0,0)
Maximal	0.5	(3,3,2)	-0.33	(3,2,2)	0.5	(2,2,2)	0.25	(3,2,2)
Avgerage	-0.968		-0.9306		-0.988		-0.7206	

8.2.6 Analysis of Non-Terminal Cycles

As mentioned above, all four model specifications contain non-terminal strongly connected components. Outputs from the individual analyses suggest the presence of two types of cycles: one in which all three variables are preserved and another where one variable temporarily drops to zero. The following analysis aims to confirm this hypothesis.

We cannot use the algorithm to separate non-terminal cycles that contain states with all non-zero components from attractors. However, it can verify the presence of non-terminal cycles containing states with at least one zero component, as none of the specifications contain attractors with such cycles. This applies even to Setup 2, which includes such states in the attractor, but they do not form a cycle.

To detect transient cycles of this type, we evaluate the following formula over initial states with at least one component equal to zero:

$$\mathsf{EG}\,(X \leq 0 \vee Y \leq 0 \vee Z \leq 0)$$

Table 8.6 summarizes the results for all four models. The maximal value of 0.33 indicates that each model includes a non-terminal cycle with states where one variable is zero. Minimum values show that some states cannot reach such a cycle. Still, the positive average values suggest that most states can, with slight variation between models.

Table 8.6: Output of the algorithm for the four model specifications of the incoherent feed-forward loop with negative feedback, evaluated against the formula $\mathsf{EG}\,(X \leq 0 \vee Y \leq 0 \vee Z \leq 0)$. Initial states are all the states having at least one zero component. The table reports the minimal, maximal, and average satisfaction degrees over all the initial states, including example states.

	Model 1		**Model 2**		**Model 3**		**Model 4**	
	Value	**State**	**Value**	**State**	**Value**	**State**	**Value**	**State**
Minimal	-0.5	(4, 2, 0)	-0.5	(4, 2, 0)	-0.5	(2,3,0)	-0.5	(2,3,0)
Maximal	0.33	(2,0,2)	0.33	(2,0,2)	0.33	(2,0,2)	0.33	(2,0,2)
Avgerage	0.1844		0.1844		0.1434		0.1338	

8.3 Single Input Module

The Single Input Module (SIM) is a common regulatory motif where a single transcription factor (TF) regulates a set of target genes. This configuration allows for coordinated expression of functionally related genes, typically enabling sequential activation depending on the threshold sensitivity of each target. Such modules are found in bacterial stress responses, metabolic regulation, and developmental gene cascades. The interaction graph is depicted in the Figure 8.4.

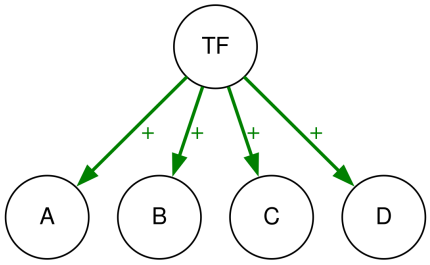


Figure 8.4: Interaction graph of single input module with a transcription factor. TF activates four target genes, A, B, C, and D, through direct positive regulation with different kinetics, resulting in a chained activation pattern.

8.3.1 Model Specifications

In this section, we focus on the complexity of the algorithm we present. The algorithm's time complexity with respect to the input parameters of the model and formula is declared in Section 6.9. The implementation of the proposed solution that we provide is a prototype. Therefore, its primary goal is not efficiency but functionality itself. Despite this, it makes some calculation steps more efficient. From the discussion of the algorithm's time complexity, it is obvious that the algorithm parameters that contribute most to the algorithm's complexity are the number of atomic propositions (due to the signed distance computation using Dijkstra's algorithm) and the size of the state space.

This evaluation aims to determine a user-friendly range of input parameters for which the proposed implementation remains practically usable. It is evident that computation becomes too slow for

large instances, so our goal is to identify realistic usability limits. We use a simple SIM motif whose topology is both straightforward and easily scalable. By gradually increasing the maximum activity levels of target genes and the number of atomic propositions and formulas in the verified formula, we expand the state space and the number of algorithm iterations. This allows us to visualize the relationship between computation time and these two parameters.

The model topologies used for this study are formally defined as follows. The notation emphasizes the gradual increase in maximum activity values for individual target genes A-D:

$$
\begin{array}{ll}
\mathcal{V} = \{TF, A, B, C, D\} & \mathcal{E} = \{TFA, TFB, TFC, TFD\} \\
\mu_{TF} = 7 & \rho_{TFA} = \{1, 3\} \\
\mu_A = 3, 4, 5 & \rho_{TFB} = \{2, 4\} \\
\mu_B = 3, 4, 5 & \rho_{TFC} = \{3, 5\} \\
\mu_C = 3, 4, 5 & \rho_{TFD} = \{4, 7\} \\
\mu_D = 3, 4, 5 &
\end{array}
$$

We use the following dynamic settings, which always depend on the current maximum activity levels for the target genes. It can be noted that the triggering of the expression of individual genes is gradual, with three activity intervals defined for each of the target genes.

$$
\tau_{TF} = \left\{ \begin{array}{ll} \emptyset & \to 7 \end{array} \right.
$$

$$
\tau_A = \begin{cases} I_1^{TFA} & \to 0 \\ I_2^{TFA} & \to 1 \\ I_3^{TFA} & \to \mu_A \end{cases}
\qquad
\tau_C = \begin{cases} I_1^{TFC} & \to 0 \\ I_2^{TFC} & \to 1 \\ I_3^{TFC} & \to \mu_C \end{cases}
$$

$$
\tau_B = \begin{cases} I_1^{TFB} & \to 0 \\ I_2^{TFB} & \to 1 \\ I_3^{TFB} & \to \mu_B \end{cases}
\qquad
\tau_D = \begin{cases} I_1^{TFD} & \to 0 \\ I_2^{TFD} & \to 1 \\ I_3^{TFD} & \to \mu_D \end{cases}
$$

8.3.2 Results

For the evaluation, we used formulas with the following structure:

$$\mathsf{EF}(\Phi_{a_1} \wedge \mathsf{EF}(\Phi_{a_2} \wedge \mathsf{EF}(\Phi_{a_3} \wedge \dots)))$$

Each Φ_a includes exactly two APs. This pattern allows us to extend the formula systematically and test whether the model contains a state sequence satisfying the constraints defined in each Φ_a.

We ran the algorithm five times for each combination of input parameters and averaged the computation times. The following input ranges were used:

- **Number of atomic formulas:** 2 to 6, with each EF operator containing exactly 2 APs. The formula structure was preserved throughout, ensuring consistent scaling of logical complexity.

- **Maximum activity levels of variables** (with the corresponding total number of states in parentheses):

  - 7,2,2,2,2 (648 states)
  - 7,3,3,3,3 (2048)
  - 7,3,3,4,4 (3200)
  - 7,4,4,4,4 (5000)
  - 7,4,4,5,5 (7200)
  - 7,5,5,5,5 (10368)

The results are visualized in Figure 8.5. It is worth noting that the number of APs remains relatively small in typical use cases, as complex formulas often lose interpretability. In contrast, scalability with respect to the number of states, driven by increasing variables and their activity ranges, is a more critical concern, given the exponential growth of the state space.

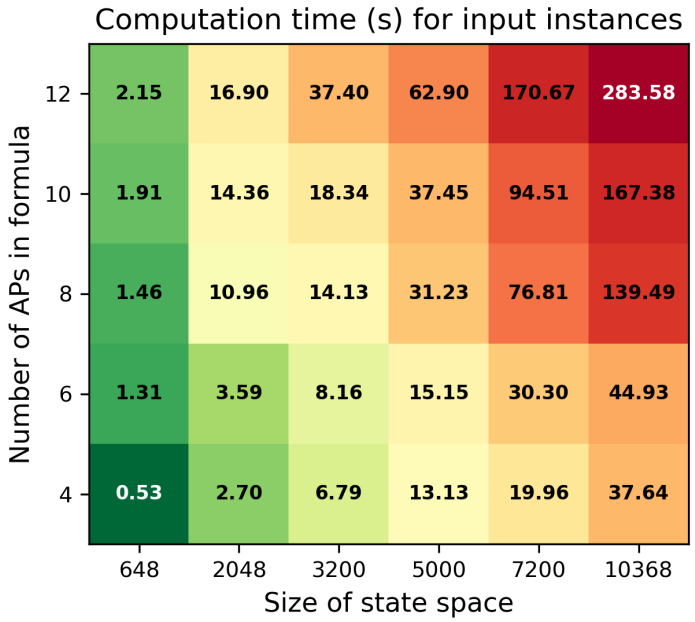


Figure 8.5: The plot shows computation time (in seconds) for varying numbers of atomic propositions (y-axis) and state space sizes (x-axis). Each cell displays the average runtime across five independent runs using the corresponding parameter settings. The green-to-red color gradient reflects increasing computational cost, illustrating the growth of runtime with respect to both parameters.

Despite being a prototype primarily designed to demonstrate the proposed approach, the implementation is capable of handling computations over moderately large state spaces within a relatively feasible computation time. The main bottleneck is the computation of DoVs and signed distances for individual atomic formulas, which are not well-optimized by the algorithm.

9 Conclusion

In this thesis, we addressed the problem of computing the satisfaction degree of CTL formulas over Multivalued Gene Regulatory Networks. Motivated by limitations arising from the Boolean nature of standard CTL model checking, we proposed a novel algorithm that evaluates CTL formulas on a bounded real-valued scale, capturing not only the truth value but also the degree of satisfaction or violation.

To enable this approach, we extended existing frameworks for modeling MvGRNs. MvGRNs are suitable for our goal due to their ability to represent multiple discrete activity levels of gene expression, allowing us to measure distances between states. We formally defined an extended MvGRN formalism that captures the behavior of interactions under all possible configurations of their regulators. This formalism was transformed into a practical modeling framework implemented in Python.

We then proposed a method for computing satisfaction degrees based on signed Manhattan distances from the validity domains of atomic formulas. These values are propagated through the CTL formula's structure, resulting in an overall satisfaction degree that reflects both - the truth value and strength of satisfaction or violation for the whole formula. The algorithm, including the propagation mechanisms and formal concepts, was implemented as a Python prototype with minor optimizations.

We evaluated three case studies to demonstrate the algorithm's utilization and efficiency. These cases were designed to showcase the algorithm's ability to quantify the satisfaction of temporal properties while testing its scalability and robustness across diverse input specifications. The first case study highlights the comparative potential of the algorithm, where we distinguished between two slightly different specifications of the same model. The second provided a more thorough analysis, in which the algorithm is executed multiple times on multiple model specifications, combining different initial state sets and CTL formulas to explore and characterize the system's behavior. The third focused on a model with substantially larger state spaces and more complex formulas to examine the computational limits of the algorithm.

Since the method is mostly enumerative, the evaluation highlighted scalability limitations due to state-space explosion in larger models. Additionally, the increasing number of atomic formulas in the verified formula further increases complexity. Based on these observations, we propose the following directions for future improvement:

- The representation of DoV of atomic formulas (and their boundary) in the current implementation is done in a purely enumerative way by simply listing the states that satisfy the formula. This approach was chosen because DoVs can be non-convex sets of states, which makes compact representations challenging. However, a more efficient structure that can capture non-convexity and identify boundaries precisely would significantly reduce the algorithm's computational complexity. One potential improvement is to incorporate symbolic representations such as Multivalued Decision Diagrams (MDDs), which are well-suited for encoding sets of multivalued states and provide a more compact and efficient alternative to complete enumeration.

- Calculations of weighted distances and extreme depths are currently done separately for each atomic formula. However, the underlying graph representing the state space is the same for all atomic formulas; only the DoV boundaries change. Therefore, it would be useful to consider how to share common information across atomic formulas to improve the overall calculation of weighted distances. The current design performs the same calculation steps redundantly for each atomic formula.

- The output of the current version of the algorithm provides information only about the minimum, maximum, and average satisfaction degree across the initial states. However, it does not provide more detailed information about the distribution of satisfaction degree values across the initial states. For temporal formulas with multiple temporal operators, the algorithm provides information about satisfaction degree values for the entire formula, not for individual subformulas. As revealed in the enumeration, such information can provide additional useful findings of the system. Thus, providing a more detailed output of the performed model analysis would be useful.

Bibliography

1. DONZÉ, Alexandre; FERRÈRE, Thomas; MALER, Oded. Efficient Robust Monitoring for STL. In: *Computer Aided Verification*. Springer Berlin Heidelberg, 2013, pp. 264–279.
2. FAINEKOS, Georgios E.; PAPPAS, George J. Robustness of Temporal Logic Specifications. In: *Formal Approaches to Software Testing and Runtime Verification*. Springer Berlin Heidelberg, 2006, pp. 178–192.
3. FAGES, François; RIZK, Aurélien. From Model-Checking to Temporal Logic Constraint Solving. In: *Principles and Practice of Constraint Programming - CP 2009*. Springer Berlin Heidelberg, 2009, pp. 319–334.
4. KLARNER, Hannes; SIEBERT, Heike. Approximating Attractors of Boolean Networks by Iterative CTL Model Checking. *Frontiers in Bioengineering and Biotechnology*. 2015, vol. 3.
5. ROCCA, Alexandre; RIBEIRO, Tony; INOUE, Katsumi. Inference and learning of Boolean networks using answer set programming. *LNMR 2013*. 2013, p. 17.
6. KLARNER, Hannes; HEINITZ, Frederike; NEE, Sarah; SIEBERT, Heike. Basins of Attraction, Commitment Sets, and Phenotypes of Boolean Networks. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*. 2020, vol. 17, no. 4, pp. 1115–1124.
7. THOMAS, René. Regulatory networks seen as asynchronous automata: A logical description. *Journal of Theoretical Biology*. 1991, vol. 153, no. 1, pp. 1–23.
8. KLARNER, Hannes; STRECK, Adam; ŠAFRÁNEK, David; KOLČÁK, Juraj; SIEBERT, Heike. Parameter Identification and Model Ranking of Thomas Networks. In: *Computational Methods in Systems Biology*. Springer Berlin Heidelberg, 2012, pp. 207–226.
9. CHAOUIYA, Claudine; REMY, Elisabeth; MOSSÉ, Brigitte; THIEFFRY, Denis. Qualitative Analysis of Regulatory Graphs: A Computational Tool Based on a Discrete Formal Framework. In: *Positive Systems*. Springer Berlin Heidelberg, 2003, pp. 119–126.

10. NALDI, Aurélien; REMY, Elisabeth; THIEFFRY, Denis; CHAOUIYA, Claudine. Dynamically consistent reduction of logical regulatory graphs. *Theoretical Computer Science*. 2011, vol. 412, no. 21, pp. 2207–2218.

11. GONZALEZ, A. Gonzalez; NALDI, A.; SÁNCHEZ, L.; THIEFFRY, D.; CHAOUIYA, C. GINsim: A software suite for the qualitative modelling, simulation and analysis of regulatory networks. *Biosystems*. 2006, vol. 84, no. 2, pp. 91–100.

12. NALDI, A.; BERENGUIER, D.; FAURÉ, A.; LOPEZ, F.; THIEFFRY, D.; CHAOUIYA, C. Logical modelling of regulatory networks with GINsim 2.3. *Biosystems*. 2009, vol. 97, no. 2, pp. 134–139.

13. NALDI, Aurélien; HERNANDEZ, Céline; ABOU-JAOUDÉ, Wassim; MONTEIRO, Pedro T.; CHAOUIYA, Claudine; THIEFFRY, Denis. Logical Modeling and Analysis of Cellular Regulatory Networks With GINsim 3.0. *Frontiers in Physiology*. 2018, vol. 9.

14. BAIER, Christel; KATOEN, Joost-Pieter. *Principles of Model Checking*. Vol. 26202649. The MIT Press, 2008.

15. HEMEDAN, Ahmed Abdelmonem; NIARAKIS, Anna; SCHNEIDER, Reinhard; OSTASZEWSKI, Marek. Boolean modelling as a logic-based dynamic approach in systems medicine. *Computational and Structural Biotechnology Journal*. 2022, vol. 20, pp. 3161–3172.

16. SHI, Ning; ZHU, Zexuan; TANG, Ke; PARKER, David; HE, Shan. ATEN: And/Or tree ensemble for inferring accurate Boolean network topology and dynamics. *Bioinformatics*. 2020, vol. 36, no. 2, pp. 578–585.

17. GAO, Shuhua; SUN, Changkai; XIANG, Cheng; QIN, Kairong; LEE, Tong Heng. Learning asynchronous boolean networks from single-cell data using multiobjective cooperative genetic programming. *IEEE Transactions on Cybernetics*. 2022, vol. 52, no. 5, pp. 2916–2930.

18. KARLEBACH, Guy. A Novel Algorithm for the Maximal Fit Problem in Boolean Networks. *arXiv preprint arXiv:1505.07335*. 2015.

19. BENEŠ, Nikola; BRIM, Luboš; PASTVA, Samuel; POLÁČEK, Jakub; ŠAFRÁNEK, David. Formal Analysis of Qualitative Long-Term Behaviour in Parametrised Boolean Networks. In: *Formal Methods and Software Engineering*. Springer International Publishing, 2019, pp. 353–369.

20. GARG, Abhishek; DI CARA, Alessandro; XENARIOS, Ioannis; MENDOZA, Luis; DE MICHELI, Giovanni. Synchronous versus asynchronous modeling of gene regulatory networks. *Bioinformatics*. 2008, vol. 24, no. 17, pp. 1917–1925.

21. ZHENG, Desheng; YANG, Guowu; LI, Xiaoyu; WANG, Zhicai; LIU, Feng; HE, Lei. An efficient algorithm for computing attractors of synchronous and asynchronous Boolean networks. *PloS one*. 2013, vol. 8, no. 4, e60593.

22. RODIONOVA, Alëna; LINDEMANN, Lars; MORARI, Manfred; PAPPAS, George. Temporal robustness of temporal logic specifications: Analysis and control design. *ACM Transactions on Embedded Computing Systems*. 2022, vol. 22, no. 1, pp. 1–44.

23. FOUNDATION, Python Software. *lark-parser 0.12.0* [`https://pypi.org/project/lark-parser/`]. Python Package Index, 2021 [visited on 2025-05-19].

A Github Repository Content

This section provides an overview of the structure and contents of the accompanying GitHub repository. The repository consists of three main subdirectories: `data/` contains JSON files specifying case studies used; `src/` includes all implementation source files and unit test directory `test/`; and `stg/` stores visual representations of studied STGs as PNG images.

```
CTL_sat_degree/
├── data/
│   ├── incoherent_ffl1.json
│   ├── incoherent_ffl2.json
│   ├── incoherent_ffl3.json
│   ├── incoherent_ffl4.json
│   ├── predator_prey1.json
│   ├── predator_prey2.json
│   └── single_input_module.json
├── src/
│   ├── ctl_formulae.py
│   ├── custom_types.py
│   ├── kripke_structure.py
│   ├── lark_ctl_parser.py
│   ├── main.py
│   ├── multivalued_grn.py
│   ├── priority_queue.py
│   ├── weighted_distance.py
│   └── test/
│       ├── formula_methods_test.py
│       ├── mvgrn_test.py
│       ├── parser_test.py
│       ├── priority_queue_test.py
│       └── weighted_distance_test.py
└── stg/
    ├── incoherent_ffl1.png
    ├── incoherent_ffl2.png
    ├── incoherent_ffl3.png
    ├── incoherent_ffl4.png
    ├── predator_prey1.png
    ├── predator_prey2.png
    └── single_input_module.png
```

B Specification of Evaluated Models

This chapter provides supplementary model specifications of evaluated cases.

B.1 Predator-Prey Model

Model specification for the first evaluated case, in which the prey reproduces slowly and the predator hunts effectively, leading to stronger regulatory control and a higher risk of prey extinction.

$$\tau_H = \begin{cases} I_2^{HH}, I_1^{LH} & \to 4 \\ I_3^{HH}, I_3^{LH} & \to 3 \\ I_4^{HH}, I_3^{LH} & \to 3 \\ I_5^{HH}, I_3^{LH} & \to 4 \\ I_3^{HH}, I_4^{LH} & \to 1 \\ I_4^{HH}, I_4^{LH} & \to 2 \\ I_5^{HH}, I_4^{LH} & \to 3 \\ I_1^{HH} & \to 0 \\ I_2^{HH} & \to 0 \\ I_2^{LH} & \to 4 \\ I_1^{LH} & \to 4 \end{cases} \qquad \tau_L = \begin{cases} I_3^{HL}, I_3^{LL} & \to 2 \\ I_4^{HL}, I_3^{LL} & \to 3 \\ I_5^{HL}, I_3^{LL} & \to 3 \\ I_3^{HL}, I_4^{LL} & \to 2 \\ I_4^{HL}, I_4^{LL} & \to 3 \\ I_5^{HL}, I_4^{LL} & \to 3 \\ I_1^{LL} & \to 0 \\ I_1^{HL} & \to 0 \\ I_2^{HL} & \to 0 \\ I_2^{LL} & \to 2 \end{cases}$$

Model specification for the second evaluated case. The prey population proliferates rapidly while predators hunt less efficiently, resulting in weaker feedback but greater resilience to extinction.

$$\tau_H = \begin{cases} I_2^{HH}, I_2^{LH} & \to 2 \\ I_3^{HH}, I_4^{LH} & \to 1 \\ I_1^{HH} & \to 0 \\ I_1^{LH} & \to 4 \\ I_2^{HH} & \to 0 \\ I_2^{LH} & \to 4 \\ I_3^{LH} & \to 4 \\ I_4^{LH} & \to 2 \end{cases} \qquad \tau_L = \begin{cases} I_3^{HL}, I_4^{LL} & \to 3 \\ I_1^{LL} & \to 0 \\ I_1^{HL} & \to 0 \\ I_2^{HL} & \to 1 \\ I_3^{HL} & \to 3 \\ I_4^{HL} & \to 3 \\ I_5^{HL} & \to 3 \end{cases}$$

B.2 Incoherent Feed-Forward Loop with Negative Feedback

In this section we provide supplementary model specifications of all four examined model specifications evaluated in Section 8.2.

B.2.1 Model 1

Regulations from X $\rightarrow$ Y and Z $\dashv$ X exhibit slower onset but develop stronger effect over time; regulation of Z having balanced effects of activation X $\rightarrow$ Z and inhibition Y $\dashv$ Z. This specification leads to a bifurcation, resulting in two distinct steady states.

$$\tau_X = \begin{cases} I_3^{ZX} & \rightarrow 3 \\ I_4^{ZX} & \rightarrow 2 \\ I_5^{ZX} & \rightarrow 0 \\ \emptyset & \rightarrow 4 \end{cases}$$

$$\tau_Y = \begin{cases} I_3^{XY} & \rightarrow 1 \\ I_4^{XY} & \rightarrow 3 \\ I_5^{XY} & \rightarrow 4 \\ \emptyset & \rightarrow 0 \end{cases}$$

$$\tau_Z = \begin{cases} I_1^{XZ}, I_1^{YZ} & \rightarrow 1 \\ I_2^{XZ}, I_1^{YZ} & \rightarrow 1 \\ I_2^{XZ}, I_2^{YZ} & \rightarrow 1 \\ I_3^{XZ}, I_1^{YZ} & \rightarrow 3 \\ I_3^{XZ}, I_2^{YZ} & \rightarrow 3 \\ I_3^{XZ}, I_3^{YZ} & \rightarrow 2 \\ I_3^{XZ}, I_4^{YZ} & \rightarrow 1 \\ I_3^{XZ}, I_5^{YZ} & \rightarrow 1 \\ I_4^{XZ}, I_1^{YZ} & \rightarrow 4 \\ I_4^{XZ}, I_2^{YZ} & \rightarrow 4 \\ I_4^{XZ}, I_3^{YZ} & \rightarrow 3 \\ I_4^{XZ}, I_4^{YZ} & \rightarrow 2 \\ I_4^{XZ}, I_5^{YZ} & \rightarrow 1 \\ I_5^{XZ}, I_4^{YZ} & \rightarrow 3 \\ I_5^{XZ}, I_5^{YZ} & \rightarrow 2 \\ I_1^{XZ} & \rightarrow 0 \\ I_2^{XZ} & \rightarrow 0 \\ I_5^{XZ} & \rightarrow 4 \end{cases}$$

B.2.2 Model 2

Regulations from X $\rightarrow$ Y and Z $\dashv$ X similarly exhibit slower onset but develop stronger effect over time; the inhibition Y $\dashv$ Z is slightly favored over activation X $\rightarrow$ Z in regulation of Z. This subtle shift destabilizes the system into a disordered attractor, where Z can intermittently drop to zero activity.

$$\tau_X = \begin{cases} I_3^{ZX} & \rightarrow 3 \\ I_4^{ZX} & \rightarrow 2 \\ I_5^{ZX} & \rightarrow 0 \\ \emptyset & \rightarrow 4 \end{cases}$$

$$\tau_Y = \begin{cases} I_3^{XY} & \rightarrow 1 \\ I_4^{XY} & \rightarrow 3 \\ I_5^{XY} & \rightarrow 4 \\ \emptyset & \rightarrow 0 \end{cases}$$

$$\tau_Z = \begin{cases} I_2^{XZ}, I_1^{YZ} & \rightarrow 1 \\ I_2^{XZ}, I_2^{YZ} & \rightarrow 1 \\ I_3^{XZ}, I_1^{YZ} & \rightarrow 3 \\ I_3^{XZ}, I_2^{YZ} & \rightarrow 2 \\ I_3^{XZ}, I_3^{YZ} & \rightarrow 1 \\ I_3^{XZ}, I_4^{YZ} & \rightarrow 0 \\ I_3^{XZ}, I_5^{YZ} & \rightarrow 0 \\ I_4^{XZ}, I_1^{YZ} & \rightarrow 4 \\ I_4^{XZ}, I_2^{YZ} & \rightarrow 3 \\ I_4^{XZ}, I_3^{YZ} & \rightarrow 2 \\ I_4^{XZ}, I_4^{YZ} & \rightarrow 1 \\ I_4^{XZ}, I_5^{YZ} & \rightarrow 0 \\ I_5^{XZ}, I_1^{YZ} & \rightarrow 4 \\ I_5^{XZ}, I_2^{YZ} & \rightarrow 4 \\ I_5^{XZ}, I_3^{YZ} & \rightarrow 3 \\ I_5^{XZ}, I_4^{YZ} & \rightarrow 2 \\ I_5^{XZ}, I_5^{YZ} & \rightarrow 2 \\ I_1^{XZ} & \rightarrow 0 \\ I_2^{XZ} & \rightarrow 0 \end{cases}$$

B.2.3 Model 3

Both the activation X $\rightarrow$ Y and inhibition Z $\dashv$ X have linear effects on their targets; the activation X $\rightarrow$ Z is slightly favored over inhibition Y $\dashv$ Z in regulation of Z. This creates a stable long-term behavior characterized by a single steady state at intermediate activity levels.

$$\tau_X = \begin{cases} I_1^{ZX} & \rightarrow 4 \\ I_2^{ZX} & \rightarrow 3 \\ I_3^{ZX} & \rightarrow 2 \\ I_4^{ZX} & \rightarrow 1 \\ I_5^{ZX} & \rightarrow 0 \end{cases}$$

$$\tau_Y = \begin{cases} I_1^{XY} & \rightarrow 0 \\ I_2^{XY} & \rightarrow 1 \\ I_3^{XY} & \rightarrow 2 \\ I_4^{XY} & \rightarrow 3 \\ I_5^{XY} & \rightarrow 4 \end{cases}$$

$$\tau_Z = \begin{cases} I_1^{XZ}, I_1^{YZ} & \rightarrow 1 \\ I_1^{XZ}, I_2^{YZ} & \rightarrow 1 \\ I_2^{XZ}, I_1^{YZ} & \rightarrow 2 \\ I_2^{XZ}, I_2^{YZ} & \rightarrow 1 \\ I_3^{XZ}, I_1^{YZ} & \rightarrow 3 \\ I_3^{XZ}, I_2^{YZ} & \rightarrow 3 \\ I_3^{XZ}, I_3^{YZ} & \rightarrow 2 \\ I_3^{XZ}, I_4^{YZ} & \rightarrow 1 \\ I_3^{XZ}, I_5^{YZ} & \rightarrow 1 \\ I_4^{XZ}, I_1^{YZ} & \rightarrow 4 \\ I_4^{XZ}, I_2^{YZ} & \rightarrow 4 \\ I_4^{XZ}, I_3^{YZ} & \rightarrow 3 \\ I_4^{XZ}, I_4^{YZ} & \rightarrow 2 \\ I_4^{XZ}, I_5^{YZ} & \rightarrow 1 \\ I_5^{XZ}, I_1^{YZ} & \rightarrow 4 \\ I_5^{XZ}, I_2^{YZ} & \rightarrow 4 \\ I_5^{XZ}, I_3^{YZ} & \rightarrow 3 \\ I_5^{XZ}, I_4^{YZ} & \rightarrow 3 \\ I_5^{XZ}, I_5^{YZ} & \rightarrow 2 \\ \emptyset & \rightarrow 0 \end{cases}$$

B.2.4 Model 4

The inhibition Z ⊣ X exhibit slower onset but develop stronger effect over time is combined with linear activation X → Y, the activation X → Z is slightly favored over inhibition Y ⊣ Z in regulation of Z. This results in a disordered attractor where all components fluctuate, yet their activity levels stay above zero, avoiding complete extinction.

$$\tau_X = \begin{cases} I_3^{ZX} & \to 3 \\ I_4^{ZX} & \to 2 \\ I_5^{ZX} & \to 0 \\ \emptyset & \to 4 \end{cases}$$

$$\tau_Y = \begin{cases} I_1^{XY} & \to 0 \\ I_2^{XY} & \to 1 \\ I_3^{XY} & \to 2 \\ I_4^{XY} & \to 3 \\ I_5^{XY} & \to 4 \end{cases}$$

$$\tau_Z = \begin{cases} I_2^{XZ}, I_1^{YZ} & \to 2 \\ I_2^{XZ}, I_2^{YZ} & \to 1 \\ I_2^{XZ}, I_3^{YZ} & \to 1 \\ I_2^{XZ}, I_4^{YZ} & \to 0 \\ I_2^{XZ}, I_5^{YZ} & \to 0 \\ I_3^{XZ}, I_1^{YZ} & \to 3 \\ I_3^{XZ}, I_2^{YZ} & \to 2 \\ I_3^{XZ}, I_3^{YZ} & \to 1 \\ I_3^{XZ}, I_4^{YZ} & \to 1 \\ I_3^{XZ}, I_5^{YZ} & \to 0 \\ I_4^{XZ}, I_1^{YZ} & \to 4 \\ I_4^{XZ}, I_2^{YZ} & \to 3 \\ I_4^{XZ}, I_3^{YZ} & \to 2 \\ I_4^{XZ}, I_4^{YZ} & \to 1 \\ I_4^{XZ}, I_5^{YZ} & \to 1 \\ I_5^{XZ}, I_1^{YZ} & \to 4 \\ I_5^{XZ}, I_2^{YZ} & \to 4 \\ I_5^{XZ}, I_3^{YZ} & \to 3 \\ I_5^{XZ}, I_4^{YZ} & \to 2 \\ I_5^{XZ}, I_5^{YZ} & \to 2 \\ I_1^{X} & \to 0 \end{cases}$$