# Learning Temporal Properties for Explainability and Verification

Thesis approved by
the Department of Computer Science
University of Kaiserslautern-Landau
for the award of the Doctoral Degree
Doctor of Natural Sciences (Dr. rer. nat.)

to

Rajarshi Roy

| | |
|---|---|
| Date of Defense: | 13.09.2024 |
| Dean: | Prof. Dr. Christoph Garth |
| Reviewer: | Prof. Dr. Daniel Neider |
| Reviewer: | Prof. Dr. Rupak Majumdar |
| Reviewer: | Dr. Benedikt Bollig |

DE-386

# Summary

The past decade has witnessed the rise of several intelligent systems fueled by the large-scale adoption of data-driven techniques. However, due to the inherent black-box nature of data-driven techniques, intelligent systems tend to have relatively complex designs. At times, even the designers of these systems struggle to comprehend them fully. Consequently, it is not surprising that the failures of intelligent systems are commonplace.

To deploy intelligent systems for widespread use, it is crucial for humans to place significant trust in them. Towards the goal of developing trust, in this thesis, we focus on techniques that are targeted towards (i) explaining the behavior of systems in a human-interpretable manner, and (ii) facilitating their verification through formal methods.

Our primary intent is to explain and formalize the *temporal* behavior of systems. To represent temporal behavior, we exploit formalisms that originate from formal language theory, such as *temporal logic* and *finite automata*. Such formalisms are considered to be easy-to-understand and, at the same time, are formal enough to unambiguously express temporal behavior. Consequently, both formalisms have had numerous applications: as specifications for formal verification, as interpretable descriptions for Explainable AI, etc.

Our approach to formalizing temporal behavior involves the automated *learning* of temporal properties, considering system executions as input data. By varying different aspects of the input data and the formalism used for expressing temporal properties, we formulate several learning problems suited to diverse practical scenarios. We consider learning in the presence of noise, from only positive data, etc. Moreover, we consider well-known temporal logics such as Linear Temporal Logic (LTL), Metric Temporal Logic (MTL), Property Specification Language (PSL), etc. and finite automata such as Deterministic Finite Automata (DFAs).

We study various aspects of the considered learning problems. Most importantly, we design efficient algorithms to tackle the learning problems. Our algorithms rely on popular techniques such as satisfiability problems, combinatorial search, decision-tree learning, etc. A crucial feature of our algorithms is that they have *guarantees* (e.g., termination, soundness, completeness, etc.) that are validated through formal proofs. Moreover, we investigate associated decision problems and present complexity results, providing further insights into the learning problems. We implement all algorithms in user-friendly and open-source tools and test them on a wide range of synthetic and real-world benchmarks. Through our experiments, we successfully learned meaningful temporal properties in a number of scenarios.

# Zusammenfassung

Das vergangene Jahrzehnt hat das Aufkommen mehrerer intelligenter Systeme erlebt, die durch die umfassende Anwendung datengetriebener Techniken angetrieben wurden. Aufgrund der inhärenten Black-Box-Natur dieser Techniken neigen intelligente Systeme dazu, relativ komplexe Designs zu haben. Manchmal haben selbst die Designer dieser Systeme Schwierigkeiten, sie vollständig zu verstehen. Folglich ist es nicht überraschend, dass Fehler bei intelligenten Systemen alltäglich sind.

Um intelligente Systeme für den weit verbreiteten Einsatz bereitzustellen, ist es entscheidend, dass Menschen ihnen signifikantes Vertrauen entgegenbringen. Im Streben nach Vertrauensbildung konzentrieren wir uns in dieser Arbeit auf Techniken, die darauf abzielen, (i) das Verhalten von Systemen auf eine für Menschen interpretierbare Weise zu erklären und (ii) deren Überprüfung durch formale Methoden zu erleichtern.

Unsere Hauptabsicht besteht darin, das *temporale* Verhalten von Systemen zu erklären und zu formalisieren. Um temporales Verhalten darzustellen, nutzen wir Formalismen, die aus der formalen Sprachtheorie stammen, wie *temporale Logik* und *endliche Automaten*. Solche Formalismen gelten als leicht verständlich und sind gleichzeitig formal genug, um temporales Verhalten eindeutig auszudrücken. Folglich haben beide Formalismen zahlreiche Anwendungen, beispielsweise als Spezifikationen für formale Verifikation, als interpretierbare Beschreibungen für Explainable AI, etc.

Unser Ansatz zur Formalisierung des temporalen Verhaltens beinhaltet das automatisierte *Lernen* von temporalen Eigenschaften, wobei Systemausführungen als Eingabedaten betrachtet werden. Durch Variation verschiedener Aspekte der Eingabedaten und des Formalismus zur Darstellung temporaler Eigenschaften formulieren wir mehrere Lernprobleme, die für verschiedene praktische Szenarien geeignet sind. Wir betrachten das Lernen aus nur positiven Daten usw. Darüber hinaus berücksichtigen wir bekannte temporale Logiken wie Linear Temporal Logic (LTL), Metric Temporal Logic (MTL), Property Specification Language (PSL), etc. und endliche Automaten wie Deterministic Finite Automata (DFAs).

Wir untersuchen verschiedene Aspekte der betrachteten Lernprobleme. Am wichtigsten ist, dass wir effiziente Algorithmen entwerfen, um die Lernprobleme zu bewältigen. Unsere Algorithmen stützen sich auf beliebte deduktive Techniken wie Erfüllbarkeitsprobleme, kombinatorische Suche, Entscheidungsbaum-Lernen, etc. Ein entscheidendes Merkmal unserer Algorithmen ist, dass sie *Garantien*, Beendigung, Solidität, Vollständigkeit, usw. haben, die durch formale Beweise validiert werden. Darüber hinaus untersuchen wir zugehörige Entscheidungsprobleme und präsentieren Komplexitätsergebnisse, die weitere Einblicke in die Lernprobleme bieten. Wir implementieren alle Algorithmen in benutzerfreundliche Open-Source-Tools und testen sie an einer Vielzahl synthetischer und realer Benchmarks. Durch

unsere Experimente konnten wir in zahlreichen Szenarien sinnvolle temporale Eigenschaften erfolgreich erlernen.

# *Acknowledgements*

This thesis marks the end of a small journey and the beginning of perhaps a longer one; it surely calls for a moment of gratitude toward the people who have and will make significant contributions to my journey.

I owe a great deal to my teachers at CMI who introduced me to computer science and inspired me enough to pursue research in this field. I am especially grateful to Professors B. Srivathsan, Madhavan Mukund and C. Aiswarya for igniting my interest in formal methods. My advisor, Daniel Neider, has probably had the single most impact on my research and personal growth. He has been a remarkable mentor, guiding me with patience as I navigated through the nuances of academic research.

I am fortunate to have collaborated with brilliant researchers such as Benedikt Bollig, Nathanaël Fijalkow, Dana Fisman, Martin Leucker, Guillermo Perez, Zhe Xu, and many others, who have considerably enriched my scientific knowledge. I am also thankful to Rupak Majumdar, Anne-Katherin Schmuck, and Damien Zuffrey for providing insightful feedback on my work and proposing stimulating research directions during our group meetings.

I am grateful to MPI-SWS administration for creating an incredibly supportive work environment with amazing staff members: Andy, Christian, Corinna, Geraldine, Mary-Lou, Mouna, Pascal, Ruth, Susanne, Tobias, Torsten, and Vera. They have always gone above and beyond to make things easier for me.

This journey has been significantly more enjoyable due to the amazing group of friends and colleagues who accompanied me along the way. I am thankful to Ivan, Kaushik, Mahmoud, Marco M., Marco P., Marko, Pascal, Rosa, Simin, Shayari, Stratis, and Supriya for a smooth introduction to life in MPI-SWS (and generally as a graduate student). I am thankful to my friends from CMI, Anirban, Ananyo, Arijit, Arghya, Chayan, Debraj, Kush, Rajat, Ritwik, Nanoty, Niladri, Sougata, Suman, Sushant, Vasudha, and Utsab, who made the virtual life during the pandemic seem bearable. I am grateful to the lively group of colleagues who brought back fun and energy to MPI-SWS: Aman, Ana, Arabinda, Arash, Aris, Bala, Eleni, Ellen, Felix, Filip, Germano, Hasan, Iason, Irmak, Ivi, Kimaya, Khushraj, Leo, Lia, Mahdi, Marin, Michalis, Mohammad, Munko, Nastaran, Nina, Numair, Pavel, Ram, Suhas, and Xuan. Lastly, I thank Ashwani, Eirene, Satya, Sathiya, and Ritam for simply making life a more enjoyable experience.

Above all, none of this was possible without the unconditional support of my family: my parents, my sister, my uncle, and my grandparents. They hold incredible importance in my life, and they are the pillars of my strength.

# Contents

# Chapter 1

# Introduction

Over the past decade, technological development has witnessed an unparalleled surge in the integration of automation in software systems. One of the major driving forces behind this surge has been the adoption of data-driven techniques, or what is popularly known as Artificial Intelligence (AI), in system design. With the proliferation of AI, we are able to design "intelligent" systems that are capable of achieving feats once unimaginable. Examples of such systems include autonomous vehicles [65], human-like chat assistants [220], and robot workers in household and industrial applications [4], to name only a few.

The ability of intelligent systems to perform complex tasks often necessitates a rather complex design. Often, even the designers of such systems cannot fully comprehend their inner workings. This, typically, is an artifact of the nature of the majority of data-driven algorithms: they are difficult to explain or interpret but easy to implement [73, 162]. Systems developed using such algorithms are often treated as "black-boxes".

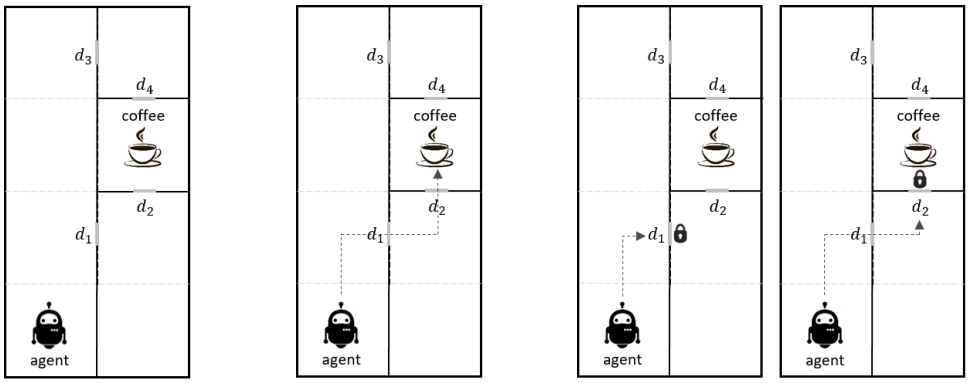
Despite not being aware of their functioning, intelligent systems are deployed to perform complex tasks solely based on their performance in a limited number of scenarios. As a consequence, news of failures of such systems is commonplace. Road accidents caused by autonomous vehicles [64, 156], manipulative remarks made by chat assistants [216], and robots injuring factory workers [194] are all examples of such failures.

Thus, to deploy intelligent systems in widespread usage, especially in safety-critical applications, it is crucial for us to place significant trust in them. Towards the goal of developing trust, in this thesis, we design several techniques that are targeted towards:

1. explaining the behavior of systems in a human-interpretable manner, and
2. facilitating the verification of systems using formal methods.

Our main focus in this thesis is to explain and formalize the *temporal* behavior of systems. This choice is driven by the observation that numerous systems, especially cyber-physical systems such as autonomous vehicles, avionics systems, etc., display complex temporal dynamics involving several parameters of both the system and its environment. A better understanding of the temporal patterns and trends of such systems yields insights into tasks such as ensuring safe future behavior, designing effective controllers, enabling strategic planning, and more.

To formally represent temporal behavior, we exploit formalisms that originate from formal language theory, such as temporal logic and finite state automata. Such formalisms are




(a) Office-like environment (b) Positive example where the agent is able to collect coffee (c) Negative examples where the agent is unable to collect coffee

FIGURE 1.1: An office-like environment where an autonomous agent is deployed to collect coffee. The environment consists of several doors (which may or may not be open) to reach the coffee machine.

mathematically rigorous and are able to express temporal properties unambiguously. At the same time, they are considered human-interpretable and employed as simple descriptions of complex systems. Consequently, both the formalisms have had numerous applications: as specifications for formal verification [179, 14] and reactive synthesis [181, 84], as task descriptions in motion planning [81, 99] and reinforcement learning [46, 128], as interpretable descriptions for Explainable AI [169, 47], etc.

To illustrate an interpretable temporal property, let us consider a prototypical scenario from motion planning where an autonomous agent is deployed to deliver coffee. Figure 1.1a illustrates the office-like environment where we consider the agent to be deployed. The environment includes a room with a coffee machine and four doors $d_1$, $d_2$, $d_3$, and $d_4$. Let us assume that the doors $d_1$ and $d_2$ may not be open sometimes (in which case, they are marked with a lock symbol), while doors $d_3$ and $d_4$ are always open. The agent is free to move through the open doors. We now consider a few executions of the agent in this environment. Figure 1.1b demonstrates a positive execution, where the agent was able to collect coffee via doors $d_1$ and $d_2$. Figure 1.1c illustrates two negative executions, where the agent was not able to collect coffee since it got stuck after finding doors $d_1$ or $d_2$ closed.

Based on the executions, one can deduce a possible explanation for why the agent was not able to collect coffee on certain occasions as the formula:

$$\varphi_1 \coloneqq \mathsf{F}((\neg d_1 \vee \neg d_2) \rightarrow \mathsf{G}\,\mathsf{X}(\neg \texttt{coffee})).$$

The above formula, expressed in a popular temporal logic Linear Temporal Logic (LTL), simply says the following:

*"if the agent eventually finds $d_1$ or $d_2$ to be closed,*
*then it never collects* `coffee` *later".*

Along with boolean operators, $\varphi_1$ uses standard temporal modalities $\mathsf{G}$ (stands for *globally*), $\mathsf{F}$ (stands for *finally*), and $\mathsf{X}$ (stands for *next*). Such temporal modalities can be used in combination to represent several temporal relations such as *always*, *never*, *eventually*,

*next*, *later*, etc. In this case, $\varphi_1$, through a combination of boolean and temporal operators, succinctly and unambiguously explains why the agent failed in its task.

Also, based on the executions, observe that the agent tries to use the path via doors $d_1$ and $d_2$, while there is an alternate, albeit longer, path via doors $d_3$ and $d_4$. To help the agent find this path, one can use another LTL formula:

$$\varphi_2 := \mathsf{F}((\neg d_1 \vee \neg d_2) \rightarrow \mathsf{F}(d_3 \wedge \mathsf{F}\, d_4)),$$

which says the following:

*"if the agent eventually finds $d_1$ or $d_2$ to be closed,*
*then it must eventually find $d_3$ followed by $d_4$ to be open".*

By formally verifying whether the system satisfies $\varphi_2$, one can ensure better performance of the agent.

To systematically deduce such temporal properties, we devise techniques that automatically *learn* (or infer) them by relying on observed data. The data we consider consists of observed executions of the system being studied. By varying the type of input data and the formalism used for expressing temporal properties, we formulate various learning problem settings that arise in practical scenarios. We consider learning in the presence of noise, from only positive data, based on infinite executions, based on finite executions, and so on. As formalisms, we rely on well-known temporal logics such as Linear Temporal Logic (LTL) [179], Metric Temporal Logic (MTL) [141], Signal Temporal Logic (STL) [157], Property Specification Language (PSL) [79] and, sometimes, finite state automata such as Deterministic Finite Automata (DFAs) [183].

Temporal logics and finite state machines have a variety of applications, and many of them require automated learning. However, each application comes with its unique requirements and thus demands the learning process to be tailored accordingly. This thesis will focus on two significant application domains: Explainable AI and Formal Verification. We formulate our learning problems by accommodating the nuances of these two domains. We now briefly introduce the domains and how our techniques fit into these domains.

### Interpretable Descriptions for Explainable AI

Explainable AI (XAI) [162] is a rapidly growing field with a major focus on explaining black-box systems. One of the challenge areas in XAI is to explain the temporal behavior of (cyber-physical or software) systems using interpretable descriptions. Formally, based on XAI terminology, the task is to infer *model-agnostic* explanations (that is, explanations not specific to any model) of *local* temporal behavior (that is, behavior relating to a specific scenario).

In the past, most works that fit in this setting focused on learning finite state automata for a concise description of the behaviors (see Neider [167] and López et al. [151] for a summary). In contrast, our techniques are designed primarily for learning *small* formulas in temporal logic. Temporal logics have an inherent resemblance to natural language, and it is often easy to generate natural language descriptions for small formulas [55]. Thus, from an interpretability perspective, learning formulas in temporal logic have started gaining traction.

Unlike many approaches in XAI, our learning techniques provide theoretical guarantees on the explanations based on the input data, validated through formal proofs. Moreover, they are designed for practical situations such as noise in the input data (Chapter 4) and lack of negative examples (Chapter 6). Further, we explore means to scale the learning process to suit real-world AI applications (Chapter 3).

A distinctive feature of our techniques in the XAI domain is that the input data typically consists of finite executions. This is because, in many AI applications, systems usually terminate after a finite duration [100]. As a result, we employ variants of temporal logic—such as $\text{LTL}_\text{f}$—which are adaptations of the traditional ones to reason about finite executions.

### Formal Specifications for Verification

Formal verification is a rigorous method of verifying systems against formal specifications by mathematically analyzing their behavior. However, there is an often overlooked catch with formal verification: manually designing correct specifications that express the desired requirements precisely is a tedious and error-prone task. Thus, one of the biggest challenges in formal verification has been the design of functional and correct specifications [31]. To alleviate this serious challenge, there have been concentrated efforts to automatically synthesize specifications from system executions [52, 144, 127].

Our learning techniques are also useful in generating specifications in temporal logic. We explore techniques to incorporate the intuition of the system designer (Chapter 5). Further, we consider learning specifications in temporal logics that have industrial applications such as PSL (Chapter 7) and continuous-time logics such as MTL (Chapter 8). Our contributions for learning specifications enhance the existing work for LTL [169] and STL [22].

A distinctive feature of our techniques in the verification domain is that the input data typically consists of infinite executions. This is because formal verification was originally developed to verify (non-terminating) reactive systems that produce infinite executions. In this domain, we mostly employ traditional variants of temporal logics designed for infinite executions.

Despite the differences in the application domains, we will present all the techniques in a unified manner in the thesis. Wherever possible, we will mention how to adapt the techniques to suit to the other domain.

## 1.1 Outline of the Chapters

We now briefly summarize the contents of each chapter of this thesis.

**Chapter 2: Basics of Learning Temporal Properties.** In this chapter, we will set up the notation and introduce the main concepts to be used throughout the thesis.

**Chapter 3: Scaling the Learning Process via Combinatorial Search.** In this chapter, we will consider the *classical* LTL learning problem, where the goal is to learn a concise

LTL formula from positive and negative examples. To this end, we will devise a scalable algorithm that searches through fragments of LTL via efficient combinatorial search.

**Chapter 4: Learning in the presence of Noise.** Since noise is ubiquitous in real-world data, in this chapter, we will consider the problem of learning LTL and STL formulas when the input data contains noise. We will present learning algorithms that are robust to noise by allowing formulas that tolerate a bounded amount of misclassifications.

**Chapter 5: Incorporating Intuition as Specification Sketches.** Often, designers have a high-level intuition of the desired property for their system. In this chapter, we will formalize such intuition as a *sketch* and consider the problem of learning LTL formulas based on a sketch. We will investigate results that indicate when a sketch proves useful and devise algorithms for incorporating sketches in the learning process.

**Chapter 6: Learning from Positive Examples Only.** Negative examples are hard to observe or design in many safety-critical applications. Thus, in this chapter, we will consider the problem of learning LTL formulas and DFAs from positive examples only. We will argue why learning from positive examples is typically an *ill-posed* problem and propose some effective means to learn meaningful models in this setting.

**Chapter 7: Learning Properties in Property Specification Language.** In this chapter, we will consider the problem of learning temporal properties in PSL, a formalism that augments LTL with regular expressions and is consequently more expressive than LTL. We will adapt existing learning techniques for LTL to work for PSL.

**Chapter 8: Learning Properties in Continuous-Time Logics.** Many applications related to cyber-physical systems heavily rely on the continuous time-logic MTL. Thus, in this chapter, we will consider the problem of learning temporal properties in MTL. We will devise some learning techniques specifically designed to be useful for verification purposes.

**Chapter 9: Conclusion and Future Works.** In this final chapter, we will summarize the contributions of the thesis in both Explainable AI and Formal Verification. We will also discuss ideas and insights that will provide directions for future work.

## 1.2 List of Publications

The content of the PhD thesis is based on the following original published research:

1. "Learning Interpretable Models in the Property Specification Language.", with Dana Fisman and Daniel Neider, In the 29th International Joint Conference on Artificial Intelligence, IJCAI 2020 [195].

2. "Learning Linear Temporal Properties from Noisy Data: A MaxSAT-Based Approach.", with Jean-Raphaël Gaglione, Daniel Neider, Ufuk Topcu, and Zhe Xu, In Automated

Technology for Verification and Analysis - 19th International Symposium, ATVA 2021 [93].

3. "MaxSAT-based temporal logic inference from noisy data.", with Jean-Raphaël Gaglione, Daniel Neider, Ufuk Topcu, and Zhe Xu, In Innovations in Systems and Software Engineering, ISSE 2022 [94].

4. "Scalable Anytime Algorithms for Learning Fragments of Linear Temporal Logic", with Ritam Raha, Nathanaël Fijalkow and Daniel Neider, In Tools and Algorithms for the Construction and Analysis of Systems - 28th International Conference, TACAS 2022 [187].

5. "Learning Interpretable Temporal Properties from Positive Examples Only", with Jean-Raphaël Gaglione, Nasim Baharisangari, Daniel Neider, Zhe Xu and Ufuk Topcu, In the 37th AAAI Conference on Artificial Intelligence, AAAI 2023 [196].

6. "Specification Sketching for Linear Temporal Logic", with Simon Lutz and Daniel Neider, In Automated Technology for Verification and Analysis - 21st International Symposium, ATVA 2023 [155].

7. "Scarlet: Scalable Anytime Algorithms for Learning Fragments of Linear Temporal Logic", with Ritam Raha, Nathanaël Fijalkow and Daniel Neider, In the Journal of Open Source Software, JOSS 2023 [188].

8. "Synthesizing Efficiently Monitorable Formulas in Metric Temporal Logic", with Ritam Raha, Nathanaël Fijalkow, Daniel Neider and Guillermo A. Perez, In Verification, Model Checking, and Abstract Interpretation - 25th International Conference, VMCAI 2024 [190].

All the publications are the result of collaborations with many great researchers. Hence, throughout the thesis, I will describe the results using the first-person plural (for instance, 'we' instead of 'I', 'our' instead of 'my', etc.).

During my PhD, I also contributed to other projects related to explainability and verification of neural networks [33, 19, 134, 132, 133, 230]. While I provide references for interested readers, I have not included these in the thesis to maintain a focused discussion on learning of temporal properties.

# Chapter 2

# Basics of Learning Temporal Properties

In this chapter, we will introduce the background knowledge required to understand the technical contributions of the thesis. We first familiarize the readers with the necessary concepts from formal language theory and mathematical logic.

## 2.1 Words, Traces, and Languages

Let $\mathbb{N} = \{0, 1, 2, 3, \dots\}$ be the set of natural numbers, $\mathbb{R}$ be the set of real numbers, and $\mathbb{B} = \{0, 1\}$ be the set of Boolean numbers.

### 2.1.1 Words.

Unless specified otherwise, in this thesis, we model system executions as words over an alphabet consisting of relevant system events. Formally, an *alphabet* $\Sigma$ is a non-empty, finite set whose elements are called *symbols*.

A *finite word* over $\Sigma$ is a finite sequence $u = a_0 \dots a_n$ where $a_i \in \Sigma$, $i \in \{0, \dots, n\}$. The *empty word* $\varepsilon$ is the empty sequence. The length $|u|$ of a finite word $u$ is the number of its symbols (note that $|\varepsilon| = 0$). We define $u[i] = a_i$ where $i \in \{0, \dots, n\}$ to be the symbol at position $i$. Moreover, we define $u[i, j) = a_i \dots a_{j-1}$ where $i, j \in \{0, \dots, n+1\}$ to be the finite *infix* of $u$ starting from position $i$ up to (and excluding) position $j$ (note that $u[i, i) = \varepsilon$). The set of all finite words over $\Sigma$ is denoted by $\Sigma^*$, and the set of non-empty finite words over $\Sigma$ is denoted by $\Sigma^+ = \Sigma^* \setminus \{\varepsilon\}$.

An *infinite word* over $\Sigma$ is an infinite sequence $\alpha = a_0 a_1 \dots$, where $a_i \in \Sigma$, $i \in \mathbb{N}$. The set of all infinite words over $\Sigma$ is denoted by $\Sigma^\omega$. The infinite word $u^\omega = uuu \dots$ where $u \in \Sigma^+$ is called the *infinite repetition* of $u$. Moreover, an infinite word $\alpha$ is said to be *ultimately periodic* if it is of the form $\alpha = uv^\omega$, where $u$ and $v$ are finite words such that $u \in \Sigma^*$ and $v \in \Sigma^+$. We define the length of an ultimately periodic word $uv^\omega$ as the sum $|u| + |v|$ of the length of the finite words $u$ and $v$.

Analogous to finite words, we define $\alpha[i] = a_i$ to be the symbol at position $i$ and $\alpha[i, j) = a_i \dots a_{j-1}$ to be the finite infix of $\alpha$ starting from position $i$ up to (and excluding) position $j$, where $i, j \in \mathbb{N}$. Additionally, we define $\alpha[i, \infty) = a_i a_{i+1} \dots$, where $i \in \mathbb{N}$, to be the infinite *suffix* of $\alpha$ starting from position $i$.

### 2.1.2 Traces.

We represent relevant events of the underlying system using a finite set $\mathcal{P}$ of atomic *propositions*. Further, we formalize the system executions involving the relevant events as traces. Formally, a *trace* is a word over the alphabet $\Sigma = 2^{\mathcal{P}}$ consisting of all possible subsets of $\mathcal{P}$. A trace can be either finite or infinite, depending on whether the execution is finite or infinite. Formally, a finite trace is an element of the set $(2^{\mathcal{P}})^*$, while an infinite trace is an element of the set $(2^{\mathcal{P}})^\omega$.

### 2.1.3 Languages.

A language is a set of words over an alphabet $\Sigma$. Specifically, a language of finite words is a subset of $\Sigma^*$, while a language of infinite words is a subset of $\Sigma^\omega$. For all languages $L_1$, $L_2$, we allow the standard set operations such as union $L_1 \cup L_2$, intersection $L_1 \cap L_2$, complement $L_1^c$, and set difference $L_1 \setminus L_2$ and so on, and the standard set relations such as subset $L_1 \subseteq L_2$, proper subset $L_1 \subset L_2$, and so on.

For any model of computation $M$, the *language* $L(M)$ of $M$ is the set of words that $M$ allows or accepts. The expressive power of a *class* of models is the set of languages that the class can capture. Formally, the *expressive power* of a class $\mathcal{M}$ is defined as $\mathcal{L}(\mathcal{M}) = \{L \subseteq \Sigma^* \mid L = L(M) \text{ for some } M \in \mathcal{M}\}$.

## 2.2 Formal Logic

### 2.2.1 Propositional Boolean Logic.

Several learning techniques described in this thesis rely on satisfiability problems—the mathematical basis for which is propositional Boolean logic or simply *propositional logic*. Propositional logic is built upon a finite set $\mathcal{X}^{\mathbb{B}}$ of propositional variables. These variables take Boolean values $\mathbb{B} = \{0, 1\}$ (0 represents *false* and 1 represents *true*).

Mathematically, formulas in propositional logic—denoted by capital Greek letters—are defined inductively as follows:

$$\Phi := x \in \mathcal{X}^{\mathbb{B}} \mid \neg\Phi \mid \Phi_1 \vee \Phi_2, \tag{2.1}$$

where $\neg$ denotes the negation operator and $\vee$ the disjunction operator. As syntactic sugar, we include the following formulas in propositional logic: $\mathit{true} := x \vee \neg x$, $\mathit{false} := \neg(x \vee \neg x)$, $\Phi_1 \wedge \Phi_2 := \neg(\neg\Phi_1 \vee \neg\Phi_2)$, $\Phi_1 \rightarrow \Phi_2 := \neg\Phi_1 \vee \Phi_2$ and $\Phi_1 \leftrightarrow \Phi_2 := (\neg\Phi_1 \vee \Phi_2) \wedge (\neg\Phi_1 \vee \Phi_2)$, where $\wedge$ denotes the conjunction operator, $\rightarrow$ the implication operator and $\leftrightarrow$ the equivalence operator.

An *assignment in propositional logic* is a mapping $v\colon \mathcal{X}^{\mathbb{B}} \mapsto \{0, 1\}$, which assigns propositional variables to Boolean values. Based on an assignment $v$, we define the semantics of propositional logic using a valuation function $V(\Phi, v)$. This function is defined inductively

as follows:

$$V(x, v) = v(x) \tag{2.2}$$

$$V(\neg\Phi, v) = 1 - V(\Phi, v), \tag{2.3}$$

$$V(\Phi_1 \vee \Phi_2, v) = \max\{V(\Phi_1, v), V(\Phi_2, v)\}. \tag{2.4}$$

In the case that $V(\Phi, v) = 1$, we say that $v$ *satisfies* $\Phi$ and call $v$ a *model* or *satisfying assignment* $\Phi$. Moreover, a propositional formula $\Phi$ is said to be *satisfiable* if there exists a model $v$ of $\Phi$.

The satisfiability problem of propositional logic, abbreviated as SAT, is the problem of determining whether a propositional formula is satisfiable or not. The SAT problem is well-known to be an NP-complete problem. Despite that, due to its relevance in practical applications, there have been concentrated efforts to solve SAT effectively, as evidenced by annual SAT competitions [16]. Present-day tools designed for handling SAT—referred to as SAT solvers—implement optimized decision procedures that are capable of checking satisfiability propositional formulas with millions of variables [163, 57, 18]. In fact, virtually all SAT solvers can even return a model for a given satisfiable formula.

### 2.2.2 First-order Logic of Reals.

Some learning techniques in this thesis rely on a specific fragment of first-order logic—linear real arithmetic (LRA)—which we introduce now. This logic, in contrast to propositional logic, is built upon a set $\mathcal{X}^{\mathbb{R}}$ of variables that take values in $\mathbb{R}$. In addition to variables, formulas in LRA rely on real-valued constants, functions $\{+, \cdot\}$, and predicates $\{\leq, <, =, >, \geq\}$.

To define formulas in LRA, we first introduce *terms*, which are defined inductively as follows:

$$t := \; c \in \mathbb{R} \mid x \in \mathcal{X}^{\mathbb{R}} \mid c \cdot t \mid t_1 + t_2, \tag{2.5}$$

where $+$ denotes the addition operator and $\cdot$ the multiplication operator. Based on standard convention, we drop the multiplication operator while writing terms. Some examples of terms are $5$, $x$, and $3x + 2y$.

*Atomic formulas* combine terms by applying predicates. Precisely, atomic formulas are of the form $t_1 \circ t_2$ where $t_1$ and $t_2$ are terms and $\circ \in \{\leq, <, =, >, \geq\}$ a predicate. Some examples of atomic formulas are $x = 5$, $3x + 2y > 5$, and $3x + 2y \leq 5x$. *Formula in LRA*—also denoted by capital Greek letters[1]— are defined inductively as follows:

$$\Phi := A \mid \neg\Phi \mid \Phi_1 \vee \Phi_1, \tag{2.6}$$

where $A$ denotes atomic formulas. Analogous to propositional logic, we add standard syntactic sugar: $\Phi_1 \wedge \Phi_2$, $\Phi_1 \rightarrow \Phi_2$, and $\Phi_1 \leftrightarrow \Phi_2$.

[1]To avoid notational clutter, we reuse some notations for both propositional and first-order formulas; we explicitly specify which logic while using them.

Similar to propositional logic, an *assignment in LRA* is a mapping $v\colon \mathcal{X}^{\mathbb{R}} \to \mathbb{R}$, which assigns variables to real values. We can easily lift an assignment $v$ to terms inductively as: $v(c) = c$, $v(c \cdot t) = v(c) \cdot v(t)$, and $v(t_1 + t_2) = v(t_1) + v(t_2)$. Based on an assignment $v$, we define the semantics of LRA using a valuation function $V(\Phi, v)$, defined inductively as follows:

$$V(t_1 \circ t_2, v) = \mathbb{1}\{v(t_1) \circ v(t_2)\} \text{ where } \circ \in \{\leq, <, =, >, \geq\} \tag{2.7}$$

$$V(\neg\Phi, v) = 1 - V(\Phi, v) \tag{2.8}$$

$$V(\Phi_1 \vee \Phi_2, v) = \max\{V(\Phi_1, v), V(\Phi_2, v)\} \tag{2.9}$$

where $\mathbb{1}\{R\}$ is 1 if the relation $R$ holds, otherwise 0. Similar to propositional logic, if $V(\Phi, v) = 1$, we say that $v$ *satisfies* $\Phi$ and $v$ is a model for $\Phi$. Also, a formula $\Phi$ is *satisfiable* if there exists a model for it.

The satisfiability problem for formulas in Linear Real Arithmetic (LRA) is of immense practical significance. Consequently, SAT solvers are frequently enhanced with optimized decision procedures for Satisfiability Modulo Theories (SMT) [142]. This augmentation enables checking satisfiability of formulas not only in LRA but also in various other (typically quantifier-free) fragments of first-order logic, also known as theories.

### 2.2.3 Linear Temporal Logic.

We now introduce the most commonly used temporal logic in this thesis—Linear Temporal Logic (LTL). *Linear Temporal Logic* (LTL) is a logic that enables reasoning about sequences of events by extending propositional logic with temporal modalities.

**Syntax.** Given a finite set $\mathcal{P}$ of propositions, formulas in LTL—denoted by small Greek letters—are defined inductively as:

$$\varphi := p \in \mathcal{P} \mid \neg\varphi \mid \varphi_1 \vee \varphi_2 \mid \mathsf{X}\,\varphi \mid \varphi_1 \,\mathsf{U}\, \varphi_2, \tag{2.10}$$

where $\mathsf{X}$ denotes the next operator and $\mathsf{U}$ the until operator. As syntactic sugar, we allow formulas used in propositional logic such as *true*, *false*, $\varphi_1 \wedge \varphi_2$, and $\varphi_1 \rightarrow \varphi_2$. Additionally, we include formulas based on temporal operators $\mathsf{F}$ (Finally) operator and $\mathsf{G}$ (Globally) operators, defined as follows: $\mathsf{F}\,\varphi := \mathit{true}\,\mathsf{U}\,\varphi$ and $\mathsf{G}\,\varphi := \neg\,\mathsf{F}\,\neg\varphi$. We define $\Lambda = \{\neg, \vee, \wedge, \rightarrow, \mathsf{X}, \mathsf{F}, \mathsf{G}, \mathsf{U}\}$ to be the set of all operators, $\Lambda_U = \{\neg, \mathsf{X}, \mathsf{F}, \mathsf{G}\}$ to be the set of unary operators and $\Lambda_B = \{\vee, \wedge, \rightarrow, \mathsf{U}\}$ to be the set of binary operators. We also define the set of all LTL formulas as $\mathcal{F}_{LTL}$.

We define the set $\mathit{subf}(\varphi)$ of subformulas of an LTL formula $\varphi$ inductively as follows:

$$\mathit{subf}(p) = \{p\} \tag{2.11}$$

$$\mathit{subf}(\circ\varphi) = \{\circ\varphi\} \cup \mathit{subf}(\varphi) \text{ for } \circ \in \Lambda_U \tag{2.12}$$

$$\mathit{subf}(\varphi_1 \circ \varphi_2) = \{\varphi_1 \circ \varphi_2\} \cup \mathit{subf}(\varphi_1) \cup \mathit{subf}(\varphi_2) \text{ for } \circ \in \Lambda_B. \tag{2.13}$$


(a) Syntax Tree (b) Syntax DAG

FIGURE 2.1: Representations of LTL formula $\varphi = (p \,\mathsf{U}\, \mathsf{G}\, q) \vee \mathsf{X}(\mathsf{G}\, q)$

As an example, the set of subformulas for $\varphi = (p \,\mathsf{U}\, \mathsf{G}\, q) \vee \mathsf{X}(\mathsf{G}\, q)$ is $subf(\varphi) = \{(p \,\mathsf{U}\, \mathsf{G}\, q) \vee \mathsf{X}(\mathsf{G}\, q), (p \,\mathsf{U}\, \mathsf{G}\, q), \mathsf{X}(\mathsf{G}\, q), \mathsf{G}\, q, p, q\}$. We define the size $|\varphi|$ of a formula $\varphi$ as the number $|subf(\varphi)|$ of its subformulas. For instance, the size of $\varphi = (p \,\mathsf{U}\, \mathsf{G}\, q) \vee \mathsf{X}(\mathsf{G}\, q)$ is 6.

We often view LTL (and other temporal logic) formulas using their syntax DAG representation. A *syntax DAG* is a syntax tree in which common subformulas are merged. Figure 2.1 illustrates the syntax tree (Figure 2.1a) and the syntax DAG (Figure 2.1b) representations of the formula $\varphi = (p \,\mathsf{U}\, \mathsf{G}\, q) \vee \mathsf{X}(\mathsf{G}\, q)$. Following the definition of syntax DAGs, the number of nodes in the syntax DAG of an LTL formula coincides with its size.

LTL is typically interpreted over traces. The exact interpretation of LTL depends on whether the traces are finite or infinite. Several of our learning techniques can handle both interpretations, and hence, we introduce them both. To distinguish between the interpretations, LTL over finite traces is often written as $\text{LTL}_\text{f}$ while LTL over infinite traces is simply written as LTL.

**Infinite semantics.** The semantics of LTL over infinite traces [180] is defined using a satisfaction relation denoted by $\models$. For an infinite trace $u$, a timepoint $t \in \mathbb{N}$ and an LTL formula $\varphi$, the relation $u, t \models \varphi$ denotes that $u$ satisfies $\varphi$ at timepoint $t$ or, alternatively, $\varphi$ holds on $u$ at timepoint $t$. It is defined inductively as follows:

$$u, t \models p \text{ if and only if } p \in u[t] \tag{2.14}$$

$$u, t \models \neg\varphi \text{ if and only if } u, t \not\models \varphi \tag{2.15}$$

$$u, t \models \varphi_1 \vee \varphi_2 \text{ if and only if } u, t \models \varphi_1 \text{ or } u, t \models \varphi_2 \tag{2.16}$$

$$u, t \models \mathsf{X}\, \varphi \text{ if and only if } u, t+1 \models \varphi \tag{2.17}$$

$$u, t \models \varphi_1 \,\mathsf{U}\, \varphi_2 \text{ if and only if for some } t \leq t' : u, t' \models \varphi_2 \text{ and for all } t \leq t'' < t : u, t'' \models \varphi_1. \tag{2.18}$$

In the case that $u, 0 \models \varphi$, we simply write $u \models \varphi$ and say that $u$ satisfies $\varphi$, or alternatively, $\varphi$ holds on $u$.

**Finite semantics.** The semantics of LTL over finite traces [67] is defined using a satisfaction relation denoted by $\models_\text{f}$. For a finite trace $u$, a timepoint $t < |u| \in \mathbb{N}$ and an $\text{LTL}_\text{f}$ formula $\varphi$, the relation $u, t \models_\text{f} \varphi$ denotes that $u$ satisfies $\varphi$ at timepoint $t$ or, alternatively, $\varphi$ holds on $u$

at timepoint $t$. The definition of $\models_{\mathrm{f}}$ differs from that of $\models$ only for the temporal operators, which we describe below.

$$u, t \models_{\mathrm{f}} \mathsf{X}\,\varphi \text{ if and only if } t < |u| \text{ and } u, t+1 \models \varphi \tag{2.19}$$

$$u, t \models_{\mathrm{f}} \varphi_1 \,\mathsf{U}\, \varphi_2 \text{ if and only if for some } t \leq t' < |u| : u, t' \models \varphi_2 \text{ and for all } t \leq t'' < t' : u, t'' \models \varphi_1. \tag{2.20}$$

Similar to the infinite case, if $u, 0 \models_{\mathrm{f}} \varphi$, we write $u \models \varphi$ and say that $u$ satisfies $\varphi$, or alternatively, $\varphi$ holds on $u$.

Finally, we define the language of a formula $\varphi$ as the set of traces that satisfies $\varphi$. Formally, for an LTL formula $\varphi$, it is defined as $L(\varphi) = \{u \in (2^{\mathcal{P}})^{\omega} \mid u \models \varphi\}$, and for an $\mathrm{LTL_f}$ formula, it is defined as $L(\varphi) = \{u \in (2^{\mathcal{P}})^{*} \mid u \models_{\mathrm{f}} \varphi\}$.

## 2.3 Finite State Automata

While typically finite state automata are defined over (general) words, in this thesis, we define automata over traces to unify the notation with temporal logic.

### 2.3.1 Automata over finite traces.

A *non-deterministic finite automaton* (NFA) is a tuple $\mathcal{A} = (Q, 2^{\mathcal{P}}, \Delta, Q_I, F)$ where $Q$ is a finite set of states, $2^{\mathcal{P}}$ the alphabet, $Q_I \subseteq Q$ the set of initial states, $F \subseteq Q$ the set of final states, and $\Delta \subseteq Q \times \Sigma \times Q$ the transition relation.

Given a trace $u = a_0 \dots a_n \in (2^{\mathcal{P}})^{*}$, a run $\rho$ of an NFA $\mathcal{A}$ on $u$ is a sequence of states and symbols $q_0 a_0 q_1 \cdots q_n a_n q_{n+1}$, such that $q_0 \in Q_I$ and for each $t \in \{0, \cdots, n\}$, $(q_t, a_t, q_{t+1}) \in \Delta$. An NFA $\mathcal{A}$ is said to *accept* a trace $u$ if some run of $\mathcal{A}$ on $u$ ends in a final state, that is, a state in $F$. The language of an NFA $\mathcal{A}$ is defined as $L(\mathcal{A}) = \{u \in (2^{\mathcal{P}})^{*} \mid \mathcal{A} \text{ accepts } u\}$.

A *deterministic finite automaton* (DFA) is an NFA in which for each state $q \in Q$ and symbol $a \in 2^{\mathcal{P}}$, there is exactly one state $q' \in Q$ with a transition $(q, a, q') \in \Delta$. Thus, for each trace $u$, the run $\rho$ of a DFA $\mathcal{A}$ on $u$ is unique.

The language of an NFA (or an DFA) $\mathcal{A}$ is defined as $L(\mathcal{A}) = \{u \in (2^{\mathcal{P}})^{*} \mid \mathcal{A} \text{ accepts } u\}$. It is well-known that the expressive power of DFAs and NFAs are the same [202]. Further, the expressive power of DFAs/NFAs is strictly more than that of $\mathrm{LTL_f}$ [67].

### 2.3.2 Automata over infinite traces.

A *non-deterministic Büchi automaton* (NBA) is a tuple $\mathcal{A} = (Q, 2^{\mathcal{P}}, \Delta, Q_I, F)$ that is syntactically identical to an NFA.

In contrast to NFAs, NBAs accept infinite traces, and thus, a run of an NBA is an infinite sequence. Formally, given an infinite trace $u = a_0 a_1 \ldots \in (2^{\mathcal{P}})^{\omega}$, a run $\rho$ of an NBA $\mathcal{A}$ on $u$ is a infinite sequence of states and symbols $q_0 a_0 q_1 a_1 q_2 \cdots$, such that $q_0 \in Q_I$ and for $t \in \mathbb{N}$, $(q_t, a_t, q_{t+1}) \in \Delta$. The set $inf(\rho)$ is the set of states that appear infinitely often in a run $\rho$. An

NFA $\mathcal{A}$ is said to *accept* a trace $u$ if there is some run $\rho$ of $\mathcal{A}$ on $u$ in which a final state occurs infinitely often, that is, $inf(\rho) \cap F \neq \emptyset$.

A *deterministic Büchi automaton* (DBA) is an NBA in which for each state $q \in Q$ and symbol $a \in 2^{\mathcal{P}}$, there is exactly one state $q' \in Q$ with a transition $(q, a, q') \in \Delta$. Also, for each trace $u$, the run $\rho$ of a DBA $\mathcal{A}$ on $u$ is unique.

The language of an NBA (or an DBA) $\mathcal{A}$ is defined as $L(\mathcal{A}) = \{u \in (2^{\mathcal{P}})^{\omega} \mid \mathcal{A} \text{ accepts } u\}$. It is known that the expressive power of NBAs is strictly more than that of LTL [210, 91] and DBAs [209]. The expressive powers of LTL and DBAs are incomparable.

## 2.4 Background on Learning Temporal Properties

The learning problems that we consider in the thesis can be broadly classified as passive learning. *Passive learning*, stated generally, is the following: given labeled input data, the goal is to find a concise model[2] that fits the data. The input data will usually consist of traces with their labels indicating whether the traces are accepted (that is, positive) or rejected (that is, negative) by the prospective model.

For passive learning, the models of computation that have been of primary focus in the past are finite automata. There is, in fact, a vast literature on learning algorithms for finite automata with significant tool support [184, 35, 165]. Most of the works have focussed on deterministic finite automata (DFAs) [30, 174, 108, 116] and non-deterministic finite automata (NFAs) [68, 62] for finite words, and Büchi automata [32] for infinite words (see Neider [167] and López et al. [151] for extensive related work).

### 2.4.1 Comparison to Automata Learning

While we discuss a few algorithms for learning automata, the primary focus of this thesis will be on learning formulas in temporal logics. We believe that learning temporal logic substantially complements the existing work on learning automata in the general area of passive learning. To bolster this fact, we first highlight the differences in general properties of temporal logic and automata as models of computation. We then argue why it might be beneficial to develop learning algorithms for temporal logic if the application prefers temporal logic.

**Function.** Temporal logics and finite automata, on a conceptual level, serve different purposes. Temporal logics are considered to be *declarative* models for expressing temporal properties: one typically formalizes the intuitive understanding of a property as a temporal logic formula. In fact, temporal logics were developed with the intention of formalizing the temporal aspects of natural language unambiguously (see Goranko et al. [105] for a summary). Finite automata, on the other hand, are considered to be *operational* models for expressing temporal properties: one typically represents all possible behaviors allowed by a property as a compact state machine. Finite automata

[2]The considered models, typically, have a natural notion of size

exploit techniques, such as using loops within states, to concisely capture all possible behaviors.

**Interpretability.** Before we dive into details about interpretability, we emphasize that it is a highly subjective issue. Interpretability often depends on the familiarity and expertise of a user with the particular model at hand. To our knowledge, there is no well-established measure that can compare the interpretability of temporal logics and finite automata. One could possibly conduct psychological studies with human users to view their perception of the models. However, such studies are beyond the scope of this thesis. Nevertheless, we present certain evidence that demonstrates the resemblance of temporal logic to natural language (which may or may not be a proper notion of interpretability) as compared to finite automata.

First, we consider some $\text{LTL}_\text{f}$ formulas[3] that are commonly used in practice. For these formulas, we wrote a possible compact natural language (NL) translation and constructed their equivalent minimal DFAs. We present all the results in Figure 2.2. We ensured that the equivalent DFAs are minimal using the tool LTLf2DFA [89].

From the NL translations in Figure 2.2, we observe that it is often possible to translate $\text{LTL}_\text{f}$ formulas into English by directly replacing an LTL operator with a word (or combination of words) that resembles its function. For instance, one can replace Boolean operators $\vee$, $\wedge$, $\neg$, and $\rightarrow$ with 'or', 'and', 'does not', and 'if-then', respectively. One can translate temporal operators $\mathsf{G}$, $\mathsf{F}$, and $\mathsf{U}$ with 'always' (or 'never' if a negation appears), 'eventually', and 'until,' respectively. While the NL translations can become complex and ambiguous as the formulas become large (and involve nesting of operators), they can still provide some insights into the meaning of the formula.

From the equivalent DFAs in Figure 2.2, we observe that for formulas in Figures 2.2a, 2.2b, and 2.2c, which are rather small, one could view the equivalent DFAs to be similar to their NL translations. For formulas in Figures 2.2d, 2.2e, 2.2f and 2.2g, which are larger than the previous formulas, the equivalent DFAs may seem complex as the DFAs consist of multiple states and transitions serving different roles. The formula in Figure 2.2h, despite being larger than all the considered formulas, has a compact equivalent DFA. This DFA exploits several loops to concisely represent the possible behaviors of the $\text{LTL}_\text{f}$ formula, making it appear significantly different from its NL translation.

Due to the resemblance of NL and temporal logic, plethora of tools have been designed to translate temporal logic to natural language and vice-versa. Early works [85, 172, 98, 102] translated NL to temporal logic formulas by efficiently parsing English sentences. As data-driven techniques became pervasive, several neural-network based techniques [173, 110, 55] started relying on human-labeled NL-formula pairs. Currently, many works [61, 177, 150, 90] are utilizing the impressive understanding of

[3]The $\text{LTL}_\text{f}$ formulas are based on common LTL patterns [77] that use at most two propositions.

natural language by Large Language Models (LLMs) to further boost the translation capabilities.

In contrast, to the best of our knowledge, there are no such tools that directly translate finite automata to natural language. Finite automata are better employed as useful tools in natural language processing (NLP) (follow the FSMNLP conferences [217]).

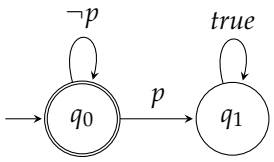
**Succinctness.** Some properties when represented in temporal logic are more succinct (in terms of size) than they are when expressed as finite automata. It is known that the $\text{LTL}_\text{f}$ formulas can be double-exponentially more succinct than their equivalent DFAs [100] and LTL formulas can be exponentially more succinct than their equivalent NBAs [143]. For instance, a simple $\text{LTL}_\text{f}$ formula that admits a large DFA is the formula $\varphi(n) := \mathsf{G}(p \rightarrow \mathsf{X}^n(q))$ which roughly translates to "always, if $p$ holds, $n$ time-steps later $q$ holds". While $\varphi(n)$ is of size $\mathcal{O}(n)$, one can show that the equivalent minimal DFA $\mathcal{A}_{\varphi(n)}$ is of size at least $\mathcal{O}(2^n)$, since intuitively $\mathcal{A}_{\varphi(n)}$ needs states to represent all possible $2^n$ behaviors in $n$ time-steps.

As we see above, temporal logics and finite automata are distinct formalisms, each having their unique characteristics and applicative nuances. We now argue that if the application demands formulas in temporal logic, then it is more convenient to use an algorithm that learns temporal logic instead of the *standard* automata learning algorithms. This is primarily because learning automata and then converting them to their equivalent temporal logic formulas is often either not possible or results in large formulas. We expand on these reasons below.

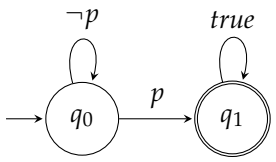
**Expressivity.** The expressive powers of finite automata and temporal logics are often very different. DFAs/NFAs (respectively, NBAs) are more expressive that $\text{LTL}_\text{f}$ (respectively, LTL) [67, 91, 210]. Continuous-time logics (such as MTL and STL) are often not known to have automata models that have the same expressive power as them.

We, however, note that there are some well-formed (formally, counter-free) finite automata with certain syntactic restrictions that have the same expressive power as $\text{LTL}\backslash\text{LTL}_\text{f}$ and can be used to learn formulas [48]. However, such automata models are by no means common in the automata learning literature. Thus, we can say that automata obtained through *standard* automata learning may not have an equivalent representation in the desired temporal logic.

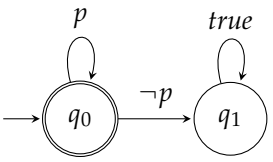
**Shape and size.** Even if one could learn automata that can be converted to temporal logic formulas, it may not yield small formulas. For instance, converting a counter-free DFA—which belongs to a subclass of DFAs that have the same expressive power as $\text{LTL}_\text{f}$—has a double exponential blow-up in size [221]. Also, such conversions often lead to formulas of a specific syntactic structure, which may not be always desirable. In fact, several questions about the shape and size of the converted temporal logic formulas are still open [34]. Thus, a priori, it is not clear whether converting automata to temporal logic formulas would result in small and easy-to-understand formulas.



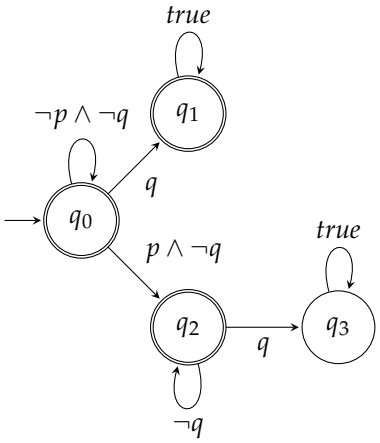

(a) $\mathsf{G}(\neg p)$: $p$ never holds.



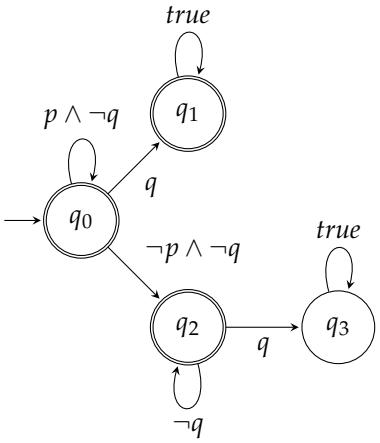

(b) $\mathsf{F}(p)$: $p$ holds eventually.



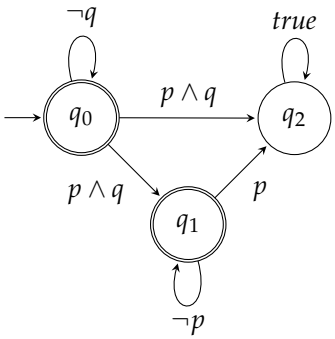

(c) $\mathsf{G}(p)$: $p$ always holds.



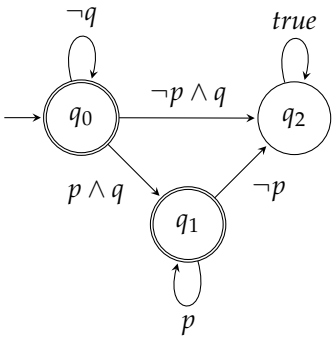

(d) $\mathsf{F}(q) \rightarrow (\neg p \,\mathsf{U}\, q)$: If $q$ holds eventually, then $p$ does not hold until $q$ holds.



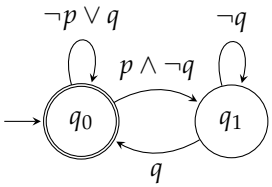

(e) $\mathsf{F}(q) \rightarrow (p \,\mathsf{U}\, q)$: If $q$ holds eventually, then $p$ holds until $q$ holds.




(f) $\mathsf{G}(q \rightarrow \mathsf{G}(\neg p))$: Always, if $q$ holds, then $p$ never holds.




(g) $\mathsf{G}(q \rightarrow \mathsf{G}(p))$: Always, if $q$ holds, then $p$ always holds.




(h) $\mathsf{G}((\neg p \vee \mathsf{F}(p \wedge F(q)))$: Always, either $p$ does not hold or if $p$ holds then q holds eventually.

FIGURE 2.2: Commonly used $\text{LTL}_\text{f}$ formulas along with a compact natural language (NL) translation and equivalent minimal DFAs. We present the equivalent DFAs as subfigures, while the $\text{LTL}_\text{f}$ formulas, along with their NL translations, are in the subcaptions.

Again, there are some syntactically restricted automata which have similar shape and size as their temporal logic equivalent; for instance, counter-free alternating finite automata (AFA) has same size as its $\text{LTL}_\text{f}$ equivalent. However, such automata models are not typical in automata learning and often specifically used for learning formulas [48].

Overall, we posit that the learning algorithms for temporal logic complement existing research in automata learning, thereby enhancing the general area of passive learning.

# Chapter 3

# Scaling the Learning Process via Combinatorial Search.

In this chapter, we consider the standard *passive learning* problem for $\text{LTL}_\text{f}$, which asks to infer explainable models in the form of an $\text{LTL}_\text{f}$ formula from system executions. Specifically, given a sample $\mathcal{S}$ of system executions partitioned into two finite sets of positive and negative examples, the goal is to learn a concise $\text{LTL}_\text{f}$ formula $\varphi$ which satisfies all the positive examples and none of the negative examples. In that case, we say the formula $\varphi$ is *consistent* with the given sample. Based on machine learning terminology, the formula $\varphi$ can be thought of as a perfect classifier (that is, a classifier with full accuracy) of the sample $\mathcal{S}$. Since we search for a perfect classifier, this setting is also often known as *exact* learning. We refer to Section 3.1 for further details on the problem.

As also described in Chapter 1, this learning problem for $\text{LTL}_\text{f}$ has significant applications in the domains of Explainable AI and Formal verification including specification mining [145], fault detection [37], etc. (see Camacho et al. [47] for a comprehensive list of applications). As a result, in recent years, several approaches have been proposed for learning $\text{LTL}\backslash\text{LTL}_\text{f}$ formulas. Some approaches leverage SAT (or SMT) solving, utilizing either the syntax-DAG representation of $\text{LTL}\backslash\text{LTL}_\text{f}$ [169, 193, 8] or the alternating automata-based representation of $\text{LTL}_\text{f}$ [47]. Some other approaches exploit Bayesian inference [135, 200], although restricted to learning certain LTL templates. Further, recent approaches exploit deep learning based on various neural-network architectures such as GNNs [153], RNNs [88], etc.; they have strong empirical performance, but are unable to guarantee perfect classification.

Existing approaches for exact learning[1] do not scale beyond formulas of size 10, making them hard to deploy for industrial cases. A second serious limitation is that they often exhaust computational resources without returning any results. Indeed, theoretical studies [83, 39, 159] have shown that learning minimal $\text{LTL}\backslash\text{LTL}_\text{f}$ formulas (for many of its subclasses) is NP-hard, explaining the difficulties found in practice.

To address both issues, we design an algorithm that exploits two features: *approximation* and *anytime*. By *approximation*, we mean that (i) our algorithm does not always produce a minimal formula, although it does produce a formula that is consistent with the input sample

[1] When this research was done, existing works for exact learning relied predominately on constraint solving [169, 193, 47]. Some subsequent works [124, 97] explored other approaches.

and (ii) it targets a strict fragment of $\mathrm{LTL_f}$, which does not contain the Until operator (nor its dual Release operator).

By *anytime*, we mean that our algorithm keeps finding better and better solutions the longer it runs; in other words, it works by refining solutions. As we will see in the experiments, due to the anytime feature, our algorithm may yield some good albeit non-optimal formula even if it times out.

Our algorithm combines two ingredients:

- *Searching for directed-LTL formulas*: we define a space-efficient dynamic programming algorithm for enumerating formulas from a subclass of $\mathrm{LTL_f}$ that we call directed-LTL.
- *Combining directed-LTL formulas*: we construct two algorithms for combining formulas using Boolean operators. The first is an off-the-shelf *decision tree algorithm*, and the second is a new greedy algorithm called *Boolean subset cover*.

The two ingredients yield two subprocedures: the first one finds directed-LTL formulas of increasing size, which are then fed to the second procedure in charge of combining them into a consistent formula. This yields an anytime algorithm as both subprocedures can output consistent formulas even with a low computational budget and refine them over time; we refer to Section 3.2 for an overview of the algorithm.

To illustrate the subprocedures, we consider a simple scenario from motion planning. Consider an autonomous agent that is deployed to collect wastebin contents from an office and then empty them in a trash container. The environment, let us say, consists of an office o, a hallway h, a container c and a wet area w. We now consider a sample of executions of the agent consisting of the following traces:

$$\begin{aligned} u_1 &= \{\mathrm{h}\}\{\mathrm{h}\}\{\mathrm{h}\}\{\mathrm{h}\}\{\mathrm{o}\}\{\mathrm{h}\}\{\mathrm{c}\}\{\mathrm{h}\} \\ v_1 &= \{\mathrm{h}\}\{\mathrm{h}\}\{\mathrm{h}\}\{\mathrm{h}\}\{\mathrm{h}\}\{\mathrm{c}\}\{\mathrm{h}\}\{\mathrm{o}\}\{\mathrm{h}\}\{\mathrm{h}\} \end{aligned}$$

Based on the agent's performance, the trace $u_1$ is positive, while the trace $v_1$ is negative. Observe that in $v_1$ the agent visits the trash container c before collecting the waste from the office o, which could be a reason why $v_1$ is negative. Thus, the $\mathrm{LTL_f}$ formula $\varphi_1 := \mathsf{F}(\mathrm{o} \wedge \mathsf{F}\,\mathsf{X}\,\mathrm{c})$, which simply fixes the order in which the office o and the container c are visited, is consistent with the above sample. Here, as usual, $\mathsf{F}$-operator stands for finally and $\mathsf{X}$ for next.

The formula $\varphi_1$ is a so-called directed-LTL formula. The first procedure enumerates such directed-LTL formulas of increasing size; we refer to Section 3.3 for the details for this step. The directed-LTL formula $\mathsf{F}(\mathrm{o} \wedge \mathsf{F}\,\mathsf{X}\,\mathrm{c})$ has a small size and, hence, will be generated early on.

Let us now assume that there are two more negative traces in the sample:

$$\begin{aligned} v_2 &= \{\mathrm{h}\}\{\mathrm{h}\}\{\mathrm{h}\}\{\mathrm{h}\}\{\mathrm{h}\}\{\mathrm{o}\}\{\mathrm{w}\}\{\mathrm{c}\}\{\mathrm{h}\}\{\mathrm{h}\}\{\mathrm{h}\} \\ v_3 &= \{\mathrm{h}\}\{\mathrm{h}\}\{\mathrm{h}\}\{\mathrm{h}\}\{\mathrm{h}\}\{\mathrm{w}\}\{\mathrm{o}\}\{\mathrm{w}\}\{\mathrm{c}\}\{\mathrm{w}\}\{\mathrm{w}\} \end{aligned}$$

Observe that in these two traces, the agent visits the wet area w, which could be a reason why they are negative. Thus, an $\text{LTL}_\text{f}$ formula $\varphi_2 := \mathsf{G}(\neg \text{w})$, which simply says that the wet area must never be visited, can explain why $u_1$ is positive while $v_2$ and $v_3$ are not is. As usual, here $\mathsf{G}$ stands for globally.

The formula $\varphi_2$ is a (dual) directed-LTL formula, and the first procedure can detect it early on during its search. Now observe that if the sample has $u_1$ as positive, and $v_1$, $v_2$ and $v_3$ as negative, neither $\varphi_1$ nor $\varphi_2$ individually is consistent with the sample. This is because, $\varphi_1$ satisfies $v_2$ and $v_3$, while $\varphi_2$ satisfies $v_1$.

However, the boolean combination $\varphi_1 \wedge \varphi_2$ works. Towards this, the second procedure obtains $\varphi_1$ and $\varphi_2$ from the first procedure and constructs possible Boolean combinations of them; we refer to Section 3.4 for the details of this step.

We implement our algorithm, with its two subprocedures, in an open-source software—`SCARLET` [188]. We conduct experiments comparing the performance of `SCARLET` against two state-of-the-art tools for learning temporal properties [169, 8]. Further, we test its scalability on large samples and also the benefits of the anytime feature. We refer to Section 3.6 for the details of the experiments and Section 3.7 for a final discussion.

## 3.1 Problem Formulation

As input, we consider a sample $\mathcal{S} = (P, N)$ of finite traces from $(2^{\mathcal{P}})^*$, partitioned into a set $P$ of positive examples and a set $N$ of negative, such that $P \cap N = \emptyset$.

We say an $\text{LTL}_\text{f}$ formula $\varphi$ is *consistent* with a sample $\mathcal{S} = (P, N)$ if $u \models_\text{f} \varphi$ for all positive examples $u \in P$ and $u \not\models_\text{f} \varphi$ for all negative examples $u \in N$. When the sample is clear from the context, we simply say $\varphi$ is consistent.

The problem we consider is the following:

**Problem 1.** *Given a sample $\mathcal{S} = (P, N)$, learn a minimal $\text{LTL}_f$ formula that is consistent with $\mathcal{S}$.*

As mentioned before, we consider an approximate version of the above problem. Specifically, we learn concise formulas in a certain subclass of $\text{LTL}_\text{f}$, helping us to alleviate the theoretical difficulties in learning arbitrary $\text{LTL}_\text{f}$ formulas [83].

To make our search easier, throughout this chapter, we fix $\text{LTL}_\text{f}$ to be in its negation normal form (NNF), where the $\neg$-operator appears only before a proposition; NNF for $\text{LTL}_\text{f}$ is a syntactic restriction that is known not to affect the expressive power. Moreover, we rely on two fragments of $\text{LTL}_\text{f}$: $\text{LTL}_\text{f}(\mathsf{F}, \mathsf{X})$, which includes only $\mathsf{F}$ and $\mathsf{X}$ as temporal operators, and $\text{LTL}_\text{f}(\mathsf{G}, \mathsf{X})$ which includes only $\mathsf{G}$ and $\mathsf{X}$ as temporal operators. We consider both the fragments to have all Boolean operators.

## 3.2 High-level Overview of the Algorithm

Before the overview of our learning algorithm, let us consider the fundamental steps behind any exact learning algorithm for $\text{LTL}_\text{f}$. Typically, one searches through all $\text{LTL}_\text{f}$ formulas,

fixing an order on their size, and simultaneously checks whether any of the formulas is consistent with our sample. Checking whether an $\text{LTL}_\text{f}$ formula is consistent can be done using standard methods (such as dynamic programming [158], bit vector operations [20], etc.). However, the major drawback of this idea is that we have to search through all $\text{LTL}_\text{f}$ formulas, which is hard as the number of $\text{LTL}_\text{f}$ formulas grows very quickly[2].

To tackle this issue, instead of the entire $\text{LTL}_\text{f}$ fragment, our algorithm (as outlined in Algorithm 1) performs an iterative search through a subclass of $\text{LTL}_\text{f}$, which we call directed-LTL (Line 4). We also devise an efficient enumeration technique based on dynamic programming that can (i) generate these directed-LTL formulas in a particular "size order" (note, not the usual size of $\text{LTL}_\text{f}$ formulas), and (ii) also, evaluate these formulas over the traces in the sample.

The size order has a trivial (and large) upper-bound since there always exists a large $\text{LTL}_\text{f}$ formula that is consistent with any given sample. This large formula intuitively enumerates the differences between the positive and negative traces [169]. Formally, it is $\bigvee_{u \in P} \bigwedge_{v \in N} \varphi_{u,v}$, where $\varphi_{u,v}$ is a formula that uses $\mathsf{X}$ operator and propositions to specify the difference between the positive trace $u$ and the negative trace $v$.

To include more than just directed-LTL formulas, we generate and search through the Boolean combinations of the most promising directed-LTL formulas (Line 11). Note that the subclass of $\text{LTL}_\text{f}$ that our algorithm searches through ultimately does not include formulas with $\mathsf{U}$ operator.

**Algorithm 1** Overview of our algorithm

1: $B \leftarrow \emptyset$
2: $\psi \leftarrow \emptyset$: {stores the best formula found}
3: **for all** $s$ in increasing "size order" **do**
4: $\quad D \leftarrow$ all directed-LTL formulas of parameter $s$
5: $\quad$ **for all** $\varphi \in D$ **do**
6: $\quad\quad$ **if** $\varphi$ is consistent and smaller than $\psi$ **then**
7: $\quad\quad\quad \psi \leftarrow \varphi$
8: $\quad\quad$ **end if**
9: $\quad$ **end for**
10: $\quad B \leftarrow B \cup D$
11: $\quad B \leftarrow$ Boolean combinations of the promising formulas in $B$
12: $\quad$ **for all** $\varphi \in B$ **do**
13: $\quad\quad$ **if** $\varphi$ is consistent and smaller than $\psi$ **then**
14: $\quad\quad\quad \psi \leftarrow \varphi$
15: $\quad\quad$ **end if**
16: $\quad$ **end for**
17: **end for**
18: **return** $\psi$

During the search for formulas, our algorithm searches for smaller consistent formulas (if any, found at an earlier step) at each iteration. In fact, as a heuristic, once a consistent formula is found, we only search through formulas that are smaller than the found consistent formula.

[2]The number of $\text{LTL}_\text{f}$ formulas of size $k$ is asymptotically equivalent to $\frac{\sqrt{14} \cdot 7^k}{2\sqrt{\pi k^3}}$ [87]

Such a heuristic, along with aiding the search for minimal formulas, also reduces the search space significantly.

**Anytime feature.** The anytime feature of our algorithm is also a consequence of storing the smallest formula seen so far (Line 7 and 14). Once we find a consistent formula, we can output it and continue the search for smaller, consistent formulas.

## 3.3 Directed LTL

The first insight of our algorithm is the definition of directed-LTL; we often use the shorthand dLTL for directed-LTL.

A *partial symbol* is a conjunction of positive or negative atomic propositions. We write $s = p_0 \wedge p_2 \wedge \neg p_1$ for the partial symbol specifying that $p_0$ and $p_2$ hold and $p_1$ does not. The definition of a symbol satisfying a partial symbol is natural: for instance, the symbol $\{p_0, p_2, p_4\}$ satisfies $s$. The *width* of a partial symbol is the number of atomic propositions it uses.

We now define the syntax of directed-LTL as follows:

$$\varphi = \mathsf{X}^n s \quad | \quad \mathsf{F}\,\mathsf{X}^n s \quad | \quad \mathsf{X}^n(s \wedge \varphi) \quad | \quad \mathsf{F}\,\mathsf{X}^n(s \wedge \varphi),$$

where $s$ is a partial symbol, $n \in \{0, 1, \cdots\}$ and $\mathsf{X}^n\,\varphi$ is a shorthand for $\underbrace{\mathsf{X}\ldots\mathsf{X}}_{n \text{ times}}\varphi$.

As an example, the dLTL formula

$$\mathsf{F}((p \wedge q) \wedge \mathsf{F}\,\mathsf{X}^2\,\neg p)$$

reads: there exists a timepoint satisfying $p \wedge q$, and at least two timepoints later, there exists a timepoint satisfying $\neg p$. The intuition behind the term "directed" is that a dLTL formula fixes the order in which the partial symbols occur. A non-dLTL formula is $\mathsf{F}\,p \wedge \mathsf{F}\,q$: there is no order between $p$ and $q$. Note that dLTL only uses the $\mathsf{X}$ and $\mathsf{F}$ operators as well as conjunctions and atomic propositions.

**Generating directed formulas.** Let us consider the following problem: given the sample $\mathcal{S}$, we want to generate all dLTL formulas together with a list of traces in $\mathcal{S}$ that they satisfy. Our first technical contribution and key to the scalability of our approach is an efficient solution to this problem based on dynamic programming.

Let us define a natural order in which we want to generate dLTL formulas. They have two parameters: *length*, which is the number of partial symbols in the dLTL formula, and *width*, which is the maximum of the widths of the partial symbols in the dLTL formula. We combine these two parameters as the pair (*length*, *width*) and consider the following order on them:

$$(1,1), (2,1), (1,2), (3,1), (2,2), (1,3), \ldots$$

(We note that in practice, slightly more complicated orders on pairs are useful since we want to increase the length more often than the width.) Our enumeration algorithm works by generating dLTL formulas of a given pair of parameters from the sample in a recursive fashion. Assuming that we already generated all dLTL formulas for the pair of parameters $(\ell, w)$, we define two procedures, one for generating the dLTL formulas for the parameters $(\ell+1, w)$, and the other one for $(\ell, w+1)$.

When we generate the dLTL formulas, we also keep track of which traces in the sample they satisfy by exploiting a dynamic programming table called LastPos. We define it is as follows, where $\varphi$ is a dLTL formula and $u$ a trace in $\mathcal{S}$:

$$\textsc{LastPos}(\varphi, u) = \{t \in \{0, \ldots, |u|-1\} \mid u[0, t+1) \models \varphi\}.$$

This table stores the timepoints $t$ in trace $u$ such that dLTL formula $\varphi$ satisfies prefixes $u[0, t+1)$ The main benefit of LastPos is that it meshes well with dLTL formulas: it is algorithmically easy to compute them recursively on the structure of dLTL formulas. Moreover, one can exploit LastPos to check whether a dLTL formula $\varphi$ is consistent: $\textsc{LastPos}(\varphi, u)$ must be non-empty for positive traces and empty for negative traces.

A useful idea is to change the representation of the set of traces in $\mathcal{S}$, by precomputing the lookup table Index defined as follows, where $u$ is a trace in $\mathcal{S}$, $s$ a partial symbol, and $t$ in $\{0, \ldots, |u|\}$:

$$\textsc{Index}(u, s, t) = \left\{t' \in \{t, \ldots, |u|\} \mid u[t'] \models s\right\}.$$

This table stores the timepoints $t' \geq t$ in the trace $u$ where the partial formula $s$ holds. It can be precomputed in linear time from $\mathcal{S}$, and makes the dynamic programming algorithm easier to formulate.

Having defined the important ingredients, we now present the pseudocode (Algorithm 2) for both increasing the length and width of a formula. For the length increase algorithm, we define two extension operators $\wedge_{=k}$ and $\wedge_{\geq k}$ that "extend" the length of a dLTL formula $\varphi$ by including a partial symbol $s$ in the formula. Precisely, the operator $s \wedge_{=k} \varphi$ replaces the rightmost partial symbol $s'$ in $\varphi$ with $(s' \wedge \mathsf{X}^k s)$, while $s \wedge_{\geq k} \varphi$ replaces $s'$ with $(s' \wedge \mathsf{F}\,\mathsf{X}^k s)$. For instance, $c \wedge_{=2} \mathsf{X}(a \wedge \mathsf{X} b) = \mathsf{X}(a \wedge \mathsf{X}(b \wedge \mathsf{X}^2 c))$ and $c \wedge_{\geq 2} \mathsf{X}(a \wedge \mathsf{X} b) = \mathsf{X}(a \wedge \mathsf{X}(b \wedge \mathsf{F}\,\mathsf{X}^2 c))$.

For the width increase algorithm, we say that two dLTL formulas are *compatible* if they are equal except for partial symbols. For two compatible formulas, we define a *pointwise-and* ($\barwedge$) operator that takes the conjunction of the corresponding partial symbols at the same timepoints. For instance, $\mathsf{X}(a \wedge \mathsf{X} b) \barwedge \mathsf{X}(b \wedge \mathsf{X} c) = \mathsf{X}((a \wedge b) \wedge \mathsf{X}(b \wedge c))$.

The actual implementation of the algorithm refines certain parts of the sub-procedures. We provide some such examples here.

- In Line 3, instead of considering all partial symbols, we restrict to those appearing in at least one positive trace.
- In Line 13, some computations for $\varphi_{\geq j}$ can be made redundant; a finer data structure factorizes the computations.

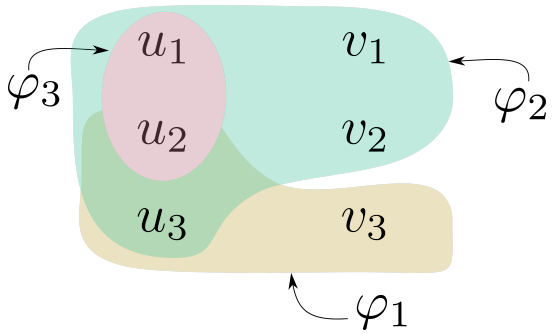



FIGURE 3.1: The Boolean subset cover problem: the formulas $\varphi_1, \varphi_2$, and $\varphi_3$ satisfy the words encircled in the corresponding area; in this case $(\varphi_1 \wedge \varphi_2) \vee \varphi_3$ is a consistent formula.

- In Line 25, using a refined data structure, we only enumerate compatible dLTL formulas.

**Lemma 1.** *Given a sample $\mathcal{S}$, Algorithm 2 generates all dLTL formulas from $\mathcal{S}$ and correctly computes the tables* LASTPOS.

This lemma can be proved using an induction on the natural order in which the dLTL formulas are searched.

**The dual point of view.** We use the same algorithm to produce formulas in a dual fragment to dLTL, which uses the $\mathsf{X}$ and $\mathsf{G}$ operators, the *last* predicate, as well as disjunctions and atomic propositions. The only difference is that we swap the positive and negative traces in the sample; that is, we consider the sample to be $\mathcal{S}' = (N, P)$. We obtain a dLTL formula from $\mathcal{S}'$ and apply its negation as shown below:

$$\neg \mathsf{X}\,\varphi = \mathit{last} \vee \mathsf{X} \neg \varphi \quad ; \quad \neg \mathsf{F}\,\varphi = \mathsf{G} \neg \varphi \quad ; \quad \neg(\varphi_1 \wedge \varphi_2) = \neg\varphi_1 \vee \neg\varphi_2.$$

## 3.4 Boolean Combination of Formulas

As explained in the previous section, we can efficiently generate dLTL formulas and dual dLTL formulas. We now explain how to form a Boolean combination of these formulas in order to construct consistent formulas, as illustrated in the beginning of the chapter.

Let us consider the following subproblem: given a set of formulas, does there exist a Boolean combination of some of the formulas that is a consistent formula? We call this the *Boolean subset cover* problem, which is illustrated in Figure 3.1. In this example, we have three formulas $\varphi_1, \varphi_2$, and $\varphi_3$, each satisfying subsets of $u_1, u_2, u_3, v_1, v_2, v_3$ as represented in the drawing. Inspecting the three subsets reveals that $(\varphi_1 \wedge \varphi_2) \vee \varphi_3$ is a consistent formula.

The Boolean subset cover problem is a generalization of the well-known and extensively studied subset cover problem, where we are given $S_1, \ldots, S_m$ subsets of $\{1, \ldots, n\}$, and the goal is to find a subset $I$ of $\{1, \ldots, m\}$ such that $\bigcup_{i \in I} S_i$ covers all of $\{1, \ldots, n\}$ – such a set $I$ is called a cover. Indeed, it corresponds to the case where all formulas satisfy none of the negative traces: in that case, conjunctions are not useful, and we can ignore the

**Algorithm 2** Generation of dLTL formulas for the traces in $\mathcal{S}$

**procedure** SEARCH DLTL FORMULAS – LENGTH INCREASE($\ell, w$)
  **for all** dLTL formulas $\varphi$ of length $\ell$ and width $w$ **do**
    **for all** partial symbols $s$ of width at most $w$ **do**
      **for all** $u \in S$ **do**
        $I = \text{LASTPOS}(\varphi, u)$
        **for all** $t_1 \in I$ **do**
          $J = \text{INDEX}(u, s, t_1)$
          **for all** $t_2 \in J$ **do**
            $\varphi_{=t_2} \leftarrow s \wedge_{=(t_2-t_1)} \varphi$
            add $t_2$ to $\text{LASTPOS}(\varphi_{=t_2}, u)$
          **end for**
          **for all** $t_1 \leq t_3 \leq \max(J)$ **do**
            $\varphi_{\geq t_3} \leftarrow s \wedge_{\geq(t_2-t_1+1)} \varphi$
            add $J \cap \{t_3, \ldots, |u|\}$ to $\text{LASTPOS}(\varphi_{\geq t_3}, u)$
          **end for**
        **end for**
      **end for**
    **end for**
  **end for**
**end procedure**

**procedure** SEARCH DLTL FORMULAS – WIDTH INCREASE($\ell, w$)
  **for all** dLTL formulas $\varphi$ of length $\ell$ and width $w$ **do**
    **for all** dLTL formulas $\varphi'$ of length $\ell$ and width 1 **do**
      **if** $\varphi$ and $\varphi'$ are compatible **then**
        $\varphi'' \leftarrow \varphi \wedge \varphi'$
        **for all** $u \in S$ **do**
          $\text{LASTPOS}(\varphi'', u) \leftarrow \text{LASTPOS}(\varphi, u) \cap \text{LASTPOS}(\varphi', u)$
        **end for**
      **end if**
    **end for**
  **end for**
**end procedure**

negative traces. The subset cover problem is known to be NP-complete. However, there exists a polynomial-time $\log(n)$-approximation algorithm called the greedy algorithm: it is guaranteed to construct a cover that is at most $\log(n)$ times larger than the minimum cover. This approximation ratio is optimal in the following sense [70]: there is no polynomial time $(1 - o(1))\log(n)$-approximation algorithm for subset cover unless P = NP. Informally, the greedy algorithm for the subset cover problem does the following: it iteratively constructs a cover $I$ by sequentially adding the most "promising" subset to $I$, which is the subset $S_i$ maximizing how many more elements of $\{1, \ldots, n\}$ are covered by adding $i$ to $I$.

**Greedy Approximation.** We introduce an extension of the greedy algorithm to the Boolean subset cover problem. The first ingredient is a scoring function. This function helps us in gathering the formulas that are *promising*. There are many possibilities for such scoring

functions. We consider the following function:

$$s_{\text{con}}(\varphi) = |\{u \in P \mid u \models \varphi\}| + |\{u \in N \mid u \not\models \varphi\}|,$$

which assigns a score based on how close the formula is to being consistent. We also consider another function

$$s_{\text{size}}(\varphi) = \frac{|\{u \in P \mid u \models \varphi\}| + |\{u \in N \mid u \not\models \varphi\}|}{\sqrt{|\varphi|}+1},$$

which assigns a score taking into account also how large the formula is. We found empirical success with the later score, in the sense that we were able to obtain a concise Boolean combination of formulas. The operation $\sqrt{\cdot}+1$ in the denominator helped us factor in the importance of size over being consistent.

The pseudocode is given in Algorithm 3. The algorithm maintains a set $B$ of formulas, which, initially, is the set of formulas given as input. It adds new formulas to $B$ until a consistent formula is found. We rely on a parameter $K$ that tracks how many *promising* formulas we like to consider in the sub-procedure (in the implementation, $K$ was set to 5). At each point in time, the algorithm chooses the $K$ formulas $\varphi_1, \ldots, \varphi_K$ with the highest score in $B$ and constructs all disjunctions and conjunctions of $\varphi_i$ with formulas in $B$. For each $i$, we keep the disjunction or conjunction with a maximal score and add this formula to $B$ if it has a higher score than $\varphi_i$. We repeat this procedure until we find a consistent formula or no formula is added to $B$. An important optimization is to keep an upper bound on the size of a consistent formula, which we use to cut off computations that cannot lead to smaller formulas in the greedy algorithm for the Boolean subset cover problem.

**Algorithm 3** Greedy algorithm for the Boolean subset cover problem

**Input:** $u_1, \ldots, u_n, v_1, \ldots, v_n$, and a set $F$ of formulas

```
Set K < |F|
procedure GREEDY(F)
    choose the K formulas φ_1, ..., φ_K in F with the highest score
    for all ψ ∈ F do
        for all 1 ≤ i ≤ K do
            construct ψ ∧ φ_i and ψ ∨ φ_i
            compute their scores
            if one of the two formulas is consistent then
                return the consistent formula
            end if
        end for
        let θ be the formula with the highest score computed using ψ
        if θ has higher score than ψ then
            add θ to F
        end if
    end for
end procedure
```

**Decision Tree Learning.** Another natural approach to the Boolean subset cover problem is to use decision trees. This approach is rather straightforward: we consider the set $F$ of dLTL formulas obtained from the first sub-procedure as the primitives (or features) for decision tree learning. We then follow any standard decision tree learning algorithm [182] (such as ID3, C4.5, CART) to find a decision tree that perfectly classifies the sample. The $\text{LTL}_\text{f}$ formula corresponding to the decision tree can be obtained in a standard manner [169].

We experimented with both approaches and found that the greedy algorithm is both faster and yields smaller formulas. We do not report on these experiments because the formulas obtained using decision tree learning are always larger than the greedy approach and, therefore, less useful for practical purposes. Let us, however, remark that using decision trees we get the theoretical guarantee that the algorithm always terminates with a consistent formula, as explained in Theorem 2.

## 3.5 Theoretical Guarantees

We present a result that shows the relevance of the subclass directed-LTL and their Boolean combinations.

**Theorem 1.** *Every formula of* $\text{LTL}_f(\mathsf{F},\mathsf{X})$ *is equivalent to a Boolean combination of dLTL formulas. Equivalently, every formula of* $\text{LTL}_f(\mathsf{G},\mathsf{X})$ *is equivalent to a Boolean combination of dual dLTL formulas.*

In other words, this theorem says that the expressive power of $\text{LTL}_\text{f}(\mathsf{F},\mathsf{X})$ and $\text{LTL}_\text{f}(\mathsf{G},\mathsf{X})$ is the same as the Boolean combination of directed-LTL and dual directed-LTL, respectively.

To get an insight behind the proof of Theorem 1, let us consider the formula $\mathsf{F}\,p \wedge \mathsf{F}\,q$, which is not directed. This formula is equivalent to $\mathsf{F}(p \wedge \mathsf{F}\,q) \vee \mathsf{F}(q \wedge \mathsf{F}\,p)$, which is obtained by the Boolean combination of dLTL formulas $\mathsf{F}(p \wedge \mathsf{F}\,q)$ and $\mathsf{F}(q \wedge \mathsf{F}\,p)$. Here, we simply rewrite the first formula using a disjunction over the possible orderings of $p$ and $q$. The formal proof generalizes this rewriting idea, which we present now.

For the proof, we denote the Boolean combination of directed-LTLformulas as $\text{dLTL}(\wedge,\vee)$. We begin with a lemma necessary to prove the above theorem.

**Lemma 2.** *Let* $\Delta_1$, $\Delta_2$ *be two dLTL formulas. Then,* $\Delta_1 \wedge \Delta_2$ *can be written as a disjunction of formulas in dLTL.*

*Proof of Lemma 2.* To prove the lemma, we use induction over the structure of $\Delta_1 \wedge \Delta_2$ to show that it can be written as a disjunction of dLTL formulas. As induction hypothesis, we consider all formulas $\Delta_1' \wedge \Delta_2'$, where at least one of $\Delta_1'$ and $\Delta_2'$ is structurally smaller than $\Delta_1$ and $\Delta_2$ respectively, can be written as a disjunction of dLTL formulas.

The base case of the induction is when either $\Delta_1$ or $\Delta_2$ is a partial symbol. In this case, $\Delta_1 \wedge \Delta_2$ is itself a dLTL formula by definition of dLTL formulas.

The induction step proceeds via case analysis on the possible root operators of the formulas $\Delta_1$ and $\Delta_2$

- Case: either $\Delta_1$ or $\Delta_2$ is of the form $s \wedge \Delta$ for some partial symbol $s$. Without loss of generality, let us say $\Delta_1 = s \wedge \Delta$. In this case, $\Delta_1 \wedge \Delta_2 = (s \wedge \Delta) \wedge \Delta_2 = s \wedge (\Delta \wedge \Delta_2)$. By hypothesis, $\Delta \wedge \Delta_2 = \bigvee_i \Gamma_i$ for some $\Gamma_i$ in dLTL. Thus, $\Delta_1 \wedge \Delta_2 = s \wedge \bigvee_i \Gamma_i = \bigvee_i (s \wedge \Gamma_i)$, which is a disjunction of dLTL formulas.

- Case: $\Delta_1$ is of the form $\mathsf{X}\,\delta_1$ and $\Delta_2$ is of the form $\mathsf{X}\,\delta_2$. In this case, $\Delta_1 \wedge \Delta_2 = \mathsf{X}(\delta_1 \wedge \delta_2)$. By hypothesis, $\delta_1 \wedge \delta_2 = \bigvee_i \gamma_i$ for some $\gamma_i$'s in dLTL. Thus, $\Delta_1 \wedge \Delta_2 = \mathsf{X}(\bigvee_i \gamma_i) = \bigvee_i \mathsf{X}\,\gamma_i$, which is a disjunction of dLTL formulas.

- Case: $\Delta_1$ is of the form $\mathsf{X}\,\delta_1$ and $\Delta_2$ is of the form $\mathsf{F}\,\delta_2$. In this case, $\Delta_1 \wedge \Delta_2 = \mathsf{X}\,\delta_1 \wedge \mathsf{F}\,\delta_2 = (\mathsf{X}\,\delta 1 \wedge \delta_2) \vee (\mathsf{X}\,\delta_1 \wedge \mathsf{F}\,\mathsf{X}\,\delta_2) = (\mathsf{X}\,\delta 1 \wedge \delta_2) \vee \mathsf{X}(\delta_1 \wedge \mathsf{F}\,\delta_2)$. By hypothesis, both formulas $(\mathsf{X}\,\delta 1 \wedge \delta_2)$ and $(\delta_1 \wedge \mathsf{F}\,\delta_2)$ can be written as a disjunction of dLTL formulas. Thus, $\Delta_1 \wedge \Delta_2$ can also be written as a disjunction of dLTL formulas

- Case: $\Delta_1$ is of the form $\mathsf{F}\,\delta_1$ and $\Delta_2$ is of the form $\mathsf{F}\,\delta_2$. In this case, $\Delta_1 \wedge \Delta_2 = \mathsf{F}\,\delta_1 \wedge \mathsf{F}\,\delta_2 = \mathsf{F}(\delta_1 \wedge \mathsf{F}\,\delta_2) \vee \mathsf{F}(\delta_2 \wedge \mathsf{F}\,\delta_1)$. By hypothesis, both formulas $\delta 1 \wedge \mathsf{F}\,\delta_2$ and $\delta_2 \wedge \mathsf{F}\,\delta_1$ can be written as a disjunction of dLTL formulas. Thus, $\Delta_1 \wedge \Delta_2$ can also be written as a disjunction of dLTL formulas.

□

*Proof of Theorem 1.* We prove the first statement of the theorem since the second statement follows analogously. This proof proceeds via induction on the structure of formulas $\varphi$ in $\mathrm{LTL}_\mathrm{f}(\mathsf{F},\mathsf{X})$. As the induction hypothesis, we consider that all formulas $\varphi'$ which are structurally smaller than $\varphi$ can be expressed in $\mathrm{dLTL}(\wedge,\vee)$.

As the base case of the induction, we observe that formulas $p$ for all $p \in \mathcal{P}$, are dLTL formulas and thus, in $\mathrm{dLTL}(\wedge,\vee)$.

For the induction step, we perform a case analysis based on the root operator of $\varphi$.

- Case $\varphi = \varphi_1 \vee \varphi_2$ or $\varphi = \varphi_1 \wedge \varphi_2$: By hypothesis, $\varphi_1$ is in $\mathrm{dLTL}(\wedge,\vee)$ and $\varphi_2$ is in $\mathrm{dLTL}(\wedge,\vee)$. Now, $\varphi$ is in $\mathrm{dLTL}(\wedge,\vee)$ since $\mathrm{dLTL}(\wedge,\vee)$ is closed under positive boolean combinations.

- Case $\varphi = \mathsf{X}\,\varphi_1$: By hypothesis, $\varphi_1 \in \mathrm{dLTL}(\wedge,\vee)$ and thus $\varphi_1 = \bigvee_j(\bigwedge_i \Delta_i)$. Now, $\varphi = \mathsf{X}(\bigvee_j(\bigwedge_i \Delta_i)) = \bigvee_j(\mathsf{X}(\bigwedge_i \Delta_i)) = \bigvee_j(\bigwedge_i \mathsf{X}\,\Delta_i) = \bigvee_i \bigwedge_i \Delta'_i$ ($\mathsf{X}\,\Delta_i$ is a dLTL formula). Thus, $\varphi$ is in $\mathrm{dLTL}(\wedge,\vee)$.

- Case $\varphi = \mathsf{F}\,\varphi_1$: By hypothesis, $\varphi_1 \in \mathrm{dLTL}(\wedge,\vee)$ and thus $\varphi_1 = \bigvee_j(\bigwedge_i \Delta_i)$. Now $\varphi = \mathsf{F}\,\varphi_1 = \bigvee_j(\mathsf{F}(\bigwedge_i \Delta_i))$. Using lemma 2, we can re-write $\bigwedge_i \Delta_i$ as $\bigvee_i \Gamma_i$ for some $\Gamma_i$'s in dLTL. As a result, $\varphi = \bigvee_j \bigvee_i \mathsf{F}(\Gamma_i)$. Thus, $\varphi$ is in $\mathrm{dLTL}(\wedge,\vee)$.

□

We can now state the theoretical guarantees[3] for our overall algorithm.

**Theorem 2.** *Given a sample $\mathcal{S}$, Algorithm 1 has the following guarantees:*

[3]Termination and soundness hold for both greedy approximation and decision tree learning as Sub-procedure 3. Completeness holds only for decision-tree learning as Sub-procedure 3.

**Termination:** *it always terminates;*

**Soundness:** *if it returns a formula, then the formula is consistent with $\mathcal{S}$; and*

**Completeness:** *when decision tree learning is used as Sub-procedure 3, it always returns a consistent formula.*

*Proof.* For termination, we highlight that there exists a large $\text{LTL}_\text{f}$ formula, say of size $\mathcal{B}$, consistent with the given sample $\mathcal{S}$. There are only finitely many dLTL formulas and their Boolean combinations that are possible of size at most $B$. As a result, our algorithm searches through the formulas smaller than $\mathcal{B}$ and, thus, terminates.

For soundness, we rely on Lemma 1 to ensure that during the enumeration of dLTL formulas in sub-procedure 2, the tables LastPos are calculated correctly. We check whether a dLTL formula is consistent based on LastPos and, thus, must be correct. In the subprocedure 3.4, the greedy algorithm also exploits LastPos to check whether the Boolean combination of formulas is consistent. Also, standard decision tree learning algorithms always terminate when perfect classification is achieved which, in this case, means that the corresponding $\text{LTL}_\text{f}$ formula is consistent.

For completeness, we show that if decision tree learning is used in sub-procedure 2, the algorithm can find a consistent formula, albeit large. To show this, we first observe that the formula $\varphi_{u,v}$ that distinguishes between a positive trace $u$ and negative trace $v$ using X operators and propositions (see third paragraph, Section 3.2) is a dLTL formula. Thus, for any positive trace $u$ and negative trace $v$, there exists a dLTL formula that can distinguish them. Based on the property of (TDIDT) decision tree learning algorithms, when any pair of positive and negative examples can be classified by some feature, as is the case here, perfect classification can be achieved. Thus, the corresponding $\text{LTL}_\text{f}$ formula must be consistent with the sample. □

## 3.6 Experimental Evaluation

In this section, we answer the following research questions to assess the performance of our overall learning algorithm.

**RQ1:** How effective are we in learning concise $\text{LTL}_\text{f}$ formulas from samples?

**RQ2:** How much scalability do we achieve through our algorithm?

**RQ3:** What do we gain from the anytime feature of our algorithm?

**Experimental Setup.** To answer the questions above, we have implemented a prototype of our algorithm in Python 3 in a tool named `SCARLET`[4] (SCalable Anytime algoRithm for LEarning lTl). We run `SCARLET` on several benchmarks generated synthetically from $\text{LTL}_\text{f}$ formulas used in practice. To answer each research question precisely, we choose different sets of $\text{LTL}_\text{f}$ formulas. We discuss them in detail in the corresponding sections.

[4] https://github.com/rajarshi008/Scarlet

However, note that we did not include any formulas with U-operator, since SCARLET is not designed to find such formulas. It is possible that if a sample requires an $\text{LTL}_\text{f}$ formula with U-operator, SCARLET can not find the formula. Nevertheless, SCARLET covers a significant subclass of $\text{LTL}_\text{f}$ (as shown in Theorem 1).

To assess the performance of SCARLET, we compare it against two state-of-the-art tools for learning temporal logic formulas from examples:

1. FLIE[5], developed by Neider et al. [169], infers minimal $\text{LTL}_\text{f}$ formulas using a learning algorithm that is based on constraint solving (SAT solving).

2. SYSLITE[6], developed by Arif et al. [8], originally infers minimal past-time $\text{LTL}_\text{f}$ formulas using an enumerative algorithm implemented in a tool called CVC4SY [192]. For our comparisons, we use a version of SYSLITE that we modified (which we refer to as $\text{SYSLITE}_\text{L}$) to infer $\text{LTL}_\text{f}$ formulas rather than past-time $\text{LTL}_\text{f}$ formulas. Our modifications include changes to the syntactic constraints generated by $\text{SYSLITE}_\text{L}$ as well as changing the semantics from past-time $\text{LTL}_\text{f}$ to ordinary LTL.

To obtain a fair comparison against SCARLET, in both the tools, we disabled the U-operator. This is because if we allow U-operator, this will only make the tools slower since they will have to search through all formulas containing U.

All the experiments are conducted on a single core of a Debian machine with Intel Xeon E7-8857 CPU (at 3 GHz) using up to 6 GB of RAM. We set the timeout to be 900 s for all experiments. We include scripts to reproduce all experimental results in a publicly available artifact [191].

TABLE 3.1: Common $\text{LTL}_\text{f}$ formulas used in practice

| | |
|---|---|
| Absence: | $\mathsf{G}(\neg p)$, $\mathsf{G}(q \rightarrow \mathsf{G}(\neg p))$ |
| Existence: | $\mathsf{F}(p)$, $\mathsf{G}(\neg p) \vee \mathsf{F}(p \wedge \mathsf{F}(q))$ |
| Universality: | $\mathsf{G}(p)$, $\mathsf{G}(q \rightarrow \mathsf{G}(p))$ |
| Disjunction of patterns: | $\mathsf{G}(\neg p) \vee \mathsf{F}(p \wedge \mathsf{F}(q) \vee \mathsf{G}(\neg s) \vee \mathsf{F}(r \wedge \mathsf{F}(s))$, $\mathsf{F}(r) \vee \mathsf{F}(p) \vee \mathsf{F}(q)$ |

**Sample generation.** To provide a comparison among the learning tools, we follow the literature [169] and use synthetic benchmarks generated from real-world $\text{LTL}_\text{f}$ formulas. For benchmark generation, earlier works rely on a fairly naive generation method. In this method, starting from a formula $\varphi$, a sample is generated by randomly drawing traces and categorizing them into positive and negative examples depending on the satisfaction with respect to $\varphi$. This method, however, often results in samples that can be separated by formulas much smaller than $\varphi$. Moreover, it often requires a prohibitively large amount of time to generate samples

[5] https://github.com/ivan-gavran/samples2LTL
[6] https://github.com/CLC-UIowa/SySLite

(for instance, for $\mathsf{G}\, p$, where almost all traces satisfy a formula) and, hence, often does not terminate in a reasonable time.

To alleviate the issues in the existing method, we have designed a novel generation method for the quick generation of large samples. The outline of the generation algorithm is presented in Algorithm 4. The crux of the algorithm is to convert the $\text{LTL}_\text{f}$ formula $\varphi$ into its equivalent DFA $\mathcal{A}_\varphi$ and then extract random traces from the DFA to obtain a sample of desired length and size.

To convert $\varphi$ into its equivalent DFA $\mathcal{A}_\varphi$ (Line 2), we rely on a python tool $\text{LTL}_\text{f}$2DFA[7]. Essentially, this tool converts $\varphi$ into its equivalent formula in First-order Logic and then obtains a minimal DFA from the formula using a tool named MONA [115].

For extracting random traces from the DFA (Line 4 and 8), we use a procedure suggested by Bernardi et al. [27]. The procedure involves generating traces by choosing symbols that have a higher probability of leading to an accepting state. This requires assigning appropriate probabilities to the transitions of the DFA. In this step, we add our modifications to the procedure. The main idea is that we adjust the probabilities of the transitions appropriately to ensure that we obtain *distinct* traces in each iteration.

**Algorithm 4** Sample generation algorithm

**Input:** Formula $\varphi$, length $l$, number of positive traces $n_P$, number of negative traces $n_N$

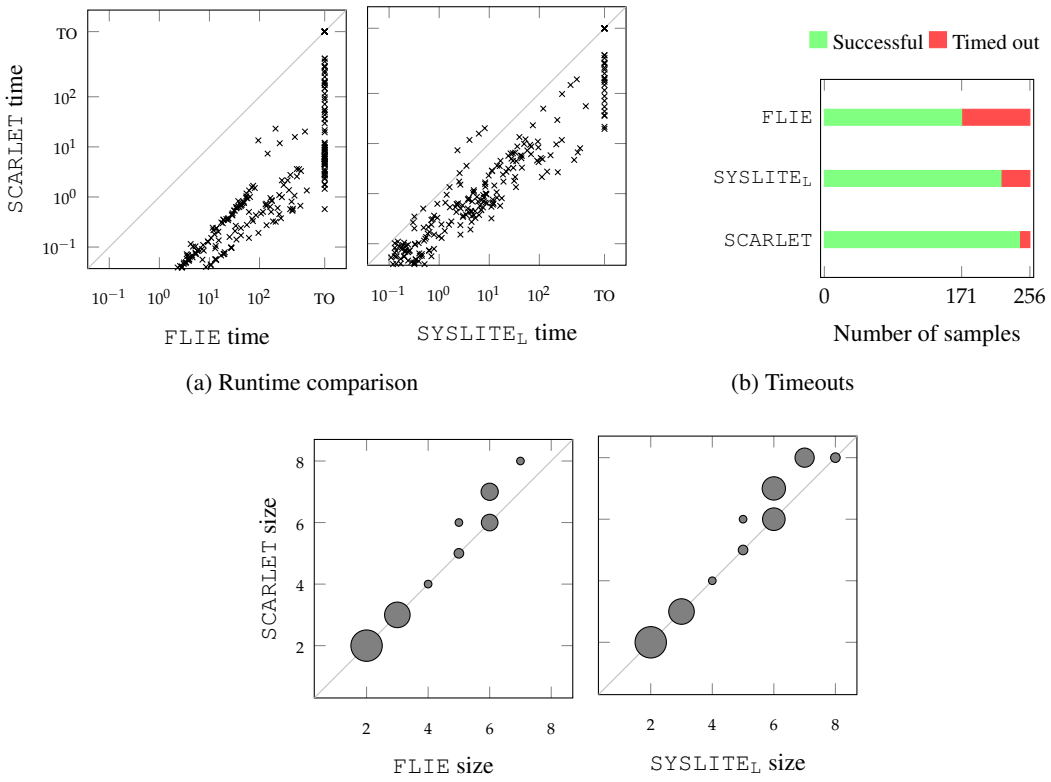
1: $P \leftarrow \{\}, N \leftarrow \{\}$
2: $\mathcal{A}_\varphi \leftarrow$ convert2DFA($\varphi$)
3: **for** $1 \cdots n_P$ **do**
4: $\quad u \leftarrow$ random accepted trace of length $l$ from $\mathcal{A}_\varphi$.
5: $\quad P \leftarrow P \cup \{u\}$
6: **end for**
7: **for** $1 \cdots n_N$ **do**
8: $\quad u \leftarrow$ random accepted trace of length $l$ from $\mathcal{A}^{\mathsf{c}}_\varphi$.
9: $\quad N \leftarrow N \cup \{u\}$
10: **end for**
11: **return** $S = (P, N)$

Unlike existing sample generation methods, our method does not create random traces and try to classify them as positive or negative. This results in a much faster generation of large and better *quality* samples.

### 3.6.1 RQ1: Performance Comparison

To address our first research question, we have compared all three tools on a synthetic benchmark suite generated from eight $\text{LTL}_\text{f}$ formulas. These formulas originate from a study by Dwyer et al. [75], who have collected a comprehensive set of $\text{LTL}_\text{f}$ formulas arising in real-world applications (see Table 3.1 for an excerpt). The selected $\text{LTL}_\text{f}$ formulas have, in fact, also been used by `FLIE` for generating its benchmarks. While `FLIE` also considered

[7] https://github.com/whitemech/ltlf2DFA




(a) Runtime comparison

(b) Timeouts

(c) Size comparison

FIGURE 3.2: Comparison of SCARLET, FLIE and SYSLITE$_L$ on synthetic benchmarks. In Figure 3.2a, all times are in seconds, and 'TO' indicates timeouts. The size of the bubbles in the figure indicates the number of samples for each data point.

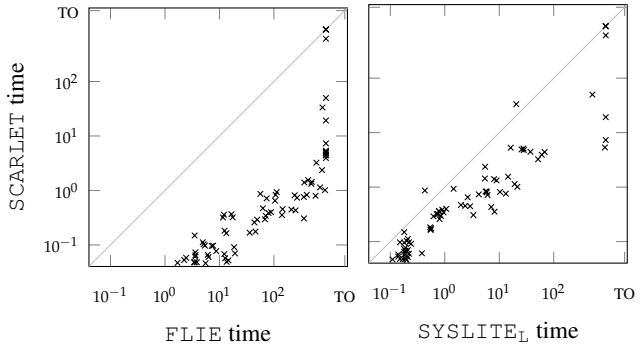
formulas with U-operator, we did not consider them for generating our benchmarks due to reasons mentioned in the experimental setup.

Our benchmark suite consists of a total of 256 samples (32 for each of the eight LTL$_f$ formulas) generated using our generation method. The number of traces in the samples ranges from 50 to 2 000, while the length of traces ranges from 8 to 15.

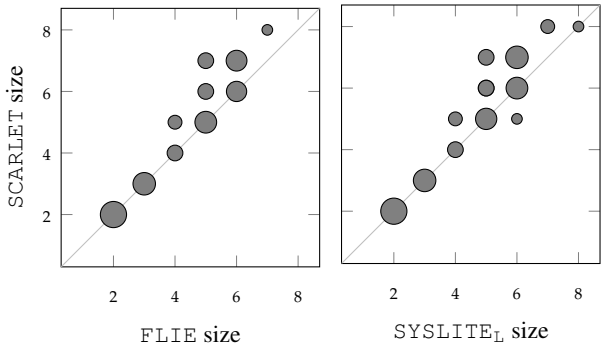
Figure 3.2a presents the runtime comparison of FLIE, SYSLITE$_L$ and SCARLET on all 256 samples. From the scatter plots, we observe that SCARLET ran faster than FLIE on all samples. Likewise, SCARLET was faster than SYSLITE$_L$ on all but eight (out of 256) samples. SCARLET timed out on only 13 samples, while FLIE and SYSLITE$_L$ timed out on 85 and 36, respectively (see Figure 3.2b).

The good performance of SCARLET can be attributed to its efficient formula search technique. In particular, SCARLET only considers formulas that have a high potential of being a consistent formula since it extracts directed-LTL formulas from the sample itself. FLIE and SYSLITE$_L$, on the other hand, search through arbitrary formulas (in order of increasing size), each time checking if the current one separates the sample.

Figure 3.2c presents the comparison of the size of the formulas inferred by each tool. On 170 out of the 256 samples, all tools terminated and returned an LTL$_f$ formula with size at most 7. In 150 out of these 170 samples, SCARLET, FLIE, and SYSLITE$_L$ inferred formulas of




(a) Runtime comparison




(b) Size comparison

FIGURE 3.3: Comparison of SCARLET, FLIE and SYSLITE$_L$ on existing benchmarks. In Figure 3.3a, all times are in seconds and 'TO' indicates timeouts. The size of bubbles indicate the number of samples for each datapoint.

equal size, while on the remaining 20 samples, SCARLET inferred formulas that were larger. The latter observation indicates that SCARLET misses certain small, consistent formulas, in particular, the ones that are not a Boolean combination of dLTL formulas.

However, it is important to highlight that the formulas learned by SCARLET are, in most cases, not significantly larger than those learned by FLIE and SYSLITE$_L$. This can be seen from the fact that the average size of formulas inferred by SCARLET (on benchmarks in which none of the tools timed out) is 3.21, while the average size of formulas inferred by FLIE and SYSLITE$_L$ is 3.07.

To ensure that SCARLET performs well, not only in our generated benchmarks, we compared the performance of the tools on the benchmark suite[8] [92]. The benchmark suite has been generated using a fairly naive generation method from the same set of LTL$_f$ formulas listed in Table 3.1. We introduced this benchmark suite while working on noisy data, which we discuss in the next chapter.

Figure 3.3a represents the runtime comparison of FLIE, SYSLITE$_L$ and SCARLET on 98 samples from the existing benchmark. From the scatter plots, we observe that SCARLET runs much faster than FLIE on all samples and than SYSLITE$_L$ on all but two samples. Also,

[8] https://github.com/cryhot/samples2LTL

SCARLET timed out only on three samples while $\texttt{SYSLITE}_{\texttt{L}}$ timed out on six samples and FLIE timed out on 15 samples.

Figure 3.3b presents the comparison of formula size inferred by each tool. On 84 out of 98 samples, where none of the tools timed out, we observe that on 65 samples, SCARLET inferred formula size equal to the one inferred by $\texttt{SYSLITE}_{\texttt{L}}$ and FLIE. Further, in the samples where SCARLET learns larger formulas than other tools, the size gap is not significant. This is evident from the fact that the average formula size learned by SCARLET is 4.13, which is slightly higher than that by FLIE and $\texttt{SYSLITE}_{\texttt{L}}$, 3.84.

Overall, SCARLET displayed significant speed-up over both FLIE and $\texttt{SYSLITE}_{\texttt{L}}$ while learning a formula similar in size, answering question RQ1 in the positive.

### 3.6.2 RQ2: Scalability

To address the second research question, we investigate the scalability of SCARLET in two dimensions: the size of the sample and the size of the formula from which the samples are generated.

**Scalability with respect to the size of the samples.** For demonstrating the scalability with respect to the size of the samples, we consider two formulas $\varphi_{cov} = \mathsf{F}(a_1) \wedge \mathsf{F}(a_2) \wedge \mathsf{F}(a_3)$ and $\varphi_{seq} = \mathsf{F}(a_1 \wedge \mathsf{F}(a_2 \wedge \mathsf{F}\, a_3))$, both of which appear commonly in robotic motion planning [81]. While the formula $\varphi_{cov}$ describes the property that a robot eventually visits (or covers) three regions $a_1$, $a_2$, and $a_3$ in arbitrary order, the formula $\varphi_{seq}$ describes that the robot has to visit the regions in the specific order $a_1 a_2 a_3$.

We have generated two sets of benchmarks for both formulas, for which we varied the number of traces and their length, respectively. More precisely, the first benchmark set contains 90 samples of an increasing number of traces (5 samples for each number), ranging from 200 to 100 000, each consisting of traces of fixed length 10. On the other hand, the second benchmark set contains 90 samples of 200 traces, containing traces from length 10 to length 50.

Figure 3.4a shows the average runtime results of SCARLET, FLIE, and $\texttt{SYSLITE}_{\texttt{L}}$ on the first benchmark set. We observe that SCARLET substantially outperformed the other two tools on all samples. This is because both $\varphi_{cov}$ and $\varphi_{seq}$ are of size eight and inferring formulas of such size is computationally challenging for FLIE and $\texttt{SYSLITE}_{\texttt{L}}$. In particular, FLIE and $\texttt{SYSLITE}_{\texttt{L}}$ need to search through all formulas of size upto eight to infer the formulas, while, SCARLET, due to its efficient search order (using length and width of a formula), infers them faster.

From Figure 3.4a, we further observe a significant difference between the run times of SCARLET on samples generated from formula $\varphi_{cov}$ and from formula $\varphi_{seq}$. This is evident from the fact that SCARLET failed to infer formulas for samples of $\varphi_{seq}$ starting at a size of 6 000, while it could infer formulas for samples of $\varphi_{cov}$ up to a size of 50 000. Such a result is again due to the search order used by SCARLET: while $\varphi_{cov}$ is a Boolean combination of dLTL formulas of length 1 and width 1, $\varphi_{seq}$ is a dLTL formula of length 3 and width 1.

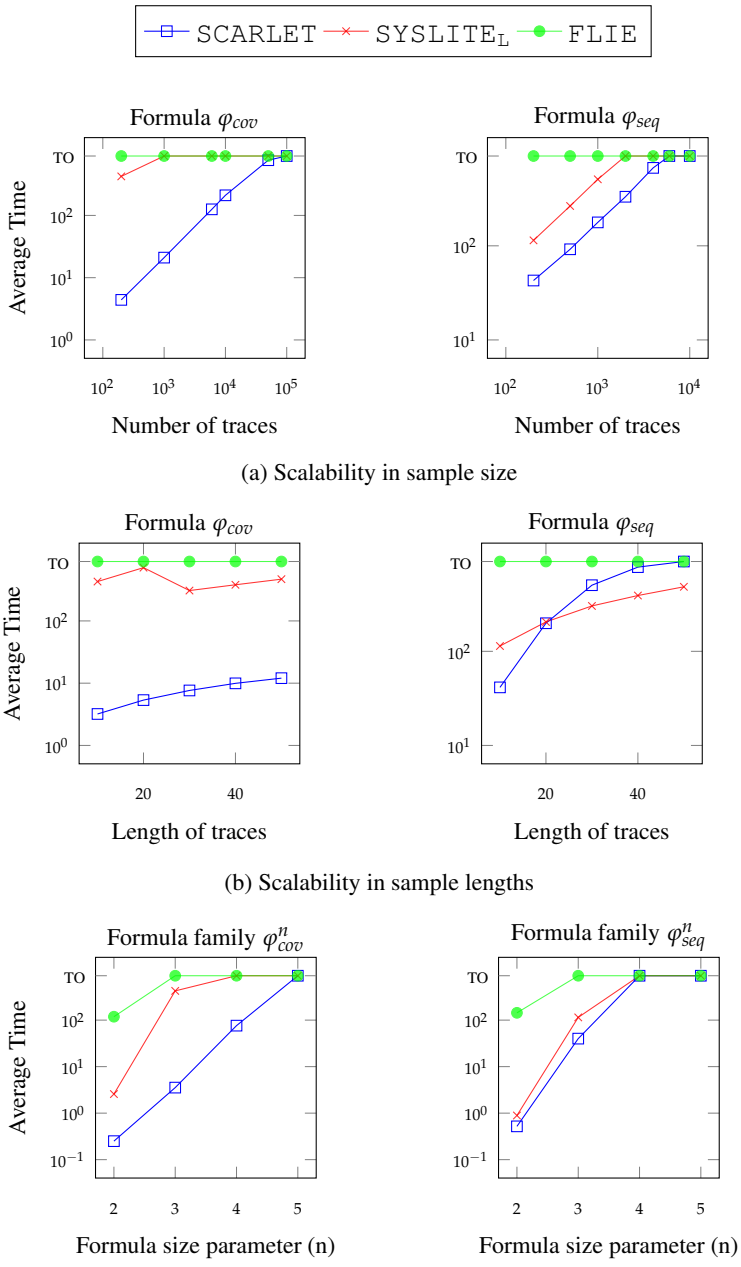
Figure 3.4b depicts the results we obtained by running all the second benchmark set with varying trace lengths. Some trends we observe here are similar to the ones we observe in the first benchmark set. For instance, SCARLET performs better on the samples from $\varphi_{cov}$ than it does on samples from $\varphi_{seq}$. The reason for this remains similar: it is easier to find a formula that is a Boolean combination of length 1, width 1 dLTL formulas, compared to dLTL formula of length 3 and width 1.

Contrary to the results on the first benchmark set, we observe that the increase of runtime with the length of the sample is quadratic. This explains why on samples from $\varphi_{seq}$ on large lengths such as 50, SCARLET faces time-out. However, for samples from $\varphi_{cov}$, SCARLET displays the ability to scale way beyond length 50.

**Scalability with respect to the size of the formula.** To demonstrate the scalability with respect to the size of the formula used to generate samples, we have extended $\varphi_{cov}$ and $\varphi_{seq}$ to families of formulas $(\varphi^n_{cov})_{n\in\mathbb{N}\setminus\{0\}}$ with $\varphi^n_{cov} = \mathsf{F}(a_1) \wedge \mathsf{F}(a_2) \wedge \ldots \wedge \mathsf{F}(a_n)$ and $(\varphi^n_{seq})_{n\in\mathbb{N}\setminus\{0\}}$ with $\varphi^n_{seq} = \mathsf{F}(a_1 \wedge \mathsf{F}(a_2 \wedge \mathsf{F}(\ldots \wedge \mathsf{F}\, a_n)))$, respectively. This family of formulas describes properties similar to that of $\varphi_{cov}$ and $\varphi_{seq}$, but the number of regions is parameterized by $n \in \mathbb{N} \setminus \{0\}$. We consider formulas from the two families by varying $n$ from 2 to 5 to generate a benchmark suite consisting of samples (5 samples for each formula) having 200 traces of length 10.

Figure 3.4c shows the average run time comparison of the tools for samples from increasing formula sizes. We observe a trend similar to Figure 3.4a: SCARLET performs better than the other two tools and infers formulas of the family $\varphi^n_{cov}$ faster than that of $\varphi^n_{seq}$. However, contrary to the near linear increase of the runtime with the number of traces, we notice an almost exponential increase of the runtime with the formula size.

Overall, our experiments show better scalability with respect to sample and formula size compared to the other tools, answering RQ2 in the positive.




(a) Scalability in sample size

(b) Scalability in sample lengths

(c) Scalability in formula size

FIGURE 3.4: Comparison of SCARLET, FLIE and SYSLITE$_L$ on synthetic benchmarks. In Figures 3.4a and 3.4b, all times are in seconds and 'TO' indicates timeouts.

### 3.6.3 RQ3: Anytime feature

To answer RQ3, we list two advantages of the anytime feature of our algorithm. We demonstrate these advantages by showing evidence from the runs of SCARLET on benchmarks used in RQ1 and RQ2.

First, in the instance of a timeout, our algorithm may find a "concise" consistent formula while the other tools will not. In our experiments, we observed that for all benchmarks used in RQ1 and RQ2, SCARLET obtained a formula even when it timed out. In fact, in the samples from $\varphi^5_{cov}$ used in RQ2, SCARLET (see Figure 3.4c) obtained the exact original formula, that too within one second (0.7 seconds in average), although timed out later. The timeout was because SCARLET continued to search for smaller formulas even after finding the original formula.

Second, our algorithm can actually output the final formula earlier than its termination. This is evident from the fact that, for the 243 samples in RQ1 where SCARLET does not time out, the average time required to find the final formula is 10.8 seconds, while the average termination time is 25.17 seconds. Thus, there is a chance that even if one stops the algorithm earlier than its termination, one can still obtain the final formula.

Our observations from the experiments clearly indicate the advantages of anytime feature to obtain a concise consistent formula and thus, answering RQ3 in the positive.

## 3.7 Conclusion

We have proposed a scalable approach for learning $\text{LTL}_\text{f}$ formulas from examples, fleshing it out in an approximation anytime algorithm. We have shown in experiments that our algorithm outperforms existing tools in two ways: it scales to larger formulas and input samples, and even when it timeouts, it often outputs a consistent formula.

Our algorithm targets a strict fragment of $\text{LTL}_\text{f}$, restricting its expressivity in two aspects: it does not include the U-operator, and we cannot nest the F and G operators. Moreover, the approach is designed to cater to $\text{LTL}_\text{f}$ and not LTL. Extending it to LTL would require novel techniques to extract dLTL formulas from infinite traces. We leave such extensions to our algorithm as future work.

An important open question concerns the theoretical guarantees offered by the greedy algorithm for the Boolean subset cover problem. It extends a well-known algorithm for the classic subset cover problem, and this restriction has been proved to yield an optimal $\log(n)$-approximation. Do we have similar guarantees in our more general setting?

# Chapter 4

# Learning in the Presence of Noise

In this chapter, we consider the problem of learning temporal logic formulas when the input data can be potentially noisy. Noise is omnipresent in real-world data; it can arise from, for instance, imperfections in sensors, disturbances in the environment, unintended user intervention, etc. Similar to the last chapter, we consider the input data to be a sample $\mathcal{S}$ of system executions partitioned into two finite sets of positive and negative examples. However, since the data maybe noisy, it is possible some of the examples are wrongly labeled, either as positive or as negative.

Since noise in data is common in cyber-physical systems (CPS) applications, along with $\text{LTL}_\text{f}$, in this chapter, we study the learning problem for Signal Temporal Logic (STL) [157]. STL is essentially an extension of LTL that reasons about signals that are real-valued finite or infinite time series, typically appearing in CPS applications. In STL, one augments temporal operators with intervals of time to express real-time properties. For instance, using the STL formula $\mathsf{G}_{[0,60]}(\texttt{speed} < 30 \wedge \texttt{angle} < 60)$, one can describe the property "for the first 60 seconds, the speed of the vehicle should be less than 30 km/h, and the steering angle should be less than 60°".

The task of learning temporal logic formulas consistent with data has been studied extensively for both $\text{LTL}\backslash\text{LTL}_\text{f}$ and STL [137, 37, 169, 48, 135]. Most of the existing works that are able to handle noise [137, 37, 135] typically search for formulas that arise from certain handcrafted *templates*. Such an approach has several drawbacks. First, handcrafting templates may not be a straightforward task: it requires adequate knowledge about the underlying system. Second, by restricting its structure, one potentially increases the size of the learned formula. This makes the formulas difficult to comprehend and also amplifies the computation efforts required to find a formula.

Nevertheless, there are approaches [169, 48] that avoid the use of templates. These approaches reduce the learning problem to satisfiability problems in propositional logic and use highly optimized constraint solvers to systematically search for solutions. This results in exact algorithms that can learn formulas perfectly classifying the input data. However, such exact algorithms suffer from the limitation that they are susceptible to failure in the presence of noise. In particular, trying to learn a formula that perfectly classifies a noisy sample often results in a complex formula, hampering interpretability.

To alleviate the limitation of the earlier approaches[1], in this chapter, we present two novel algorithmic frameworks for learning temporal logic formulas from a sample consisting of positive and negative examples. Based on these frameworks, we devise algorithms for learning formulas in both $\text{LTL}_\text{f}$ and STL. While our presented algorithms learn temporal logic formulas over finite horizon, they can be seamlessly extended to also learn formulas over infinite horizon with minor modifications, which we explicitly mention in the respective sections.

The general goal of our algorithms is to learn a concise (and thus, interpretable) formula that achieves a low *loss* on the sample, where loss $l(\mathcal{S}, \varphi)$ refers to the fraction of examples in the sample $\mathcal{S}$ that the learned formula $\varphi$ misclassified. The precise problem solved by our algorithms is the following: given a sample $\mathcal{S}$ partitioned into a set of positive and a set of negative examples, and a threshold $\kappa$, find a minimal formula $\varphi$ that has $l(\mathcal{S}, \varphi) \leq \kappa$.

Our algorithmic frameworks derive ideas from the SAT-based learning algorithms introduced by Neider et al. [169]. Our first framework reduces the problem of learning a formula to problems in *maximum satisfiability*. Roughly speaking, in this framework, we first encode the learning problem using formulas having appropriate weights assigned to various clauses. We then search for such assignments to the formulas that maximize the total weight of the satisfied clauses. Finally, using an assignment that maximizes the weights of the satisfied clauses, we construct an appropriate formula minimizing loss in a straightforward manner. We refer to Sections 4.1 and 4.2 for the details of the first framework for $\text{LTL}_\text{f}$ and STL, respectively.

The first framework constructs a series of monolithic formulas to encode the learning problem and is, thus, often inefficient for learning larger formulas. Our second framework solves the learning problem by dividing the problem into smaller subproblems based on a decision tree learning algorithm. Instead of finding formulas that achieve a loss of less than $\kappa$ in one step, we exploit algorithms from the first framework to learn small formulas in $\text{LTL}_\text{f}$ or STL for each decision node in the tree. We refer to Section 4.3 for the details of this decision tree-based algorithmic framework.

We have implemented a prototype of our algorithms in a publicly available tool. We have also verified the efficacy of our tool on synthetic as well as real-world data. From our observations, we conclude that our algorithms are effective in learning concise $\text{LTL}_\text{f}$ and STL formulas, particularly from the samples that contain noise. We refer to Section 4.4 for the discussion on both the implementation and its performance on synthetic and real-world examples. Finally, we conclude and provide direction for future works in Section 4.5.

### Related Work

The most prominent works in the area of LTL learning are the works by Neider et al. [169] (which is the basis of this work) and Camacho et al. [48]. Both of these works exploit SAT-based methods. While Neider et al. [169] uses a syntax DAG representation of LTL for the SAT formulation, Camacho et al. [48] use alternating finite automata (AFA). However, both works suffer from failure when the input sample consists of noise.

[1]When this research was done, existing works that handle noisy data relied predominately on formula templates [137, 37, 135]. Some subsequent works [153, 88] explored other neural-network based approaches, which do not provide theoretical guarantees.

The work by Kim et al. [135] is a prominent work that can learn LTL formulas from noisy samples. They exploit the Bayesian inference problem for learning satisfactory LTL formulas from noisy data. They, however, rely on templates for the learned LTL formulas that are often undesirable.

The work of Bombara et al. [37] is one of the first works in the learning Signal Temporal Logic (STL) formulas. Their algorithm also relies on decision trees for learning STL classifiers. While their algorithm can, in fact, learn STL formulas with arbitrary misclassification error on the data, the STL formulas used for the nodes of the decision trees come from a predefined set. Another notable work is by Mohammadinejad et al. [160], who present an algorithm for searching STL formulas using enumerative search. They exploit STL grammar to iteratively generate all STL formulas of a particular size. Further, they employ strategies to eliminate equivalent formulas by checking the semantics of STL on the sample. Our work, in contrast, relies on constraint solvers to search for formulas and thus, will benefit from any advancement in solver technologies. There are several other works in the general area of STL mining [121, 127]. The problem setting of such works is different from ours. In particular, these works aim at extracting STL patterns from data which necessarily need not separate two classes of trace.

In general, the problem of learning temporal logic has been in the spotlight for a number of years. Clear evidence of the fact is the variety of temporal logics for which the learning problem has been looked at—Past Time Linear Temporal Logic (PLTL) [8], Property Specification Language (PSL) [195], Interval Temporal Logic [42] and several others [223, 224, 225].

## 4.1 Learning minimal LTL from Noise

We discuss the details of the first algorithmic framework that relies on maximum satisfiability for learning $\text{LTL}_\text{f}$ formulas, robust to noisy data. Towards this, we formally introduce the details of the problem and the prerequisites for solving it.

### 4.1.1 Problem Formulation

The input data for this problem is the standard for passive learning: a sample $\mathcal{S} = (P, N)$ of finite traces from $(2^{\mathcal{P}})^*$, partitioned into a set $P$ of positive examples and a set $N$ of negative, such that $P \cap N = \emptyset$. We denote the number of traces in a sample $\mathcal{S}$ by $\|\mathcal{S}\| = |P| + |N|$ (note, this is different from the size $|\mathcal{S}|$, which counts the total number of symbols appearing in $\mathcal{S}$).

We now define a *loss* function which, intuitively, evaluates how "well" an $\text{LTL}_\text{f}$ formula $\varphi$ classifies a sample. For this, we exploit a functional view on the semantics of $\text{LTL}_\text{f}$. In particular, we rely on the valuation function $V(\varphi, u)$, which is defined as follows: $V(\varphi, u) = 1$ if $u \models_\text{f} \varphi$, otherwise 0.

While there are numerous ways of defining *loss* (e.g., quadratic loss function, regret, etc.), we use the following natural definition:

$$l(\mathcal{S}, \varphi) = \frac{\sum_{u \in P}(1 - V(\varphi, u)) + \sum_{u \in N} V(\varphi, u)}{\|\mathcal{S}\|}, \tag{4.1}$$

which calculates the fraction of traces in $\mathcal{S}$ which the $\text{LTL}_\text{f}$ formula $\varphi$ misclassified.

Having defined the setting, we now formally describe the problem we solve:

**Problem 2.** *Given a sample $\mathcal{S} = (P, N)$ and threshold $\kappa \in [0, 1]$, learn an $\text{LTL}_f$ formula $\varphi$ such that $l(\mathcal{S}, \varphi) \leq \kappa$.*

Intuitively, the margin on the achieved loss $\kappa$ allows for a bounded fraction of the traces to be considered as noise.

The above problem, generally speaking, is trivial if no constraint is imposed on the size of the output formula since one can always find a large $\text{LTL}_\text{f}$ formula with zero loss on a given sample, as indicated by the following remark.

**Remark 1.** *Given sample $\mathcal{S}$, there exists an $\text{LTL}_f$ formula $\varphi$ such that $l(\mathcal{S}, \varphi) = 0$.*

One can construct such a formula by enumerating the differences in the positive and negative traces; formally, it is $\bigvee_{u \in P} \bigwedge_{v \in N} \varphi_{u,v}$, where $\varphi_{u,v}$ is a formula that uses $\mathsf{X}$ operator and propositions to specify the difference between $u$ and $v$. Such a formula, however, is large in size (of the order of $\mathcal{O}(|P| \cdot |N| \cdot max_{u \in P \cup N} |u|)$) and it does not help towards the goal of learning a concise description of the data.

In the next section, we present an algorithm to learn minimal $\text{LTL}_\text{f}$ formulas based on maximum satisfiability, which is our first algorithmic framework.

### 4.1.2 MaxSAT-based Algorithm

Our solution to Problem 2 relies on MaxSAT solvers, which we introduce next.

**MaxSAT** MaxSAT is an extension of the SAT problem, which asks to find an assignment that maximizes the number of satisfied clauses in a given propositional formula provided in CNF. For our problem, we rely on a more general variant of MaxSAT, known as Partial Weighted MaxSAT. In this variant, a weight function $w \colon \mathcal{C} \mapsto \mathbb{R} \cup \{\infty\}$ assigns a weight to every clause in the set of clauses $\mathcal{C}$ of a propositional formula. The problem is to then find a valuation $v$ that maximizes $\Sigma_{C_i \in \mathcal{C}} w(C_i) \cdot V(C_i, v)$.

While the MaxSAT problem and its variants can be solved using dedicated solvers, standard SMT solvers like Z3 [164] are also able to handle such problems. According to terminology derived from the theory behind such solvers, clauses $C_i$ for which $w(C_i) = \infty$ are termed as *hard* constraints, while clauses $C_i$ for which $w(C_i) < \infty$ are termed as *soft* constraints. Given a propositional formula with weights assigned to clauses, MaxSAT solvers try to find a valuation that satisfies all the hard constraints and maximizes the total weight of the soft constraints that can be satisfied.

Given that we are using MaxSAT solvers that possess the capability of handling Partial Weighted MaxSAT problems, we can solve a stronger version of Problem 2. In this stronger version, the loss based on which we search for $\text{LTL}_\text{f}$ formulas takes the following form:

$$wl(\mathcal{S}, \varphi, \Omega) = \sum_{u \in P} \Omega(u) \cdot (1 - V(\varphi, u)) + \sum_{u \in N} \Omega(u) \cdot V(\varphi, u), \tag{4.2}$$

**Algorithm 5** Learning algorithm based on maximum satisfiability

**Input:** Sample $\mathcal{S}$, weight-function $\Omega$, Threshold $\kappa$

1: $n \leftarrow 0$
2: **while** sum of weights of soft constraints $\geq 1 - \kappa$ **do**
3: $\quad n \leftarrow n + 1$
4: $\quad$ Construct formula $\Phi_n^{\mathcal{S}} = \Phi_n^{LTL} \wedge \Phi_n^{sem} \wedge \Phi_n^{con}$
5: $\quad$ Assign weights to soft constraints $\Phi_n^{con}$ in $\Phi_n^{\mathcal{S}}$:
$\qquad w(y_{1,0}^u) = \Omega(u)$ for $u \in P$, and $w(\neg y_{1,0}^u) = \Omega(u)$ for $u \in N$
6: $\quad$ Find assignment $v$ using MaxSAT solver
7: **end while**
8: **return** $\varphi_v$

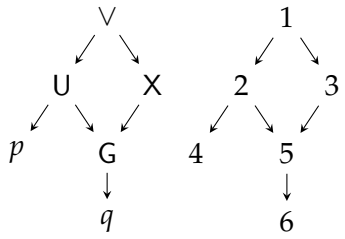



FIGURE 4.1: Syntax DAG and identifiers of the formula $(p \,\mathsf{U}\, \mathsf{G}\, q) \vee \mathsf{X}(\mathsf{G}\, q)$

where $\Omega$ is a function that assigns a positive real-valued weight to each $u$ in the sample in such a way that $\sum_{u \in P \cup N} \Omega(u) = 1$. Observe that by considering $\Omega(u) = 1/|\mathcal{S}|$ for all traces in the sample, we have exactly $wl(\mathcal{S}, \varphi, \Omega) = l(\mathcal{S}, \varphi)$ which is used in Problem 2. In this section, we will solve the stronger version since not only does it enable us to solve Problem 2 but also makes our algorithmic framework versatile enough to assist the decision trees learning algorithm, described in Section 4.3.

For solving this problem, we devise an algorithm based on ideas from the exact learning algorithm of Neider et al. [169]. Following their algorithm, we translate the problem of learning $\text{LTL}_\text{f}$ formulas into problems in Partial Weighted MaxSAT and then use an optimized MaxSAT solver to find a solution. More precisely, we construct a propositional formula $\Phi_n^{\mathcal{S}}$ and assign weights to its clauses in such a way that an assignment $v$ that satisfies all the hard constraints of $\Phi_n^{\mathcal{S}}$ has the following two properties:

1. it contains sufficient information to extract an $\text{LTL}_\text{f}$ formula $\varphi_v$ of size $n$, and
2. the sum of weights of the soft constraints satisfied by it is equal to $1 - wl(\mathcal{S}, \varphi_v, \Omega)$.

To obtain a complete algorithm, we increase the value of $n$ (starting from 1) until we find an assignment $v$ of $\Phi_n^{\mathcal{S}}$ that satisfies the hard constraints and ensures that the sum of weights of the soft constraints is greater than $1 - \kappa$. The termination of this algorithm is guaranteed by the existence of an $\text{LTL}_\text{f}$ formula with zero loss on the sample (see Remark 1).

On a technical level, the formula $\Phi_n^{\mathcal{S}}$ in Algorithm 5 is the conjunction

$$\Phi_n^{\mathcal{S}} = \Phi_n^{LTL} \wedge \Phi_n^{sem} \wedge \Phi_n^{con}, \tag{4.3}$$

where $\Phi_n^{LTL}$ encodes the *structure* of the prospective $\text{LTL}_\text{f}$ formula $\varphi$ of size $n$, $\Phi_n^{sem}$ ensures that the *semantics* of $\text{LTL}_\text{f}$ is used to interpret $\varphi$ on the traces in $\mathcal{S}$, and $\Phi_n^{con}$ tracks whether $\varphi$ is consistent with $\mathcal{S}$. We now explain each of the conjuncts in greater detail.

**Structural Constraints.** For designing the formula $\Phi_n^{LTL}$, we rely on the canonical *syntax DAG* representation of $\text{LTL}_\text{f}$ formulas (see Section 2.2.3). To uniquely identify the nodes in a syntax DAG, we assign identifiers $\{1,\ldots,n\}$ in such a way that the root node is always indicated by 1 and every node has an identifier smaller than that of its children if it has any. An example of identifiers of a syntax DAG is shown in Figure 4.1. We denote the subformula rooted at Node $i$ as $\varphi[i]$.

To encode the structure of a syntax DAG using propositional logic, we introduce the following propositional variables: (i) $x_{i,\lambda}$ for $i \in \{1,\cdots,n\}$ and $\lambda \in \Lambda$, and (ii) $l_{i,j}$ and $r_{i,j}$ for $i \in \{1,\cdots,n-1\}$ and $j \in \{i+1,\cdots,n\}$. The variable $x_{i,\lambda}$ encodes that Node $i$ is labeled by operator $\lambda$ (includes propositional variables). The variable $l_{i,j}$ (respectively, $r_{i,j}$) encodes that the left (respectively, the right) child of Node $i$ is Node $j$. Based on the meaning of the variables, we must have $x_{1,\wedge}$, $l_{1,2}$, and $r_{1,3}$ to be true in order to obtain a syntax DAG (similar to the one in Figure 4.1) where Node 1 is labeled with $\wedge$, has the left child to be Node 2, and the right child to be Node 3.

We now introduce constraints on the variables to ensure that they encode a valid syntax DAG. Towards this, we impose the following constraints:

$$\Big[\bigwedge_{1\le i\le n}\bigvee_{\lambda\in\Lambda} x_{i,\lambda}\Big] \wedge \Big[\bigwedge_{1\le i\le n}\bigwedge_{\lambda\neq\lambda'\in\Lambda} \neg x_{i,\lambda} \vee \neg x_{i,\lambda'}\Big] \tag{4.4}$$

$$[\bigwedge_{1\le i<n}\bigvee_{i<j\le n} l_{i,j}] \wedge [\bigwedge_{1\le i<n}\bigwedge_{i<j\le j'\le n} \neg l_{i,j} \vee \neg l_{i,j'}] \tag{4.5}$$

$$[\bigwedge_{1\le i<n}\bigvee_{i<j\le n} r_{i,j}] \wedge [\bigwedge_{1\le i<n}\bigwedge_{i<j\le j'\le n} \neg r_{i,j} \vee \neg l_{i,j'}] \tag{4.6}$$

$$\bigvee_{p\in\mathcal{P}} x_{n,p} \tag{4.7}$$

Formula 4.4 ensures that each node of the syntax DAG has a unique label. Similarly, Formulas 4.5 and 4.6 ensure that each node of a syntax DAG has a unique left and a unique right child, respectively. Finally, Formula 4.7 ensures that Node $n$ is labeled by a proposition.

Observe that from a valuation $v$ satisfying $\Phi_n^{LTL}$ one can extract a unique syntax DAG describing an $\text{LTL}_\text{f}$ formula $\varphi_v$ as follows: label Node $i$ of the syntax DAG with the unique $\lambda$ for which $v(x_{i,\lambda}) = 1$; assign Node $n$ to be the root node, and assign edges from a node to its children based on the values of $l_{i,j}$ and $r_{i,j}$.

**Semantic Constraints.** Towards the definition of the formula $\Phi_n^{sem}$, we define propositional formulas $\Phi_n^u$ for each trace $u$ that tracks the semantics of the prospective $\text{LTL}_\text{f}$ formula on $u$. These formulas are built using variables $y_{i,t}^u$ for $i \in \{1,\ldots,n\}$ and $t \in \{0,\ldots,|u|-1\}$. The variable $y_{i,t}^u$ corresponds to the satisfaction value $V(\varphi[i], u[t,|u|))$ of $\varphi[i]$ on $u$ at timepoint $t$.

Again, to make sure that these variables have the desired meaning, we impose constraints based on the semantics of the $\text{LTL}_\text{f}$ operators.

$$\bigwedge_{1\leq i\leq n}\bigwedge_{p\in\mathcal{P}} x_{i,p} \rightarrow \Big[\bigwedge_{0\leq t<|u|} \begin{cases} y^u_{i,t} \text{ if } p \in u[t] \\ \neg y^u_{i,t} \text{ if } p \notin u[t] \end{cases}\Big] \tag{4.8}$$

$$\bigwedge_{\substack{1\leq i\leq n\\ i<j\leq n}} x_{i,\neg} \wedge l_{i,j} \rightarrow \Big[\bigwedge_{0\leq t<|u|} \Big[y^u_{i,t} \leftrightarrow \neg y^u_{j,t}\Big]\Big] \tag{4.9}$$

$$\bigwedge_{\substack{1\leq i\leq n\\ i<j,j'\leq n}} x_{i,\vee} \wedge l_{i,j} \wedge r_{i,j'} \rightarrow \Big[\bigwedge_{0\leq t<|u|} \Big[y^u_{i,t} \leftrightarrow y^u_{j,t} \vee y^u_{j',t}\Big]\Big] \tag{4.10}$$

$$\bigwedge_{\substack{1\leq i\leq n\\ i<j\leq n}} x_{i,\mathsf{X}} \wedge l_{i,j} \rightarrow \Big[\bigwedge_{0\leq t<|u|-1} \Big[y^u_{i,t} \leftrightarrow y^u_{j,t+1}\Big] \wedge \neg y^u_{i,|u|-1}\Big] \tag{4.11}$$

$$\bigwedge_{\substack{1\leq i\leq n\\ i<j,j'\leq n}} x_{i,\mathsf{U}} \wedge l_{i,j} \wedge r_{i,j'} \rightarrow \Big[\bigwedge_{0\leq t<|u|} \Big[y^u_{i,t} \leftrightarrow \bigvee_{t\leq t'<|u|} \Big[y^u_{j',t'} \wedge \bigwedge_{t\leq t''<t'} y^u_{j,t''}\Big]\Big]\Big] \tag{4.12}$$

Formula 4.8 implements the semantics of propositions and states that if Node $i$ is labeled with $p \in \mathcal{P}$, then $y^u_{i,t}$ is set to 1 if and only if $p \in u[i]$. Formulas 4.9 and 4.10 implement the semantics of negation and disjunction, respectively: if Node $i$ is labeled with $\neg$ and Node $j$ is its left child, then $y^u_{i,t}$ equals the negation of $y^u_{j,t}$; on the other hand, if Node $i$ is labeled with $\vee$, Node $j$ is its left child, and Node $j'$ is its right child, then $y^u_{i,t}$ equals the disjunction of $y^u_{j,t}$ and $y^u_{j',t}$. Formula 4.11 implements the semantics of the X-operator and states that if Node $i$ is labeled with X and its left child is Node $j$, then $y^u_{i,t}$ equals $y^u_{j,t+1}$. Finally, Formula 4.12 implements the semantics of the U-operator; it states that if Node $i$ is labeled with U, its left child is Node $j$, and its right child is Node $j'$, then $y^u_{i,t}$ is set to 1 if and only if there exists a timepoint $t'$ for which $y^u_{j',t'}$ is set to 1 and for all timepoints $t$ lying between $t$ and $t'$, $y^u_{j,t}$ is set to 1. The formula $\Phi^n_u$ is the conjunction of all such semantic constraints.

The above constraints are similar to the ones proposed by Neider and Gavran, except that they have been adapted to comply with the semantics of $\text{LTL}_\text{f}$. We make an important remark here: if we use the same constraints as provided by Neider and Gavran, then the approach works seamlessly for LTL (over infinite traces).

We now define $\Phi^{sem}_n$ as follows:

$$\Phi^{sem}_n := \bigwedge_{u\in P\cup N} \Phi^n_u, \tag{4.13}$$

which is simply the conjunction of the semantic constraints $\Phi^u_n$ for each trace $u$ in $\mathcal{S}$.

**Consistency Constraints.** We define $\Phi^{con}_n$ to be the following:

$$\Phi^{con}_n = \bigwedge_{u\in P} y^u_{1,0} \wedge \bigwedge_{u\in N} \neg y^u_{1,0} \tag{4.14}$$

This constraint ensures that the positive traces in $P$ must satisfy the prospective $\text{LTL}_\text{f}$ formula, while the negative traces in $N$ must not.

**Weight assignment.** For assigning weights to the clauses of $\Phi_n^{\mathcal{S}}$, we first convert the formulas $\Phi_n^{LTL}$ and $\Phi_n^{sem}$ into CNF. Towards this, we simply exploit the Tseitin transformation [212], which converts a formula into an equivalent formula in CNF whose size is linear in the size of the original formula.

We now assign weights to constraints starting with the hard constraints as follows: $w(\Phi_n^{LTL}) = \infty$ and $w(\Phi_n^{sem}) = \infty$, meaning $w(\Phi_n^u) = \infty$ for all traces $u \in P \cup N$. Here, $w(\Phi) = \omega$ is a shorthand to denote $w(C_i) = \omega$ for all clauses $C_i$ in $\Phi$, where $\omega \in \mathbb{R}$. The constraint $\Phi_n^{LTL}$ is hard since that ensures that the syntax DAG we obtain from $\Phi_n^{LTL}$ is of a valid $\text{LTL}_\text{f}$ formula. The constraint $\Phi_n^{sem}$ is also hard since it ensure that we rely on the proper $\text{LTL}_\text{f}$ semantics to interpret the prospective formula on the traces in $\mathcal{S}$.

The soft constraints are the ones that enforce correct classification, that is, $\Phi_n^{con}$, and we assign weights to them as follows: $w(y_{1,0}^u) = \Omega(u)$ for all $u \in P$ and $w(\neg y_{1,0}^u) = \Omega(u)$ for all $u \in N$. Recall that $\Omega$ refers to the function assigning weights to the traces.

To prove the correctness of our learning algorithm, we first ensure that the formula $\Phi_n^{\mathcal{S}}$ along with the weight assigned to its clauses serves our purpose.

**Lemma 3.** *Let $\mathcal{S}$ be a sample, $\Omega$ the weight function, $n \in \mathbb{N} \setminus \{0\}$ and $\Phi_n^{\mathcal{S}}$ the formula with the associated weights as defined above. Then,*

1. *the hard constraints are satisfiable; and*
2. *if $v$ is an assignment that satisfies the hard constraints and maximizes the sum of the weight of the satisfied soft constraints, then $\varphi_v$ is an $\text{LTL}_f$ formula of size $n$, such that $wl(\mathcal{S}, \varphi_v, \Omega) \leq wl(\mathcal{S}, \varphi, \Omega)$ for all $\text{LTL}_f$ formulas $\varphi$ of size $n$.*

*Proof.* The hard constraints of $\Phi_n^{\mathcal{S}}$ are $\Phi_n^{LTL}$ and $\Phi_n^{sem}$. Now, $\Phi_n^{LTL}$ is satisfiable since there always exists a valid $\text{LTL}_\text{f}$ formula of size $n$. As a result, using the syntax DAG of a $\text{LTL}_\text{f}$ formula of size $n$, we can find an assignment to the variables of $\Phi_n^{LTL}$ that makes it satisfiable. Next, the formula $\Phi_n^{sem}$ consists of the constraints $\Phi_u^n$ and each of them simply track the valuation of the prospective formula on traces $u$ in $\mathcal{S}$. One can easily find an assignment of the variables of $\Phi_u^n$ using the semantics of $\text{LTL}_\text{f}$.

For proving the second part, let us assume $v$ to be an assignment that satisfies the hard constraints. We now claim that the sum of the weights of the satisfied soft constraints is equal to $1 - wl(S, \varphi_v, \Omega)$. If we prove this, then we have the following: if $v$ is an assignment that maximizes the weight of the satisfied soft constraints, then this directly implies that $\varphi_v$ minimizes the $wl$ function. To avoid notational clutter, we rely on a function $u$ that labels traces: $b(u) = 1$ if $u \in P$, $0$ if $u \in N$. We now have the following:

$$\begin{aligned} wl(S, \varphi_v, \Omega) &= \sum_{V(\varphi_v, u) \neq b(u)} \Omega(u) = \sum \Omega(u) - \sum_{V(\varphi_v, u) = b(u)} \Omega(u) \\ &= 1 - \sum_{V(\varphi_v, u) = b(u)} \Omega(u) = 1 - \sum_{v(y_{1,0}^u) = b(u)} \Omega(u) \end{aligned}$$

All the summations in the above equation are over all traces $u \in P \cup N$ in the sample. Moreover, the quantity $\sum_{v(y^u_{1,0})=b} \Omega(u)$, appearing in the final line, refers to sum of the weights of the satisfied soft constraints, since the constraints in which $v(y^u_{1,0}) = b(u)$ are the ones that are satisfied. □

The termination and the correctness of Algorithm 5, which is established using the following theorem, is a consequence of Lemma 3.

**Theorem 3.** *Given a sample $\mathcal{S}$ and threshold $\kappa \in \mathbb{R}$, Algorithm 5 terminates and learns an LTL$_f$ formula $\varphi$ that has $wl(\mathcal{S}, \varphi, \Omega) \leq \kappa$ and is the minimal in size among all LTL$_f$ formulas that have $wl(\mathcal{S}, \varphi, \Omega) \leq \kappa$.*

*Proof.* The termination of Algorithm 5 is guaranteed by the fact that there always exists an LTL$_\text{f}$ formula $\varphi$ for which $wl(\varphi, S, \Omega) = 0$ as indicated by Remark 1. Second, the fact that $\varphi$ has $wl(\varphi, S, \Omega) \leq \kappa$ is a consequence of Lemma 3. Finally, the minimality of the formula is a consequence of the fact that Algorithm 5 searches for an LTL$_\text{f}$ formula in increasing order of size. □

## 4.2 Learning Minimal STL from Noise

In this section, we formally introduce the learning problem for STL and the necessary ingredients. We then discuss how to adapt the first framework to learn STL formulas; in particular, focussing on the differences between this algorithm and the one in the previous section, Section 4.1.

**Signals.** A signal is a time series that indicates the evolution of system features over time. Unlike traces, features assume real values in signals. Thus, formally, a signal is defined as a function $u : \mathbb{T} \to \mathbb{R}^m$, which assigns values to $m$ system features over a time domain $\mathbb{T}$. In this chapter, we assume the time domain $\mathbb{T} = \{0, \cdots, n\} \subset \mathbb{N}$ to be discrete and finite. Moreover, given a signal $u$ and $t \in \mathbb{T}$, we use $u[t]$ to denote the value of $u$ at timepoint $t$, and $u_j[t]$ to denote the value of its $j^{\text{th}}$ feature. Since we use discrete time, we can define the length of a signal by $|u| = |\mathbb{T}|$. Finally, we denote the set of all signals by $(\mathbb{R}^m)^*$.

**Signal Temporal Logic.** Signal Temporal Logic (STL) is an extension of LTL$_\text{f}$ defined over signals [10, 157], which branches out LTL$_\text{f}$ in two directions: it employs temporal operators defined over time intervals, and it is interpreted over signals [44]. Formulas in STL—denoted by small Greek letters—are defined inductively by:

$$\varphi := \pi \mid \neg\varphi \mid \varphi \vee \varphi \mid \varphi \,\mathsf{U}_I\, \varphi$$

Here, $\pi$ is a predicate of the form $f_\pi(x_1, \ldots, x_m) \geq \theta$ where $f_\pi : \mathbb{R}^m \to \mathbb{R}$ is a function over the features, and $\theta \in \mathbb{R}$ is a predicate threshold. $I$ is a time interval of the form $I := [a, b)$, with $0 \leq a < b$ two *integers*. The extended set of all operators is defined as

$\Lambda = \{\neg, \vee, \wedge, \rightarrow, \mathsf{U}_I, \mathsf{F}_I, \mathsf{G}_I\} \cup \{\pi, \cdots\}$, where $\mathsf{U}_I$ is parameterized with $[a, b)$, and each $\pi$ is parameterized with $\theta$.

We interpret STL over *finite* signals, that is, signals that only last for a finite duration. For a finite signal $u$, a timepoint $t \in \mathbb{N}$ and an STL formula $\varphi$, we define the semantics $u, t \models \varphi$ inductively as follows:

$$
\begin{aligned}
u, t &\models_{\mathrm{f}} \pi \text{ if and only if } f_\pi(u[t]) \geq \theta \\
u, t &\models_{\mathrm{f}} \neg\varphi \text{ if and only if } u, t \not\models_{\mathrm{f}} \varphi \\
u, t &\models_{\mathrm{f}} \varphi_1 \vee \varphi_2 \text{ if and only if } u, t \models_{\mathrm{f}} \varphi_1 \text{ or } u, t \models_{\mathrm{f}} \varphi_2 \\
u, t &\models_{\mathrm{f}} \varphi_1 \,\mathsf{U}_{[a,b)}\, \varphi_2 \text{ if and only if for some } t + a \leq t' < \min(t + b, |u|) : u, t' \models \varphi_2 \\
&\qquad\qquad \text{and for all } t \leq t'' < t' : u, t'' \models \varphi_1.
\end{aligned}
$$

If $u, 0 \models \varphi$, we simply write $u \models \varphi$ and say that $u$ satisfies $\varphi$, or alternatively, $\varphi$ holds on $u$. We again exploit the functional view on the semantics of STL using the valuation function $V$, whose definition mirrors that of $\mathrm{LTL}_\mathrm{f}$.

### 4.2.1 Problem Formulation

As input, we have a sample $\mathcal{S} = (P, N)$ of finite signals from $(\mathbb{R}^m)^*$, partitioned into a set $P$ of positive examples and a set $N$ of negative, such that $P \cap N = \emptyset$. Additionally, we have a finite set of predicate templates $\Pi$. The predicate template provides a means for guessing the prospective atoms for the STL formulas. In this problem, we assume the templates in $\Pi$ must have the function $f_\pi(\cdot)$ specified, while the predicate threshold $\theta$ may remain unspecified.

We state the problem of learning an STL formula from noisy data as follows:

**Problem 3.** *Given a sample $\mathcal{S} = (P, N)$ of finite signals and a set of predicate templates $\Pi$, learn a minimal STL formula $\varphi$ using templates from $\Pi$ such that $l(\mathcal{S}, \varphi) \leq \kappa$.*

Unlike Problem 2, the existence of a solution to Problem 3 is not always guaranteed. This is because the existence of an STL formula with zero loss depends on the input predicate templates. Thus, in order to guarantee the existence of a solution, we restrict the predicate templates to have a specific structure. In particular, we propose the following set of templates: $\Pi = \{u_j \geq \theta \mid 1 \leq j \leq m\}$. Note that such a restriction is required only for the theoretical guarantees. Our algorithm,, in fact,, works for any arbitrary set of templates if there exists an appropriate STL formula using them.

The restriction discussed provides us with the following guarantee:

**Remark 2.** *Given sample $\mathcal{S}$ and predicate templates $\Pi = \{u_j \geq \theta \mid 1 \leq j \leq m\}$, there exist an STL formula $\varphi$ using templates from $\Pi$ such that $l(\mathcal{S}, \varphi) = 0$.*

We briefly discuss how to construct the (large) STL formula with predicate templates $\Pi = \{u_j \geq \theta \mid 1 \leq j \leq m\}$. Similar to the formula from Remark 1, the formula here is $\bigvee_{u \in P} \bigwedge_{v \in N} \varphi_{u,v}$, where $\varphi_{u,v}$ is an STL formula that distinguishes between positive signal $u$ and negative signal $v$. We construct these formulas $\varphi_{u,v}$ as $\mathsf{F}_{[t,t+1)}\, u_j \geq \frac{u_j[t] + v_j[t]}{2}$, where $t$ is

the time-point and $j$ is the coordinate where $u$ and $v$ differ. Here, we assume $u_j[t] > v_j[t]$; if $u_j[t] < v_j[t]$ we can simply include a negation in the predicate of $\varphi_{u,v}$.

### 4.2.2 MaxSMT-based Algorithm

Our solution to the learning problem relies on MaxSMT solvers, which we will introduce next.

**MaxSMT.** Unlike SAT, SMT deals with the satisfiability of first-order formulas over background theories (see Section 2.2.2). Similar to MaxSAT, MaxSMT is the problem of finding assignments that maximize the number of satisfiable clauses [199]. The formal problem definition remains the same as in the case of MaxSAT. For our algorithm, we will exploit the Partial Weighted MaxSMT for the theory of Linear Real Arithmetic (LRA). Standard SMT solvers like Z3 [164] can handle such problems.

The algorithm for learning STL formulas follows the same framework as that for $\text{LTL}_\text{f}$ formulas. However, the syntax and semantics of STL being different from $\text{LTL}_\text{f}$, we modify the certain conjuncts of the propositional formula $\Phi_n^{\mathcal{S}}$. In particular, the structural constraint $\Phi_n^{STL}$, additionally, encodes the time intervals $I$ for temporal operators $\mathsf{U}_I$ and the value of the predicate thresholds $\theta$ for the predicates. The semantic constraints $\Phi_n^{sem}$ change to ensure that proper semantics of STL is used.

**Structural Constraints.** To include the features of STL in the structure of the syntax DAG, we introduce the following additional variables: $a_i \in \mathbb{N}$ and $b_i \in \mathbb{N}$ for $i \in \{1, \ldots, n\}$, and $\theta_i \in \mathbb{R}$ for $i \in \{1, \ldots, n\}$. The variable $a_i$ (respectively, $b_i$) encodes the lower (respectively, the upper) bound of the interval $I$ of a temporal operator (that is, one of $\mathsf{U}_I$, $\mathsf{F}_I$ or $\mathsf{G}_I$) labeled at Node $i$. The variable $\theta_i$ encodes the value of the threshold when a predicate template from $\Pi$ is labeled at Node $i$.

In addition to the constraints specified in Section 4.1, $\Phi_n^{STL}$ has constraints $0 \leq a_i < b_i$ for $i \in \{1, \cdots, n\}$. By imposing these structural constraints, one can ensure that, from a valuation $v$ of $\Phi_n^{STL}$, one can extract a unique STL formula $\varphi_v$. In addition to the operators, the valuation $v$ uniquely assigns an interval $[a_i, b_i)$ and a parameter $\theta_i$ at Node $i$ if it is labeled with a temporal operator and a predicate template, respectively.

**Semantic Constraints.** We now define $\Phi_n^u$, which tracks the satisfaction of the prospective STL formula on a signal $u$ in $\mathcal{S}$, as follows:

$$\bigwedge_{1 \leq i \leq n} \bigwedge_{\pi \in \Pi} x_{i,\pi} \rightarrow \Big[ \bigwedge_{0 \leq t < |u|} y_{i,t}^u \leftrightarrow f_\pi(u[t]) \geq \theta_i \Big] \tag{4.15}$$

$$\bigwedge_{\substack{1 \leq i \leq n \\ 1 \leq j < i}} x_{i,\neg} \wedge l_{i,j} \rightarrow \Big[ \bigwedge_{0 \leq t < |u|} \Big[ y_{i,t}^u \leftrightarrow \neg y_{j,t}^u \Big] \Big] \tag{4.16}$$

$$\bigwedge_{\substack{1\le i\le n\\ 1\le j,j'<i}} x_{i,\vee} \wedge l_{i,j} \wedge r_{i,j'} \rightarrow \Big[ \bigwedge_{0\le t<|u|} \Big[ y^u_{i,t} \leftrightarrow y^u_{j,t} \vee y^u_{j',t} \Big] \Big] \tag{4.17}$$

$$\bigwedge_{\substack{1\le i\le n\\ 1\le j,j'<i}} x_{i,\mathsf{U}_I} \wedge l_{i,j} \wedge r_{i,j'} \rightarrow \Big[ \bigwedge_{0\le t<|u|} \Big[ y^u_{i,t} \leftrightarrow \bigvee_{t\le t'<|u|} \Big[ [t+a_i \le t' < \min(t+b_i, |u|)] \wedge y^u_{j',t'} \wedge \bigwedge_{t\le t''<t'} [t+a_i \le t < t'] \rightarrow y^u_{j,t''} \Big] \Big] \Big] \tag{4.18}$$

The constraints above are similar to the corresponding semantic constraints for $\mathrm{LTL_f}$. While Formula 4.18 also similarly implements the semantics of $\mathsf{U}_I$, it differs from Formula 4.12 slightly: it ensures that the timepoints follow the prospective time interval $I$ of a temporal operator. Also, the definition of $\Phi^{sem}_n$ is same as in Formula 4.13.

**Consistency Constraints.** The consistency constraints $\Phi^{con}_n$ remains identical as in Formula 4.14.

**Weight assignment** The assignment of weights remains the same as in $\mathrm{LTL_f}$. In particular, the hard constraints are $w(\Phi^{STL}_n)$ and $w(\Phi^{sem}_n)$. The soft constraints are the ones that enforce correct classification: $w(y^u_{1,0}) = \Omega(u)$ for all $u \in P$ and $w(\neg y^u_{1,0}) = \Omega(u)$ for all $u \in N$.

The correctness of the algorithm adapted to learn STL formulas follows from the correctness of the formula $\Phi^{\mathcal{S}}_n$.

**Theorem 4.** *Given a sample $\mathcal{S}$, predicates template $\Pi$ as indicated in Remark 2 and threshold $\kappa \in \mathbb{R}$, the MaxSMT-based STL learning algorithm terminates and outputs an STL formula $\varphi$ that has $wl(\mathcal{S}, \varphi, \Omega) \le \kappa$ and is minimal in size among all STL formulas that have predicates in $\Pi$ and $wl(\mathcal{S}, \varphi, \Omega) \le \kappa$.*

The proof of the above theorem follows the same reasoning as that of Theorem 3.

## 4.3 Learning Decision Trees over Formulas

In this section, we present our second algorithmic framework for learning temporal logic formulas. While learning using such a framework does not guarantee minimal formulas, on the bright side, we obtain decision trees over temporal logic formulas that are structured and interpretable objects. The framework works identically for learning $\mathrm{LTL_f}$ and STL formulas, and thus, in this section, we only describe the algorithm for learning $\mathrm{LTL_f}$ formulas.

**Decision Trees over $\mathrm{LTL_f}$ formulas** A decision tree over $\mathrm{LTL_f}$ formulas is a tree-like structure where all nodes of the tree are labeled by $\mathrm{LTL_f}$ formulas. While the leaf nodes of a decision tree are labeled by either *true* or *false*, the inner nodes are labeled by (non-trivial) $\mathrm{LTL_f}$ formulas, which represent decisions to predict the class of a trace. Each inner node leads

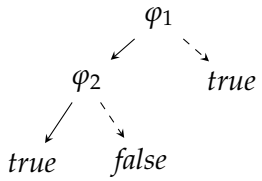



FIGURE 4.2: A decision tree over $\text{LTL}_\text{f}$ formulas

to two subtrees connected by edges, where the left edge is represented with a solid edge and the right edge with a dashed one. Figure 4.2 depicts a decision tree over $\text{LTL}_\text{f}$ formulas.

A decision tree $T$ over $\text{LTL}_\text{f}$ formula corresponds to an $\text{LTL}_\text{f}$ formula $\varphi_t := \bigvee_{\rho \in \Pi} \bigwedge_{\varphi \in \rho} \varphi'$, where $\Pi$ is the set of paths that originate in the root node and end in a leaf node labeled by *true* and $\varphi' = \varphi$ if it appears before a solid edge in $\rho \in \Pi$, otherwise $\varphi' = \neg\varphi$. For the decision tree in Figure 4.2, the equivalent $\text{LTL}_\text{f}$ formula is $(\varphi_1 \wedge \varphi_2) \vee \neg\varphi_1$.

For evaluating a decision tree $T$ on a trace $u$, we use the valuation $V(\varphi_T, u)$ of the equivalent $\text{LTL}_\text{f}$ formula $\varphi$ on $u$. We can, in fact, extend the valuation function and loss function for $\text{LTL}_\text{f}$ formulas to decision trees as $V(T, u) = V(\varphi_T, u)$ and $l(T, \varphi) = l(\mathcal{S}, \varphi)$.

### 4.3.1 Decision-Tree Learning

Our decision tree learning algorithm shares similarities with the class of decision tree learning algorithms known as Top-Down Induction of Decision Trees (TDIDT) [182]. Popular decision tree learning algorithms such as ID3, C4.5, CART are all part of the TDIDT algorithm family. In such algorithms, decision trees are constructed in a top-down fashion by finding suitable features (i.e., predicates over the attributes) of the data to partition it and then applying the same method inductively to the individual partitions.

Algorithm 6 outlines our approach to learning a decision tree over $\text{LTL}_\text{f}$ formulas. In our algorithm, we first check the stopping criterion (Line 1) that is responsible for the termination of the algorithm. If the stopping criterion is met, we return a leaf node. We discuss the exact stopping criterion used in our algorithm in Section 4.3.3.

If the stopping criterion fails, we search for an appropriate $\text{LTL}_\text{f}$ formula $\varphi$ using Algorithm 5 for the current node of the decision tree. Our search for $\varphi$ is based on a *score* function, and we learn the minimal one that achieves a score greater than a user-defined *minimum score* $\mu$ on the sample. The choice of the score function and parameter $\mu$ is a crucial aspect of the algorithm, and we discuss more about this in Section 4.3.2.

Having learned formula $\varphi$, next we split the sample into two sub-samples $\mathcal{S}_1$ and $\mathcal{S}_2$ with respect to $\varphi$: $\mathcal{S}_1$ consists of the traces in $\{u \in P \cup N \mid V(\varphi, u) = 1\}$, while $\mathcal{S}_2$ consists of the traces in $\{u \in P \cup N \mid V(\varphi, u) = 0\}$. In both $\mathcal{S}_1$ and $\mathcal{S}_2$, the label of the traces, that is, whether the traces are positive or negative, remains the same as in the sample $\mathcal{S}$. The final step is to recursively apply the decision tree learning on each of the sub-samples $\mathcal{S}_1$ and $\mathcal{S}_2$ (Line 6) to obtain trees $T_1$ and $T_2$, respectively. The decision tree returned is a tree with root node $\varphi$ and subtrees $T_1$ and $T_2$.

**Algorithm 6** Decision tree learning algorithm

**Input:** Sample $\mathcal{S}$, Minimum score value $\mu$, Threshold $\kappa$
**Parameter:** Stopping criterion *stop*, Score function $s$

1: **if** $stop(\mathcal{S}, \kappa)$ **then**
2: **return** $leaf(\mathcal{S})$
3: **else**
4: Learn minimal formula $\varphi$ with $s(\mathcal{S}, \varphi) \geq \mu$ using Algorithm 5
5: Split $\mathcal{S}$ into $\mathcal{S}_1$, $\mathcal{S}_2$ using $\varphi$
6: Learn trees $T_1$, $T_2$ by recursively applying algorithm to $\mathcal{S}_1$ and $\mathcal{S}_2$
7: **return** decision tree with root node $\varphi$ and subtrees $T_1$, $T_2$
8: **end if**

### 4.3.2 LTL Formulas for Decision Nodes

Ideally, we aim to learn $\text{LTL}_\text{f}$ formulas at each decision node that, in addition to being small, also ensure that the resulting sub-samples after a split are as "homogenous" as possible. In simpler words, we want the sub-samples obtained after a split to predominantly consist of traces with similar labels (that is, positive or negative). More homogenous splits result in early termination of the algorithm, resulting in small decision trees. To achieve this, one can simply learn a minimal $\text{LTL}_\text{f}$ formula that perfectly classifies (that is, consistent with) the sample. While in principle, this solves our problem, in practice, learning an $\text{LTL}_\text{f}$ formula that perfectly classifies a sample is a computationally expensive process [169]. Moreover, it results in a trivial decision tree consisting of a single decision node. Thus, to avoid that, we wish to learn concise $\text{LTL}_\text{f}$ formulas that classify most traces correctly on the given sample.

To mechanize the search for concise $\text{LTL}_\text{f}$ formulas for the splits, we measure the quality of an $\text{LTL}_\text{f}$ formula using a *score* function. In our algorithm, we use this function to learn a minimal $\text{LTL}_\text{f}$ formula having a score greater than a user-defined threshold $\mu$. The parameter $\mu$ regulates the trade-off between the height of the tree and the size of the $\text{LTL}_\text{f}$ formulas in the decision nodes of a tree. While all TDIDT algorithms involve certain metrics (e.g., Gini impurity, entropy) to measure the efficacy of a feature to perform a split, these metrics are based on non-linear operations on the fraction of examples of each class in a sample. Searching $\text{LTL}_\text{f}$ formulas, however, based on such metrics, cannot be handled using a MaxSAT framework.

A natural choice for the score function is the following:

$$s_l(\mathcal{S}, \varphi) = 1 - l(\mathcal{S}, \varphi), \tag{4.19}$$

which relies on the loss function. A formula $\varphi$ with $s_l(\mathcal{S}, \varphi) \geq \mu$ is a formula with $l(\mathcal{S}, \varphi) \leq 1 - \mu$. Thus, for learning $\text{LTL}_\text{f}$ formulas with score greater than $\mu$, we invoke Algorithm 5 to produce a minimal $\text{LTL}_\text{f}$ formula $\varphi$ with $l(\mathcal{S}, \varphi) \leq 1 - \mu$. Note that, for this score, one must choose the $\mu$ to be smaller than $1 - \kappa$, or else one will end up with a trivial decision tree with a single decision node.

While $s_l$ as the metric seems to be an obvious choice, it often results in a problem that we refer to as *empty splits*. Precisely, the problem of empty splits occurs when one of the

sub-samples, i.e., either $\mathcal{S}_1$ or $\mathcal{S}_2$ becomes empty. Empty splits lead to an unbounded recursion branch of the learning algorithm since using the best $\text{LTL}_\text{f}$ formula (with respect to $s_l$) does not produce any meaningful splits. This problem is more prominent in samples which are skewed, meaning, it consists of traces of mostly one label. For instance, consider a sample $\mathcal{S}$ with one positive traces $u$ and 99 negative traces $v_1, v_2, \cdots, v_{99}$; for this sample, if one searches for an $\text{LTL}_\text{f}$ formula with $\mu = 0.9$, *false* is a minimal formula. This formula, however, results in empty splits since $\mathcal{S}_1$ is empty.

To address this problem, we use a score that relies on $wl$ from Equation 4.2 with a weight function $\Omega_r$ defined as follows:

$$\Omega_r(u) = \begin{cases} \frac{0.5}{|P|} \text{ for } u \in P, \\ \frac{0.5}{|N|} \text{ for } u \in N \end{cases} \tag{4.20}$$

Intuitively, the above $\Omega_r$ function normalizes the weight provided to traces based on the number of traces with a certain label.

Our choice of score function based on the above $\Omega_r$ function is the following:

$$s_r(\mathcal{S}, \varphi) = max\{wl(\mathcal{S}, \varphi, \Omega_r), 1 - wl(\mathcal{S}, \varphi, \Omega_r)\}. \tag{4.21}$$

Using such a score, we also avoid having *asymmetric splits*. We say a split is asymmetric when the fraction of positive traces in $\mathcal{S}_1$ is greater than or equal to 0.5. Choosing the score to be $1 - wl(\mathcal{S}, \varphi, \Omega_r)$ always leads to asymmetric splits, since several positive traces need to end up in $\mathcal{S}_1$ to minimize $wl(\mathcal{S}, \varphi, \Omega_r)$.

Now, to find an $\text{LTL}_\text{f}$ formula based on $s_r$, we need to invoke Algorithm 5 twice with $\kappa = 1 - \mu$; once with the original sample $\mathcal{S} = (P, N)$ and once with the sample $\mathcal{S} = (N, P)$ with positives and negatives swapped. One then chooses the formula with a better split from the two invocations of Algorithm 5.

While any score function that avoids the problem of empty and asymmetric splits is sufficient for our learning algorithm, we have used $s_r$ as a score function in our experiments. We show that if we learn an $\text{LTL}_\text{f}$ formula $\varphi$ such that $s_r(\mathcal{S}, \varphi) > 0.5$, we never encounter empty splits using the following lemma.

**Lemma 4.** *Given a sample $\mathcal{S}$ and an $\text{LTL}_f$ formula $\varphi$, if $s_r(\mathcal{S}, \varphi) > 0.5$, there exists traces $u_1, u_2$ in $\mathcal{S}$ such that $V(u_1, \varphi) = 1$ and $V(u_2, \varphi) = 0$.*

*Proof.* Towards contradiction, let us assume without loss of generality that for all $u$ in $S$ and formula $\varphi$ with $s_r(S, \varphi) > 0.5$, we have $V(u, \varphi) = 1$. We can compute $wl(\mathcal{S}, \varphi, \Omega_r)$ as follows:

$$\begin{aligned} wl(\mathcal{S}, \varphi, \Omega_r) &= \sum_{u \in P} \Omega_r(u) \cdot (1 - V(\varphi, u)) + \sum_{u \in N} \Omega_r(u) \cdot V(\varphi, u) \\ &= \sum_{u \in P} \frac{0.5}{|P|} \cdot (1 - 1) + \sum_{u \in N} \frac{0.5}{|N|} \cdot 1 = 0.5 \end{aligned}$$

Now, $s_r(\mathcal{S}, \varphi) = max\{wl(\mathcal{S}, \varphi, \Omega_r), 1 - wl(\mathcal{S}, \varphi, \Omega_r)\} = 0.5$. This violates our assumption of $s_r(\mathcal{S}, \varphi) > 0.5$. □

### 4.3.3 Stopping Criterion

The stopping criterion is essential for the termination of the algorithm. Towards the definition of the stopping criterion, we rely on two quantities: $p_1(\mathcal{S}) = \frac{|P|}{\|S\|}$, $p_2(\mathcal{S}) = \frac{|N|}{\|S\|}$.

We now define the stopping criterion as follows:

$$stop(\mathcal{S}) = \begin{cases} true \text{ if } p_1(\mathcal{S}) \leq \kappa \text{ or } p_2(\mathcal{S}) \leq \kappa, \\ false \text{ otherwise.} \end{cases} \tag{4.22}$$

Intuitively, the stopping criterion ensures that the algorithm terminates when the fraction of positive traces or fraction of negative traces in a resulting sample is less or equal to $\kappa$. When the stopping criterion holds, the algorithm halts and returns a leaf node labeled by $leaf(\mathcal{S})$ where $leaf$ is defined as follows:

$$leaf(\mathcal{S}) = \begin{cases} false \text{ if } p_1(\mathcal{S}) \leq \kappa, \\ true \text{ if } p_2(\mathcal{S}) \leq \kappa. \end{cases} \tag{4.23}$$

The following theorem ensures the correctness and termination of Algorithm 6.

**Theorem 5.** *Given a sample $\mathcal{S}$ and a threshold $\kappa \in [0, 1]$, Algorithm 6 terminates and returns a decision tree over LTL$_f$ formula $T$ such that $l(\mathcal{S}, T) \leq \kappa$.*

*Proof.* For termination, first observe that at each decision node, we can always infer an LTL$_f$ formula $\varphi$ for which $s_r(S, \varphi) \geq \mu$, for any value of $\mu$. This is because there always exists an LTL$_f$ formula $\varphi$ that produces perfect classification, and for this, $s_r(S, \varphi) = 1$. Second, observe that whenever a split is made during the learning algorithm, sub-samples $S_1$ and $S_2$ are both non-empty due to Lemma 4. This implies that the algorithm terminates since a sample can be only split finitely many times.

To ensure the decision tree $T$ achieves a $l(S, T) \leq \kappa$, we use induction over the structure of the decision tree. If $T$ is leaf node *true* or *false*, then $l(S, T) \leq \kappa$ using the stopping criteria. Now, say that $T$ is a decision tree with root $\varphi$ and subtrees $T_1$ and $T_2$, meaning $\varphi_T = (\varphi \wedge \varphi_{T_1}) \vee (\neg\varphi \wedge \varphi_{T_2})$. Also, say that the sub-samples produced by $\varphi$ are $S_1$ and $S_2$. By induction hypothesis, we can say that $l(S_1, T_1) \leq \kappa$ and $l(S_2, T_2) \leq \kappa$. Now, it is easy to observe that $l(S_1, (\varphi \wedge \varphi_{T_1})) \leq \kappa$ and $l(S_2, (\neg\varphi \wedge \varphi_{T_2})) \leq \kappa$, since $\varphi$ satisfies all traces in $S_1$ and $\neg\varphi$ does not satisfy any trace in $S_2$. We, thus, have $l(S, T) = l(S_1 \uplus S_2, (\varphi \wedge \varphi_{T_1}) \vee (\neg\varphi \wedge \varphi_{T_2})) \leq \kappa$. □

## 4.4 Experimental Evaluation

In this section, we present the implementation and the experimental results of this work. We split the section into results for learning LTL$_f$ in Section 4.4.1 and for learning STL in Section 4.4.2.

TABLE 4.1: $\text{LTL}_\text{f}$ patterns used for generation of samples

| Absence | Existence | Universality |
|---|---|---|
| $\mathsf{G}(\neg p_0)$ | $\mathsf{F}(p_0)$ | $\mathsf{G}(p_0)$ |
| $\mathsf{F}(p_1) \rightarrow (\neg p_0 \,\mathsf{U}\, p_1)$ | $\mathsf{G}(\neg p_0) \vee \mathsf{F}(p_0 \wedge \mathsf{F}(p_1))$ | $\mathsf{F}(p_1) \rightarrow (p_0 \,\mathsf{U}\, p_1)$ |
| $\mathsf{G}(p_1 \rightarrow \mathsf{G}(\neg p_0))$ | $\mathsf{G}(p_0 \wedge (\neg p_1 \rightarrow (\neg p_1 \,\mathsf{U}\, (p_2 \wedge \neg p_1))))$ | $\mathsf{G}(p_1 \rightarrow \mathsf{G}(p_0))$ |
| Disjunction of common patterns | | |
| $\mathsf{F}(p_0) \vee \mathsf{F}(p_1) \vee \mathsf{F}(p_2))$ | | |
| $\mathsf{G}(\neg p_0) \vee \mathsf{F}(p_0 \wedge \mathsf{F}(p_1)) \vee \mathsf{G}(\neg p_3) \vee \mathsf{F}(p_2 \wedge (\mathsf{F}\, p_3))$ | | |
| $\mathsf{G}(p_0 \wedge (\neg p_1 \rightarrow (\neg p_1 \,\mathsf{U}(p_2 \wedge (\neg p_1))))) \vee \mathsf{G}(p_3 \wedge (\neg p_4 \rightarrow (\neg p_4 \,\mathsf{U}(p_5 \wedge (\neg p_4)))))$ | | |

### 4.4.1 RQ1: Performance Gain in LTL learning

In this section, we evaluate the performance of our proposed algorithms for $\text{LTL}_\text{f}$ and compare them to the SAT-based learning algorithms by Neider and Gavran [169]. Specifically, we compare the following four algorithms:

1. *SAT-flie*: the SAT-based learning algorithms introduced by Neider and Gavran (Algorithm 1 from [169]),
2. *MaxSAT-flie*: our MaxSAT-based algorithm (Algorithm 5),
3. *SAT-DT*: the decision tree-based learning algorithm introduced by Neider and Gavran (Algorithm 2 from [169])[2] and
4. *MaxSAT-DT*: our decision tree learning algorithm (Algorithm 6).

We implement our learning algorithms in a Python tool[3] using Microsoft Z3 [164].

For the comparisons, we rely on synthetic data: we generate samples based on common $\text{LTL}_\text{f}$ patterns that can be found in practice [76]; Table 4.1 lists the set of the $\text{LTL}_\text{f}$ formulas used. For our first benchmark set, we generate 148 samples with the generation method proposed by Neider and Gavran [169]. The size of the generated samples ranges between 12 and 1000, consisting of traces of lengths up to 15. For our second benchmark set, we modify the first benchmark set: we introduce 5% noise in each sample by randomly inverting the labels of up to 5% of the traces. We call the first benchmark set as the benchmark without noise, while the second one as the benchmark with 5% noise.

We evaluate the performance of all the algorithms on both benchmark sets with a timeout of 900 seconds for each run. All experiments ran on a Debian machine with Intel Xeon E7-8857 CPU at 3GHz using up to 6GB of RAM.

Table 4.2 presents the summary of all the experimental results. Notice that we experiment with different values of the parameters $\kappa$ and $\mu$ to study their impact on the algorithms. We now discuss the results in detail.

We first compare *MaxSAT-flie* (proposed in this chapter) and *SAT-flie* (proposed in [169]). Figure 4.3 presents the detailed comparison of runtimes of *SAT-flie* and *MaxSAT-flie* for

[2] We adapted *SAT-DT* to learn decision trees with a similar stopping criteria as ours.

[3] https://github.com/cryhot/samples2LTL

TABLE 4.2: Summary of all the tested algorithms – comparison of numbers of timeouts, runtimes in seconds, learned formula sizes

| | Benchmark without noise | | | Benchmark with 5% noise | | |
|---|---|---|---|---|---|---|
| Algorithm | Timeouts | Avg. time | Avg. size | Timeouts | Avg. time | Avg. size |
| *SAT-flie* | 36/148 | 293.31 | 3.76 | 124/148 | 780.51 | 5.96 |
| *MaxSAT-flie*($\kappa = 0.001$) | 47/148 | 357.26 | 3.47 | 130/148 | 801.03 | 4.89 |
| *MaxSAT-flie*($\kappa = 0.05$) | 27/148 | 218.46 | 2.86 | 87/148 | 548.65 | 2.95 |
| *MaxSAT-flie*($\kappa = 0.1$) | 26/148 | 211.81 | 2.59 | 40/148 | 275.97 | 2.54 |
| *SAT-DT*($\kappa = 0.05$) | 51/148 | 342.35 | 5.92 | 127/148 | 786.16 | 9.62 |
| *MaxSAT-DT*($\kappa = 0.05, \mu = 0.8$) | 23/148 | 174.58 | 6.77 | 85/148 | 543.50 | 7.05 |
| *MaxSAT-DT*($\kappa = 0.05, \mu = 0.6$) | 7/148 | 74.97 | 30.91 | 38/148 | 281.60 | 56.55 |

different values of $\kappa$. In the figure, along with the comparison of the absolute runtime in seconds, we present the ratio of the runtime of *MaxSAT-flie* over *SAT-flie*.

We observe that, with $\kappa = 0.001$, *MaxSAT-flie* performs worse than *SAT-flie*. This is due to the fact that a MaxSAT problem is computationally more difficult to solve than a SAT problem [111]. For exact learning of an $\text{LTL}_\text{f}$ formula, the SAT problem suffices, and thus, *SAT-flie* performs better than *MaxSAT-flie*. For greater values of $\kappa$, *MaxSAT-flie* performs better than *SAT-flie*, especially on the benchmark with noise.

Next, we compared the size of the $\text{LTL}_\text{f}$ formula learned by *MaxSAT-flie* and *SAT-flie*. Figure 4.3b presents the comparison of size for $\kappa = 0.10$. By design, the size of the formula by *MaxSAT-flie* is less than or equal to that of by *SAT-flie*, as confirmed by the figure.

Finally, we noticed that when the size of the formula learned by *MaxSAT-flie* is smaller than *SAT-flie*, there was runtime gain. This typically happened when we considered the benchmark with noise. However, if the size of the formulas is similar, as it often happened in the benchmark without noise, the runtime of *MaxSAT-flie* and *SAT-flie* were similar; hence, the median ratio of the runtime was often equal to 1, as shown in Figure 4.3b.

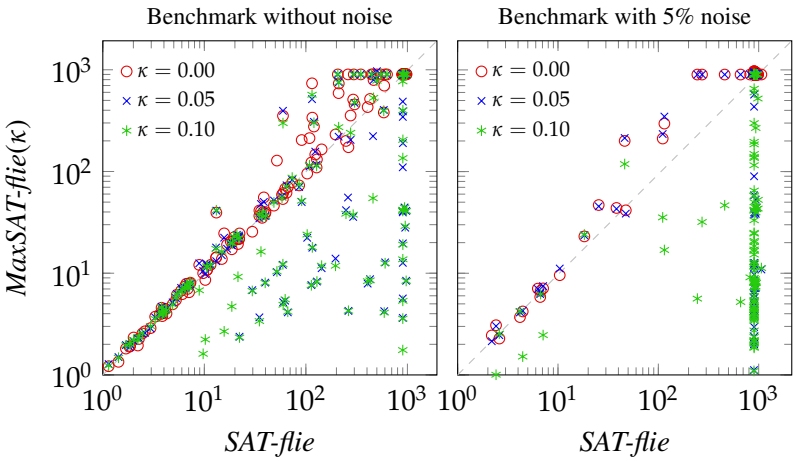
We now compare the two algorithms proposed in this chapter: did *MaxSAT-DT* perform any better than *MaxSAT-flie*? To be able to compare the size of decision trees to $\text{LTL}_\text{f}$ formulas, we measure the size of a tree $t$ in terms of the formula size $\varphi_t$ that this tree encodes.

Figure 4.5 presents a comparison of the runtime ratio as well as the formula size ratio of these two algorithms, on both benchmark sets. We observe that the runtime is generally lower for *MaxSAT-DT* than for *MaxSAT-flie*. However, *MaxSAT-DT* tends to learn formulas larger than that by *MaxSAT-flie*. This tradeoff between runtime and formula size is more pronounced for lower values of $\mu$.

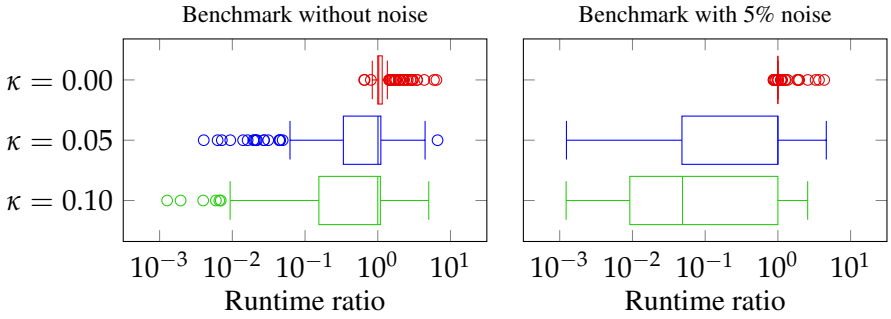
Regarding *SAT-DT* (proposed in [169]), we observe a large number of timeouts, especially when evaluated on the benchmark with noise.

### 4.4.2 RQ2: Performance Comparison for STL learning

In this section, we evaluate the performance of our proposed algorithms when adapted to learning STL formulas: we compare the performance of *MaxSAT-DT* and *MaxSAT-flie* for learning STL. We implement both learning algorithms in a C++ tool using Microsoft Z3 [164].




(a) Comparison of runtime in seconds




(b) Comparison of the ratio of the runtime of *MaxSAT-flie*($\kappa$) over the runtime of *SAT-flie*

FIGURE 4.3: Comparison of (absolute and ratio of) runtimes of *SAT-flie* and *MaxSAT-flie* on all benchmark sets

In this case, our benchmark set consists of samples with traces generated by policies learned from reinforcement learning (RL) using *model-based reinforcement learning* (MBRL) algorithm [166]. These traces describe a Pusher-robot that interacts with a ball and a wall. The system consists of seven features in total: two Boolean features with corresponding predicates of the form $u_j = \theta$ for $j \in \{1,2\}$ (for example, $u_1 = 1$ when the ball is in contact with the robot) and five continuous features with corresponding predicates of the form $u_j > \theta$ for $j \in \{3,\cdots,7\}$ (for example, $u_4$ represents the total upper arm movement of the Pusher-robot). We note that this system is hybrid, but we simply consider Boolean features as continuous features.

The benchmark has a total of four samples, each of them corresponding to an identified strategy of the Pusher-robot, that we like to describe using an STL formula. Each sample contains 300 traces: 150 positive traces from the current strategy, and 150 negative traces from the other three strategies. We set a timeout of 900*s* on each run.

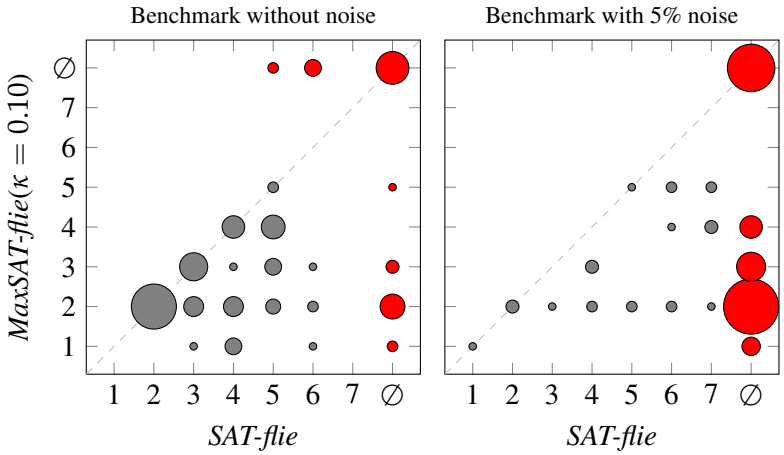



FIGURE 4.4: Learned $\mathrm{LTL_f}$ formula size comparison of *SAT-flie* and *MaxSAT-flie* with threshold $\kappa = 0.10$ on both benchmark sets. The surface of a bubble is proportional to the number of samples it represents. The timed out instances are represented by $\emptyset$.

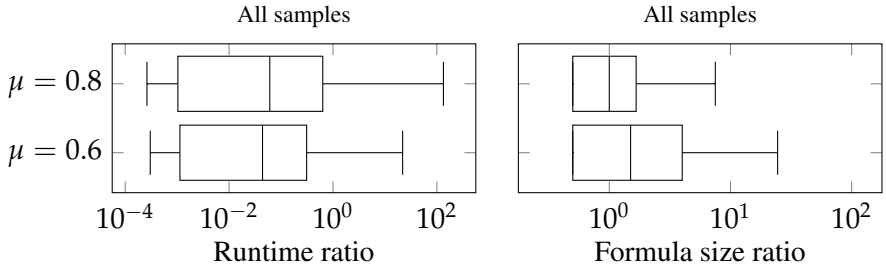



FIGURE 4.5: Comparison of the ratio of the performances of *MaxSAT-DT*($\mu$) over that of *MaxSAT-flie*, with $\kappa = 0.05$ for both algorithms where they did not time out.

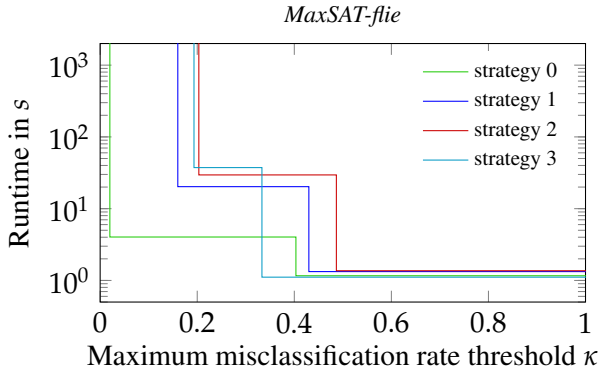



FIGURE 4.6: Impact of the threshold $\kappa$ on the runtime of *MaxSAT-flie*, represented as a step function, for each strategy. Each step corresponds to a certain number of iterations in Algorithm 5, that is, to a learned STL formula of a certain size, with a misclassification rate lower than or equal to $\kappa$.

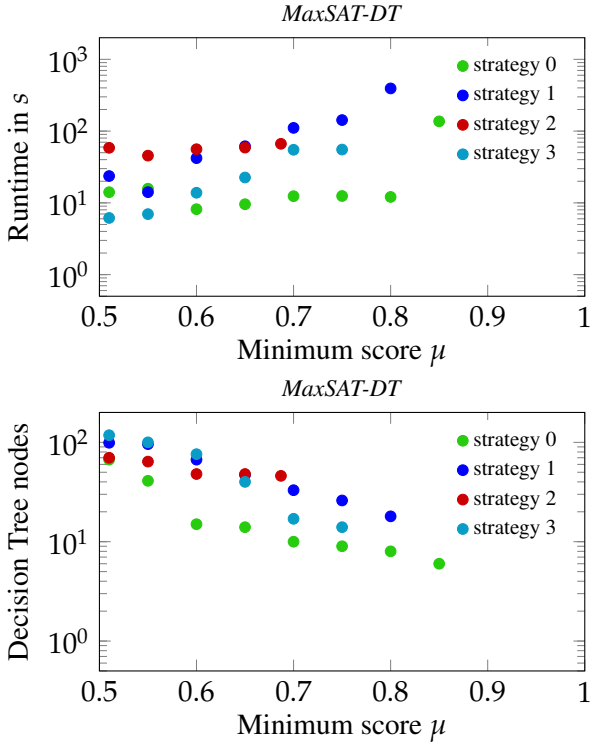



FIGURE 4.7: Impact of the minimum score hyper-parameter $\mu$ on the runtime and the number of Decision-Tree nodes of *MaxSAT-DT*($\kappa = 0$), for each strategy. Each strategy timed out for $\mu$ greater than or equal to 0.9, 0.85, 0.7 and 0.8 respectively.

Figure 4.6 shows the runtime of *MaxSAT-flie* for different numbers of iteration in Algorithm 5, presented by misclassification rate. For example, on the strategy 3 sample, we could learn the formula $\mathsf{F}_{[1,3)}\, s_0 = 0$ of size 2 with a misclassification rate of 19.33% (any $\kappa \in [0.1933, 0.3333)$ would have the same effect), with a runtime of 37 seconds. On the same sample, with some additional time beyond our timeout (2200 seconds instead of 900 seconds), we could learn an interesting STL formula $(s_5 > 0.003)\, \mathsf{U}_{[1,3)} (s_0 = 0)$ of size 3 with a misclassification rate of 15.67%.

We run *MaxSAT-DT* on each of the four samples, with results depicted in Figure 4.7. *MaxSAT-DT* could produce STL formulas perfectly classifying each sample, i.e., with $\kappa = 0$, where *MaxSAT-flie* timed out for the same $\kappa$. Increasing the hyper-parameter $\mu$ produces better quality splits of the sample: this way, the number of nodes in the decision tree is reduced, but the runtime is increased. We observe that the runtime of *MaxSAT-DT* increases in the shape of a step function shape when $\mu$ increases. This is similar to increase in the runtime of *MaxSAT-flie* with the decrease in $\kappa$ decreases: for example, the strategy 2 sample times out abruptly with $\mu > 0.67$ because one of the decision tree nodes now requires an STL formula of larger size in order to satisfy the criteria.

## 4.5 Conclusion

We developed two novel algorithms for learning $\text{LTL}_\text{f}$/STL formulas from a set of labeled traces/signals, allowing misclassifications. Moreover, we demonstrated that our algorithms are efficient in learning formulas, especially from noisy data, and can be used to interpret AI-generated data. As a part of future work, we would like to apply our MaxSAT-based approach for learning models in other formalisms such as PSL and MTL, and perform an extensive evaluation of the algorithms.

# Chapter 5

# Incorporating Intuition as Specification sketches

In this chapter, we study the problem of learning LTL formulas by incorporating expert knowledge of formal methods practitioners. As we discussed in Chapter 1, verification through formal methods has had several success stories in numerous domains such as in communication systems [82, 152], railway transportations [12, 13], aerospace [95, 60], operating systems [215, 136], etc.

There is, however, an essential and often overlooked catch with formal verification. Virtually all verification techniques assume that the specification required for the verification process is available in a suitable format, is functionally correct, and expresses precisely the properties the engineers had in mind. Formalizing system requirements is notoriously difficult and error-prone [198, 176, 197, 41]. Even worse, the training effort required to reach proficiency with specification languages can be disproportionate to the expected benefits [63], and the use of, for instance, temporal logics require a level of sophistication that many users might never develop [119, 107].

To aid the process of formalizing specifications, in this chapter, we introduce an approach—*specification sketching*—for writing formal specifications based on knowledge of engineers/practitioners about the underlying system. Our new paradigm is inspired by recent advances in automated program synthesis [204, 205]. It allows engineers to express their high-level insights about a system in terms of a partial specification, named *specification sketch*, where parts that are difficult or error-prone to formalize can be left out. To single out their desired specification, we rely on a sample of system execution, partitioned into positive and negative examples. Based on this sample, a so-called *sketching algorithm* fills in the missing low-level details to obtain a complete specification.

To demonstrate how our paradigm works, let us consider a simple scenario. Imagine that an engineer wishes to formalize the following request-response property $P$: every request $p$ has to be answered eventually by a response $q$. This property can be expressed in LTL as $\mathsf{G}(p \rightarrow \mathsf{X}\,\mathsf{F}\,q)$ using usual temporal operators $\mathsf{F}$, $\mathsf{G}$, and $\mathsf{X}$. However, for the sake of this example, assume that the engineer is unsure of how exactly to formalize $P$. In such a situation, our sketching paradigm allows them to express their high-level insights in the form of a sketch, say $\mathsf{G}(p \rightarrow ?)$, where the question mark indicates the missing part of the specification. Additionally, they can provide a sample of example infinite executions of the

system: (i) a positive trace $\{p\}\{q\}\{p\}\{q\}\{p\}\{q\}\cdots$, in which every request is answered by a response in the next timepoint, and (ii) a negative trace $\{p\}\{q\}\{p\}\{p\}\{p\}\cdots$, in which there are infinitely many requests that are not answered by a response. Our sketching algorithm then computes a substitution for the question mark such that the completed LTL formula is consistent with the sample (e.g., $? \coloneqq \mathsf{X}\,\mathsf{F}\,q$). In this example, the engineer left out an entire temporal formula in the sketch. However, our paradigm also allows one to leave out Boolean and temporal operators. For instance, one could also provide $?(p \rightarrow \mathsf{X}\,\mathsf{F}\,p)$ as a sketch, where the question mark now indicates a missing unary operator ($\mathsf{G}$ in our example).

While the concept of specification sketching can be conceived for a wide range of specification languages, in this chapter, we mainly focus on Linear Temporal Logic (LTL) [179]. LTL is the de facto temporal logic in formal methods, being popular both in academia and in industry [120, 213, 86, 95]. Moreover, LTL is well-understood and enjoys good algorithmic properties [59, 179]. Note that we rely on the variant of LTL for infinite traces since it is more prevalent in the verification community.

The problem of specification sketching for LTL, or LTL sketching in short, is the following: given a sample $\mathcal{S}$ partitioned into a set of positive and a set of negative examples and an LTL sketch $\varphi^?$, complete $\varphi^?$ to obtain a concise LTL formula $\varphi$ that is consistent with $\mathcal{S}$. In contrast to the standard (passive) LTL learning, the LTL sketching problem may not always have a solution: there are sketches for which there are no substitutions that make them consistent with the sample. We expand on why this is the case and other details of the LTL sketching problem in Section 5.1.

Next, we show that, while a solution may not exist for the LTL sketching problem, the problem of checking whether a solution exists is in NP. Moreover, we develop an effective decision procedure that reduces the original question to a satisfiability problem in propositional logic. This reduction permits us to apply highly-optimized, off-the-shelf SAT solvers to check whether a consistent substitution exists. We also describe a procedure to fix a sketch in case no substitution exists. We refer to details of the complexity result and the corresponding procedures related to existence of solution in Section 5.2.

We then develop two sketching algorithms for LTL. Following Occam's razor principle, both algorithms are biased towards finding "small" (concise) substitutions for the question marks in a sketch. The rationale behind this choice is that small formulas are arguably easier for engineers to understand and, thus, can be safely deployed in practice.

By exploiting the decision procedure of Section 5.2 as a sub-routine, our first algorithm transforms the sketching problem into several passive LTL learning tasks. This transformation allows us to apply a diverse array of algorithms for LTL passive learning, which have been proposed during the last ten years [169, 48, 193]. In addition, our algorithm immediately benefits from any advances in this field of research.

While the first algorithm builds on top of existing work and, hence, is easy to use, we observed that it tends to produce non-optimal substitutions for the unspecified parts of a sketch. Our second algorithm tackles this by searching for substitutions of increasing size using a SAT-based approach that is inspired by Neider et al. [169]. We formally prove that

this algorithm can, in fact, produce small substitutions (if they exist). We expand more on the algorithms in Section 5.3.

Finally, we present an experimental evaluation of our algorithms using a prototype implementation `LTL-Sketcher`. We demonstrate that our algorithms are effective in completing sketches with different types of missing information. Further, we compare `LTL-Sketcher` against two state-of-the-art specification mining tools for LTL. From the comparison, we demonstrate that `LTL-Sketcher`'s ability to complete missing temporal formulas and temporal operators enables it to complete more specifications. Moreover, we observe that providing high-level insights as a sketch reduces the number of examples required to derive the correct specification. We present all of the experimental results in Section 5.4. We conclude in Section 5.5 with a discussion on future work.

### Related Work.

Specification sketching can be seen as a form of specification mining [5]. In this area, the general idea of allowing partial specifications is not entirely new, but it has not yet been investigated as generally as in this work. For instance, a closely related setting is the one in which so-called templates are used to mine temporal specifications from system executions. In this context, a template is a partial formula similar to a sketch. Unlike a sketch, however, a template is typically completed with a single atomic proposition or a simple, usually Boolean formula (e.g., a restricted Boolean combination of atomic propositions). A prime example of this approach is Texada [144, 146], a specification miner for $\text{LTL}_\text{f}$ formula. Texada takes a template (property type in their terminology) and a set of system executions as input and completes the template with atomic propositions such that the resulting LTL formula satisfies all system executions. In contrast to Texada, our paradigm assists engineers in completing more complex temporal formulas in their specifications, thus alleviating an even larger burden off an engineer. Another example in this setting is the concept of temporal logic queries, introduced by Chan [52] for CTL, and later developed by Bruns and Godefroid [43] for a wide range of temporal logics. However, unlike our paradigm, temporal logic queries allow only a single placeholder in their template that can be filled with only atomic propositions.

Various other techniques operate in settings where the templates are even more restricted. For example, Li et al. [147] mine LTL specification based on templates from the GR(1)-fragment of LTL (e.g., $\mathsf{G}\,\mathsf{F}?$, $\mathsf{G}(?_1 \rightarrow \mathsf{X}?_2)$, etc.), while Shah et al. [200] mine LTL formulas that are conjunctions of the set of common temporal properties identified by Dwyer et al. [77]. In addition, Kim et al. [135] consider a set of interpretable LTL templates, widely used in the development of software systems, to obtain LTL formulas robust to noise in the input data. In the context of CTL, on the other hand, Wasylkowski et al. [219] mine specifications using templates of the form $\mathsf{A}\,\mathsf{F}?$, $\mathsf{A}\,\mathsf{G}(?_1 \rightarrow \mathsf{F}?_2)$, etc. However, all the approaches above complete the templates only with atomic propositions (and their negations in some cases).

Another setting is where general (and complex) temporal specifications are learned from system executions without any information about the structure of the specification. This is what we often describe in this thesis as the passive learning problem for LTL. The most notable work in this setting is by Neider et al. [169], who learn LTL formulas using a SAT

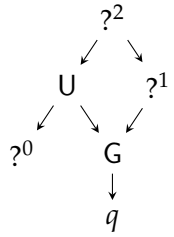



FIGURE 5.1: An LTL sketch

solver and by Camacho et al. [48], who propose SAT-based learning for $\text{LTL}_\text{f}$ formulas via alternating finite automata representations. Also, we present an approach for learning formulas in fragments of $\text{LTL}_\text{f}$ without the U-operator in Chapter 3. We present many other passive learning approaches (for, for instance, PSL in Chapter 7 and MTL in Chapter 8) in this thesis that fall into this setting. However, all of these works are "unguided" in that none of them exploit insights about the structure of the specification to aid the learning/mining process.

Finally, it is worth mentioning that LTL sketching can also be seen as a particular case of syntax-guided synthesis (SyGuS), where syntactic constraints on the resulting formulas are expressed in terms of a context-free grammar. An example of a syntax-guided approach is SySLite [8], a CVC4-based tool for learning Past-time LTL over finite executions. However, to the best of our knowledge, we are unaware of any SyGuS engine that can infer specifications in LTL over infinite (that is, ultimately-periodic) system executions.

## 5.1 Problem Formulation

Since the problem of *LTL sketching* relies heavily on LTL sketches, we begin with formalizing them first.

### 5.1.1 LTL Sketch.

An *LTL sketch* is an incomplete LTL formula in which parts that are difficult to formalize can be left out. The left-out parts are represented using placeholders, denoted by ?'s. An example of an LTL sketch can be seen in Figure 5.1. We comment on the superscripts on the placeholders in the figure shortly.

Formally, an LTL sketch $\varphi^?$ is simply an LTL formula whose syntax is augmented with placeholders. The placeholders we allow can be of three types: placeholders of arity zero referred to as Type-0 placeholders, that replace missing LTL formulas; placeholders of arity one referred to as Type-1 placeholders, that replace missing unary operators; and placeholders of arity two referred to as Type-2 placeholders, that replace missing binary operators. In Figure 5.1 (and throughout the paper), Type-$i$ placeholders are represented using $?^i$.

Given (possibly empty) sets $\Pi^0$, $\Pi^1$ and $\Pi^2$ consisting of Type-0, Type-1 and Type-2 placeholders, respectively, we define LTL sketches inductively as follows:

- each element of $\mathcal{P} \cup \Pi^0$ is an LTL sketch; and

- if $\varphi_1^?$ and $\varphi_2^?$ are LTL sketches, $\circ\,\varphi_1^?$ is an LTL sketch for $\circ \in \Lambda_U \cup \Pi^1$ and so is $\varphi_1^? \circ \varphi_2^?$ for $\circ \in \Lambda_B \cup \Pi^2$.

Note that an LTL sketch in which $\Pi^0 = \Pi^1 = \Pi^2 = \emptyset$ is simply an LTL formula. Further, let $\Pi_{\varphi^?} = \Pi^0 \cup \Pi^1 \cup \Pi^2$ denote the set of all placeholders in a sketch $\varphi^?$. For the sketch in Figure 5.1, $\Pi_{\varphi^?} = \{?^0, ?^1, ?^2\}$. For brevity, in the rest of the paper, we refer to an LTL sketch as a sketch.

The placeholders are abstract symbols that apriori do not have any meaning. To assign meaning to a sketch, we need to substitute all Type-0 placeholders with LTL formulas, all Type-1 placeholders with unary operators, and all Type-2 placeholders with binary operators. We do this using a so-called substitution function (or substitution for short).

Formally, a *substitution* function $s$ maps placeholders and operators present in a sketch to LTL operators and LTL formulas in such a way that:

$$\begin{aligned} s(?) \in \mathcal{F}_{\text{LTL}} \text{ if } ? \in \Pi^0, \\ s(?) \in \Lambda_U \text{ if } ? \in \Pi^1, \\ s(?) \in \Lambda_B \text{ if } ? \in \Pi^2, \\ s(\lambda) = \lambda \text{ if } \lambda \in \Lambda. \end{aligned}$$

Recall that $\mathcal{F}_{\text{LTL}}$ denotes the set of all LTL formulas.

Moreover, a substitution $s$ is said to be *complete* for a sketch $\varphi^?$ if $s$ is defined for every element in $\Lambda \cup \Pi_{\varphi^?}$ in $\varphi^?$. For example, a possible complete substitution $s$ for the sketch $\varphi^?$ in Figure 5.1 can be $s(?^0) = p$, $s(?^1) = \mathsf{F}$, $s(?^2) = \vee$, and $s(\lambda) = \lambda$ for $\lambda \in \Lambda$.

A complete substitution $s$ can be applied to a sketch $\varphi^?$ to obtain an LTL formula. To make this precise, we define a function $f_s$, which is defined recursively on the structure of $\varphi^?$ as follows:

$$\begin{aligned} f_s(\varphi_1^? \,?^2\, \varphi_2^?) &= f_s(\varphi_1^?) \circ f_s(\varphi_2^?), \text{ where } \circ = s(?^2), \\ f_s(?^1 \varphi^?) &= \circ\, f_s(\varphi^?), \text{ where } \circ = s(?^1), \\ f_s(?^0) &= s(?^0), \\ f_s(\varphi^?) &= \varphi^? \text{ if } \Pi_{\varphi^?} = \emptyset. \end{aligned}$$

For the complete substitution $s$ for $\varphi^?$ defined in the last paragraph we get $f_s(\varphi^?) = (p \,\mathsf{U}\, \mathsf{G}\, q) \vee (\mathsf{F}(\mathsf{G}\, q))$.

### 5.1.2 The Sketching Problem

While there can be many ways to complete a sketch, we direct our search based on a sample $\mathcal{S} = (P, N)$ of infinite traces from $(2^{\mathcal{P}})^\omega$, partitioned into a set $P$ of positive examples and a set $N$ of negative examples, such that $P \cap N = \emptyset$. As infinite traces, we specifically consider ultimately periodic traces, that is, traces of the form $uv^\omega$ where $u \in (2^{\mathcal{P}})^*$ and $v \in (2^{\mathcal{P}})^+$. Ultimately period traces are known to be sufficient in order to uniquely characterize $\omega$-regular

languages [45] (and thus, LTL formulas). In this case, we define size of a sample to be $|\mathcal{S}| = \sum_{uv^\omega \in P \cup N} |uv|$.

Similar to previous chapters, we say that an LTL formula $\varphi$ is *consistent* with a sample $\mathcal{S} = (P, N)$ if $uv^\omega \models \varphi$ for all positive examples $uv^\omega \in P$ and $uv^\omega \not\models \varphi$ for all negative examples $uv^\omega \in N$.

We now state the central problem of the paper.

**Problem 4** (*LTL sketching*)**.** *Given a sample $\mathcal{S} = (P, N)$ and an LTL sketch $\varphi^?$, find a complete substitution $s$ for $\varphi^?$ such that $f_s(\varphi^?)$ is consistent with $\mathcal{S}$.*

Unlike the passive *LTL learning* problem, a solution to the *LTL sketching* problem does not always exist. This can be illustrated using the following simple example. Consider the sample $\mathcal{S}$ consisting of a single positive trace $\alpha = \{p\}\{q\}^\omega$ and a single negative trace $\beta = \{q\}^\omega$ and the sketch $\mathsf{G}(?^0)$. For this sample and sketch, there does not exist any substitution that leads to an LTL formula consistent with the sample. Towards contradiction, let us assume that there exists an LTL formula $\mathsf{G}(\varphi)$ that is consistent with $\mathcal{S}$, meaning, $\alpha \models \mathsf{G}(\varphi)$ and $\beta \not\models \mathsf{G}(\varphi)$. Based on the semantics of the G-operator, also $\alpha[1, \infty) \models \mathsf{G}(\varphi)$. On the other hand, since $\beta = \alpha[1, \infty)$, $\alpha[1, \infty) \not\models \mathsf{G}(\varphi)$.

Since, for a given sample and LTL sketch, there might not exist any complete substitution, a naive enumeration-like algorithm to search over all substitutions may not terminate. To show that one can indeed design a terminating sketching algorithm, in the next section, we prove the decidability of *LTL sketching*.

## 5.2 Existence of a Complete Sketch

To devise a terminating algorithm for the *LTL sketching* problem, we first introduce the related decision problem, which is the following:

**Problem 5** (*LTL sketch existence*)**.** *Given a sample $\mathcal{S} = (P, N)$ and an LTL sketch $\varphi^?$, does there exist a complete substitution $s$ for $\varphi^?$ such that $f_s(\varphi^?)$ is consistent with $\mathcal{S}$.*

In what follows, we prove that this problem is indeed decidable and belongs to the complexity class NP. Thereafter, we devise a decision procedure for the problem by exploiting the satisfiability (SAT) problem.

### 5.2.1 Decidability Result

For the decidability result, we begin by introducing some concepts as a preparation. Let us first observe the following key property of ultimately periodic traces.

**Observation 1.** *Let $uv^\omega \in (2^{\mathcal{P}})^\omega$ and $\varphi$ be an LTL formula. Then, $uv^\omega[|u| + t_1] = uv^\omega[|u| + t_2]$ for $t_2 \equiv t_1 \mod |v|$. Thus, $uv^\omega[|u| + t_1, \infty) \models \varphi$ if and only if $uv^\omega[|u| + t_2, \infty) \models \varphi$.*

This observation indicates that, for a trace $uv^\omega$, there exists only a finite number of distinct suffixes of $uv^\omega$, all of which originate in the initial $uv$ portion of $uv^\omega$. Let us then define

| | 0 | 1 | 2 |
|---|---|---|---|
| $p$ | 1 | 1 | 0 |
| $q$ | 1 | 0 | 1 |
| $\mathsf{X}q$ | 0 | 1 | 1 |
| $p \vee \mathsf{X}q$ | 1 | 1 | 1 |

FIGURE 5.2: Table $T^{\psi}_{\alpha}$ for $\psi = p \vee \mathsf{X}q$ and $\alpha = \{p,q\}\{p\}\{q\}^{\omega}$

$suf(uv^{\omega}) = \{uv^{\omega}[t,\infty) \mid 0 \leq t < |uv|\}$ as the set of all (possibly) distinct suffixes of $uv^{\omega}$. Moreover, let

$$suf(\mathcal{S}) = \bigcup_{uv^{\omega} \in (P \cup N)} suf(uv^{\omega})$$

be the set of suffixes of all traces in $\mathcal{S}$.

Observation 1 also indicates that, to determine the evaluation of an LTL formula $\varphi$ on an ultimately periodic trace $uv^{\omega}$, it is sufficient to determine its evaluation on the initial $|uv|$ suffixes of $uv^{\omega}$.

Thus, for a compact representation of the evaluation of $\varphi$ on $uv^{\omega}$, we introduce a table notation $T^{\varphi}_{uv^{\omega}}$. Mathematically speaking, a table $T^{\varphi}_{uv^{\omega}}$ is a $|\varphi| \times |uv|$ matrix that consists of the satisfaction of all the subformulas $\varphi'$ of $\varphi$ on the suffixes of $uv^{\omega}$. We define the entries of this matrix as:

$$T^{\varphi}_{uv^{\omega}}[\varphi', t] = \begin{cases} 1 \text{ if } uv^{\omega}[t,\infty) \models \varphi', \\ 0 \text{ if } uv^{\omega}[t,\infty) \not\models \varphi', \end{cases}$$

for all subformulas $\varphi'$ of $\varphi$ and $0 \leq t < |uv|$.

Based on the above definition of the table $T^{\varphi}_{uv^{\omega}}$, we identify three properties of these tables, which form the main building blocks of the decidability proof (that is, proof of Theorem 6), as we see later.

The first property, or as we call it, the *Semantic* property, is that various rows of the table are related to each other in a way that reflects the semantics of LTL. To explain this further, we use $T^{\varphi}_{uv^{\omega}}[\varphi', \cdot]$ to represent the row of $T^{\varphi}_{uv^{\omega}}$ corresponding to subformula $\varphi'$.

We first demonstrate the Semantic property on an example. Consider the formula $\psi = p \vee \mathsf{X}q$ and the trace $\alpha = \{p,q\}\{p\}\{q\}^{\omega}$. The table $T^{\psi}_{\alpha}$ is illustrated in Figure 5.2. From the figure, one can see that the row $T^{\psi}_{\alpha}[p \vee \mathsf{X}q, \cdot]$ corresponds to the bitwise-OR of the rows $T^{\psi}_{\alpha}[p, \cdot]$ and $T^{\psi}_{\alpha}[\mathsf{X}q, \cdot]$, reflecting the semantics of the $\vee$-operator that combines formulas $p$ and $\mathsf{X}q$.

To define these semantic relations between the rows, we must uniquely identify the subformula that corresponds to each row. As a result, we assign unique identifiers $i \in \{1, \ldots, n\}$ to each node of the syntax DAG of $\varphi$, and we denote the subformula rooted at Node $i$ using $\varphi[i]$. For assigning identifiers, we follow the same strategy as we did in the previous chapter, Section 4.1: the root node has identifier 1, and every node has an identifier smaller than its children (that is, if it has any). One can also analogously assign identifiers to

syntax DAGs of sketches. We additionally rely on the function $\ell\colon \{1,\ldots,n\} \mapsto \Lambda$ that maps the identifiers to the corresponding operators in the syntax DAG.

We now describe the set of equations that formalize the relation between the rows. How a row $T^{\varphi}_{uv^\omega}[\varphi[i],\cdot]$ relates to the others depends on the operator $\ell(i)$ in the root node of $\varphi[i]$. For instance, if $\ell(i) = p$ for some proposition $p$, then we have the following relation:

$$T^{\varphi}_{uv^\omega}[\varphi[i],t] = \begin{cases} 1 \text{ if } p \in uv^\omega[t] \\ 0 \text{ otherwise} \end{cases} \tag{5.1}$$

If, on the other hand, $\ell(i)$ is a unary-operator and Node $j$ is the left child of Node $i$, we have the following relations:

$$\text{if } \ell(i) = \neg:\ T^{\varphi}_{uv^\omega}[\varphi[i],t] = 1 - T^{\varphi}_{uv^\omega}[\varphi[j],t] \ \text{ for } 0 \le t < |uv| \tag{5.2}$$

$$\text{if } \ell(i) = \mathsf{X}:\ T^{\varphi}_{uv^\omega}[\varphi[i],t] = \begin{cases} T^{\varphi}_{uv^\omega}[\varphi[j],t+1] \ \text{ for } 0 \le t < |uv|-1 \\ T^{\varphi}_{uv^\omega}[\varphi[j],|u|] \ \text{ for } t = |uv|-1 \end{cases} \tag{5.3}$$

The first equation simply follows from the semantics of $\neg$-operator, while the second one follows the semantics of X-operator. To implement the semantics of X-operator in the periodic part of $uv^\omega$, the second equation exploits Observation 1 and determines the entry $T^{\varphi}_{uv^\omega}[\varphi[i],|uv|-1]$ using the evaluation of $\varphi[j]$ at $uv^\omega[|u|,\infty)$, that is, the start of the periodic part.

If $\ell(i)$ is a binary operator, and Node $j$ and Node $j'$ are the left and right children of Node $i$, respectively, then we have the following relations:

$$\text{if } \ell(i) = \vee : T^{\varphi}_{uv^\omega}[\varphi[i],t] = T^{\varphi}_{uv^\omega}[\varphi[j],t] \vee T^{\varphi}_{uv^\omega}[\varphi[j'],t] \text{ for } 0 \le t < |uv| \tag{5.4}$$

$$\text{if } \ell(i) = \mathsf{U}:\ T^{\varphi}_{uv^\omega}[\varphi[i],t] = \tag{5.5}$$

$$\begin{cases} \bigvee_{t \le t'' < |uv|} \left[ T^{\varphi}_{uv^\omega}[\varphi[j'],t''] \wedge \bigwedge_{t \le t' < t''} T^{\varphi}_{uv^\omega}[\varphi[j],t'] \right] \text{ for } 0 \le t < |u| \\ \bigvee_{|u| \le t'' < |uv|} \left[ T^{\varphi}_{uv^\omega}[\varphi[j'],t''] \wedge \bigwedge_{t' \in t \looparrowright_{u,v} t''} T^{\varphi}_{uv^\omega}[\varphi[j],t'] \right] \text{ for } |u| \le t < |uv| \end{cases}$$

The first equation above follows from the semantics of $\vee$-operator. The second equation follows from the semantics of U-operator and consists of two cases: the first case provides the relation for entries $t \in \{0,\cdots,|u|-1\}$ in the initial part u; the second case covers the entries $t \in \{|u|,\cdots,|uv|-1\}$ in the periodic part of $uv^\omega$. Thereby, the second case uses Observation 1 to "loop back" into the periodic part using the set $t \looparrowright_{u,v} t''$ defined as:

$$t \looparrowright_{u,v} t'' = \begin{cases} \{t,\cdots,t''-1\} \text{ if } t < t''; \\ \{|u|,\cdots,t''-1,t,\cdots,|uv|-1\} \text{ otherwise.} \end{cases}$$

Next, we describe the second property, the *Consistency* property. This property ensures that $T^{\varphi}_{uv^\omega}[\varphi,0] = 1$ if and only if $uv^\omega$ satisfies $\varphi$. Thus, for an LTL formula $\varphi$ consistent with

$\mathcal{S}$, we have the following relation:

$$T^{\varphi}_{uv^{\omega}}[\varphi, 0] = 1 \text{ for all } uv^{\omega} \in P, \text{ and } T^{\varphi}_{uv^{\omega}}[\varphi, 0] = 0 \text{ for all } uv^{\omega} \in N \tag{5.6}$$

The final property we observe is called the *Suffix* property. This property originates from the fact that LTL, being a future-time logic, has the same evaluation on equal suffixes, that is, for all $u_1v_1^{\omega}[t,\infty) = u_2v_2^{\omega}[t',\infty)$, $u_1v_1^{\omega}[t,\infty \models \varphi$ if and only if $u_2v_2^{\omega}[t',\infty \models \varphi$. Formally, we state the property as follows:

$$T^{\varphi}_{u_1v_1^{\omega}}[\varphi, t] = T^{\varphi}_{u_2v_2^{\omega}}[\varphi, t'] \text{ for all } u_1v_1^{\omega}[t,\infty) = u_2v_2^{\omega}[t',\infty) \tag{5.7}$$

This property becomes significant later, especially for constructing LTL formulas to substitute Type-0 placeholders.

With the prerequisites set up, we now proceed to describe an NP algorithm for deciding the *LTL sketch existence* problem. For an easy presentation of the algorithm, we consider the simple (but crucial) case where the only missing information in $\varphi^?$ is a single Type-0 placeholder. While one might assume that non-deterministically guessing a substitution for the placeholder should suffice; it does not. This is because, apriori, the size of the LTL formula required to substitute the Type-0 placeholder is not known.

Thus, in our NP algorithm, instead of guessing substitutions, we guess the entries of the table $T^{\varphi^?}_{uv^{\omega}}$ for each $uv^{\omega}$ in $\mathcal{S}$. Note that the tables have a finite dimension, precisely $|\varphi^?| \times |uv|$, for each trace $uv^{\omega}$. Thus, the overall process of simply guessing the table entries can be done in time $\mathcal{O}(poly(|\varphi^?|, |\mathcal{S}|))$.

After guessing the table entries, we must verify that the guessed tables satisfy the three properties, Semantic, Consistency, and Suffix, discussed earlier in this section. It is easy to verify that checking the first two properties for the tables requires time $\mathcal{O}(poly(|\varphi^?|, |uv|))$ (that is, polynomial in $|\varphi^?|$ and $|uv|$) for each $uv^{\omega}$ in $\mathcal{S}$. For checking the Suffix property, one must identify the equal suffixes in $suf(\mathcal{S})$. This can be also done in time $\mathcal{O}(poly(|\mathcal{S}|))$, simply by unrolling the periodic part of the suffixes to a fixed length. This is formalized in the following lemma, which, intuitively, states that two traces are equal if they are equal only on a finite portion $b$ of size $poly(|u_1|, |u_2|, |v_1|, |v_2|)$.

**Lemma 5.** *$u_1v_1^{\omega}[t,\infty) = u_2v_2^{\omega}[t',\infty)$ if and only if $u_1v_1^{\omega}[t, t+b) = u_2v_2^{\omega}[t', t'+b)$, where $b = \max(|u_1[t, |u_1|)|, |u_2[t', |u_2|)|) + lcm(|v_1|, |v_2|)$.*

*Proof.* For the forward direction, consider $u_1v_1^{\omega}[t,\infty) = u_2v_2^{\omega}[t',\infty)$. Clearly, all prefixes of $u_1v_1^{\omega}[t,\infty)$ and $u_2v_2^{\omega}[t',\infty)$ are equal, that is, $u_1v_1^{\omega}[t, t+b) = u_2v_2^{\omega}[t', t'+b)$ for all $b \in \mathbb{N}$.

For the other direction, we consider

$$u_1v_1^{\omega}[t, t+b) = u_2v_2^{\omega}[t', t'+b), \text{ for } b = \max(|u_1[t, |u_1|)|, |u_2[t', |u_2|)|) + lcm(|v_1|, |v_2|).$$

Also, without loss of generality, let us assume that $|u_1[t, |u_1|)| \geq |u_2[t', |u_2|)|$. To avoid clutter of notation, we denote $\mu = |u_1[t, |u_1|)|$ and $\nu = lcm(|v_1|, |v_2|)$. Thus, in this case, $b = \mu + \nu$.

The proof, now, is based on two main observations. First, we begin with the simple observation:

$$u_1 v_1^\omega[t, t+\mu) = u_2 v_2^\omega[t', t'+\mu); \text{ and}$$
$$u_1 v_1^\omega[t+\mu, t+b) = u_2 v_2^\omega[t'+\mu, t'+b)$$

Second, we have that

$$(u_1 v_1^\omega[t+\mu, t+b))^\omega = v_1^\omega; \text{ and}$$
$$(u_2 v_2^\omega[t'+\mu, t'+b))^\omega = (v_2^\omega[t'+\mu, t'+\mu+|v_2|))^\omega$$

The above observation is due to the fact that $u_1 v_1^\omega[t+\mu, t+b) = v_1^\kappa$ for $\kappa = \nu/|v_1|$ and $u_2 v_2^\omega[t'+\mu, t'+b) = (v_2^\omega[t'+\mu, t'+\mu+|v_2|))^\kappa$ for $\kappa = \nu/|v_2|$.

Now, combining the two observations, we have the following:

$$\begin{aligned} u_1 v_1^\omega &= u_1 v_1^\omega[t, t+\mu) \cdot (u_1 v_1^\omega[t+\mu, t+b))^\omega \\ &= u_2 v_2^\omega[t', t'+\mu) \cdot (u_2 v_2^\omega[t'+\mu, t'+b))^\omega \\ &= u_2[t', |u_2|) \cdot u_2 v_2^\omega[|u_2|, t'+\mu) \cdot (v_2^\omega[t'+\mu, t'+\mu+|v_2|))^\omega \\ &= u_2 v_2^\omega \end{aligned}$$

□

The NP algorithm based on guessing tables naturally extends to multiple Type-0 placeholder. The following lemma now asserts that if the guessed tables satisfy the three properties, then one can find a suitable complete LTL formula.

**Lemma 6.** *Let $\mathcal{S} = (P, N)$ be a sample and $\varphi^?$ be a sketch with only Type-0 placeholders. Then, the following holds: there exists tables $T^{\varphi^?}_{uv^\omega}$ (that is, $|\varphi^?| \times |uv|$ matrices with $\{0,1\}$ entries) for each $uv^\omega \in P \cup N$ that satisfy the Semantic, Consistency, and Suffix properties if and only if there exists a substitution s such that LTL formula $f_s(\varphi^?)$ is consistent with $\mathcal{S}$.*

*Proof.* For simplicity, we again consider that $\varphi^?$ consists of only one Type-0 placeholder $?^0$. The proof can be seamlessly extended to multiple Type-0 placeholders.

For the forward direction, we show the existence of the substitution $s$ by explicit construction of an LTL formula for $?^0$. Towards this, we first construct a sample $\mathcal{S}' = (P', N')$ as follows:

$$P' = \{uv^\omega[t, \infty) \in suf(\mathcal{S}) \mid T^{\varphi^?}_{uv^\omega}[?^0, t] = 1,\ uv^\omega \in P \cup N,\ 0 \leq t < |uv|\}$$
$$N' = \{uv^\omega[t, \infty) \in suf(\mathcal{S}) \mid T^{\varphi^?}_{uv^\omega}[?^0, t] = 0,\ uv^\omega \in P \cup N,\ 0 \leq t < |uv|\}.$$

Since the tables satisfy the Suffix property, we have that $P' \cap N' = \emptyset$. We can now construct the generic LTL formula $\psi$ consistent with $\mathcal{S}'$ using the *LTL learning* problem [169]. We claim that this formula $\psi$ can be substituted in $?^0$ to obtain a consistent LTL formula.

Towards this, we first prove that $T^{f_s(\varphi^?)}_{uv^\omega}[\psi,\cdot] = T^{\varphi^?}_{uv^\omega}[?,\cdot]$ for all $uv^\omega \in P \cup N$. To prove this, we exploit two simple observations. First, using the definition of tables, we have $T^{f_s(\varphi^?)}_{uv^\omega}[\psi,t] = 1$ if and only if $uv^\omega[t,\infty) \models \psi$ for each $uv^\omega[t,\infty) \in suf(\mathcal{S})$. Second, since $\psi$ is consistent with $\mathcal{S}'$, we know $T^{\varphi^?}_{uv^\omega}[?,t] = 1$ if and only if $uv^\omega[t,\infty)) \models \psi$. Together, we have $T^{f_s(\varphi^?)}_{uv^\omega}[\psi,t] = T^{\varphi^?}_{uv^\omega}[?,t]$.

Next, we prove that $T^{f_s(\varphi^?)}_{uv^\omega}[f_s(\varphi^?)[i],\cdot] = T^{\varphi^?}_{uv^\omega}[\varphi^?[i],\cdot]$ for each $0 \leq i < |\varphi^?|$ and trace $uv^\omega \in P \cup N$. (Note that we denote the same nodes in $f_s(\varphi^?)$ and $\varphi^?$ using the same identifiers.) Towards contradiction, we assume that there exists some $uv^\omega \in P \cup N$ and some $0 \leq i < |\varphi^?|$ and such that $T^{f_s(\varphi^?)}_{uv^\omega}[f_s(\varphi^?)[i],\cdot] \neq T^{\varphi^?}_{uv^\omega}[\varphi^?[i],\cdot]$. Let $i^*$ be the maximum row for which the tables become unequal. The proof, in general, will proceed via a case analysis on the operator $\ell(i^*)$ labeled in Node $i^*$. However, since for proof is similar for all the operators, we assume $\ell(i) = \neg$ and Node $j^*$ is the left child of Node $i^*$. Recall that $j^* > i^*$ based on our assignment of identifiers. Further, based on the Semantic property, $T^{f_s(\varphi^?)}_{uv^\omega}[f_s(\varphi^?)[i^*],t] = 1 - T^{f_s(\varphi^?)}_{uv^\omega}[f_s(\varphi^?)[j^*],t]$ and $T^{\varphi^?}_{uv^\omega}[\varphi^?[i^*],\cdot] = 1 - T^{\varphi^?}_{uv^\omega}[\varphi^?[j^*],\cdot]$ for each $0 \leq t \leq |uv|$ (Equation 5.2). This implies that $T^{f_s(\varphi^?)}_{uv^\omega}[f_s(\varphi^?)[j^*],\cdot] \neq T^{\varphi^?}_{uv^\omega}[\varphi^?[j^*],\cdot]$, contradicting the maximality of $i^*$.

Finally, observe that $T^{f_s(\varphi^?)}_{uv^\omega}[f_s(\varphi^?),\cdot] = T^{\varphi^?}_{uv^\omega}[\varphi^?,\cdot]$. As a consequence, since tables $T^{\varphi^?}_{uv^\omega}$ satisfy the Consistency property, so do tables $T^{f_s(\varphi^?)}_{uv^\omega}$. This implies that $f_s(\varphi^?)$ is consistent with $\mathcal{S}$.

For the other direction, we construct tables $T^{\varphi^?}_{uv^\omega}$ based on the tables $T^{f_s(\varphi^?)}_{uv^\omega}$. In particular, we have $T^{\varphi^?}_{uv^\omega}[\varphi^?[i],\cdot] = T^{f_s(\varphi^?)}_{uv^\omega}[f_s(\varphi^?)[i],\cdot]$ for each $0 \leq i < |\varphi^?|$ and $uv^\omega \in P \cup N$. Since tables $T^{f_s(\varphi^?)}_{uv^\omega}$ satisfy the Semantic, the Consistency and the Suffix properties, so does the tables $T^{\varphi^?}_{uv^\omega}$. □

With this, we conclude the NP algorithm for the case where $\varphi^?$ only has Type-0 placeholders. We can easily extend the algorithm to the case where $\varphi^?$ consists of Type-1 and Type-2 placeholders. In particular, we first guess the operators to be substituted for the Type-1 and Type-2 placeholders and substitute them. We then obtain a sketch consisting of only Type-0 placeholders. We now apply our algorithm that relies on guessing tables, as described above.

**Theorem 6.** *The LTL sketch existence problem is in* NP.

We make an important remark here: the above result also holds for $\text{LTL}_\text{f}$, given that the sample consists of finite traces. The proof goes through almost identically. The only difference is in the definition of the Semantic property of $T^{\varphi^?}_u$; it simply needs to be modified to the semantics of $\text{LTL}_\text{f}$ on finite traces.

Moreover, we conjecture that the complexity lower-bound of *LTL sketch existence* is NP-hard based on the NP-hardness of *LTL learning* for certain fragments of LTL [83]. However, we leave the exact lower-bound of the problem for future work.

### 5.2.2 SAT-based Decision Procedure

Based on the NP algorithm described above, we now devise a decision procedure to decide the *LTL sketch existence* problem. The decision procedure relies upon reducing the existence of

tables $T^{\varphi}_{uv^\omega}$ satisfying the three properties discussed in Section 5.2.1 to a satisfiability (SAT) problem.

This reduction relies on a symbolic encoding of the entries of the tables. To this end, we introduce propositional variables $y^{u,v}_{i,t}$ for each $i \in \{1,\ldots,n\}$, $t \in \{0,\ldots,|uv|-1\}$, and $uv^\omega \in P \cup N$. A variable $y^{u,v}_{i,t}$ encodes the entry $T^{\varphi}_{uv^\omega}[\varphi[i],t]$. Further, we encode the operators to be substituted for the Type-1 and Type-2 placeholders in $\varphi^?$ using the following variables: (i) $x_{i,\lambda}$ for each Node $i$ where $\ell(i)$ is a Type-1 placeholder and each $\lambda \in \Lambda_U$; and (ii) $x_{i,\lambda}$ for each Node $i$ where $\ell(i)$ is a Type-2 placeholder and each $\lambda \in \Lambda_B$.

We now impose constraints on the introduced variables to ensure that the prospective tables satisfy the three properties necessary for inferring a consistent LTL formula. We achieve this by constructing a propositional formula $\Phi^{\varphi^?,\mathcal{S}}$. This formula ensures that variables $y^{u,v}_{i,t}$ encode appropriate tables and using Lemma 6, its satisfiability ensures the existence of a suitable substitution for $\varphi^?$.

Internally,

$$\Phi^{\varphi^?,\mathcal{S}} := \Phi^{1,2}_? \wedge \Phi^{sem} \wedge \Phi^{con} \wedge \Phi^{suf} \tag{5.8}$$

is a conjunction of four formulas. The first conjunct $\Phi^{1,2}_?$ ensures that the Type-1 and Type-2 placeholders are substituted by appropriate operators. The conjuncts $\Phi^{sem}$, $\Phi^{con}$ and $\Phi^{suf}$ ensure that the variables $y^{u,v}_{i,t}$ encode entries of tables that satisfy the Semantic property (Equations 5.2, 5.3, 5.4 and 5.5), the Consistency property (Equation 5.6) and the Suffix property (Equation 5.7), respectively. In the remainder of the section, we describe the construction of each of the four formulas.

We begin by introducing the constraints required for $\Phi^{1,2}_?$. For each Node $i$ labeled with a Type-1 placeholder (that is, $\ell(i) \in \Pi^1$), we design the following constraint:

$$\Big[\bigvee_{\lambda\in\Lambda_U} x_{i,\lambda}\Big] \wedge \Big[\bigwedge_{\lambda\neq\lambda'\in\Lambda_U} \neg x_{i,\lambda} \vee \neg x_{i,\lambda'}\Big], \tag{5.9}$$

which ensures that the Type-1 placeholders are substituted with a unique unary operator. For Type-2 placeholders, we have the exact same constraint except that the operators range from the set of binary operators $\Lambda_B$. We now construct $\Phi^{1,2}_?$ simply by taking a conjunction of all such constraints for the nodes labeled with Type-1 and Type-2 placeholders.

Next, we define $\Phi^{sem}$ as the conjunction $\bigwedge_{uv^\omega\in P\cup N} \Phi^{u,v}$, where $\Phi^{u,v}$ denotes a formula that ensures that the variables $y^{u,v}_{i,t}$ satisfy the semantic relations for the trace $uv^\omega$. In the formula $\Phi^{u,v}$, for each Node $i$ labeled with X-operator (that is, $\ell(i) = \mathsf{X}$) and having Node $j$ as its left child, we have the following constraint:

$$\Big[\bigwedge_{0\le t<|uv|-1} \Big[y^{u,v}_{i,t} \leftrightarrow y^{u,v}_{j,t+1}\Big]\Big] \wedge \Big[y^{u,v}_{i,|uv|-1} \leftrightarrow y^{u,v}_{j,|u|}\Big] \tag{5.10}$$

This constraint ensures that the variables $y^{u,v}_{i,t}$ satisfy Equation 5.3 for the trace $uv^\omega$. For nodes labeled with other operators, we construct similar constraints based on their corresponding semantic relations. If the nodes are labeled with Type-1 or Type-2 placeholders, we additionally

rely on variables $x_{i,\lambda}$ to determine the operator $\lambda$ to be substituted in Node $i$. Based on the operator label $\lambda$, we devise appropriate semantic constraints. Finally, we construct $\Phi^{u,v}$ as the conjunction of all such semantic constraints.

We construct the following constraint to ensure Equation 5.6 is satisfied for the prospective tables:

$$\Phi^{con} := [\bigwedge_{uv^\omega \in P} y^{u,v}_{1,0}] \wedge [\bigwedge_{uv^\omega \in N} \neg y^{u,v}_{1,0}] \tag{5.11}$$

Finally, for $\Phi^{suf}$, we have the following constraint for each Node $i$ labeled with a Type-0 placeholder (that is, $\ell(i) \in \Pi^0$):

$$\bigwedge_{u_1 v_1^\omega[t,\infty) = u_2 v_2^\omega[t',\infty) \in suf(\mathcal{S})} \left[ y^{u_1,v_1}_{i,t} \leftrightarrow y^{u_2,v_2}_{j,t'} \right], \tag{5.12}$$

which ensures that Equation 5.7 is satisfied for the prospective tables.

Overall, we construct a formula $\Phi^{\varphi^?,\mathcal{S}}$ that ranges over $\mathcal{O}(n + nm)$ variables and is of size $\mathcal{O}(n + nm^3 + m^2)$, where $n = |\varphi^?|$ and $m = |\mathcal{S}|$. We conclude this section by stating the correctness of $\Phi^{\varphi^?,\mathcal{S}}$.

**Theorem 7.** *Let $\mathcal{S}$ be a sample, $\varphi^?$ a sketch, and $\Phi^{\varphi^?,\mathcal{S}}$ the formula as defined above. Then, $\Phi^{\varphi^?,\mathcal{S}}$ is satisfiable if and only if there exists a complete substitution $s$ such that $f_s(\varphi^?)$ is consistent with $\mathcal{S}$.*

*Proof.* For the forward direction, based on a model $V$ of $\Phi^{\varphi^?,\mathcal{S}}$, we construct a complete substitution $s$ such that $f_s(\varphi^?)$ is consistent with $\mathcal{S}$. First, due to constraints like Formula 5.9, we can substitute any Type-1 or Type-2 placeholder, say at Node $i$, with the unique operator $\lambda$ for which $V(x_{i,\lambda}) = 1$. Second, we construct substitutions for Type-0 placeholders by relying on tables $T^{\varphi^?}_{uv^\omega}$ that we construct from $V$ as follows: $T^{\varphi^?}_{uv^\omega}[\varphi^?[i], uv^w[t,\infty)] = V(y^{u,v}_{i,t})$ for each $uv^\omega \in P \cup N$ and $i \in \{1,\ldots,n\}$. Due to Formulas 5.10, 5.11, and 5.12, the constructed tables $T^{\varphi^?}_{uv^\omega}$ satisfy the Semantic, Consistency, and Suffix properties. As a result, one can now explicitly construct substitutions for Type-0 placeholders based on tables $T^{\varphi^?}_{uv^\omega}$, exploiting Lemma 6.

For the other direction, we construct a satisfying assignment $v$ using the substitution function $s$ and tables $T^{f_s(\varphi)}_{uv^\omega}$ for $uv^\omega \in P \cup N$. First, we assign $V(x_{i,\lambda}) = 1$ if and only if $s(?) = \lambda$ for a Node $i$ labeled with a Type-1 or Type-2 placeholder ?. Second, we assign $V(y^{u,v}_{i,t}) = T^{f_s(\varphi)}_{uv^\omega}[f_s(\varphi^?)[i], t]$ for each $uv^\omega \in P \cup N$ and $0 \leq t \leq |uv|$. This assignment $V$ satisfies $\Phi^{1,2}_?$, since we obtain $V$ from the syntax DAG of a valid LTL formula. Further, this assignment satisfies $\Phi^{sem}$, $\Phi^{con}$ and $\Phi^{suf}$ because the tables satisfy Semantic, Consistency and Suffix properties, respectively, on which the constraints are based. Overall, $V$ is a model for $\Phi^{\varphi^?,\mathcal{S}}$. □

### 5.2.3 Fixing an Incorrect Sketch

If a complete substitution does not exist for a sketch, then there must have been an error while formulating the sketch. One potential cause of the error could be that the system engineer

incorrectly specified the operators in the sketch due to the lack of expertise in temporal logic. For instance, continuing the example from Section 5.1.2, if the sketch were $\mathsf{F}(?^0)$ instead of $\mathsf{G}(?^0)$, then there is a possible completion $\mathsf{F}(p)$, which holds on $\alpha = \{p\}\{q\}^\omega$ but not on $\beta = \{q\}^\omega$. In this example, the engineer may have confused the temporal operators $\mathsf{G}$ and $\mathsf{F}$.

We now describe some procedures to suggest corrections to the sketch, in case there are no complete substitutions. Such procedures must be executed after the decision procedure based on $\Phi^{\varphi^?,\mathcal{S}}$. In particular, they must be executed only after $\Phi^{\varphi^?,\mathcal{S}}$ turns out to be unsatisfiable since there is a high likelihood of an error in this case.

We first describe a naive procedure: we first generate a new sketch $\varphi^?_{empty}$ from $\varphi^?$ by replacing all LTL operators with placeholders; in particular, we replace all unary operators with Type-1 placeholders and all binary operators with Type-2 placeholders. This sketch $\varphi^?_{empty}$ only retains the syntactic structure of the original sketch $\varphi^?$.

One can then apply the decision procedure based on the formula $\Phi^{\varphi^?_{empty},\mathcal{S}}$ to check whether $\varphi^?_{empty}$ admits a possible substitution. If $\Phi^{\varphi^?_{empty},\mathcal{S}}$ is satisfiable, one can generate a new sketch $\varphi^?_{new}$ using a model $v$: one substitutes each newly introduced Type-1 or Type-2 placeholder, say at Node $i$, with the unique operator $\lambda$ for which $v(x_{i,\lambda}) = 1$.

While the above simple procedure could suggest a new sketch $\varphi^?_{new}$ that admits a possible substitution, it is possible that $\varphi^?_{new}$ is rather different from $\varphi^?$. This is because the intermediate sketch $\varphi^?_{empty}$, on which the procedure relies, retains only the syntactic structure of $\varphi^?$ and none of the existing LTL operators. One could manually design $\varphi^?_{empty}$ to retain specific operators from $\varphi^?$. However, this requires manual intervention, which could be time-consuming, error-prone, etc.

To suggest corrections to $\varphi^?$ without much manual intervention, we suggest an improvement over the above procedure. In this procedure, we rely on the maximum satisfiability (MaxSAT) problem, introduced in the previous chapter, Section 4.1.2. We use $\Phi^{\varphi^?_{empty},\mathcal{S}}$ as a hard constraint. As a soft constraint, we have the following:

$$\Phi^{\varphi^?} := \bigwedge_{\substack{1\le i\le n \\ \ell(i)\in\Lambda_U\cup\Lambda_B}} x_{i,\ell(i)} \tag{5.13}$$

As weights to the soft constraints, we set $w(x_{i,\ell(i)}) = 1$ for each Node $i$ that is labeled with a unary or binary operator in $\varphi^?$. Such soft constraints ensure that the new sketch $\varphi^?_{new}$ is structurally as close as possible to the original sketch $\varphi^?$.

The correctness of the procedures described here is due to the correctness of the encoding $\Phi^{\varphi^?_{empty},\mathcal{S}}$, which follows directly from Theorem 7.

## 5.3 Algorithms to complete an LTL sketch

We now describe two novel algorithms for solving the *LTL sketching* problem, which aim at searching for concise LTL formulas from sketches, as alluded to in the introduction. Thus, our first algorithm relies on existing techniques to learn *minimal* LTL formulas. Our second algorithm, alternatively, searches for formulas of increasing size based on constraint solving.

### 5.3.1 Algorithm based on LTL learning

This algorithm, which we refer to as `Algo1`, builds upon the decision procedure for checking the existence of a complete substitution presented in Section 5.2.2. In particular, it relies on $\Phi^{\varphi^?,\mathcal{S}}$ from the decision procedure to construct substitutions for placeholders of a sketch. While it is straightforward to substitute Type-1 and Type-2 placeholders, the algorithm relies on the classic LTL learning problem to substitute Type-0 placeholders. The pseudocode of the algorithm is sketched in Algorithm 7.

**Algorithm 7** Algorithm based on LTL learning

**Input:** Sample $\mathcal{S}$, Sketch $\varphi^?$

1: Construct $\Phi^{\varphi^?,\mathcal{S}} = \Phi^{1,2}_? \wedge \Phi^{sem} \wedge \Phi^{con} \wedge \Phi^{suf}$
2: **if** $\Phi^{\varphi^?,\mathcal{S}}$ is not satisfiable **then**
3: **return** LTL formula does not exist
4: **end if**
5: Substitute Type-1 and Type-2 placeholders in $\varphi^?$ using $V$
6: **for** every $i$ such that $\ell(i) \in \Pi^0$ **do**
7: Construct $\mathcal{S}_i = (P_i, N_i)$
8: $\varphi_i \leftarrow$ `Learn`$(\mathcal{S}_i)$
9: Substitute Node $i$ with $\varphi_i$ in $\varphi^?$
10: **end for**
11: **return** $\varphi^?$

The first step of the algorithm is to construct $\Phi^{\varphi^?,\mathcal{S}}$ from the given sample (Line 1) and sketch, as described in Section 5.2.2. If $\Phi^{\varphi^?,\mathcal{S}}$ is unsatisfiable, the algorithm straight-away returns that no solution exists, as established by Theorem 7. (In this case, one can use procedures from Section 5.2.3 to fix the sketch.) If satisfiable, we use a model $v$ of $\Phi^{\varphi^?,\mathcal{S}}$ (obtained from any off-the-shelf SAT solver) to complete $\varphi^?$, the details of which we describe next.

Given a model $V$ of $\Phi^{\varphi^?,\mathcal{S}}$, one can substitute the Type-1 and Type-2 placeholders in $\varphi^?$ (Line 5) as follows: for each Node $i$ where $\ell(i)$ is a Type-1 and Type-2 placeholders, assign $s(\ell(i)) = \lambda$, where $\lambda$ is the unique operator for which $V(x_{i,\lambda}) = 1$.

The Type-0 placeholders, however, are more challenging to substitute. This is because they represent entire LTL formulas. Towards substituting Type-0 placeholders (Line 7), for every Node $i$ for which $\ell(i)$ is a Type-0 placeholder (that is, $\ell(i) \in \Pi^0$), we first construct a sample $\mathcal{S}_i = (P_i, N_i)$ as

$$P_i = \{uv^\omega[t,\infty) \in suf(\mathcal{S}) \mid V(y^{u,v}_{i,t}) = 1\}, \text{ and} \tag{5.14}$$

$$N_i = \{uv^\omega[t,\infty) \in suf(\mathcal{S}) \mid V(y^{u,v}_{i,t}) = 0\}. \tag{5.15}$$

We now learn a minimal LTL formula $\varphi_i$ consistent with the sample $\mathcal{S}_i$ (using some *LTL learning* algorithm [169, 193, 187]) for substituting $\ell(i)$. Intuitively, such formulas $\varphi_i$ ensure that the tables $T^{\varphi}_{uv^\omega}$ of $\varphi$ obtained by completing $\varphi^?$ satisfy the Semantic, Consistency and Suffix properties described in Section 5.2.1.

We now establish the correctness of the algorithm using the following theorem:

**Theorem 8.** *Given sketch $\varphi^?$ and sample $\mathcal{S}$, `Algo1` completes $\varphi^?$ to output an LTL formula that is consistent with $\mathcal{S}$ if such a formula exists, otherwise returns no such formula exists.*

Observe that this algorithm constructs new samples for each Type-0 placeholder, each of which have size $\mathcal{O}(|suf(\mathcal{S})|) = \mathcal{O}(|\mathcal{S}|^2)$. This poses a challenge to the scalability of this algorithm. Furthermore, the new samples are not optimized to produce the minimal possible substitutions. Our next algorithm improves both the runtime and the size of the inferred specification.

### 5.3.2 Algorithm based on incremental SAT solving

We now describe an algorithm, abbreviated as `Algo2`, that reduces *LTL sketching* to a series of SAT solving problems, inspired by the SAT-based algorithm of Neider et al. [169]. The pseudocode of the algorithm is sketched in Algorithm 8.

Similar to Neider et al. [169], given a sample $\mathcal{S}$ and a number $n \in \mathbb{N} \setminus \{0\}$, the crux of this algorithm is to construct a propositional formula $\Psi_n^{\varphi^?,\mathcal{S}}$, of size $poly(|\varphi^?|, |\mathcal{S}|)$ to search for the desired formula. The formula $\Psi_n^{\varphi^?,\mathcal{S}}$ we construct has the following properties:

1. $\Psi_n^{\varphi^?,\mathcal{S}}$ is satisfiable if and only if one can complete $\varphi^?$ to obtain an LTL formula of size at most $n$ that is consistent with $\mathcal{S}$; and
2. using a model $v$ of $\Psi_n^{\varphi^?,\mathcal{S}}$, one can complete $\varphi^?$ to construct a consistent LTL formula of size at most $n$.

**Algorithm 8** Algorithm based on incremental SAT solving (Section 5.3.2)

**Input:** Sample $\mathcal{S}$, Sketch $\varphi^?$

Construct $\Phi^{\varphi^?,\mathcal{S}} := \Phi_?^{1,2} \wedge \Phi^{sem} \wedge \Phi^{con} \wedge \Phi^{suf}$
**if** $\Phi^{\varphi^?,\mathcal{S}}$ is not satisfiable **then**
    **return** LTL formula does not exist
**end if**
$n \leftarrow |\varphi^?| - 1$
**repeat**
    $n \leftarrow n + 1$
    Construct $\Psi_n^{\varphi^?,\mathcal{S}} := \Phi_?^{1,2} \wedge \tilde{\Phi}^{sem} \wedge \Phi^{con} \wedge \Phi_{?,n}^{0}$
**until** $\Psi_n^{\varphi^?,\mathcal{S}}$ is satisfiable (say with model $v$)
Construct $\varphi^?$ using $v$
**return** $\varphi^?$

However, in contrast to the algorithms by Neider and Gavran, we first solve $\Phi^{\varphi^?,\mathcal{S}}$ (discussed in Section 5.2.2) to determine the existence of a complete substitution. (If $\Phi^{\varphi^?,\mathcal{S}}$ is unsatisfiable, one can use the procedures from Section 5.2.3 to fix the sketch.) If and only if $\Phi^{\varphi^?,\mathcal{S}}$ is satisfiable, our algorithm checks the satisfiability of $\Psi_n^{\varphi^?,\mathcal{S}}$ for increasing values of $n$ (starting from $|\varphi^?| - 1$) to search for an LTL formula of size at most $n$ that has the same syntactic structure as $\varphi^?$. We construct the resulting LTL formula by substituting the placeholders in $\varphi^?$ based on a model $V$ of the formula $\Psi_n^{\varphi^?,\mathcal{S}}$, similar to what we do in

`Algo1`. The termination of this algorithm is guaranteed by the decision procedure encoded by $\Phi^{\varphi^?,\mathcal{S}}$. The procedure ensures that we search for a solution only if there exists a complete and consistent LTL formula, to begin with. Moreover, the properties of $\Psi_n^{\varphi^?,\mathcal{S}}$ ensure that we find the suitable LTL formula if there exists one.

On a technical level, the formula $\Psi_n^{\varphi^?,\mathcal{S}}$ is obtained by modifying certain parts of the formula $\Phi^{\varphi^?,\mathcal{S}}$. Precisely, $\Psi_n^{\varphi^?,\mathcal{S}} := \Phi_?^{1,2} \wedge \tilde{\Phi}^{sem} \wedge \Phi^{con} \wedge \Phi_{?,n}^0$ and it introduces two modifications in $\Phi^{\varphi^?,\mathcal{S}}$: a new formula $\Phi_{?,n}^0$ replaces $\Phi^{suf}$; and $\tilde{\Phi}^{sem}$ adds more constraints to $\Phi^{sem}$. The formula $\Phi_{?,n}^0$ encodes the structure of LTL formulas that substitute the Type-0 placeholders. $\tilde{\Phi}^{sem}$, again as in $\Phi^{sem}$, ensures that the variables $y_{i,t}^{u,v}$ encode table entries $T_{uv^\omega}^{\varphi}[\varphi[i],t]$ that satisfy equations (that is, Equations 5.1 to 5.5, etc.) describing the Semantic property. We now briefly describe the constraints for the newly introduced formulas.

The formula $\Phi_{?,n}^0$ relies on an additional set of variables: (i) $x_{i,\lambda}$ for each Node $i$ where $\ell(i)$ is a Type-0 placeholder or $i \in \{|\varphi^?|+1,\ldots,n\}$, and each $\lambda \in \Lambda$ ; and (ii) $l_{i,j}$ and $r_{i,j}$ for each Node $i$ where $\ell(i)$ is a Type-0 placeholder or $i \in \{|\varphi^?|+1,\ldots,n\}$, and each $j \in \{\max(i,|\varphi^?|),\ldots,n\}$. The variable $x_{i,\lambda}$, again, encodes that Node $i$ is labeled with $\lambda$. The variables $l_{i,j}$ (respectively, $r_{i,j}$) encode that the left (respectively, the right) child of Node $i$ is Node $j$. Together the new variables encode the structure of the prospective LTL formulas for Type-0 placeholders.

We now impose constraints, similar to Formula 5.9, on the variables $x_{i,\lambda}$ to ensure each node is labeled by a unique LTL operator from $\Lambda$. Further, we impose constraints to ensure that each Node $i$ has unique left and right children. Finally, we construct $\Phi_{?,n}^0$ as the conjunction of all such structural constraints.

The formula $\tilde{\Phi}^{sem}$ also relies on new variables $y_{i,t}^{u,v}$ for each Node $i$ labeled with a Type-0 variables or $i \in \{|\varphi^?|+1,\ldots,n\}$, each $t \in \{0,\ldots,|uv|-1\}$ and each $uv^\omega$ in $\mathcal{S}$. Now, we construct semantic constraints such as:

$$\left[x_{i,\mathsf{X}} \wedge l_{i,j}\right] \rightarrow \bigwedge_{0 \le t < |uv|-1} \left[y_{i,t}^{u,v} \leftrightarrow y_{j,t+1}^{u,v}\right] \wedge \left[y_{i,|uv|-1}^{u,v} \leftrightarrow y_{j,|u|}^{u,v}\right], \tag{5.16}$$

that ensures that the $y_{i,t}^{u,v}$ variables encode entries of table that satisfy Equation 5.3. We construct $\tilde{\Phi}_{sem}$ as the conjunction of $\Phi^{sem}$ and the new semantic constraints.

We establish the correctness guarantees using the following theorem:

**Theorem 9.** *Given sketch $\varphi^?$ and sample $\mathcal{S}$, `Algo2` completes $\varphi^?$ to output an LTL formula that is consistent with $\mathcal{S}$ if such a formula exists, otherwise returns no such formula exists.*

`Algo2` searches for substitutions of Type-0 placeholders of increasing size and, thus, is able to find small substitutions for the sketch. However, it may not always find a minimal consistent LTL formula because a minimal formula may require the parts of the substitution to share subformulas from the existing sketch.

To demonstrate this, consider the sketch $\mathsf{F}(?_0) \vee \mathsf{F}\,\mathsf{G}\,p$ and the sample consisting of one positive trace $\{\}\{p\}^\omega$ and one negative trace $\{\}^\omega$. For this input, a possible output by `Algo2` is the formula $\mathsf{F}\,p \vee \mathsf{F}\,\mathsf{G}\,p$, which is of size 5. However, the minimal consistent formula

$\mathsf{F}\,\mathsf{G}\,p \vee \mathsf{F}\,\mathsf{G}\,p$ is of size 4. In this example, substituting $?_0$ with $\mathsf{G}\,p$ produces a smaller formula than substituting it with $p$, since $\mathsf{G}\,p$ allows more sharing of subformulas.

While `Algo2` may not always return a minimal formula, we can provide an upper bound on its size, thus ensuring its conciseness. To compute this bound, we define the syntax size $|\varphi|_s$ of a formula $\varphi$ to be the number of operators and propositions appearing in $\varphi$. Typically, the syntax size $|\varphi|_s$ is larger than the (DAG) size $|\varphi|$, since it counts all the operators and propositions, including the repeating ones. For instance, for $\varphi = \mathsf{F}\,\mathsf{G}\,q \vee \mathsf{F}\,\mathsf{G}\,q$, $|\varphi|_s = 7$, while $|\varphi| = 4$.

We now state the guarantee on the size of the formula returned by `Algo2` in the following theorem. Intuitively, the theorem states the size $|\varphi|$ of the formula that `Algo2` returns is bounded by the syntax size $|\varphi^*|_s$ of the minimal (DAG size) solution $\varphi^*$.

**Theorem 10.** *For a given sample $\mathcal{S}$ and sketch $\varphi^?$, let $\varphi^*$ be a minimal formula that is consistent with $\mathcal{S}$ and can be obtained by completing $\varphi^?$. Then, `Algo2` returns a formula $\varphi$ that is consistent with $\mathcal{S}$, can be obtained by completing $\varphi^?$ and has size $|\varphi| \leq |\varphi^*|_s$.*

*Proof.* Towards contradiction, we assume that the LTL formula $\varphi$ returned by `Algo2` has size $|\varphi| > |\varphi^*|_s$. Now, based on the property of $\Psi_n^{\varphi^?,\mathcal{S}}$ (see first paragraph of Section 5.3.2), $\Psi_n^{\varphi^?,\mathcal{S}}$ is satisfiable for $n = |\varphi^*|_s$. This is because there exists a consistent LTL formula of size at most $n = |\varphi^*|_s$, $\varphi^*$ itself. Thus, `Algo2`, due to its incremental search, should have returned $\varphi^*$, contradicting our assumption. □ □

## 5.4 Experimental Evaluation

In this section, we design experiments to answer the following research questions:

**RQ1**: Which of the two presented sketching algorithms is more effective?

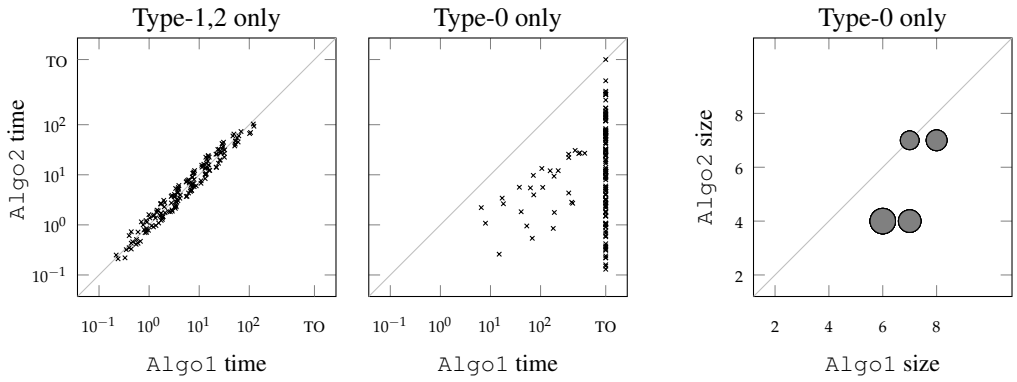
**RQ2**: How do our algorithms compare against other specification mining tools for LTL?

To answer these questions, we have implemented a prototype of our algorithms in `Python3`, named `LTL-Sketcher`[1]. In `LTL-Sketcher`, we additionally implement two heuristics to improve the runtime of our algorithms, both of which are directed toward optimizing the SAT encoding used in the algorithms. We briefly mention the idea behind the heuristics.

The first heuristic is inspired by the SAT encoding used in Bounded Model Checking [29]. The encoding exploits a succinct description of the semantics of LTL using expansion laws [14]. Exemplarily, the expansion law for the U-operator is $\varphi\,\mathsf{U}\,\psi = \psi \vee (\varphi \wedge \mathsf{X}(\varphi\,\mathsf{U}\,\psi))$, which relies on checking satisfaction in the next position using X-operator. Using the LTL expansion laws reduces the number of variables required in $\Phi^{sem}$.

In the second heuristic, we create variables $y_{i,t}^{u,v}$ only for the distinct suffixes $uv^\omega$ in $\mathcal{S}$. This is sufficient because LTL formulas have the same evaluation on equal suffixes (which is also the basis for Equation 5.7). Hence, if two traces share a suffix, we can create the

[1]The code can be found in https://github.com/rajarshi008/LTLSketcher




(a) Runtime comparison

(b) Size comparison

FIGURE 5.3: Comparison of `Algo1` and `Algo2` with respect to runtime (in seconds) and the size of inferred formulas. The points below the diagonal are where `Algo2` performs better. In Figure 5.3a, "TO" denotes timeouts. In Figure 5.3b, the size of a bubble is proportional to the number of cases.

variables encoding the semantics only once, reducing the total number of variables. Also, in this heuristic, the constraint $\Phi^{suf}$ imposing the Suffix property becomes unnecessary.

**Benchmarks.** For evaluating our algorithms, following the literature in *LTL learning*, we rely on benchmarks generated synthetically using common LTL formulas used in practice [77]. We choose the same nine formulas also chosen by Neider and Gavran [169] for generating their benchmarks. We, however, deviate from their method of generating benchmarks. This is because, as observed by Raha et al. [187], their method, being fairly naive, consumes more time and often does not generate adequately different trajectories from a chosen LTL formula. We, in contrast, design a novel method of generating samples based on random sampling of traces from Büchi automata [28] constructed from the LTL formulas (using `Spot` [74]). Overall, we generate 18 samples for each of the nine formulas (that is, 162 samples in total), with the number of examples varying from 20 to 800 and the length of traces varying from 4 to 16. We conduct all the experiments on a single core of an Intel Xeon E7-8857 CPU (at 3 GHz) using upto 6 GB of RAM.

### 5.4.1 RQ1: Comparison of Sketching algorithms

To answer RQ1, we compare `Algo1` (from Section 5.3.1) and `Algo2` (from Section 5.3.2) based on their running times and the size of formula inferred. For this comparison, as sketches, we remove parts (upto 50% in size) of each formula to construct two kinds of sketches: one with only Type-1 or Type-2 placeholders and one with only Type-0 placeholders. All the chosen formulas and their corresponding sketches are presented in Table 5.1. We now run the algorithms on the 18 samples and two sketches generated from each of the nine formulas with a timeout of 900 secs.

We depict the runtime comparisons in Figure 5.3a. We observe that while both the algorithms have comparable runtime on sketches with only Type-1 or Type-2 placeholders, `Algo1` performs significantly worse on sketches with only Type-0 placeholders with 134 timeouts.

TABLE 5.1: Sketches considered in RQ1

| Original Formula | Type-0 sketch | Type-1-2 sketch |
|---|---|---|
| $\mathsf{F}(p)$ | $\mathsf{F}(?^0)$ | $?^1(p)$ |
| $\mathsf{G}(p)$ | $\mathsf{G}(?^0)$ | $?^1(p)$ |
| $\mathsf{G}(\neg(p))$ | $\mathsf{G}(\neg(?^0))$ | $?^1(\neg(p))$ |
| $\mathsf{F}(q) \rightarrow (\neg p \,\mathsf{U}\, q)$ | $?^0 \rightarrow (\neg p \,\mathsf{U}\, q)$ | $?^1(q) \rightarrow (\neg p ?^2 q)$ |
| $\mathsf{F}(q) \rightarrow (p \,\mathsf{U}\, q)$ | $?^0 \rightarrow (\neg p \,\mathsf{U}\, q)$ | $?^1(q) \rightarrow (p ?^2 q)$ |
| $\mathsf{G}(q \rightarrow \mathsf{G}(p))$ | $\mathsf{G}(q \rightarrow ?^0)$ | $?^1_1(q \rightarrow ?^1_2(p))$ |
| $\mathsf{G}(q \rightarrow \mathsf{G}(\neg p))$ | $\mathsf{G}(q \rightarrow ?^0)$ | $?^1_1(q \rightarrow ?^2_1(\neg p))$ |
| $\mathsf{G}(\neg p) \vee \mathsf{F}(p \wedge \mathsf{F}(q)))$ | $?^0 \vee \mathsf{F}(p \wedge \mathsf{F}(q)))$ | $\mathsf{G}(\neg p ?^1_2 \mathsf{F}(p ?^2_2 \mathsf{F}(q)))$ |
| $\mathsf{G}(p \wedge (\neg q \rightarrow (\neg q \,\mathsf{U}(r \wedge \neg q))))$ | $\mathsf{G}(p \wedge (\neg q \rightarrow (\neg q \,\mathsf{U} ?^0)))$ | $\mathsf{G}(p \wedge (\neg q ?^2_1 (\neg q ?^2_2 (r \wedge \neg q))))$ |

TABLE 5.2: Formulas and their corresponding sketches considered for comparison against Texada in RQ2

| Formula | full sketch | medium sketch | small sketch |
|---|---|---|---|
| $\mathsf{F}(q) \rightarrow (\neg p \,\mathsf{U}\, q)$ | $\mathsf{F}(?^0_1) \rightarrow (\neg ?^0_2 \,\mathsf{U}\, ?^0_3)$ | $\mathsf{F}(?^0_1) \rightarrow ?^0_2$ | $?^0_1$ |
| $\mathsf{F}(q) \rightarrow (p \,\mathsf{U}\, q)$ | $\mathsf{F}(?^0_1) \rightarrow (?^0_2 \,\mathsf{U}\, ?^0_3)$ | $\mathsf{F}(?^0_1) \rightarrow ?^0_2$ | $?^0_1$ |
| $\mathsf{G}(q \rightarrow \mathsf{G}(\neg p))$ | $\mathsf{G}(?^0_1 \rightarrow \mathsf{G}(\neg ?^0_2))$ | $\mathsf{G}(?^0_1 \rightarrow ?^0_2)$ | $?^0_1$ |
| $\mathsf{G}(\neg p \vee \mathsf{F}(p \wedge \mathsf{F}(q)))$ | $\mathsf{G}(\neg ?^0_1 \vee \mathsf{F}(?^0_2 \wedge \mathsf{F}(?^0_3)))$ | $\mathsf{G}(\neg ?^0_1 \vee ?^0_2)$ | $?^0_1$ |
| $\mathsf{G}(p \wedge (\neg q \rightarrow (\neg q \,\mathsf{U}(r \wedge \neg q))))$ | $\mathsf{G}(?^0_1 \wedge (\neg ?^0_2 \rightarrow (\neg ?^0_3 \,\mathsf{U}(?^0_4 \wedge \neg ?^0_5))))$ | $\mathsf{G}(?^0_1 \wedge (\neg ?^0_2 \rightarrow ?^0_3))$ | $?^0_1$ |
| $\mathsf{G}(q \rightarrow \mathsf{G}(p))$ | $\mathsf{G}(?^0_1 \rightarrow \mathsf{G}(?^0_2))$ | $\mathsf{G}(?^0_1 \rightarrow ?^0_2)$ | $?^0_1$ |

We also depict the comparison of formula size in Figure 5.3b. We notice that `Algo2` returns smaller formulas than `Algo1` in many cases. The reason `Algo1` performs slow and returns large formulas is that it solves *LTL learning* on potentially large intermediate samples for sketches with Type-0 placeholders. Thus, we answer RQ1 in favor of `Algo2`.

### 5.4.2 RQ2: Comparison against LTL mining tools.

To address RQ2, we compared `LTL-Sketcher` against two prominent approaches for mining specifications in LTL. The first approach completes user-defined templates with (Boolean combinations of) atomic propositions. For this approach, we select the popular LTL miner `Texada` [144]. The second approach learns LTL formulas of minimal size without syntactic constraints. For this approach, we choose `Flie` [169] as a prototypical example of this class of algorithms.

The setting of `Texada` differs from ours in that it permits positive examples only, and these examples have to be finite traces. Thus, in order to have a fair comparison, we make minor modifications to our SAT encoding (specifically to the X-operator) to handle finite traces. Furthermore, our tool does not require one to provide negative examples and, hence, can immediately be applied.

To compare `Texada` and `LTL-Sketcher`, we considered six of the nine formulas used in RQ1, dropping the smallest three (see Table 5.2). For each formula, we created ten samples with only positive, finite traces by truncating ultimately periodic and ensuring consistency with the formula. Also, we created three sketches for each formula, retaining different amounts

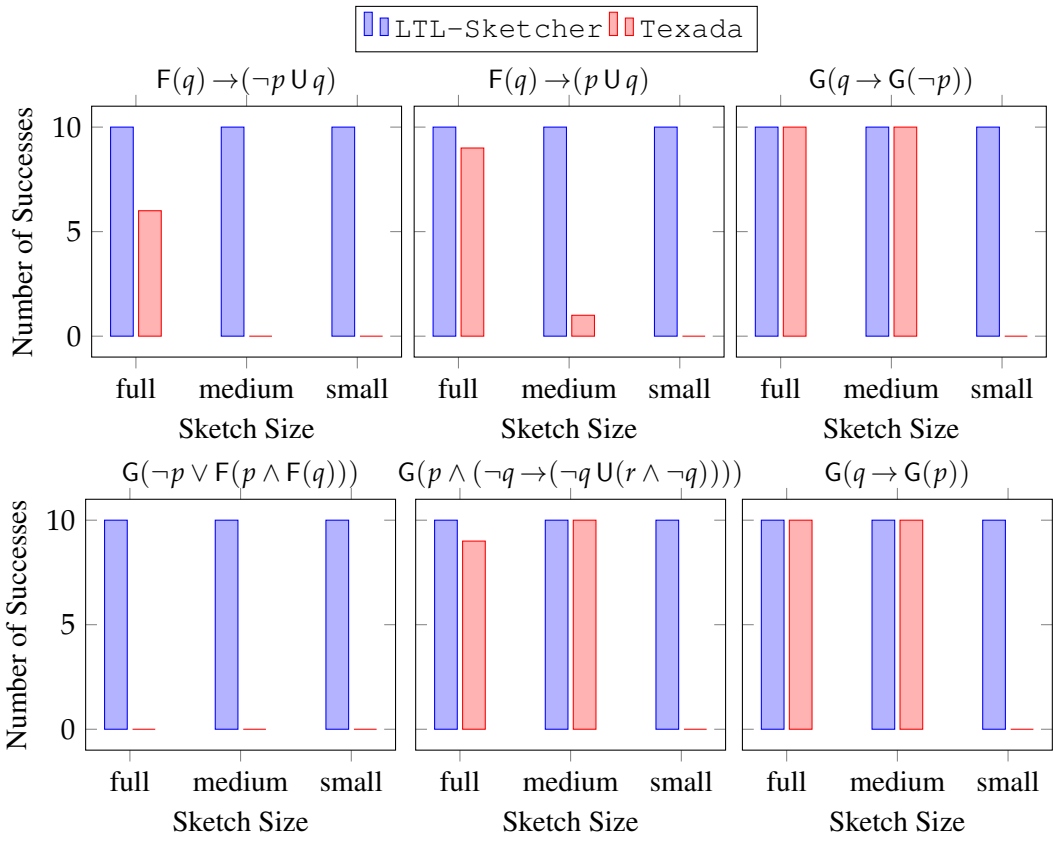



FIGURE 5.4: Number of successes by `Texada` and `LTL-Sketcher` for completing sketches of different LTL formulas (indicated in the chart titles).

of information: in a *full sketch*, we only replaced each atomic proposition with a different Type-0 placeholder; in a *medium sketch*, we replaced a larger subformula containing at least one temporal operator; and in a *small sketch*, we replaced the formula with a single Type-0 placeholder. As an example, from formula $\mathsf{F}(q) \rightarrow (\neg p \,\mathsf{U}\, q)$, we constructed the full sketch $\mathsf{F}(?_1^0) \rightarrow (\neg ?_2^0 \,\mathsf{U}\, ?_3^0)$, the medium sketch $\mathsf{F}(?_1^0) \rightarrow ?_2^0$ and the small sketch $?_1^0$. All the chosen formulas and their corresponding sketches are presented in Table 5.2.

We ran `Texada` and `LTL-Sketcher` on each of these sketches and all corresponding samples and counted the cases in which the tools could provide a substitution. The results are shown in Figure 5.4. We notice that `Texada` found substitutions for the full sketches in most cases. However, when we removed more structural information from the specifications (that is, medium and small sketches), `Texada` was rarely able to complete a sketch. By contrast, `LTL-Sketcher` provided a substitution in every benchmark. The reason is that `Texada`'s strategy of exclusively searching for atomic propositions is only feasible if the user can provide a detailed template where all temporal operators are specified. Our tool, in contrast, alleviates the burden of writing complex temporal operators and, thus, is more flexible.

To compare `Flie` and `LTL-Sketcher`, we estimated how many examples are required to infer the desired specification. For this experiment, we used the same set of nine LTL formulas and sketches with varying amounts of missing information. To calculate the number of examples required, we designed a counterexample-guided strategy to compute a *minimal*

**Algorithm 9** Minimal Sample generation algorithm

**Input:** Desired formula $\varphi$, Sketch $\varphi^?$

$P \leftarrow \emptyset, N \leftarrow \emptyset$
$\varphi' \leftarrow \mathit{true}$
**while** $\varphi' \not\equiv \varphi$ (semantic equivalence) **do**
  **if** $\varphi \not\subseteq \varphi'$ **then**
    Generate one of the shortest $u$ such that $u \models \varphi \wedge \neg\varphi'$
    $P \leftarrow P \cup \{u\}$
    $\varphi' \leftarrow$ `Algo2` with inputs $(\mathcal{S} = (P, N), \varphi^?)$
  **else**
    Generate one of the shortest $u$ such that $u \models \varphi' \wedge \neg\varphi$
    $N \leftarrow N \cup \{u\}$
    $\varphi' \leftarrow$ `Algo2` with inputs $(\mathcal{S} = (P, N), \varphi^?)$
  **end if**
**end while**

**return** $\mathcal{S} = (P, N)$

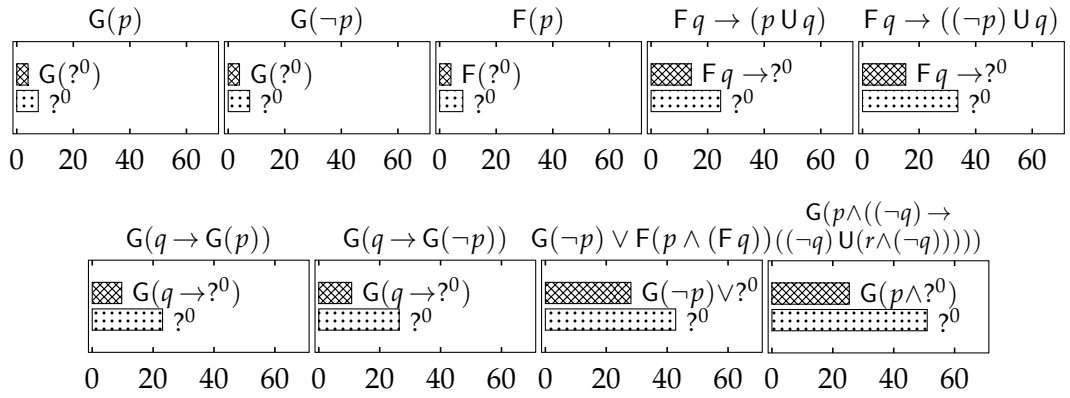



FIGURE 5.5: Comparison against Flie in terms of the average sample sizes required to recover the original formula for RQ2. In each chart, the original formula is indicated in the chart title, the sample size for `Flie` is indicated using the bar with the trivial sketch $?^0$, while the one for `LTL-Sketcher` is indicated using the other bar.

sample required for both tools to obtain the desired formula from a sketch of it. In this strategy, if a tool does not return the desired formula with the current sample, we add one of the shortest counterexamples to the sample that helps eliminate the current solution formula. We continue this process and end up with a minimal sample of both tools for each sketch. The exact algorithm used is sketched in Algorithm 9.

Figure 5.5 presents the average size of minimal samples (over ten runs) required to recover the desired formulas from their sketches. We observed that `Flie` required more examples than `LTL-Sketcher` to single out the correct specifications in all the cases. This asserted the fact that providing high-level insights as a sketch reduces the number of examples required to derive the desired specification. Thus, to answer RQ2, the ability to handle sketches provides `LTL-Sketcher` an edge over existing LTL mining tools.

## 5.5 Conclusion

In this work, we introduce LTL sketching—a novel way of writing formal specifications in LTL. The key idea is that a user can write a partial specification, that is, a sketch, which is then completed based on given examples of desired and undesired system behavior. We have shown that the sketching problem is in NP, presented two SAT-based sketching algorithms and some heuristics to improve their performance. Our experimental evaluation has shown that our algorithms can effectively complete sketches consisting of different types of missing information. Further, the ability to handle sketches provides our algorithms an edge over existing LTL mining approaches.

A natural direction for future work is to lift the idea of specification sketching to other specification languages, such as Signal Temporal Logic (STL) [157], the Property Specification Language (PSL) [79], or even visual specifications, such as UML (high-level) message sequence charts [112]. Moreover, we intend to extend the notion of sketching beyond the use of examples (e.g., by allowing the engineer to constrain placeholders using simple logical formulas or regular expressions).

# Chapter 6

# Learning from Positive Examples Only

In this chapter, we consider the input data for passive learning to be different from what we have seen so far; we consider *only* positive examples. This setting is motivated by the fact that negative examples are hard to observe in many scenarios, especially in safety-critical domains. Obtaining negative examples from systems such as medical equipments and self-driving cars can be unrealistic and may involve, for instance, harming patients and hitting pedestrians, respectively. Also, it is possible that one has access only to a black-box implementation of the system from which one can extract only its possible (that is, positive) executions.

Due to the relevance of learning from positive examples, we study the problem for two fundamental models of computation: deterministic finite automata (DFAs) [183]; and linear temporal logic (LTL) [179]. As we explained in Chapter 1, both models have a wide range of applications due to their useful algorithmic properties and interpretable structures.

Most works on learning DFAs and $\text{LTL}_\text{f}$ (for that matter, other automata and logics) have primarily focused on the standard passive learning problem, which asks to learn from a sample partitioned into positives and negatives. This problem is also known as the *binary classification* problem since one searches for models that classify between two classes of examples.

In spite of being relevant, the problem of learning concise DFAs and $\text{LTL}_\text{f}$ formulas from only positive examples, that is, the corresponding *one class classification* (OCC) problem, has garnered little attention. The primary reason, we believe, is that, like most OCC problems, this problem is an ill-posed one. Specifically, a concise model that classifies all the positive examples correctly is the trivial model that classifies all examples as positive. This corresponds to a single state DFA or, in LTL, the formula *true*. These models, unfortunately, convey no insights about the underlying system.

To ensure a well-defined problem, Avellaneda et al. [11], who study the OCC problem for DFAs, propose the use of the (accepted) *language* of a model as a regularizer. Searching for a model that has minimal language, however, results in one that classifies exactly the given examples as positive. To avoid this overfitting, they additionally impose an upper bound on the size of the model. Thus, the OCC problem that they state is the following: given a set $P$ of positive traces and a size bound $n$, learn a DFA that (i) classifies traces in $P$ correctly, (ii) has size at most $n$, and (iii) is language minimal. For language comparison, the order chosen is set inclusion.

To solve this OCC problem, Avellaneda et al. [11] then propose a *counterexample-guided* algorithm. This algorithm relies on generating suitable negative examples (that is, counterexamples) iteratively to guide the learning process. Since only the negative examples dictate the algorithm, in many iterations of their algorithm, the learned DFAs do not have a language smaller (in terms of inclusion) than the previous hypothesis DFAs. This results in searching through several unnecessary DFAs.

To alleviate this drawback, our first contribution is a *symbolic* algorithm for solving the OCC problem for DFA. Our algorithm converts the search for a language minimal DFA symbolically to a series of satisfiability problems in Boolean propositional logic, eliminating the need for counterexamples. The key novelty is an efficient encoding of the language inclusion check for DFAs in a propositional formula, which is polynomial in the size of the DFAs. We then exploit an off-the-shelf SAT solver to check the satisfiability of the generated propositional formulas and, thereafter, construct a suitable DFA. We expand on this algorithm in Section 6.1.

We then present two novel algorithms for solving the OCC problem for formulas in $\text{LTL}_\text{f}$. While our algorithms extend smoothly to traditional LTL (over infinite traces), our focus here is on $\text{LTL}_\text{f}$ due to its numerous applications in AI [100]. Also, $\text{LTL}_\text{f}$ being a strict subclass of DFAs, the learning algorithms for DFAs cannot be applied directly to learn $\text{LTL}_\text{f}$ formulas.

Our first algorithm for $\text{LTL}_\text{f}$ is a *semi-symbolic* algorithm, which combines ideas from both the symbolic and the counterexample-guided approaches. Roughly, this algorithm exploits negative examples to overcome the theoretical difficulties of symbolically encoding language inclusion for $\text{LTL}_\text{f}$; $\text{LTL}_\text{f}$ inclusion check is known to be inherently harder than that for DFAs [203]. Our second algorithm is simply a counterexample-guided algorithm that relies solely on the generation of negative examples for learning. We refer to Section 6.2 for details of both algorithms.

To further study the presented algorithms, we empirically evaluate them in several case studies. We demonstrate that our symbolic algorithm solves the OCC problem for DFA in fewer (approximately one-tenth) iterations and runtime comparable to the counterexample-guided algorithm, skipping thousands of counterexample generations. Further, we demonstrate that our semi-symbolic algorithm solves the OCC problem for $\text{LTL}_\text{f}$ (in average) thrice as fast as the counterexample-guided algorithm. We present all of our experimental results in Section 6.3 and end with a discussion in Section 6.4.

### Related Work.

The OCC problem described here belongs to the body of works labeled as passive learning [103]. As we discussed already in Chapter 2, the most popular problem in passive learning is the binary classification problem for DFAs [30, 108, 116] and $\text{LTL}\backslash\text{LTL}_\text{f}$ formulas [169, 48, 187].

The OCC problem of learning formal models from positive examples was first studied by Gold [104]. This work showed that the exact identification (in the limit) of certain models (which include DFAs and $\text{LTL}\backslash\text{LTL}_\text{f}$ formulas) from positive examples is not possible. Thereby, works have mostly focussed on models that are learnable easily from positive examples, such as

pattern languages [6], stochastic finite state machines [50], and hidden Markov Models [207]. None of these works considered learning DFAs or LTL formulas, mainly due to the lack of a meaningful regularizer.

Recently, Avellaneda et al. [11] proposed the use of language minimality as a regularizer and, thereafter, developed an effective algorithm for learning DFAs. While their algorithm cannot overcome the theoretical difficulties shown by Gold [104], they still produce a DFA that is a concise description of the positive examples. We significantly improve their algorithm for learning DFAs by relying on a novel encoding of language minimality using propositional logic. We additionally expanded their algorithm to $\text{LTL}_\text{f}$.

For temporal logics, there are a few works that consider the OCC problem. Notably, Ehlers et al. [78] proposed a learning algorithm for a fragment of LTL that permits a representation known as universally very-weak automata (UVWs). However, since their algorithm relies on UVWs, which have strictly less expressive power than LTL, it cannot be extended to full LTL. Further, there are works on learning LTL [56] and STL [126] formulas from trajectories of high-dimensional systems. These works based their learning on the assumption that the underlying system optimizes some cost functions. Our method, in contrast, is based on the natural notion of language minimality to find tight descriptions without any assumptions on the system. There are some other works that consider the OCC problem for logics similar to temporal logic [227, 226, 206].

A problem similar to our OCC problem is studied in the context of inverse reinforcement learning (IRL) to learn temporal rewards for RL agents from (positive) demonstrations. For instance, Kasenberg et al. [130] learn concise LTL formulas that can distinguish between the provided demonstrations from random executions of the system. To generate the random executions, they relied on a Markov Decision Process (MDP) implementation of the underlying system. Our regularizers, in contrast, assume the underlying system to be a black-box and need no access to its internal mechanisms. Vazquez-Chanlatte et al. [214] also learn LTL-like formulas from demonstrations. Their search required a pre-computation of the lattice of formulas induced by the subset order, which can be a bottleneck for scaling to full LTL. Recently, Hasanbeig et al. [113] devised an algorithm to infer automaton for describing high-level objectives of RL agents. Unlike ours, their algorithm relied on user-defined hyper-parameters to regulate the degree of generalization of the inferred automaton.

## 6.1 Learning DFA from Positive Examples

In this section, we formally introduce the OCC problem for DFAs and our symbolic algorithm for learning DFAs from positive examples.

We first describe the learning setting. The OCC problem relies on a set of positive examples, which we represent using a finite set $P \subset (2^{\mathcal{P}})^*$ of finite traces. Additionally, the problem requires a bound $n$ to restrict the size of the learned DFA. The role of this size bound is two-fold: (i) it ensures that the learned DFA does not overfit $P$; and (ii) using a suitable bound, one can enforce the learned DFAs to be concise and, thus, interpretable.

Finally, we define a DFA $\mathcal{A}$ to be an $n$-description of $P$ if $P \subseteq L(\mathcal{A})$ and $|\mathcal{A}| \leq n$. When $P$ is clear from the context, we simply say $\mathcal{A}$ is an $n$-description.

We can now state the OCC problem for DFAs:

**Problem 6** (OCC problem for DFAs)**.** *Given a set $P$ of positive traces and a size bound $n$, learn a DFA $\mathcal{A}$ such that:*

1. *$\mathcal{A}$ is an $n$-description; and*
2. *for every DFA $\mathcal{A}'$ that is an $n$-description, $L(\mathcal{A}') \not\subset L(\mathcal{A})$.*

Intuitively, the above problem asks to search for a DFA that is an $n$-description and has minimal language. Note that several such DFAs can exist since the language inclusion order is a partial order on the languages of DFA. We, here, are interested in learning only one such DFA, leaving the problem of learning all such DFAs as interesting future work.

### 6.1.1 The Symbolic Algorithm

We now present our algorithm for solving Problem 6. Its underlying idea is to reduce the search for an appropriate DFA to a series of satisfiability checks of propositional formulas. Each satisfiable propositional formula enables us to construct a guess, or a so-called *hypothesis* DFA $\mathcal{A}$. In each step, using the hypothesis $\mathcal{A}$, we construct a propositional formula $\Psi^{\mathcal{A}}$ to search for the next hypothesis $\mathcal{A}'$ with a language smaller (in the inclusion order) than the current one. The properties of the propositional formula $\Psi^{\mathcal{A}}$ we construct are:

1. $\Psi^{\mathcal{A}}$ is satisfiable if and only if there exists a DFA $\mathcal{A}'$ that is an $n$-description and $L(\mathcal{A}') \subset L(\mathcal{A})$; and
2. based on a model $v$ of $\Psi^{\mathcal{A}}$, one can construct a prospective DFA $\mathcal{A}'$.

Based on the main ingredient $\Psi^{\mathcal{A}}$, we design our learning algorithm as sketched in Algorithm 10. Our algorithm initializes the hypothesis DFA $\mathcal{A}$ to be $\mathcal{A}^*$, the one-state DFA that accepts all traces in $(2^{\mathcal{P}})^*$. Observe that $\mathcal{A}^*$ is trivially an $n$-description, since $P \subset (2^{\mathcal{P}})^*$ and $|\mathcal{A}^*| = 1$. The algorithm then iteratively exploits $\Psi^{\mathcal{A}}$ to construct the next hypothesis DFAs until $\Psi^{\mathcal{A}}$ becomes unsatisfiable. Once this happens, we terminate and return the current hypothesis $\mathcal{A}$ as the solution. This algorithm is guaranteed to terminate and produce an optimal result. We formally state the guarantees later in this section after diving into the details of the main ingredient $\Psi^{\mathcal{A}}$.

To achieve its aforementioned properties, we define $\Psi^{\mathcal{A}}$ as follows:

$$\Psi^{\mathcal{A}} := \Psi^{\mathtt{DFA}} \wedge \Psi^{P} \wedge \Psi^{\subseteq \mathcal{A}} \wedge \Psi^{\not\supseteq \mathcal{A}} \tag{6.1}$$

The first conjunct $\Psi^{\mathtt{DFA}}$ ensures that the propositional variables we will use encode a valid DFA $\mathcal{A}'$. The second conjunct $\Psi^{P}$ ensures that $\mathcal{A}'$ accepts all positive traces. The third conjunct $\Psi^{\subseteq \mathcal{A}}$ ensures that $L(\mathcal{A}')$ is a subset of $L(\mathcal{A})$. The final conjunct $\Psi^{\not\supseteq \mathcal{A}}$ ensures that $L(\mathcal{A}')$ is not a superset of $L(\mathcal{A})$. Together, conjuncts $\Psi^{\subseteq \mathcal{A}}$ and $\Psi^{\not\supseteq \mathcal{A}}$ ensure that $L(\mathcal{A}')$ is a proper subset of $L(\mathcal{A})$. In what follows, we detail the construction of each conjunct.

**Algorithm 10** Symbolic Algorithm for Learning DFA

**Input**: Positive traces $P$, bound $n$

1: $\mathcal{A} \leftarrow \mathcal{A}^*$, $\Psi^{\mathcal{A}} := \Psi^{\texttt{DFA}} \wedge \Psi^{P}$
2: **while** $\Psi^{\mathcal{A}}$ is satisfiable (with model $v$) **do**
3: $\quad \mathcal{A} \leftarrow$ DFA constructed from $v$
4: $\quad \Psi^{\mathcal{A}} := \Psi^{\texttt{DFA}} \wedge \Psi^{P} \wedge \Psi^{\subseteq \mathcal{A}} \wedge \Psi^{\not\supseteq \mathcal{A}}$
5: **end while**
6: **return** $\mathcal{A}$

To encode the hypothesis DFA $\mathcal{A}' = (Q', 2^{\mathcal{P}}, \delta', q'_I, F')$ symbolically, following Heule et al. [116], we rely on the propositional variables: (i) $d_{p,a,q}$ where $p, q \in \{1, \ldots, n\}$ and $a \in 2^{\mathcal{P}}$; and (ii) $f_q$ where $q \in \{1, \ldots, n\}$. The variables $d_{p,a,q}$ and $f_q$ encode the transition function $\delta'$ and the final states $F'$, respectively, of $\mathcal{A}'$. Mathematically speaking, if $d_{p,a,q}$ is set to true, then $\delta'(p,a) = q$ and if $f_q$ is set to true, then $q \in F'$. To streamline notation, we here identify the states $Q'$ using the set $\{1, \ldots, n\}$ and the initial state $q'_I$ using the numeral 1.

Now, to ensure that the introduced variables encode a valid DFA, $\Psi^{\texttt{DFA}}$ asserts the following constraint:

$$\bigwedge_{1 \leq p \leq n} \bigwedge_{a \in 2^{\mathcal{P}}} \Big[ \bigvee_{1 \leq q \leq n} d_{p,a,q} \wedge \bigwedge_{1 \leq q \neq q' \leq n} \big[ \neg d_{p,a,q} \vee \neg d_{p,a,q'} \big] \Big] \tag{6.2}$$

The above constraint simply ensures that $\mathcal{A}'$ has a deterministic transition function.

Based on a model $v$ of the variables $d_{p,a,q}$ and $f_q$, we can simply construct $\mathcal{A}'$. We set $\delta'(p,a)$ to be the unique state $q$ for which $v(d_{p,a,q}) = 1$ and $q \in F'$ if $v(f_q) = 1$.

Next, to construct conjunct $\Psi^{P}$, we introduce variables $x_{u,q}$ where $u \in \mathit{Pref}(P)$ and $q \in \{1, \ldots, n\}$, which track the run of $\mathcal{A}'$ on all traces in $\mathit{Pref}(P) = \{u \in (2^{\mathcal{P}})^* \mid uv \in P \text{ for some } v \in (2^{\mathcal{P}})^*\}$, the set of prefixes of all traces in $P$. Precisely, if $x_{u,q}$ is set to true, then there is a run of $\mathcal{A}'$ on $u$ ending in the state $q$, that is, $\mathcal{A}' \colon q'_I \xrightarrow{u} q$; we use the shorthand $\mathcal{A} \colon q_1 \xrightarrow{u} q_2$ to express that there is a run of $\mathcal{A}$ on $u$ from $q_1$ to $q_2$.

Using the introduced variables, $\Psi^{P}$ ensures that the traces in $P$ are accepted by imposing the following constraints:

$$x_{\varepsilon,1} \wedge \bigwedge_{2 \leq q \leq n} \neg x_{\varepsilon,q} \tag{6.3}$$

$$\bigwedge_{ua \in \mathit{Pref}(P)} \bigwedge_{1 \leq p,q \leq n} [x_{u,p} \wedge d_{p,a,q}] \rightarrow x_{ua,q} \tag{6.4}$$

$$\bigwedge_{u \in P} \bigwedge_{1 \leq q \leq n} x_{u,q} \rightarrow f_q \tag{6.5}$$

Formula 6.3 ensures that any run of $\mathcal{A}'$ must start in the initial state $q'_I$ (which we identify as 1). Formula 6.4 ensures that the runs of $\mathcal{A}'$ must adhere to the transition function. Finally, Formula 6.5 ensures that the runs of $\mathcal{A}'$ on every positive trace $u \in P$ ends in a final state and, hence, is accepted.

For the third conjunct $\Psi^{\subseteq \mathcal{A}}$, we must track the synchronized runs of the current hypothesis $\mathcal{A}$ and the next hypothesis $\mathcal{A}'$ to compare their behavior on all traces in $(2^{\mathcal{P}})^*$. To this end,

we introduce auxiliary variables, $y^{\mathcal{A}}_{q,q'}$ where $q, q' \in \{1, \cdots, n\}$. Precisely, $y^{\mathcal{A}}_{q,q'}$ is set to true, if there exists a trace $u \in (2^{\mathcal{P}})^*$ such that there are runs $\mathcal{A}\colon q_I \xrightarrow{u} q$ and $\mathcal{A}'\colon q'_I \xrightarrow{u} q'$.

To ensure $L(\mathcal{A}') \subseteq L(\mathcal{A})$, $\Psi^{\subseteq\mathcal{A}}$ imposes the following constraints:

$$y^{\mathcal{A}}_{1,1} \tag{6.6}$$

$$\bigwedge_{1\le p',q'\le n} \bigwedge_{q=\delta(p,a)} \bigwedge_{a\in 2^{\mathcal{P}}} \left[ \left[ y^{\mathcal{A}}_{p,p'} \wedge d_{p',a,q'} \right] \rightarrow y^{\mathcal{A}}_{q,q'} \right] \tag{6.7}$$

$$\bigwedge_{1\le p'\le n} \bigwedge_{p\notin F} \left[ y^{\mathcal{A}}_{p,p'} \rightarrow \neg f_{p'} \right] \tag{6.8}$$

Formula 6.6 ensures that any synchronized runs of $\mathcal{A}$ and $\mathcal{A}'$ must start in the respective initial states. Formula 6.7 ensures that the synchronized runs must adhere to the respective transition functions of the DFAs. Formula 6.8 ensures that if a synchronized run ends in a non-final state in $\mathcal{A}$, it must also end in a non-final state in $\mathcal{A}'$, hence forcing $L(\mathcal{A}') \subseteq L(\mathcal{A})$.

For constructing the final conjunct $\Psi^{\not\supseteq\mathcal{A}}$, the variables we use rely on the following result:

**Lemma 7.** *Let $\mathcal{A}$, $\mathcal{A}'$ be DFAs such that $|\mathcal{A}| = |\mathcal{A}'| = n$ and $L(\mathcal{A}') \subset L(\mathcal{A})$. Also, let $K = n^2$. Then, there exists a trace $u \in (2^{\mathcal{P}})^*$ such that $|u| \le K$ and $u \in L(\mathcal{A}) \setminus L(\mathcal{A}')$.*

This result provides an upper bound to the length of a trace that can distinguish between DFAs $\mathcal{A}$ and $\mathcal{A}'$. The proof of the above lemma relies on a simple pumping argument on the cross product $\mathcal{A} \times \mathcal{A}'$ of size at most $K = n^2$.

Based on this result, we introduce variables $z_{i,q,q'}$ where $i \in \{1, \ldots, n^2\}$ and $q, q' \in \{1, \ldots, n\}$ to track the synchronized run of $\mathcal{A}$ and $\mathcal{A}'$ on a trace of length at most $K = n^2$. Precisely, if $z_{i,q,q'}$ is set to true, then there exists a trace $u$ of length $i$ with the runs $\mathcal{A}\colon q_I \xrightarrow{u} q$ and $\mathcal{A}' : q'_I \xrightarrow{u} q'$.

Now, $\Psi^{\not\supseteq\mathcal{A}}$ imposes the following constraints:

$$z_{0,1,1} \tag{6.9}$$

$$\bigwedge_{1\le i\le n^2} \left[ \bigvee_{1\le q,q'\le n} z_{i,q,q'} \wedge \left[ \bigwedge_{\substack{1\le p\neq q\le n \\ 1\le p'\neq q'\le n}} \neg z_{i,p,p'} \vee \neg z_{i,q,q'} \right] \right] \tag{6.10}$$

$$\bigwedge_{\substack{1\le p,q\le n \\ 1\le p',q'\le n}} \left[ \left[ z_{i,p,p'} \wedge z_{i+1,q,q'} \right] \rightarrow \bigvee_{\substack{a\in 2^{\mathcal{P}} \text{ where} \\ q=\delta(p,a)}} d_{p',a,q'} \right] \tag{6.11}$$

$$\bigvee_{1\le i\le n^2} \bigvee_{\substack{q\in F \\ 1\le q'\le n}} \left[ z_{i,q,q'} \wedge \neg f_{q'} \right] \tag{6.12}$$

Formula 6.9 ensures that there exists a trace of length 0, that is, $\varepsilon$, on which the synchronized run ends in the respective initial states of the DFAs $\mathcal{A}$ and $\mathcal{A}'$. Formula 6.10 ensures that the synchronized run ends in unique states in the DFAs. Formula 6.11 ensures that the synchronized run adheres to the respective transition functions of the DFAs. Finally, Formula 6.12 ensures that there is a trace of length $\le n^2$ on which the synchronized run ends in a final state in $\mathcal{A}$ but not in $\mathcal{A}'$, ensuring $L(\mathcal{A}) \not\subseteq L(\mathcal{A}')$.

We now assert the correctness of the propositional formula $\Psi^{\mathcal{A}}$ that we constructed above:

**Theorem 11.** *Let $\Psi^{\mathcal{A}}$ be as defined above. Then, we have the following:*

1. *If $\Psi^{\mathcal{A}}$ is satisfiable, then there exists a DFA $\mathcal{A}'$ that is an $n$-description and $L(\mathcal{A}') \subset L(\mathcal{A})$.*
2. *If there exists a DFA $\mathcal{A}'$ that is an $n$-description and $L(\mathcal{A}') \subset L(\mathcal{A})$, then $\Psi^{\mathcal{A}}$ is satisfiable.*

To prove the above theorem, we propose intermediate claims, all of which we prove first. For the proofs, we assume $\mathcal{A} = (Q, 2^{\mathcal{P}}, \delta, q_I, F)$ to be the current hypothesis, $v$ to be a model of $\Psi^{\mathcal{A}}$, and $\mathcal{A}' = (Q', 2^{\mathcal{P}}, \delta', q'_I, F')$ to be the DFA constructed from the model $v$ of $\Psi^{\mathcal{A}}$.

**Claim 1.** *For all $u \in Pref(P)$, $\mathcal{A}' : q'_I \xrightarrow{u} q$ implies $v(x_{u,q}) = 1$.*

*Proof.* We prove the claim using induction on the length $|u|$ of the trace $u$.

*Base case:* Let $u = \varepsilon$. Based on the definition of runs, $\mathcal{A}' : q'_I \xrightarrow{\varepsilon} q$ implies $q = q'_I$. Also, using Formula 6.3, we have $v(x_{\varepsilon,q}) = 1$ if and only if $q = q'_I$ (note $q'_I$ is indicated using numeral 1). Combining these two facts proves the claim for the base case.

*Induction step:* As induction hypothesis, let $\mathcal{A}' : {q_I}' \xrightarrow{u} q$ implies $v(x_{u,q}) = 1$ for all traces $u \in Pref(P)$ of length $\leq k$. Now, consider the run $\mathcal{A}' : q'_I \xrightarrow{u} q \xrightarrow{a} q'$ for some $ua \in Pref(P)$. For this run, based on the induction hypothesis and the construction of $\mathcal{A}'$, we have $v(x_{u,q}) = 1$ and $v(d_{p,a,q}) = 1$. Now, using Formula 6.4, $v(x_{u,q}) = 1$ and $v(d_{p,a,q}) = 1$ implies $v(x_{ua,q}) = 1$, thus, proving the claim.

□

**Claim 2.** *For all $u \in (2^{\mathcal{P}})^*$, $\mathcal{A} : q_I \xrightarrow{u} q$ and $\mathcal{A}' : q'_I \xrightarrow{u} q'$ imply $v(y^{\mathcal{A}}_{q,q'}) = 1$.*

*Proof.* We prove this using induction of the length $|u|$ of the trace $u$.

*Base case:* Let $u = \varepsilon$. Based on the definition of runs, $\mathcal{A} : q_I \xrightarrow{\varepsilon} q$, $\mathcal{A}' : q'_I \xrightarrow{\varepsilon} q'$ implies $q = q_I$ and $q' = q'_I$. Also, using Formula 6.6, $q = q_I$ and $q' = q'_I$ imply $v(y^{\mathcal{A}}_{q,q'}) = 1$ (note $q_I$ and $q'_I$ are both indicated using numeral 1). Combining these two facts proves the claim for the base case.

*Induction step:* As induction hypothesis, let $\mathcal{A} : q_I \xrightarrow{u} q$ and $\mathcal{A}' : q'_I \xrightarrow{u} q'$ imply $v(y^{\mathcal{A}}_{q,q'}) = 1$ for all traces $u \in (2^{\mathcal{P}})^*$ of length $\leq k$. Now, consider the runs $\mathcal{A} : q_I \xrightarrow{u} p \xrightarrow{a} q$ and $\mathcal{A}' : {q_I}' \xrightarrow{u} p' \xrightarrow{a} q'$ for some trace $ua \in (2^{\mathcal{P}})^*$. For these runs, based on the induction hypothesis and the construction of $\mathcal{A}'$, we have $v(y^{\mathcal{A}}_{p,p'}) = 1$ and $v(d_{p',a,q'}) = 1$. Now, using Formula 6.7, we can say that $v(y^{\mathcal{A}}_{p,p'}) = 1$ and $v(d_{p',a,q'}) = 1$ imply $v(y^{\mathcal{A}}_{q,q'}) = 1$ (where $q = \delta(p,a)$), thus, proving the claim.

□ □

**Claim 3.** *$v(z_{i,q,q'}) = 1$ implies there exists $u \in (2^{\mathcal{P}})^i$ with runs $\mathcal{A} : q_I \xrightarrow{u} q$ and $\mathcal{A}' : q'_I \xrightarrow{u} q'$.*

*Proof.* We prove this using induction on the parameter $i$.

*Base case:* Let $i = 0$. Based on the Formulas 6.9 and 6.10, $v(z_{0,q,q'}) = 1$ implies $q = q_I$ and $q = q'_I$. Now, there always exists a trace of length 0, that is, $u = \varepsilon$, for which $\mathcal{A} : q_I \xrightarrow{\varepsilon} q$ and $\mathcal{A}' : q'_I \xrightarrow{\varepsilon} q'$ proving the claim for the base case.

*Induction step:* As induction hypothesis, let $v(z_{k,p,p'}) = 1$ and thus, let $u$ be a trace of length $k$ such that $\mathcal{A} : q_I \xrightarrow{u} p$ and $\mathcal{A}' : q'_I \xrightarrow{u} p'$. Now, assume $v(z_{k+1,q,q'}) = 1$. Based on Formula 6.11, for some $a \in 2^{\mathcal{P}}$ such that $q = \delta(p,a)$, $v(d_{p',a,q'}) = 1$. Thus, on the trace $ua$, there are runs $\mathcal{A} : q_I \xrightarrow{u} p \xrightarrow{a} q$ and $\mathcal{A}' : q'_I \xrightarrow{u} p' \xrightarrow{a} q'$, proving the claim.

□

We are now ready to prove Theorem 11, that is, the correctness of the encoding $\Psi^{\mathcal{A}}$.

*Proof of Theorem 11.* For the first statement in Theorem 11, consider that $\Psi^{\mathcal{A}}$ is satisfiable with a model $v$ and $\mathcal{A}'$ is the DFA constructed using the model $v$. First, DFA constructed with model $v$ trivially has size $|\mathcal{A}'| \leq n$. Second, using Claim 1, $\mathcal{A}' : q'_I \xrightarrow{u} q$ implies $v(x_{u,q}) = 1$ for all $u \in P$. Now, based on Formula 6.5, $v(x_{u,q}) = 1$ implies $v(f_q) = 1$ for all $u \in P$. As a result, for each $u \in P$, its run $\mathcal{A}' : q'_I \xrightarrow{u} q$ must end in a final state $q \in F'$ in $\mathcal{A}'$. Thus, $\mathcal{A}'$ accepts all positive traces and, hence, is an $n$-description.

Next, using Claim 2, $\mathcal{A} : q_I \xrightarrow{u} q$ and $\mathcal{A}' : q'_I \xrightarrow{u} q'$ imply $v(y^{\mathcal{A}}_{q,q'}) = 1$ for all traces $u \in (2^{\mathcal{P}})^*$. Thus, based on Formula 6.8, if $q \notin F$, then $q' \notin F$, implying $L(\mathcal{A}') \subseteq L(\mathcal{A})$.

Finally, using Claim 3, $v(z_{i,q,q'}) = 1$ implies that there exists $u \in (2^{\mathcal{P}})^i$ with runs $\mathcal{A} : q_I \xrightarrow{u} q$ and $\mathcal{A}' : q'_I \xrightarrow{u} q'$. Now, based on Formula 6.12, there exists some $i \leq n^2$, $q \in F$ and $q'$ in $\mathcal{A}'$ such that $v(z_{i,q,q'}) = 1$ and $v(f_{q'}) = 0$. Combining this fact with Claim 3, we deduce that there exists $u \in (2^{\mathcal{P}})^*$ with length $\leq n^2$ with run $\mathcal{A} : q_I \xrightarrow{u} q$ ending in final state $q \in F$ and run $\mathcal{A}' : q'_I \xrightarrow{u} q'$ not ending in a final state $q' \in F'$. This shows that $L(\mathcal{A}) \neq L(\mathcal{A}')$. We thus prove the first statement, which claims $\mathcal{A}$ to be an $n$-description and $L(\mathcal{A}') \subset L(\mathcal{A})$.

For the second statement in Theorem 11, based on a suitable DFA $\mathcal{A}'$, we construct an assignment $v$ for all the introduced propositional variables. First, we set $v(d_{p,a,q}) = 1$ if $\delta'(p,a) = q$ and $v(f_q) = 1$ if $q \in F'$. Since $\delta'$ is a deterministic function, $v$ satisfies the Formula 6.2. Similarly, we set $v(x_{u,q}) = 1$ if $\mathcal{A}' : q'_I \xrightarrow{u} q$ for some $u \in Pref(P)$. It is a simple exercise to check that $v$ satisfies Formulas 6.3 to 6.5. Next, we set $v(y^{\mathcal{A}}_{q,q'}) = 1$ if there are runs $\mathcal{A} : q_I \xrightarrow{u} q$ and $\mathcal{A}' : q'_I \xrightarrow{u} q'$ on some trace $u \in (2^{\mathcal{P}})^*$. Algorithmically, we set $v(y^{\mathcal{A}}_{q,q'}) = 1$ if states $q$ and $q'$ can be reached in the synchronized run of $\mathcal{A}$ and $\mathcal{A}'$ on some trace (which is typically computed using a breadth-first search on the product DFA). It is easy to see that such an assignment $v$ satisfies Formulas 6.6 to 6.8. Finally, we set assignment to $z_{i,q,q'}$ exploiting a trace $u$ which permits runs $\mathcal{A} : q_I \xrightarrow{u} q$ and $\mathcal{A}' : q'_I \xrightarrow{u} q'$, where $q$ is in $F$ but $q'$ not in $F'$. In particular, we set $v(z_{i,q,q'}) = 1$ for $i = |u'|$ and $\mathcal{A} : q_I \xrightarrow{u'} q$ and $\mathcal{A}' : q'_I \xrightarrow{u'} q'$ for all prefixes $u'$ of $u$. Such an assignment encodes a synchronized run of the DFAs $\mathcal{A}$ and $\mathcal{A}'$ on trace $u$ that ends in a final state in $\mathcal{A}$, but not in $\mathcal{A}'$. Thus, $v$ satisfies Formulas 6.9 to 6.12. □

**Theorem 12.** *Given positive traces $P$ and a size bound $n$, Algorithm 10 always terminates and learns a DFA $\mathcal{A}$ that is an $n$-description and for every DFA $\mathcal{A}'$ that is an $n$-description, $L(\mathcal{A}') \not\subset L(\mathcal{A})$.*

*Proof.* For termination, observe there are finitely many $n$-descriptions for a given bound $n$, and in each iteration, Algorithm 10 finds a new $n$-description. Thus, after a finite number of iterations, the algorithm will have exhaustively searched through all possible $n$-descriptions.

For correctness, first observe that, by design, the algorithm terminates when $\Psi^{\mathcal{A}}$ is unsatisfiable. Now, based on the properties of $\Psi^{\mathcal{A}}$ established in Theorem 11, if $\Psi^{\mathcal{A}}$ is unsatisfiable for some DFA $\mathcal{A}$, then there are no $n$-description DFA $\mathcal{A}'$ for which $L(\mathcal{A}') \subset L(\mathcal{A})$, thus, proving the correctness of the algorithm. □

## 6.2 Learning LTL formulas from Positive Examples

We now switch our focus to algorithms for learning $\text{LTL}_\text{f}$ formulas. The OCC problem for $\text{LTL}_\text{f}$ formulas, similar to Problem 6, relies upon a set of positive traces $P \subset (2^{\mathcal{P}})^*$ and a size upper bound $n$.

Moreover, an $\text{LTL}_\text{f}$ formula $\varphi$ is an $n$-description of $P$ if, for all $u \in P$, $u \models_\text{f} \varphi$, and $|\varphi| \leq n$. Again, when $P$ is clear from the context, we simply use $n$-description . Also, in this section, an $n$-description refers only to an $\text{LTL}_\text{f}$ formula.

We state the OCC problem for $\text{LTL}_\text{f}$ formulas as follows:

**Problem 7** ($\text{LTL}_\text{f}$ formulas)**.** *Given a set $P$ of positive traces and a size bound $n$, learn an $\text{LTL}_f$ formula $\varphi$ such that:*

1. *$\varphi$ is an $n$-description; and*
2. *for every $\text{LTL}_f$ formula $\varphi'$ that is an $n$-description, $\varphi' \not\rightarrow \varphi$ or $\varphi \rightarrow \varphi'$.*

Intuitively, the above problem searches for an $\text{LTL}_\text{f}$ formula $\varphi$ that is an $n$-description and holds on a minimal set of traces. The combination $\varphi' \not\rightarrow \varphi$ or $\varphi \rightarrow \varphi'$ expresses that $L(\varphi') \not\subset L(\varphi)$. Once again, like Problem 6, there can be several solution $\text{LTL}_\text{f}$ formulas for the above problem, but we are interested in learning exactly one.

### 6.2.1 The Semi-Symbolic Algorithm

Our *semi*-symbolic, in contrast to the symbolic algorithm in Algorithm 10, does not solely depend on the current hypothesis, an $\text{LTL}_\text{f}$ formula $\varphi$. In addition, it relies on a set $N$ of negative examples accumulated during the algorithm. Concretely, $\Phi^{\varphi,N}$ has the properties that:

1. $\Phi^{\varphi,N}$ is satisfiable if and only if there exists an $\text{LTL}_\text{f}$ formula $\varphi'$ that is an $n$-description, does not hold on any $u \in N$, and $\varphi \not\rightarrow \varphi'$; and
2. based on a model $v$ of $\Phi^{\varphi,N}$, one can construct a prospective $\text{LTL}_\text{f}$ formula $\varphi'$.

**Algorithm 11** Semi-symbolic Algorithm for learning $\text{LTL}_\text{f}$ formula

**Input**: Positive traces $P$, bound $n$

$N \leftarrow \emptyset$
$\varphi \leftarrow \mathit{true}$, $\Phi^{\varphi,N} := \Phi^{\text{LTL}} \wedge \Phi^{P}$
**while** $\Phi^{\varphi,N}$ is satisfiable (with model $v$) **do**
  $\varphi' \leftarrow$ LTL formula constructed from $v$
  **if** $\varphi' \rightarrow \varphi$ **then**
    Update $\varphi$ to $\varphi'$
  **else**
    Add $u$ to $N$, where $u \models \varphi' \wedge \neg\varphi$
  **end if**
  $\Phi^{\varphi,N} := \Phi^{\text{LTL}} \wedge \Phi^{P} \wedge \Phi^{N} \wedge \Phi^{\not\leftarrow\varphi}$
**end while**
**return** $\varphi$

The semi-symbolic algorithm, outlined in Algorithm 11, follows a paradigm similar to the one illustrated in Algorithm 10. However, unlike the previous algorithm, the next hypothesis $\varphi'$ derived from a model of $\Phi^{\varphi,N}$ may not always satisfy the relation $\varphi' \rightarrow \varphi$. In other words, the condition $L(\varphi') \subseteq L(\varphi)$ may not hold.

In such cases, we generate a trace, denoted as $u$, that satisfies $\varphi'$ but not $\varphi$, specifically $u \in L(\varphi') \setminus L(\varphi)$. We then add $u$ to the set $N$ of negative examples. This aids us in eliminating $\varphi'$ from the search space in subsequent iterations. We generate negative traces by constructing DFAs from the $\text{LTL}_\text{f}$ formulas [231] and subsequently performing a breadth-first search over them.

It is also possible that the current hypothesis $\varphi'$ satisfies the relation $\varphi' \rightarrow \varphi$, that is, $L(\varphi') \subseteq L(\varphi)$ holds. In this case, we simply update our current hypothesis as $\varphi'$ and continue the algorithm until $\Phi^{\varphi,N}$ is unsatisfiable.

We now focus on the construction of $\Phi^{\varphi,N}$, which admits various differences from that of $\Psi^{\mathcal{A}}$. It is defined as follows:

$$\Phi^{\varphi,N} := \Phi^{\text{LTL}} \wedge \Phi^{P} \wedge \Phi^{N} \wedge \Phi^{\not\leftarrow\varphi}. \tag{6.13}$$

The first conjunct $\Phi^{\text{LTL}}$ ensures that the new hypothesis $\varphi'$ is a valid $\text{LTL}_\text{f}$ formula. The second conjunct $\Phi^{P}$ ensures that $\varphi'$ holds on all positive traces, while the third conjunct $\Phi^{N}$ ensures that it does not hold on any negative traces. The final conjunct $\Phi^{\not\leftarrow\varphi}$ ensures that $\varphi \not\rightarrow \varphi'$; that is, $L(\varphi) \not\subseteq L(\varphi')$.

Similar to the previous chapters (Chapters 4 and 5), all of our conjuncts rely on the *syntax DAG* as canonical syntactic representation for $\text{LTL}_\text{f}$. In fact, to encode a valid syntax DAG for hypothesis $\varphi'$, $\Phi^{\text{LTL}}$ relies on the same set of propositional variables $x_{i,\lambda}$, $l_{i,j}$ and $r_{i,j}$ where $i \in \{1,\ldots,n\}$, $j \in \{i+1,\ldots,n\}$ and $\lambda \in \Lambda$ that were used in the previous chapters. Also, the constraints imposed on the variables remain the same as in Formulas 4.4 to 4.7.

To define $\Phi^{P}$ and $\Phi^{N}$, we rely on the propositional formula $\Phi^{u}$ for each trace $u$ in $P \cup N$, which tracks the semantics of $\varphi'$ on $u$. Similar to Section 4.1.2, these formulas are constructed using variables $y^{u}_{i,t}$ where $i \in \{1,\ldots,n\}$ and $t \in \{0,\ldots,|u|-1\}$. The variable $y^{u}_{i,t}$ indicates

whether $\varphi'[i]$ holds on $u$ at timepoint $t$.

To ensure the intended meaning of variables $y^u_{i,t}$, $\Phi^u$ is a conjunction of Formulas 4.8 to 4.12 from Chapter 4. We then have $\Phi^P := \bigwedge_{u \in P} \Phi^u \wedge y^u_{1,0}$ to ensure that $\varphi'$ holds on positive traces and $\Phi^N := \bigwedge_{u \in N} \Phi^u \wedge \neg y^u_{1,0}$ to ensure $\varphi'$ does not hold on negative traces.

Next, to construct $\Phi^{\not\leftarrow \varphi}$, we symbolically encode a trace $w$ that distinguishes formulas $\varphi$ and $\varphi'$. We bound the length of the symbolic trace by a *time horizon* $K = 2^{2^{n+1}}$. The choice of $K$ is derived from Lemma 7 and the fact that the size of the equivalent DFA for an $\mathrm{LTL_f}$ formula can be at most doubly exponential [101].

Our encoding of a symbolic trace $w$ relies on variables $p_{t,a}$ where $t \in \{0, \ldots, K-1\}$ and $a \in 2^{\mathcal{P}} \cup \{\varepsilon\}$. If $p_{t,a}$ is set to true, then $w[t] = a$. To ensure that the variables $p_{t,a}$ encode their desired meaning, we generate a formula $\Phi^{trace}$ that consists of the following constraint:

$$\bigwedge_{0 \leq t < K} \Big[ \bigvee_{a \in 2^{\mathcal{P}} \cup \{\varepsilon\}} p_{t,a} \wedge \bigwedge_{a \neq a' \in 2^{\mathcal{P}} \cup \{\varepsilon\}} \big[\neg p_{t,a} \vee \neg p_{t,a'}\big] \Big] \tag{6.14}$$

The above constraint ensures that, in the symbolic trace $w$, each timepoint $0 \leq t < K$ has a unique symbol from $2^{\mathcal{P}} \cup \{\varepsilon\}$.

Further, to track whether $\varphi$ and $\varphi'$ hold on $w$, we have variables $z^{w,\varphi}_{i,t}$ and $z^{w,\varphi'}_{i,t}$ where $i \in \{1, \ldots, n\}$, and $t \in \{0, \ldots, K-1\}$. These variables are similar to $y^u_{i,t}$, in the sense that, $z^{w,\varphi}_{i,t}$ (respectively, $z^{w,\varphi'}_{i,t}$) is set to true, if $\varphi[i]$ (respectively, $\varphi'[i]$) holds on the symbolic trace $w$ at timepoint $t$.

To ensure desired meaning of $z^{w,\varphi'}_{i,t}$, we construct $\Phi^{w,\varphi'}$ similar to that of $\Phi^u$. We list these constraints in $\Phi^{w,\varphi'}$ as follows:

$$\bigwedge_{1 \leq i \leq n} \bigwedge_{a \in 2^{\mathcal{P}}} x_{i,a} \rightarrow \Big[ \bigwedge_{0 \leq t < K} z^{w,\varphi'}_{i,t} \leftrightarrow p_{t,a} \Big] \tag{6.15}$$

$$\bigwedge_{\substack{1 \leq i \leq n \\ i < j, j' \leq n}} x_{i,\vee} \wedge l_{i,j} \wedge r_{i,j'} \rightarrow \Big[ \bigwedge_{0 \leq t < K} \Big[ z^{w,\varphi'}_{i,t} \leftrightarrow z^{w,\varphi'}_{j,t} \vee z^{w,\varphi'}_{j',t} \Big] \Big] \tag{6.16}$$

$$\bigwedge_{\substack{1 \leq i \leq n \\ i < j \leq n}} x_{i,\mathsf{X}} \wedge l_{i,j} \rightarrow \Big[ \bigwedge_{0 \leq t < K-1} z^{w,\varphi'}_{i,t} \leftrightarrow z^{w,\varphi'}_{j,t+1} \Big] \wedge \neg z^{w,\varphi'}_{j,K-1} \tag{6.17}$$

$$\bigwedge_{\substack{1 \leq i \leq n \\ i < j, j' \leq n}} x_{i,\mathsf{U}} \wedge l_{i,j} \wedge r_{i,j'} \rightarrow \Big[ \bigwedge_{0 \leq t < K} \Big[ z^{w,\varphi'}_{i,t} \leftrightarrow \bigvee_{t \leq t' < K} \Big[ z^{w,\varphi'}_{j',t'} \wedge \bigwedge_{t \leq \tau < t'} z^{w,\varphi'}_{j,\tau} \Big] \Big] \Big] \tag{6.18}$$

Clearly, the above constraints are identical to the Formulas 4.8 to 4.12 on variables $y^u_{i,t}$.

We construct $\Phi^{w,\varphi}$ on variables $z^{w,\varphi}_{i,t}$ using constraints similar to those above, Formulas 6.15 to 6.18. The only difference is that, in contrast to $\varphi'$, the hypothesis $\varphi$ is already known. Thus, one can directly exploit the syntax DAG of $\varphi$ in $\Phi^{w,\varphi}$ instead of using a symbolic encoding.

Finally, we define $\Phi^{\not\leftarrow \varphi}$ as follows:

$$\Phi^{\not\leftarrow \varphi} := \Phi^{\mathtt{trace}} \wedge \big[\Phi^{w,\varphi} \wedge z^{w,\varphi}_{1,0}\big] \wedge \big[\Phi^{w,\varphi'} \wedge \neg z^{w,\varphi'}_{1,0}\big].$$

Intuitively, the above conjunction ensures that there exists a trace on which $\varphi$ holds and $\varphi'$ does not.

We now prove the correctness of the encoding $\Phi^{\varphi,N}$ described using the constraints above.

**Theorem 13.** *Let $\Phi^{\varphi,N}$ be as defined above. Then, we have the following:*

1. *If $\Phi^{\varphi,N}$ is satisfiable, then there exists an LTL formula $\varphi'$ that is an $n$-description, $\varphi'$ does not hold on $u \in N$ and $\varphi \not\rightarrow \varphi'$.*
2. *If there exists a LTL formula $\varphi'$ that is an $n$-description, $\varphi'$ does not hold on $u \in N$ and $\varphi \not\rightarrow \varphi'$, then $\Phi^{\varphi,N}$ is satisfiable.*

To prove this theorem, we rely on intermediate claims. In the claims, $v$ is a model of $\Phi^{\varphi,N}$, $\varphi'$ is the LTL formula constructed from $v$ and $\varphi$ is the current hypothesis LTL formula.

**Claim 4.** *For all $u \in P \cup N$, $v(y^u_{i,t}) = 1$ if and only if $u[t,|u|) \models \varphi'[i]$.*

The proof proceeds via structural induction over $\varphi'[i]$. We reuse the variables $y^u_{i,t}$ and the constraints $\Phi^u$ from Neider et al. [169] and, thus, the proof directly follows from the existing proof (it can be found in the appendix of the paper [170]).

**Claim 5.** *$v(z^{w,\varphi'}_{i,t}) = 1$ (resp. $v(z^{w,\varphi}_{i,t}) = 1$) if and only for a trace $w$, $w[t,K) \models \varphi'$ (resp. $w[t,K) \models \varphi \models \varphi$).*

The proof again proceeds via a structural induction on $\varphi'$ (similar to the previous one).

We are now ready to prove Theorem 13, i.e., the correctness of $\Omega^{N,D}$.

*Proof of Theorem 13.* For the first statement, consider that $\Phi^{\varphi,N}$ is satisfiable with a model $v$ and $\varphi'$ is the LTL formula constructed using the model $v$. First, using Claim 4, we have that $v(y^u_{1,t}) = 1$ if and only if $u[t,|u|) \models \varphi'$. Based on the construction of $\Phi^P$ and $\Phi^N$, we observe that $v(y^u_{1,0}) = 1$ for all traces $u \in P$ and $v(y^u_{1,0}) = 0$ for all traces $u \in N$. Thus, combining the two above observations, we conclude $u \models \varphi'$ for $u \in P$ and $u \not\models \varphi'$ for $u \in N$ and hence, $\varphi'$ is an $n$-description.

Next, using Claim 5 and the construction of $\Phi^{\not\rightarrow\varphi}$, we conclude that there exists a trace $w$ on which $\varphi$ holds and $\varphi'$ does not. Thus, in total, we obtain $\varphi'$ to be an $n$-description which does not hold on $u \in N$ and $\varphi \not\rightarrow \varphi'$

For the second statement, based on a suitable hypothesis $\varphi'$, we construct an assignment $v$ for all the introduced propositional variables. First, we set $v(x_{i,\lambda}) = 1$ if Node $i$ is labeled with operator $\lambda$ and $v(l_{i,j}) = 1$ (respectively, $v(r_{i,j}) = 1$) if the left (respectively, the right) child of Node $i$ is Node $j$. Since $\varphi'$ is a valid LTL formula, it is clear that the structural constraints will be satisfied by $v$. Similarly, we set $v(y^u_{i,t}) = 1$ if and only if $u[t,|u|) \models \varphi'[i]$ for $u \in P \cup N$ and $t \in [|w|]$. This satisfies the structural constraints, as shown in prior work [170]. Next, we set $v(z^{\varphi}_{w,t}) = 1$ and $v(z^{\varphi',n}_{w,t}) = 0$ for a trace $w$ for which $w \models \varphi$ and $w \not\models \varphi'$. Similar to the other semantic constraints, one can see that $v$ satisfies $\Phi^{\not\rightarrow\varphi}$. □

We now prove the overall termination and correctness of Algorithm 11.

**Theorem 14.** *Given positive traces $P$ and size bound $n$, Algorithm 11 terminates and learns an $LTL_f$ formula $\varphi$ that is an $n$-description and for every $LTL_f$ formulas $\varphi'$ that is an $n$-description, $\varphi' \not\rightarrow \varphi$ or $\varphi \rightarrow \varphi'$.*

*Proof.* For termination, first observe that Algorithm 11 there are only finitely many LTL formulas that are $n$-descriptions because of the size bound $n$. We now show that our algorithm (in the worst case) exhaustively searches through all $n$-descriptions.

To address this, let us assume that the algorithm identifies the same hypothesis in iterations $k$ and $l$ (say $k < l$) within its while loop. For clarity, let us denote the hypothesis produced in iteration $k$ as $\varphi_k$. We will now present our argument through a case analysis.

First, consider the scenario where the condition $\varphi' \rightarrow \varphi$ (refer to Line 6) holds in all iterations between $k$ and $l$. In this case, it follows that $\varphi_l \rightarrow \varphi_k$ and $\varphi_k \not\rightarrow \varphi_l$. Consequently, this implies $\varphi_l \neq \varphi_k$, thereby contradicting our initial assumption.

Now, consider the other case where, in at least one iteration between $k$ and $l$, the else condition $\varphi' \not\rightarrow \varphi$ (refer to Line 8) holds. In this case, a trace $u$ on which $\varphi_k$ holds is added to the set $N$ of negative traces. As $\varphi_l$ must not hold on this negative trace $u$, it once again leads to the conclusion that $\varphi_l \neq \varphi_k$, contradicting our initial assumption.

For the correctness of the learned LTL formula $\varphi$, we again rely on a contradiction. Let us assume the existence of an LTL formula $\overline{\varphi}$ that is an $n$-description and satisfies the relations $\overline{\varphi} \rightarrow \varphi$ and $\varphi \not\rightarrow \overline{\varphi}$, meaning $L(\overline{\varphi}) \subset L(\varphi)$.

It is crucial to note that the algorithm terminates when $\Phi^{\varphi,N}$ becomes unsatisfiable. Drawing on the properties of $\Phi^{\varphi,N}$, as outlined in Theorem 13, if $\Phi^{\varphi,N}$ is unsatisfiable for a given LTL formula $\varphi$ and set $N$ of negative traces, then any LTL formula $\varphi'$ that is an $n$-description must either hold in one of the traces in $N$ or satisfy $\varphi \rightarrow \varphi'$.

If $\varphi \rightarrow \overline{\varphi}$ holds, the assumption $\varphi \not\rightarrow \overline{\varphi}$ is contradicted. Alternatively, if $\overline{\varphi}$ holds in one of the negative traces, the assumption $\overline{\varphi} \rightarrow \varphi$ is contradicted. Thus, we establish the correctness of the learned formula $\varphi$. □

### 6.2.2 The Counterexample-guided Algorithm

We now design a *counterexample-guided* algorithm to solve Problem 7. In contrast to the symbolic (or semi-symbolic) algorithm, this algorithm does not guide the search based on propositional formulas built out of the hypothesis $\text{LTL}_\text{f}$ formula. Instead, this algorithm relies entirely on two sets: a set $N$ of negative traces and a set of discarded $\text{LTL}_\text{f}$ formulas $D$. Based on these two sets, we design a propositional formula $\Omega^{N,D}$ that has the properties that:

1. $\Omega^{N,D}$ is satisfiable if and only if there exists an $\text{LTL}_\text{f}$ formula $\varphi$ that is an $n$-description, does not hold on $w \in N$, and is not one of the formulas in $D$; and
2. based on a model $v$ of $\Omega^{N,D}$, one can construct such an $\text{LTL}_\text{f}$ formula $\varphi'$.

Being a counterexample-guided algorithm, the construction of the sets $N$ and $D$ forms the crux of the algorithm. In each iteration, these sets are updated based on the relation between the hypothesis $\varphi$ and the current guess $\varphi'$ derived from a model of $\Omega^{N,D}$. There are exactly three relevant cases, which we discuss briefly.

**Algorithm 12** CEG Algorithm for $\text{LTL}_\text{f}$ formulas

**Input**: Positive traces $P$, bound $n$

$N \leftarrow \emptyset$, $D \leftarrow \emptyset$
$\varphi \leftarrow \mathit{true}$, $\Omega^{N,D} := \Phi^{\text{LTL}} \wedge \Phi^{P}$
**while** $\Omega^{N,D}$ is satisfiable (with model $v$) **do**
  $\varphi' \leftarrow \varphi^{v}$
  **if** $\varphi' \leftrightarrow \varphi$ **then**
    Add $\varphi'$ to $D$
  **else**
    **if** $\varphi' \rightarrow \varphi$ **then**
      Add $u$ to $N$, where $u \models \neg\varphi \wedge \varphi'$
      $\varphi \leftarrow \varphi'$
    **else**
      Add $u$ to $N$, where $u \models \neg\varphi' \wedge \varphi$
    **end if**
  **end if**
  $\Omega^{N,D} := \Phi^{\text{LTL}} \wedge \Phi^{P} \wedge \Phi^{N} \wedge \Phi^{D}$
**end while**
**return** $\varphi$

- First, $\varphi' \leftrightarrow \varphi$, i.e., $\varphi'$ and $\varphi$ hold on the exact same set of traces. In this case, the algorithm discards $\varphi'$, due to its equivalence to $\varphi$, by adding it to $D$.
- Second, $\varphi' \rightarrow \varphi$ and $\varphi \not\rightarrow \varphi'$, i.e., $\varphi'$ holds on a proper subset of the set of traces on which $\varphi$ hold. In this case, our algorithm generates a trace that satisfies $\varphi$ and not $\varphi'$, which it adds to $N$ to eliminate $\varphi$.
- Third, $\varphi' \not\leftarrow \varphi$, i.e., $\varphi'$ does not hold on a subset of the set of traces on which $\varphi$ hold. In this case, our algorithm generates a trace $w$ that satisfies $\varphi'$ and not $\varphi$, which it adds to $N$ to eliminate $\varphi'$.

By handling the cases mentioned above, we obtain an algorithm (sketched in Algorithm 12) with guarantees (formalized in Theorem 14) exactly the same as the semi-symbolic algorithm in Section 6.2.1.

## 6.3 Experimental Evaluation

To evaluate the performance of our algorithms, we consider the following research questions in this section:

**RQ1**: What is the performance gain for learning DFAs compared to existing work?

**RQ2**: Which of the presented $\text{LTL}_\text{f}$ learning algorithms demonstrate better performance?

**RQ3**: Can our $\text{LTL}_\text{f}$ learning algorithms produce useful formulas?

We address the above research questions by implementing all the algorithms in Python 3[1]. For constraint solving, we rely on PySAT [125] for the DFA learning algorithm, and clingo [96][2] for the $\text{LTL}_\text{f}$ learning algorithms.

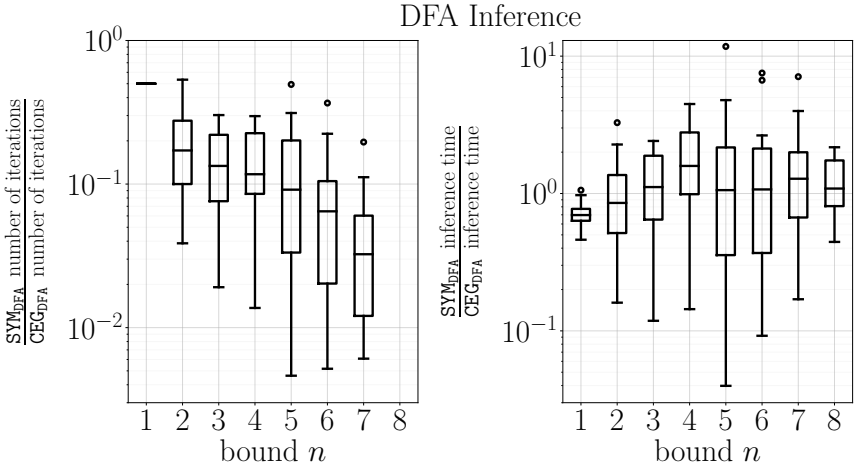



FIGURE 6.1: Comparison of $\text{SYM}_\text{DFA}$ and $\text{CEG}_\text{DFA}$ in terms of the runtime and the number of iterations of the main loop.

In addition, we implemented two existing heuristics following Avellaneda et al. [11] for all the algorithms. First, in every algorithm, we learned models in an incremental manner: we started by learning DFAs (respectively, $\text{LTL}_\text{f}$ formulas) of size one and then increased the size by one. We repeated the process until bound $n$. This heuristic guarantees that the learned model is small in size.

Second, we used a subset of positive traces $P' \subseteq P$ that starts as an empty set. At each iteration of the algorithm, if the language of the learned model does not contain some traces from $P$, we then extended $P'$ with a trace from $P \setminus P'$, preferably the shortest one. This heuristic helped when dealing with large input samples because it used as few traces as possible from the positive examples $P$.

Also, we imposed a restriction on the semi-symbolic algorithm for learning $\text{LTL}_\text{f}$, Algorithm 11. We fixed the time horizon $K$ to a natural number instead of the double exponential theoretical upper bound of $2^{2^{n+1}}$. Using this heuristic means that the semi-symbolic algorithm does not solve Problem 7 in its full generality, but we demonstrate that we produced good enough formulas in practice.

Overall, we ran all the experiments using 8 GiB of RAM and two CPU cores with clock speed of 3.6 GHz.

### 6.3.1 RQ1: Performance Gain in Learning DFAs

To address RQ1, we compare the performance of our symbolic algorithm $\text{SYM}_\text{DFA}$ for learning DFAs, Algorithm 10, against the counterexample-guided algorithm $\text{CEG}_\text{DFA}$ by Avellaneda

[1]The code can be found online at https://github.com/cryhot/samp2symb/tree/paper/posdata

[2]We use an equivalent ASP [17] based encoding for optimization.

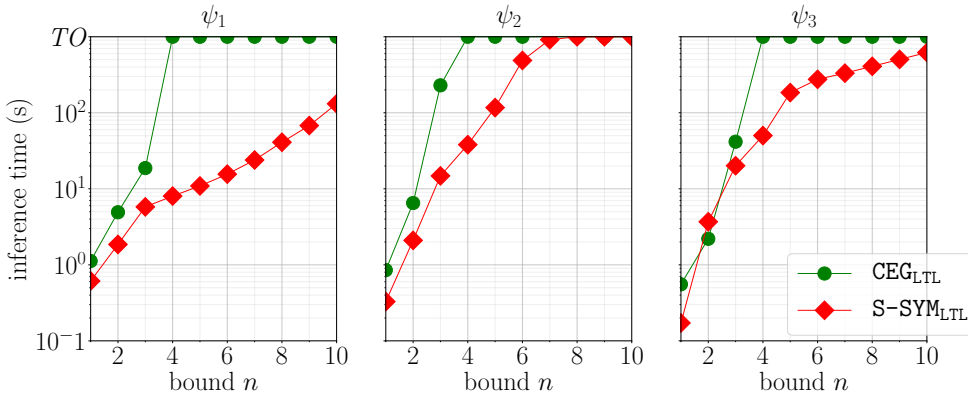



FIGURE 6.2: Comparison of S−SYM$_{\text{LTL}}$ and CEG$_{\text{LTL}}$ in terms of the inference time for three LTL ground truth formulas.

et al. [11].

For this research question, we considered a set of 28 random DFAs of size 2 to 10 generated using AALpy [165]. Using each random DFA, we generated samples of 1000 positive traces of lengths 1 to 10. We ran algorithms CEG$_{\text{DFA}}$ and SYM$_{\text{DFA}}$ with a timeout $TO = 1000$s, and for $n$ up to 10.

Figure 6.1 shows a comparison between the performance of SYM$_{\text{DFA}}$ and CEG$_{\text{DFA}}$ in terms of the learning time and the number of iterations required in the main loop. On the left plot, the average ratio of the number of iterations is 0.14, which, in fact, shows that SYM$_{\text{DFA}}$ required noticeably less number of iterations compared to CEG$_{\text{DFA}}$. On the right plot, the average ratio of the inference time is 1.09, which shows that the inference of the two algorithms is comparable, and yet SYM$_{\text{DFA}}$ is computationally less expensive since it requires fewer iterations.

### 6.3.2 RQ2: Performance Comparison for LTL Learning

To answer RQ2, we evaluated the performance of the proposed semi-symbolic algorithm S−SYM$_{\text{LTL}}$, Algorithm 11, and the counterexample-guided algorithm CEG$_{\text{LTL}}$, Algorithm 12.

For this research question, we generated samples (of only positive examples) based on 12 common LTL patterns (same as the ones from Table 4.1). For each sample from these 12 ground truth LTL$_{\text{f}}$ formulas, we generated 10000 positive traces of length 10. We ran CEG$_{\text{LTL}}$ and S−SYM$_{\text{LTL}}$ on the generated samples by setting the maximum formula size $n = 10$ and a timeout of $TO = 1000$s. For S−SYM$_{\text{LTL}}$, we additionally set the time horizon $K = 8$.

Figure 6.2 represents a comparison between the mentioned algorithms in terms of learning time for the ground truth LTL$_{\text{f}}$ formulas $\psi_1 = \mathsf{G}(p)$, $\psi_2 = \mathsf{G}(q \rightarrow (\mathsf{G}(\neg p)))$, and $\psi_3 = \mathsf{G}(\neg p) \vee \mathsf{F}(p \wedge \mathsf{F}(q))$. On average, S−SYM$_{\text{LTL}}$ ran 173.9% faster than CEG$_{\text{LTL}}$ for all the 12 samples. Our results showed that the LTL$_{\text{f}}$ formulas $\varphi$ inferred by S−SYM$_{\text{LTL}}$ were more or equally specific than the ground truth LTL$_{\text{f}}$ formulas $\psi$ (that is, $\varphi \rightarrow \psi$) for five out of the 12 samples, while the LTL$_{\text{f}}$ formulas $\varphi'$ inferred by CEG$_{\text{LTL}}$ were equally or more specific than the ground truth LTL$_{\text{f}}$ formulas $\psi$ (that is, $\varphi' \rightarrow \psi$) for three out of the 12 samples.

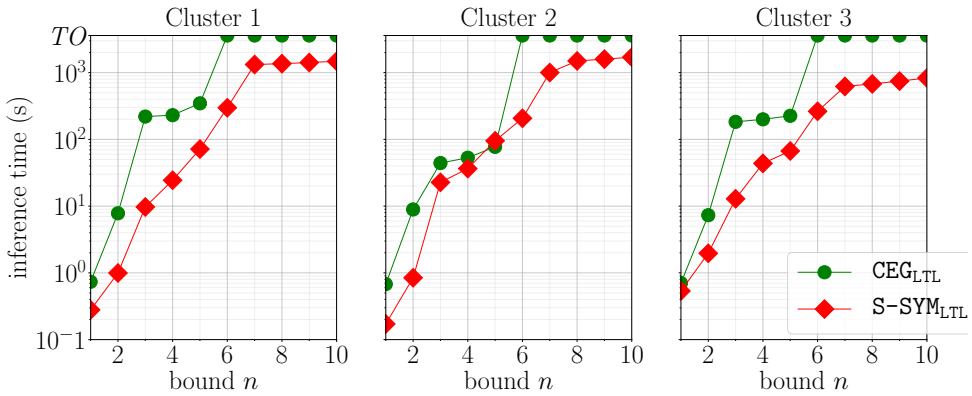



FIGURE 6.3: Comparison of S−SYM$_{\text{LTL}}$ and CEG$_{\text{LTL}}$ in terms of the runtime for three clusters of traces taken from a UAV.

### 6.3.3 RQ3: Learning LTL from Trajectories of an Aerial Vehicle

To answer RQ3, we ran S−SYM$_{\text{LTL}}$ and CEG$_{\text{LTL}}$ on trajectories of a simulated unmanned aerial vehicle (UAV). As input, we considered three samples of the trajectories. These samples were obtained by clustering 10000 traces demonstrated by the UAV into three bundles using the $k$-means clustering approach. For running the algorithms, we set $n = 10$, $K = 8$, and a timeout of $TO = 3600$s.

We now present some interesting formulas obtained by the algorithms. These formulas were based on several propositions that describe various attributes of a UAV. We here mention only the relevant propositions along with their meaning: $x_0$ indicates the desired target is reached, $x_1$ indicates the UAV is gliding (as opposed to thrusting), $x_2$ indicates there is a change in yaw angle, $x_3$ indicates there is a change in roll angle, and $x_4$ indicates the battery is low.

Using the above propositions, we found the following formulas:

$\mathsf{F}\, x_1 \rightarrow \mathsf{G}\, x_1$ : *either the UAV always glides, or it never glides;*

$\mathsf{G}(x_2 \rightarrow x_3)$ : *a change in yaw angle is always accompanied by a change in roll angle;*

$\mathsf{G}((\mathsf{F}\, x_0) \rightarrow x_0)$ : *if the UAV can eventually reach the target, then it is in the target;*

$\mathsf{G}(x_4 \rightarrow \mathsf{G}\, x_4)$ : *if the battery is low, then its stays low.*

We also provide possible natural language descriptions of the formulas.

We also compared the runtime of CEG$_{\text{LTL}}$ and S−SYM$_{\text{LTL}}$ for the three samples; we present the results in Figure 6.3. Our results showed that, on average, S−SYM$_{\text{LTL}}$ is 260.73% faster than CEG$_{\text{LTL}}$.

## 6.4 Conclusion

We presented novel algorithms for learning DFAs and LTL$_{\text{f}}$ formulas from positive examples only. Our algorithms rely on conciseness and language minimality as regularizers to learn meaningful models. We demonstrated the efficacy of our algorithms in three case studies.

A natural direction of future work is to lift our techniques to tackle learning from positive examples for other finite state machines (for instance, non-deterministic finite automata) and more expressive temporal logics (for instance, Property Specification Language).

# Chapter 7

# Learning Properties in the Property Specification Language

As we have seen in the earlier chapters, the temporal logic of primary focus has been Linear Temporal Logic (LTL). This choice has been rightfully driven by several theoretical advantages of LTL (for instance, easy translation to finite automata, simple model-checking, etc.), as well as concise, variable-free syntax and intuitive semantics.

However, one of the major downsides of LTL is its limited expressive power as compared to other temporal logics. As a consequence, many properties that arise naturally cannot be expressed in LTL; for instance, an event happening at every $n$-th point in time. The class of properties that can be expressed in LTL corresponds exactly to that of star-free $\omega$-languages [222], which excludes—among others—all properties involving modulo counting.

To overcome this limitation, the Property Specification Language (PSL) has been proposed, which has since been adopted by IEEE as an industrial standard for expressing temporal properties [123]. Although PSL is an extension of LTL and, hence, shares many of its beneficial properties, PSL differs from LTL in three important aspects:

1. The expressive power of PSL exceeds that of LTL: it is as expressive as the full class of regular $\omega$-languages [9]. In particular, properties involving modulo counting—as mentioned above—can easily be expressed in PSL.

2. PSL integrates easy-to-understand regular expressions in its syntax.

3. When learning from example traces, formulas, when expressed in PSL, can be arbitrarily more succinct than those expressed in LTL (see Lemma 8).

We believe that these three properties make PSL particularly well-suited as an interpretable description language. We refer to Section 7.1 for a detailed description of PSL.

The main focus of this chapter is an algorithm for learning formulas in PSL. Following earlier works on SAT-based algorithms for learning LTL\LTL$_\mathrm{f}$ formulas, the precise learning problem our algorithm solves is as follows: given a sample $\mathcal{S}$ consisting of two finite sets of positive and negative examples, learn a PSL formula $\varphi$ that is consistent with $\mathcal{S}$.

Although we cannot expect algorithms that learn consistent formulas to scale as well as statistical methods that allow for misclassifications (for instance, the one by Kim et al. [135]), being able to learn an exact model describing the given data is essential in a multitude of

applications, including few-shot learning, debugging of software systems, and many situations in which the observed data is without noise. We refer the reader to Neider et al. [169] and Camacho et al. [48] for more examples where learning consistent formulas is important.

To be as general and succinct as possible, we here assume examples to be infinite, ultimately periodic traces (that is, traces of the form $uv^\omega$, where $u, v$ are finite traces and $v^\omega$ is the infinite repetition of $v$) and focus on the core fragment of PSL. However, our algorithm can easily be adapted to learn from finite traces and extends smoothly to other future-time temporal operators of PSL. We expand on the learning problem in Section 7.2.

Our learning algorithm builds on top of the work by Neider et al. [169] for learning LTL formulas. Its key idea is to reduce the learning task to a series of constraint satisfaction problems in propositional logic and use a highly optimized SAT solver to search for a solution. By design, our algorithm infers a minimal PSL formula that is consistent with the examples, which is a particularly valuable property in our setting: we seek to learn human-interpretable formulas, and the size of the learned formula is a crucial metric for their interpretability (since larger formulas are generally harder to understand than smaller ones).

One key difference from LTL is that PSL extensively utilizes regular expressions. As a result, the learning algorithm presented in this chapter required innovation to effectively incorporate the learning of minimal regular expressions. This algorithm for learning regular expressions constitutes a contribution of independent interest and finds applications, particularly in natural language processing [23]. We refer to Section 7.3 for the technical description of the algorithms.

We empirically evaluate a prototype of our PSL learning algorithm on benchmarks that reflect typical patterns of both LTL and PSL formulas used in practice. The evaluation shows that our algorithm can infer informative PSL formulas and that these formulas are often more succinct than pure LTL formulas learned from the same examples. Moreover, the runtime of our prototype is comparable to the state-of-the-art tool for learning LTL formulas by Neider et al. [169]. We present the experimental results in Section 7.4 and end with a final discussion in Section 7.5.

### Related Works

Learning of temporal properties has recently attracted increasing attention. The literature in this area can be broadly structured along three dimensions.

The first dimension is the type of logic used to express models. Examples include learning of models expressed in Signal Temporal Logic (STL) [138], in Linear Temporal Logic (LTL) [48, 193, 169] and branching time logics, such as Computational Tree Logic (CTL) [218]. To the best of our knowledge, we are the first to consider the learning of PSL or equally expressive logic.

The second dimension is whether the learning algorithm requires the user to provide templates. Examples of algorithms that require templates are the works of [147] and [145], whereas the algorithms for LTL mentioned above do not require templates. Providing templates is often a challenging task as it requires the user to have a good understanding of the data. By contrast, our algorithm can learn arbitrary formulas without any assistance from the user.

The third dimension distinguishes between algorithms that learn an exact model and those that learn an approximate one. The learning algorithm we devise in this paper is exact (that is, it learns models that describe the data perfectly; due to our minimality constraint, however, these models generalize the data rather than overfit it). On the other hand, there also exists work that uses statistical methods to derive approximate formulas from noisy data [135].

This work is built upon the SAT-based learning algorithm by [169]. In fact, constraint solving is often used in learning problems. The perhaps most prominent examples are passive automata learning [116, 168] and counterexample-guided inductive synthesis [3].

## 7.1 Preliminaries

It is well-known that LTL cannot express natural properties such as modulo counting. To alleviate this serious restriction, the Property Specification Language (PSL) has been developed (for instance, see Eisner et al. [79]), which makes extensive use of regular expressions. The remainder of this section introduces regular expressions and PSL in detail.

### 7.1.1 Regular Expressions

To simplify the definition of PSL, we define regular expressions in a slightly non-standard way. Firstly, we use propositional formulas rather than symbols of an alphabet as atomic expressions. For example, for $\mathcal{P} = \{p, q\}$, the formula $p \vee q$ represents the set $\{\{p\}, \{q\}, \{p, q\}\}$ of symbols from $2^{\mathcal{P}}$, whereas $p \wedge \neg q$ represents the singleton set $\{\{p\}\}$. Secondly, we take an operational view of regular expressions in terms of a matching relation rather than the classical view as generators of regular languages.

*Regular expressions* are inductively constructed as follows, where the first grammar describes the construction of atomic expressions, while the second grammar describes the construction of general regular expressions:

$$\xi ::= p \in \mathcal{P} \mid \neg\xi \mid \xi \vee \xi$$
$$\rho ::= \varepsilon \mid \xi \mid \rho + \rho \mid \rho \circ \rho \mid \rho^*$$

As usual, the regular operator $+$ stands for choice, $\circ$ stands for concatenation, and $^*$ for finite repetition (Kleene star). As syntactic sugar, we also allow the Boolean operators $\wedge$, $\rightarrow$, and $\leftrightarrow$ in atomic expressions.

Let us first give atomic expressions a meaning. To this end, we assign to each atomic expression $\xi$ a set $[\![\xi]\!] \subseteq 2^{\mathcal{P}}$ of symbols in the following way:

$$[\![p]\!] = \{a \in 2^{\mathcal{P}} \mid p \in a\};$$
$$[\![\neg\xi]\!] = 2^{\mathcal{P}} \setminus [\![\xi]\!];$$
$$[\![\xi_1 \vee \xi_2]\!] = [\![\xi_1]\!] \cup [\![\xi_2]\!].$$

To define the semantics of regular expressions, we introduce a *matching relation* $\vdash$, which formalizes when an infix $u[t_1, t_2)$ of a finite trace $u \in (2^{\mathcal{P}})^*$ *matches* a regular expression.

Formally, the matching relation is defined as follows:

$$
\begin{aligned}
u[t_1, t_2) &\vdash \varepsilon \text{ if and only if } t_1 = t_2;\\
u[t_1, t_2) &\vdash \xi \text{ if and only if } t_2 = t_1 + 1 \text{ and } u[t_1] \in [\![\xi]\!];\\
u[t_1, t_2) &\vdash \rho_1 + \rho_2 \text{ if and only if } u[t_1, t_2) \vdash \rho_1 \text{ or } u[t_1, t_2) \vdash \rho_2;\\
u[t_1, t_2) &\vdash \rho_1 \circ \rho_2 \text{ if and only if for some } t_1 \leq t \leq t_2 : u[t_1, t) \vdash \rho_1 \text{ and } u[t, t_2) \vdash \rho_2;\\
u[t_1, t_2) &\vdash \rho^* \text{ if and only if } t_1 = t_2 \text{ or for some } t_1 + 1 \leq t \leq t_2 :\\
&\qquad\qquad u[t_1, t) \vdash \rho \text{ and } u[t, t_2) \vdash \rho^*.
\end{aligned}
$$

Note that this definition applies to finite infixes $u[t_1, t_2)$ of infinite traces $u \in (2^{\mathcal{P}})^\omega$ as well.

### 7.1.2 Property Specification Language

In this chapter, we consider the core fragment of the *Property Specification Language* [79], which we here abbreviate as *PSL* for the sake of brevity. This fragment extends LTL with a so-called *triggers operator* $\rho \mapsto \varphi$ where $\rho$ is a regular expression and $\varphi$ is a PSL formula. Intuitively, a trace $u \in (2^{\mathcal{P}})^\omega$ satisfies the PSL formula $\rho \mapsto \varphi$ if $\varphi$ holds every time the regular expression $\rho$ matches on a finite prefix of $u$. To define the semantics of the triggers operator formally, we extend the satisfaction relation of LTL as follows:

$$u \models \rho \mapsto \varphi \text{ if and only if for all } t \in \mathbb{N} \setminus \{0\} : u[0, t) \vdash \rho \text{ implies } u[t-1, \infty) \models \varphi.$$

Finally, we define the *size* $|\varphi|$ of a PSL formula $\varphi$ to be the number of its unique subformulas and subexpressions.

PSL is a popular specification language in industrial applications, having been standardized by IEEE [123]. It is as expressive as $\omega$-regular languages [9] (that is, languages accepted by nondeterministic Büchi automata) and, hence, exceeds the expressive power of LTL [222]. A simple property that cannot be expressed in LTL is that a proposition $p$ holds at every second point in time, which can be expressed in PSL as $(\mathit{true} \circ \mathit{true})^* \mapsto p$.

## 7.2 Problem Formulation

In this section, we formally define the learning problem studied in this chapter. Similar to Chapter 5, we consider the input to be a sample $\mathcal{S} = (P, N)$ of infinite traces from $(2^{\mathcal{P}})^\omega$, partitioned into a set $P$ of positive examples and a set $N$ of negative examples, such that $P \cap N = \emptyset$. As infinite traces, we specifically consider ultimately periodic traces, that is, traces of the form $uv^\omega$ where $u \in (2^{\mathcal{P}})^*$ and $v \in (2^{\mathcal{P}})^+$. Ultimately periodic traces are known to be sufficient in order to uniquely characterize $\omega$-regular languages [45] (and thus, PSL formulas).

Similar to LTL, we say that a PSL formula $\varphi$ is *consistent* with a sample $\mathcal{S} = (P, N)$ if $uv^\omega \models \varphi$ for all positive examples $uv^\omega \in P$ and $uv^\omega \not\models \varphi$ for all negative examples $uv^\omega \in N$.

We can now state the learning problem as follows:

**Problem 8.** *Given a sample $\mathcal{S} = (P, N)$, learn a minimal PSL formula that is consistent with $\mathcal{S}$*

Since LTL forms a subclass of PSL, one can easily construct a large PSL formula $\bigvee_{u \in P} \bigwedge_{v \in N} \varphi_{u,v}$ for the above problem, where $\varphi_{u,v}$ is simply an LTL formula that distinguishes $u$ and $v$ using a sequence of X-operators and appropriate propositions. However, enumerating all differences of a sample is clearly of little help towards the goal of learning a descriptive model. We require small formulas since they are easier for humans to interpret than large ones. Also, small formulas tend to provide better generalization for a given sample.

Before we explain our learning algorithm in detail, let us show that models expressed in PSL can be arbitrarily more succinct than those expressed in LTL, which follows from Theorem 4.1 of Wolper [222].

**Lemma 8.** *Let $n \in \mathbb{N}$ and $\mathcal{S}_n = (P_n, N_n)$ over $\mathcal{P} = \{p\}$ with $P_n = \{\{p\}^{2n}\{\}\{p\}^{\omega}\}$ and $N_n = \{\{p\}^{2n+1}\{\}\{p\}^{\omega}\}$. Then $(p \circ p)^* \mapsto \mathsf{X}\, p$ is a PSL formula (of constant size) consistent with $\mathcal{S}_n$, whereas every LTL formula that is consistent with $\mathcal{S}_n$ has size greater or equal to $2n$.*

## 7.3 SAT-based Learning Algorithm

The idea underlying our algorithm is to reduce the construction of a minimally consistent PSL formula to a constraint satisfaction problem in propositional logic and to use a highly optimized SAT solver to search for a solution. More precisely, given a sample $\mathcal{S}$ and size $n$, we rely on a propositional formula $\Phi_n^{\mathcal{S}}$ that has the following properties:

1. there exists a PSL formula of size $n \in \mathbb{N} \setminus \{0\}$ that is consistent with $\mathcal{S}$ if and only if $\Phi_n^{\mathcal{S}}$ is satisfiable; and
2. given a model $v$ of $\Phi_n^{\mathcal{S}}$, we can extract a PSL formula $\varphi_v$ of size $n$ that is consistent with $\mathcal{S}$.

By incrementing $n$ (starting from 1) until $\Phi_n^{\mathcal{S}}$ becomes satisfiable, we obtain an effective learning algorithm for models expressed in PSL, as shown in Algorithm 13. Note that the termination of this algorithm follows from the existence of a trivial solution (see Section 7.2). Moreover, its correctness follows from Properties 1 and 2 of $\Phi_n^{\mathcal{S}}$.

Roughly speaking, the formula $\Phi_n^{\mathcal{S}}$ is the conjunction

$$\Phi_n^{\mathcal{S}} := \Phi_n^{\text{PSL}} \wedge \Phi_n^{\text{sem}} \wedge \Phi_n^{\text{con}}, \tag{7.1}$$

where $\Phi_n^{\text{str}}$ encodes the structure of the prospective PSL formula $\varphi$, $\Phi_n^{\text{sem}}$ enforces that $\varphi$ is interpreted on the traces in $\mathcal{S}$ using the proper semantics of PSL and $\Phi_n^{\text{con}}$ ensures that $\varphi$ is consistent. In the remainder of this section, we describe both $\Phi_n^{\text{str}}$ and $\Phi_n^{\text{cst}}$ in detail.

**Algorithm 13** SAT-based learning algorithm for PSL

**Input:** Sample $\mathcal{S}$

1: $n \leftarrow 0$
2: **while** $\Phi_n^{\mathcal{S}}$ is satisfiable (say with model $V$) **do**
3: $\quad n \leftarrow n+1$
4: $\quad$ Construct $\Phi_n^{\mathcal{S}}$ and check its satisfiability
5: **end while**
6: construct $\varphi^V$
7: **return** $\varphi^V$

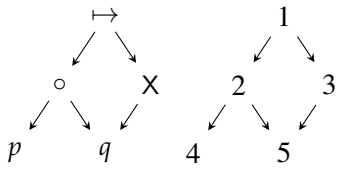



FIGURE 7.1: Syntax DAG of the PSL formula $(p \circ q) \mapsto \mathsf{X}\, q$ along with its identifiers

**Structural Constraints** The formula $\Phi_n^{\text{str}}$ relies on *syntax DAG*s as a canonical syntactic representation of PSL formulas. Syntax DAGs of PSL formulas are similar to that of LTL formulas (see Section 2.2.3). Figure 7.1 illustrates a syntax DAG along with the identifiers we use to identify the nodes of the syntax DAG.

Towards the definition of $\Phi_n^{\text{str}}$, we also rely on $\Lambda_R = \{\neg, \vee, +, \circ, ^*\} \cup \mathcal{P}$, the set of operators and propositions that can appear in regular expressions and $\Lambda_P = \Lambda_R \cup \{\mathsf{X}, \mathsf{U}, \mapsto\}$, the set of all PSL operators and propositions.

We encode the syntax DAG of a PSL formula using the following propositional variables: (i) $x_{i,\lambda}$ where $i \in \{1, \ldots, n\}$ and $\lambda \in \Lambda_P$ (ii) $l_{i,j}$ and $r_{i,j}$ where $i \in \{1, \ldots, n-1\}$ and $j \in \{i+1, \ldots, n\}$. Intuitively, the variables $x_{i,\lambda}$ encode the labeling of a syntax DAG in the sense that if $x_{i,\lambda}$ is set to true, then Node $i$ is labeled by $\lambda$. Similarly, the variables $l_{i,j}$ (respectively, $r_{i,j}$) encode the left (respectively, the right) child of Node $i$. By convention, we ignore the variables $r_{i,j}$ (respectively, $r_{i,j}$ and $l_{i,j}$) if Node $i$ is labeled with an unary operator (respectively, an proposition).

To enforce that these variables encode the syntax DAG of a PSL formula, we construct $\Phi_n^{\text{str}}$ using constraints similar to what we used for LTL. In particular, we employ Formulas 4.4 through 4.7, as defined in Chapter 4, by modifying the operator set to be $\Lambda_P$.

Additionally, we ensure that the labeling of the syntax DAG respects the type of the operators. In particular, we assign the following constraints to assign a valid ordering between the different types of operators.

$$\bigwedge_{\lambda \in \{+,*,\circ\}} \bigwedge_{\substack{1 \le i < n \\ i < j, j' \le n}} [x_{i,\lambda} \wedge l_{i,j} \wedge r_{i,j'}] \rightarrow \Big[ \bigvee_{\lambda' \in \Lambda_R} x_{j,\lambda'} \wedge \bigvee_{\lambda' \in \Lambda_R} x_{j',\lambda'} \Big] \tag{7.2}$$

$$\bigwedge_{\lambda \in \{\mathsf{X},\mathsf{U},\neg,\vee\}} \bigwedge_{\substack{1 \le i < n \\ i < j, j' \le n}} [x_{i,\lambda} \wedge l_{i,j} \wedge r_{i,j'}] \rightarrow \Big[ \bigvee_{\lambda' \in \Lambda_P} x_{j,\lambda'} \wedge \bigvee_{\lambda' \in \Lambda_P} x_{j',\lambda'} \Big] \tag{7.3}$$

$$\bigwedge_{\substack{1\leq i<n\\ i<j,j'\leq n}} [x_{i,\mapsto} \wedge l_{i,j} \wedge r_{i,j'}] \rightarrow \Big[ \bigvee_{\lambda'\in\Lambda_R} x_{j,\lambda'} \wedge \bigvee_{\lambda'\in\Lambda_P} x_{j',\lambda'} \Big] \tag{7.4}$$

Formulas 7.2, 7.3 and 7.4 ensure the ordering required for regular expression operators, LTL operators, and the triggers operator, respectively. One noteworthy observation is that Formula 7.4 for the triggers operator ensures that it combines a regular expression operator and a PSL operator.

$\Phi_n^{\text{str}}$ is the conjunction of all constraints discussed above. One can construct a syntax DAG from a model $V$ of $\Phi_n^{\text{str}}$ in a straightforward manner: label Node $i$ with the unique $\lambda \in \Lambda_P$ such that $V(x_{i,\lambda}) = 1$, designate Node 1 as the root, and arrange the nodes as described uniquely by $V(l_{i,j})$ and $V(r_{i,j})$.

**Semantic Constraints.** We exploit a simple observation about PSL to construct the propositional formula $\Phi_n$.

**Observation 2.** *Let $uv^\omega \in (2^{\mathcal{P}})^\omega$ and $\varphi$ be a PSL formula. Then, $uv^\omega[|u|+t_1,\infty) = uv^\omega[|u|+t_2,\infty)$ for $t_2 \equiv t_1 \mod |v|$. Thus, $uv^\omega[|u|+t_1,\infty) \models \varphi$ if and only if $uv^\omega[|u|+t_2,\infty) \models \varphi$.*

This observation is similar to one made for LTL in Chapter 5, Observation 1. (In fact, the above observation holds for any future-time temporal logic.) Intuitively, it states that there exists only a finite number of distinct infinite suffixes of a trace $uv^\omega$; and one can determine whether an infinite trace $uv^\omega$ satisfies a PSL formula based only on its finite prefix $uv$.

For a simple illustration of Observation 2, consider the formula $\mathsf{X}\,\varphi$, and suppose that we want to determine whether $uv^\omega[|uv|-1,\infty) \models \mathsf{X}\,\varphi$ holds; in other words, we like to check the satisfaction of $\mathsf{X}\,\varphi$ at the end of the prefix $uv$. Then, Observation 2 allows us to reduce this question to checking whether $uv^\omega[|u|,\infty) \models \varphi$ holds, instead of the original semantics of the X-operator, which depends on whether $uv^\omega[|uv|,\infty) \models \varphi$ is satisfied.

For reasoning about matchings of regular expressions, however, it is not enough to just consider the prefix $uv$. For instance, consider the ultimately periodic trace $uv^\omega = \{\}\{p\}(\{\})^\omega$ and the PSL formula $\varphi := (true \circ true)^* \mapsto p$ (stating that $p$ is true at every second position). By just considering the prefix $uv = \{\}\{p\}\{\}$, it seems that $uv^\omega \models \varphi$. However, *unrolling* the repeating part $v = \{\}$ once more, resulting in the prefix $uvv = \{\}\{p\}\{\}\{\}$, immediately shows that $uv^\omega \not\models \varphi$.

Similar to Observation 2, the next lemma provides a bound $b \in \mathbb{N}$ on the amount of unrolling required to gather enough information to determine the satisfaction of a triggers operator. This bound depends on the number $n$ of nodes of the syntax DAG and the function $M_{u,v}\colon \mathbb{N} \to \mathbb{N}$ defined by

$$M_{u,v}(t) = \begin{cases} t & \text{if } t < |uv|; \text{ and} \\ |u| + ((t-|u|)\,\%|v|) & \text{if } t \geq |uv|, \end{cases}$$

where $a \,\%\, b$ is the remainder of the division $\frac{a}{b}$. Intuitively, $M_{u,v}$ maps a timepoint $t$ in the trace $uv^\omega$ to an appropriate timepoint within the prefix $uv$. The lemma uses finite automata as representations of regular expressions to derive the bound.

**Lemma 9.** *Let $uv^\omega \in (2^{\mathcal{P}})^\omega$, $\psi = \rho \mapsto \varphi$ with $|\psi| = n$, and $b = 2^n + 1$. Then, $uv^\omega[t,\infty) \models \psi$ if and only if for all $t' \leq |u| + b|v|$, $uv^\omega[t,t') \vdash \rho$ implies $uv^\omega[M_{u,v}(t'-1),\infty) \models \varphi$.*

Note an important property of Lemma 9: reasoning about regular expressions and the triggers operator $\mapsto$ requires us to consider the prefix $uv^b$, while the prefix $uv$ is sufficient for reasoning about the remaining PSL operators.

We now prove the above lemma, which relies on a series of intermediate results. Since the triggers operator $\rho \mapsto \varphi$ uses regular expression $\rho$, as the first step, we reason about the length of the prefix of $uv^\omega$ that matches $\rho$. To this end, we prove a more general result Lemma 10, which reason about the length of prefix $uv^\omega[t_1,t_2)$ of any infinite trace $uv^\omega[t_1,\infty)$, where $0 \leq t_1 \leq |u|$.

For the proof of Lemma 10, we exploit the equivalence between regular expression and non-deterministic finite automaton (NFA) [211] and use NFA $\mathcal{A}_\rho$ as a representation of $\rho$. In particular, for formulating the bounds in the lemma, we rely on the size $m = |\mathcal{A}_\rho|$ of the smallest equivalent NFA for $|\rho|$. In order to obtain the bounds using the size $|\rho|$ of the regular expression, we can exploit relations between the size of a regular expression and its equivalent NFA, such as $|\mathcal{A}_\rho| = \mathcal{O}(2^{|\rho|})$ (see Gruber et al. [109] for tighter bounds).

**Lemma 10.** *Let $uv^\omega[t_1,t_2) \vdash \rho$ for some $t_1,t_2 \in \mathbb{N}$, where $t_1 \leq |u|$. Then, there exists $t \in \mathbb{N}$, $t \leq |u| + m|v|$ such that $uv^\omega[t_1,t) \vdash \rho$ and $t \equiv t_2 \mod |v|$.*

*Proof.* If $t_2 \leq |u| + m|v|$, we are done since we simply take $t = t_2$. However, if $t_2 > |u| + m|v|$, finding the suitable $t$ is slightly more involved, which we describe next.

The first observation we make is that, since $uv^\omega[t_1,t_2) \vdash \rho$, there is an accepting run $R$ of the NFA $\mathcal{A}_\rho$ (of size $m$) for $\rho$ on $uv^\omega[t_1,t_2)$. Figure 7.2 provides a pictorial depiction of $R$ (the first run). Notice that the portion of the run on $v^\omega[0,t_2)$ has a length greater than $m|v|$. We consider this portion of the run to be a sequence of the tuples $(state, index)$ (representing automaton configuration), where *state* refers to the current state of $\mathcal{A}_\rho$ and *index* refers to the timepoint in $v$ which will be read next. Now, due to the pigeonhole principle, if this run is longer than $m|v|$, then there exists a tuple, say $(q,\tau)$, which repeats during the run. Interestingly, the run from the first occurrence of $(q,\tau)$ to the second, which we refer to as $R_c$, forms a cycle in $\mathcal{A}_\rho$. In other traces, $\mathcal{A}_\rho$ reaches the same configuration after the run $R_c$.

Now, based upon the above observation, we create another accepting run $R'$ of $\mathcal{A}_\rho$, but, on a smaller prefix $uv^\omega[t_1,t')$, with $t' < t_2$ (the second run in Figure 7.2). $R'$ is quite similar to $R$, except that, we skip over $R_c$ and simply follow the run after $R_c$ starting from the first occurrence of $(q,\tau)$ itself. $R'$ is also a valid run, which is smaller than $R$ by the length of $R_c$. Moreover, notice that the length of $R$ is a multiple of $|v|$, which implies that $t' \equiv t \mod |v|$. Now, if $t' \leq |u| + m|v|$, we have found the desired $t$. However, if that is not the case, we repeat the same process of removing the cycle from the newly generated runs until the length

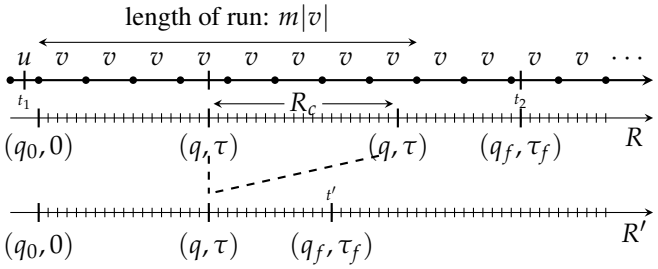



FIGURE 7.2: Removal of cycle $R_c$ from run $R$ of $\mathcal{A}_\rho$ on $uv^\omega$ to create a new run $R'$. Here, $(q, \tau)$ is the configuration that repeats, $(q_f, \tau_f)$ is the accepting configuration.

of the prefix is less than $|u| + m|v|$. Note that this process terminates since we begin with a prefix of finite length. □

We now use Lemma 10 to prove Lemma 11, which provides us an upper bound $b = m + 1$ on the number of unrollings required.

**Lemma 11.** *Let $b = m + 1$. Then, we have $uv^\omega[t, \infty) \models \rho \mapsto \varphi$ where $0 \leq t \leq |uv| - 1$ if and only if for all $t' < |u| + b|v|$, $uv^\omega[t, t') \vdash \rho$ implies $uv^\omega[t' - 1, \infty) \models \varphi$.*

*Proof.* The forward direction of the theorem follows from the semantics of the triggers operator.

The other direction is a direct consequence of Lemma 10. An additional $|v|$ term appears in the bound $b$ because here $t$ could range between 0 and $|uv| - 1$ in contrast to Lemma 10. When $t > |u|$, similar argument as in Lemma 10 works just by considering $u = v$. □

For the proof of Lemma 9, we exploit Lemma 11 along with Observation 2 to construct the function $M_{u,v}$.

We now return to constructing the semantic constraints in $\Phi_n^{\text{sem}}$. Towards its definition, we construct for each ultimately periodic trace $uv^\omega$ in $\mathcal{S}$ a propositional formula $\Phi_n^{u,v}$ that tracks the satisfaction of the prospective PSL formula on $uv^\omega$. Each of these formulas is built using auxiliary variables: (i) $y_{i,t}^{u,v}$ where $i \in \{1, \ldots, n\}$ and $t \in \{0, \ldots, |uv| - 1\}$; and (ii) $z_{i,t,t'}^{u,v}$ where $i \in \{1, \ldots, n\}$ and $t \leq t' \in \{0, \ldots, |uv^b| - 1\}$ ($b = 2^n + 1$ as in Lemma 9).

The variable $y_{i,t}^{u,v}$ is set to true if and only if $uv^\omega[t, \infty)$ satisfies the PSL formula $\varphi[i]$ (that is, if that node is labeled with a PSL operator); similarly, $z_{i,t,t'}^{u,v}$ is set to true if and only if $uv^\omega[t, t')$ matches the regular expression $\varphi[i]$ (that is, if that node is labeled with a regular expression operator). Observe that we have to create both the variables $y_{i,t}^{u,v}$ and $z_{i,t,t'}^{u,v}$ for each node since the type of a node (whether it roots a PSL formula or a regular expression) is determined dynamically during SAT solving.

We now list the constraints that establish the desired meaning of the variables $z_{i,t,t'}^{u,v}$.

$$\bigwedge_{1 \leq i \leq n} x_{i,\varepsilon} \rightarrow \left[ \bigwedge_{0 \leq t \leq t' < |uv^b|} z_{i,t,t'}^{u,v} \leftrightarrow [t = t'] \right] \tag{7.5}$$

$$\bigwedge_{1 \leq i \leq n} \bigwedge_{p \in \mathcal{P}} x_{i,p} \rightarrow \left[ \bigwedge_{0 \leq t \leq t' < |uv^b|} \begin{cases} z_{i,t,t'}^{u,v} \text{ if } p \in uv^b[t, t') \\ \neg z_{i,t,t'}^{u,v} \text{ if } p \notin uv^b[t, t') \end{cases} \right] \tag{7.6}$$

$$\bigwedge_{\substack{1\le i\le n\\ i<j,j'\le n}} x_{i,+} \wedge l_{i,j} \wedge r_{i,j'} \rightarrow \Big[ \bigwedge_{0\le t\le t'<|uv^b|} \Big[ z^{u,v}_{i,t,t'} \leftrightarrow z^{u,v}_{j,t,t'} \vee z^{u,v}_{j',t,t'} \Big] \Big] \tag{7.7}$$

$$\bigwedge_{\substack{1\le i\le n\\ i<j,j'\le n}} x_{i,\circ} \wedge l_{i,j} \wedge r_{i,j'} \rightarrow \Big[ \bigwedge_{0\le t\le t'<|uv^b|} \Big[ z^{u,v}_{i,t,t'} \leftrightarrow \bigvee_{t\le t''\le t'} z^{u,v}_{j,t,t''} \wedge z^{u,v}_{j',t'',t'} \Big] \Big] \tag{7.8}$$

$$\bigwedge_{\substack{1\le i\le n\\ i<j\le n}} x_{i,*} \wedge l_{i,j} \rightarrow \Big[ \bigwedge_{0\le t\le t'<|uv^b|} \Big[ z^{u,v}_{i,t,t'} \leftrightarrow [t=t'] \vee \bigvee_{t<t''\le t'} z^{u,v}_{i,t,t''} \wedge z^{u,v}_{j,t'',t'} \Big] \Big] \tag{7.9}$$

The above constraints simply encode the semantics of all of the regular expression operators.

To ensure the meaning of variables $y^{u,v}_{i,t}$, we reuse the already introduced constraints for the Boolean and LTL operators, Formulas 4.8 to 4.12.

For the triggers operator, we impose the following constraint to relate the variables $y^{u,v}_{i,k}$ and $z^{u,v}_{i,j,k}$.

$$\bigwedge_{\substack{1\le i\le n\\ i<j,j'\le n}} [x_{i,\mapsto} \wedge l_{i,j} \wedge r_{i,j'}] \rightarrow \bigwedge_{0\le t<|uv|} \Big[ y^{u,v}_{i,t} \leftrightarrow \bigwedge_{t\le t'<|uv^b|} \Big[ z^{u,v}_{j,t,t'} \rightarrow y^{u,v}_{j',M_{u,v}(t'-1)} \Big] \Big] \tag{7.10}$$

We now define $\Phi^{sem}_n := \bigwedge_{uv^\omega\in P\cup N} \Phi^{u,v}_n$ as a conjunction of all the semantic constraints discussed above.

**Consistency Constraints.** We finally define the formula $\Phi^{\text{con}}_n$ as follows:

$$\Phi^{\text{con}}_n := \bigwedge_{uv^\omega\in P} y^{u,v}_{1,0} \wedge \bigwedge_{uv^\omega\in N} \Phi^{u,v}_n \wedge \neg y^{u,v}_{1,0},$$

which enforces the consistency of the prospective formula with $\mathcal{S}$.

We now assert the correctness of the encoding $\Phi^{\mathcal{S}}_n$ using the following result.

**Lemma 12.** *Let $\mathcal{S} = (P, N)$ be a sample, $n \in \mathbb{N}\setminus\{0\}$, and $\Phi^{\mathcal{S}}_n$ be the propositional formula defined above. Then, the following holds:*

1. *If there exists a PSL formula $\varphi$ of size $n$ that is consistent with $\mathcal{S}$, then $\Phi^{\mathcal{S}}_n$ is satisfiable.*
2. *If $\Phi^{\mathcal{S}}_n$ is satisfiable, then there exists a PSL formula $\varphi$ of size $n$ that is consistent with $\mathcal{S}$.*

*Proof.* For proving the first statement, we use the syntax DAG of the formula $\varphi$, to formulate an assignment $V$ for the propositional variables in $\Phi^{\mathcal{S}}_n$. Towards this, we set $V(x_{i,\lambda}) = 1$ if and only if Node $i$ is labeled with operator $\lambda$, and $V(l_{i,j}) = 1$ (respectively, $V(r_{i,j}) = 1$) if and only if the left (respectively, the right) child of Node $i$ is Node $j$. It is easy to check that such an assignment $V$ satisfies the structural constraints $\Phi^{str}_n$ (refer to the appendix of Neider et al. [170] for more details). Further, we assign $V(y^{u,v}_{i,t}) = 1$ if and only if $\varphi[i]$ is a PSL formula and $uv^\omega[t,\infty) \models \varphi[i]$; and $V(z^{u,v}_{i,t,t'}) = 1$ if and only if $\varphi[i]$ is a regular expression and $uv^\omega[t,t') \vdash \varphi[i]$. One can again check that such an assignment $V$ satisfies $\Phi^{sem}_n$. Finally, since $\varphi$ is consistent with $\mathcal{S}$, $V(y^{u,v}_{1,0}) = 1$ for $uv^\omega \in P$ and $V(y^{u,v}_{1,0}) = 0$ for $uv^\omega \in N$, and thus $V$ satisfies $\Phi^{con}_n$.

For the second statement, first observe that $v \models \Phi_n^{\text{str}}$ and, thus, we can exploit the valuation of the variables $x_{i,\lambda}$, $l_{i,j}$, and $r_{i,j}$ to construct the syntax DAG of a PSL formula $\varphi^V$. We now need to show that $\varphi^V$ is indeed consistent with the sample $\mathcal{S}$.

To this end, we show that $V(y_{i,t}^{u,v}) = 1$ if and only if $uv^\omega[t,\infty) \models \varphi^V[i]$ for any $t \in \{0,\dots,|uv|-1\}$; and also $V(z_{i,t,t'}^{u,v}) = 1$ if and only if $uv^b[t,t') \vdash \varphi^V[i]$ for any $t,t' \in \{0,\dots,|uv^b|-1\}$. This proof proceeds via induction on the structure of $\varphi^V$, similar to the proof by Neider et al. [170].

We present the induction steps for the operators that appear only in PSL and not in LTL.

- In the case $\varphi^V[i] = \varphi^V[j] + \varphi^V[j']$, we have $V(x_{i,+})$, $V(l_{i,j})$, and $V(r_{i,j'})$ set to 1. Now, based on Formula 7.7 we have the following:

$$\begin{aligned} V(z_{i,t,t'}^{u,v}) = 1 &\iff V(z_{j,t,t'}^{u,v}) = 1 \text{ or } V(z_{j',t,t'}^{u,v}) = 1 \\ &\iff uv^b[t,t') \vdash \varphi^V[j] \text{ or } uv^b[t,t') \vdash \varphi^V[j'] \\ &\iff uv^b[t,t') \vdash \varphi^V[j] + \varphi^V[j']. \end{aligned}$$

- In the case $\varphi^V[i] = \varphi^V[j] \circ \varphi^V[j']$, we have $V(x_{i,\circ})$, $V(l_{i,j})$, and $V(r_{i,j'})$ all set to 1. Thus, based on Formula 7.8, we have the following:

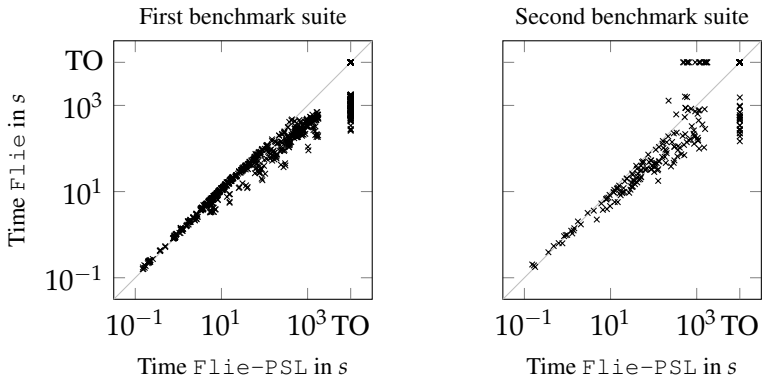
$$\begin{aligned} &V(z_{i,t,t'}^{u,v}) = 1 \\ &\quad\iff \text{for some } t \le t'' \le t',\ V(z_{j,t,t''}^{u,v}) = 1 \text{ and } V(y_{j',t'',t'}^{u,v}) = 1 \\ &\quad\iff \text{for some } t \le t'' \le t',\ uv^b[t,t'') \vdash \varphi^V[j], \text{ and } uv^b[t'',t) \vdash \varphi^V[j'] \\ &\quad\iff uv^b[t,t') \vdash \varphi^V[j] \circ \varphi^V[j']. \end{aligned}$$

- In the case $\varphi^V[i] = (\varphi^V[j])^*$, we have $V(x_{i,*})$ and $V(l_{i,j})$ set to 1. Thus, based on Formula 7.9, we have the following:

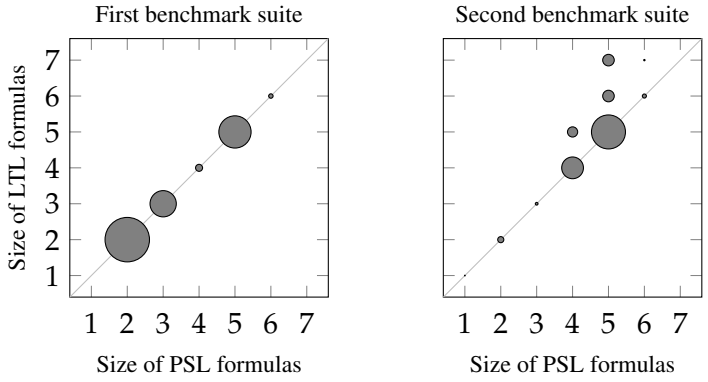
$$\begin{aligned} &V(z_{i,t,t'}^{u,v}) = 1 \\ &\quad\iff \begin{cases} t = t'; \text{ or} \\ \text{for some } t < t'' \le t',\ V(z_{j,t,t''}^{u,v}) = 1, \text{ and } V(z_{i,t'',t'}^{u,v}) = 1 \end{cases} \\ &\quad\iff \begin{cases} t = t'; \text{ or} \\ \text{for some } t < t'' \le t',\ uv^b[t,t'') \vdash \varphi^V[j] \text{ and } uv^b[t'',t') \vdash (\varphi^V[i])^* \end{cases} \\ &\quad\iff uv^b[t,t') \vdash (\varphi^V[j])^* \end{aligned}$$

- In the case $\varphi^V[i] = \varphi^V[j] \mapsto \varphi^V[j']$, we have $V(x_{i,\mapsto})$, $V(l_{i,j})$, and $V(r_{i,j'})$ set to 1. Thus, based on Formula 7.10 we make the following deductions:

$$\begin{aligned} &V(y_{i,t}^{u,v}) = 1 \\ &\quad\iff \text{for all } t \le t' < |uv^b|,\ V(z_{j,t,t'}^{u,v}) = 1 \text{ implies } V(y_{M_{j',u,v}(t'-1)}^{u,v}) = 1 \\ &\quad\iff \text{for all } t \le t' < |u| + b|v|,\ uv^b[t,t') \vdash \varphi^V[j] \text{ implies} \end{aligned}$$




(a) Comparison of runtime in seconds




(b) Comparison of size

FIGURE 7.3: Comparison of `Flie-PSL` and `Flie`. The size of the bubbles reflects the number of formulas. "TO" indicates timeouts.

$$
\begin{aligned}
& uv^\omega[M_{u,v}(t'-1),\infty) \models \varphi[j'] \\
\iff \quad & \text{for all } t \le t' < |u| + b|v|,\ uv^b[t,t') \vdash \varphi^V[j] \text{ implies} \\
& uv^\omega[t'-1,\infty) \models \varphi[j'] \\
\iff \quad & uv^\omega[t,\infty) \models \varphi^V[j] \mapsto \varphi^V[j'].
\end{aligned}
$$

□

**Theorem 15.** *Given a sample $\mathcal{S}$, Algorithm 13 terminates and learns a minimal PSL formula that is consistent with $\mathcal{S}$.*

*Proof.* The termination of this algorithm is guaranteed by the existence of a trivially large PSL formula consistent with $\mathcal{S}$. The correctness of $\Phi_n^{\mathcal{S}}$ ensures that the algorithm finds a consistent PSL formula of size $n$ if one exists. Finally, the minimality of the formula is guaranteed by the search for increasing size. □

**Corollary 15.1.** *Since PSL uses regular expressions in its syntax, a simple modification of Algorithm 13 learns minimal regular expressions from (finite) samples of finite traces.*

## 7.4 Experimental Evaluation

We have implemented a prototype of our learning algorithm, named `Flie-PSL` (Formal Language Inference Engine for PSL)[1]. This prototype is written in Python and uses Z3 [164] as SAT solver.

Deviating slightly from the general algorithm presented in Section 7.3, we have implemented the following improvement: instead of generating the variables $y_{i,t}^{u,v}$ and $z_{i,t,t'}^{u,v}$ for each node, we generate the latter variables (and their constraints) only for $0 \leq m < n$ nodes and the former variables (and their constraints) for the remaining $n - m$ nodes. This effectively limits the size of a regular expression in the final PSL formula to $m$. To obtain a complete algorithm, we iterate over all valid values for $m$ before increasing $n$.

To assess the performance of our prototype, we have compared it to an implementation of the LTL learning algorithm `Flie` by Neider et al. [169].

### 7.4.1 RQ: Comparison to LTL Learning

To make the comparison to LTL learning as fair as possible, we have used two benchmark suites. The first benchmark suite is taken directly from Neider et al. and contains 1217 samples, which were generated from common LTL properties. The second benchmark suite is meant to simulate real-world PSL use-cases and contains 390 synthetic samples, which we have generated from PSL formulas that commonly appear in practice (for instance, $(p_1 \circ p_2)^* \mapsto q$; see [79] for more examples).

Our procedure to generate these samples is similar to the one by Neider et al. [169] and proceeds as follows: first, we select a formula $\varphi$ from our pool of PSL formulas; second, we generate up to 500 ultimately periodic traces $uv^\omega$ with length $\leq 15$; third, we partition these traces into sets $P$ and $N$ depending on their satisfaction of $\varphi$. In total, the median size of the samples in the second benchmark suite is 100 traces. All experiments were conducted on a single core of an Intel Xeon E7-8857 V2 CPU (at 3.6 GHz) with a timeout of 1800 $s$.

Figure 7.3a presents the comparison of the runtime of `Flie-PSL` and `Flie` on the two benchmark suites. In general, `Flie-PSL` is moderately slower than `Flie` and timed out 1.34 times more often (`Flie-PSL` timed out 38.4% and 56.2% of the times on the first and second benchmark suite, respectively, whereas `Flie` timed out 24.8% and 53.6% of the times). This came as a surprise to us because the SAT encoding in the case of PSL is much more involved than the one for LTL. In fact, there were even 25 benchmarks on which `Flie-PSL` outperformed `Flie` because it was able to learn smaller formulas.

Figure 7.3b presents the comparison of the sizes of the formulas learned by both tools. On the first benchmark suite, we observe that `Flie-PSL` mainly produced pure LTL formulas of the same size as `Flie` (a likely explanation for this is that these benchmarks have explicitly been designed to capture LTL properties). However, on 68 benchmarks of the second suite, `Flie-PSL` learned PSL formulas that used non-LTL operators and was able to recover the exact PSL property that was used to generate the sample in 40 of the benchmarks. Overall, `Flie-PSL` learned a smaller formula than `Flie` for 52 benchmarks.

[1] The code is publicly available at https://github.com/ifm-mpi/Flie-PSL

## 7.5 Conclusion

We have developed an algorithm for learning human-interpretable models expressed in PSL and have shown empirically that this algorithm infers interesting PSL formulas with only little overhead as compared to learning LTL formulas.

An interesting direction for future work would be to syntactically restrict the class of regular expressions to reduce the number $b$ of unrolling required for the variables $z^{u,v}_{i,t,t'}$ and, hence, improve performance. Moreover, we plan to extend our algorithm to be able to handle noisy data and, orthogonally, to learn models expressed as $\omega$-regular expressions.

# Chapter 8

# Learning Properties in Continuous-time Temporal Logics

In this chapter, we study the passive learning of continuous-time logics, with a particular emphasis on Metric Temporal Logic (MTL) [141]. MTL stands out for two main reasons: first, it is widely used as a specification language for describing cyber-physical systems (CPS) [118, 157]; and second, it serves as a relatively straightforward continuous-time extension of Linear Temporal Logic (LTL), retaining several advantages of LTL.

While MTL offers various semantics, such as discrete, dense-timepointwise, etc. [175], we focus on the *dense-time continuous* semantics. This semantics is often considered to be more natural and general than the counterparts [24, 15]. Moreover, due to its involved definition, there is little work in automated learning for this semantics.[1]

Formal verification involving MTL properties is typically computationally demanding. To mitigate this, in practice, lightweight techniques such as *runtime verification* are often employed to ensure the correctness of CPS during execution. Runtime verification techniques strike a balance between the rigor of formal verification and the resource efficiency of conventional testing [66]. Thus, in this chapter, we tailor the passive learning problem with a focus on its application in runtime verification.

Specifically, in runtime verification, we consider *monitoring* system executions against formal specifications. Over the years, numerous monitoring techniques have been proposed for a variety of specification languages [114, 72, 69, 21].

Virtually all monitoring techniques for MTL rely on the availability of a formal specification. However, manually writing specifications is a tedious and error-prone task, as we have argued in Chapter 1. Formulating effective specifications for verification tasks remains a significant challenge [31, 197].

To tackle the lack of formal specifications, there have been efforts to automatically learn specifications from system executions. Most of the existing works have targeted specification languages such as Linear Temporal Logic (LTL) [49, 169, 186] and Signal Temporal Logic (STL) [10, 149, 160, 201], with few works for MTL [121, 228]. Many of the works tend to learn specifications that are *concise* in size. Concise specifications are preferred over

[1] This work is considerably different from STL learning in Chapter 4 due to the choice of different (arguably, more involved) semantics.

large ones because, based on the principle of Occam's razor, they are easier for humans to understand, as we have argued in many of the previous chapters.

However, conciseness is not the only measure of interest for specifications, especially in the context of online monitoring. In online monitoring, specifically in *stream-based* runtime monitoring, a monitor reads an execution as a stream of data and verifies if a given specification is invariant (that is, holds at all timepoints) in the execution. Many stream-based monitors [106, 129, 148] support MTL formulas. Typically, such monitors produce a stream of (Boolean) verdicts with some "latency", which depends on the lookahead of the formula. The lookahead required for an MTL formula is often formalized as its *future-reach* [118, 122], which is the amount of time required to determine its satisfaction at any timepoint.

With the aim of reducing the latency for efficient online monitoring, we focus on automatically learning MTL specifications based on two regularizers, size and future-reach. As input data, we rely on a sample $\mathcal{S}$ consisting of executions of a system that are observed for a finite duration. We consider the sample to be partitioned into a set $P$ of positive (or desirable) executions and a set $N$ of negative (or undesirable) executions.

We now formulate the central problem of learning MTL formulas as follows: given a sample $\mathcal{S} = (P, N)$ and a future-reach bound $K$, learn a minimal size MTL formula $\varphi$ that (i) is *globally-separating* for $\mathcal{S}$, in that $\varphi$ holds at all timepoints in the positive executions and does not hold at some timepoint in the negative executions, and (ii) the future-reach of $\varphi$ is smaller than $K$. The property of being globally-separating for $\mathcal{S}$ ensures that prospective formula $\varphi$ is invariant in the desirable executions and not in the undesirable executions, as is typically preferred in specifications for online monitoring [26]. We expand on the problem formulation in Section 8.2.

Also, interestingly, without a future-reach bound, the most concise MTL formula that can be learned can have a large future-reach value, increasing the latency required for online monitoring. To illustrate this, assume that we observe some simulations of an autonomous vehicle. During the simulations, we sample executions (shown below) of the vehicle every second for six seconds. We classify them as positive (denoted using $u_i$'s) or negative (denoted using $v_i$'s) based on whether the vehicle encountered a collision or not.

| | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| $u_1$: | $\{p,q\}$ | $\{p\}$ | $\{q\}$ | $\{p,q\}$ | $\{p\}$ | $\{p\}$ |
| $u_2$: | $\{q\}$ | $\{\}$ | $\{q\}$ | $\{p\}$ | $\{p\}$ | $\{p,q\}$ |
| $v_1$: | $\{p\}$ | $\{q\}$ | $\{\}$ | $\{\}$ | $\{\}$ | $\{\}$ |
| $v_2$: | $\{p\}$ | $\{p,q\}$ | $\{p\}$ | $\{\}$ | $\{p\}$ | $\{\}$ |

In the executions, we use $p$ to denote that there is no obstacle within a particular unsafe distance ahead of the vehicle and $q$ to denote that the vehicle's brake is triggered. Our setting considers executions to be *continuous*. Thus, to ensure continuity of execution, in the above example, if $p$ occurs at timepoint $t$, we interpret it as $p$ holding during the entire interval $[t, t+1)$. We also assume that the executions last up to a final timepoint $T$, which is 6 for this example. Thus, for the execution $u_1$, $p$ holds in the intervals $[0,2)$ and $[3,6)$.

In the sample, a minimal globally separating formula is $\varphi_1 = \mathsf{F}_{[0,3]}\, q$. The formula $\varphi_1$ being globally separating indicates that in all positive executions, the brake is triggered every

three seconds (that is, within the interval $[t, t+3]$ for every timepoint $t$), irrespective of whether there is an obstacle within the unsafe distance. The formula $\varphi_1$ has size two and a future-reach of three seconds, meaning that any online monitor requires a three second lookahead window to check the satisfaction of $\varphi_1$. There is another formula $\varphi_2 = \neg p \rightarrow \mathsf{F}_{[0,1]}\, q$ that is globally separating for the sample. The formula $\varphi_2$ being globally-separating indicates that in all positive executions, for every timepoint $t$, if an obstacle is within the unsafe distance, then the brake is triggered within one second (that is, within the interval $[t, t+1]$). Although of size five, $\varphi_2$ has a future-reach of one second and will be typically preferred over $\varphi_1$ for online monitoring in a safety-critical scenario.

For the problem of learning MTL formulas, we first study whether a solution exists. It turns out that there are samples $\mathcal{S}$ and future-reach bound $K$ for which there might not exist any formula that is globally-separating for $\mathcal{S}$ and has future-reach within $K$. To aid in checking whether a prospective formula exists, we identify a simple characterization of $\mathcal{S}$ based on the future-reach $K$. Such a characterization enables us to design an NP algorithm that can decide whether a prospective algorithm exists. Also, it provides an upper-bound, which is polynomial in the inputs $\mathcal{S}$ and $K$, on the size of the prospective formula if one exists. We mention the details of the existence check in Section 8.3.

To learn a prospective formula, we rely on a reduction to constraint satisfaction problems. In particular, following other works in learning formulas [169, 195], our algorithm encodes the problem in a series of satisfiability modulo theory (SMT) problems in Linear Real Arithmetic (LRA). To our knowledge, we design the first SMT-based algorithm that can learn MTL formulas of arbitrary syntactic structure. Such an SMT-based algorithm allows us to extend our algorithm to work for other settings that are common in the learning of formulas [93, 154].

Further, we analyze the complexity of the decision version of the problem of learning MTL formulas. While the exact complexity lower bounds are open, we show that the corresponding decision problem is in NP. The central SMT-based algorithm with all the theoretical results is in Section 8.4.

We also implement our algorithm using a popular SMT solver in a prototype named TEAL. We evaluate the ability of TEAL to learn MTL formulas typically employed for monitoring cyber-physical systems. We also empirically study the interplay between the size and future-reach of a formula. We present all the experimental results in Section 8.5, with a final discussion in Section 8.6.

**Related works.**

To our knowledge, there are only a limited number of works for learning MTL formulas. One of them [228] infers MTL formulas as decision trees for representing task knowledge in Reinforcement Learning. Some other works [121, 229] consider the parameter search problem for MTL where, given a parametric MTL formula (that is, an MTL formula with missing temporal bounds), they infer the ranges of parameters where the formula holds/does not hold on a given system. Unlike our work, none of these works aim to learn concise MTL specifications for monitoring tasks.

There are, nevertheless, numerous runtime monitoring procedures for MTL [208, 15, 71, 118, 25, 54, 131, 148], clearly indicating the need for efficiently monitorable MTL specifications. Many of them also rely on the future-reach of a specification [118, 25] or other similar measures (for instance, horizon [71], worst-case propagation delay [131], etc.) to quantify the efficiency of their monitoring procedure.

Interestingly, several works focus on learning formulas in STL, an extension of MTL to reason about real-valued signals. Bartocci et al. [22] provide a comprehensive survey of the existing works on inferring STL. Many of them [10, 139, 138] solve the parameter search for STL, while others [37, 36] learn decision trees over STL formulas, which typically do not result in concise formulas. There are few works [160, 171] that prioritize the conciseness of formulas during inference. These works cannot be directly applied to solve our problem for two main reasons. First, these works assume inputs to be *piecewise-affine continuous* signals. While the above assumption is natural for learning STL formulas inference from real-valued signals, in our setting, we must rely on the assumption that our inputs are *piecewise-constant* signals, which is natural for Boolean-valued signals. Second, these works do not employ any measure, apart from conciseness, that directly influences the efficiency of runtime monitoring.

Finally, there are works on learning formulas in other temporal logics such as Linear Temporal Logic (LTL) [169, 193, 49, 186], Property Specification Language (PSL) [195], etc., which are not easily extensible to our setting.

## 8.1 Preliminaries

In this section, we introduce the basic notations used throughout the paper.

**Signals and Prefixes.** We represent continuous system executions as signals. A *signal* $\boldsymbol{x}\colon \mathbb{R}_{\geq 0} \to 2^{\mathcal{P}}$ over a set of propositions $\mathcal{P}$ is an infinite time series that describes relevant system events over time. A prefix of a signal $\boldsymbol{x}$ restricted to domain $\mathbb{T} = [0, T), T \in \mathbb{R}_{\geq 0}$ is a function $\boldsymbol{x}_{\mathbb{T}}\colon \mathbb{T} \to 2^{\mathcal{P}}$ where $\boldsymbol{x}_{\mathbb{T}}(t) = \boldsymbol{x}(t)$ for all $t \in \mathbb{T}$.

To learn MTL formulas, we rely on finite observations that are sequences of the form $\Omega = \langle (t_i, \delta_i) \rangle_{i \leq n}$, $n \in \mathbb{N}$ such that (i) $t_0 = 0$, (ii) $t_n < T$, and (ii) for all $i \leq n$, $\delta_i \subseteq \mathcal{P}$ is the set of propositions that hold at timepoint $t_i$. To construct well-defined signal prefixes, we approximate each observation $\Omega$ as a *piecewise-constant* signal prefix $\boldsymbol{x}^{\Omega}_{\mathbb{T}}$ using interpolation as: (i) for all $i < n$, for all $t \in [t_i, t_{i+1})$, $\boldsymbol{x}_{\mathbb{T}}(t) = \delta_i$; and (ii) for all $t \in [t_n, T)$, $\boldsymbol{x}_{\mathbb{T}}(t) = \delta_n$. For brevity, we refer to signal prefixes simply as 'prefixes' when clear from the context.

**Metric Temporal Logic.** MTL is a logic formalism for specifying real-time properties of a system. We consider the following syntax of MTL:

$$\varphi := p \in \mathcal{P} \mid \neg p \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \vee \varphi_2 \mid \varphi_1 \,\mathsf{U}_I\, \varphi_2 \mid \mathsf{F}_I\, \varphi \mid \mathsf{G}_I\, \varphi$$

where $p \in \mathcal{P}$ is a proposition, $\neg$ is the negation operator, $\wedge$ and $\vee$ are the conjunction and disjunction operators respectively, and $\mathsf{U}_I, \mathsf{F}_I$ and $\mathsf{G}_I$ are the timed-Until, timed-Finally

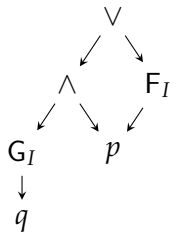



FIGURE 8.1: Syntax DAG of $(p \wedge \mathsf{G}_I\, q) \vee (\mathsf{F}_I\, p)$

and timed-Globally operators respectively. Here, $I$ is a closed interval of non-negative real numbers of the form $[a, b]$ where $0 \leq a \leq b$[2]. Note that the syntax is presented in *negation normal form*, meaning that the $\neg$ operator can only appear before a proposition.

As a syntactic representation of an MTL formula, we rely on *syntax-DAGs*. A syntax-DAG is similar to the parse tree of a formula but with shared common subformulas. We define the size $|\varphi|$ of an MTL formula $\varphi$ as the number of nodes in its syntax-DAG, the same as LTL. For instance, the size of $(p \wedge \mathsf{G}_I\, q) \vee (\mathsf{F}_I\, p)$ is six as its syntax-DAG has six nodes, as shown in Figure 8.1.

As mentioned already, we follow the continuous semantics of MTL. First, we mention the standard continuous semantics ($\models$) of MTL over infinite signals following the work of Ouaknine et al. [175].

Given an infinite signal $\boldsymbol{x}$, an MTL formula $\varphi$ and a timepoint $0 \leq t \in \mathbb{T}$, we define the relation $(\boldsymbol{x}, t) \models \varphi$, $\varphi$ holds on $\boldsymbol{x}$ at timepoint $t$, as follows:

$$
\begin{aligned}
(\boldsymbol{x}, t) &\models p \text{ if and only if } p \in \boldsymbol{x}(t) \\
(\boldsymbol{x}, t) &\models \neg p \text{ if and only if } p \notin \boldsymbol{x}(t) \\
(\boldsymbol{x}, t) &\models \varphi_1 \wedge \varphi_2 \text{ if and only if } (\boldsymbol{x}, t) \models \varphi_1 \text{ and } (\boldsymbol{x}, t) \models \varphi_2 \\
(\boldsymbol{x}, t) &\models \varphi_1 \vee \varphi_2 \text{ if and only if } (\boldsymbol{x}, t) \models \varphi_1 \text{ or } (\boldsymbol{x}, t) \models \varphi_2 \\
(\boldsymbol{x}, t) &\models \varphi_1\, \mathsf{U}_{[a,b]}\, \varphi_2 \text{ if and only if } \exists t' \in [t+a, t+b] : (\boldsymbol{x}, t') \models \varphi_2 \text{ and} \\
&\qquad\qquad \forall t'' \in [t, t'] : (\boldsymbol{x}, t'') \models \varphi_1
\end{aligned}
$$

In case $(\boldsymbol{x}, 0) \models \varphi$, we simply write $\boldsymbol{x} \models \varphi$ and say that $\boldsymbol{x}$ satisfies $\varphi$ or $\boldsymbol{x}$ holds on $\varphi$. The semantics of the $\mathsf{F}_I$ and the $\mathsf{G}_I$ operators can be derived using standard syntactic relations: $\mathsf{F}_I\, \varphi := \mathit{true}\, \mathsf{U}_I\, \varphi$ and $\mathsf{G}_I\, \varphi = \neg\, \mathsf{F}_I\, \varphi$.

While the above semantics is the standard one, our setting demands semantics of MTL over finite prefixes such that the learned formulas will be 'useful' while monitoring over infinite signals. Intuitively, we want an 'optimistic' semantics ($\models_{\mathrm{f}}$) of an MTL formula $\varphi$ over a prefix $\boldsymbol{x}_{\mathbb{T}}$ such that $\boldsymbol{x}_{\mathbb{T}} \models_{\mathrm{f}} \varphi$ if there exists an infinite signal *extending* $\boldsymbol{x}_{\mathbb{T}}$ that satisfies $\varphi$. In other words, $\boldsymbol{x}_{\mathbb{T}}$ "carries no evidence against" the formula $\varphi$. Formally, we want the definition of $\models_{\mathrm{f}}$ to satisfy the following lemma.

[2]Since we infer MTL formulas with bounded lookahead, we restrict $I$ to be bounded.

**Lemma 13.** *Given a prefix $x_{\mathbb{T}}$, let $ext(x_{\mathbb{T}}) = \{x \mid x_{\mathbb{T}}$ is a prefix of $x\}$ be the set of all infinite extensions of $x_{\mathbb{T}}$. Then given an MTL formula $\varphi$, $x_{\mathbb{T}} \models_f \varphi$ if there exists $x \in ext(x_{\mathbb{T}})$ such that $x \models \varphi$.*

Towards this, we follow the idea of 'weak semantics' of MTL defined in [118][3] and interpret MTL over finite prefixes. Given a prefix $x_{\mathbb{T}}$, we inductively define $(x_{\mathbb{T}}, t) \models_{\mathrm{f}} \varphi$, that is, $\varphi$ holds on $x$ at timepoint $t \in \mathbb{T}$, as follows:

$(x_{\mathbb{T}}, t) \models_{\mathrm{f}} p$ if and only if $p \in x_{\mathbb{T}}(t)$;

$(x_{\mathbb{T}}, t) \models_{\mathrm{f}} \neg p$ if and only if $p \notin x_{\mathbb{T}}(t)$;

$(x_{\mathbb{T}}, t) \models_{\mathrm{f}} \varphi_1 \wedge \varphi_2$ if and only if $(x_{\mathbb{T}}, t) \models_{\mathrm{f}} \varphi_1$ and $(x_{\mathbb{T}}, t) \models_{\mathrm{f}} \varphi_2$;

$(x_{\mathbb{T}}, t) \models_{\mathrm{f}} \varphi_1 \vee \varphi_2$ if and only if $(x_{\mathbb{T}}, t) \models_{\mathrm{f}} \varphi_1$ or $(x_{\mathbb{T}}, t) \models_{\mathrm{f}} \varphi_2$;

$(x_{\mathbb{T}}, t) \models_{\mathrm{f}} \varphi_1 \mathsf{U}_{[a,b]} \varphi_2$ if and only if

- $\exists t' \in [t+a, t+b] \cap \mathbb{T} : (x_{\mathbb{T}}, t') \models_{\mathrm{f}} \varphi_2$ and $\forall t'' \in [t, t'] : (x_{\mathbb{T}}, t'') \models_{\mathrm{f}} \varphi_1$, or
- $T \leq t + b$ and $\forall t'' \in [t, T) : (x_{\mathbb{T}}, t'') \models_{\mathrm{f}} \varphi_1$

$(x_{\mathbb{T}}, t) \models_{\mathrm{f}} \mathsf{F}_{[a,b]} \varphi$ if and only if $t + b \geq T$ or $\exists t' \in [t+a, t+b] \cap \mathbb{T} : (x_{\mathbb{T}}, t') \models_{\mathrm{f}} \varphi$;

$(x_{\mathbb{T}}, t) \models_{\mathrm{f}} \mathsf{G}_{[a,b]} \varphi$ if and only if $t + a \geq T$ or $\forall t' \in [t+a, t+b] \cap \mathbb{T}, (x_{\mathbb{T}}, t') \models_{\mathrm{f}} \varphi$

In case $(x_{\mathbb{T}}, 0) \models_{\mathrm{f}} \varphi$, we simply write $x_{\mathbb{T}} \models_{\mathrm{f}} \varphi$ and say that $x_{\mathbb{T}}$ satisfies $\varphi$ or $\varphi$ holds on $x_{\mathbb{T}}$. Also, for ensuring that our semantics complies with Lemma 13, we define $(x_{\mathbb{T}}, t) \models_{\mathrm{f}} \varphi$ for all $t \geq T$ for any $\varphi$.

We prove that our chosen semantics satisfy the property described in Lemma 13.

*Proof of Lemma 13.* We, in fact, prove a stronger statement from which Lemma 13 follows: for all $t \in [0, T)$, $(x_{\mathbb{T}}, t) \models_{\mathrm{f}} \varphi$ if there exists a signal $x \in ext(x_{\mathbb{T}})$ such that $(x, t) \models \varphi$.

The proof now proceeds via an induction on the MTL formula $\varphi$.

- For the base case, let $\varphi = p \in \mathcal{P}$. Then, for all $t \in [0, T)$, if there exists $x \in ext(x_{\mathbb{T}})$ such that $(x, t) \models p$, then $(x_{\mathbb{T}}, t) \models_{\mathrm{f}} p$ since $(x, t) \models \varphi$ and thus, $(x_{\mathbb{T}}, t) \models \varphi$. The same argument extends to the $\neg$ operator.

- Let $\varphi = \varphi_1 \wedge \varphi_2$. Then, for all $t \in [0, T)$, if there exists $x \in ext(x_{\mathbb{T}})$ such that $(x, t) \models \varphi_1$ and $(x, t) \models \varphi_2$. Then, $(x_{\mathbb{T}}, t) \models_{\mathrm{f}} \varphi_1$ and $(x_{\mathbb{T}}, t) \models_{\mathrm{f}} \varphi_2$ by induction hypothesis. The same argument extends to the $\vee$ operator.

- Let $\varphi = \varphi_1 \mathsf{U}_{[a,b]} \varphi_2$ and fix a timepoint $t \in [0, T)$. We have to prove if there exists a signal $x \in ext(x_{\mathbb{T}})$ such that, $(x, t) \models \varphi$, then $(x_{\mathbb{T}}, t) \models_{\mathrm{f}} \varphi$. Now by definition of $\models$, $\exists t' \in [t+a, t+b]$ such that, $(x, t') \models \varphi_2$ and for all $t'' \in [t, t']$, $(x, t') \models \varphi_1$. Now there are three cases: (i) $t + b < T$: in this case, $(x_{\mathbb{T}}, t) \models_{\mathrm{f}} \varphi_1 \mathsf{U}_{[a,b]} \varphi_2$ by definition of $\models_{\mathrm{f}}$, (ii) $T \leq t' \leq t + b$: in this case, $\forall t'' \in [t, T), (x_{\mathbb{T}}, t'') \models_{\mathrm{f}} \varphi_1$ and hence, $(x_{\mathbb{T}}, t'') \models_{\mathrm{f}} \varphi$, and (iii) $t' < T \leq t + b$: this case is similar to the first case.

The cases for $\varphi = \mathsf{F}_{[a,b]} \psi$ and $\varphi = \mathsf{G}_{[a,b]} \psi$ can be proved similarly using case analysis. □

[3] Following Eisner et al. [80], Ho et al. [118] defined the weak semantics of MTL for the pointwise setting, which we adapt here for the continuous setting.

## 8.2 Problem Formulation

Next, we formally introduce the various aspects of the central problem of the paper.

**Sample.** The input data consists of a set of labeled (piecewise-constant) prefixes. Formally, we rely on a sample $\mathcal{S} = (P, N)$ consisting of a set $P$ of positive prefixes and a set $N$ of negative prefixes such that $P \cap N = \emptyset$.

We say an MTL formula $\varphi$ is *globally-separating* ($\mathsf{G}$-sep, for short) for $\mathcal{S}$ if it satisfies all the positive prefixes at each timepoint and does not satisfy negative prefixes at some timepoint[4]. Formally, given a sample $\mathcal{S}$, we define an MTL formula $\varphi$ to be $\mathsf{G}$-sep for $\mathcal{S}$ if (i) for all $\boldsymbol{x}_{\mathbb{T}} \in P$ and for all $t \in [0, T)$, $(\boldsymbol{x}_{\mathbb{T}}, t) \models_{\mathrm{f}} \varphi$; and (ii) for all $\boldsymbol{y}_{\mathbb{T}} \in N$, there exists $t \in [0, T)$ such that $(\boldsymbol{y}_{\mathbb{T}}, t) \not\models_{\mathrm{f}} \varphi$.

**Future-Reach.** To formalize the lookahead of an MTL formula $\varphi$, we rely on its future-reach $fr(\varphi)$, following [122, 118], which indicates how much of the future is required to determine the satisfaction of $\varphi$. It is defined inductively as follows:

$$
\begin{aligned}
&fr(p) = fr(\neg p) = 0 \\
&fr(\varphi_1 \wedge \varphi_2) = fr(\varphi_1 \vee \varphi_2) = \max(fr(\varphi_1), fr(\varphi_2)) \\
&fr(\varphi_1 \,\mathsf{U}_{[a,b]}\, \varphi_2) = b + \max(fr(\varphi_1), fr(\varphi_2)) \\
&fr(\mathsf{F}_{[a,b]}\, \varphi) = fr(\mathsf{G}_{[a,b]}\, \varphi) = b + fr(\varphi)
\end{aligned}
$$

To highlight that $fr(\varphi)$ quantifies the lookahead of $\varphi$, we observe the following lemma:

**Lemma 14.** *Let $\varphi$ be an MTL formula such that $fr(\varphi) \leq K$ for some $K \in \mathbb{R}^{\geq 0}$. Let $\boldsymbol{x}$ and $\boldsymbol{y}$ be two signals such that $\boldsymbol{x}_{[0,K]} = \boldsymbol{y}_{[0,K]}$. Then, for all $T \in \mathbb{R}^{\geq 0}$, $\boldsymbol{x}_{\mathbb{T}} \models_f \varphi$ if and only if $\boldsymbol{y}_{\mathbb{T}} \models_f \varphi$.*

Intuitively, the above lemma states that a formula with future-reach $\leq K$ cannot distinguish between two signals that are identical up to time $K$. We prove the lemma below.

*Proof of Lemma 14.* We will prove this by induction on the structure of $\varphi$. In particular, we will prove the following:

For any $K$, let $\varphi$ be a formula with $fr(\varphi) \leq K$ and $\boldsymbol{x}$ and $\boldsymbol{y}$ be two signals such that $\boldsymbol{x}_{[0,K]} = \boldsymbol{y}_{[0,K]}$. Then, for all $T \in \mathbb{R}^{\geq 0}$, $\boldsymbol{x}_{\mathbb{T}} \models_{\mathrm{f}} \varphi$ if and only if $\boldsymbol{y}_{\mathbb{T}} \models_{\mathrm{f}} \varphi$.

- For the base case, let $\varphi = p$. Then, $p \in \boldsymbol{x}_{\mathbb{T}}(0)$ and as $\boldsymbol{x}_{[0,K]} = \boldsymbol{y}_{[0,K]}$, $p \in \boldsymbol{y}_{\mathbb{T}}(0)$. Hence, $\boldsymbol{y}_{\mathbb{T}} \models_{\mathrm{f}} p$. This can be similarly seen for the case where $\varphi = \neg p$.

- The proof for the cases where $\varphi = \varphi_1 \vee \varphi_2$ or $\varphi = \varphi_1 \wedge \varphi_2$ can be derived easily.

- Let $\varphi = \mathsf{F}_{[a,b]}\, \varphi_1$. Let us fix a $T$ such that $\boldsymbol{x}_{\mathbb{T}} \models_{\mathrm{f}} \mathsf{F}_{[a,b]}\, \varphi$. If $b \geq t$, then $\boldsymbol{y}_{\mathbb{T}} \models_{\mathrm{f}} \mathsf{F}_{[a,b]}\, \varphi$ trivially. If not, then there exists a timepoint $t \in [a, b]$ such that $(\boldsymbol{x}_{\mathbb{T}}, t) \models_{\mathrm{f}} \varphi_1$. Now, let $\boldsymbol{x}' = \boldsymbol{x}^{[t:]}$ and $\boldsymbol{y}' = \boldsymbol{y}^{[t:]}$ be the signals obtained by shifting the original signals by $-t$. Formally, $\forall t' \in \mathbb{R}^{\geq 0}$, $\boldsymbol{x}'(t') = \boldsymbol{x}(t' + t)$ and $\boldsymbol{y}'(t') = \boldsymbol{y}(t' + t)$. Note that, $\boldsymbol{x}'_{[0,K-t]} = \boldsymbol{y}'_{[0,K-t]}$. Also,

[4] Most stream-based monitors check if the specification holds at every timepoint [26].

$fr(\varphi_1) = fr(\varphi) - b \leq K - b \leq K - t$ and $\boldsymbol{x}'_{[0,K-t]} \models_{\mathrm{f}} \varphi$. Then, following induction hypothesis, $\boldsymbol{y}'_{[0,K-t]} \models_{\mathrm{f}} \varphi_1$ which implies that $(\boldsymbol{y}_{\mathbb{T}}, t) \models_{\mathrm{f}} \varphi_1$. Hence, $\boldsymbol{y}_{\mathbb{T}} \models_{\mathrm{f}} \mathsf{F}_{[a,b]}\, \varphi$. The case where $\varphi = \mathsf{G}_{[a,b]}\, \varphi_1$ can be proved similarly.

- Let $\varphi = \varphi_1\, \mathsf{U}_{[a,b]}\, \varphi_2$. Again, similar to above, fix a $T$ such that $\boldsymbol{x}_{\mathbb{T}} \models_{\mathrm{f}} \varphi_1\, \mathsf{U}_{[a,b]}\, \varphi_2$. Let us first assume that $b \leq T$. Then, $\exists t \in [a,b]$ such that $(\boldsymbol{x}_{\mathbb{T}}, t) \models_{\mathrm{f}} \varphi_2$ and $\forall t' \in [0,t]$, $(\boldsymbol{x}_{\mathbb{T}}, t') \models_{\mathrm{f}} \varphi_1$. Now as $fr(\varphi_1)$ and $fr(\varphi_2)$ are both $\leq K - b \leq K - t$. Hence, again using similar methods as above, one can prove that $(\boldsymbol{y}_{\mathbb{T}}, t) \models_{\mathrm{f}} \varphi_2$ and $\forall t' \in [0,t]$, $(\boldsymbol{y}_{\mathbb{T}}, t') \models_{\mathrm{f}} \varphi_1$. Hence, $\boldsymbol{y}_{\mathbb{T}} \models_{\mathrm{f}} \varphi_1\, \mathsf{U}_{[a,b]}\, \varphi_2$. □

We are now ready to formally introduce the problem of learning an MTL formula. In the problem, we ensure that the MTL formula is efficient for monitoring by allowing the system designer to specify a future-reach bound.

**Problem 9** (LEARNMTL)**.** *Given a sample $\mathcal{S} = (P, N)$ and a future-reach bound $K$, learn an MTL formula $\varphi$ such that*

1. *$\varphi$ is* $\mathsf{G}$*-sep for $\mathcal{S}$;*
2. *$fr(\varphi) \leq K$; and*
3. *for every MTL formula $\varphi'$ such that $\varphi'$ is* $\mathsf{G}$*-sep for $\mathcal{S}$ and $fr(\varphi') \leq K$, $|\varphi| \leq |\varphi'|$.*

Intuitively, the above optimization problem asks to learn a minimal size MTL formula that is $\mathsf{G}$-sep for the input sample and has a future-reach within the input bound. Before we dive into the procedure for finding such an MTL formula, we first study if such an MTL formula even exists.

## 8.3 Existence of a Solution

As alluded to in the introduction, for any given sample $\mathcal{S}$ and future-reach bound $K$, the existence of a prospective formula is not always guaranteed. For an illustration, consider the sample $\mathcal{S}$ with one positive prefix $\boldsymbol{x}_{\mathbb{T}} = \langle (0, \{q\}), (2, \{\}) \rangle$ and one negative prefix $\boldsymbol{y}_{\mathbb{T}} = \langle (0, \{q\}) \rangle$, and domain $\mathbb{T} = [0, 4)$. For this sample, there is no formula $\varphi$ with $fr(\varphi) \leq 1$ that is $\mathsf{G}$-sep. To see this, assume that a prospective formula $\varphi$ exists. Since $\varphi$ must be $\mathsf{G}$-sep, $(\boldsymbol{x}_{\mathbb{T}}, 0) \models \varphi$. Next, observe that, for all time-points $t \in \mathbb{T}$, $\boldsymbol{y}_{\mathbb{T}}$ when restricted to time interval $[t, t+1] \cap \mathbb{T}$ appears identical to $\boldsymbol{x}_{\mathbb{T}}$ when restricted to time interval $[0,1]$ since $\varphi$ has future-reach is 1 (using Lemma 14). Thus, for all time-points $t \in \mathbb{T}$, $(\boldsymbol{y}_{\mathbb{T}}, t) \models \varphi$ violating that $\varphi$ is $\mathsf{G}$-sep.

What we show now is that one can check whether a prospective formula exists by relying on a simple characterization of the inputs $\mathcal{S}$ and $K$. Towards this, we introduce some terminology.

We define an *infix* of a prefix $\boldsymbol{x}_{\mathbb{T}}$ as the restriction of $\boldsymbol{x}_{\mathbb{T}}$ to a specific time interval. Specifically, given two time-points $t_1 \leq t_2 < T$ and a prefix $\boldsymbol{x}_{\mathbb{T}}$, infix $\boldsymbol{x}_{\mathbb{T}}^{[t_1,t_2]}$ is the function $\boldsymbol{x}_{\mathbb{T}}^{[t_1,t_2]} \colon [0, t_2 - t_1] \to 2^{\mathcal{P}}$ such that $\boldsymbol{x}_{\mathbb{T}}^{[t_1,t_2]}(t) = \boldsymbol{x}_{\mathbb{T}}(t + t_1)$ for all $t \in [0, t_2 - t_1]$.

Next, we define a characterization of a sample $\mathcal{S}$ based on the future-reach $K$, which we term as *$K$-infix-separable*. Intuitively, we say $\mathcal{S}$ to be *$K$-infix-separable* if there is a $K$-length

infix $\boldsymbol{y}_{\mathbb{T}}^{[t_1,t_2]}$ for every negative prefix $\boldsymbol{y}_{\mathbb{T}}$ in $\mathcal{S}$ that is not an infix of any positive prefix in $\mathcal{S}$. Formally, $\mathcal{S} = (P, N)$ is *K-infix-separable* if for every negative prefix $\boldsymbol{y}_{\mathbb{T}} \in N$, there exists an infix $\boldsymbol{y}_{\mathbb{T}}^{[t_1,t_2]}$ with $t_2 - t_1 \leq K$ such that $\boldsymbol{y}_{\mathbb{T}}^{[t_1,t_2]} \neq \boldsymbol{x}_{\mathbb{T}}^{[t_1',t_2']}$ for any infix $\boldsymbol{x}_{\mathbb{T}}^{[t_1',t_2']}$ of any positive prefix $\boldsymbol{x}_{\mathbb{T}} \in P$.

We now state the result that enables checking the existence of a solution to Problem 9.

**Lemma 15.** *For a given sample $\mathcal{S}$ and future-reach bound K, there exists an MTL formula $\varphi$ with $fr(\varphi) \leq K$ that is* G*-sep for $\mathcal{S}$ if and only if $\mathcal{S}$ is K-infix-separable.*

*Proof.* ($\Rightarrow$) For the forward direction, consider $\varphi$ to be an MTL formula with $fr(\varphi) \leq K$ that is G-sep for $\mathcal{S}$. Since $\varphi$ is G-sep, for any arbitrary negative prefix, say $\bar{\boldsymbol{y}}_{\mathbb{T}}$, there must be a time-point, say $\bar{t} < T$, such that $(\bar{\boldsymbol{y}}_{\mathbb{T}}, \bar{t}) \not\models \varphi$. If $\bar{t} + K < T$, we show by contradiction that the infix $\bar{\boldsymbol{y}}_{\mathbb{T}}^{[\bar{t},\bar{t}+K]}$ is not an infix in any positive prefix. In particular, if $\bar{\boldsymbol{y}}_{\mathbb{T}}^{[\bar{t},\bar{t}+K]} = \boldsymbol{x}_{\mathbb{T}}^{[t,t+K]}$, then $(\boldsymbol{x}_{\mathbb{T}}, t) \not\models \varphi$ as $\varphi$ cannot distinguish between signals that are identical up to time $K$ (using Lemma 14). If $\bar{t} + K \geq T$, the semantics of MTL being weak, there is an $L < K$ with $\bar{t} + L < T$ such that for any $\boldsymbol{y} \in ext(\boldsymbol{y}_{\mathbb{T}}^{[0,\bar{t}+L]})$, $(\boldsymbol{y}, t) \not\models \varphi$ (using Lemma 13). Once again, we show by contradiction that the infix $\bar{\boldsymbol{y}}_{\mathbb{T}}^{[\bar{t},\bar{t}+L]}$ is not an infix in any positive prefix. In particular, if $\bar{\boldsymbol{y}}_{\mathbb{T}}^{[\bar{t},\bar{t}+L]} = \boldsymbol{x}_{\mathbb{T}}^{[t,t+L]}$, then for all $\boldsymbol{x} \in ext(\boldsymbol{x}_{\mathbb{T}}^{[0,t+L]})$ $(\boldsymbol{x}, t) \not\models \varphi$. Also, for any $\boldsymbol{x} \in ext(\boldsymbol{x}_{\mathbb{T}})$ $(\boldsymbol{x}, t) \not\models \varphi$, meaning $(\boldsymbol{x}_{\mathbb{T}}, t) \not\models \varphi$ (again, using Lemma 13).

($\Leftarrow$) For the other direction, consider $\mathcal{S}$ to be *K-infix-separable*. Using the definition of *K-infix-separable*, for any arbitrary negative prefix, say $\bar{\boldsymbol{y}}_{\mathbb{T}}$, we have an infix $\bar{\boldsymbol{y}}_{\mathbb{T}}^{[t_1,t_2]}$ with $t_2 - t_1 \leq K$ that is not an infix in any positive prefix. We construct a formula $\varphi_{\bar{\boldsymbol{y}}_{\mathbb{T}}}$ that explicitly specifies the propositions appearing in each interval of the infix $\bar{\boldsymbol{y}}_{\mathbb{T}}^{[t_1,t_2]}$ using G and $\wedge$ operators. Observe that $fr(\varphi_{\bar{\boldsymbol{y}}_{\mathbb{T}}}) \leq K$ since $t_2 - t_1 \leq K$ in $\bar{\boldsymbol{y}}_{\mathbb{T}}^{[t_1,t_2]}$. Now, the formula $\neg\varphi_{\boldsymbol{x}_{\mathbb{T}}}$ holds at all time-points in all positive prefixes, while it does not hold at time-point $t_1$ in $\bar{\boldsymbol{y}}_{\mathbb{T}}$. We finally construct the prospective formula as $\varphi = \bigwedge_{\boldsymbol{y}_{\mathbb{T}} \in N} \neg\varphi_{\boldsymbol{y}_{\mathbb{T}}}$ which is G-sep for $\mathcal{S}$ and also, $fr(\varphi) \leq K$. □ □

We now describe an NP algorithm to check whether a sample $\mathcal{S}$ is *K-infix-separable*. The crux of the algorithm is to guess, for each negative prefix $\boldsymbol{y}_{\mathbb{T}}$, an infix $\boldsymbol{y}_{\mathbb{T}}^{[t_1,t_2]}$ with $t_2 - t_1 \leq K$ and then check whether it is an infix of any positive prefix. The procedure of checking involves comparing $\boldsymbol{y}_{\mathbb{T}}^{[t_1,t_2]}$ against the possible infixes of the positive prefixes.

To describe the checking procedure in detail, let $\bar{\boldsymbol{y}}_{\mathbb{T}}^{[t_1,t_2]}$ be an infix of the negative prefix $\bar{\boldsymbol{y}}_{\mathbb{T}}$. Suppose we like to check whether $\bar{\boldsymbol{y}}_{\mathbb{T}}^{[t_1,t_2]}$ is an infix of the positive prefix $\bar{\boldsymbol{x}}_{\mathbb{T}}$. For this, we check $\bar{\boldsymbol{y}}_{\mathbb{T}}^{[t_1,t_2]} = \bar{\boldsymbol{x}}_{\mathbb{T}}^{[t,t+(t_2-t_1)]}$ with only those infixes where the observation timepoints of $\boldsymbol{x}_{\mathbb{T}}$ and $\boldsymbol{y}_{\mathbb{T}}$ coincide. Precisely, we check $\bar{\boldsymbol{y}}_{\mathbb{T}}^{[t_1,t_2]} = \bar{\boldsymbol{x}}_{\mathbb{T}}^{[t,t+(t_2-t_1)]}$ for all those infixes of $\bar{\boldsymbol{x}}_{\mathbb{T}}$ where $t'' - t = t' - t_1$, $t''$ and $t'$ being the timepoints where $\boldsymbol{x}_{\mathbb{T}}$ and $\boldsymbol{y}_{\mathbb{T}}$ have been observed, respectively. This procedure is based on the fact that the changes in an infix occur only at the observation timepoints. Also, this procedure requires polynomial time in the number of observation timepoints of $\bar{\boldsymbol{x}}_{\mathbb{T}}$ and $\bar{\boldsymbol{y}}_{\mathbb{T}}$. We can now perform the procedure for each positive and negative prefix. Overall, we have the following result.

**Lemma 16.** *Given a sample $\mathcal{S}$ and future-reach bound K, checking whether $\mathcal{S}$ is K-infix-separable is in* NP.

**Algorithm 14** Overview of our algorithm

**Input:** Sample $\mathcal{S}$, *fr*-bound $K$

**if** $\mathcal{S}$ is not *K-infix-separable* **then**
  **return** No prospective formula
**end if**
$n \leftarrow 0$
**while** True **do**
  $n \leftarrow n + 1$
  Construct $\Phi^n_{\mathcal{S},K} := \Phi^{str}_{n,\mathcal{S},K} \wedge \Phi^{fr}_{n,\mathcal{S},K} \wedge \Phi^{sem}_{n,\mathcal{S},K} \wedge \Phi^{con}_{n,\mathcal{S},K}$
  **if** $\Phi^n_{\mathcal{S},K}$ is SAT **then**
    Construct $\varphi^v$ from a satisfying assignment $v$
    **return** $\varphi^v$
  **end if**
**end while**

## 8.4 An SMT-based Algorithm

Our algorithm relies on an SMT-based approach inspired by the numerous constraint satisfaction-based approaches for learning temporal logic formulas [169, 48, 195, 8]. Roughly speaking, our algorithm constructs a series of formulas in Linear Real Arithmetic (LRA) and uses an optimized SMT solver to search for the desired solution. We refer to the preliminaries of the thesis for the definition of LRA (Section 2.2.2).

**Algorithm Overview.** Our algorithm constructs a series of LRA formulas $\langle \Phi^n_{\mathcal{S},K} \rangle_{n=1,2,\ldots}$ to facilitate the search for a suitable MTL formula. The formula $\Phi^n_{\mathcal{S},K}$ has the following properties:

1. $\Phi^n_{\mathcal{S},K}$ is satisfiable if and only if there exists an MTL formula $\varphi$ of size $n$ such that $\varphi$ is G-sep for $\mathcal{S}$ and $fr(\varphi) \leq K$.

2. from a satisfying assignment $v$ of $\Phi^n_{\mathcal{S},K}$, one can construct an appropriate MTL formula $\varphi^v$.

In our algorithm, sketched in Algorithm 14, we first check whether $\mathcal{S}$ is *K-infix-separable* (as described in Section 8.3) which informs us whether a prospective formula exists. We now check the satisfiability of $\Phi^n_{\mathcal{S},K}$ for increasing values of size $n$ starting from 1. If $\Phi^n_{\mathcal{S},K}$ is satisfiable for some $n$, then our algorithm constructs a prospective MTL formula $\varphi^v$ from a satisfying assignment $v$ returned by the SMT solver. This algorithm terminates because of checking whether a solution exists apriori, and it returns a minimal formula because of the iterative search through MTL formulas of increasing sizes.

The crux of our algorithm lies in the construction of the formula $\Phi^n_{\mathcal{S},K}$. Internally,

$$\Phi^n_{\mathcal{S},K} := \Phi^{str}_{n,\mathcal{S},K} \wedge \Phi^{fr}_{n,\mathcal{S},K} \wedge \Phi^{sem}_{n,\mathcal{S},K} \wedge \Phi^{con}_{n,\mathcal{S},K}$$

is a conjunction of three subformulas, each with a distinct role. The subformula $\Phi^{str}_{n,\mathcal{S},K}$ encodes the structure of the prospective MTL formula $\varphi$. The subformula $\Phi^{fr}_{n,\mathcal{S},K}$ ensures

that the future-reach of $\varphi$ is less than or equal to $K$. The subformula $\Phi^{sem}_{n,\mathcal{S},K}$ ensures that $\varphi$ is interpreted on the signals in $\mathcal{S}$ using the semantics of MTL. Finally, $\Phi^{con}_{n,\mathcal{S},K}$ ensures that $\varphi$ is $\mathsf{G}$-sep for $\mathcal{S}$. In what follows, we expand on the construction of each of the introduced subformulas. We drop the subscripts $n$, $\mathcal{S}$, and $K$ from the subformulas when clear from the context.

**Structural Constraints.** Following Neider and Gavran [169], we symbolically encode the syntax-DAG of the prospective MTL formula using the formula $\Phi^{str}$. We fix a naming convention for the nodes of the syntax-DAG of an MTL formula, similar to that of LTL (see Section 4.1.2).

Next, to encode a syntax-DAG symbolically, we introduce the following variables[5]: (i) Boolean variables $x_{i,\lambda}$ for $i \in \{1,\dots,n\}$ and $\lambda \in \mathcal{P} \cup \{\neg, \vee, \wedge, \mathsf{U}_I, \mathsf{F}_I, \mathsf{G}_I\}$; (ii) Boolean variables $l_{i,j}$ and $r_{i,j}$ for $i \in \{1,\dots,n\}$ and $j \in \{i+1,\dots,n\}$; (iii) real variables $a_i$ and $b_i$ for $i \in \{1,\dots,n\}$.

The variable $x_{i,\lambda}$ tracks the operator labeled in Node $i$, meaning, $x_{i,\lambda}$ is set to true if and only if Node $i$ is labeled with $\lambda$. The variable $l_{i,j}$ (respectively, $r_{i,j}$) tracks the left (respectively, right) child of Node $i$, meaning, $l_{i,j}$ (respectively, $r_{i,j}$) is set to true if and only if the left (respectively, right) child of Node $i$ is Node $j$. Finally, the variable $a_i$ (respectively, $b_i$) tracks the lower (respectively, upper) bound of the interval $I$ of a temporal operator (that is, operators $\mathsf{U}_I$, $\mathsf{F}_I$ and $\mathsf{G}_I$), meaning that, if $a_i$ (respectively $b_i$) is set to $a \in \mathbb{R}$ (respectively, $b \in \mathbb{R}$), then the lower (respectively, upper) bound of the interval of the operator in Node $i$ is $a$ (respectively, $b$). While we introduce variables $a_i$ and $b_i$ for each node, they become relevant only for the nodes that are labeled with a temporal operator.

We now impose structural constraints on the introduced variables to ensure they encode valid MTL formulas. These constraints are similar to the ones proposed by Neider and Gavran [169] and the ones shown in Formulas 4.4 to 4.7. The subformula $\Phi^{str}$ is a conjunction of all the structural constraints we described. Using a satisfying assignment $v$ of $\Phi^{str}$, one can construct the syntax DAG of a unique MTL formula $\varphi^v$.

**Future-reach Constraints.** To symbolically compute the future-reach of the prospective formula $\varphi$, we encode the inductive definition of the future-reach, as described in Section 8.2 in an LRA formula. To this end, we introduce real variables $f_i$ for $i \in \{1,\dots,n\}$ to encode the future-reach of the subformula $\varphi[i]$. Precisely, $f_i$ is set to $f \in \mathbb{R}$ if and only if $fr(\varphi[i]) = f$.

To ensure the desired meaning of the $f_i$ variables, we impose the following constraints:

$$\bigwedge_{1\le i\le n} x_{i,p} \rightarrow [f_i = 0] \wedge \bigwedge_{\substack{1\le i\le n\\ i<j\le n}} (x_{i,\neg} \wedge l_{i,j}) \rightarrow [f_i = f_j] \wedge$$

$$\bigwedge_{\substack{1\le i\le n\\ i\le j,j'\le n}} ((x_{i,\vee} \vee x_{i,\wedge}) \wedge l_{i,j} \wedge r_{i,j'}) \rightarrow [f_i = \max(f_j, f'_j)] \wedge$$

[5] We include Boolean variables in our LRA formulas since Boolean variables can always be simulated using real variables that are constrained to be either 0 or 1.

$$\bigwedge_{\substack{1\le i\le n\\ i<j\le n}} (x_{i,\mathsf{F}_I} \wedge l_{i,j}) \rightarrow [f_i = f_j + b_i] \wedge \bigwedge_{\substack{1\le i\le n\\ i<j\le n}} (x_{i,\mathsf{G}_I} \wedge l_{i,j}) \rightarrow [f_i = f_j + b_i]$$

The above constraints are based on the definition of future-reach of different MTL operators, as outlined in Section 8.2.

Finally, to enforce that the future-reach of the prospective MTL formula is within $K$, along with the constraints mentioned above, we have $f_1 \le K$in $\Phi^{fr}$.

**Semantic Constraints.** To symbolically check whether the prospective formula is $\mathsf{G}$-sep, we must encode the procedure of checking the satisfaction of an MTL formula into an LRA formula. To this end, we rely on the monitoring procedure devised by Maler et al. [157] for efficiently checking when a signal satisfies an MTL formula. Since our setting is slightly different, we take a brief detour via the description of our adaptation of the monitoring algorithm.

Given an MTL formula $\varphi$ and a signal prefix $\boldsymbol{x}_{\mathbb{T}}$, our monitoring algorithm computes the (lexicographically) ordered set $\mathcal{I}_\varphi(\boldsymbol{x}_{\mathbb{T}}) = \{I_1, \cdots, I_\eta\}$ of *maximal disjoint time intervals* $I_1, \cdots, I_\eta$ where $\varphi$ holds on $\boldsymbol{x}_{\mathbb{T}}$. Mathematically speaking, the following property holds for the set $\mathcal{I}_\varphi(\boldsymbol{x}_{\mathbb{T}})$ we construct:

**Lemma 17.** *Given an MTL formula $\varphi$ and a prefix $\boldsymbol{x}_{\mathbb{T}}$, for all $t \in \mathbb{T}$, $(\boldsymbol{x}_{\mathbb{T}}, t) \models_f \varphi$ if and only if $t \in I$ for some $I \in \mathcal{I}_\varphi(\boldsymbol{x}_{\mathbb{T}})$.*

In our monitoring algorithm, we compute the set $\mathcal{I}_\varphi(\boldsymbol{x}_{\mathbb{T}})$ inductively on the structure of the formula $\varphi$. To describe the induction, we use the notation $\mathcal{I}^{\cup}_\varphi(\boldsymbol{x}_{\mathbb{T}}) = \bigcup_{I\in\mathcal{I}_\varphi(\boldsymbol{x}_{\mathbb{T}})} I$ to denote the union of the intervals in $\mathcal{I}_\varphi(\boldsymbol{x}_{\mathbb{T}})$. For the base case, we compute $\mathcal{I}_p(\boldsymbol{x}_{\mathbb{T}})$ for every $p \in \mathcal{P}$ by accumulating the timepoints $t \in [0, T)$ where $(\boldsymbol{x}_{\mathbb{T}}, t) \models_{\mathrm{f}} p$ into maximal disjoint time intervals. In the inductive step, we exploit the relations presented in Table 8.1 for the different MTL operators. In the table, $[t_1, t_2) \ominus [a, b] = [t_1 - b, t_2 - a) \cap \mathbb{T}$ and $\mathcal{I}^c = \mathbb{T} - \mathcal{I}$. While the table presents the computation of $\mathcal{I}^{\cup}_\varphi(\boldsymbol{x}_{\mathbb{T}})$, we can obtain $\mathcal{I}_\varphi(\boldsymbol{x}_{\mathbb{T}})$ by

TABLE 8.1: The relations for inductive computation of $\mathcal{I}^{\cup}_\varphi(\boldsymbol{x}_{\mathbb{T}})$.

$$\mathcal{I}^{\cup}_{\neg p}(\boldsymbol{x}_{\mathbb{T}}) = \left(\mathcal{I}^{\cup}_p(\boldsymbol{x}_{\mathbb{T}})\right)^c$$

$$\mathcal{I}^{\cup}_{\varphi_1\vee\varphi_2}(\boldsymbol{x}_{\mathbb{T}}) = \mathcal{I}^{\cup}_{\varphi_1}(\boldsymbol{x}_{\mathbb{T}}) \cup \mathcal{I}^{\cup}_{\varphi_2}(\boldsymbol{x}_{\mathbb{T}})$$

$$\mathcal{I}^{\cup}_{\varphi_1\wedge\varphi_2}(\boldsymbol{x}_{\mathbb{T}}) = \mathcal{I}^{\cup}_{\varphi_1}(\boldsymbol{x}_{\mathbb{T}}) \cap \mathcal{I}^{\cup}_{\varphi_2}(\boldsymbol{x}_{\mathbb{T}})$$

$$\mathcal{I}^{\cup}_{\mathsf{F}_{[a,b]}\varphi}(\boldsymbol{x}_{\mathbb{T}}) = \left(\bigcup_{I\in\mathcal{I}_\varphi(\boldsymbol{x}_{\mathbb{T}})} I \ominus [a, b]\right) \cup [T - b, T)$$

$$\mathcal{I}^{\cup}_{\mathsf{G}_{[a,b]}\varphi}(\boldsymbol{x}_{\mathbb{T}}) = \left(\bigcup_{I\in(\mathcal{I}_\varphi(\boldsymbol{x}_{\mathbb{T}}))^c} I \ominus [a, b]\right)^c \cup [T - a, T)$$

$$\mathcal{I}^{\cup}_{\varphi\,\mathsf{U}_{[a,b]}\,\psi}(\boldsymbol{x}_{\mathbb{T}}) = \bigcup_{I_\varphi\in\mathcal{I}_\varphi(\boldsymbol{x}_{\mathbb{T}})} \bigcup_{I_\psi\in\mathcal{I}_\psi(\boldsymbol{x}_{\mathbb{T}})} \Big(\big((I_\varphi \cap I_\psi) \ominus [a, b]\big) \cap I_\varphi\Big) \cup I_{\mathbb{T}},$$

$$\text{where } I_{\mathbb{T}} = \begin{cases} [\max(T - b, t), T), & \text{if } \exists t \text{ s.t. } [t, T) \in \mathcal{I}_\varphi(\boldsymbol{x}_{\mathbb{T}}) \\ \emptyset, & \text{otherwise} \end{cases}$$

simply partitioning $\mathcal{I}^{\cup}_\varphi(\boldsymbol{x}_{\mathbb{T}})$ into maximal disjoint intervals.

For an illustration, we consider the example from the introduction and compute $\mathcal{I}_{\varphi_2}(u_1)$ where $u_1$ is the first positive prefix, $\varphi_2 = p \vee \mathsf{F}_{[0,1]}\, q$, and $\mathbb{T} = [0,6)$. First, we have $\mathcal{I}_p(u_1) = \{[0,2),[3,6)\}$ and $\mathcal{I}_q(u_1) = \{[0,1),[2,4)\}$. Now, we can compute $\mathcal{I}_{\mathsf{F}_{[0,1]}\, q}(u_1) = \{[0,4),[5,6)\}$ and then $\mathcal{I}_{p \vee \mathsf{F}_{[0,1]}\, q}(u_1) = \{[0,6)\}$. Now, we formally prove Lemma 17 that proves the correctness of our construction of $\mathcal{I}_\varphi(\boldsymbol{x}_\mathbb{T})$ given a prefix $\boldsymbol{x}_\mathbb{T}$.

*Proof of Lemma 17.* We prove both directions together by induction on the structure of the formula $\varphi$.

For the base case, one can check that for all $t \in [0,T)$, $t \in \mathcal{I}_p(\boldsymbol{x}_\mathbb{T})$ if and only if $t \in I$ for some $I \in \mathcal{I}_\varphi(\boldsymbol{x}_\mathbb{T})$ by construction. The proof for the $\neg$ operator and the boolean connectives $\wedge$ and $\vee$ follow from the correctness of the construction in the work of Maler et al. [157]. Here, we provide the proof for the $\mathsf{F}_{[a,b]}$ operator. The proofs for the $\mathsf{U}_{[a,b]}$ and $\mathsf{G}_{[a,b]}$ can be obtained similarly.

Let $\varphi = \mathsf{F}_{[a,b]}\, \psi$. To show the forward direction, let $t \in I$ for some $I \in \mathcal{I}_\varphi(\boldsymbol{x}_\mathbb{T})$. We now need to prove $(\boldsymbol{x}_\mathbb{T}, t) \models_\mathrm{f} \mathsf{F}_{[a,b]}\, \psi$. In particular, $t \in \mathcal{I}^{\cup}_\varphi(\boldsymbol{x}_\mathbb{T})$ by definition, that is, $t \in \left(\bigcup_{I \in \mathcal{I}_\psi(\boldsymbol{x}_\mathbb{T})} I \ominus [a,b]\right) \cup [T-b,T)$. There are two cases: (i) $t \in [T-b,T)$: in this case, $t + b \geq T$ and by definition of $\models_\mathrm{f}$, $(\boldsymbol{x}_\mathbb{T}, t) \models_\mathrm{f} \varphi$, or (ii) $t \in \left(\bigcup_{I \in \mathcal{I}_\psi(\boldsymbol{x}_\mathbb{T})} I \ominus [a,b]\right)$: Fix the interval $I' = [t_1,t_2) \in \mathcal{I}_\psi(\boldsymbol{x}_\mathbb{T})$ such that, $t \in (I' \ominus [a,b])$. By induction hypothesis, for all $t' \in I'$, $(\boldsymbol{x}_\mathbb{T}, t') \models_\mathrm{f} \psi$. Now, $t < t_2 - a \implies t + a < t_2$ and $t \geq t_1 - b \implies t + b \geq t_1$. Hence, $I' = [t_1,t_2) \supset [t+a, t+b]$. Hence, $\exists t' \in [t+a, t+b]$ such that, $(\boldsymbol{x}_\mathbb{T}, t') \models_\mathrm{f} \psi$ and henceforth, $(\boldsymbol{x}_\mathbb{T}, t) \models_\mathrm{f} \varphi$.

For the other direction, we assume that $(\boldsymbol{x}_\mathbb{T}, t) \models_\mathrm{f} \mathsf{F}_{[a,b]}\, \psi$ and prove that $t \in I$ for some $I \in \mathcal{I}_\varphi(\boldsymbol{x}_\mathbb{T})$. In particular, we show that $t \in \mathcal{I}^{\cup}_\varphi(\boldsymbol{x}_\mathbb{T}) = \left(\bigcup_{I \in \mathcal{I}_\psi(\boldsymbol{x}_\mathbb{T})} I \ominus [a,b]\right) \cup [T-b,T)$. The rest of the argument follows from the fact that $\mathcal{I}_\varphi(\boldsymbol{x}_\mathbb{T})$ is obtained by taking the maximal disjoint intervals of $\mathcal{I}^{\cup}_\varphi(\boldsymbol{x}_\mathbb{T})$. Now, by definition of $\models_\mathrm{f}$, there are two possibilities: (i) $t + b \geq T$: then, $t \in [T-b,T)$ and, hence, $t \in \mathcal{I}^{\cup}_\varphi(\boldsymbol{x}_\mathbb{T})$, or (ii) $\exists t' \in [t+a, t+b]$ such that, $(\boldsymbol{x}_\mathbb{T}, t') \models_\mathrm{f} \psi$. Now, by induction hypothesis, $t' \in I$ for some $I \in \mathcal{I}_\psi(\boldsymbol{x}_\mathbb{T})$. Let $I = [t_1,t_2)$. Now, $t_2 - a > t' - a \geq t$ and $t_1 - b \leq t' - b \leq t$. This implies that, $t \in [t_1 - b, t_2 - a) = (I \ominus [a,b])$ which proves that, $t \in \mathcal{I}^{\cup}_\varphi(\boldsymbol{x}_\mathbb{T})$. □

In the monitoring algorithm, the number of maximal intervals required in $\mathcal{I}_\varphi(\boldsymbol{x}_\mathbb{T})$ is upper-bounded by $\mathcal{M} = \mu|\varphi|$, where $\mu = \max(\{|\mathcal{I}_p(\boldsymbol{x}_\mathbb{T})| \mid p \in \mathcal{P}\})$, as also observed by Maler et al. [157]. The computation of this bound can also be done inductively on the structure of $\varphi$.

Now, in the subformula $\Phi^{sem}$, we symbolically encode the set $\mathcal{I}_\varphi(\boldsymbol{x}_\mathbb{T})$ of our prospective MTL formula $\varphi$. To this end, we introduce variables $t^l_{i,m,s}$ and $t^r_{i,m,s}$ where $i \in \{1,\ldots,n\}$, $m \in \{1,\ldots,\mathcal{M}\}$, and $s \in \{1,\ldots,|\mathcal{S}|\}$, $s$ being an identifier for the $s^{th}$ prefix $\boldsymbol{x}^s_\mathbb{T}$ in $\mathcal{S}$. The variables $t^l_{i,m,s}$ and $t^r_{i,m,s}$ encode the $m^{th}$ interval of $\mathcal{I}_{\varphi[i]}(\boldsymbol{x}^s_\mathbb{T})$ for the subformula $\varphi[i]$. In other words, $t^l_{i,m,s} = t_1$ and $t^r_{i,m,s} = t_2$ if and only if $[t_1,t_2)$ is the $m^{th}$ interval of $\mathcal{I}_{\varphi[i]}(\boldsymbol{x}^s_\mathbb{T})$.

Now, to ensure that the variables $t^l_{i,m,s}$ and $t^r_{i,m,s}$ have their desired meaning, we introduce constraints for each operator based on the relations defined in Table 8.1. We now present these constraints for the different MTL operators.

For the $\neg$ operator, we have the following constraints:

$$\bigwedge_{\substack{1\le i\le n\\ i<j\le n}} x_{i,\neg} \wedge l_{i,j} \rightarrow \Big[ \bigwedge_{1\le s\le|\mathcal{S}|} comp_s(i,j)\Big],$$

where, for every $\boldsymbol{x}^s_{\mathbb{T}}$ in $\mathcal{S}$, $comp_s(i,j)$ encodes that $\mathcal{I}^{\cup}_{\varphi[i]}(\boldsymbol{x}^s_{\mathbb{T}})$ is the complement of $\mathcal{I}^{\cup}_{\varphi[j]}(\boldsymbol{x}^s_{\mathbb{T}})$ . We construct $comp_s(i,j)$ as follows:

$$\texttt{ite}(t^l_{j,1,s} = 0, \tag{8.1}$$

$$\bigwedge_{1\le m\le\mathcal{M}-1} t^l_{i,m,s} = t^r_{j,m,s} \wedge t^r_{i,m,s} = t^l_{j,m+1,s}, \tag{8.2}$$

$$t^l_{i,1,s} = 0 \wedge t^r_{i,1,s} = t^l_{j,1,s} \wedge \tag{8.3}$$

$$\bigwedge_{1\le m\le\mathcal{M}-1} t^l_{i,m+1,s} = t^r_{j,m,s} \wedge t^r_{i,m+1,s} = t^l_{j,m+1,s}),$$

where `ite` is a syntactic sugar for the "if-then-else" construct over LRA formulas, which is standard in many SMT solvers. Here, Condition 8.1 checks whether the left bound of the first interval of $\mathcal{I}_{\varphi[j]}(\boldsymbol{x}^s_{\mathbb{T}})$, encoded by $t^l_{j,1,s}$, is 0. If that holds, as specified by Formula 8.2, the left bound of the first interval of $\mathcal{I}_{\varphi[i]}(\boldsymbol{x}^s_{\mathbb{T}})$, encoded by $t^l_{1,i,s}$, will be the right bound of the first interval of $\mathcal{I}_{\varphi[j]}(\boldsymbol{x}^s_{\mathbb{T}})$, encoded $t^r_{1,j,s}$ and so on. If Condition 8.1 does not hold, as specified by Formula 8.3, the left bound of the first interval of $\mathcal{I}_{\varphi[i]}(\boldsymbol{x}^s_{\mathbb{T}})$ will start with 0, and so on.

As an example, for a prefix $\boldsymbol{x}^s_{\mathbb{T}}$ and $\mathbb{T} = [0,7)$, let $\mathcal{I}_{\varphi[j]}(\boldsymbol{x}^s_{\mathbb{T}}) = \{[0,4),[6,7)\}$. Then, Formula 8.2 ensures that $\mathcal{I}_{\varphi[i]}(\boldsymbol{x}^s_{\mathbb{T}}) = \{[4,6)\}$[6]. Conversely, if $\mathcal{I}_{\varphi[j]}(\boldsymbol{x}^s_{\mathbb{T}}) = \{[1,4),[6,7)\}$, then Formulas 8.3 ensures that $\mathcal{I}_{\varphi[i]}(\boldsymbol{x}^s_{\mathbb{T}}) = \{[0,1),[4,6)\}$.

For the $\vee$ operator, we have the following constraint:

$$\bigwedge_{\substack{1\le i\le n\\ i<j,j'\le n}} x_{i,\vee} \wedge l_{i,j} \wedge r_{i,j'} \rightarrow \Big[ \bigwedge_{1\le s\le|\mathcal{S}|} union_s(i,j,j')\Big],$$

where, for every $\boldsymbol{x}^s_{\mathbb{T}}$ in $\mathcal{S}$, $union_s(i,j,j')$ encodes that $\mathcal{I}_{\varphi[i]}(\boldsymbol{x}^s_{\mathbb{T}})$ consists of the maximal disjoint intervals obtained from the union of the intervals in $\mathcal{I}_{\varphi[j]}(\boldsymbol{x}^s_{\mathbb{T}})$ and $\mathcal{I}_{\varphi[j']}(\boldsymbol{x}^s_{\mathbb{T}})$. We construct $union_s(i,j,j')$ as follows:

$$\bigwedge_{\sigma\in[l,r]}\bigwedge_{1\le m\le\mathcal{M}}\left(\bigvee_{1\le m'\le\mathcal{M}}(t^\sigma_{i,m,s} = t^\sigma_{j,m',s}) \vee \bigvee_{1\le m'\le\mathcal{M}}(t^\sigma_{i,m,s} = t^\sigma_{j',m',s})\right) \wedge \tag{8.4}$$

$$\bigwedge_{\sigma\in[l,r]}\bigwedge_{1\le m\le\mathcal{M}}\left(\bigvee_{1\le m'\le\mathcal{M}}(t^\sigma_{i,m,s} = t^\sigma_{j,m',s}) \iff \bigwedge_{1\le m''\le\mathcal{M}}(t^\sigma_{j,m',s} \notin I_{j',m'',s})\right) \wedge \tag{8.5}$$

$$\bigwedge_{\sigma\in[l,r]}\bigwedge_{1\le m\le\mathcal{M}}\left(\bigvee_{1\le m'\le\mathcal{M}}(t^\sigma_{i,m,s} = t^\sigma_{j',m',s}) \iff \bigwedge_{1\le m''\le\mathcal{M}}(t^\sigma_{j',m',s} \notin I_{j,m'',s})\right), \tag{8.6}$$

where $I_{k,m,s}$ denotes the interval encoded by bounds $t^l_{k,m,s}$ and $t^r_{k,m,s}$[7]. Here, Formula 8.4

[6] $|\mathcal{I}_{\varphi[i]}(\boldsymbol{x}^s_{\mathbb{T}})|$ may differ for different subformulas $\varphi[i]$; we address this at the end of this section.

[7] In LRA, $t \notin [t_1,t_2)$ can be encoded as $t < t_1 \vee t \ge t_2$.

states that the left (respectively, right) bound of each interval of $\mathcal{I}_{\varphi[i]}(x^s_{\mathbb{T}})$, encoded by $t^l_{i,m,s}$ (respectively, $t^r_{i,m,s}$) corresponds to one of the left (respectively, right) bounds of the intervals in $\mathcal{I}_{\varphi[j]}(x^s_{\mathbb{T}})$ or in $\mathcal{I}_{\varphi[j']}(x^s_{\mathbb{T}})$. Then, Formula 8.5 states that for each interval $I$ in $\mathcal{I}_{\varphi[j]}(x^s_{\mathbb{T}})$, the left (respectively, right) bound of $I$ should appear as the left (respectively, right) bound of some interval in $\mathcal{I}_{\varphi[i]}(x^s_{\mathbb{T}})$ if and only if the left (respectively, right) bound of $I$ is not included in any of the intervals in $\mathcal{I}_{\varphi[j']}(x^s_{\mathbb{T}})$. Formula 8.6 mimics the statement made by Formula 8.5 but for the bounds of the intervals in $\mathcal{I}_{\varphi[j']}(x^s_{\mathbb{T}})$.

For an illustration, assume that $\mathcal{I}_{\varphi[j]}(x^s_{\mathbb{T}}) = \{[1,4),[6,7)\}$ and $\mathcal{I}_{\varphi[j']}(x^s_{\mathbb{T}}) = \{[3,5),[6,7)\}$ for a prefix $x^s_{\mathbb{T}}$ and $T = 7$. Now, if $\varphi[i] = \varphi[j] \vee \varphi[j']$, then $\mathcal{I}_{\varphi[i]}(x^s_{\mathbb{T}}) = \{[1,5),[6,7)\}$ based on the relation for $\vee$-operator in Table 8.1. Observe that all the bounds of the intervals in $\mathcal{I}_{\varphi[i]}(x^s_{\mathbb{T}})$, that is, 1, 5, 6, and 7, are present as the bounds of the intervals in either $\mathcal{I}_{\varphi[j]}(x^s_{\mathbb{T}})$ or $\mathcal{I}_{\varphi[j']}(x^s_{\mathbb{T}})$. This fact is in accordance with Formula 8.4. Also, the right bound of $[1,4)$ in $\mathcal{I}_{\varphi[j]}(x^s_{\mathbb{T}})$ does not appear as a bound of any intervals in $\mathcal{I}_{\varphi[i]}(x^s_{\mathbb{T}})$, as it is included in an interval in $\mathcal{I}_{\varphi[j']}(x^s_{\mathbb{T}})$, that is, $4 \in [3,5)$. This is in accordance with Formula 8.5.

Next, for the $\mathsf{F}_I$-operator where $I$ is encoded using $a_i$ and $b_i$, we have the following constraint:

$$\bigwedge_{\substack{1\le i\le n\\ i<j\le n}} x_{i,\mathsf{F}_I} \wedge l_{i,j} \rightarrow \Big[ \bigwedge_{1\le s\le |\mathcal{S}|} union'_s(i,k,k) \wedge \ominus_s^{[a_i,b_i]}(k,j)\Big].$$

based on the relation for the $\mathsf{F}_{[a,b]}$ operator in Table 8.1. We here rely on an intermediate set of intervals $\tilde{\mathcal{I}}_k$ encoded using some auxiliary variables $\tilde{t}^l_{k,m,s}$ and $\tilde{t}^r_{k,m,s}$ where $m \in \{1,\ldots,\mathcal{M}\}$ and $s \in \{1,\ldots,|\mathcal{S}|\}$. Also, we use the formula $\ominus_s^{[a_i,b_i]}(k,j)$ to encode that the intervals in $\tilde{\mathcal{I}}_k$ can be obtained by performing $I \ominus [a,b]$ to each interval $I$ in $\mathcal{I}_{\varphi[j]}(x^s_{\mathbb{T}})$, where $a_i = a$ and $b_i = b$. Finally, the formula $union'(i,k,k)$ encodes that $\mathcal{I}_{\varphi[i]}(x^s_{\mathbb{T}})$ consists of the maximal disjoint intervals obtained from the union of the intervals in $\tilde{\mathcal{I}}_k$ and $\{[T-b,T)\}$.

The construction of $union'(i,k,k)$ is similar to that of $union(i,j,j')$, in the sense that the constraints involved are similar to Formulas 8.4 to 8.6. For $\ominus_s^{[a_i,b_i]}(k,j)$, we have the following constraint:

$$\bigwedge_{1\le m\le \mathcal{M}-1} \big[\tilde{t}^l_{k,m,s} = \max\{0,\left(t^l_{j,m,s} - b_i\right)\} \wedge \tilde{t}^r_{k,m,s} = \max\{0,\left(t^r_{j,m,s} - a_i\right)\}\big] \tag{8.7}$$

As an example, consider $\mathcal{I}_{\varphi[j]}(x^s_{\mathbb{T}}) = \{[1,4),[6,7)\}$ for a prefix $x^s_{\mathbb{T}}$ and $T = 7$. Now, if $\varphi[i] = \mathsf{F}_{[1,4]}\,\varphi[j]$, then first we have $\tilde{\mathcal{I}}_k = \{[0,3),[2,6)\}$ based on Formula 8.7 [8]. Next, we have $\mathcal{I}_{\varphi[i]}(x^s_{\mathbb{T}}) = \{[0,7)\}$ which consists of the maximal disjoint intervals from $\tilde{\mathcal{I}}_k \cup \{[T-4,T)\} = \{[0,3),[2,6),[3,7)\}$ using $union'(i,k,k)$.

[8] While the intervals in $\tilde{\mathcal{I}}_k$ may not be disjoint, $union'(i,k,k)$ ensures that $\mathcal{I}_{\varphi[i]}(x^s_{\mathbb{T}})$ consists of only maximal disjoint intervals.

For the $\mathsf{U}_I$ operator, we have the following constraint:

$$\bigwedge_{\substack{1\le i\le n\\ i<j,j'\le n}} x_{i,\mathsf{U}_I} \wedge l_{i,j} \wedge r_{i,j'} \rightarrow \Big[ \bigwedge_{1\le s\le|\mathcal{S}|} intersection_s(k_1,j,j') \wedge \ominus_s^{[a_i,b_i]}(k_2,k_1) \\ \wedge cond{-}int_s(k_3,k_2,j) \wedge union_s(i,k_3,k_3)\Big]$$

Here, we introduce three intermediate set of intervals $\tilde{\mathcal{I}}_{k_1}$, $\tilde{\mathcal{I}}_{k_2}$ and $\tilde{\mathcal{I}}_{k_3}$ encoded using auxiliary variables $\tilde{t}^l_{k_i,m,s}$ and $\tilde{t}^r_{k_i,m,s}$ where $i \in \{1,2,3\}$, $m \in \{1,\ldots,\mathcal{M}\}$ and $s \in \{1,\ldots,|\mathcal{S}|\}$. Similar to the constraints for the $\vee$ operator, we denote an interval in $\tilde{\mathcal{I}}_{k_i}$ as $I_{k_i,m,s}$ where, $I_{k_i,m,s} = [\tilde{t}^l_{k_i,m,s}, \tilde{t}^r_{k_i,m,s})$. Now, $intersection_s(k_1,j,j')$ encodes that $\tilde{\mathcal{I}}_{k_1}$ consists of the maximal disjoint intervals obtained from the intersection of the intervals in $\mathcal{I}_{\varphi[j]}(\boldsymbol{x}^s_{\mathbb{T}})$ and $\mathcal{I}_{\varphi[j']}(\boldsymbol{x}^s_{\mathbb{T}})$. Note that the intersection can be achieved using the $union_s$ and the $comp_s$ operators based on De Morgan's law, $A \cap B = (A^c \cup B^c)^c$. Then, $\ominus_s^{[a_i,b_i]}(k_2,k_1)$ denotes that the intervals in $\tilde{\mathcal{I}}(k_2)$ can be obtained by performing $I \ominus [a,b]$ to each interval in $\tilde{\mathcal{I}}_{k_1}$ using constraint 8.7.

Next, the operator $cond - int_s(k_3,k_2,j)$ denotes that the $m^{th}$ interval in $\tilde{\mathcal{I}}(k_3)$ ($I_{k_3,m,s}$) is obtained by taking the intersection of the $m^{th}$ interval in $\tilde{\mathcal{I}}(k_2)$ ($I_{k_2,m,s}$) and the $m'^{th}$ interval in $\mathcal{I}_{\varphi[j]}(\boldsymbol{x}^s_{\mathbb{T}})$ ($I_{j,m',s}$) such that, $I_{k_1,m,s}$ ($I_{k_2,m,s} = I_{k_1,m,s} \ominus [a,b]$, by construction) is a subset of $I_{j,m',s}$. This can be achieved by encoding $cond - int_s(k_3,k_2,j)$ as the following constraint:

$$\bigwedge_{1\le m\le\mathcal{M}} \bigwedge_{1\le m'\le\mathcal{M}} (I_{k_1,m,s} \subseteq I_{j,m',s}) \rightarrow I_{k_3,m,s} = I_{k_2,m,s} \cap I_{j,m',s}$$

Note that the subset check and the intersection of two intervals both allow simple encodings in LRA. Finally, the formula $union_s(i,k_3,k_3)$ encodes that $\mathcal{I}_{\varphi[i]}(\boldsymbol{x}^s_{\mathbb{T}})$ consists of the maximal disjoint intervals obtained from the union of the intervals in $\tilde{\mathcal{I}}_{k_3}$.

For an illustration, assume that $\mathcal{I}_{\varphi[j]}(\boldsymbol{x}^s_{\mathbb{T}}) = \{[1,3),[5,8)\}$ and $\mathcal{I}_{\varphi[j']}(\boldsymbol{x}^s_{\mathbb{T}}) = \{[4,6),[7,9)\}$ for a prefix $\boldsymbol{x}^s_{\mathbb{T}}$ and $T = 9$. Now, let $\varphi[i] = \varphi[j]\,\mathsf{U}_{[0,3]}\,\varphi[j']$. Then, $\mathcal{I}_{\varphi[i]}(\boldsymbol{x}^s_{\mathbb{T}}) = \{[5,8)\}$ using the computation in Table 8.1.

Note that, following the constraint, $\tilde{\mathcal{I}}_{k_1} = \{[5,6),[7,8)\}$ after taking the intersection of $\mathcal{I}_{\varphi[j]}(\boldsymbol{x}^s_{\mathbb{T}})$ and $\mathcal{I}_{\varphi[j']}(\boldsymbol{x}^s_{\mathbb{T}})$. Then, the Minkowski minus results into the set of intervals $\tilde{\mathcal{I}}_{k_2} = \{[2,6),[4,8)\}$ with $a = 0$ and $b = 3$. The conditional intersection of $\tilde{\mathcal{I}}_{k_2}$ and $\mathcal{I}_{\varphi[j}(\boldsymbol{x}^s_{\mathbb{T}})$ produces the set of intervals $\tilde{\mathcal{I}}_{k_3} = \{[5,6),[5,8)\}$. Note that this is because both the intervals in $\tilde{\mathcal{I}}_{k_1}$ are subsets of the interval $[5,8)$ in $\mathcal{I}_{\varphi[j]}(\boldsymbol{x}^s_{\mathbb{T}})$ and not of $[1,3)$, and we intersect the intervals in $\tilde{\mathcal{I}}_{k_2}$ with only $[5,8)$. Finally, the operator $union_s$ on $\tilde{\mathcal{I}}_{k_3}$ results in $\mathcal{I}_{\varphi[i]}(\boldsymbol{x}^s_{\mathbb{T}})$ to be $\{[5,8)\}$ that complies with the actual semantics of the $\mathsf{U}_I$ operator. It can be also checked that taking a normal intersection instead of the conditional one would have wrongly resulted in $\mathcal{I}_{[}(\boldsymbol{x}^s_{\mathbb{T}})\varphi[i]]$ to be $\{[2,3),[5,8)\}$ that depicts the intricacy in computing the satisfaction intervals for $\mathsf{U}_I$ as shown in Figure 3(a) in Maler et al. [157].

For the $\wedge$-operator, we have the following constraint:

$$\bigwedge_{\substack{1\le i\le n\\ i<j,j'\le n}} x_{i,\wedge} \wedge l_{i,j} \wedge r_{i,j'} \rightarrow \Big[ \bigwedge_{1\le s\le|\mathcal{S}|} intersection_s(i,j,j')\Big],$$

This encodes the relation for $\wedge$ operator as described in Table 8.1, that is, encoding the fact that the set $\mathcal{I}^{\cup}_{\varphi[i]}(\boldsymbol{x}^s_{\mathbb{T}})$ contains maximal disjoint intervals of *intersection* of $\mathcal{I}^{\cup}_{\varphi[j]}(\boldsymbol{x}^s_{\mathbb{T}})$ and $\mathcal{I}^{\cup}_{\varphi[j']}(\boldsymbol{x}^s_{\mathbb{T}})$.

For the $\mathsf{G}_I$ operator where $I$ is encoded using $a_i$, $b_i$ we have the following constraint:

$$\bigwedge_{\substack{1\le i\le n\\ i<j\le n}} x_{i,\mathsf{G}_I} \wedge l_{i,j} \rightarrow \Big[\bigwedge_{1\le s\le|\mathcal{S}|} union''_s(i,k',k') \wedge comp_s(k',k) \wedge \ominus_s^{[a_i,b_i]}(k,j)\Big].$$

based on the relation for the $\mathsf{G}_{[a,b]}$ operator in Table 8.1. Similar to the encoding of $\mathsf{F}_I$ operator, we rely on an intermediate set of intervals $\tilde{\mathcal{I}}_k$ and $\tilde{\mathcal{I}}_{k'}$ encoded using some auxiliary variables. Also, we use the formula $\ominus_s^{[a_i,b_i]}(k,j)$ to encode that the intervals in $\tilde{\mathcal{I}}_k$ can be obtained by performing $I \ominus [a,b]$ to each interval $I$ in $\mathcal{I}_{\varphi[j]}(\boldsymbol{x}^s_{\mathbb{T}})$, where $a_i = a$ and $b_i = b$. Then $comp_s(k',k)$ encodes that $\tilde{\mathcal{I}}_{k'}$ is the complement of $\tilde{\mathcal{I}}_k$. Finally, the formula $union''(i,k',k')$ encodes that $\mathcal{I}_{\varphi[i]}(\boldsymbol{x}^s_{\mathbb{T}})$ consists of the maximal disjoint intervals obtained by taking the union of the complement of $\mathcal{I}^{\cup}_{\varphi[j]}(\boldsymbol{x}^s_{\mathbb{T}})$ and $\{[T-a,T)\}$.

Similar to $union'$ in the semantic constraints for $\mathsf{F}_I$ operator, the construction of $union''(i,k,k)$ matches that of $union(i,j,j')$, in that, the constraints involved are similar to Formulas 8.4 to 8.6.

We now assert the correctness of the formulas encoding the set operations as follows:

**Lemma 18.** *The formulas $comp_s(i,j)$, $union_s(i,j,j')$, $\ominus_s^{[a_i,b_i]}(k,j)$, $intersection_s(i,j,j')$and $cond{-}int_s(i,j,j')$ correctly encode the complement, union, $\ominus$, intersection and conditional intersection operations on a set of intervals, respectively*

To offer a glimpse into the proof of correctness for the introduced formulas, we present some of them here.

**Lemma 19** (Correctness of $union_s$)**.** *Let $v$ be a satisfying assignment of $union_s(i,j,j')$. Then, the set $\mathcal{I}_i = \{[v(t^l_{i,1,s}), v(t^r_{i,1,s})), \ldots, [v(t^l_{i,m,s}), v(t^r_{i,m,s}))\}$ consists of the maximal disjoint intervals of the union of $\mathcal{I}_j = \{[v(t^l_{j,1,s}), v(t^r_{j,1,s})), \ldots, [v(t^l_{j,m,s}), v(t^r_{j,m,s}))\}$ and $\mathcal{I}_{j'} = \{[v(t^l_{j',1,s}), v(t^r_{j',1,s})), \ldots, [v(t^l_{j',m,s}), v(t^r_{j',m,s}))\}$.*

*Proof.* For simplicity of the proof, we name $v(t^{\sigma}_{\kappa,m})$ as $\tau^{\sigma}_{\kappa,m}$ for $\sigma \in \{l,r\}$ and $\kappa \in \{i,j,j'\}$, and $[\tau^l_{\kappa,m}, \tau^r_{\kappa,m})$ as $\Gamma_{\kappa,m}$ for $\kappa \in \{i,j,j'\}$. Note that we drop the identifier $s$ representing the prefix since the prefix is fixed throughout the proof.

For the forward direction, we show that any timepoint $t \in \Gamma_{i,m}$ belongs to some $\Gamma_{j,m'} \in \mathcal{I}_j$ or some $\Gamma_{j',m''} \in \mathcal{I}_{j'}$. Towards contradiction, we assume that $t \notin \Gamma_{j,m'}$ for any $\Gamma_{j,m'} \in \mathcal{I}_j$ and $t \notin \Gamma_{j',m''}$ for any $\Gamma_{j',m''} \in \mathcal{I}_{j'}$. Now, based on Formula 8.4, both $\tau^l_{i,m}$ and $\tau^r_{i,m}$ appear in some intervals in $\mathcal{I}_j$ and $\mathcal{I}_{j'}$ as left and right bound, respectively. We consider two cases based on where $\tau^l_{i,m}$ and $\tau^r_{i,m}$ appear. First, $\tau^l_{i,m}$ and $\tau^r_{i,m}$ both appears, w.l.o.g, in $\mathcal{I}_j$. Now, let $\Gamma_{j,m_1}$ and $\Gamma_{j,m_1+1}$ be such that $\tau^r_{j,m_1} \le t < \tau^l_{j,m_1+1}$. Intuitively, this means that $t$ lies in between (and is adjacent to) the intervals $\Gamma_{j,m_1}$ and $\Gamma_{j,m_1+1}$. Note that both $\tau^r_{j,m_1}$ and $\tau^l_{j,m_1+1}$ is not included in $\mathcal{I}_i$ since $\mathcal{I}_i$ consists of maximal disjoint intervals and $[\tau^r_{j,m_1}, \tau^l_{j,m_1+1}] \subset \Gamma_{i,m}$. Now, based on Formula 8.5, $\tau^r_{j,m_1}$ and $\tau^l_{j,m_1+1}$ are included in some intervals in $\mathcal{I}_{j'}$. Note that if they are

included in the same interval, then that interval also contains $t$, raising the contradiction to our assumption that $t \notin \Gamma_{j',m''}$ for any $\Gamma_{j',m''} \in \mathcal{I}_{j'}$. Then $\tau^r_{j,m_1}$ and $\tau^l_{j,m_1+1}$ are not included in the same interval in $\mathcal{I}_{j'}$. Then, there exists $\Gamma_{j',m_2} \in \mathcal{I}_{j'}$ and $\Gamma_{j',m_2+1} \in \mathcal{I}_{j'}$ such that,

$$\tau^r_{j,m_1} < \tau^r_{j',m_2} \leq t < \tau^l_{j',m_2+1} < \tau^l_{j,m_1+1}$$

Now note that, $\tau^r_{j',m_2}$ and $\tau^l_{j',m_2+1}$ both are not included in any of the intervals in $\mathcal{I}_j$. Now, based on Formula 8.6, both appear in $\mathcal{I}_i$. But that contradicts our assumption that $t \in \Gamma_{i,m}$.

For the other direction, we show that any timepoint, w.l.o.g, $t \in \Gamma_{j,m}$ belongs to some $\Gamma_{i,m'} \in \mathcal{I}_j$. For this, there can be three cases based on whether the bounds of $\Gamma_{j,m}$ appear as bounds in some interval $\Gamma_{i,m'} \in \mathcal{I}_i$ or not.

First, assume that both $\tau^l_{j,m}$ and $\tau^r_{j,m}$ appear as bounds $\tau^l_{i,m_1}$ and $\tau^r_{i,m_2}$ in $\mathcal{I}_k$ as stated by Formula 8.4. We now claim that $m_1 = m_2$ meaning that $\tau^l_{i,m_1}$ and $\tau^r_{i,m_2}$ are bounds of the same intervals. Towards contradiction, let $m_1 + 1 \leq m_2$. Then, $\tau^r_{i,m_1}$ belongs to the interval $\Gamma_{j,m}$, and based on Formula 8.5, and cannot be one of the bounds of $\Gamma_{i,m_1}$. Then, we have $\tau^l_{j,m} = \tau^l_{i,m_1} \leq t < \tau^r_{i,m_1} = \tau^r_{j,m}$

Second, assume that $\tau^l_{j,m}$ does not appear, while $\tau^r_{j,m}$ appears as bounds in $\mathcal{I}_k$. Now, based on Formula 8.5, $\tau^l_{j,m}$ appears in one of the intervals $\Gamma_{j',m'}$ in $\mathcal{I}_{j'}$. Also, in that case, $\tau^l_{j',m'}$ appears as a left bound in $\mathcal{I}_k$, say $\mathcal{I}_{i,m_1}$. We now claim that $\tau^r_{i,m_1} > \tau^r_{j,m}$. Towards contradiction, we assume two cases. In the first case,

$$\tau^l_{j',m'} = \tau^l_{i,m_1} < \tau^r_{i,m_1} < \tau^l_{j,m} < \tau^r_{j',m}$$

contradicting Formula 8.5. In the second case,

$$\tau^l_{j',m'} = \tau^l_{i,m_1} < \tau^l_{j,m} < \tau^r_{i,m_1} < \tau^r_{j,m}$$

contradicting Formula 8.6. From the two cases, we conclude $\tau^r_{i,m_1} > \tau^r_{j,m}$ and hence, $\tau^l_{i,m_1} < \tau^l_{j,m} \leq t < \tau^r_{j,m} < \tau^r_{i,m_1}$. The argument in the third case is similar to those in the other two cases and can be seen easily. □

**Lemma 20** (Correctness of $comp_s$)**.** *Let $v$ be a satisfying assignment of $comp_s(i,j)$. Then, the set $\mathcal{I}_i = \{[v(t^l_{i,1,s}), v(t^r_{i,1,s})), \ldots, [v(t^l_{i,m,s}), v(t^r_{i,m,s}))\}$ consists of the maximal disjoint intervals of the complement of $\mathcal{I}_j = \{[v(t^l_{j,1,s}), v(t^r_{j,1,s})), \ldots, [v(t^l_{j,m,s}), v(t^r_{j,m,s}))\}$.*

*Proof.* We reuse the naming conventions for $\tau^\sigma_{\kappa,m}$ and $\Gamma_{\kappa,m}$ from the last proof. For the forward direction, we show that if $t \in \Gamma_{i,m}$ for some $\Gamma_{i,m} \in \mathcal{I}_i$ then $t \notin \Gamma_{j,m'}$ for any $\Gamma_{j,m'} \in \mathcal{I}_j$. First, let $m = 1$. Then, if $\tau^l_{j,1} = 0$, then Condition 8.1 gets triggered and $\tau^l_{i,1} = \tau^r_{j,1}$ and $\tau^r_{i,1} = \tau^r_{j,2}$. Hence, $\tau^r_{j,1} = \tau^l_{i,1} \leq t < \tau^r_{i,1} = \tau^l_{j,2}$. Also, if $\tau^l_{j,1} \neq 0$, then Condition 8.1 does not get triggered and $\tau^l_{i,1} = 0$ and $\tau^r_{i,1} = \tau^l_{j,1}$. Hence, $0 = \tau^l_{i,1} \leq t < \tau^r_{i,1} = \tau^l_{j,1}$. For $m \neq 1$, the reasoning works similarly.

For the other direction, we show that if $t \in \Gamma_{j,m}$ for some $\Gamma_{j,m} \in \mathcal{I}_j$ then $t \notin \Gamma_{i,m'}$ for any $\Gamma_{i,m'} \in \mathcal{I}_j$. The proof for this direction is almost identical to the proof for the forward direction. □

**Lemma 21** (Correctness of $\ominus_s^{[a_i,b_i]}$)**.** *Let $v$ be a satisfying assignment of $\ominus_s^{[a_i,b_i]}(k,j)$. Then, the set $\mathcal{I}_i = \{[v(t^l_{i,1,s}), v(t^r_{i,1,s})), \ldots, [v(t^l_{i,m,s}), v(t^r_{i,m,s}))\}$ consists of the maximal disjoint intervals by applying $I \ominus [a,b]$ to the intervals $I$ of $\mathcal{I}_j = \{[v(t^l_{j,1,s}), v(t^r_{j,1,s})), \ldots, [v(t^l_{j,m,s}), v(t^r_{j,m,s}))\}$, where $v(a_i) = a$ and $v(b_i) = b$.*

*Proof.* The proof of the above claim follows directly from the construction of the formula $\ominus_s^{[a_i,b_i]}(k,j)$. □

It is worth noting that although the number of intervals in $\mathcal{I}_{\varphi[i]}(\boldsymbol{x}^s_{\mathbb{T}})$ for each subformula $\varphi[i]$ is bounded by $\mathcal{M}$, it may not contain the same number of intervals. For instance, $\mathcal{I}_p(\boldsymbol{x}^s_{\mathbb{T}}) = \{[0,1),[6,7)\}$ has two intervals, while, assuming $T = 7$, $\mathcal{I}_{\neg p}(\boldsymbol{x}^s_{\mathbb{T}}) = \{[1,6)\}$ has only one interval.

To circumvent this, we introduce some variables $num_{i,s}$ for $i \in \{1,\ldots,n\}$ and $s \in \{1,\ldots,|\mathcal{S}|\}$ to track of the number of intervals in $\mathcal{I}_{\varphi[i]}(\boldsymbol{x}^s_{\mathbb{T}})$ for each subformula $\varphi[i]$ for each prefix $\boldsymbol{x}^s_{\mathbb{T}}$. We now impose $\bigwedge_{1\le i\le n, 1\le m\le \mathcal{M}}[m > num_{i,s}] \rightarrow [t^l_{i,m,s} = T \wedge t^r_{i,m,s} = T]$. This ensures that all the unused variables $t^\sigma_{i,m,s}$ for each Node $i$ and prefix $\boldsymbol{x}^s_{\mathbb{T}}$ in $\mathcal{S}$ are all set to $T$. We also use the $num_{i,s}$ variables in the constraints for easier computation of $\mathcal{I}_{\varphi[i]}(\boldsymbol{x}^s_{\mathbb{T}})$ for each operator. We include this in our implementation but omit it here for a simpler presentation.

$\Phi^{sem}$ is simply the conjunction of all the semantic constraints mentioned above.

**Consistency Constraints.** Finally, to ensure that the prospective formula $\varphi$ is G-sep for $\mathcal{S}$, we add:

$$\bigwedge_{\boldsymbol{x}^s_{\mathbb{T}}\in P}\left[(t^l_{n,1,s} = 0) \wedge (t^r_{n,1,s} = T)\right] \wedge \bigwedge_{\boldsymbol{x}^s_{\mathbb{T}}\in N}\left[(t^l_{n,1,s} \neq 0) \vee (t^r_{n,1,s} \neq T)\right].$$

This constraint says that $\mathcal{I}_{\varphi[n]}(\boldsymbol{x}^s_{\mathbb{T}}) = \{[0,T)\}$ for all the positive prefixes $\boldsymbol{x}^s_{\mathbb{T}}$, while $\mathcal{I}_{\varphi[n]}(\boldsymbol{x}^s_{\mathbb{T}}) \neq \{[0,T)\}$ for any negative prefixes $\boldsymbol{x}^s_{\mathbb{T}}$.

The correctness of our algorithm follows from the correctness of the inductive computation of $\mathcal{I}_\varphi(\boldsymbol{x}_{\mathbb{T}})$ in Lemma 17 and its encoding using the formulas described in Lemma 18. We state the correctness result formally as follows:

**Theorem 16** (Correctness)**.** *Given a sample $\mathcal{S}$ and a future-reach bound $K$, Algorithm 14 terminates and outputs a minimal MTL formula $\varphi$ such that $\varphi$ is globally separating for $\mathcal{S}$ and $fr(\varphi) \le K$, if such a formula exists.*

*Proof.* The termination of Algorithm 14 is guaranteed by the decision procedure of checking whether $\mathcal{S}$ is *K-infix-separable* (Section 8.3). The minimality of the learned formula is due to the iterative search of formulas of increasing size and the correct encoding of $\Phi^n_{\mathcal{S},K}$. The correctness of $\Phi^n_{\mathcal{S},K}$ follows from the correctness of the encoding of set operations described in Lemma 18 and the correctness of computation of the sets $\mathcal{I}_\varphi(\boldsymbol{x}_{\mathbb{T}})$ using Lemma 17. □

Our learning algorithm solves the optimization problem LEARNMTL by constructing formulas in LRA. We now analyze the computational hardness of LEARNMTL and, thus, consider its corresponding decision problem LEARNMTL$_d$: given a sample $\mathcal{S}$, a future-reach

bound $K$ and size bound $B$ (in unary), does there exist an MTL formula $\varphi$ such that $\varphi$ is $\mathsf{G}$-sep for $\mathcal{S}, fr(\varphi) \leq K$, and $|\varphi| \leq B$. Following our algorithm, we can encode the LEARNMTL$_d$ problem in an LRA formula $\Phi = \bigvee_{n \leq B} \Phi^n_{\mathcal{S},K}$, where $\Phi^n_{\mathcal{S},K}$ is as described in Algorithm 14. One can check that the size of $\Phi$ is $\mathcal{O}(|\mathcal{S}||K|B^3\mathcal{M}^3)$. Now, the fact that the satisfiability of an LRA formula is NP-complete [58] proves the following:

**Theorem 17.** LEARNMTL$_d$ *is in* NP.

While the exact complexity lower bound for LEARNMTL$_d$ is unknown, we conjecture that LEARNMTL$_d$ is NP-hard. Our hypothesis stems from the fact that the problem is already NP-hard for simple fragments of *LTL* [83]. Note that the hardness result does not directly extend to MTL: the complexity might be either lower or higher since the logic is a priori more expressive. We leave the hardness result for full MTL as an open problem.

## 8.5 Experimental Evaluation

In this section, we answer the following research questions to assess the performance of our algorithm for learning MTL formulas.

**RQ1**: Can our algorithm learn concise formulas with small future-reach?

**RQ2**: How does lowering the future-reach bound affect the size of the formulas?

**RQ3**: How does our algorithm scale for different sample sizes?

To answer the research questions above, we have implemented a prototype TEAL[9] [189] of our algorithm in Python 3 using Z3 [164] as the SMT solver. To our knowledge, TEAL is the only tool for learning minimal MTL formulas for runtime monitoring (see related works). In TEAL, we implement a heuristic on top of Algorithm 14. We set the maximum number of intervals $\mathcal{M}$ in sets $\mathcal{I}_{\varphi[i]}(\boldsymbol{x}_{\mathbb{T}})$ to be $\mu + 2$ where $\mu = \max(\{|\mathcal{I}_p(\boldsymbol{x}_{\mathbb{T}})| \mid p \in \mathcal{P}\})$. This heuristic improves the runtime of TEAL significantly since most $\mathsf{G}$-sep formulas $\varphi$ never require the worst-case upper bound[10] of $\mathcal{M} = \mu|\varphi|$. To ensure that TEAL returns a correct MTL formula with this heuristic, we implement a verifier based on the inductive computation of $\mathcal{I}_\varphi(\boldsymbol{x}_{\mathbb{T}})$ from Table 8.1. In our experiments, the verifier ensured that all of the learned MTL formulas were correct. One can fine-tune the heuristic based on the sample and the expected MTL formulas.

As typically done in the literature of learning formulas [8, 169, 186], we evaluate TEAL on benchmarks generated synthetically from MTL formulas. To obtain useful MTL formulas, we identify a number of MTL patterns, listed in Table 8.2, commonly used for monitoring cyber-physical systems. For instance, the time-sensitive requirement of an electronically controlled steering (ECS) system "operational checks like RAM verification must be done every 20 secs" can be monitored globally using the *bounded recurrence* formula

[9] TEAL is available at https://github.com/ritamraha/Teal

[10] The operators $\mathsf{F}_I$, $\mathsf{G}_I$, $\wedge$, and $\neg$ increase the number of required intervals by at most one. Only the $\vee$ operator can double it in the worst-case.

$\mathsf{F}_{[0,20]}$ `operational_check` [140]; the requirement of an autonomous vehicle (from the introductory example) "brake should be triggered until within 2 secs the vehicle has no obstacle in an unsafe distance" can be monitored globally using the *bounded until* formula `brake` $\mathsf{U}_{[0,2]}$ `no_obstacle`.

TABLE 8.2: Typical MTL patterns used for monitoring cyber-physical systems

| | |
|---|---|
| Bounded Recurrence: | $\mathit{Globally}(\mathsf{F}_{[t_1,t_2]}\, p)$ |
| Bounded Response: | $\mathit{Globally}(p \rightarrow \mathsf{F}_{[t_1,t_2]}\, q)$ |
| Bounded Invariance: | $\mathit{Globally}(p \rightarrow \mathsf{G}_{[t_1,t_2]}\, q)$ |
| Bounded Until: | $\mathit{Globally}(p\, \mathsf{U}_{[t_1,t_2]}\, q)$ |

In our experiments, we construct MTL formulas from the patterns in Table 8.2 by replacing time interval $[t_1, t_2]$ with different values. Now, to generate a sample from an MTL formula $\varphi$, we generated a set of random prefixes and then classified them into positive or negative depending on whether $\varphi$ holds at all time-points of the prefix or not. We conducted all the experiments on a single core of an AMD EPYC 7702 64-core CPU (at 2GHz) using up to 10GB of RAM. The timeout was set to be 5400 secs for all the experiments.

### 8.5.1 RQ1: Recovering Concise Formulas

To address RQ1, we ran TEAL on a benchmark suite generated from nine MTL formulas obtained from the three MTL patterns in Table 8.2 by replacing $t_1$ with 0 and $t_2$ with 1,2, and 3. The suite consists of 36 samples for each pattern (12 samples for each formula), with the number of prefixes ranging from 10 to 40 and the length of prefixes (that is, the number of sampled timepoints) ranging from 4 to 6. For each sample $\mathcal{S}$, we set the future-reach bound $K$ to be $fr(\varphi)$, where $\varphi$ is the formula from which $\mathcal{S}$ was generated.

TABLE 8.3: Summary of the learned formulas.

| Formula pattern | Successful runs | | Timed out | Avg Size | Avg Time |
|---|---|---|---|---|---|
| | Matched | Not Matched | | | (in sec) |
| Bounded Recurrence | 36 | 0 | 0 | 2 | 17.5 |
| Bounded Response | 25 | 5 | 6 | 3.7 | 1860.3 |
| Bounded Invariance | 15 | 7 | 14 | 3.6 | 1397.2 |
| Bounded Until | 32 | 4 | 0 | 2.9 | 362.4 |

We depict the summary of the results for RQ1 in Table 8.3. For each run, we noted the formula learned, its size, and the total time taken. We also noted whether the learned formula matched the pattern of the original formula using which the sample was generated. We observed that the learned formulas matched the pattern of the original formula in 87.1% of the cases in which TEAL did not time out. This shows that the randomly generated samples captured the behavior of the original formula rather well, enabling a fair evaluation of TEAL.

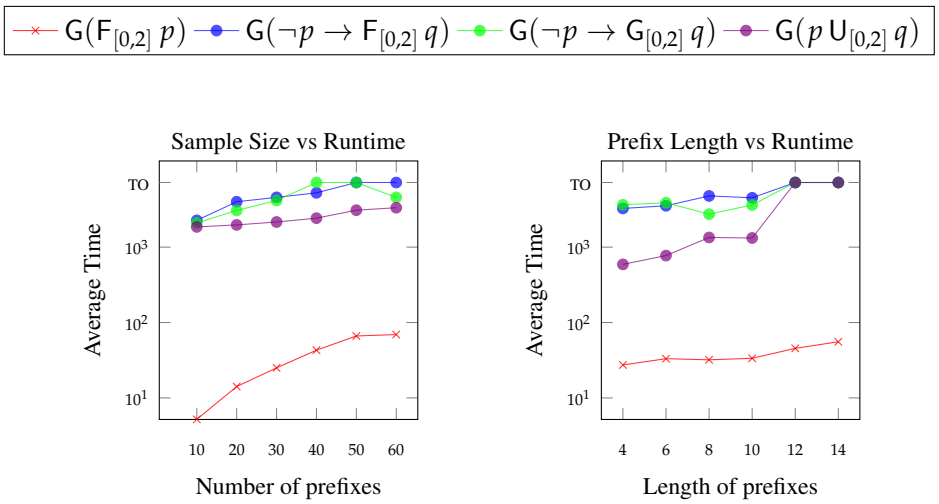



FIGURE 8.2: Runtime change with respect to the number of prefixes and prefix lengths

Further, we observed that the size of the learned formula is always less than or equal to that of the original formula. This demonstrates that TEAL always finds a concise formula for a given future-reach bound, answering RQ1 in positive.

### 8.5.2 RQ2: Future-Reach and Size tradeoff

To address RQ2, we investigate how the size of the learned formula changed over varying future-reach bounds. For this, we ran TEAL on the same benchmark suite from RQ1 but, this time, by varying the future-reach bound $K$ from 1 to 4. We investigate the average size of the minimal formula we get over the generated 144 samples for each future-reach bound.

We observed that for future-reach bounds $K$ of 1, 2, 3, and 4, the average size of the learned minimal formulas were 3.904, 3.734, 3.370, and 3.361, respectively. Thus, with an increase in $K$, the average size of the minimal formula decreased. This is because an increase in $K$ allows a bigger search space of formulas. However, the decrease in size with the increase in future-reach bound is not significant. This asserts the need for a future-reach bound for learning formulas and confirms the efficacy of our algorithm.

### 8.5.3 RQ3: Scalability

To address RQ3, we ran TEAL on a benchmark suite generated from MTL formulas which originate from the MTL patterns in Table 8.2, setting $t_1 = 0$ and $t_1 = 2$. The suite consists of 36 samples for each formula, with the number of prefixes varying from 10 to 60 and the length of prefixes varying from 4 to 14. We set the future-reach bound $K$ to be two.

Figure 8.2 illustrates the runtime variation of TEAL in two cases: increasing the number of prefixes fixing the length of them and increasing the length of prefixes fixing the number of them. We observe that to learn a larger formula, the time required grows significantly. This trend can be noticed in both cases.

## 8.6 Conclusion

We have presented a novel SMT-based algorithm for automatically learning MTL specifications from finite system executions. To be useful for efficient monitoring, we ensure that the learned formulas are both concise and have low future-reach. We have shown that our algorithm can learn concise formulas from benchmarks generated from commonly used MTL patterns.

While our algorithm is tailored to learn globally separating formulas particularly useful for monitoring, we can adapt our algorithm easily to learn only *separating* formulas as in the standard temporal logic inference setting [169, 160]. Our algorithm includes all the standard temporal operators that are typically used in MTL. However, we believe it is possible to improve the performance of the algorithm by omitting a temporal operator such as $\mathsf{U}_I$ for which the encoding can be substantially large.

From a practical point of view, an interesting future direction will be to lift our techniques to automatically learn STL formulas for verification. A straightforward approach towards this using the above-mentioned constraint-based methods has been explained by Raha [185]. However, for industrial use and scalability, clever heuristics and optimizations are needed to be explored in future work.

# Chapter 9

# Conclusion and Future Works

In this thesis, our goal was to study and understand the behavior of data-driven intelligent systems. Such systems are often, by design, highly complex and their behavior is difficult for humans to interpret. In particular, intelligent systems involving cyber-physical components, such as autonomous vehicles, surveillance drones, warehouse robots, etc., display complex temporal interactions involving various parameters of the system and the environment.

To explain the temporal behavior of systems, in this thesis, we investigated the problems of automated learning of temporal properties for systems. Towards this goal, we relied on two fundamental models that arise in formal reasoning: *finite state automata* and *temporal logics*. Both models can be said to have existed since the inception of formal methods, and their properties have been studied in depth. Moreover, both models have easy-to-grasp syntax and semantics, making them suitable from an interpretability perspective.

The main learning problem addressed in this thesis was that of *passive learning* finite automata and temporal logics. Historically, finite state automata were the models of primary interest for the passive learning problem. However, in this thesis, we emphasize the significance of extending the study to include temporal logics. We considered several variants of passive learning, keeping practical situations in mind.

In Chapter 3, we presented methods to expedite the learning of $\text{LTL}_\text{f}$ through efficient combinatorial search. We leveraged subclasses of $\text{LTL}_\text{f}$ that enable structured search through potential $\text{LTL}_\text{f}$ formulas. To improve the expressive power of the subclasses, we explored their boolean combinations via a novel greedy approximation algorithm. By incorporating several heuristics, our algorithms were able to produce rather concise $\text{LTL}_\text{f}$ formulas approximately $10\times$ faster than the existing approaches.

In Chapter 4, we considered a typical practical setting: learning from noisy data. Along with LTL, we considered learning STL formulas (with discrete semantics) due to their relevance in cyber-physical systems. To tolerate the noise in data, our algorithms focussed on finding formulas that allow a bounded amount of misclassification. Our algorithms exploited the maximum satisfiability problem and combined it with decision tree learning. With careful tuning of the parameters, our algorithms could effectively learn formulas from noisy samples better than existing approaches.

In Chapter 5, we considered enhancing the LTL learning process by instilling designer knowledge as sketches. Sketches provide a simple means to express designer intuition by leaving complex parts of a formula unspecified. We studied when a sketch can aid the learning

process and designed a decision procedure to check it automatically. Further, we extended existing learning algorithms to support sketches. Our sketching algorithms outperformed existing LTL miners in handling sketches.

In Chapter 6, we considered an often neglected setting: learning from positive examples. We showed that this problem can often be ill-posed, resulting in trivial solutions if not formulated properly. To this end, we relied on the size and language of models as regularizers to learn useful DFAs and $\text{LTL}_\text{f}$ formulas. We devised symbolic and counter-example guided algorithms to solve our learning problems. Our algorithms were able to extract meaningful descriptions in DFAs and $\text{LTL}_\text{f}$ from positive examples in several scenarios.

In Chapter 7, we explored the learning of properties in PSL, a logic that combines LTL with regular expressions. We modified a SAT-based LTL learning approach to incorporate regular expressions, thus extending the expressive power of our learning algorithms to $\omega$-regular properties.

In Chapter 8, we explored the learning of properties in the continuous-time logics MTL, specifically focusing on continuous dense-time semantics. Our learning algorithm symbolically encoded a standard monitoring procedure for continuous-time logics, enabling it to learn several commonly used MTL formulas successfully.

Through this thesis, we lay the foundations of automatically learning temporal properties of black-box systems. We hope that this thesis serves as an enhanced starting point for future research in passive learning of formal models.

## 9.1 Future Works

In each chapter of this thesis, we already alluded to various future directions. Most of them describe low-level technical extensions, which may be achieved by simple modifications of the presented techniques. Here, I will list some high-level challenges that may demand detailed investigation and have the potential to catalyze new areas of research.

### 9.1.1 Syntax-Guided Learning

System designers often have high-level intuition about the behavior of their systems. In Chapter 5, we formalized such high-level intuition as a temporal logic sketch and utilized it to learn the underlying temporal property of a system. In the future, one can consider a generalization of this setting, in which the intuition is formalized as *temporal logic grammar*. A temporal logic grammar is a set of rules that define the syntax of the prospective temporal logic formulas. Based on the rules of a given grammar, one can learn a concise temporal formula that is consistent with the system executions. Since a sketch can always be expressed using grammar-based rules, a grammar provides the designer more flexibility in expressing their intuition.

One can investigate several problems in this setting. For instance, one can consider the following decision problem: given a sample $\mathcal{S}$ and a temporal logic grammar $\Gamma$, does there exist a formula that is derivable from $\Gamma$ and is consistent with $\mathcal{S}$. Moreover, one also needs to design efficient algorithms that can derive minimal formulas from a given grammar that are

consistent with a given sample. There are some indications that techniques like SyGUS [8] can be used to incorporate grammar in the learning process. Needless to say, the above problems can be studied for different temporal logic formalisms such as LTL, PSL, MTL, etc.

### 9.1.2 Heuristics for Scalability

The scalability of the learning algorithms remains one of the major challenges in its industry usage. An industrial application that requires identifying temporal patterns on a large scale is that of Business Process Mining (BPM) [1]. Based on recorded event logs, a typical problem in BPM is to identify the temporal patterns that conform with the event logs.

To express patterns appearing in event logs, formalisms such as Declare [178], RCons [51], etc., are typically employed in BPM applications. Often, these formalisms (or their existing mining approaches) cannot express (or learn) properties that can be expressed using temporal logics such as LTL or PSL. Thus, one needs to develop learning techniques for different temporal logics that can handle large amounts of event-log data.

In order to improve the scalability of the learning techniques, there are a variety of heuristics that one can try. One can identify temporal logic grammars that are (i) suited to the application at hand (e.g., BPM, RL, etc.) and, at the same time, (ii) allows for efficient enumeration techniques (similar to that of directed LTL from Chapter 3). Another heuristic, particularly relevant for learning from positive examples, is to consider regularizers that are computationally simpler compared to that of language minimality. One could rely on probabilistic measures that identify formulas that can distinguish the given executions from random executions.

One can also harness the power of neural networks in the learning techniques [153, 88]. Recent works have focussed on designing neural network architectures that are able to learn structured objects efficiently. Also, there are early signs of Large Language Models (LLMs) understanding formal languages [61, 177, 150, 90]. Thus, exploiting hints from LLMs could significantly expedite the learning process.

### 9.1.3 Learnability of Temporal Properties

While we present several algorithms for learning temporal properties in this thesis, many theoretical results related to the learnability of temporal properties remain open. For instance, the computational complexity of learning temporal properties is not yet well understood. One needs to explore hardness results for the following problem: given a sample $\mathcal{S}$ and a size parameter $B$, does there exist a formula of size $\leq B$ that is consistent with $\mathcal{S}$? There are preliminary results of this problem for LTL\LTL$_\text{f}$ [83, 39, 159, 40]. One can ask the same problem for other formalisms such as PSL, MTL, and so on. Moreover, one can study how the complexity changes in the setting of only positive examples, specification sketches and temporal logic grammars.

Further, little is known about the number of examples that are required to adequately learn the underlying temporal property. This is often formalized as measures such as sample

complexity, characteristic samples, etc. [117]. One can also study a related concept, VC-dimension [161], which will provide an indication of the capabilities of temporal logic in classification tasks.

### 9.1.4 Explanation of Multi-Agent Systems

Most of the techniques in the thesis focus on explaining the behavior of individual systems. In the future, one can consider explaining the complex interactions between several agents organized in a multiagent system. For expressing interesting properties of multiagent systems, temporal logics such as Alternating-time Temporal Logic (ATL) [2] and Strategy Logic [53] are particularly well-suited. We already show that constraint-based learning algorithms can be designed for branching-time logics such as Computation Tree Logic (CTL) and Alternating-time Temporal Logic (ATL) [38].

### 9.1.5 Active Learning for Temporal Properties

One of the significant drawbacks of passive learning is its inefficiency in adapting to additional or online data. This is where active learning, introduced by Angluin [7], excels.

In active learning, there is a *learner*, which is responsible for inferring the temporal property and a *teacher*, which has access to the property. Through several interactions, the teacher assists the learner in converging to the desired temporal property, providing hints such as counterexamples based on the current hypothesis. A well-designed learner stores these hints in an efficient data structure and utilizes them effectively in devising new hypotheses.

The only known active learner for temporal logics simply invokes passive learning for each interaction with the teacher [48]. This naive method is, by no means, practical since the passive learning typically ignores the knowledge about the current hypothesis while building the new one. One could, thus, devise passive learning algorithms particularly to cater to active learning by utilizing knowledge about the existing hypothesis.

# Bibliography


[1] Wil van der Aalst. *Process Mining: Data Science in Action*. 2nd. Springer Publishing Company, Incorporated, 2016. ISBN: 3662498502.

[2] Rajeev Alur, Thomas A. Henzinger, and Orna Kupferman. "Alternating-time temporal logic". In: *J. ACM* 49.5 (2002), pp. 672–713. DOI: 10.1145/585265.585270. URL: https://doi.org/10.1145/585265.585270.

[3] Rajeev Alur, Rishabh Singh, Dana Fisman, and Armando Solar-Lezama. "Search-based program synthesis". In: *Commun. ACM* 61.12 (2018), pp. 84–93. DOI: 10.1145/3208071. URL: https://doi.org/10.1145/3208071.

[4] "Amazon introduces Sparrow—a state-of-the-art robot that handles millions of diverse products". In: *Amazon* (2022). URL: https://www.aboutamazon.com/news/operations/amazon-introduces-sparrow-a-state-of-the-art-robot-that-handles-millions-of-diverse-products (visited on 11/10/2022).

[5] Glenn Ammons, Rastislav Bodík, and James R. Larus. "Mining specifications". In: *Conference Record of POPL 2002: The 29th SIGPLAN-SIGACT Symposium on Principles of Programming Languages, Portland, OR, USA, January 16-18, 2002*. Ed. by John Launchbury and John C. Mitchell. ACM, 2002, pp. 4–16. DOI: 10.1145/503272.503275. URL: https://doi.org/10.1145/503272.503275.

[6] Dana Angluin. "Finding patterns common to a set of strings". In: *Journal of Computer and System Sciences* 21.1 (1980), pp. 46–62. ISSN: 0022-0000. DOI: https://doi.org/10.1016/0022-0000(80)90041-0. URL: https://www.sciencedirect.com/science/article/pii/0022000080900410.

[7] Dana Angluin. "Learning Regular Sets from Queries and Counterexamples". In: *Inf. Comput.* 75.2 (1987), pp. 87–106.

[8] M. Fareed Arif, Daniel Larraz, Mitziu Echeverria, Andrew Reynolds, Omar Chowdhury, and Cesare Tinelli. "SYSLITE: Syntax-Guided Synthesis of PLTL Formulas from Finite Traces". In: *FMCAD*. IEEE, 2020, pp. 93–103.

[9] Roy Armoni, Limor Fix, Alon Flaisher, Rob Gerth, Boris Ginsburg, Tomer Kanza, Avner Landver, Sela Mador-Haim, Eli Singerman, Andreas Tiemeyer, Moshe Y. Vardi, and Yael Zbar. "The ForSpec Temporal Logic: A New Temporal Property-Specification Language". In: *8th International Conference of Tools and Algorithms for the Construction and Analysis of Systems, TACAS '02*. Vol. 2280. LNCS. Springer,

2002, pp. 296–211. DOI: 10.1007/3-540-46002-0_21. URL: https://doi.org/10.1007/3-540-46002-0_21.

[10] Eugene Asarin, Alexandre Donzé, Oded Maler, and Dejan Nickovic. "Parametric Identification of Temporal Properties". In: *Runtime Verification*. Ed. by Sarfraz Khurshid and Koushik Sen. Vol. 7186 LNCS. RV'11 September. San Francisco, CA: Springer Berlin Heidelberg, 2012, pp. 147–160. ISBN: 9783642298592. DOI: 10.1007/978-3-642-29860-8_12. URL: https://doi.org/10.1007/978-3-642-29860-8_12.

[11] Florent Avellaneda and Alexandre Petrenko. "Inferring DFA without Negative Examples". In: *Proceedings of the 14th International Conference on Grammatical Inference, ICGI 2018, Wrocław, Poland, September 5-7, 2018*. Ed. by Olgierd Unold, Witold Dyrka, and Wojciech Wieczorek. Vol. 93. Proceedings of Machine Learning Research. PMLR, 2018, pp. 17–29. URL: http://proceedings.mlr.press/v93/avellaneda19a.html.

[12] Stefano Bacherini, Alessandro Fantechi, Matteo Tempestini, and Niccolò Zingoni. "A Story About Formal Methods Adoption by a Railway Signaling Manufacturer". In: *FM*. Vol. 4085. Lecture Notes in Computer Science. Springer, 2006, pp. 179–189.

[13] Frédéric Badeau and Arnaud Amelot. "Using B as a High Level Programming Language in an Industrial Project: Roissy VAL". In: *ZB*. Vol. 3455. Lecture Notes in Computer Science. Springer, 2005, pp. 334–354.

[14] Christel Baier and Joost-Pieter Katoen. *Principles of model checking*. MIT press, 2008.

[15] Kevin Baldor and Jianwei Niu. "Monitoring Dense-Time, Continuous-Semantics, Metric Temporal Logic". In: *RV*. Vol. 7687. Lecture Notes in Computer Science. Springer, 2012, pp. 245–259.

[16] Tomas Balyo, Marijn Heule, Markus Iser, Matti Järvisalo, and Martin Suda, eds. *Proceedings of SAT Competition 2023: Solver, Benchmark and Proof Checker Descriptions*. English. Department of Computer Science Series of Publications B. Finland: Department of Computer Science, University of Helsinki, 2023.

[17] Chitta Baral. *Knowledge Representation, Reasoning and Declarative Problem Solving*. Cambridge University Press, 2003. DOI: 10.1017/CBO9780511543357.

[18] Haniel Barbosa, Clark W. Barrett, Martin Brain, Gereon Kremer, Hanna Lachnitt, Makai Mann, Abdalrhman Mohamed, Mudathir Mohamed, Aina Niemetz, Andres Nötzli, Alex Ozdemir, Mathias Preiner, Andrew Reynolds, Ying Sheng, Cesare Tinelli, and Yoni Zohar. "cvc5: A Versatile and Industrial-Strength SMT Solver". In: *TACAS (1)*. Vol. 13243. Lecture Notes in Computer Science. Springer, 2022, pp. 415–442.

[19] Benoît Barbot, Benedikt Bollig, Alain Finkel, Serge Haddad, Igor Khmelnitsky, Martin Leucker, Daniel Neider, Rajarshi Roy, and Lina Ye. "Extracting Context-Free Grammars from Recurrent Neural Networks using Tree-Automata Learning and A* Search". In: *Proceedings of the 15th International Conference on Grammatical Inference, 23-27 August 2021, Virtual Event*. Ed. by Jane Chandlee, Rémi Eyraud, Jeff Heinz, Adam Jardine, and Menno van Zaanen. Vol. 153. Proceedings of Machine Learning Research. PMLR, 2021, pp. 113–129. URL: https://proceedings.mlr.press/v153/barbot21a.html.

[20] Luciano Baresi, Mohammad Mehdi Pourhashem Kallehbasti, and Matteo Rossi. "Efficient Scalable Verification of LTL Specifications". In: *ICSE (1)*. IEEE Computer Society, 2015, pp. 711–721.

[21] Ezio Bartocci, Jyotirmoy V. Deshmukh, Alexandre Donzé, Georgios Fainekos, Oded Maler, Dejan Nickovic, and Sriram Sankaranarayanan. "Specification-Based Monitoring of Cyber-Physical Systems: A Survey on Theory, Tools and Applications". In: *Lectures on Runtime Verification*. Vol. 10457. Lecture Notes in Computer Science. Springer, 2018, pp. 135–175.

[22] Ezio Bartocci, Cristinel Mateis, Eleonora Nesterini, and Dejan Nickovic. "Survey on mining signal temporal logic specifications". In: *Information and Computation* 289 (2022), p. 104957. ISSN: 0890-5401. DOI: https://doi.org/10.1016/j.ic.2022.104957. URL: https://www.sciencedirect.com/science/article/pii/S0890540122001122.

[23] Alberto Bartoli, Giorgio Davanzo, Andrea De Lorenzo, Eric Medvet, and Enrico Sorio. "Automatic Synthesis of Regular Expressions from Examples". In: *IEEE Computer* 47.12 (2014), pp. 72–80. DOI: 10.1109/MC.2014.344. URL: https://doi.org/10.1109/MC.2014.344.

[24] David A. Basin, Felix Klaedtke, and Eugen Zalinescu. "Algorithms for Monitoring Real-Time Properties". In: *Runtime Verification - Second International Conference, RV 2011, San Francisco, CA, USA, September 27-30, 2011, Revised Selected Papers*. Ed. by Sarfraz Khurshid and Koushik Sen. Vol. 7186. Lecture Notes in Computer Science. Springer, 2011, pp. 260–275. DOI: 10.1007/978-3-642-29860-8_20.

[25] David A. Basin, Srdan Krstic, and Dmitriy Traytel. "Almost Event-Rate Independent Monitoring of Metric Dynamic Logic". In: *RV*. Vol. 10548. Lecture Notes in Computer Science. Springer, 2017, pp. 85–102.

[26] David A. Basin, Srdjan Krstic, and Dmitriy Traytel. "AERIAL: Almost Event-Rate Independent Algorithms for Monitoring Metric Regular Properties". In: *RV-CuBES*. Vol. 3. Kalpa Publications in Computing. EasyChair, 2017, pp. 29–36.

[27] O. Bernardi and Omer Giménez. "A Linear Algorithm for the Random Sampling from Regular Languages". In: *Algorithmica* 62 (2010), pp. 130–145.

[28] Olivier Bernardi and Omer Giménez. "A Linear Algorithm for the Random Sampling from Regular Languages". In: *Algorithmica* 62.1-2 (2012), pp. 130–145.

[29] Armin Biere, Alessandro Cimatti, Edmund M. Clarke, Ofer Strichman, and Yunshan Zhu. "Bounded Model Checking". In: vol. 58. Advances in Computers. Elsevier, 2003, pp. 117–148. DOI: https://doi.org/10.1016/S0065-2458(03)58003-2. URL: https://www.sciencedirect.com/science/article/pii/S0065245803580032.

[30] Alan W. Biermann and Jerome A. Feldman. "On the Synthesis of Finite-State Machines from Samples of Their Behavior". In: *IEEE Trans. Computers* 21.6 (1972), pp. 592–597.

[31] Dines Bjørner and Klaus Havelund. "40 Years of Formal Methods - Some Obstacles and Some Possibilities?" In: *FM*. Vol. 8442. Lecture Notes in Computer Science. Springer, 2014, pp. 42–61.

[32] León Bohn and Christof Löding. "Passive Learning of Deterministic Büchi Automata by Combinations of DFAs". In: *49th International Colloquium on Automata, Languages, and Programming, ICALP 2022, July 4-8, 2022, Paris, France*. Ed. by Mikolaj Bojanczyk, Emanuela Merelli, and David P. Woodruff. Vol. 229. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022, 114:1–114:20. DOI: 10.4230/LIPICS.ICALP.2022.114. URL: https://doi.org/10.4230/LIPIcs.ICALP.2022.114.

[33] Benedikt Böing, Rajarshi Roy, Emmanuel Müller, and Daniel Neider. "Quality Guarantees for Autoencoders via Unsupervised Adversarial Attacks". In: *Machine Learning and Knowledge Discovery in Databases - European Conference, ECML PKDD 2020, Ghent, Belgium, September 14-18, 2020, Proceedings, Part II*. Ed. by Frank Hutter, Kristian Kersting, Jefrey Lijffijt, and Isabel Valera. Vol. 12458. Lecture Notes in Computer Science. Springer, 2020, pp. 206–222. DOI: 10.1007/978-3-030-67661-2_13. URL: https://doi.org/10.1007/978-3-030-67661-2_13.

[34] Udi Boker, Karoliina Lehtinen, and Salomon Sickert. "On the Translation of Automata to Linear Temporal Logic". In: *Foundations of Software Science and Computation Structures - 25th International Conference, FOSSACS 2022, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2022, Munich, Germany, April 2-7, 2022, Proceedings*. Ed. by Patricia Bouyer and Lutz Schröder. Vol. 13242. Lecture Notes in Computer Science. Springer, 2022, pp. 140–160. DOI: 10.1007/978-3-030-99253-8_8. URL: https://doi.org/10.1007/978-3-030-99253-8_8.

[35] Benedikt Bollig, Joost-Pieter Katoen, Carsten Kern, Martin Leucker, Daniel Neider, and David R. Piegdon. "libalf: The Automata Learning Framework". In: *Computer Aided Verification, 22nd International Conference, CAV 2010, Edinburgh, UK, July 15-19, 2010. Proceedings*. Ed. by Tayssir Touili, Byron Cook, and Paul B. Jackson. Vol. 6174. Lecture Notes in Computer Science. Springer, 2010, pp. 360–364. DOI: 10.1007/978-3-642-14295-6_32. URL: https://doi.org/10.1007/978-3-642-14295-6_32.

[36] Giuseppe Bombara and Calin Belta. "Offline and Online Learning of Signal Temporal Logic Formulae Using Decision Trees". In: *ACM Trans. Cyber-Phys. Syst.* 5.3 (2021). ISSN: 2378-962X. DOI: 10.1145/3433994. URL: https://doi.org/10.1145/3433994.

[37] Giuseppe Bombara, Cristian Ioan Vasile, Francisco Penedo, Hirotoshi Yasuoka, and Calin Belta. "A Decision Tree Approach to Data Classification Using Signal Temporal Logic". In: *Proceedings of the 19th International Conference on Hybrid Systems: Computation and Control*. HSCC '16. New York, NY, USA: Association for Computing Machinery, 2016, 1–10. ISBN: 978-1-4503-3955-1. DOI: 10.1145/2883817.2883843. URL: https://doi.org/10.1145/2883817.2883843.

[38] Benjamin Bordais, Daniel Neider, and Rajarshi Roy. "Learning Branching-Time Properties in CTL and ATL via Constraint Solving". In: *Formal Methods - 26th International Symposium, FM 2024, Milan, Italy, September 9-13, 2024, Proceedings, Part I*. Ed. by André Platzer, Kristin Yvonne Rozier, Matteo Pradella, and Matteo Rossi. Vol. 14933. Lecture Notes in Computer Science. Springer, 2024, pp. 304–323. DOI: 10.1007/978-3-031-71162-6_16. URL: https://doi.org/10.1007/978-3-031-71162-6_16.

[39] Benjamin Bordais, Daniel Neider, and Rajarshi Roy. "Learning Temporal Properties is NP-hard". In: *CoRR* abs/2312.11403 (2023). DOI: 10.48550/ARXIV.2312.11403. arXiv: 2312.11403. URL: https://doi.org/10.48550/arXiv.2312.11403.

[40] Benjamin Bordais, Daniel Neider, and Rajarshi Roy. "The Complexity of Learning Temporal Properties". In: *CoRR* abs/2408.04486 (2024). DOI: 10.48550/ARXIV.2408.04486. arXiv: 2408.04486. URL: https://doi.org/10.48550/arXiv.2408.04486.

[41] Jonathan P. Bowen. "Gerard O'Regan: Concise Guide to Formal Methods: Theory, Fundamentals and Industry Applications". In: *Formal Aspects Comput.* 32.1 (2020), pp. 147–148.

[42] Andrea Brunello, Guido Sciavicco, and Ionel Eduard Stan. "Interval Temporal Logic Decision Tree Learning". In: *JELIA*. Vol. 11468. Lecture Notes in Computer Science. Springer, 2019, pp. 778–793.

[43] Glenn Bruns and Patrice Godefroid. "Temporal Logic Query Checking". In: *16th Annual IEEE Symposium on Logic in Computer Science, Boston, Massachusetts, USA, June 16-19, 2001, Proceedings*. IEEE Computer Society, 2001, pp. 409–417. DOI: 10.1109/LICS.2001.932516. URL: https://doi.org/10.1109/LICS.2001.932516.

[44] Carlos E Budde, Pedro R D Argenio, Arnd Hartmanns B, and Sean Sedwards. "Qualitative and Quantitative Trace Analysis with Extended Signal Temporal Logic". In: 1 (2018), pp. 340–358. DOI: 10.1007/978-3-319-89963-3. URL: http://dx.doi.org/10.1007/978-3-319-89963-3_20.

[45] Hugues Calbrix, Maurice Nivat, and Andreas Podelski. "Ultimately Periodic Words of Rational *w*-Languages". In: *MFPS*. Vol. 802. Lecture Notes in Computer Science. Springer, 1993, pp. 554–566.

[46] Alberto Camacho, Rodrigo Toro Icarte, Toryn Q. Klassen, Richard Anthony Valenzano, and Sheila A. McIlraith. "LTL and Beyond: Formal Languages for Reward Function Specification in Reinforcement Learning". In: *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI 2019, Macao, China, August 10-16, 2019*. Ed. by Sarit Kraus. ijcai.org, 2019, pp. 6065–6073. DOI: 10.24963/ijcai.2019/840. URL: https://doi.org/10.24963/ijcai.2019/840.

[47] Alberto Camacho and Sheila A. McIlraith. "Learning Interpretable Models Expressed in Linear Temporal Logic". In: *International Conference on Automated Planning and Scheduling, ICAPS* (2019). URL: https://ojs.aaai.org/index.php/ICAPS/article/view/3529.

[48] Alberto Camacho and Sheila A. McIlraith. "Learning Interpretable Models Expressed in Linear Temporal Logic". In: *ICAPS*. AAAI Press, 2019, pp. 621–630.

[49] Alberto Camacho and Sheila A. McIlraith. "Learning Interpretable Models Expressed in Linear Temporal Logic". In: *Proceedings of the International Conference on Automated Planning and Scheduling* 29.1 (2021), pp. 621–630. DOI: 10.1609/icaps.v29i1.3529. URL: https://ojs.aaai.org/index.php/ICAPS/article/view/3529.

[50] Rafael C. Carrasco and José Oncina. "Learning deterministic regular grammars from stochastic samples in polynomial time". In: *RAIRO Theor. Informatics Appl.* 33.1 (1999), pp. 1–20.

[51] Alessio Cecconi, Claudio Di Ciccio, Giuseppe De Giacomo, and Jan Mendling. "Interestingness of Traces in Declarative Process Mining: The Janus LTLp _f Approach". In: *BPM*. Vol. 11080. Lecture Notes in Computer Science. Springer, 2018, pp. 121–138.

[52] William Chan. "Temporal-Logic Queries". In: *CAV*. Vol. 1855. Lecture Notes in Computer Science. Springer, 2000, pp. 450–463.

[53] Krishnendu Chatterjee, Thomas A. Henzinger, and Nir Piterman. "Strategy Logic". In: *CONCUR 2007 - Concurrency Theory, 18th International Conference, CONCUR 2007, Lisbon, Portugal, September 3-8, 2007, Proceedings*. Ed. by Luís Caires and Vasco Thudichum Vasconcelos. Vol. 4703. Lecture Notes in Computer Science. Springer, 2007, pp. 59–73. DOI: 10.1007/978-3-540-74407-8_5. URL: https://doi.org/10.1007/978-3-540-74407-8_5.

[54] Agnishom Chattopadhyay and Konstantinos Mamouras. "A Verified Online Monitor for Metric Temporal Logic with Quantitative Semantics". In: *RV*. Vol. 12399. Lecture Notes in Computer Science. Springer, 2020, pp. 383–403.

[55] Himaja Cherukuri, Alessio Ferrari, and Paola Spoletini. "Towards Explainable Formal Methods: From LTL to Natural Language with Neural Machine Translation". In: *Requirements Engineering: Foundation for Software Quality - 28th International Working Conference, REFSQ 2022, Birmingham, UK, March 21-24, 2022, Proceedings*. Ed. by Vincenzo Gervasi and Andreas Vogelsang. Vol. 13216. Lecture Notes in Computer Science. Springer, 2022, pp. 79–86. DOI: 10.1007/978-3-030-98464-9_7. URL: https://doi.org/10.1007/978-3-030-98464-9_7.

[56] Glen Chou, Necmiye Ozay, and Dmitry Berenson. "Learning temporal logic formulas from suboptimal demonstrations: theory and experiments". In: *Auton. Robots* 46.1 (2022), pp. 149–174.

[57] Alessandro Cimatti, Alberto Griggio, Bastiaan Joost Schaafsma, and Roberto Sebastiani. "The MathSAT5 SMT Solver". In: *TACAS*. Vol. 7795. Lecture Notes in Computer Science. Springer, 2013, pp. 93–107.

[58] Barrett Clark and Tinelli Cesare. "Satisfiability Modulo Theories". In: *Handbook of Model Checking*. Springer International Publishing, 2018, pp. 305–343. DOI: 10.1007/978-3-319-10575-8_11. URL: https://doi.org/10.1007/978-3-319-10575-8_11.

[59] Edmund M. Clarke and E. Allen Emerson. "Design and Synthesis of Synchronization Skeletons Using Branching-Time Temporal Logic". In: *Logics of Programs, Workshop, Yorktown Heights, New York, USA, May 1981*. Vol. 131. Lecture Notes in Computer Science. Springer, 1981, pp. 52–71. DOI: 10.1007/BFb0025774.

[60] Darren D. Cofer and Steven P. Miller. "DO-333 Certification Case Studies". In: *NASA Formal Methods*. Vol. 8430. Lecture Notes in Computer Science. Springer, 2014, pp. 1–15.

[61] Matthias Cosler, Christopher Hahn, Daniel Mendoza, Frederik Schmitt, and Caroline Trippel. "nl2spec: Interactively Translating Unstructured Natural Language to Temporal Logics with Large Language Models". In: *Computer Aided Verification - 35th International Conference, CAV 2023, Paris, France, July 17-22, 2023, Proceedings, Part II*. Ed. by Constantin Enea and Akash Lal. Vol. 13965. Lecture Notes in Computer Science. Springer, 2023, pp. 383–396. DOI: 10.1007/978-3-031-37703-7_18. URL: https://doi.org/10.1007/978-3-031-37703-7_18.

[62] François Coste and Daniel Fredouille. "Unambiguous Automata Inference by Means of State-Merging Methods". In: *Machine Learning: ECML 2003, 14th European Conference on Machine Learning, Cavtat-Dubrovnik, Croatia, September 22-26, 2003, Proceedings*. Ed. by Nada Lavrac, Dragan Gamberger, Ljupco Todorovski, and Hendrik Blockeel. Vol. 2837. Lecture Notes in Computer Science. Springer, 2003, pp. 60–71. DOI: 10.1007/978-3-540-39857-8_8. URL: https://doi.org/10.1007/978-3-540-39857-8_8.

[63] Pierre-Jacques Courtois, F. Seidel, F. Gallardo, and M. Bowell. "Licensing of safety critical software for nuclear reactors. Common position of international nuclear regulators and authorised technical support organisations." In: (Dec. 2015). DOI: 10.13140/RG.2.1.2789.8968.

[64] Jin Cui, Lin Shen Liew, Giedre Sabaliauskaite, and Fengjun Zhou. "A review on safety failures, security attacks, and available countermeasures for autonomous vehicles". In: *Ad Hoc Networks* 90 (2019). Recent advances on security and privacy in Intelligent Transportation Systems, p. 101823. ISSN: 1570-8705. DOI: https://doi.org/10.1016/j.adhoc.2018.12.006. URL: https://www.sciencedirect.com/science/article/pii/S1570870518309260.

[65] Jenny Cusack. "How driverless cars will change our world". In: *BBC* (2021). URL: https://www.bbc.com/future/article/20211126-how-driverless-cars-will-change-our-world (visited on 11/30/2017).

[66] Thao Dang and Volker Stolz, eds. *Runtime Verification - 22nd International Conference, RV 2022, Tbilisi, Georgia, September 28-30, 2022, Proceedings*. Vol. 13498. Lecture Notes in Computer Science. Springer, 2022.

[67] Giuseppe De Giacomo and Moshe Y. Vardi. "Linear Temporal Logic and Linear Dynamic Logic on Finite Traces". In: *International Joint Conference on Artificial Intelligence, IJCAI*. 2013. DOI: 10.5555/2540128.2540252.

[68] François Denis, Aurélien Lemay, and Alain Terlutte. "Learning Regular Languages Using Non Deterministic Finite Automata". In: *Grammatical Inference: Algorithms and Applications, 5th International Colloquium, ICGI 2000, Lisbon, Portugal, September 11-13, 2000, Proceedings*. Ed. by Arlindo L. Oliveira. Vol. 1891. Lecture Notes in Computer Science. Springer, 2000, pp. 39–50. DOI: 10.1007/978-3-540-45257-7_4. URL: https://doi.org/10.1007/978-3-540-45257-7_4.

[69] Jyotirmoy V. Deshmukh, Alexandre Donzé, Shromona Ghosh, Xiaoqing Jin, Garvit Juniwal, and Sanjit A. Seshia. "Robust Online Monitoring of Signal Temporal Logic". In: *RV*. Vol. 9333. Lecture Notes in Computer Science. Springer, 2015, pp. 55–70.

[70] Irit Dinur and David Steurer. "Analytical approach to parallel repetition". In: *Symposium on Theory of Computing, STOC*. 2014, pp. 624–633. DOI: 10.1145/2591796.2591884.

[71] Adel Dokhanchi, Bardh Hoxha, and Georgios Fainekos. "On-Line Monitoring for Temporal Logic Robustness". In: *RV*. Vol. 8734. Lecture Notes in Computer Science. Springer, 2014, pp. 231–246.

[72] Alexandre Donzé, Thomas Ferrère, and Oded Maler. "Efficient Robust Monitoring for STL". In: *CAV*. Vol. 8044. Lecture Notes in Computer Science. Springer, 2013, pp. 264–279.

[73] Finale Doshi-Velez and Been Kim. *Towards A Rigorous Science of Interpretable Machine Learning*. 2017. arXiv: 1702.08608 [stat.ML].

[74] Alexandre Duret-Lutz, Alexandre Lewkowicz, Amaury Fauchille, Thibaud Michaud, Etienne Renault, and Laurent Xu. "Spot 2.0 - A Framework for LTL and \omega -Automata Manipulation". In: *ATVA*. Vol. 9938. Lecture Notes in Computer Science. 2016, pp. 122–129.

[75] Matthew B. Dwyer, George S. Avrunin, and James C. Corbett. "Patterns in Property Specifications for Finite-State Verification". In: *Proceedings of the 1999 International Conference on Software Engineering, ICSE' 99, Los Angeles, CA, USA, May 16-22, 1999*. Ed. by Barry W. Boehm, David Garlan, and Jeff Kramer. ACM, 1999, pp. 411–420. DOI: 10.1145/302405.302672. URL: https://doi.org/10.1145/302405.302672.

[76] Matthew B. Dwyer, George S. Avrunin, and James C. Corbett. "Property Specification Patterns for Finite-State Verification". In: *Proceedings of the Second Workshop on Formal Methods in Software Practice*. FMSP 1998. Association for Computing Machinery, 7–15.

[77] Matthew B. Dwyer, George S. Avrunin, and James C. Corbett. "Property specification patterns for finite-state verification". In: *FMSP*. ACM, 1998, pp. 7–15.

[78] Rüdiger Ehlers, Ivan Gavran, and Daniel Neider. "Learning Properties in LTL ∩ ACTL from Positive Examples Only". In: *FMCAD*. IEEE, 2020, pp. 104–112.

[79] Cindy Eisner and Dana Fisman. *A Practical Introduction to PSL*. Series on Integrated Circuits and Systems. Springer, 2006. ISBN: 978-0-387-35313-5. DOI: 10.1007/978-0-387-36123-9.

[80] Cindy Eisner, Dana Fisman, John Havlicek, Yoad Lustig, Anthony McIsaac, and David Van Campenhout. "Reasoning with Temporal Logic on Truncated Paths". In: *Computer Aided Verification*. Ed. by Warren A. Hunt and Fabio Somenzi. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 27–39.

[81] Georgios E. Fainekos, Hadas Kress-Gazit, and George J. Pappas. "Temporal Logic Motion Planning for Mobile Robots". In: *ICRA*. IEEE, 2005, pp. 2020–2025. DOI: 10.1109/ROBOT.2005.1570410.

[82] Mariusz A. Fecko, M. Ümit Uyar, Paul D. Amer, Adarshpal S. Sethi, Theodore Dzik, R. Menell, and Michael McMahon. "A success story of formal description techniques: Estelle specification and test generation for MIL-STD 188-220". In: *Comput. Commun.* 23.12 (2000), pp. 1196–1213.

[83] Nathanaël Fijalkow and Guillaume Lagarde. "The Complexity of Learning Linear Temporal Formulas from Examples". In: *International Conference on Grammatical Inference, ICGI*. Vol. 153. Proceedings of Machine Learning Research. PMLR, 2021, pp. 237–250. URL: https://proceedings.mlr.press/v153/fijalkow21a.html.

[84] Bernd Finkbeiner. "Synthesis of Reactive Systems". In: *Dependable Software Systems Engineering*. Vol. 45. NATO Science for Peace and Security Series - D: Information and Communication Security. IOS Press, 2016, pp. 72–98.

[85] Cameron Finucane, Gangyuan Jing, and Hadas Kress-Gazit. "LTLMoP: Experimenting with language, Temporal Logic and robot control". In: *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems, October 18-22, 2010, Taipei, Taiwan*. IEEE, 2010, pp. 1988–1993. DOI: 10.1109/IROS.2010.5650371. URL: https://doi.org/10.1109/IROS.2010.5650371.

[86] Limor Fix. "Fifteen Years of Formal Property Verification in Intel". In: *25 Years of Model Checking*. Vol. 5000. Lecture Notes in Computer Science. Springer, 2008, pp. 139–144.

[87] Philippe Flajolet and Robert Sedgewick. *Analytic Combinatorics*. Cambridge University Press, 2008. URL: http://algo.inria.fr/flajolet/Publications/AnaCombi/anacombi.html.

[88] Nicole Fronda and Houssam Abbas. "Differentiable Inference of Temporal Logic Formulas". In: *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.* 41.11 (2022), pp. 4193–4204. DOI: 10.1109/TCAD.2022.3197506. URL: https://doi.org/10.1109/TCAD.2022.3197506.

[89] Francesco Fuggitti. *LTLf2DFA*. Version 1.0.3. 2019. DOI: 10.5281/zenodo.3888410.

[90] Francesco Fuggitti and Tathagata Chakraborti. "NL2LTL - a Python Package for Converting Natural Language (NL) Instructions to Linear Temporal Logic (LTL) Formulas". In: *Proceedings of the Thirty-Seventh AAAI Conference on Artificial Intelligence and Thirty-Fifth Conference on Innovative Applications of Artificial Intelligence and Thirteenth Symposium on Educational Advances in Artificial Intelligence*. AAAI'23/IAAI'23/EAAI'23. AAAI Press, 2023. ISBN: 978-1-57735-880-0. DOI: 10.1609/aaai.v37i13.27068. URL: https://doi.org/10.1609/aaai.v37i13.27068.

[91] Dov M. Gabbay, Amir Pnueli, Saharon Shelah, and Jonathan Stavi. "On the Temporal Analysis of Fairness". In: *Conference Record of the Seventh Annual ACM Symposium on Principles of Programming Languages, Las Vegas, Nevada, USA, January 1980*. Ed. by Paul W. Abrahams, Richard J. Lipton, and Stephen R. Bourne. ACM Press, 1980, pp. 163–173. DOI: 10.1145/567446.567462. URL: https://doi.org/10.1145/567446.567462.

[92] Jean-Raphaël Gaglione, Daniel Neider, Rajarshi Roy, Ufuk Topcu, and Zhe Xu. "Learning Linear Temporal Properties from Noisy Data: A MaxSAT Approach". In: *CoRR* abs/2104.15083 (2021).

[93] Jean-Raphaël Gaglione, Daniel Neider, Rajarshi Roy, Ufuk Topcu, and Zhe Xu. "Learning Linear Temporal Properties from Noisy Data: A MaxSAT-Based Approach". In: *ATVA*. Vol. 12971. Lecture Notes in Computer Science. Springer, 2021, pp. 74–90.

[94] Jean-Raphaël Gaglione, Daniel Neider, Rajarshi Roy, Ufuk Topcu, and Zhe Xu. "MaxSAT-based temporal logic inference from noisy data". In: *Innov. Syst. Softw. Eng.* 18.3 (2022), pp. 427–442.

[95] Marco Gario, Alessandro Cimatti, Cristian Mattarei, Stefano Tonetta, and Kristin Yvonne Rozier. “Model Checking at Scale: Automated Air Traffic Control Design Space Exploration”. In: *CAV (2)*. Vol. 9780. Lecture Notes in Computer Science. Springer, 2016, pp. 3–22.

[96] Martin Gebser, Roland Kaminski, Benjamin Kaufmann, and Torsten Schaub. “Multi-shot ASP solving with clingo”. In: *CoRR* abs/1705.09811 (2017).

[97] Enrico Ghiorzi, Michele Colledanchise, Gianluca Piquet, Stefano Bernagozzi, Armando Tacchella, and Lorenzo Natale. “Learning Linear Temporal Properties for Autonomous Robotic Systems”. In: *IEEE Robotics Autom. Lett.* 8.5 (2023), pp. 2930–2937. DOI: 10.1109/LRA.2023.3263368. URL: https://doi.org/10.1109/LRA.2023.3263368.

[98] Shalini Ghosh, Daniel Elenius, Wenchao Li, Patrick Lincoln, Natarajan Shankar, and Wilfried Steiner. “ARSENAL: Automatic Requirements Specification Extraction from Natural Language”. In: *NASA Formal Methods, NFM*. 2016. DOI: 10.1007/978-3-319-40648-0_4.

[99] Giuseppe De Giacomo and Moshe Y. Vardi. “Automata-Theoretic Approach to Planning for Temporally Extended Goals”. In: *ECP*. Vol. 1809. Lecture Notes in Computer Science. Springer, 1999, pp. 226–238.

[100] Giuseppe De Giacomo and Moshe Y. Vardi. “Linear Temporal Logic and Linear Dynamic Logic on Finite Traces”. In: *IJCAI*. IJCAI/AAAI, 2013, pp. 854–860.

[101] Giuseppe De Giacomo and Moshe Y. Vardi. “Synthesis for LTL and LDL on Finite Traces”. In: *IJCAI*. AAAI Press, 2015, pp. 1558–1564.

[102] Dimitra Giannakopoulou, Thomas Pressburger, Anastasia Mavridou, Julian Rhein, Johann Schumann, and Nija Shi. “Formal Requirements Elicitation with FRET”. In: *International Conference on Requirements Engineering: Foundation for Software Quality, REFSQ*. 2020. URL: http://ceur-ws.org/Vol-2584/PT-paper4.pdf.

[103] E. Mark Gold. “Complexity of Automaton Identification from Given Data”. In: *Inf. Control.* 37.3 (1978), pp. 302–320.

[104] E. Mark Gold. “Language Identification in the Limit”. In: *Inf. Control.* 10.5 (1967), pp. 447–474.

[105] Valentin Goranko and Antje Rumberg. “Temporal Logic”. In: *The Stanford Encyclopedia of Philosophy*. Ed. by Edward N. Zalta and Uri Nodelman. Fall 2023. Metaphysics Research Lab, Stanford University, 2023.

[106] Felipe Gorostiaga and César Sánchez. “HLola: a Very Functional Tool for Extensible Stream Runtime Verification”. In: *Tools and Algorithms for the Construction and Analysis of Systems - 27th International Conference, TACAS 2021*. Ed. by Jan Friso Groote and Kim Guldstrand Larsen. Vol. 12652. Lecture Notes in Computer Science. Springer, 2021, pp. 349–356. DOI: 10.1007/978-3-030-72013-1_18.

[107] Ben Greenman, Sam Saarinen, Tim Nelson, and Shriram Krishnamurthi. "Little Tricky Logic: Misconceptions in the Understanding of LTL". In: *Art Sci. Eng. Program.* 7.2 (2023).

[108] Olga Grinchtein, Martin Leucker, and Nir Piterman. "Inferring Network Invariants Automatically". In: *IJCAR*. Vol. 4130. Lecture Notes in Computer Science. Springer, 2006, pp. 483–497.

[109] Hermann Gruber and Markus Holzer. "From Finite Automata to Regular Expressions and Back-A Summary on Descriptional Complexity". In: *14th International Conference on Automata and Formal Languages, AFL 2014*. Vol. 151. EPTCS. 2014, pp. 25–48. DOI: 10.4204/EPTCS.151.2. URL: https://doi.org/10.4204/EPTCS.151.2.

[110] Christopher Hahn, Frederik Schmitt, Jens U. Kreber, Markus Norman Rabe, and Bernd Finkbeiner. "Teaching Temporal Logics to Neural Networks". In: *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021. URL: https://openreview.net/forum?id=dOcQK-f4byz.

[111] M. E. Halaby. "On the Computational Complexity of MaxSAT". In: *Electron. Colloquium Comput. Complex.* 23 (2016), p. 34.

[112] David Harel and P. S. Thiagarajan. "Message Sequence Charts". In: *UML for Real - Design of Embedded Real-Time Systems*. Kluwer, 2003, pp. 77–105. DOI: 10.1007/0-306-48738-1_4.

[113] Mohammadhosein Hasanbeig, Natasha Yogananda Jeppu, Alessandro Abate, Tom Melham, and Daniel Kroening. "DeepSynth: Automata Synthesis for Automatic Task Segmentation in Deep Reinforcement Learning". In: *AAAI*. AAAI Press, 2021, pp. 7647–7656.

[114] Klaus Havelund and Doron Peled. "Runtime Verification: From Propositional to First-Order Temporal Logic". In: *RV*. Vol. 11237. Lecture Notes in Computer Science. Springer, 2018, pp. 90–112.

[115] J.G. Henriksen, J. Jensen, M. Jørgensen, N. Klarlund, B. Paige, T. Rauhe, and A. Sandholm. "Mona: Monadic Second-order logic in practice". In: *Tools and Algorithms for the Construction and Analysis of Systems, First International Workshop, TACAS '95, LNCS 1019*. 1995.

[116] Marijn Heule and Sicco Verwer. "Exact DFA Identification Using SAT Solvers". In: *10th International Colloquium of Grammatical Inference: Theoretical Results and Applications, ICGI '10*. Vol. 6339. LNCS. Springer, 2010, pp. 66–79. DOI: 10.1007/978-3-642-15488-1_7.

[117] Colin de la Higuera. *Grammatical Inference: Learning Automata and Grammars*. USA: Cambridge University Press, 2010. ISBN: 0521763169.

[118] Hsi-Ming Ho, Joël Ouaknine, and James Worrell. "Online Monitoring of Metric Temporal Logic". In: *Runtime Verification - 5th International Conference, RV 2014, Toronto, ON, Canada, September 22-25, 2014. Proceedings*. Ed. by Borzoo Bonakdarpour and Scott A. Smolka. Vol. 8734. Lecture Notes in Computer Science. Springer, 2014, pp. 178–192. DOI: 10.1007/978-3-319-11164-3_15. URL: https://doi.org/10.1007/978-3-319-11164-3_15.

[119] Gerard J. Holzmann. "The logic of bugs". In: *SIGSOFT FSE*. ACM, 2002, pp. 81–87.

[120] Gerard J. Holzmann. "The Model Checker SPIN". In: *IEEE Trans. Software Eng.* 23.5 (1997), pp. 279–295.

[121] Bardh Hoxha, Adel Dokhanchi, and Georgios Fainekos. "Mining parametric temporal logic properties in model-based design for cyber-physical systems". In: *Int. J. Softw. Tools Technol. Transf.* 20.1 (2018), pp. 79–93.

[122] Paul Hunter, Joël Ouaknine, and James Worrell. "Expressive Completeness for Metric Temporal Logic". In: *2013 28th Annual ACM/IEEE Symposium on Logic in Computer Science*. 2013, pp. 349–357. DOI: 10.1109/LICS.2013.41.

[123] IEEE Standards Association. *IEEE 1850-2010 – IEEE Standard for Property Specification Language (PSL)*. 2010.

[124] Antonio Ielo, Mark Law, Valeria Fionda, Francesco Ricca, Giuseppe De Giacomo, and Alessandra Russo. "Towards ILP-Based LTL f Passive Learning". In: *Inductive Logic Programming - 32nd International Conference, ILP 2023, Bari, Italy, November 13-15, 2023, Proceedings*. Ed. by Elena Bellodi, Francesca Alessandra Lisi, and Riccardo Zese. Vol. 14363. Lecture Notes in Computer Science. Springer, 2023, pp. 30–45. DOI: 10.1007/978-3-031-49299-0_3. URL: https://doi.org/10.1007/978-3-031-49299-0_3.

[125] Alexey Ignatiev, Antonio Morgado, and Joao Marques-Silva. "PySAT: A Python Toolkit for Prototyping with SAT Oracles". In: *SAT*. 2018, pp. 428–437. DOI: 10.1007/978-3-319-94144-8_26. URL: https://doi.org/10.1007/978-3-319-94144-8_26.

[126] Susmit Jha, Ashish Tiwari, Sanjit A. Seshia, Tuhin Sahai, and Natarajan Shankar. "TeLEx: learning signal temporal logic from positive examples using tightness metric". In: *Formal Methods Syst. Des.* 54.3 (2019), pp. 364–387. DOI: 10.1007/s10703-019-00332-1. URL: https://doi.org/10.1007/s10703-019-00332-1.

[127] Xiaoqing Jin, Alexandre Donzé, Jyotirmoy V. Deshmukh, and Sanjit A. Seshia. "Mining requirements from closed-loop control models". In: *HSCC*. ACM, 2013, pp. 43–52.

[128] Kishor Jothimurugan, Suguman Bansal, Osbert Bastani, and Rajeev Alur. "Compositional Reinforcement Learning from Logical Specifications". In: *NeurIPS*. 2021, pp. 10026–10039.

[129] Aaron Kane, Omar Chowdhury, Anupam Datta, and Philip Koopman. “A Case Study on Runtime Monitoring of an Autonomous Research Vehicle (ARV) System”. In: *RV*. Vol. 9333. Lecture Notes in Computer Science. Springer, 2015, pp. 102–117.

[130] Daniel Kasenberg and Matthias Scheutz. “Interpretable apprenticeship learning with temporal logic specifications”. In: *CDC*. IEEE, 2017, pp. 4914–4921.

[131] Brian Kempa, Pei Zhang, Phillip H. Jones, Joseph Zambreno, and Kristin Yvonne Rozier. “Embedding Online Runtime Verification for Fault Disambiguation on Robonaut2”. In: *FORMATS*. Vol. 12288. Lecture Notes in Computer Science. Springer, 2020, pp. 196–214.

[132] Igor Khmelnitsky, Serge Haddad, Lina Ye, Benoît Barbot, Benedikt Bollig, Martin Leucker, Daniel Neider, and Rajarshi Roy. “Analyzing Robustness of Angluin’s L* Algorithm in Presence of Noise”. In: *Proceedings of the 13th International Symposium on Games, Automata, Logics and Formal Verification, GandALF 2022, Madrid, Spain, September 21-23, 2022*. Ed. by Pierre Ganty and Dario Della Monica. Vol. 370. EPTCS. 2022, pp. 81–96. DOI: 10.4204/EPTCS.370.6. URL: https://doi.org/10.4204/EPTCS.370.6.

[133] Igor Khmelnitsky, Daniel Neider, Rajarshi Roy, Xuan Xie, Benoît Barbot, Benedikt Bollig, Alain Finkel, Serge Haddad, Martin Leucker, and Lina Ye. “Analysis of recurrent neural networks via property-directed verification of surrogate models”. In: *Int. J. Softw. Tools Technol. Transf.* 25.3 (2023), pp. 341–354. DOI: 10.1007/S10009-022-00684-W. URL: https://doi.org/10.1007/s10009-022-00684-w.

[134] Igor Khmelnitsky, Daniel Neider, Rajarshi Roy, Xuan Xie, Benoît Barbot, Benedikt Bollig, Alain Finkel, Serge Haddad, Martin Leucker, and Lina Ye. “Property-Directed Verification and Robustness Certification of Recurrent Neural Networks”. In: *Automated Technology for Verification and Analysis - 19th International Symposium, ATVA 2021, Gold Coast, QLD, Australia, October 18-22, 2021, Proceedings*. Ed. by Zhe Hou and Vijay Ganesh. Vol. 12971. Lecture Notes in Computer Science. Springer, 2021, pp. 364–380. DOI: 10.1007/978-3-030-88885-5_24. URL: https://doi.org/10.1007/978-3-030-88885-5_24.

[135] Joseph Kim, Christian Muise, Ankit Shah, Shubham Agarwal, and Julie Shah. “Bayesian Inference of Linear Temporal Logic Specifications for Contrastive Explanations”. In: *IJCAI*. ijcai.org, 2019, pp. 5591–5598. DOI: 10.24963/ijcai.2019/776. URL: https://doi.org/10.24963/ijcai.2019/776.

[136] Gerwin Klein, June Andronick, Kevin Elphinstone, Gernot Heiser, David Cock, Philip Derrin, Dhammika Elkaduwe, Kai Engelhardt, Rafal Kolanski, Michael Norrish, Thomas Sewell, Harvey Tuch, and Simon Winwood. “seL4: formal verification of an operating-system kernel”. In: *Commun. ACM* 53.6 (2010), pp. 107–115.

[137] Z. Kong, A. Jones, and C. Belta. “Temporal Logics for Learning and Detection of Anomalous Behavior”. In: *IEEE Trans. Autom. Control* 62.3 (2017), pp. 1210–1222.

[138] Zhaodan Kong, Austin Jones, and Calin Belta. "Temporal Logics for Learning and Detection of Anomalous Behavior". In: *IEEE Transactions on Automatic Control* 62.3 (2017), pp. 1210–1222. DOI: 10.1109/TAC.2016.2585083. URL: https://doi.org/10.1109/TAC.2016.2585083.

[139] Zhaodan Kong, Austin Jones, Ana Medina Ayala, Ebru Aydin Gol, and Calin Belta. "Temporal Logic Inference for Classification and Prediction from Data". In: *Proceedings of the 17th International Conference on Hybrid Systems: Computation and Control*. HSCC '14. New York, NY, USA: Association for Computing Machinery, 2014, 273–282. ISBN: 9781450327329. DOI: 10.1145/2562059.2562146.

[140] Sascha Konrad and Betty H. C. Cheng. "Real-time specification patterns". In: *27th International Conference on Software Engineering (ICSE 2005), 15-21 May 2005, St. Louis, Missouri, USA*. Ed. by Gruia-Catalin Roman, William G. Griswold, and Bashar Nuseibeh. ACM, 2005, pp. 372–381. DOI: 10.1145/1062455.1062526. URL: https://doi.org/10.1145/1062455.1062526.

[141] Ron Koymans. "Specifying Real-Time Properties with Metric Temporal Logic". In: *Real Time Syst.* 2.4 (1990), pp. 255–299.

[142] Daniel Kroening and Ofer Strichman. *Decision Procedures - An Algorithmic Point of View, Second Edition*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2016. ISBN: 978-3-662-50496-3. DOI: 10.1007/978-3-662-50497-0. URL: https://doi.org/10.1007/978-3-662-50497-0.

[143] Orna Kupferman and Moshe Y. Vardi. "Model Checking of Safety Properties". In: *Computer Aided Verification, 11th International Conference, CAV '99, Trento, Italy, July 6-10, 1999, Proceedings*. Ed. by Nicolas Halbwachs and Doron A. Peled. Vol. 1633. Lecture Notes in Computer Science. Springer, 1999, pp. 172–183. DOI: 10.1007/3-540-48683-6_17. URL: https://doi.org/10.1007/3-540-48683-6_17.

[144] Caroline Lemieux and Ivan Beschastnikh. "Investigating Program Behavior Using the Texada LTL Specifications Miner". In: *ASE*. IEEE Computer Society, 2015, pp. 870–875.

[145] Caroline Lemieux, Dennis Park, and Ivan Beschastnikh. "General LTL Specification Mining". In: *International Conference on Automated Software Engineering, ASE*. IEEE Computer Society, 2015, pp. 81–92. DOI: 10.1109/ASE.2015.71. URL: https://doi.org/10.1109/ASE.2015.71.

[146] Caroline Lemieux, Dennis Park, and Ivan Beschastnikh. "General LTL Specification Mining (T)". In: *ASE*. IEEE Computer Society, 2015, pp. 81–92.

[147] Wenchao Li, Lili Dworkin, and Sanjit A. Seshia. "Mining assumptions for synthesis". In: *MEMOCODE*. IEEE, 2011, pp. 43–50. DOI: 10.1109/MEMCOD.2011.5970509. URL: https://doi.org/10.1109/MEMCOD.2011.5970509.

[148] Leonardo Lima, Andrei Herasimau, Martin Raszyk, Dmitriy Traytel, and Simon Yuan. "Explainable Online Monitoring of Metric Temporal Logic". In: *TACAS (2)*. Vol. 13994. Lecture Notes in Computer Science. Springer, 2023, pp. 473–491.

[149] Alexis Linard and Jana Tumova. "Active Learning of Signal Temporal Logic Specifications". In: *2020 IEEE 16th International Conference on Automation Science and Engineering (CASE)*. 2020, pp. 779–785. DOI: 10.1109/CASE48305.2020.9216778.

[150] Jason Xinyu Liu, Ziyi Yang, Ifrah Idrees, Sam Liang, Benjamin Schornstein, Stefanie Tellex, and Ankit Shah. "Lang2LTL: Translating Natural Language Commands to Temporal Robot Task Specification". In: *CoRR* abs/2302.11649 (2023). DOI: 10.48550/ARXIV.2302.11649. arXiv: 2302.11649. URL: https://doi.org/10.48550/arXiv.2302.11649.

[151] Damián López and Pedro García. "On the Inference of Finite State Automata from Positive and Negative Data". In: *Topics in Grammatical Inference*. Ed. by Jeffrey Heinz and José M. Sempere. Berlin, Heidelberg: Springer Berlin Heidelberg, 2016, pp. 73–112. ISBN: 978-3-662-48395-4. DOI: 10.1007/978-3-662-48395-4_4. URL: https://doi.org/10.1007/978-3-662-48395-4_4.

[152] Gavin Lowe. "Breaking and Fixing the Needham-Schroeder Public-Key Protocol Using FDR". In: *Softw. Concepts Tools* 17.3 (1996), pp. 93–102.

[153] Weilin Luo, Pingjia Liang, Jianfeng Du, Hai Wan, Bo Peng, and Delong Zhang. "Bridging LTLf Inference to GNN Inference for Learning LTLf Formulae". In: *Thirty-Sixth AAAI Conference on Artificial Intelligence, AAAI 2022, Thirty-Fourth Conference on Innovative Applications of Artificial Intelligence, IAAI 2022, The Twelveth Symposium on Educational Advances in Artificial Intelligence, EAAI 2022 Virtual Event, February 22 - March 1, 2022*. AAAI Press, 2022, pp. 9849–9857. DOI: 10.1609/AAAI.V36I9.21221. URL: https://doi.org/10.1609/aaai.v36i9.21221.

[154] Simon Lutz, Daniel Neider, and Rajarshi Roy. "Specification sketching for Linear Temporal Logic". In: *CoRR* abs/2206.06722 (2022). DOI: 10.48550/arXiv.2206.06722. arXiv: 2206.06722. URL: https://doi.org/10.48550/arXiv.2206.06722.

[155] Simon Lutz, Daniel Neider, and Rajarshi Roy. "Specification Sketching for Linear Temporal Logic". In: *Automated Technology for Verification and Analysis - 21st International Symposium, ATVA 2023, Singapore, October 24-27, 2023, Proceedings, Part II*. Ed. by Étienne André and Jun Sun. Vol. 14216. Lecture Notes in Computer Science. Springer, 2023, pp. 26–48. DOI: 10.1007/978-3-031-45332-8_2. URL: https://doi.org/10.1007/978-3-031-45332-8_2.

[156] Carl Macrae. "Learning from the Failure of Autonomous and Intelligent Systems: Accidents, Safety, and Sociotechnical Sources of Risk". In: *Risk Analysis* 42.9 (2022), pp. 1999–2025. DOI: https://doi.org/10.1111/risa.13850. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/risa.

13850. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/risa.13850.

[157] Oded Maler and Dejan Nickovic. "Monitoring temporal properties of continuous signals". In: *Formal Techniques, Modelling and Analysis of Timed and Fault-Tolerant Systems*. Ed. by Yassine Lakhnech and Sergio Yovine. Vol. 3253. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 152–166. ISBN: 3540231676. DOI: 10.1007/978-3-540-30206-3_12.

[158] Nicolas Markey and Philippe Schnoebelen. "Model Checking a Path". In: *CONCUR*. Vol. 2761. Lecture Notes in Computer Science. Springer, 2003, pp. 248–262.

[159] Corto Mascle, Nathanaël Fijalkow, and Guillaume Lagarde. "Learning temporal formulas from examples is hard". In: *CoRR* abs/2312.16336 (2023). DOI: 10.48550/ARXIV.2312.16336. arXiv: 2312.16336. URL: https://doi.org/10.48550/arXiv.2312.16336.

[160] Sara Mohammadinejad, Jyotirmoy V. Deshmukh, Aniruddh Gopinath Puranic, Marcell Vazquez-Chanlatte, and Alexandre Donzé. "Interpretable classification of time-series data using efficient enumerative techniques". In: *HSCC*. Ed. by Aaron D. Ames, Sanjit A. Seshia, and Jyotirmoy Deshmukh. ACM, 2020, 9:1–9:10. DOI: 10.1145/3365365.3382218. URL: https://doi.org/10.1145/3365365.3382218.

[161] Mehryar Mohri, Afshin Rostamizadeh, and Ameet Talwalkar. *Foundations of Machine Learning*. Adaptive computation and machine learning. MIT Press, 2012. ISBN: 978-0-262-01825-8. URL: http://mitpress.mit.edu/books/foundations-machine-learning-0.

[162] Christoph Molnar. *Interpretable Machine Learning. A Guide for Making Black Box Models Explainable*. 2nd ed. 2022. URL: https://christophm.github.io/interpretable-ml-book.

[163] Leonardo Mendonça de Moura and Nikolaj S. Bjørner. "Satisfiability modulo theories: introduction and applications". In: *Commun. ACM* 54.9 (2011), pp. 69–77.

[164] Leonardo Mendonça de Moura and Nikolaj S. Bjørner. "Z3: An Efficient SMT Solver". In: *TACAS*. Vol. 4963. Lecture Notes in Computer Science. Springer, 2008, pp. 337–340. DOI: 10.1007/978-3-540-78800-3_24. URL: https://doi.org/10.1007/978-3-540-78800-3_24.

[165] Edi Muškardin, Bernhard Aichernig, Ingo Pill, Andrea Pferscher, and Martin Tappler. "AALpy: an active automata learning library". In: *Innovations in Systems and Software Engineering* (Mar. 2022), pp. 1–10. DOI: 10.1007/s11334-022-00449-3.

[166] Anusha Nagabandi, Kurt Konoglie, Sergey Levine, and Vikash Kumar. "Deep Dynamics Models for Learning Dexterous Manipulation". In: (2019), pp. 1–12. arXiv: arXiv:1909.11652v1.

[167] Daniel Neider. "Applications of automata learning in verification and synthesis". PhD thesis. RWTH Aachen University, 2014. URL: http://darwin.bth.rwth-aachen.de/opus3/volltexte/2014/5169.

[168] Daniel Neider. "Computing Minimal Separating DFAs and Regular Invariants Using SAT and SMT Solvers". In: *10th International Symposium of Automated Technology for Verification and Analysis, ATVA '12*. Vol. 7561. LNCS. Springer, 2012, pp. 354–369. DOI: 10.1007/978-3-642-33386-6_28.

[169] Daniel Neider and Ivan Gavran. "Learning Linear Temporal Properties". In: *2018 Formal Methods in Computer Aided Design, FMCAD 2018, Austin, TX, USA, October 30 - November 2, 2018*. Ed. by Nikolaj Bjørner and Arie Gurfinkel. IEEE, 2018, pp. 1–10. DOI: 10.23919/FMCAD.2018.8603016. URL: https://doi.org/10.23919/FMCAD.2018.8603016.

[170] Daniel Neider and Ivan Gavran. *Learning Linear Temporal Properties*. 2018. DOI: 10.48550/ARXIV.1806.03953. URL: https://arxiv.org/abs/1806.03953.

[171] Laura Nenzi, Simone Silvetti, Ezio Bartocci, and Luca Bortolussi. "A Robust Genetic Algorithm for Learning Temporal Specifications from Data". In: *Quantitative Evaluation of Systems*. Ed. by Annabelle McIver and Andras Horvath. Cham: Springer International Publishing, 2018, pp. 323–338. ISBN: 978-3-319-99154-2.

[172] Allen P. Nikora and Galen Balcom. "Automated Identification of LTL Patterns in Natural Language Requirements". In: *ISSRE 2009, 20th International Symposium on Software Reliability Engineering, Mysuru, Karnataka, India, 16-19 November 2009*. IEEE Computer Society, 2009, pp. 185–194. DOI: 10.1109/ISSRE.2009.15. URL: https://doi.org/10.1109/ISSRE.2009.15.

[173] Yoonseon Oh, Roma Patel, Thao Nguyen, Baichuan Huang, Ellie Pavlick, and Stefanie Tellex. "Planning with State Abstractions for Non-Markovian Task Specifications". In: *Robotics: Science and Systems XV, University of Freiburg, Freiburg im Breisgau, Germany, June 22-26, 2019*. Ed. by Antonio Bicchi, Hadas Kress-Gazit, and Seth Hutchinson. 2019. DOI: 10.15607/RSS.2019.XV.059. URL: https://doi.org/10.15607/RSS.2019.XV.059.

[174] J. Oncina and P. García. "INFERRING REGULAR LANGUAGES IN POLYNOMIAL UPDATED TIME". In: *Pattern Recognition and Image Analysis*, pp. 49–61. DOI: 10.1142/9789812797902_0004. eprint: https://www.worldscientific.com/doi/pdf/10.1142/9789812797902_0004. URL: https://www.worldscientific.com/doi/abs/10.1142/9789812797902_0004.

[175] Joël Ouaknine and James Worrell. "Some Recent Results in Metric Temporal Logic". In: *Formal Modeling and Analysis of Timed Systems*. Ed. by Franck Cassez and Claude Jard. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 1–13. ISBN: 978-3-540-85778-5.

[176] Antti Pakonen, Cheng Pang, Igor Buzhinsky, and Valeriy Vyatkin. “User-friendly formal specification languages - conclusions drawn from industrial experience on model checking”. In: *ETFA*. IEEE, 2016, pp. 1–8.

[177] Jiayi Pan, Glen Chou, and Dmitry Berenson. “Data-Efficient Learning of Natural Language to Linear Temporal Logic Translators for Robot Task Specification”. In: *IEEE International Conference on Robotics and Automation, ICRA 2023, London, UK, May 29 - June 2, 2023*. IEEE, 2023, pp. 11554–11561. DOI: 10.1109/ICRA48891.2023.10161125. URL: https://doi.org/10.1109/ICRA48891.2023.10161125.

[178] Maja Pesic, M. H. Schonenberg, Natalia Sidorova, and Wil M. P. van der Aalst. “Constraint-Based Workflow Models: Change Made Easy”. In: *On the Move to Meaningful Internet Systems 2007: CoopIS, DOA, ODBASE, GADA, and IS, OTM Confederated International Conferences CoopIS, DOA, ODBASE, GADA, and IS 2007, Vilamoura, Portugal, November 25-30, 2007, Proceedings, Part I*. Ed. by Robert Meersman and Zahir Tari. Vol. 4803. Lecture Notes in Computer Science. Springer, 2007, pp. 77–94. DOI: 10.1007/978-3-540-76848-7_7. URL: https://doi.org/10.1007/978-3-540-76848-7_7.

[179] Amir Pnueli. “The Temporal Logic of Programs”. In: *18th Annual Symposium of Foundations of Computer Science, FOCS ’77*. IEEE Computer Society, 1977, pp. 46–57. DOI: 10.1109/SFCS.1977.32. URL: https://doi.org/10.1109/SFCS.1977.32.

[180] Amir Pnueli. “The Temporal Logic of Programs”. In: *Proc. 18th Annu. Symp. Found. Computer Sci.* 1977, pp. 46–57.

[181] Amir Pnueli and Roni Rosner. “On the Synthesis of a Reactive Module”. In: *POPL*. ACM Press, 1989, pp. 179–190.

[182] J. Ross Quinlan. “Induction of Decision Trees”. In: *Mach. Learn.* 1.1 (1986), pp. 81–106.

[183] Michael Rabin and Dana Scott. “Finite Automata and Their Decision Problems”. In: *IBM Journal of Research and Development* 3 (Apr. 1959), pp. 114–125. DOI: 10.1147/rd.32.0114.

[184] Harald Raffelt, Bernhard Steffen, Therese Berg, and Tiziana Margaria. “LearnLib: a framework for extrapolating behavioral models”. In: *Int. J. Softw. Tools Technol. Transf.* 11.5 (2009), pp. 393–407. DOI: 10.1007/S10009-009-0111-8. URL: https://doi.org/10.1007/s10009-009-0111-8.

[185] Ritam Raha. “Learning and verifying temporal specifications for cyber-physical systems”. PhD thesis. University of Antwerp, Belgium, 2023. URL: https://hdl.handle.net/10067/1986580151162165141.

[186] Ritam Raha, Rajarshi Roy, Nathanaël Fijalkow, and Daniel Neider. "Scalable Anytime Algorithms for Learning Fragments of Linear Temporal Logic". In: *Tools and Algorithms for the Construction and Analysis of Systems*. Ed. by Dana Fisman and Grigore Rosu. Cham: Springer International Publishing, 2022, pp. 263–280. ISBN: 978-3-030-99524-9.

[187] Ritam Raha, Rajarshi Roy, Nathanaël Fijalkow, and Daniel Neider. "Scalable Anytime Algorithms for Learning Fragments of Linear Temporal Logic". In: *TACAS (1)*. Vol. 13243. Lecture Notes in Computer Science. Springer, 2022, pp. 263–280.

[188] Ritam Raha, Rajarshi Roy, Nathanaël Fijalkow, and Daniel Neider. "Scarlet: Scalable Anytime Algorithms for Learning Fragments of Linear Temporal Logic". In: *J. Open Source Softw.* 9.93 (2024), p. 5052. DOI: 10.21105/JOSS.05052. URL: https://doi.org/10.21105/joss.05052.

[189] Ritam Raha, Rajarshi Roy, Nathanaël Fijalkow, and Daniel Neider. "Scarlet: Scalable Anytime Algorithms for Learning Fragments of Linear Temporal Logic". In: *J. Open Source Softw.* 9.93 (2024), p. 5052. DOI: 10.21105/JOSS.05052. URL: https://doi.org/10.21105/joss.05052.

[190] Ritam Raha, Rajarshi Roy, Nathanaël Fijalkow, Daniel Neider, and Guillermo A. Pérez. "Synthesizing Efficiently Monitorable Formulas in Metric Temporal Logic". In: *Verification, Model Checking, and Abstract Interpretation - 25th International Conference, VMCAI 2024, London, United Kingdom, January 15-16, 2024, Proceedings, Part II*. Ed. by Rayna Dimitrova, Ori Lahav, and Sebastian Wolff. Vol. 14500. Lecture Notes in Computer Science. Springer, 2024, pp. 264–288. DOI: 10.1007/978-3-031-50521-8_13. URL: https://doi.org/10.1007/978-3-031-50521-8_13.

[191] Ritam Raha, Rajarshi Roy, Nathanaël Fijalkow, and Daniel Neider. *SCARLET: Scalable Anytime Algorithm for Learning LTL*. Jan. 2022. DOI: 10.5281/zenodo.5890149. URL: https://doi.org/10.5281/zenodo.5890149.

[192] Andrew Reynolds, Haniel Barbosa, Andres Nötzli, Clark W. Barrett, and Cesare Tinelli. "cvc4sy: Smart and Fast Term Enumeration for Syntax-Guided Synthesis". In: *Computer-Aided Verification, CAV*. 2019. DOI: 10.1007/978-3-030-25543-5_5.

[193] Heinz Riener. "Exact Synthesis of LTL Properties from Traces". In: *FDL*. IEEE, 2019, pp. 1–6. DOI: 10.1109/FDL.2019.8876900. URL: https://doi.org/10.1109/FDL.2019.8876900.

[194] "Robot kills worker at Volkswagen plant in Germany". In: *The Guardian* (2015). URL: https://www.theguardian.com/world/2015/jul/02/robot-kills-worker-at-volkswagen-plant-in-germany (visited on 11/10/2022).

[195] Rajarshi Roy, Dana Fisman, and Daniel Neider. "Learning Interpretable Models in the Property Specification Language". In: *IJCAI*. ijcai.org, 2020, pp. 2213–2219.

[196] Rajarshi Roy, Jean-Raphaël Gaglione, Nasim Baharisangari, Daniel Neider, Zhe Xu, and Ufuk Topcu. "Learning Interpretable Temporal Properties from Positive Examples Only". In: *Proceedings of the AAAI Conference on Artificial Intelligence* 37.5 (2023), pp. 6507–6515. DOI: 10.1609/aaai.v37i5.25800. URL: https://ojs.aaai.org/index.php/AAAI/article/view/25800.

[197] Kristin Yvonne Rozier. "Specification: The Biggest Bottleneck in Formal Methods and Autonomy". In: *VSTTE*. Vol. 9971. Lecture Notes in Computer Science. 2016, pp. 8–26. DOI: 10.1007/978-3-319-48869-1_2.

[198] Rainer Schlör, Bernhard Josko, and Dieter Werth. "Using a Visual Formalism for Design Verification in Industrial Environments". In: *Services and Visualization: Towards User-Friendly Design*. Vol. 1385. Lecture Notes in Computer Science. Springer, 1998, pp. 208–221.

[199] Roberto Sebastiani and Patrick Trentin. "On Optimization Modulo Theories, MaxSMT and Sorting Networks". In: *CoRR* abs/1702.02385 (2017). arXiv: 1702.02385. URL: http://arxiv.org/abs/1702.02385.

[200] Ankit Shah, Pritish Kamath, Julie A. Shah, and Shen Li. "Bayesian Inference of Temporal Task Specifications from Demonstrations". In: *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*. Ed. by Samy Bengio, Hanna M. Wallach, Hugo Larochelle, Kristen Grauman, Nicolò Cesa-Bianchi, and Roman Garnett. 2018, pp. 3808–3817. URL: https://proceedings.neurips.cc/paper/2018/hash/13168e6a2e6c84b4b7de9390c0ef5ec5-Abstract.html.

[201] Simone Silvetti, Laura Nenzi, Luca Bortolussi, and Ezio Bartocci. "A Robust Genetic Algorithm for Learning Temporal Specifications from Data". In: *CoRR* (2017). arXiv: 1711.06202. URL: http://arxiv.org/abs/1711.06202.

[202] Michael Sipser. *Introduction to the theory of computation*. PWS Publishing Company, 1997.

[203] A. Prasad Sistla and Edmund M. Clarke. "The Complexity of Propositional Linear Temporal Logics". In: *J. ACM* 32.3 (1985), pp. 733–749.

[204] Armando Solar-Lezama. "Program sketching". In: *Int. J. Softw. Tools Technol. Transf.* 15.5-6 (2013), pp. 475–495.

[205] Armando Solar-Lezama, Rodric M. Rabbah, Rastislav Bodík, and Kemal Ebcioglu. "Programming by sketching for bit-streaming programs". In: *PLDI*. ACM, 2005, pp. 281–294.

[206] Roni Stern and Brendan Juba. "Efficient, Safe, and Probably Approximately Complete Learning of Action Models". In: *IJCAI*. ijcai.org, 2017, pp. 4405–4411.

[207] Andreas Stolcke and Stephen Omohundro. "Hidden Markov Model Induction by Bayesian Model Merging". In: *Advances in Neural Information Processing Systems*. Ed. by S. Hanson, J. Cowan, and C. Giles. Vol. 5. Morgan-Kaufmann, 1992. URL: https://proceedings.neurips.cc/paper/1992/file/5c04925674920eb58467fb52ce4ef728-Paper.pdf.

[208] Prasanna Thati and Grigore Rosu. "Monitoring Algorithms for Metric Temporal Logic Specifications". In: *Proceedings of the Fourth Workshop on Runtime Verification, RV@ETAPS 2004, Barcelona, Spain, April 3, 2004*. Ed. by Klaus Havelund and Grigore Rosu. Vol. 113. Electronic Notes in Theoretical Computer Science. Elsevier, 2004, pp. 145–162. DOI: 10.1016/j.entcs.2004.01.029. URL: https://doi.org/10.1016/j.entcs.2004.01.029.

[209] Wolfgang Thomas. "Automata on Infinite Objects". In: *Handbook of Theoretical Computer Science, Volume B: Formal Models and Semantics*. Ed. by Jan van Leeuwen. Elsevier and MIT Press, 1990, pp. 133–191. DOI: 10.1016/B978-0-444-88074-1.50009-3. URL: https://doi.org/10.1016/b978-0-444-88074-1.50009-3.

[210] Wolfgang Thomas. "Star-Free Regular Sets of omega-Sequences". In: *Inf. Control.* 42.2 (1979), pp. 148–156. DOI: 10.1016/S0019-9958(79)90629-6. URL: https://doi.org/10.1016/S0019-9958(79)90629-6.

[211] Ken Thompson. "Programming Techniques: Regular Expression Search Algorithm". In: *Commun. ACM* 11.6 (June 1968), 419–422. ISSN: 0001-0782. DOI: 10.1145/363347.363387. URL: https://doi.org/10.1145/363347.363387.

[212] G. S. Tseitin. "On the Complexity of Derivation in Propositional Calculus". In: *Automation of Reasoning: 2: Classical Papers on Computational Logic 1967–1970*. Springer Berlin Heidelberg, 1983, pp. 466–483.

[213] Moshe Y. Vardi. "Branching vs. Linear Time: Final Showdown". In: *TACAS*. Vol. 2031. Lecture Notes in Computer Science. Springer, 2001, pp. 1–22.

[214] Marcell Vazquez-Chanlatte, Susmit Jha, Ashish Tiwari, Mark K. Ho, and Sanjit A. Seshia. "Learning Task Specifications from Demonstrations". In: *NeurIPS*. 2018, pp. 5372–5382.

[215] Eric Verhulst and Gjalt G. de Jong. "OpenComRTOS: An Ultra-Small Network Centric Embedded RTOS Designed Using Formal Modeling". In: *SDL Forum*. Vol. 4745. Lecture Notes in Computer Science. Springer, 2007, pp. 258–271.

[216] James Vincent. "Microsoft's Bing is an emotionally manipulative liar, and people love it". In: *The Verge* (2023). URL: https://www.theverge.com/2023/2/15/23599072/microsoft-ai-bing-personality-conversations-spy-employees-webcams (visited on 02/15/2022).

[217] Heiko Vogler and Andreas Maletti, eds. *Proceedings of the 14th International Conference on Finite-State Methods and Natural Language Processing, FSMNLP 2019, Dresden, Germany, September 23-25, 2019*. Association for Computational Linguistics, 2019. URL: https://aclanthology.org/volumes/W19-31/.

[218] Andrzej Wasylkowski and Andreas Zeller. "Mining Temporal Specifications from Object Usage". In: *24th IEEE/ACM International Conference on Automated Software Engineering, ASE '09*. IEEE Computer Society, 2009, pp. 295–306. DOI: 10.1109/ASE.2009.30. URL: https://doi.org/10.1109/ASE.2009.30.

[219] Andrzej Wasylkowski and Andreas Zeller. "Mining temporal specifications from object usage". In: *Autom. Softw. Eng.* 18.3-4 (2011), pp. 263–292.

[220] "What is generative AI". In: *Mckinsey & Company* (2023). URL: https://www.mckinsey.com/featured-insights/mckinsey-explainers/what-is-generative-ai (visited on 11/30/2017).

[221] Thomas Wilke. "Past, Present, and Infinite Future". In: *43rd International Colloquium on Automata, Languages, and Programming, ICALP 2016, July 11-15, 2016, Rome, Italy*. Ed. by Ioannis Chatzigiannakis, Michael Mitzenmacher, Yuval Rabani, and Davide Sangiorgi. Vol. 55. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016, 95:1–95:14. DOI: 10.4230/LIPICS.ICALP.2016.95. URL: https://doi.org/10.4230/LIPIcs.ICALP.2016.95.

[222] Pierre Wolper. "Temporal Logic Can Be More Expressive". In: *22nd Annual Symposium on Foundations of Computer Science, FOCS '81*. IEEE Computer Society, 1981, pp. 340–348. DOI: 10.1109/SFCS.1981.44. URL: https://doi.org/10.1109/SFCS.1981.44.

[223] Z. Xu, A. J. Nettekoven, A. Agung Julius, and U. Topcu. "Graph Temporal Logic Inference for Classification and Identification". In: *2019 IEEE 58th Conference on Decision and Control (CDC)*. 2019, pp. 4761–4768.

[224] Z. Xu, M. Ornik, A. A. Julius, and U. Topcu. "Information-Guided Temporal Logic Inference with Prior Knowledge". In: *2019 American Control Conference (ACC)*. 2019, pp. 1891–1897.

[225] Zhe Xu and A Agung Julius. "Robust Temporal Logic Inference for Provably Correct Fault Detection and Privacy Preservation of Switched Systems". In: *IEEE Systems Journal* 13.3 (2019), pp. 3010–3021.

[226] Zhe Xu, Alexander J Nettekoven, A. Agung Julius, and Ufuk Topcu. "Graph Temporal Logic Inference for Classification and Identification". In: *2019 IEEE 58th Conference on Decision and Control (CDC)*. Nice, France: IEEE Press, 2019, 4761–4768. DOI: 10.1109/CDC40024.2019.9029181. URL: https://doi.org/10.1109/CDC40024.2019.9029181.

[227] Zhe Xu, Melkior Ornik, A. Agung Julius, and Ufuk Topcu. "Information-Guided Temporal Logic Inference with Prior Knowledge". In: *2019 American Control Conference (ACC)*. 2019, pp. 1891–1897. DOI: 10.23919/ACC.2019.8815145.

[228] Zhe Xu and Ufuk Topcu. "Transfer of Temporal Logic Formulas in Reinforcement Learning". In: *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI 2019, Macao, China, August 10-16, 2019*. Ed. by Sarit Kraus. ijcai.org, 2019, pp. 4010–4018. DOI: 10.24963/ijcai.2019/557. URL: https://doi.org/10.24963/ijcai.2019/557.

[229] Hengyi Yang, Bardh Hoxha, and Georgios E. Fainekos. "Querying Parametric Temporal Logic Properties on Embedded Systems". In: *Testing Software and Systems - 24th IFIP WG 6.1 International Conference, ICTSS 2012, Aalborg, Denmark, November 19-21, 2012. Proceedings*. Ed. by Brian Nielsen and Carsten Weise. Vol. 7641. Lecture Notes in Computer Science. Springer, 2012, pp. 136–151. DOI: 10.1007/978-3-642-34691-0_11. URL: https://doi.org/10.1007/978-3-642-34691-0_11.

[230] Lina Ye, Igor Khmelnitsky, Serge Haddad, Benoît Barbot, Benedikt Bollig, Martin Leucker, Daniel Neider, and Rajarshi Roy. "Analyzing Robustness of Angluin's L$^*$ Algorithm in Presence of Noise". In: *Log. Methods Comput. Sci.* 20.1 (2024). DOI: 10.46298/LMCS-20(1:22)2024. URL: https://doi.org/10.46298/lmcs-20(1:22)2024.

[231] Shufang Zhu, Lucas M. Tabajara, Jianwen Li, Geguang Pu, and Moshe Y. Vardi. "Symbolic LTLf Synthesis". In: *IJCAI*. ijcai.org, 2017, pp. 1362–1369.

# Curriculum Vitae

## Research Interests

Formal Verification, Explainable AI, Reinforcement Learning

## Education and employment

*2024 –* Research Associate, University of Oxford, UK.

*2019 – 2024* Doctoral student, Max Planck Institute for Software Systems (MPI-SWS), Kaiserslautern, Germany.

*2017 – 2019* Masters in Computer Science, Chennai Mathematical Institute (CMI), India.

*2014 – 2017* Bachelors in Mathematics and Computer Science, Chennai Mathematical Institute (CMI), India.