TU WIEN Informatics

Specification Mining for Cyber-Physical Systems

DISSERTATION

submitted in partial fulfillment of the requirements for the degree of

Doktorin der Technischen Wissenschaften

by

MSc Eleonora Nesterini, BSc
Registration Number 12122058

to the Faculty of Informatics
at the TU Wien

Advisor: Prof. Ezio Bartocci
Second advisor: Priv-doz. Dr. Dejan Ničković

The dissertation has been reviewed by:

Thao Dang

Andrea Vandin

Vienna, May 16, 2025

Eleonora Nesterini

Erklärung zur Verfassung der Arbeit

MSc Eleonora Nesterini, BSc

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.
Ich erkläre weiters, dass ich mich generativer KI-Tools lediglich als Hilfsmittel bedient habe und in der vorliegenden Arbeit mein gestalterischer Einfluss überwiegt. Im Anhang „Übersicht verwendeter Hilfsmittel“ habe ich alle generativen KI-Tools gelistet, die verwendet wurden, und angegeben, wo und wie sie verwendet wurden. Für Textpassagen, die ohne substantielle Änderungen übernommen wurden, habe ich jeweils die von mir formulierten Eingaben (Prompts) und die verwendete IT-Anwendung mit ihrem Produktnamen und Versionsnummer/Datum angegeben.

Wien, 16. Mai 2025

Eleonora Nesterini

To the brightest lights in all my worlds,
mio babbo Francesco e mia nonna Dina

Acknowledgements

This thesis and the work carried out over the past four years wouldn't have been possible without the amazing people around me —some who have always been by my side and others who joined me along the way. I'd like to take a moment to thank all of them.

First, my infinite and everlasting gratitude goes to my loving and supportive family.
I want to thank my mum Gabriella for being the best role model I could have asked for. You are the perfect mix of strength, empathy, dedication, self-confidence, affection, and care, and I'm so proud to call you *Mamma*. Thank you for always telling me that I can achieve whatever I want in life; I've started believing it.
I need to thank my brother Alessandro for being my quiet and loyal ally in this life. People have always guessed we were twins, and I think they were somehow right. Our souls are more alike than I ever imagined, and I am so happy to have you by my side. I will always be there for you.
I struggle to properly express my gratitude for my nonna Dina. My admiration for her was born with me. You have filled my life with peace, kindness, love, and sweetness and I still can't believe how lucky I am to have the imprint of your calm and pure smile guiding me through life.
I am deeply grateful to my dad Francesco for staying with me this whole time and sending help countless times. I could never have managed all of this without you. You are my strength.
Thank you to the rest of my family —Laura, Enea, Ambra, Graziella, Luca, Renato, Bruna, and Nello— for supporting and respecting my decisions, and for always arranging endless, delicious lunches every time I'm back in Caldine or Castiglioncello.
I also want to express my love to the adorable little Bowie. You are the purest and cutest creature that ever existed. Thank you for blessing my family with your silent yet comforting presence.

My heart is of full of appreciation for the warm and constant support of my best friends since forever: Giuditta, Giulia, Linda, Sara, and Silvia. Before leaving for Vienna, I was sure that our bond would not be scratched by the time apart, and I was right. I'm so happy that every time I see you, it feels like we just met the day before. In a world full of uncertainties, you have always been my safe harbor. When the ground started shaking under my feet, you were my solid pillars. When my heart was broken into thousands of pieces, you sat with me and helped me reconstruct the puzzle, piece by piece. I feel blessed to have shared almost our entire lives together, and I am so excited to see what the rest will look like. I love you so much.

Timmi, we both know that this crazy adventure would have never started if it weren't for you. I might have cursed you for this (just once or twice, though), but now I don't know how to even begin expressing my gratitude for you. You stood by me, lent me your hand to step out of my comfort zone, encouraged me, and believed in me even more than I believed in myself. You helped me grow and evolve. You were there to listen to all my complaints and to share my awe for this new life. You have been the first person I have reached out to with every single piece of news about my PhD journey. Our connection survived the wildest storm, only to become even stronger and more mature. Thank you for helping me shape my past, present, and future life.

Thank you to my supervisors, Ezio and Dejan, for being my guides during the past four years. I'm deeply grateful for the opportunity you gave me, as well as the constant and patient support you offered throughout the ups and downs of this journey. Even when I felt most discouraged and doubtful, your trust in the quality of our work and your vision for the next steps were the fuel I needed to dive into new challenges and keep my motivation strong.

I'm so glad I've spent most of my days in Vienna sharing the office with some of the nicest people I've met in this city.
Thank you, Drishti, for your caring presence, for being my female support in this world full of men, and for your enthusiasm about our crazy skydiving experience.
Thank you, Francesco, for all the hours we spent venting, idling, and complaining; it meant so much to share fears and dreams with you.
Thank you, Ignacio, for being my coach in both practicing Spanish and running training; we might not have our desired AR T-shirt yet, but I'm glad we gained time together outside the office, which even led to our precious medals.
I'm still shocked that all my office mates are going (or have been) through a PhD without ever drinking coffee —chapeau! I appreciate that, despite this, none of you ever complained about the smell of my coffee filling the office multiple times a day.
Thank you also to our neighbors, Mahum and Luigi, for your daily smiles, for stopping by our door to chat on your way to the kitchen, and for the comforting light coming from your open office doors on an otherwise empty floor.
A sincere and warm thank you to the rest of the CPS group, both past and present —Ennio, Stefan, Elisa, Michele, Wasim, Monika, Andreas, Axel, Luca, Martin, Lilly, and Emery— and to my colleagues at AIT —Edgar, Cristinel, Alessio, Anagha, Felix, Stefan, Sandra, Laurenz, and Athina. Your example, support, and insightful discussions have inspired me in many ways. But even more importantly, our coffee breaks, kitchen chats, pasta cooking sessions, and shared cakes brightened the dark winters of my years in Vienna, providing just the right amount of sugar to keep both life and research sweet and energetic.

I am extremely thankful to this journey for having gifted me a wonderful group of new friends who, little by little, have become my Viennese family: Ferdinando, Ludovica,

Josephine, Luigi, Oliviero, Federica, Luisa, Elena, Francesco, Giovanni, Davide, and Michele. Thank you for the countless aperitivi at “La Stella”, for being a support network through the Austrian bureaucracy —rigorously in harsh Deutsch—, for the fruitful discussions about the best pizzerias in the city, and for the yearly Sanremo and Eurovision marathons. The time spent together was essential for recharging my social batteries and also for discovering how little I know about the rest of Italy. I promise I will catch up soon, starting with neomelodic songs and the term *cadrega*.

I would like to express my gratitude to Prof. Sanjit Seshia for hosting me and giving me the opportunity to spend a wonderful semester at the University of California, Berkeley. My time in California let me live out my teenage dream —and even run my first marathon (which was definitely not part of that dream!). But those extraordinary months in the Bay Area wouldn’t have been nearly as incredible without the amazing people I met there: Lorenzo, Birgit, Cam, Cécile, Marie, Louis, Jan, Matthew, Yessie, Dennis, Francesco, Markus, Marco, Joy, Matteo, Alberto, Ameesh, Federico, and Victoria. Thank you for making me feel at home, even on the other side of the world. I owe a special, heartfelt thanks to Cam, Oreo, and Vanilla for being my sources of lightness, calm, and joy during some of the most delicate weeks of my life. All the logic contained in this thesis will never be able to explain how it was even possible that, during that period, I could laugh and love life as much as I did when I had you around. A piece of my heart will always beat in sync with those nine hours of time difference.

Bowie celebrating the Los Angeles Marathon with a delicious artichoke leaf.

Kurzfassung

Cyber-Physical Systems (CPS) sind weit verbreitete, komplexe Systeme, die rechnergestützte und physikalische Komponenten integrieren und so eine kontinuierliche Interaktion mit der realen Umgebung ermöglichen. In den letzten Jahren hat die Integration von Machine-Learning-Komponenten zur Entwicklung lernfähiger CPS geführt, die autonom agieren können. CPS finden Anwendung in verschiedenen Bereichen, darunter autonomes Fahren, medizinische Geräte, Smart Cities, Robotersysteme und intelligente Stromnetze.

Mit zunehmender Komplexität der CPS wird die Sicherstellung ihrer Vertrauenswürdigkeit immer herausfordernder, bleibt jedoch insbesondere in sicherheitskritischen Bereichen essenziell. Formale Spezifikationen sind entscheidend, um Anforderungen an das Systemverhalten eindeutig zu formulieren. Ihre mathematische Struktur ermöglicht zudem die Automatisierung von Laufzeitverifikationsverfahren – wie Überwachung, Laufzeitdurchsetzung und Falsifikation –, die eine skalierbare Überprüfung von Systemausführungen bieten. Trotz ihrer Bedeutung sind formale Spezifikationen oft nicht verfügbar, da ihre manuelle Definition aufgrund der Systemkomplexität, unvorhersehbarer Umgebungen und der großen Menge gesammelter Daten schwierig ist. Das Forschungsfeld des Specification Mining befasst sich mit der automatischen Ableitung von Systemeigenschaften durch die Analyse von Systemausführungen und deren Interaktionen mit der Umgebung.

In dieser Arbeit adressieren wir mehrere Forschungslücken im Bereich des Specification Mining für CPS. Insbesondere konzentrieren wir uns auf Signal Temporal Logic (STL), ein weit verbreitetes formales Mittel zur verständlichen Darstellung zeitlicher Eigenschaften von CPS. Wir präsentieren eine Methode zur automatischen Generierung von STL-Spezifikationen, die Sicherheitsverletzungen frühzeitig anhand von ausschließlich beobachtbaren Systemvariablen vorhersagen kann. Darüber hinaus entwickeln wir eine neuartige Technik zur Bestimmung optimaler Parameterwerte für STL-Formeln, um eine Multi-Klassen-Klassifikation von Systemausführungen durchzuführen. Anschließend stellen wir die erste Methode zur Extraktion von STL-Hyperproperties vor, welche die Ausdruckskraft von Eigenschaften erweitert, indem sie Zusammenhänge zwischen mehreren Systemausführungen erfasst. Schließlich untersuchen wir die Form von Systemausführungen und extrahieren formale Spezifikationen zur Charakterisierung ihrer geometrischen Muster, die insbesondere für die Analyse von Zeitreihendaten mit wiederkehrenden Mustern nützlich sind. Wir bewerten die Leistung jeder vorgeschlagenen Methode anhand mehrerer Fallstudien sowie die Interpretierbarkeit der Formeln.

Abstract

Cyber-Physical Systems (CPS) are widely used complex systems that integrate computational and physical components, enabling continuous interaction with the real-world environment. In recent years, the integration of Machine Learning components has given rise to learning-enabled CPS, allowing them to operate autonomously. CPS applications span various domains, including self-driving vehicles, medical devices, smart cities, robotic systems, and smart grids.

As CPS grow in complexity, ensuring their trustworthiness has become increasingly challenging but remains essential for their adoption, especially in safety-critical domains. In this context, formal specifications are crucial for expressing requirements on system behavior in a precise and unambiguous way. Additionally, their mathematical structure enables the automation of runtime verification techniques –such as monitoring, runtime enforcement, and falsification– which provide rigorous yet scalable methods for verifying system executions. Despite their importance, formal specifications are often unavailable because manually defining them is challenging due to system complexity and unpredictable environments. Specification mining addresses this issue as it is the research field dedicated to automatically inferring system properties from its executions and interactions with the environment.

In this thesis, we address several research gaps in specification mining for CPS. Specifically, we focus on Signal Temporal Logic (STL), a popular formalism for expressing temporal properties of CPS in a rigorous yet human-understandable manner, and its extensions. We introduce a method for learning STL specifications that can predict safety violations in advance based only on variables that are observable by the system, thus allowing the autonomous assessment of its safety at runtime. Moreover, we propose a new technique for learning parameter values for STL formulas to perform multi-class classification of system executions, which is essential, for instance, in recognizing different system failures simultaneously. We then develop the first method for mining STL hyperproperties, which enhance the expressiveness of properties by capturing relationships across multiple system executions. Finally, we focus on the shape of the executions and extract formal specifications to characterize their geometric patterns, which are particularly useful for analyzing time-series data with repeated patterns, such as cardiac activity or voltage signals. We evaluate the performance of each proposed method on several case studies, as well as the interpretability of the learned formulas.

Contents

CHAPTER 1

Introduction

1.1 Motivation

Cyber-Physical Systems (CPS) are a broad class of complex systems in which computational units are designed to operate in close integration with their physical components, which are necessary for interacting with the real-world environment. CPS have increasingly been employed to replace humans in complicated, dangerous, and time-consuming tasks and can now be found everywhere: from medical devices and robotic systems to smart buildings, self-driving vehicles, smart grids, and drones. Moreover, recent advancements in Artificial Intelligence (AI) and Machine Learning (ML) have enabled the advent of autonomous CPS, which are equipped with fully independent decision-making processes [SSS22]. This new feature has led to a previously unseen increase in both CPS diffusion and complexity. However, CPS operate in a wide variety of domains where their potential incorrect or unexpected behaviors can have severe consequences, such as privacy violations, data leakage, financial loss, and harm to people. In safety-critical domains such as the automotive sector, aviation, and medical services, system failures may even threaten human lives. Given these risks, ensuring the reliability of CPS and building trust in their decisions and actions is of primary importance.

Standard verification techniques [DS19] aim to guarantee that all the executions generated by a system –potentially infinite in number– satisfy a given property. Examples of such exhaustive techniques include *model checking* [BK08], which analyzes a model of the system to automatically derive a formal correctness certificate, and *theorem proving* [ORS92, NWP02], which assists users in constructing formal proofs using deductive rules. Other approaches for formal verification include the computation of *invariants* [AD22], namely sets of states that the system can never leave once entered, and *reachability analysis* [LG10, AGD17, TYML+20], which computes (or over-approximates) the set of reachable states for a given set of initial conditions within a specified time horizon. Existing tools in this field focus on specific classes of systems. For example, SpaceEx [FLGD+11] verifies hybrid systems under the assumptions that both the continuous dynamics and the switching constraints are linear, while the method proposed in [XTJ18] estimates the reachable set of a class of neural networks (multilayer perceptrons) if the activation functions are monotonic.

However, these classical formal verification techniques are often impractical for evaluating the trustworthiness of CPS, as they face significant challenges related to scalability. In practice, the complexity of CPS makes exhaustive verification computationally infeasible, and their interactions with dynamic, unpredictable environments further complicate formal reasoning. To address these limitations, several methods have been proposed to relax the formal CPS verification problem by reasoning about system models [AHFU14, QXZ+22, HHY+23] and studying their behavior through simulations [KDJ+16, DFG+19]. However, the guarantees obtained through these approaches are inherently statistical and apply only to the model rather than the actual deployed system, leaving room for discrepancies between the verified model and real-world behavior [AMF14].

On the other hand, data-driven approaches seem to be quite suitable to analyze the huge quantities of data that CPS collect through their sensors during operation. Given the attention received by the ML community in data analysis and its promising results, it would seem natural to employ ML techniques to automatically examine, model, and predict the behavior of CPS in order to evaluate their safety and reliability. However, the outcomes from ML approaches are usually black-box and highly complex models that tend to be over-sensitive to training data and lack interpretability, which is instead essential when building trust.

In this context, the research field of *Runtime Verification* [BFFR18] bridges the gap between formal verification and machine learning by developing formal methods that are lightweight and scalable to complex systems, while remaining rigorous and accessible to human understanding. More specifically, runtime verification consists of a set of methods that study how systems behave at runtime by monitoring their executions against certain properties. It includes techniques for monitoring [BDD+18], fault-localization [BFMN18, JDDS15], failure explanation [BMM+19], runtime enforcement [RRF20,FMRS18], anomaly detection [JKB14,NKJ+17a], control synthesis [CTUB13], and falsification [ALFS11, Don10].

The role of the desired (or undesired) properties is central in all runtime verification techniques because they are used to define which are the nominal executions of the system and discriminate them from the unexpected ones. Properties are expressed in terms of *specifications*, which denote sets of traces in a certain *specification language* chosen by the design engineers depending on the particular application under study. Despite their inarguable relevance, specifications describing the behavior of CPS are often unavailable.

Specification Mining [WCTZ20,BMNN22,NCJF18,FA22] is the research field that develops techniques to automatically infer properties from system executions and their interactions with the environment. Automatically learning formal requirements from data is crucial because specifications are often unknown, and manually defining them can be challenging due to the complexity of the system and its environment. Moreover, the vast amount of collected data further renders human analysis impractical. Specification mining is essential when dealing with unfamiliar systems because it allows for the precise and rigorous definition of new information about the system's behavior. This facilitates system comprehension, modeling and interpretability, all of which are critical for building trust in the system's actions. Moreover, the mined specifications can be used for the formal verification of critical components, or serve as oracles for test generation.

Nevertheless, specification mining is also highly beneficial when there is prior knowledge or partial understanding of the system under study. More specifically, inferring formal requirements from the executions of a familiar systems helps complete, refine and update system requirements, ensuring they remain aligned with evolving needs. This supports system documentation, future maintenance, and reverse engineering. Additionally, by identifying deviations from expected behaviors, specification mining enables the detection of bugs, the diagnosis of failures, and the identification of critical components that may impact system reliability.

The rich literature in specification mining highlights the relevance of this topic, but also reveals the presence of several research gaps that we will discuss and address in this thesis.

Roadmap. This thesis is structured as follows. Section 1.2 defines the specification mining problem and its main challenges, as well as the primary specification language adopted throughout this work. Section 1.3 outlines the five key research questions investigated, along with an overview of the corresponding methodology. Section 1.4 links the research questions to the scientific publications that form this thesis. Section 1.5 provides a summary of each contribution, detailing the problem description, an illustrative running example, an overview of the proposed approach, and a discussion of its evaluation. Finally, Section 1.6 presents our conclusions and discusses potential directions for future research.

1.2 Problem Statement

Formal specifications unambiguously denote a set of executions and, therefore, allow for the rigorous definition of requirements that are essential to automatize tasks such as monitoring, control synthesis, or fault-localization. The inference of specifications from data depends on the particular application and may be driven by different goals, as sketched in Figure 1.1. In particular, in the *supervised* setting, there exists a notion of *nominal* system executions, and traces are labeled accordingly. Specification mining is then applied either to formally characterize a set of *positive examples* (namely, nominal/correct/safe executions) or to distinguish positive examples from *negative examples* (i.e., unexpected/incorrect/unsafe executions). The first task arises when engineers have access only to a set of system executions that are considered desirable. In this case, they need to mine satisfied properties to comprehend or model the system, facilitate

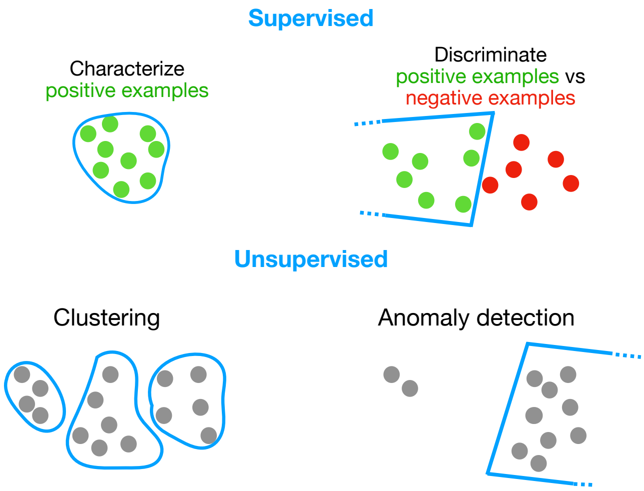


Figure 1.1: Examples of specification mining goals.The dots represent system executions, while the blue lines collect the set of executions satisfying unknown (target) specifications.

future system maintenance, or perform reverse engineering. Conversely, the second task corresponds to binary classification and must be addressed when the system has exhibited unforeseen behaviors. Here, engineers need to identify properties of correct system executions that are violated by incorrect ones in order to detect bugs, explain failures, identify critical components, or generate new tests.

In the *unsupervised* learning, there is no prior knowledge providing labels for the given dataset. As a result, some executions may be correct and others incorrect but no available information allows for their identification. In this setting, specifications need to be mined to cluster executions into groups with shared properties or to detect anomalous behaviors, assuming that anomalies are rare and qualitatively different [JKB14]. In this thesis, we will focus on tasks from the supervised setting.

We observe that examples illustrated in Figure 1.1 are idealized for the sake of simplicity: in practice, there might not exist a specification able to tightly describe a set of executions or separate exactly positive from negative examples. Due to the presence of noise and the need to avoid overfitting, even when such a specification exists, it might not be the best choice. For this reason, specification mining problems are usually formalized as search problems where the goal is to find a specification in a predetermined language that minimizes (or approximates the minimum of) an objective function. The objective function is tailored depending on the application and the task to be solved (e.g., deviation from the correct labels in the supervised setting).

The *specification language* chosen to express requirements can be either executable (such as automata or decision trees) or declarative, allowing for a more abstract definition of system properties. Since we focus on the evolution of CPS executions over time, which involve both Boolean and real-valued behaviors, we represent system executions, or *traces*, as multi-dimensional Boolean and real-valued time series. Accordingly, we will mainly adopt *Signal Temporal Logic* (STL) [MN04] and its extensions to specify our properties, as this language effectively captures temporal properties of real-valued dense-time signals in a declarative, compact, and intuitive manner.

STL extends previous existing temporal logics such as Linear Temporal Logic (LTL) [Pnu77], Metric Temporal Logic (MTL) [Koy90], and Metric Interval Temporal Logic (MITL) [AFH96] because it admits not only propositional predicates defined over Boolean variables but also numerical predicates over real-valued variables. Similarly to MITL, STL [MN04] is interpreted over dense time and its syntax allows for bounded temporal operators, such as *until* $\mathbf{U}_I$, *globally* $\mathbf{G}_I$, and *finally* $\mathbf{F}_I$, in addition to usual Boolean operators such as conjunction, disjunction, and negation. STL admits both *qualitative* semantics (expressing whether a signal satisfies a specification at a given time) and *quantitative* semantics [DM10], which measures the *robustness* of the satisfaction (or violation). Furthermore, it is important to mention that an STL specification can be abstracted to a *Parametric Signal Temporal Logic* (PSTL) [ADMN11] formula by replacing some (or all) its numerical values with parameter symbols. PSTL formulas (or, equivalently, STL *templates*) are used to express partial knowledge about a system whose overall behavior might be expected while its specific parameter values remain unavailable. Finally, HyperSTL [NKJ$^+$17b] extends STL to enable reasoning beyond individual system executions, capturing relationships between multiple executions—referred to as *hyperproperties*. This increased expressiveness is achieved by introducing variables that represent generic system executions, which are quantified using either existential or universal operators. We will provide a rigorous definition of STL, its semantics, and its parametric extension PSTL in Section 1.5.1, while we will formally define HyperSTL in Section 1.5.4.

1.3 Research Goals and Methodology

This thesis explores different problems arising in the specification mining research field and proposes solutions to the following research questions, which we investigated with the indicated methodology.

RQ1: State-of-the-Art

What is the existing literature in learning temporal properties of CPS? What are the current research gaps and their challenges?

The prolific research field of extracting STL properties from CPS executions began in 2011 with [ADMN11]. In this work, the authors aimed to complete the partially available knowledge about a set of signals provided in the form of an STL template by automatically learning missing information (i.e., numerical values) from data. Since then, many other scientific contributions have been published to address different but related problems, such as the mining of not only the numerical values but also the logical structure of the STL formula [KJMA$^+$14]. The existing methods vary in the fragment of

STL they employ, the application domain, the amount of prior knowledge available, the kind of data (labeled or unlabeled), the data processing setting (online or offline), and the availability of a model of the system. Hence, a survey summarizing the current advances in the field is needed to provide a structured overview of the vast literature, helping researchers and engineers identify existing solutions and their limitations, determine the most suitable approaches for their specific problems, and discover research gaps.

Methodology. We examined 21 scientific papers presenting innovative procedures on how to extract STL specifications from data. Given the broad spectrum of hypotheses and application goals covered by these methods, we organized them according to a few key features, and we presented a taxonomy of the state-of-the-art to help researchers and engineers understand the landscape of existing STL specification mining techniques. We then summarized each approach, providing its main ingredients and ideas in the form of text, figures, graphs, and pseudocodes. Finally, we reported the connections with other existing techniques employing different specification languages and concluded by highlighting unexplored research directions and their challenges.

RQ2: Multi-Class Classification

How to mine temporal system properties that can discriminate among different classes at the same time?

Formal specifications are often used as classifiers, separating the set of system executions into two disjoint subsets: the traces that satisfy the specifications and those that violate it. When assessing the trustworthiness of CPS, satisfying executions are usually considered expected or desired behaviors of the system, while violating executions correspond to faulty or unexpected behaviors. Whenever the discriminating specification is unknown or unavailable, specification mining techniques are applied to tackle this binary classification problem [KJMA$^+$14,BVP$^+$16,NSBB18]. However, a binary distinction between executions is not always satisfactory, as traces may need to be divided into more than two groups (e.g., when distinguishing among different kinds of trajectories or several sources of faults). In such cases, the classification problem becomes *multi-class* and each trace belongs to exactly one of several possible classes. The automatic inference of STL formulas for multi-class classification is still in its preliminary phase: existing works [MDP$^+$20b, LTLT22] focus on a restricted fragment of STL and rely on iterative binary classification. Specifically, for each class, an STL formula is learned to predict whether a trace belongs to that specific class. However, since a trace might satisfy more than one of these formulas, this approach leads to ambiguity in classification. In contrast, we look for a classification rule based on unrestricted STL specifications that unambiguously assigns each trace to a single class.

Methodology. We focused on the automatic learning of only the numerical parameters of STL templates to classify temporal signals into multiple classes. We did not restrict our search to a fragment of STL, although this is common in related works (e.g., the *monotonic* fragment [ADMN11, JDDS15]). However, we assumed that the user had some partial knowledge about the system expressed as a PSTL formula (an assumption that we mitigated by providing a semi-automatic feedback loop to refine the initial guess). Our proposed approach performs a grid search over the entire parameter space and provides a statistical guarantee of optimality for the parameter choice. We implemented our method and made it publicly available. We evaluated its performance in terms of classification accuracy, computational time, and explainability of the results on three case studies from different domains and compared to the results of three state-of-the-art methods [BVP+16, MDP+20b, NSBB18].

RQ3: Observable Safety Prediction

How to automatically learn specifications that can be used to monitor the CPS at runtime and predict a safety violation before it occurs without directly observing all relevant variables?

The formal definition of safety requirements is crucial for the employment of CPS in safety-critical applications, such as the medical and automotive domains. Indeed, safety properties must be continuously monitored during system operation so that, if safety risks are foreseen, appropriate countermeasures can be taken [BFFR18]. Given the recent advancements in Machine Learning, many CPSs integrate several components that are equipped with autonomous decision-making processes. Hence, to ensure the reliable operation of an autonomous CPS, it is essential that the system is capable of *independently* monitoring the safety requirements *on its own* in order to raise an alert or take corrective actions. However, safety requirements are usually defined in terms of system-level variables that cannot be accessed (and therefore monitored) by the CPS at runtime. Our goal is to address this issue and provide the CPS with an observable monitor that can predict its safety with the lowest rates of false negatives (missed violations) and false positives (false alarms). More specifically, given a safety specification defined over non-observable variables, we want to develop a method that automatically learns another specification that (i) is defined over variables observable directly by the CPS [TXJ+22], (ii) classifies the system executions *similarly* to the original specification, (iii) predicts violations in advance, and (iv) is explainable.

Methodology. Motivated by the relevance of the problem formulated in [TXJ+22], we aimed to address a similar challenge (i.e., learning observable and predictive monitors), while overcoming the main limitation of the approach proposed in the paper: the lack of interpretability of its outputted monitors. For this reason, we chose to learn monitors in the form of STL formulas, which are close to human understanding and, therefore, explainable (if limited by a reasonable maximum length). We framed the problem as a binary classification task and proposed a novel technique to mine STL classifiers because the existing methods [BVP+16, MDP+20b] are not suitable for handling the highly imbalanced datasets that arise from this problem. Moreover, inspired by the promising results on ensembles of monitors in [TJS+23], we proposed two new voting criteria for combining STL specifications into an ensemble. We implemented our approach, made it publicly available, and compared its performance with three state-of-the-art methods [TJS+23, BVP+16, MDP+20b]. We considered two case studies from the autonomous driving domain, which we simulated in the Webots [Web23] environment. We defined the distributions of the execution's initial conditions using Scenic [FKD+22] and then sampled from them with VerifAI [DFG+19]. We ran the resulting simulated model to generate synthetic examples, providing access to both system-level and observable variables. Different sets of synthetic examples were used to train the monitors and evaluate their performance.

RQ4: Temporal Hyperproperties

How to extend the mining of system properties to hyperproperties that relate multiple executions?

Hyperproperties [CS08] were introduced in 2008 to extend the expressiveness of properties. More specifically, while a property identifies a set of traces, a hyperproperty defines a set of sets of traces. Hence, hyperproperties can define relationships between multiple system executions instead of just describing the behavior of single executions studied individually. Hyperproperties are needed to formalize essential information-flow security policies that cannot be captured by properties only, such as the *noninterference* policy: "*Outputs observed by low-security users are the same as they would be if inputs provided by high-security users were absent*" [CFK+14], or stability conditions, such as the *robust behavior*: "*Small differences in system inputs result in small differences in system outputs*" [NKJ+17b]. Temporal logics have been extended to express hyperproperties by introducing quantification over trace variables: in particular, STL has been enhanced into Hyper Signal Temporal Logic (HyperSTL) [NKJ+17b]. However, no research has been carried out so far to increase the expressiveness of the specifications that can be automatically inferred from CPS executions, namely from STL to HyperSTL formulas.

Methodology. We defined our research problem after noticing that the literature on mining hyperproperties was still in its infancy and that the few existing works [SFA17, FHT19, RMS20] were very limited in the language they admitted. We aimed at devising a method that could automatically infer a set of temporal hyperproperties from a set of execution traces, allowing the search space to be the unconstrained HyperSTL (or any of its fragments, if required by the user). We wanted to allow for both the parameters and the structural synthesis. To do so, we relied on *Syntax-Guided Synthesis* (SyGuS) [ABJ+13] and integrated it into our mining procedure to guide the search in the space of admissible candidate specifications. The user is free to embed any available domain knowledge by specifying a grammar of admissible formulas that constrains the search space. The main bottleneck of our approach was the monitoring of hyperproperties (i.e., checking whether they are satisfied or violated by the given set of traces), whose computational cost is exponential in the number of quantifiers appearing in the formula. We mitigated it by proposing two new heuristics to efficiently monitor hyperproperties. Finally, we evaluated the performance of our mining approach by studying its precision, variability, computational time, flexibility, and scalability and by comparing it to the state-of-the-art [LB15, RMS20] (only applicable to very limited fragments of HyperSTL).

RQ5: Shape Properties

How to learn properties based on the shape of temporal signals representing system executions?

While Signal Temporal Logic is a specification language well-suited for expressing temporal relationships between variables, it does not include any notion of geometry. Hence, STL cannot characterize traces based on their shapes or patterns, such as sinusoidal waves, peaks, or exponential slopes, which might be necessary for several applications (e.g., when analyzing cardiac activity or the voltage signals of electricity suppliers). In order to rigorously describe temporal shape patterns, a novel declarative language has been proposed in [NQF+19]: *Shape Expressions* (SE). The advantage of SE over existing formalisms such as Timed Regular Expressions [EA02] is its ability to handle noisy data given by its semantics that combines regular expression semantics and statistical regression. Similarly to STL formulas, the formulation of an SE specification that characterizes a set of signals might be extremely challenging due to the huge amount of data to examine and the complexity of the signals under study. Hence, it is crucial to develop techniques that automatically extract these types of properties from data.

Methodology. Shape Expressions [NQF+19] are regular expressions whose atomic predicates represent parametrized shapes, such as lines, exponentials, or sinusoids. We restrict our focus to lines and learn specifications in the form of *Linear Shape Expressions* (LSEs) [BDG+20]. The goal is to "tightly" characterize a given set of positive examples, represented as time-series signals. The notion of tightness is defined in terms of the mean-squared error, which measures the deviation from the piecewise approximation of the signals. The mining process has been developed in [BDG+20] and consists of three main steps: (i) segmentation of the time-series data; (ii) clustering of the segments and abstraction to symbols; (iii) automata learning and translation to LSE. We developed the tool version of this method [BDG+20] and evaluated its applicability on two case studies by analyzing the computational time required by each step, its scalability with respect to the number of traces and their length, and the explainable insights learned about the shape of the signals.

1.4 Scientific Contribution

This cumulative thesis is based on the following three publications, where I contributed as the main author.

- **RV23** *Edgar A. Aguilar, Ezio Bartocci, Cristinel Mateis, Eleonora Nesterini, Dejan Ničković. 2023.* Mining Specification Parameters for Multi-class Classification. *In:*

Katsaros, P., Nenzi, L. (eds) Runtime Verification. RV 2023. Lecture Notes in Computer Science, vol 14245. Springer, Cham. *https://doi.org/10.1007/978-3-031-44267-4_5*

- **EMSOFT23** *Ezio Bartocci, Cristinel Mateis, Eleonora Nesterini, and Dejan Ničković. 2023.* Mining Hyperproperties using Temporal Logics. *ACM Trans. Embed. Comput. Syst. 22, 5s, Article 156 (October 2023), 26 pages.* *https://doi.org/10.1145/3609394*

- **ICCPS25** *Eleonora Nesterini, Ezio Bartocci, Alessio Gambi, Dejan Ničković, Sanjit A. Seshia, Hazem Torfah. 2025.* Mining Specifications for Predictive Safety Monitoring. *In: Proceedings of the ACM/IEEE 16th International Conference on Cyber-Physical Systems (with CPS-IoT Week 2025) (ICCPS '25). Article 6, 1–11.* *https://doi.org/10.1145/3716550.3722021*

For the above publications, my contributions include conducting the literature review, defining the problem, developing the main approach and its extensions/variants, formulating theorems and proving them, implementing the method, collecting data, setting up and executing the experiments, analyzing and interpreting results, writing the main draft of the paper –including the generation of tables, figures, and pseudo-codes– and revising the manuscript based on feedback from co-authors.

Additionally, I contributed as a co-author to the following papers:

- **SEFM21** *Ezio Bartocci, Jyotirmoy Deshmukh, Cristinel Mateis, Eleonora Nesterini, Dejan Ničković, Xin Qin. 2021.* Mining Shape Expressions with ShapeIt. *In: Calinescu, R., Păsăreanu, C.S. (eds) Software Engineering and Formal Methods. SEFM 2021. Lecture Notes in Computer Science, vol 13085. Springer, Cham.* *https://doi.org/10.1007/978-3-030-92124-8_7*

- **I&C22** *Ezio Bartocci, Cristinel Mateis, Eleonora Nesterini, Dejan Ničković. 2022.* Survey on mining signal temporal logic specifications, *Information and Computation, Volume 289, Part A, 2022, 104957, ISSN 0890-5401,* *https://doi.org/10.1016/j.ic.2022.104957*

In particular, for **SEFM21**, I was responsible for the evaluation section, including data collection, experiment design and execution, result interpretation, and writing of the corresponding section. For **I&C22**, I reviewed the literature, summarized the scientific works included in the survey, and generated the pseudo-codes, along with some figures and graphs.

For each research question described in Section 1.3, Table 1.1 indicates the chapter of the thesis where it is explored and the corresponding publication. Moreover, for each publication, we report the type of publication (J journal or C conference proceedings), the number of peer-reviewers who evaluated the paper, and whether I was the first author.

	Research Question	Chapter	Publication	Type	#R	1st
RQ1	State-of-the-Art	2	**I&C22**	J	2	no
RQ2	Multi-Class Classification	3	**RV23**	C	3	yes
RQ3	Observable Safety Prediction	4	**ICCPS25**	C	4	yes
RQ4	Temporal Hyperproperties	5	**EMSOFT23**	C	4	yes
RQ5	Shape Properties	6	**SEFM21**	C	3	no

Table 1.1: Summary of the publications included in the thesis.

1.5 Summary of the Thesis

In this section, we summarize the scientific contributions of the thesis. Figure 1.2 matches each research goal introduced in Section 1.3 to the section and chapter where we address it. Additionally, Figure 1.2 groups the contributions according to the specification language employed: STL for *State-of-the-Art*, *Multi-Class Classification*, and *Observable Safety Prediction*; HyperSTL, i.e. the extension of STL to hyperproperties, for *Temporal Hyperproperties*; and Linear Shape Expressions (orthogonal to STL) for *Shape Properties*.

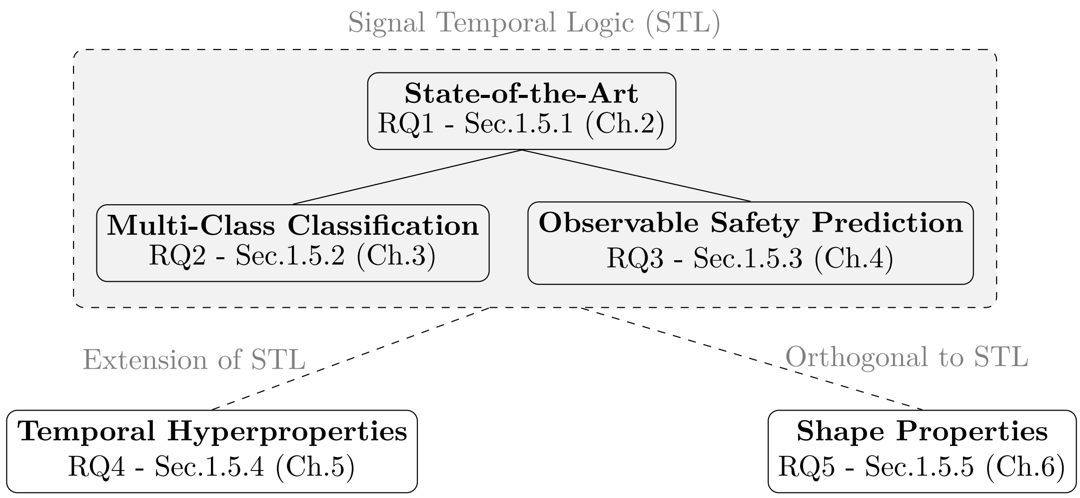


Figure 1.2: Diagram matching the research goals with the corresponding sections/chapters where they are detailed.

1.5.1 State-of-the-Art

In this section, we summarize the content of Chapter 2 [BMNN22]. We first provide the technical background for Signal Temporal Logic. Then, we detail the orthogonal axes that we used to classify the state-of-the-art in STL mining, and sketch the key ideas behind existing relevant contributions.

Background We represent system executions –or *traces*– as multi-dimensional time-series, namely finite sequences of (time,value) pairs $\boldsymbol{x} = (t_1, \boldsymbol{x}_1), \ldots, (t_n, \boldsymbol{x}_n)$, where $t_1 = 0$ and $t_i < t_{i+1}$ for all i $\in \{1, \ldots, n-1\}$. Each $\boldsymbol{x}_i$ represents the valuation of k Boolean and m real variables at time t_i and is therefore a vector in $\mathbb{B}^k \times \mathbb{R}^m$ with $k, m \in \mathbb{N}$.

The syntax of STL is defined by the following grammar [MN04]:

$$\varphi := \mathbf{true} \mid f(\boldsymbol{x}) > 0 \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \,\mathbf{U}_I\, \varphi_2 \tag{1.1}$$

where **true** is the Boolean true constant and $f(\boldsymbol{x}) > 0$ is an atomic proposition with $f\colon \mathbb{R}^n \to \mathbb{R}$. $\neg$ and $\wedge$ represent, respectively, the Boolean negation and conjunction, while $\mathbf{U}_I$ is the *until* temporal operator defined over the dense interval of time I in $\mathbb{R}_{\geq 0}$ - the interval I is generally omitted when $I = [0, \infty)$. The *finally* $\mathbf{F}_I$ and *globally* $\mathbf{G}_I$ operators are derived as follows: $\mathbf{F}_I\varphi := \mathbf{true}\,\mathbf{U}_I\varphi$ and $\mathbf{G}_I\varphi := \neg\mathbf{F}_I\neg\varphi$.

STL semantics [DM10] is defined in terms of the *robustness function* ρ that maps an STL specification φ, a trace $\boldsymbol{x}$, and time t, to the value $\rho(\varphi, \boldsymbol{x}, t)$ representing how far is $\boldsymbol{x}$ from the satisfaction boundary of φ at time t. $\rho(\varphi, \boldsymbol{x}, t)$ is defined inductively as follows:

$$\begin{aligned}
\rho(\mathbf{true}, \boldsymbol{x}, t) &= +\infty \\
\rho(f(\boldsymbol{x}) > 0, \boldsymbol{x}, t) &= f(\boldsymbol{x}(t)) \\
\rho(\neg\varphi, \boldsymbol{x}, t) &= -\rho(\varphi, \boldsymbol{x}, t) \\
\rho(\varphi_1 \wedge \varphi_2, \boldsymbol{x}, t) &= \min\left(\rho(\varphi_1, \boldsymbol{x}, t), \rho(\varphi_2, \boldsymbol{x}, t)\right) \\
\rho(\varphi_1 \,\mathbf{U}_I\, \varphi_2, \boldsymbol{x}, t) &= \sup_{t' \in t \oplus I} \Big(\min\big(\rho(\varphi_2, \boldsymbol{x}, t'), \inf_{t'' \in (t, t')} (\rho(\varphi_1, \boldsymbol{x}, t''))\big)\Big)
\end{aligned} \tag{1.2}$$

where $\oplus$ denotes the Minkowski sum. $\rho(\varphi; \boldsymbol{x})$ is used to denote $\rho(\varphi, \boldsymbol{x}, 0)$. The STL quantitative semantics is *sound*, i.e., when $\rho(\varphi, \boldsymbol{x}, t) > 0$ the signal $\boldsymbol{x}$ satisfies the specification φ, while, when $\rho(\varphi, \boldsymbol{x}, t) < 0$, the signal violates the specification.

Parametric Signal Temporal Logic (PSTL) [ADMN11] extends STL by replacing threshold constants in numerical predicates and time bounds in temporal operators with parameter symbols. The set of values that the parameter symbols in a formula may assume is called *parameter space* $\mathcal{W}$. A parameter valuation $\boldsymbol{v} \in \mathcal{W}$ *instantiates* a PSTL formula ψ into the STL formula $\psi_{\boldsymbol{v}}$ obtained by replacing its parameter symbols with $\boldsymbol{v}$. For example, given the PSTL template $\psi = (x_1 \geq p_1)\mathbf{U}_{[T_1,T_2]}(x_2 < p_2)$ where p_1, p_2 are magnitude parameter symbols, and T_1, T_2 are timing bounds symbols, the parameter valuation $\boldsymbol{v} = (11.2, 0, 7, 99)$ instantiates the STL formula $\psi_{\boldsymbol{v}} = (x_1 \geq 11.2)\,\mathbf{U}_{[0,7]}\,(x_2 < 99)$.

A relevant subclass of PSTL templates is the *monotonic* fragment [ADMN11, JDDS13]. A PSTL formula ψ is called *monotonically increasing* with respect to a parameter p if increasing the value of p makes the formula easier to satisfy. Formally, if $v \geq v'$, then the satisfaction of $\psi_{v'}$ (obtained by replacing p with v') implies the satisfaction of ψ_v (e.g. $\psi = \mathbf{G}(x_1 \leq p)$). Conversely, ψ is *monotonically decreasing* if decreasing the parameter makes the formula easier to satisfy, which means that if $v \geq v'$, the satisfaction of ψ_v implies the satisfaction of $\psi_{v'}$ (e.g. $\psi = \mathbf{G}(x_2 \geq p)$). A parameter does not necessarily influence the satisfaction of the formula in a monotonic way; for example, in $\psi = \mathbf{G}(x_1 < p \wedge x_2 \geq p)$, increasing or decreasing p does not consistently make the formula easier or harder to satisfy. Therefore, a PSTL formula is defined as *monotonic* if it is monotonic with respect to all its parameters (it may be increasing for some parameters and decreasing for others).

Taxonomy Existing specification mining techniques can be classified according to several orthogonal key features:

- *Template-based* or *template-free approaches.* In a template-based approach, the user provides a specification template as input, and the mining procedure is applied to find the numerical values that best fit the formula for the particular application under study. The template is formalized, for example, as a PSTL formula (or one of its fragments) and represents the domain knowledge available to the user, who might know what general behavior to expect from a system or might be interested in a specific property, while being unaware of the numerical values that characterize it. In general, learning specifications from (partially) known systems may be necessary for several reasons, including (i) completing or updating existing information, (ii) detecting bugs, (iii) confirming nominal behaviors, and (iv) performing system maintenance. Conversely, when no prior knowledge is available, template-free approaches are more suitable, as their mining procedures automatically infer both the formula structure and the required numerical values. These approaches are essential for learning properties of unfamiliar systems in order to perform system comprehension, modeling, and explainability.

- *Supervised* or *unsupervised learning.* As already mentioned in Section 1.2 (see Figure 1.1), system executions might be labeled or unlabeled. When available, labels indicate whether a trace represents a nominal behavior (i.e., expected or desired). When labeled data is available, supervised learning aims to extract the underlying principle that assigns labels in order to generalize it and predict labels of new, unseen data. In this context, supervised mining techniques learn specifications that either characterize a set of positive examples or distinguish between two classes: positive and negative examples. On the contrary, if data is unlabeled, there is no notion of correct/incorrect behavior. Hence, in this setting, unsupervised learning techniques are employed to either cluster data into different groups sharing similar characteristics, or to automatically detect anomalous behaviors.

- *Offline* or *online data availability.* The dataset from which specifications have to be extracted might be available entirely before the mining process starts (offline setting), or, on the contrary, its collection might be interleaved with steps of the learning procedure (online setting). In the former case, the specification is learned once from the set of traces and returned without any further changes. Conversely, in the latter case, a candidate specification is first inferred from the initial partial dataset and then is updated every time a new trace arrives. A trivial solution for the online problem would be to reapply the offline learner from scratch to the augmented dataset every time new data is available. Such an approach is clearly inefficient, and, instead, it would be preferable to implement smart modifications to the current candidate formula. It is therefore evident that mining techniques should be tailored to the kind of data availability.

- *Passive* or *active mining*. A further distinction can be made regarding data availability: if data collection is independent of the learning process, the mining is said to be *passive*, while, if the learner can interact with the system to steer the generation of data, the mining is said to be *active*. In the offline setting, the set of traces is fixed before the beginning of the mining process, so the learning is always passive. Conversely, the online setting supports both passive and active mining, depending on whether the new data available has been produced with or without inputs from the miner.

Existing works Relevant contributions to STL specification mining literature include the following works:

[ADMN11] presents an offline template-based approach to learning parameters from positive examples. The authors first present a theoretical approach to compute exactly the *validity domain* of the PSTL template, namely the region of parameter values that render the formula satisfied by the given dataset. However, since such a procedure is computationally intractable, they restrict the space of admissible templates to a fragment of PSTL –called *monotonic* [VDJS17]– whose validity domains can be efficiently approximated to an arbitrary error by combining monitoring with search. The final parameter values are selected on the boundary of the (approximated) validity domain to avoid over-generalization. A similar task is addressed by [JTS+17], where authors, instead of computing the entire validity domain, design a differentiable *tightness* metric, whose maximization returns parameter values that render the PSTL template *tightly* satisfied (i.e., an arbitrary perturbation of the parameters results in a violated formula).

Authors in [BVP+16,BB17,BB21,BB18] propose a template-free framework to passively mine specifications by leveraging classical methods on decision trees. Indeed, the key insight is the possibility to map binary decision trees to formulas in a fragment of STL, which the authors define as the conjunctions and disjunctions of unary temporal operators applied to predicates. The proposed approach consists of two steps: (i) learning a decision tree that classifies the data, and (ii) translating the tree into an STL specification. With slight changes, the same approach is used to tackle the offline anomaly detection problem and both the online and offline supervised binary classification problem.

Authors in [KJMA+14, JKB14, KJB17] leverage the partial order of specifications induced by STL semantics to generate a directed acyclic graph (DAG): the nodes are the candidate formulas, while the edges represent the order relation between them. The

DAG is constructed on the fly to guide the search of the formula template, while classical optimization methods find the numerical values of its parameters. The proposed intuition is instantiated to both tackle the passive (online and offline) binary classification problem and the offline unsupervised anomaly detection. The search is restricted to the newly proposed fragments of STL: Reactive STL and Inference STL, which both simplify the formulas by allowing temporal operators to be applied only to predicates.

Some approaches [NSBB18, SG19, AG20] employ genetic algorithms to extract an STL formula (both its structure and parameters) that classifies positive and negative examples in the offline setting. At every iteration, genetic algorithms evolve a population of STL specifications by using a crossover operator, which defines how formulas are combined to generate new ones, and a mutation operator, which introduces random changes in the formulas. Then, only a percentage of formulas "survives" and is passed to the next iteration. The survival probability depends on the fitness function that takes into account the discrimination power [NSBB18] of the formula and its length.

The only STL specification mining approach that performs *active* learning is presented by the works in [YHF12, JDDS13, JDDS15]. The goal is to learn parameter values for a given monotonic PSTL template such that any possible trace generated from the system satisfies the resulting STL formula. The procedure is based on counterexample-guided inductive synthesis and alternates between computing a candidate STL specification and attempting its falsification [NSF+10]. To achieve this, a model of the system is supposed to be available so that it can be used to generate violating traces. If a counterexample is found, a new candidate formula must be computed; otherwise, the procedure terminates.

1.5.2 Multi-Class Classification

This section describes the multi-class classification problem and summarizes the approach we proposed to address it, as detailed in Chapter 3 [ABM+23]. Additionally, we present a running example and discuss our method's evaluation.

Problem Definition. The multi-class classification task is a supervised learning problem, where each example in the dataset is labeled as belonging to exactly one class among several possibilities. The challenge of classifying data into multiple classes arises when discriminating between only two types of system executions (e.g., correct or incorrect) is not sufficient. This occurs, for example, when it is necessary to identify several types of fault in the system or to distinguish between multiple kinds of trajectories.

We suppose that the user provides as input a parametric STL (PSTL) template ψ that has to be instantiated into multiple different STL specifications so that each mined specification characterizes one class and maximally discriminates it from the other classes. We define the *optimality criteria* that our target parameter values must satisfy using the notion of *validity domain*. Given a trace $\boldsymbol{x}$ and a PSTL formula ψ, the validity domain $\mathcal{V}(\psi, \boldsymbol{x})$ is the set of parameter values that *instantiate* from ψ a specification satisfied by

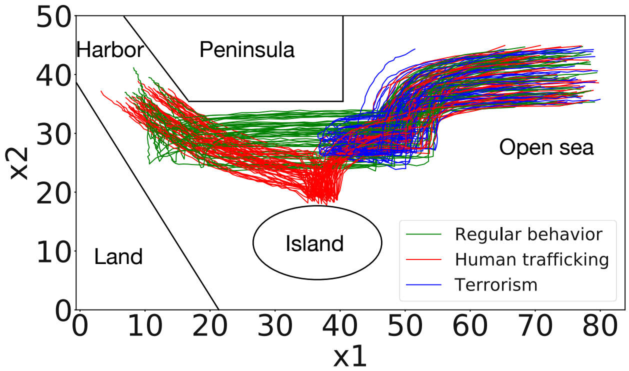


Figure 1.3: Example of traces representing ship trajectories belonging to three separate classes. This image is taken from [ABM+23] and redrawn from [KJB17].

$\boldsymbol{x}$. For a set of signals, the validity domain corresponds to the intersection of the validity domains of all the signals in the set. Hence, for each class of signals, our search aims to find that vector of parameter values $\boldsymbol{v}$ that (i) is in the class' validity domain and (ii) is at the furthest distance from the validity domains of the other classes.

The mined formulas will be helpful in characterizing each set of traces and gaining insights into what discriminates one class from the others but will not represent a proper classifier, as a new unseen trace might satisfy more than one mined specification or violate all of them. Our additional goal is, therefore, to define a procedure that combines multiple STL formulas to produce one single unambiguous multi-class classifier.

Running Example. Let us consider the example from [KJB17], depicted in Figure 1.3, where the traces represent the (x, y)-coordinates of ships approaching a harbor. Different colors represent different classes of signals: (i) green signals represent expected trajectories; (ii) red signals are anomalous trajectories that might be associated with human trafficking activities because the ships deviate to the island before reaching the harbor; (iii) blue signals are another kind of anomalous trajectories, possibly connected to terrorist activities because ships approach the harbor but turn back to the open sea before reaching it.

We consider the PSTL template from [NSBB18] as given as input to our mining procedure: $\psi = ((x > p_2)\, \mathbf{U}_{[T_1,T_2]}\, (y \leq p_1))$, where x and y are the variables for the spatial coordinates of the signals, while p_1, p_2, T_1, and T_2 are the parameters to be learned. Our goal is to find three vectors of parameter values $\boldsymbol{v}_1$, $\boldsymbol{v}_2$, $\boldsymbol{v}_3$ such that the corresponding instantiated STL formulas describe one class each while discriminating maximally from the other two classes.

The output produced by our approach is the following set of STL specifications:

$$\begin{aligned}\text{Class 1 (green): } \varphi_{\boldsymbol{v}_1} &= (x_2 > 22.44)\,\mathbf{U}_{[110,150]}\,(x_1 \le 42.0)\\ \text{Class 2 (red): } \varphi_{\boldsymbol{v}_2} &= (x_2 > 24.25)\,\mathbf{U}_{[20,75]}\,(x_1 \le 42.0)\\ \text{Class 3 (blue): } \varphi_{\boldsymbol{v}_3} &= (x_2 > 29.69)\,\mathbf{U}_{[110,300]}\,(x_1 \le 66.37).\end{aligned}$$

We use the set of mined specifications to perform the multi-class classification by assigning a (possibly unseen) trace $\boldsymbol{x}$ to the class whose specification obtains the highest robustness value ρ, i.e., to the class i such that $i := \operatorname{argmax}_{j=1,\ldots,3}\, \rho(\varphi_{\boldsymbol{v}_j}; \boldsymbol{x})$.

Our Approach. The approach we propose consists of three steps: (i) the approximation of the validity domain for each class; (ii) the selection of the *optimal* parameter values in each validity domain; and (iii) the combination of the multiple STL specifications into one classifier.

We compute an over-approximation of the validity domain by dividing the space of the parameter values in a lattice of cells and then evaluating each cell separately to decide whether it should be flagged as part of the validity domain. We propose one method –*FixedBin*– with fixed cell size, and one –*AdaBin*– with adaptive cell size. In *FixedBin*, we add a cell to the approximated validity domain if we find in it at least one parameter value that instantiates a specification satisfied by all traces in the class. Conversely, *AdaBin* leverages binary search to speed up the analysis of the parameter space by imposing the finest cell resolution only in the most uncertain areas, such as the validity domain boundaries. More specifically, it is a recursive procedure that starts with the entire parameter space as the initial cell under study and splits it only if the proportion of satisfied parameter values is low but strictly greater than zero. Then, *AdaBin* adds a cell to the approximated validity domain if it finds a high proportion of parameter values that instantiate specifications satisfied by all traces, and rejects a cell if such a proportion is equal to zero.

Once the validity domains have been approximated, we select one point in each validity domain that maximizes the point-set distance from the union set of the other validity domains. This point will be the parameter value that instantiates the PSTL template to produce the STL formula that characterizes that class of signals. For simplicity, Figure 1.4 shows an example of the approximated validity domains for only two different classes of signals. The validity domains are computed with respect to the same PSTL template with two parameters. The parameter valuation $\hat{v}_1$ is chosen within the blue validity domain to maximize the distance from the validity domain of the other class (i.e., the union of the red figures). Similarly, $\hat{v}_2$ is selected within the red validity domain to maximize the distance from the validity domain of the other class.

We prove a statistical guarantee on the optimality of the values of selected parameters for the validity domains that have been approximated by *FixedBin*. Moreover, we show that its worst-case complexity (in terms of calls to the STL monitor RTAMT [NY20]) is

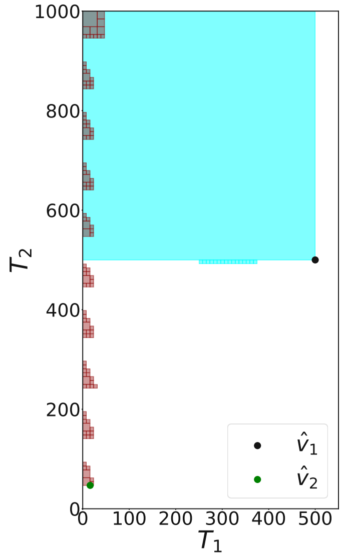


Figure 1.4: Example of the approximation of two validity domains of a PSTL template with two parameters (T_1 on the x-axis, and T_2 on the y-axis). The blue square represents the validity domain of one class of signals, while the union of the red figures is the validity domain of a different class of signals. The parameter valuation $\hat{v}_1 = (500, 500)$ is chosen to represent the first class of signals, while $\hat{v}_2 = (16, 47)$ represents the second class. This image is taken from [ABM+23] and reproduced with permission from Springer Nature.

exponential in the number of parameter symbols in the PSTL template, but linear in the number of traces in the dataset.

Finally, we define our multi-class classifier as the function that associates a trace $\boldsymbol{x}$ to the label of the class whose specification's robustness value ρ is maximum. In other words, we compute the robustness values of the trace with respect to all the mined specifications and associate $\boldsymbol{x}$ with the most compatible class, namely the one with the highest degree of satisfaction (or the lowest degree of violation in case $\boldsymbol{x}$ violates all the specifications).

We conclude by highlighting two variants that we proposed for our approach:

- An extension that enables the use of a distinct PSTL template for each class, rather than a single template common to all classes. This advancement offers greater flexibility to the user at the moment of defining the initial formula, as it is often easier to identify class-specific properties rather than to find a single property that effectively discriminates among multiple classes simultaneously;

- A speed-up in approximating the validity domains for a popular fragment of PSTL formulas, specifically the *monotonic* PSTL fragment [ADMN11]. We leverage the simplified geometry of this fragment's validity domains to significantly reduce the number of monitor calls required.

Evaluation. We implemented our approach and its variants in the publicly available tool MiniPaSTeL[1] (MINIng PArameters for Signal TEmporal Logic). We applied it to three case studies from different domains, seeking an answer to the following questions:

- **RQ2.1 [Applicability].** *Does the need for a PSTL template as input limit the applicability of our approach?*
- **RQ2.2 [Performance evaluation].** *How does our method compare to state-of-the-art?*
- **RQ2.3 [Explainability of the results].** *Can the user exploit the results to infer new insights into the traces?*

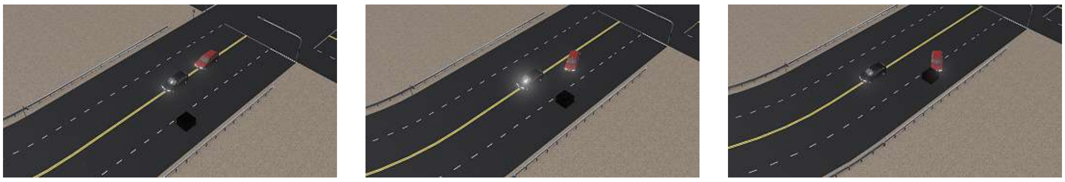
Our experiments showed that our method can be exploited to help the user refine the required PSTL template so that its definition does not limit the applicability of our approach (**RQ2.1**). More specifically, the approximation of the entire validity domain is a crucial step in our mining process that has a valuable advantage: it can be plotted to gain insights into the traces and the properties they satisfy. Hence, the user might provide a possibly inadequate guess for a discriminating template, but can then combine their partial understanding of the system and the visual analysis of the validity domain to modify or refine it.

The state-of-the-art lacked methods for mining STL formulas discriminating among several classes simultaneously (**RQ2.2**). Consequently, we chose our performance metric as the binary misclassification rates of each class against all others [MDP$^+$20b], rather than the overall misclassification rate (i.e., proportion of incorrect predictions out of the total number of predictions). Moreover, no existing methods supported mining *generic* STL formulas, so we compared performances only for the *monotonic* templates. By analyzing the experimental results, we concluded that our method is competitive in terms of accuracy but more expensive in terms of computational time. This is due to the computation of the entire validity domain, which is possibly not always necessary to achieve good classification performances, but essential for explainability reasons (**RQ2.3**). As an example of the additional interpretability achieved by our method, we highlight that the analysis of the validity domain plots allowed us to explain unexpected values of misclassification rates in a simulated autonomous driving case study. In particular, we could conclude that the simulation environment used [DRC$^+$17] rendered foggy scenarios safer than fog-free ones (possibly because it accentuated the contour of the agents in the presence of fog).

1.5.3 Observable Safety Prediction

In this section, we present another important task addressed by our target STL specifications: prediction of safety through observable variables. We summarize the content of

[1] `https://github.com/eleonoranesterini/MiniPaSTeL`



(a) (b) (c)

Figure 1.5: Three key moments in an unsafe execution of the Lead-Follower case study. These images are taken from [NBG+25].

Chapter 4 [NBG+25] by first describing the problem under study and illustrating it with an example. Then, we sketch the key points of the approach we proposed and present the takeaways from its evaluation.

Problem Definition. Runtime monitors [BFFR18] are crucial to detect potential safety risks in CPS and therefore allow for timely corrective actions. However, safety requirements are often defined in terms of *system-level variables* that cannot be accessed and monitored by the CPS at runtime, making it impossible for the system to raise a safety alarm during its execution. In contrast, *observable variables*, such as sensor data or inputs/outputs of internal controllers, can be autonomously monitored by the CPS. However, the connection between observable variables and safety properties is not straightforward, as there is no one-to-one relationship between observable and system-level variables [TXJ+22, TJS+23]. In order to address this challenge, given an STL specification φ_{sys} that defines the safe behavior of a system in terms of system-level variables, we aim at finding another STL specification φ_{obs} that is defined over observable variables and classifies traces (as safe or unsafe) in a *similar* way to φ_{sys}. The notion of similarity is data-driven as it is based on a pre-collected set of traces for which we want to minimize false negative (missed violations) and false positives (unnecessary alerts) rates. Moreover, the target formula φ_{obs} must be *predictive* to detect violations of φ_{sys} in advance. We enforce this additional task by allowing φ_{obs} to be monitored only against trimmed execution traces, hence masking a number of simulation steps equal to the user-defined *prediction horizon*. Finally, we limit the maximum length of φ_{obs} (evaluated as the number of predicates and operators) to produce an interpretable and explainable monitor and avoid the risk of overfitting the data.

Running Example. Let us illustrate this problem with an example [YTFS23] from the domain of autonomous driving. In this example, the self-driving car under study (also called *ego car*) must follow a black car by leveraging a vision-based component, which captures images from a front-facing camera, and a controller, which uses these images to estimate the position of the leading car and determine the steering angle for the next time step. The presence of dark objects on the road challenges the controller, which may mislead the car to “follow” the object until the ego car eventually crashes against it. This incorrect behavior is depicted in Figure 1.5: Figure 1.5a shows the ego car while it is following the lead, whereas, in Figure 1.5b, the ego car deviates its trajectory toward the box until it crashes in Figure 1.5c.

The system-level specification used for this case study is $\varphi_{\text{sys}} = \mathbf{G}((d > 5) \wedge (d < 15))$, where d is the ground-truth Euclidean distance between the lead and the follower. According to φ_{sys}, correct executions see the ego car always at a minimum safety distance of 5 meters, and at a maximum distance of 15 meters (to ensure that the ego car actually follows the lead). Since d is a variable whose value cannot be accessed by the ego car at runtime, d is a system-level, non-observable variable. Other examples of system-level variables include the absolute coordinates of the ego car, the color and position of the box, and the distance between the car and the box. In contrast, our goal is to find properties expressed in terms of observable variables that can predict the violation of the original system-level specification φ_{sys} in advance. In particular, we set a prediction horizon of 100 time steps so that the monitor raises an alert when the car is in a condition similar to that shown in Figure 1.5b. This allows sufficient time to take corrective actions to avoid a collision with the obstacle and the consequent violation of φ_{sys}.

An example of an observable and predictive STL formula mined with our approach is the following:

$$\varphi_{\text{obs}} = \mathbf{F}_{[321:+\infty]}\mathbf{G}(n\_black\_pix < 37.6), \tag{1.3}$$

where n_black_pix represents the number of black pixels in the image collected by the camera mounted on the car. φ_{obs} states that, apart from an initial transient phase (long at least 321 simulation steps), the value of n_black_pix must remain below 37.6. This requirement matches our intuition that, in safe executions, the ego car should not get too close to any black object (either the leading car or the box).

Our Approach. Figure 1.6 sketches the three main steps of the approach we propose: (i) dataset generation, (ii) STL learning, and (iii) ensemble prediction.

We start with a dataset (which may be given or could be generated by executing the system) of l execution traces γ, for which we assume that we have access to both the valuations σ of system-level variables and the valuations x of observable variables –meaning that $\gamma = (\sigma, x)$. We monitor the system-level trace σ against the given safety property φ_{sys} and label the corresponding observable trace x as safe or unsafe. Moreover, we cut x of its last h simulation steps –being h the user-defined prediction horizon– and add the trimmed observable trace x^{-h} to the training dataset X.

The generated training dataset X contains only trimmed executions evaluated on observable variables, which have been labeled based on the system-level variables. The problem to be solved at this point is a binary classification problem, which we aim to address by learning an STL formula φ_{obs}. We developed a novel grammar-based [BMNN23] approach for mining STL classifiers, as existing ones [BVP+16, MDP+20b] were not suitable to deal with the high imbalance of the datasets resulting from this setting. Our method employs a Simulated Annealing [KGV83] approach to guide the search in the space of admissible formulas, which is defined by the user as an STL (sub)grammar. Candidate specifications

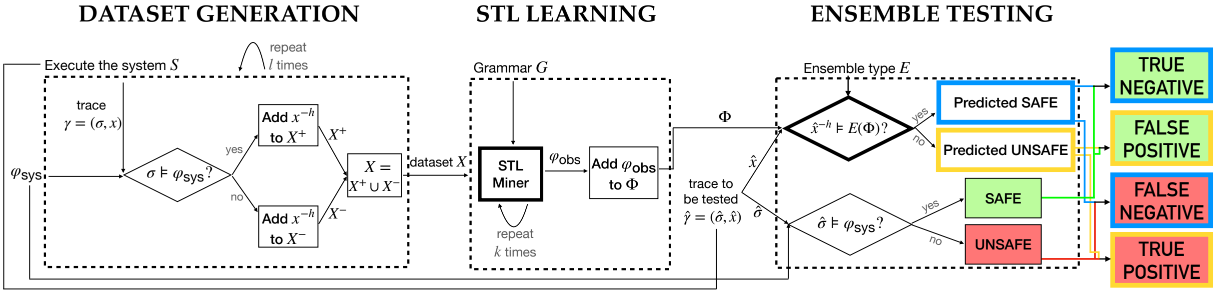


Figure 1.6: Diagram representing the three main steps to mine observable STL predictors. This image is taken from [NBG+25].

are evaluated not only based on the proportion of misclassified traces but also on the degree of each misclassification, computed in terms of the (normalized) robustness value ρ, as defined in Eq. 1.2. Moreover, we add a strong penalization for formulas that produce high ratios of false negatives or false positives in order to deal with highly imbalanced datasets and avoid that all traces are predicted in the most numerous class.

Finally, to improve the classification performance of the mined monitor, we introduce two new ensemble techniques (*Total Robustness Voting* and *Largest Robustness Voting*) based on the STL robustness function ρ. Hence, we repeat the STL learning k times to produce a set Φ of k STL formulas. When evaluating the safety of a (possibly unseen) trimmed trace $\hat{x}^{-h}$, all STL formulas in Φ express a quantitative vote that the ensemble technique E merges into a single outcome in order to predict whether the execution is safe or unsafe.

Evaluation. We implemented our approach in the publicly available tool *ObSTLearn*[2] (OBservable STL LEARN) and ran experiments to answer the following questions:

- **RQ3.1 [Predictive Performance]** *Can our method learn accurate, efficient, and explainable safety predictor monitors?*
- **RQ3.2 [Voting Criterion]** *Is robustness a suitable voting criterion?*
- **RQ3.3 [Sensitivity Analysis]** *How sensitive is our method to dataset characteristics?*

We evaluated the performance of our method (**RQ3.1**) on two case studies in the autonomous driving domain, comparing it to two existing works in the STL specification mining literature [BVP+16, MDP+20b] and an approach that addresses the observable safety prediction problem using ensembles of decision trees [TJS+23]. Although the

[2] https://github.com/eleonoranesterini/ObSTLearn.git

accuracy results are comparable across methods, the high imbalance between safe and unsafe executions in the dataset required additional evaluation metrics to assess the discriminatory power of the different methods. In particular, precision, recall, and F1-score values revealed that state-of-the-art methods perform poorly, often predicting all traces as belonging to the majority class. In contrast, our approach is able to effectively distinguish between the two types of traces. However, this superior classification performance comes at the cost of computational efficiency, as our method is significantly slower than the others. We manually evaluated the explainability of the learned monitors demonstrating their interpretability through examples such as the mined formula in Eq.(1.3).

We then considered a fixed set of mined STL specifications and employed the same metrics to assess the quality of different ensemble techniques (**RQ3.2**). The experimental results show that the novel ensemble methods, which leverage the robust semantics of STL, significantly improve the prediction accuracy compared to the classical ensemble and averaging methods.

Lastly, we evaluated the sensitivity of our approach to key features of the training dataset, such as the imbalance between safe and unsafe traces, the size of the dataset, and the length of the prediction horizon. The results indicate an overall satisfactory robustness with respect to these variations (**RQ3.3**).

1.5.4 Temporal Hyperproperties

In this section, we present an overview of the problem addressed in Chapter 5 [BMNN23]. After defining the problem and providing an example, we outline the key ideas behind our approach, highlight its main contributions, and conclude with a discussion of its evaluation.

Problem Definition. Some critical system characteristics, such as information flow security policies or system robustness to input perturbations, cannot be captured by properties defined over single system executions. To address this limitation, the authors in [CS08] introduced hyperproperties, which allow reasoning about multiple system executions at the same time by adding quantification over system traces. Consequently, several temporal logics have been extended to deal with this additional level of expressiveness, such as LTL [Pnu77] to HyperLTL [CFK+14] and STL [MN04] to HyperSTL [NKJ+17b]. The syntax of HyperSTL is defined by the following grammar:

$$\psi = \exists \pi . \psi \mid \forall \pi . \psi \mid \varphi \tag{1.4}$$

$$\varphi = \pi[\boldsymbol{b}] \mid f(\pi[\boldsymbol{x}]) > 0 \mid \neg \varphi \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \mathbf{U}_I \varphi_2 \tag{1.5}$$

where π is a trace variable with Boolean variables $\boldsymbol{b}$ and real-valued variables $\boldsymbol{x}$. We note that, compared to the definition of STL in Eq. (1.1), HyperSTL adds the quantification –existential $\exists$ and universal $\forall$– over trace variables π in Eq. (1.4). Meanwhile, Eq. (1.5) mirrors the definition of STL but explicitly incorporates trace variables π into the notation.

As only a few works [SFA17, FHT19, RMS20] address the problem of learning hyperproperties in a restricted fragment of HyperLTL, our objective is to develop the first approach that mines hyperproperties in HyperSTL, which has even greater expressiveness than HyperLTL. We assume a passive setting in which a fixed set of execution traces generated from the system is given, and we aim to learn HyperSTL formulas that are satisfied by this set of positive examples. We allow the user to constrain the HyperSTL grammar to restrict the set of admissible candidate specifications. By doing so, the same method can be applied to learn Hyper-LTL and hyper-propositions as special cases, as well as LTL and STL.

Running Example. Let us consider a case study from the autonomous driving domain, where a car is driving straight in a parking lot and, suddenly, a pedestrian crosses its way. The car is equipped with a camera and uses a state-of-the-art image detector [WBL22] to recognize the presence of pedestrians. Hence, as soon as the crossing pedestrian is detected, the car activates the autonomous emergency braking and stops. The amount of time needed by the car to get to a complete stop varies depending on the car's initial velocity. Hence, the distance between the car and the pedestrian at the moment of halt is also affected by the car's velocity.

HyperSTL formulas learned from a set of traces describing these executions of the autonomous car can be used to study stability conditions, such as the *robust behavior*, which relates how perturbations of input variables (i.e., the velocity in this case study) affect perturbations of output variables (i.e., the final distance between the car and the pedestrian). An example of this system characteristics mined by our approach in the form of a HyperSTL formula is the following:

$$\psi = \forall\pi\forall\pi'.\, \mathbf{G}(|\pi[v] - \pi'[v]| < 46.8 \rightarrow |\pi[d] - \pi'[d]| < 21.9).$$

In ψ, π and π' are execution variables, while v and d refer, respectively, to velocity and distance variables. Hence, ψ states that for every pair of executions, if their difference in the car's velocity is at most 46.8 mps, then their corresponding car-pedestrian distances must not differ by more than 21.9 meters.

Our approach is the only existing one that allows for the alternation of quantifiers in temporal hyperproperties, as in the case of the following mined HyperSTL formula:

$$\theta = \exists\pi\forall\pi'.\, \mathbf{G}(|\pi[d] - \pi'[d]| < 12.5).$$

θ affirms the existence of a system execution whose difference in the car-pedestrian distance with all the rest of the executions is always below 12.5 meters.

Finally, we note that STL formulas correspond to the HyperSTL fragment with exactly one universal quantifier. An example of an STL formula learned for this case study is the following invariant: $\varphi = \forall\pi.\, \mathbf{G}(\pi[v] < 12.75)$, stating that the car's velocity always remains below 12.75 meters per second.

Our Approach. The method we devised takes inspiration from the syntax-guided synthesis (SyGuS) [ABJ+13], where a user-defined grammar determines the set of admissible candidate solutions (in the form of programs in the case of SyGuS, and specifications in our context). The grammar definition provides great flexibility to the user, who can decide the amount of domain knowledge to embed in the mining process. In practice, this novel grammar-based specification mining method bridges the gap between *template-based* and *template-free* approaches, as the grammar can range from being as general as the entire (Hyper)STL to as restrictive as a specific template, or anywhere in between.

The main procedure consists of two distinct steps: (i) mining the formula structure (namely, the combination of quantifiers, temporal and Boolean operators, and symbolic predicates); (ii) inferring parameter values. We start with a randomly sampled formula template and then iteratively apply mutations such that the newly generated formulas are still accepted by the user-defined grammar. Each candidate hyperproperty is assigned a score that quantifies its quality with respect to the set of traces. If the score is greater than 1, the candidate formula is satisfied by the dataset, and the search terminates. Otherwise, the score affects the probability of accepting mutated formulas as new candidates. The parameter refinement step is applicable only to monotonic HyperSTL specifications, which are automatically identified using the polarity rules from [ADMN11]. This parameter learning step extends the combination of monitoring and binary search proposed in [ADMN11] from STL properties to HyperSTL hyperproperties.

A computational bottleneck in our approach is the monitoring of hyperproperties, which is used in the evaluation of the formula score and requires an exponential number of calls to the STL monitor with respect to the number of quantifiers in the formula. To mitigate this issue and improve efficiency, we introduce two heuristics for hyperproperty monitoring. More specifically, the heuristic *quantifier's properties for efficient monitoring* leverages the semantics of quantifiers to avoid unnecessary monitoring calls by introducing an early stop whenever a satisfied witness for the existential quantifier or a violation for the universal quantifier is found. The other heuristic –*robustness for efficient monitoring*– is based on the correctness property [DFM13] of the STL robustness function [DM10] and avoids monitoring tuples of traces that are "close" to an already evaluated tuple.

Evaluation. We implemented our approach in the publicly available HyTeM tool[3] (HYper-TEmporal Miner) and ran several sets of experiments to answer the following questions:

- **RQ4.1 [Performance evaluation]** *To what extent are the mined specifications a good characterization of the system properties?*
- **RQ4.2 [Ability to mine useful specifications]** *How useful are the specifications that we infer from the traces?*

[3] `https://github.com/eleonoranesterini/HyTeM`

- **RQ4.3 [Flexibility of the mining procedure]** *Can we effectively use the same method to mine not only HyperSTL requirements but also its fragments?*
- **RQ4.4 [Comparison with the state-of-the-art]** *How does our approach compare to other existing specification mining methods?*
- **RQ4.5 [Scalability]** *How does our method scale with different grammar restrictions? Do the two monitoring heuristics effectively speed up the overall mining procedure?*

We evaluated the quality of the mined hyperproperties both quantitatively (**RQ4.1**) and qualitatively (**RQ4.2**). Quantitative analysis is based on several metrics, including the proportion of mined formulas violated by unseen sets of system executions (*ratio of false negatives*), the proportion satisfied by randomly generated traces (*ratio of false positives*), and the variability of the mined formulas on seven case studies. The qualitative evaluation is instead a manual inspection of the mined formulas, which we classified as interesting, trivial, or incorrect (i.e., valid for the training dataset but not for the system in general). Our results highlight the significant impact of even minimal domain knowledge on both types of performance. With a completely unrestricted grammar, the method tended to learn overgeneralizing, trivial, and extremely variable formulas. However, by slightly constraining the grammar (e.g., specifying the number and types of quantifiers or selecting an initial temporal operator), we obtained much tighter and more useful hyperproperties.

The possibility of customizing the input grammar is crucial also to ensure the broad applicability of this mining process (**RQ4.3**). Indeed, the resulting flexibility makes the approach well-suited not only for HyperSTL requirements but also for its subsets, such as HyperLTL, STL, LTL, and even non-temporal (hyper)properties. Since no previous work had addressed the problem of mining HyperSTL specifications, the input grammar definition was essential even for enabling a comparison with the state-of-art on the available limited fragments of HyperSTL, specifically LTL for the tool *Texada* [LPB15], and alternation-free HyperLTL for *HyperMiner* [RMS20] (**RQ4.4**). The experimental results demonstrate that our method can not only learn the same properties mined by other tools with fixed user-defined templates, but also infer new interesting information when no template is imposed. However, as expected, our approach is slower due to its broader applicability, as it does not benefit from the optimizations that other tools leverage for specific templates within a fixed specification language.

Finally, the scalability evaluation (**RQ4.5**) confirms the expected monotonic increase in computational time as the maximum formula length increases. Moreover, we observe that the grammar-constrained setting required significantly more time than the free-grammar setting, possibly because it did not allow for existential quantifiers, which are more likely to produce satisfied hyperproperties compared to universal quantifiers. Finally, although the worst-case theoretical complexity of the two monitoring heuristics remains exponential in the number of quantifiers, the experimental results demonstrate that, in practice, these heuristics effectively reduce the number of STL monitors, yielding a significant speed-up of the overall process.

1.5.5 Shape Properties

In this section, we present a summary of Chapter 6, which describes the tool paper ShapeIt [BDM+21]. The specification language adopted in this work focuses on shape patterns, taking an orthogonal direction to the STL language adopted so far which, in contrast, is used to express temporal relationships between real-valued variables.

Problem Definition. *Shape Expressions* (SEs) is a declarative formalism introduced in [NQF+19] to characterize system executions according to their shapes and repeated geometrical patterns. In particular, *Linear Shape Expressions* (LSEs) [BDG+20] allows the expression of linear behaviors of real-valued signals in the presence of noise in a rigorous and human-understandable way. Formally, LSEs are regular expressions given by the following grammar [NQF+19, BDG+20]:

$$\begin{aligned} \mathsf{shape} &:= \mathsf{line}(a, b, d) \mid \mathsf{shape}_1 + \mathsf{shape}_2 \mid \mathsf{shape}_1 \,\boldsymbol{.}\, \mathsf{shape}_2 \mid (\mathsf{shape})^{\boldsymbol{*}} \\ \mathsf{cst} &:= x \;\mathsf{in}\; [\mathbf{c1},\ \mathbf{c2}] \mid \mathsf{cst}_1 \;\mathsf{and}\; \mathsf{cst}_2 \\ \mathsf{SE} &:= \mathsf{shape} : \mathsf{cst} \end{aligned} \tag{1.6}$$

Atomic predicates $\mathsf{line}(a, b, d)$ represent lines with three scalar parameters: slope a, relative offset b, and duration d. Parameterized lines are then combined using classical operators: union ($+$), concatenation ($.$), and Kleene star ($*$). The parameter symbols appearing in the constructed regular expression shape are subject to constraints cst, where $\mathbf{c1}$ and $\mathbf{c2}$ are rational constants ($\mathbf{c1} \leq \mathbf{c2}$). The semantics of LSE [BDG+20] is defined in terms of noisy matches: a signal is considered a ν-noisy match for a given LSE formula φ if the signal can be approximated by a sequence of lines such that (i) the sequence is qualitatively consistent with the regular expression in φ, (ii) its parameter values satisfy the constraints in φ, and (iii) the mean squared error (MSE) between each line in the sequence and the corresponding part of the signal it approximates remains below ν.

Given a set of system executions and a user-defined maximum error ν, our goal is to automatically learn an LSE that is a ν-noisy match with all the given traces. Hence, we aim to develop a template-free approach that passively mines a specification in the form of Linear Shape Expression to characterize a given set of positive examples.

Running Example. We illustrate our problem with an application to the *Wine dataset* from [CKH+15], where traces represent the food spectroscopy of wines. Figure 1.7 (left) shows in blue the plot of one example from the dataset. Our goal is to extract properties that characterize with a formal specification the shape of all traces in the dataset in terms of mean linear behaviors in the presence of noise. By running our tool ShapeIt,

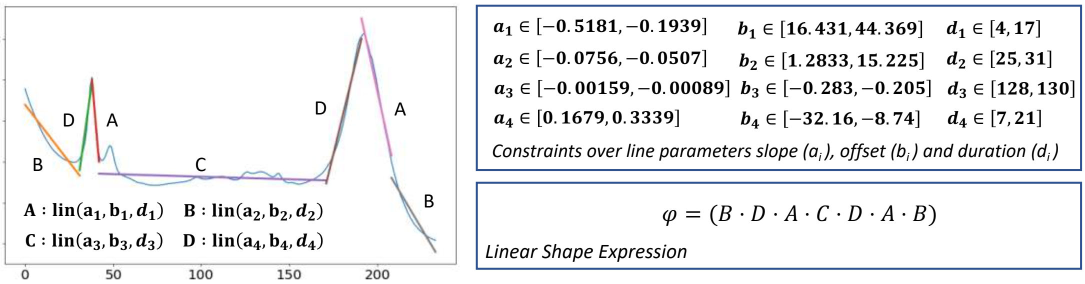


Figure 1.7: Application of ShapeIt to the *Wine dataset*. Image taken from [BDM+21] and reproduced with permission from Springer Nature.

we mine the following LSE:

$$\begin{aligned}\varphi = \; & \mathsf{line}(a_2, b_2, d_2).\mathsf{line}(a_4, b_4, d_4).\mathsf{line}(a_1, b_1, d_1). \\ & .\mathsf{line}(a_3, b_3, d_3).\mathsf{line}(a_4, b_4, d_4).\mathsf{line}(a_1, b_1, d_1).\mathsf{line}(a_2, b_2, d_2)\end{aligned}$$

whose parameters a_i, b_i, d_i are constrained by the interval values reported in Figure 1.7 (upper right). Thus, φ combines four linear atomic predicates to make a concatenation of seven lines.

By adopting the notation from Figure 1.7, we observe a repeated pattern formed by the concatenation of atomic shapes D and A, corresponding to $\mathsf{line}(a_4, b_4, d_4)$ and $\mathsf{line}(a_1, b_1, d_1)$, respectively. This pattern represents the peak that appears twice in the trace plot. The decreasing segment, represented by the linear atomic shape B, is also repeated twice - at the very beginning and the end of the signal. Conversely, the long, approximately horizontal line represented by C appears only once and is located between the two peaks. We highlight that the mined LSE is satisfied not only by the trace shown in the plot but by all the given traces, demonstrating how φ effectively summarizes the mean behavior of a potentially large set of traces.

Finally, we note that with the selected maximum error threshold ($\nu = 0.05$), the third-highest peak in the plot –the one immediately to the right of the first peak– is not detected. If a more detailed description of the traces were required, the tool could be rerun with a lower ν value. However, this would yield a more complex mined LSE, showing the trade-off between accuracy and explainability.

Our Approach. The specification mining procedure we implemented in our tool ShapeIt was proposed in [BDG+20] and is summarized in Figure 1.8. The first step (*segmentation*) consists of a piece-wise linear approximation of all the traces. For a fixed maximum threshold ν and each trace in the dataset, a dynamic programming-based method [Oza16] finds the shortest sequence of segments such that the mean squared error of each segment remains below ν. The segments are then clustered based on the values of the three parameters that uniquely define them: slope, relative offset, and

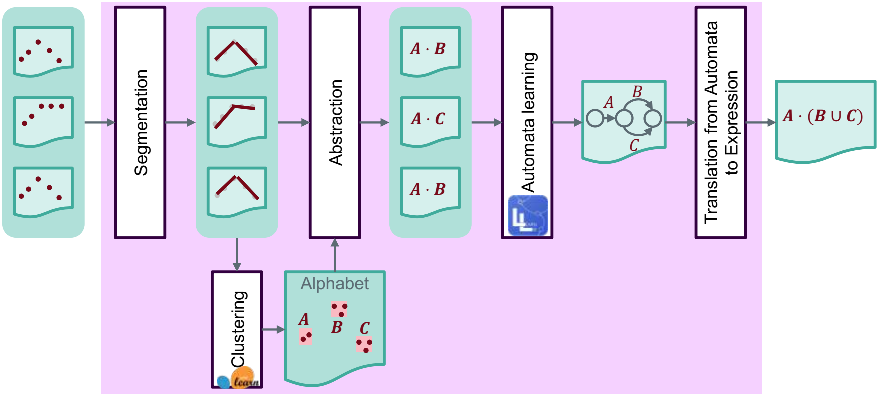


Figure 1.8: Overview of the approach implemented in ShapeIt. Image taken from [BDM+21] and reproduced with permission from Springer Nature.

duration. The algorithm adopted is the k-means *clustering* method [Mac67] with the optimal number of clusters automatically determined depending on the derivative of the within-cluster sum of squares (WCSS), which measures the compactness of clusters by summing the distances of data points from their centroids. In the following step, each cluster is over-approximated by the tightest three-dimensional box that includes all of its data points and is *abstracted* into a letter. Thus, each letter represents a set of constraints on the parameter values of slope, relative offset, and duration. This symbolic abstraction is extended to the traces: each segment is mapped to the letter corresponding to its cluster so that the segmented trace is mapped to the word obtained as the sequence of the cluster letters. The original set of traces is therefore transformed into a set of finite words: an off-the-shelf *automata learning* algorithm –reduced positive and negative inference (RPNI) [Dae10]– infers a deterministic finite automaton (DFA) that accepts all the given words, representing the positive examples in our dataset. Finally, the DFA is *translated* into a regular expression using the state elimination method.

Evaluation. We evaluated the ShapeIt[4] tool (which is publicly available online) on two applications from different domains. We analyzed its scalability in terms of the number and length of traces, as well as the impact of the maximum error threshold ν on the number of clusters and the length of the final mined LSE. Our results indicate that, regardless of the number and length of traces, the segmentation phase is consistently the most computationally expensive step in the mining procedure, whereas clustering and automata learning have negligible costs in comparison. Moreover, as expected, increasing the value of ν results in shorter mined specifications; in contrast, the number of clusters does not vary monotonically with ν. Finally, a manual evaluation of the mined LSE highlights the interpretability of the outputted formula and how it can be leveraged to gain interesting insights about relevant shape patterns in the entire dataset.

[4] https://www.doi.org/10.5281/zenodo.5569447

1.6 Conclusion and Future Work

In this thesis, we investigated and discussed several approaches for mining formal specifications from the executions of cyber-physical systems, addressing some of the research gaps in the literature. In particular, we focused on learning properties expressed in Signal Temporal Logic (STL), as it is a popular specification language suitable to describe the evolution over time of real-valued behaviors in a compact and human-understandable way. Additionally, we explored its extension to hyperproperties (HyperSTL) to infer temporal relationships between multiple system executions, as well as Linear Shape Expressions, a novel formalism that captures linear shape patterns in time-series data.

Our work began with an investigation of the state-of-the-art in STL specification mining. In particular, in Chapter 2, we provided a unified framework for categorizing existing methods based on several key aspects: template-based vs. template-free approaches, supervised vs. unsupervised learning, offline vs. online data availability, and passive vs. active mining. In addition, we summarized the literature by outlining the main ideas proposed in each work, helping researchers navigate this rich field, and facilitating the identification of unexplored research directions.

Since we noticed the lack of methods for mining STL formulas able to unambiguously discriminate among multiple types of signals, we introduced an approach to learn STL specifications for multi-class classification. As detailed in Chapter 3, we developed a passive, template-based method that approximates the optimal parameter values for each class of system executions without imposing restrictions on the expressiveness of the given formula template. We then combined multiple specifications into a single classifier by leveraging the robust quantitative semantics of STL.

In Chapter 4, we tackled the observable safety prediction problem. We developed a novel approach to learning STL formulas that can estimate in advance whether a system execution is going to become unsafe, relying solely on variables observable by the system itself. This feature enables autonomous systems to continuously assess their safety at runtime, and, if necessary, take appropriate countermeasures. Furthermore, the declarative and explainable nature of STL makes the mined monitors valuable beyond runtime prediction: engineers can leverage them to gain new insights into the system's behavior.

Then, in Chapter 5, we broadened the expressiveness of the mined specifications beyond single executions. While STL defines properties over individual traces, HyperSTL formalizes temporal hyperproperties, which enable reasoning about the relationships between multiple executions over time. We introduced the first learning approach for HyperSTL formulas. The method we developed is passive and characterizes a set of positive examples by taking as input a HyperSTL (sub)grammar that defines the set of admissible formulas. This allows users to embed any amount of prior knowledge into the mining process, making it flexible and adaptable to different application scenarios.

Finally, in Chapter 6, we focused on capturing the geometric patterns of time-series data. Since STL is not well-suited for describing such patterns, we leveraged Linear Shape Expressions (LSE), a formalism designed to represent system executions based on their (possibly noisy) linear trends. We developed a tool to automatically infer LSE from a given set of traces by integrating piecewise linear approximation, clustering, and automata learning. The mined LSE provides a formal characterization of the input dataset by describing linear and possibly repeated patterns within it.

In future work, we plan to address some of the limitations of the proposed approaches. Specifically, all the methods presented in this thesis rely on passive learning, meaning they infer formal specifications from a given, fixed set of system executions. As a result, they do not provide formal guarantees about the system that generated the data. To overcome this limitation, we aim to extend our approaches to active learning, where candidate specifications guide the generation of new traces from the system or a model of the system. Hence, the mining process will interleave the collection of unseen system executions with the refinement of previously mined formulas. By incorporating this feedback loop, we expect to provide statistical guarantees on the system's behavior.

Additionally, we plan to explore the use of syntactic and consistency rules to infer the satisfaction or violation of unseen formulas based on previously monitored specifications. Since monitoring is the most computationally expensive step in our methods –particularly for hyperproperties–, we expect that skipping unnecessary calls to the monitor can significantly accelerate the mining process, making it more computationally efficient.

We also aim to leverage the explainable nature of formal specifications to gain insights into highly complex, black-box machine learning models, such as recurrent neural networks and deep neural networks. By doing so, we seek to combine the promising results from the machine learning community in efficiently analyzing and summarizing vast amounts of collected data with the interpretability and rigor of formal specifications.

Finally, we foresee exciting new applications of specification mining through its integration with Large Language Models (LLMs)–advanced AI models trained on vast amounts of text data to understand and generate human-like language. In particular, LLMs could serve as intermediaries between the specification miner and non-technical users, such as policymakers, legal experts, and business analysts. This would enable a two-way interaction: not only could mined formal specifications be translated into natural language explanations that non-experts can understand, but users could also ask questions about specific properties, which would then be converted into template-based searches for the specification miner. Since natural language might be ambiguous, multiple possible formula templates may be formulated when translating a given property. In such cases, the system could request additional inputs and clarifications from users to refine the query. This combination of LLMs and specification mining would facilitate the generation of human-readable explanations of complex system behaviors for user manuals and developer guides, help legal and regulatory teams better understand system properties, and allow non-technical users to query specifications using natural language. Hence, this approach would make reasoning about formal specifications accessible to a much broader audience.

CHAPTER 2

Survey on Mining Signal Temporal Logic Specifications

Reference. The content of this chapter has been published in [BMNN22]: *Ezio Bartocci, Cristinel Mateis, Eleonora Nesterini, Dejan Nickovic* Survey on mining signal temporal logic specifications. *Information and Computation, Volume 289, Part A, 2022, 104957, ISSN 0890-5401,* `https://doi.org/10.1016/j.ic.2022.104957`

Abstract. Formal specifications play an essential role in the life cycle of modern systems, both at the time of their design and during their operation. Despite their importance, formal specifications are only partially (if at all) available. Specification mining is the process of learning likely system properties from the observation of its behavior and its interaction with the environment. Signal temporal logic (STL) is a popular formalism for expressing properties of cyber-physical systems (CPS). The introduction of the first methods for mining STL specifications from time series generated by CPS a decade ago led to a new vivid area of research. This survey paper overviews methods for mining STL specifications from CPS behaviors, sketches different approaches found in the literature and presents them in an intuitive and didactic manner. It aims at presenting the most influential techniques and covers the most important aspects of specification mining: template-based vs. template-free, model-based vs. model-free, passive vs. active, and supervised vs. unsupervised learning.

2.1 Introduction

Formal specifications play an important role in any system's life-cycle, including its design, implementation and operation. During the concept design phase, formal specifications can improve the process of engineering requirements, by making them more precise and rigorous. During the system implementation, formal specifications can be used for formal verification of critical components, but also as oracles in the testing activities. Finally, formal specifications can be monitored during system operation to detect violations of requirements and possibly take corrective actions.

Yet, it is often the case that formal specifications are not at all or only partially available. *Specification mining* [ACH+10, HDF18, BBS14a, JDDS15, PMM17, JTS+19] is the process of inferring likely system properties from observing its execution and the behavior of its environment. There are benefits of mining specifications both from familiar, well-understood and from unfamiliar systems. Specifications mined from a familiar system can be used (1) to complete the existing incomplete or outdated specifications, (2) for system maintenance, (3) to confirm expected behaviors, (4) to detect bugs, and (5) to generate new tests. Mined properties from unfamiliar systems are typically used for (1) system comprehension, (2) system modeling, (3) reverse engineering, (4) explaining the essential properties of black-box components (e.g., machine learning components).

In this paper, we present the state-of-the-art on mining Signal Temporal Logic (STL) [MN04] specifications from cyber-physical systems (CPS) behaviors. More specifically, we consider behaviors to be time series representing the continuous real-valued dynamics of the CPS. STL is a popular declarative language for expressing CPS properties by defining temporal and timed relations in time series. In the context of CPS, specification mining has been used for many applications, including to automate difficult tasks such as fault-localization [BFMN18, JDDS15], failure explanation [BMM+19], anomaly detection [BVP+16, NKJ+17a], verification [BDD+18] and control synthesis [CTUB13].

This paper is presented in the form of a survey, collecting the relevant methods and tools, providing the reader with the necessary references, but also sketching the main ideas behind the reviewed techniques.

An Overview of the Specification Mining Methods

We present 15 procedures for mining STL specifications from 21 scientific papers. The methods presented in this paper are influenced by the specificity of CPS applications, the time series model of the behavior and the choice of STL as the specification language. An overview of all studied methods is given in Table 2.1. There are multiple orthogonal dimensions of mining STL specifications that we briefly discuss in the remainder of this section. In some cases, when we consider that two or more methods are sufficiently similar, we select and present only one in more detail.

Template-based vs. template-free mining This is the main axe that we use in this paper to classify the specification learning methods. In template-based mining, the user provides the specification skeleton and the mining procedure consists in inferring specification parameters that are consistent with observed data. Template-free mining consists in learning both the structure of the specification as well as its parameters.

Passive vs. active mining Passive mining consists in learning a specification from a fixed set of behaviors. Active mining uses the system (or the system model), assumed to be available, to generate new behaviors and actively steer the inference process.

Supervised vs. unsupervised mining In supervised specification mining, system behaviors are labeled and the aim is to learn specifications that minimize the deviation from the labels. We further distinguish between two main scenarios for supervised mining - using only positive and using both positive and negative examples. Unsupervised learning aims at separating behaviors into multiple clusters according to their similarity, and learning a specification per cluster, where the inferred formula minimizes the misclassification rate.

Offline vs. online mining Offline mining consists in learning the specification once from the set of available behaviors. The specification remains fixed once inferred. In online mining, it is assumed that new behaviors arrive over time and the mined specification is updated and refined to take into account the effect of new behaviors.

2.2 Background

In this section, we provide some technical background needed to sketch and describe the specification mining methods. We define signals and systems, introduce *signal temporal logic* (STL), its parametric version PSTL, and its quantitative semantics interpretation.

2.2.1 Signals and Systems

Let $S = \{s_1, \ldots, s_n\}$ be a set of signal variables. We define the time domain $\mathbb{T}$ to be of the form $[0, d] \subset \mathbb{R}$. An n-dimensional *signal* w is a function $\mathbb{T} \to \mathbb{R}^n$, which we also consider to be a vector of real-valued signals $w_i : \mathbb{T} \to \mathbb{R}$ associated to variables s_i for $i = 1, \ldots, n$. Given two signals $w_1 : \mathbb{T} \to \mathbb{R}^l$ and $w_2 : \mathbb{T} \to \mathbb{R}^m$, we define their parallel composition $w_1 \| w_2 : \mathbb{T} \to \mathbb{R}^{l+m}$ in the expected way. Given a signal $w : \mathbb{T} \to \mathbb{R}^n$ and variables $R \subseteq S$ with $R = \{s_{i_1}, \ldots, s_{i_m}\}$ for some $1 \leq i_1 < \ldots < i_m \leq n$, we define the projection of w onto R as $w_R = w_{i_1} \| \ldots \| w_{i_m}$.

A *system* M is an input-output state machine that generates dynamic behaviors – it takes as input a signal u and outputs a signal $w = M(u)$.

2.2.2 Signal Temporal Logic

Signal Temporal Logic (STL) [MN04] is an extension of Linear Temporal Logic (LTL) [Pnu77]. STL expands LTL in three directions: (1) it uses time-bounded temporal operators and is interpreted over dense-time, like Metric Temporal Logic (MTL) [Koy90] and Metric Interval Temporal Logic (MITL) [AFH96], (2) it uses numeric predicates that allow to reason about real-valued variables in addition to propositional variables, and (3) it

Table 2.1: Mining STL specifications - overview of methods.

Method	Tool	Ref.	Year	STL fragment	Param only	Whole formula	Pos ex only	Also neg ex	Offline	Online	Model	Section
exact val. domain	-	[ADMN11]	2011		✓		✓		✓			2.3.1
approx. val. domain	Breach	[ADMN11]	2011	monotonic PSTL	✓		✓		✓			2.3.2
tightness metrics	TeLEx	[JTS+17]	2017		✓		✓		✓			2.3.3
computation graph	stlcg	[LAP20]	2020	no time parameters	✓		✓		✓			2.3.4
falsification	S-TaLiRo Breach	[YHF12] [JDDS13, JDDS15]	2012 2013	monotonic PSTL	✓		✓			✓	✓	2.3.5
logical clustering	-	[VDJS17] [VGD+18]	2017 2018	monotonic PSTL	✓		✓		✓			2.3.6
DAG + supervised learning	-	[KJMA+14, KJB17]	2014	rPSTL		✓	✓	✓	✓	✓		2.4.1
DAG + unsupervised learning	-	[JKB14]	2014	iPSTL		✓		✓	✓			2.4.2
grid-based clustering	-	[VIB+17]	2017			✓	✓		✓			2.4.3
supervised decision trees	-	[BVP+16, BB21, BB18]	2016	1^{st}, 2^{nd}-level primitive candidates set		✓		✓	✓	✓		2.4.4, 2.4.4
unsupervised decision trees	-	[BB17]	2017	1^{st}, 2^{nd}-level primitive candidates set		✓	✓		✓			2.4.4
enumerative search	-	[MDP+20b]	2020	monotonic PSTL		✓	✓		✓			2.4.5
robustness metric	TeLEx	[JTS+19]	2019	predicate candidates set		✓	✓		✓			2.4.6
genetic algorithm + GP-UCB	-	[NSBB18]	2018			✓		✓	✓			2.4.7
genetic algorithm	-	[SG19, AG20]	2019	ptPSTL		✓		✓	✓			2.4.7

naturally admits quantitative semantics [DM10]. STL with quantitative semantics enables measuring how far is an observed behavior from satisfying or violating a specification.

Let Θ be a set of terms of the form $f(R)$ where $R \subseteq S$ are subsets of variables and $f : \mathbb{R}^{|R|} \to \mathbb{R}$ are interpreted functions. The syntax of STL is given by the grammar

$$\varphi ::= \textbf{true} \mid f(R) > k \mid \neg\varphi \mid \varphi_1 \vee \varphi_2 \mid \varphi_1 \mathbf{U}_I \varphi_2 \mid \varphi_1 \mathbf{S}_I \varphi_2 \,,$$

where $f(R)$ are terms in Θ, k is either a constant in $\mathbb{Q}$ or a magnitude parameter and I are intervals with bounds that are either constants in $\mathbb{Q} \cup \{\infty\}$ or timing parameters. As customary, the timing interval I may be omitted when $I = [0, \infty)$ or $I = (0, \infty)$.

With respect to a signal w, the semantics of an STL formula is described via the satisfiability relation $(w, i) \models \varphi$, indicating that the signal w satisfies φ at the time t:

$$
\begin{array}{lll}
(w,t) \models \mathbf{true} & & \\
(w,t) \models f(R) > 0 & \text{iff} & f(w_R[t]) > 0 \\
(w,t) \models \neg\varphi & \text{iff} & (w,t) \not\models \varphi \\
(w,t) \models \varphi_1 \vee \varphi_2 & \text{iff} & (w,t) \models \varphi_1 \text{ or } (w,t) \models \varphi_2 \\
(w,t) \models \varphi_1 \mathbf{U}_I \varphi_2 & \text{iff} & \exists t' \in t \oplus I,\ (w,t') \models \varphi_2 \text{ and} \\
 & & \forall t'' \in (t,t'),\ (w,t'') \models \varphi_1 \\
(w,t) \models \varphi_1 \mathbf{S}_I \varphi_2 & \text{iff} & \exists t' \in t \ominus I,\ (w,t') \models \varphi_2 \text{ and} \\
 & & \forall t'' \in (t',t),\ (w,t'') \models \varphi_1 \,.
\end{array}
$$

Here we use the symbol $\oplus$ to denote the Minkowski sum extended to be defined between the scalar t and the set I as follows: $t \oplus I = \{t + a \mid a \in I\}$. Analogously, the symbol $\ominus$ denotes the Minkowski difference: $t \ominus I = \{t - a \mid a \in I\}$. We write $w \models \varphi$ when $(w, 0) \models \varphi$.

We use $\mathbf{S}$ and $\mathbf{U}$ as syntactic sugar for the *untimed* variants of the *since* $\mathbf{S}_{(0,\infty)}$ and *until* $\mathbf{U}_{(0,\infty)}$ operators. From the basic definition of STL, we can derive the following standard operators.

$$
\begin{array}{llcl}
\text{tautology} & \mathbf{true} & = & p \vee \neg p \\
\text{contradiction} & \mathbf{false} & = & \neg\mathbf{true} \\
\text{conjunction} & \varphi_1 \wedge \varphi_2 & = & \neg(\neg\varphi_1 \vee \neg\varphi_2) \\
\text{implication} & \varphi_1 \rightarrow \varphi_2 & = & \neg\varphi_1 \vee \varphi_2 \\
\text{eventually, finally} & \mathbf{F}_I\varphi & = & \mathbf{true}\,\mathbf{U}_I\,\varphi \\
\text{always, globally} & \mathbf{G}_I\varphi & = & \neg\mathbf{F}_I\neg\varphi \\
\text{once} & \mathbf{O}_I\varphi & = & \mathbf{true}\,\mathbf{S}_I\,\varphi \\
\text{historically} & \mathbf{H}_I\varphi & = & \neg\mathbf{O}_I\neg\varphi \\
\text{rising edge} & \mathbf{rise}(\varphi) & = & \varphi \wedge \neg\varphi\,\mathbf{S}\,\mathbf{true} \\
\text{falling edge} & \mathbf{fall}(\varphi) & = & \neg\varphi \wedge \varphi\,\mathbf{S}\,\mathbf{true}
\end{array}
$$

2.2.3 Parametric Signal Temporal Logic and its Validity Domains

In this section, we show how to extend STL to its *parametric* variant PSTL, following the approach introduced in [ADMN11]. Let $P = \{p_1, \ldots, p_m\}$ be a set of magnitude parameters and $Q = \{q_1, \ldots, q_k\}$ a set of timing parameters, defined over their respective domains $\mathcal{P}$ and $\mathcal{Q}$, say hyper-rectangles in $\mathbb{R}^m$ and $\mathbb{R}^k$. PSTL extends STL by allowing the amplitude constant k in $f(R) > k$ to be replaced by a magnitude parameter p and the bound constants in $\mathbf{U}_I$ and $\mathbf{S}_I$ to be replaced by timing parameters in Q. A parameter valuation $(\mathbf{u}, \mathbf{v}) \in \mathcal{P} \times \mathcal{Q}$ transforms a PSTL formula φ to an STL formula $\varphi_{\mathbf{u},\mathbf{v}}$ by substituting the values $(\mathbf{u}, \mathbf{v})$ in the parameters $(\mathbf{p}, \mathbf{q})$.

The semantics of PSTL formulas can be now defined using the notion of a *validity domain* [FR09] $\mathcal{D}(\varphi, w) \subseteq \mathcal{P} \times \mathcal{Q}$ that consists of all parameter valuations $(\mathbf{u}, \mathbf{v}) \in \mathcal{P} \times \mathcal{Q}$ such that w satisfies the STL formula $\varphi_{\mathbf{u},\mathbf{v}}$. In order to compute validity domains, we introduce the extended validity domain $D(\varphi, w) \subseteq \mathcal{P} \times \mathcal{Q} \times \mathbb{T}$, where $(\mathbf{u}, \mathbf{v}, t) \in D(\varphi, w)$ iff $(w, t) \models \varphi_{\mathbf{u},\mathbf{v}}$. The extended validity domain $D(\varphi, w)$ is defined inductively as follows:

$$
\begin{aligned}
D(\mathbf{true}, w, t) &= \mathcal{P} \times \mathcal{Q} \times \mathbb{T} \\
D(f(R) \geq k, w, t) &= \{(\mathbf{u}, \mathbf{v}, t) \mid f(R(t)) \geq k_{\mathbf{u},\mathbf{v}}\} \\
D(\neg\varphi, w, t) &= \overline{D(\varphi, w, t)} \\
D(\varphi_1 \vee \varphi_2, w, t) &= D(\varphi_1, w, t) \cup D(\varphi_2, w, t) \\
D(\varphi_1 \mathbf{U}_I \varphi_2, w, t) &= \{(\mathbf{u}, \mathbf{v}, t) \mid \exists t' \in t \oplus I_{\mathbf{u},\mathbf{v}} \text{ s.t. } (\mathbf{u}, \mathbf{v}, t') \in D(\varphi_2, w, t') \wedge \\
&\qquad \forall t'' \in [t, t'] \; (\mathbf{u}, \mathbf{v}, t'') \in D(\varphi_1, w, t'')\} \\
D(\varphi_1 \mathbf{S}_I \varphi_2, w, t) &= \{(\mathbf{u}, \mathbf{v}, t) \mid \exists t' \in t \ominus I_{\mathbf{u},\mathbf{v}} \text{ s.t. } (\mathbf{u}, \mathbf{v}, t') \in D(\varphi_2, w, t') \wedge \\
&\qquad \forall t'' \in [t', t] \; (\mathbf{u}, \mathbf{v}, t'') \in D(\varphi_1, w, t'')\}
\end{aligned}
$$

2.2.4 Signal Temporal Logic and its Quantitative Semantics

Apart from its classical qualitative semantics, presented in Section 2.2.2, STL also admits *quantitative* semantics [FP06, DM10]. The STL quantitative semantics is expressed in terms of a *robustness function* $\rho(\varphi, w, t)$ that assumes values in the set $\mathbb{R} \cup \{+\infty, -\infty\}$ of extended reals and measures how far is the signal w from violating or satisfying the specification φ at time t. It is defined inductively as follows:

$$
\begin{aligned}
\rho(\mathbf{true}, w, t) &= +\infty \\
\rho(f(R) > 0, w, t) &= f(w_R[t]) \\
\rho(\neg\varphi, w, t) &= -\rho(\varphi, w, t) \\
\rho(\varphi_1 \vee \varphi_2, w, t) &= \max(\rho(\varphi_1, w, t), \rho(\varphi_2, w, t)) \\
\rho(\varphi_1 \mathbf{U}_I \varphi_2, w, t) &= \sup_{t' \in t \oplus I} \min(\rho(\varphi_2, w, t'), \inf_{t'' \in (t, t')} \rho(\varphi_1, w, t'')) \\
\rho(\varphi_1 \mathbf{S}_I \varphi_2, w, t) &= \sup_{t' \in t \ominus I} \min(\rho(\varphi_2, w, t'), \inf_{t'' \in (t', t)} \rho(\varphi_1, w, t''))
\end{aligned}
$$

The quantitative semantics of STL have two fundamental properties. The first property states that the STL quantitative semantics definition is *sound* in the sense of [FP09] – the sign of the robustness degree $\rho(\varphi, w, t)$ (whenever it is different from zero) indicates the satisfaction status of the specification. Second, whenever w satisfies φ at time t with some (positive) robustness degree k, any other trace w' whose infinity-norm distance from w is smaller than k also satisfies φ at time t, where the infinity-norm distance between signals w and w' is defined as follows: $\|w - w'\|_\infty = \sup_{t \in \mathbb{T}} |w(t) - w'(t)|$.

2.3 Mining Parameters in PSTL Specification Templates

Template-based inference of STL specifications represents the first big family of specification mining methods. In this setting, the PSTL formula template is given by the user, and the mining procedure learns the parameter values that instantiate the template to a concrete STL specification.

2.3.1 Exact Computation of Validity Domains

The first mining procedure computes the exact validity domain of a PSTL specification φ given a signal w [ADMN11]. We consider the restricted class of *sampled* signals given as a finite sequence of (time stamp, value) pairs $(t_1, w(t_1)), \ldots, (t_l, w(t_l))$ for an increasing sequence of time stamps with $t_1 = 0$ and with the values between two consecutive time stamps being defined using *linear interpolation*. It is shown that under this restriction, the validity domains are semi-linear, meaning that they can be expressed as Boolean combinations of linear constraints over corresponding variables.

The computation of the exact validity domain follows closely the inductive definition of the extended validity domain and consists of three main steps:

1. *Base case:* we first compute the validity domain of the numeric predicates $f(R) \geq k$, using the transformation
$$\bigvee_{i=1}^{l-1} t_i \leq t < t_{i+1} \wedge \alpha_i t + \beta_i \geq k,$$
where $\alpha_i = \frac{w[t_{i+1}]-w[t_i]}{t_{i+1}-t_i}$ and $\beta_i = \frac{t_{i+1}w[t_i]-t_i w[t_{i+1}]}{t_{i+1}-t_i}$.
2. *Inductive step:* we apply the semantic definitions of extended validity domain for Boolean and temporal operators. This results in a quantified linear real arithmetic formula.
3. *Quantifier elimination:* we eliminate the quantifiers from the formula using an off-the-shelf satisfiability modulo theory (SMT) solver, obtaining a quantifier-free semi-linear constraint.

The resulting semi-linear constraint characterizes exactly the validity domain $\mathcal{D}(\varphi, w)$. Figure 2.1 illustrates the outcomes of the exact validity domain computation. Given the signal w, characterized by the sequence of sampling points $(0, 0), (2, 4), (4, 0), (5, 2), (6, 0)$ (Figure 2.1 (a)), the validity domain $\mathcal{D}(x > p, w)$ is characterized by the semi-linear constraint (depicted in Figure 2.1 (b)):

$$\begin{array}{lcr}
(0 \leq t < 2 \wedge p \leq 2t) & \vee & (1) \\
(2 \leq t < 4 \wedge p + 2t \leq 8) & \vee & (2) \\
(4 \leq t < 5 \wedge p + 8 \leq 2t) & \vee & (3) \\
(5 \leq t < 6 \wedge p + 2t \leq 12) & & (4)
\end{array}$$

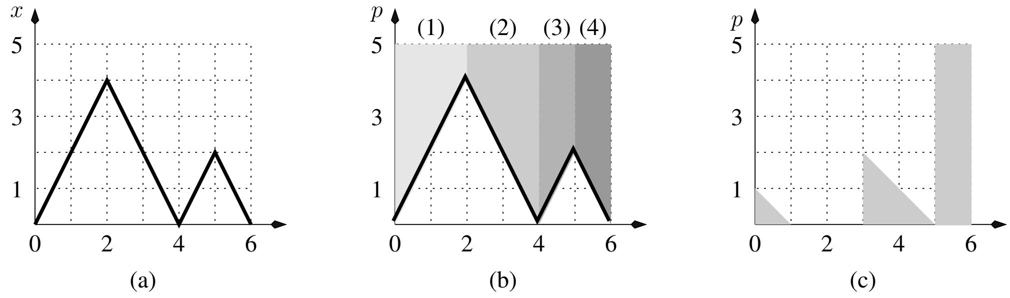


Figure 2.1: Exact computation of validity domains [ADMN11]: (a) example trace w, (b) validity domain $\mathcal{D}(x > p, w)$, and (c) validity domain $\mathcal{D}(\mathbf{G}_{[0,p]} x > 2, w)$.

The validity domain $\mathcal{D}(\mathbf{G}_{[0,p]} x > 2, w)$ is shown in Figure 2.1 (c) and is given by the semi-linear constraint:

$$\begin{array}{ll} (0 \leq t < 1 \wedge p < 1 - t) & \vee \\ (3 \leq t < 5 \wedge p < 5 - t) & \vee \\ (5 \leq t < 6 \wedge 0 \leq p < 5) & \end{array}$$

This concludes the description of the basic problem that consists in passively learning parameters from positive examples. The price paid for obtaining the exact validity domain is a high computational cost. In fact, eliminating quantifiers from linear real arithmetic formulas is exponential in the size of the formula and the number of alternating quantifiers [FR75, LW93].

2.3.2 Approximating Validity Domains with Monitoring and Search

The high cost of computing exactly validity domains can be mitigated by approximating them using runtime verification methods combined with search [ADMN11]. We assume a monitoring procedure $\mathcal{M}(\varphi, w)$ that takes as inputs an STL specification φ and a signal w. The monitor outputs **true** if $w \models \varphi$ and **false** otherwise. There are many publicly available implementations of STL monitoring libraries, including RTAMT [NY20], Reelay [Ulu19], MoonLight [BBL+20, NBB+22], Breach [Don10], S-TaLiRo [ALFS11].

The monitoring procedure can be combined with the parameter sampling to explore the parameter space of the specification and approximate the validity domain of the PSTL specification relative to a specific signal. Figure 2.2 depicts the overall idea of the iterative approach in which each iteration consists of the following steps:

1. selection of a parameter $(\mathbf{u}, \mathbf{v})$ and instantiation of the PSTL formula φ to obtain the STL formula $\varphi_{\mathbf{u},\mathbf{v}}$,
2. application of the monitoring procedure $\mathcal{M}(\varphi_{\mathbf{u},\mathbf{v}}, w)$ to check whether the trace w satisfies the instantiated specification, and

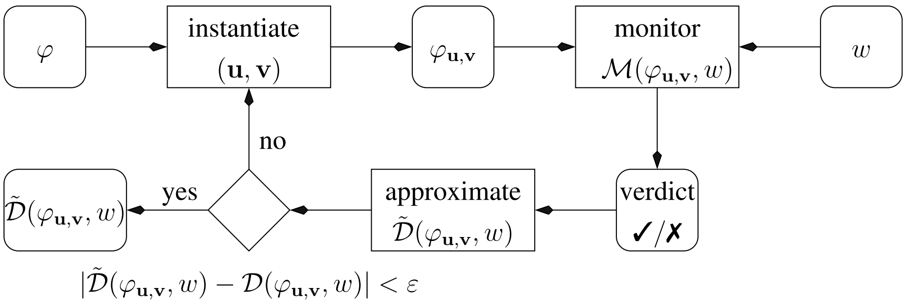


Figure 2.2: Workflow for approximating validity domain using monitoring [ADMN11].

3. approximation $\tilde{\mathcal{D}}(\varphi_{\mathbf{u},\mathbf{v}}, w)$ of the exact validity domain depending on the monitoring verdict.

The algorithm stops when the approximated validity domain $\tilde{\mathcal{D}}(\varphi_{\mathbf{u},\mathbf{v}}, w)$ is ε-close to the exact validity domain $\mathcal{D}(\varphi_{\mathbf{u},\mathbf{v}}, w)$ for some user-defined threshold ε. While the exact validity domain is not known and hence cannot be directly used in the computation of the ε-closeness, its area can be bounded in each iteration and used to conservatively estimate the measure. This high-level description of the procedure does not explain how to perform its two main steps: (i) the approximation of the validity domain from the monitoring verdict, and (ii) the estimation of the difference between the approximated and the exact validity domain. We discuss next these two steps.

To efficiently approximate a validity domain using the monitoring-based parameter exploration, we need to consider a well-behaving subset of PSTL, that we call *monotonic* PSTL, following the naming convention introduced in [VDJS17]. The definition of the monotonic PSTL fragment is based on the notion of polarity [ADMN11] of a PSTL formula φ: the *polarity* $\pi(p, \varphi)$ of a parameter p with respect to a PSTL formula φ is *positive* (*negative*) if it is easier to satisfy the formula as we increase (decrease) the value of p. In other words, for all φ with positive (negative) polarity, and all $p \geq p'$, we have that $\varphi_{p'} \rightarrow \varphi_p$ ($\varphi_p \rightarrow \varphi_{p'}$).

A PSTL formula φ is said to have *undetermined* polarity $\pi(p, \varphi)$ of a parameter p with respect to a PSTL formula φ if neither increasing nor decreasing the value of the parameter makes the satisfaction of the formula easier. Table 2.2 shows several examples of PSTL formulas and their polarity. For example, $x < p$ has positive polarity. It follows that $x < 1$ implies $x < 10$.

We say that a PSTL specification φ is monotonic in a parameter p if $\pi(p, \varphi)$ is determined. Similarly, a PSTL specification φ is said to be monotonic if for every parameter p, $\pi(p, \varphi)$ is determined. A monotonic PSTL specification φ induces a monotonic validity domain. This means that if the parameter instantiation $(v_1, \ldots, v_i, \ldots, v_n) \in \mathcal{P} \times \mathcal{Q}$ is in the validity domain $\mathcal{D}(\varphi, w)$ and the parameter p_i has positive (negative) polarity then $(v_1, \ldots, v_i', \ldots, v_n)$ is also in $\mathcal{D}(\varphi, w)$ for all $v_i' > v_i$ ($v_i' < v_i$). In practice, this means that a single check whether a parameter instantiation is in the validity domain allows to infer

Table 2.2: PSTL formulas and their polarity.

polarity		formula
positive	:	$x < p$
		$\mathbf{F}_{[0,p]}(x < 2)$
negative	:	$x > p$
		$\mathbf{G}_{[0,p]}(x < 2)$
undetermined	:	$(f(x) > p) \wedge (g(x) < p)$

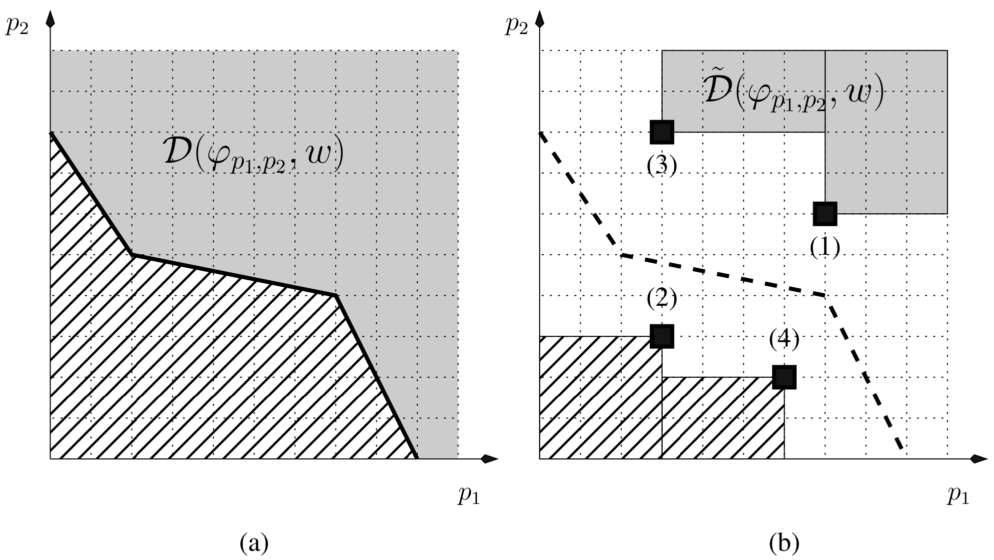


Figure 2.3: An example of validity domain approximation using monitoring.

the membership of an entire hyper-box of parameter values to that validity domain. We illustrate this idea on an example as follows.

Figure 2.3 (a) shows an exact validity domain (grey area) that we want to approximate. Figure 2.3 (b) depicts 4 steps of the approximation procedure. In each step, a sample (v_1, v_2) in the parameter space (p_1, p_2) is selected. We instantiate the PSTL formula φ to φ_{v_1,v_2}. We then use the monitoring procedure $\mathcal{M}(\varphi_{v_1,v_2}, w)$ to check whether the signal w satisfies the instantiated specification φ_{v_1,v_2}, i.e. whether $(v_1, v_2) \in \mathcal{D}(\varphi, w)$. After each membership check, we can infer regions of parameter values that are inside or outside the validity domain. These regions correspond to the 4 rectangles associated to the parameter samples, namely grey rectangles for the samples (1) and (3) and dashed rectangles for the samples (2) and (4). The white region is the uncertainty region. The area of the uncertainty region provides a distance to the exact validity domain. That is, the smaller the uncertainty region, the closer we are to the exact validity domain.

The specification mining method using the approximate computation of the validity domain presented in this section is implemented in the tool Breach [Don10].

2.3.3 Inferring PSTL Parameters using Tightness Metrics

Passive learning of PSTL parameters from positive examples has the risk of over-generalization. Selecting a parameter valuation from the validity domain is not a trivial task. The methods for identifying PSTL parameters presented in Sections 2.3.1 and 2.3.2 compute (or approximate) the entire validity domain and the selection task is left to the engineer. Good candidates for parameter values that avoid over-generalization are the points lying on the boundary of the validity domain. However, there are many of them and they are partially ordered.

In this section, we present the work that addresses the problem of passively inferring the optimal parameter values for a PSTL specification from a set of positive examples [JTS+17], without computing the entire validity domain. To this goal, the authors introduce the notion of *tightness* – ε-tight satisfaction of an STL specification φ by a signal w requires the existence of a magnitude or timing bound in φ whose perturbation by ε results in the violation of φ by w. In this setting, the parameter mining problem can be formulated as follows: given a PSTL specification template φ and a signal w, find a parameter valuation $\mathbf{v}$ such that w ε-tightly satisfies the STL formula $\varphi_{\mathbf{v}}$. The solution to this problem requires to solve an optimization problem where the objective function is the absolute value of the robustness degree of the mined formula under the additional constraint that the formula is satisfied. However, this optimization problem is hard to solve because the objective function is non-differentiable.

To remedy this problem, the authors embed a *tightness measure* as part of the new quantitative semantics for STL. The main idea is to make predicates and temporal operators *smooth*, hence differentiable. This enables the use of more efficient gradient-based numerical optimization algorithms (e.g. quasi-Newton) to solve the problem of inferring the STL specification with ε-tight satisfaction. The resulting tightness metric is shown to be sound, i.e. the adapted robustness degree is non-negative iff the formula is satisfied. This approach is implemented in the tool TeLEx [JTS+17].

2.3.4 Mining Specification Parameters with Computation Graphs

This section presents the framework stlcg [LAP20] introduced to efficiently evaluate the robustness of STL formulas using the computation graphs from the machine learning domain. The built-in backpropagation mechanism of the computation graphs allows for learning the parameter values of a given PSTL formula which e.g. maximize the STL robustness w.r.t. a given set of signal examples. In general, it is possible to infer parameter values w.r.t. any, possibly non-linear, differentiable constraints which can be encoded into the objective function of an optimization problem. We note that stlcg also enables a seamless integration of the PSTL framework into any machine learning problem which might use STL to achieve a particular objective, e.g. using the STL robustness as

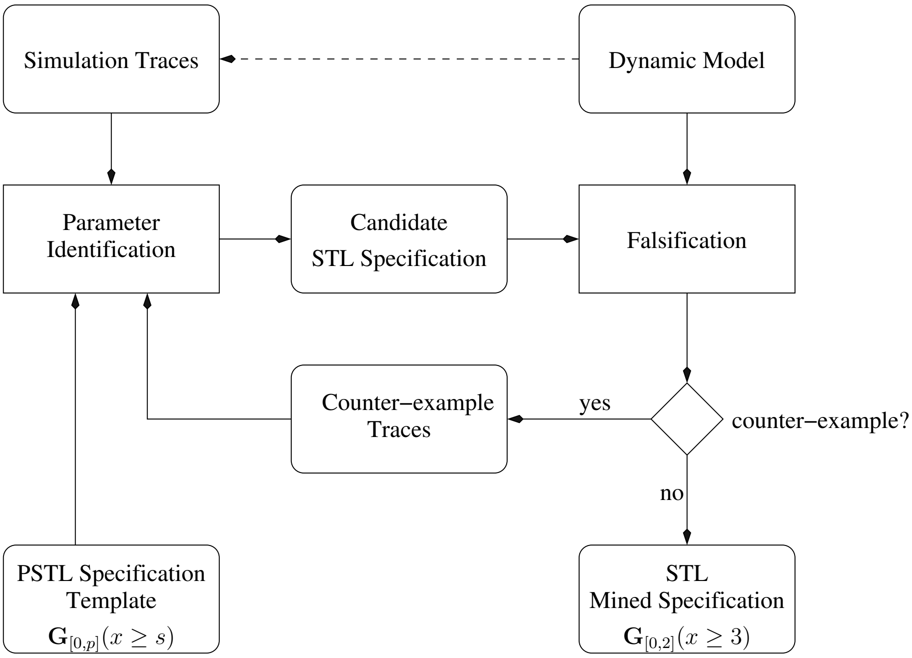


Figure 2.4: Overview of specification mining from a control model [JDDS13, JDDS15].

a reward function in a reinforcement learning problem. stlcg currently does not support the mining of time parameters which is mentioned to be covered in future work.

2.3.5 Mining Specification Parameters from Dynamic Models

So far, all presented methods assumed a passive learning approach with a fixed set of examples from which a STL specification is mined. In this section, we present an active learning approach to mining PSTL parameters from signals. The approach assumes the availability of a dynamic model that can be simulated and that can generate new signals on demand. This is in contrast to the previously described approaches that assume a fixed set of signals. The generation of new signals from the model enables refining the learned parameters.

The specification mining problem is stated as follows: given a dynamic model M and a PSTL specification template φ, find an STL instantiation $\varphi_{\mathbf{u},\mathbf{v}}$ of φ such that *all traces* generated by M satisfy $\varphi_{\mathbf{u},\mathbf{v}}$. This problem was originally stated and solved [YHF12] with the restriction to the specification templates with a single parameter. The approach was subsequently generalized to an arbitrary number of parameters, providing an efficient solution in the case of monotonic PSTL [JDDS13, JDDS15].

The problem is solved using an iterative procedure, illustrated in Figure 2.4. The procedure takes as input a PSTL specification template and a dynamic model. In every iteration, it performs the following steps:

- *Step 1:* compute a candidate STL specification, using for example the mining procedure presented in Section 2.3.2.
- *Step 2:* Check whether the dynamic model can generate a simulation that violates the candidate specification, using *falsification* [NSF+10].
- *Step 3:* If the falsification method does not find a counter-example to the candidate specification, accept and return that specification. Otherwise, add the counter-example to the set of simulation traces and repeat from Step 1.

The key step in this approach is the application of the falsification method. The falsification problem is formulated as follows: given an STL φ, find an input signal u such that $M(u)$ violates φ. To solve efficiently this problem, the STL specification is interpreted using quantitative semantics, and the search for the input u is formulated as the following optimization task:

$$\rho^* = \min_u \rho(\varphi, M(u)).$$

If the resulting $\rho^*(\varphi, M(u)) < 0$, we return the input $u^* = \text{argmin}_u \rho(\varphi, M(u))$ that yields a counter-example simulation trace. The different flavors of the specification mining method presented in this section are implemented in the tools S-TaLiRo [ALFS11] and Breach [Don10].

2.3.6 Logical Clustering with PSTL

Logical clustering is a method proposed in [VDJS17] that combines parameter inference in PSTL specification templates with unsupervised machine learning to group and characterize similar qualitative signals. The logical clustering procedure takes as input a monotonic PSTL specification template φ and a set of signals $\mathcal{W}$. The core idea of the procedure is very simple and consists of the following steps:

- *Step 1:* for each signal $w \in \mathcal{W}$, project the signal to a representative template parameter $\mathbf{v}$ in the validity domain $\mathcal{D}(\varphi, w)$.
- *Step 2:* apply an off-the-shelf clustering method with representative template parameters as features to group together qualitatively similar signals under the lens of the PSTL specification.
- *Step 3:* for each cluster, define an STL instantiation of φ that characterizes well the cluster.

There are two difficulties in realizing the above algorithm sketch: (i) finding the single point in the validity domain that is a good qualitative representative of the signal behavior, and (ii) finding an appropriate STL characterization of a cluster.

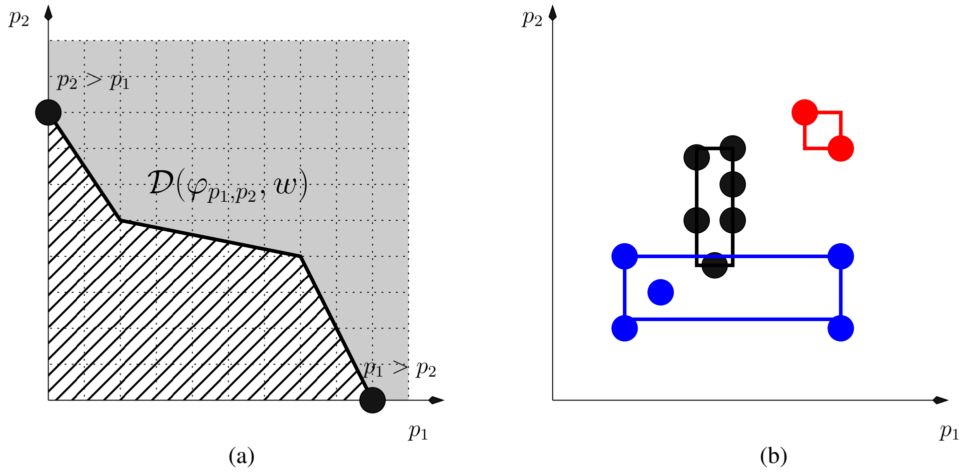


Figure 2.5: Example of logical clustering: (a) representative template parameters for different lexicographic orders, and (b) hyper-rectangle approximation of clusters.

The intuition behind Step 1 is to project the signal w to a parameter valuation $\mathbf{v}$ that yields an STL instantiation $\varphi_{\mathbf{v}}$ of the PSTL template φ such that w only marginally satisfies $\varphi_{\mathbf{v}}$ under the quantitative semantics interpretation. The choice of using the monotonic fragment of PSTL plays a key role in this step – a parameter p with determined polarity in φ induces a total order in its domain. Given a signal w and a PSTL specification φ with a single determined parameter p, the extreme value of p in $\mathcal{D}(\varphi, w)$ is arguably the best representative template parameter. The problem is that many PSTL specification templates have more than one parameter. Parameters are often independent from each other and valuations of different parameters induce a partial order. This issue can be addressed by linearizing a partial order into a total lexicographic order. Figure 2.5 (a) shows a 2-dimensional validity domain $\mathcal{D}(\varphi, w)$ for some PSTL formula and signal w and two projections of w to representative template parameters according to two lexicographic orders $p_1 > p_2$ and $p_2 > p_1$.

The off-the-shelf clustering methods do not yield an outcome that can be easily plugged in interpretable STL characterizations of the resulting clusters. For a given set of points in the cluster, the tightest-enclosing hyper-rectangle, as shown in Figure 2.5 (b), is a useful approximation of the cluster. The hyper-rectangle description can be embedded in the STL formula in a straight-forward manner by using intervals, thus providing a specification that is interpretable by the human. We note that the hyper-rectangles from different clusters can overlap (see Figure 2.5 (b)). This can be addressed either by defining a new cluster for the points in the intersection, or by requiring additional guidance from the designer.

The logical clustering method is extended in [VGD$^+$18] by dropping the lexicographic ordering in favor of a more general ordering-free approach. Given a PSTL specification template φ and two signals w and w', the authors introduce the notion of a *logical* *distance* $d_\varphi(w, w')$, which is the Hausdorff distance between the boundaries of the validity domains $\mathcal{D}(\varphi, w)$ and $\mathcal{D}(\varphi, w')$ and can be approximated with arbitrary decision.

2.4 Mining Complete STL Specifications

In this section, we present the second major family of specification mining methods that infer the entire STL formula for observed signals, without following a fixed template. All template-free mining methods follow a similar approach. First, the structure of a PSTL specification template is learned from data. Second, the PSTL template is instantiated to a concrete STL specification, using parameter inference methods similar to the ones described in Section 2.3. Most of these approaches consider only specific fragments of STL that enable the use of specific optimization techniques. Mining STL specifications with the *until* operator results to be quite challenging and it requires the use of special metaheuristics such as genetic algorithms.

2.4.1 Mining STL Specifications for Signal Classification

We first present a passive learning approach that infers an STL specification from both positive and negative examples [KJMA+14]. The resulting STL formula serves as a classifier that is able to distinguish between two types of signals.

This work restricts its attention to the *reactive* fragment of STL, which aims to capture the *cause-effect* relations from signals. Reactive STL (rSTL) is defined using the following syntax:

$$\varphi ::= \mathbf{F}_{[\tau_1,\tau_2)}(\varphi_c \Rightarrow \varphi_e), \tag{2.1}$$

where φ_c and φ_e are the *cause* and the *effect* formulas of the following form:

$$\varphi_i ::= \mathbf{F}_{[\tau_1,\tau_2)}p \mid \mathbf{G}_{[\tau_1,\tau_2)}p \mid \varphi_i \vee \varphi_i \mid \varphi_i \wedge \varphi_i, \tag{2.2}$$

where $i \in \{c, e\}$ and p is a linear predicate, i.e. an inequality over a scalar variable. Reactive PSTL (rPSTL) is defined in a similar manner. We observe that both rSTL and rPSTL admit natural partial ordering of specifications with respect to the satisfaction relation. For instance, any signal w that satisfies $\varphi_1 = \mathbf{G}_{[0,a)}x \leq k$, also satisfies $\varphi_2 = \mathbf{F}_{[0,a)}x \leq k$. We denote this fact by $\varphi_1 \preceq \varphi_2$.

The specification mining problem is stated as follows: Given a set of *labeled* signals, infer a rSTL formula $\varphi = \mathbf{F}_{[\tau_1,\tau_2)}(\varphi_c \Rightarrow \varphi_e)$ that (i) is consistent with the signal labels, and (ii) can be used to predict the label from a prefix of the signal. The procedure assumes that the effect of the rSTL specification is captured in the *suffix* of the signal. The duration d of that suffix is provided by the user. The procedure consists of two separate phases reflecting, respectively, goals (i) and (ii):

- *Step 1:* infer the effect formula φ_e from the suffix of the signals, and

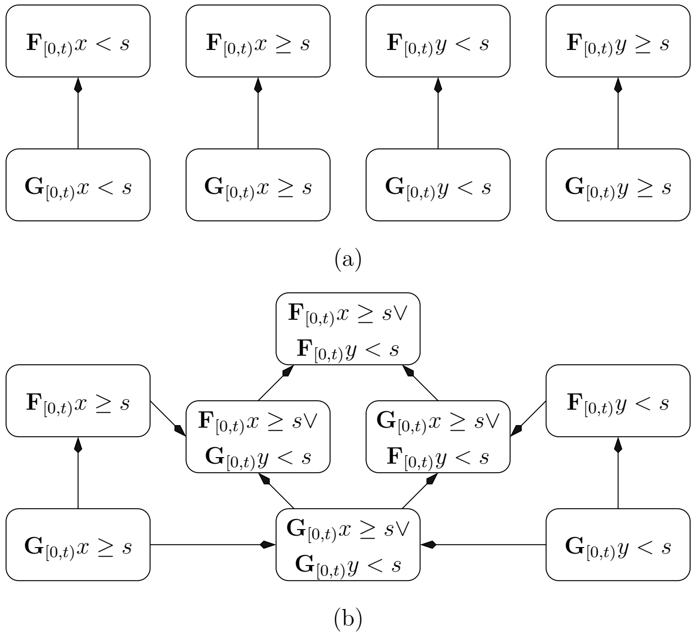


Figure 2.6: Example of a finite sub-DAG with candidate specifications built from observing a two-dimensional (x, y)-signal: (a) at initialization, and (b) after a refinement [KJMA+14].

- *Step 2:* given the (now fixed) effect formula φ_e mined in Step 1, infer the full specification $\varphi = \mathbf{F}_{[\tau_1,\tau_2)}(\varphi_c \Rightarrow \varphi_e)$ that maximizes how well φ predicts the label of training signals.

Both Step 1 and 2 are modeled as optimization problems that maximize the prediction of the label, while minimizing the size of the mined formula (the optimization problem in Step 1 is defined considering only the suffix of the signals, while the one in Step 2 deals with the entire signals). To efficiently solve the two optimization problems, the procedure uses the partial order $\preceq$ of the candidate specifications, represented by a directed acyclic graph (DAG). The DAG contains all candidate formulas as its nodes, while the directed edges represent the partial order relation between formulas. In essence, finding the optimal specification consists in simultaneously searching for the optimal formula structure in the DAG and its optimal parameters. The mined specification should ideally be the lowest order formula that differentiates between positive and negative examples.

The DAG is infinite, hence the procedure constructs a finite sub-graph of possible candidate specifications and expands it on-the-fly when needed. The procedure starts the search from the DAG bottom elements – the most exclusive formulas of the form $\mathbf{G}_{[t_1,t_2]}(x \circ s)$ with $\circ \in \{\leq, \geq\}$. It then follows the directed edges in the graph until a specification satisfied by all the positive examples is found. At each iteration, a candidate template rPSTL formula is explored and its parameters are estimated using

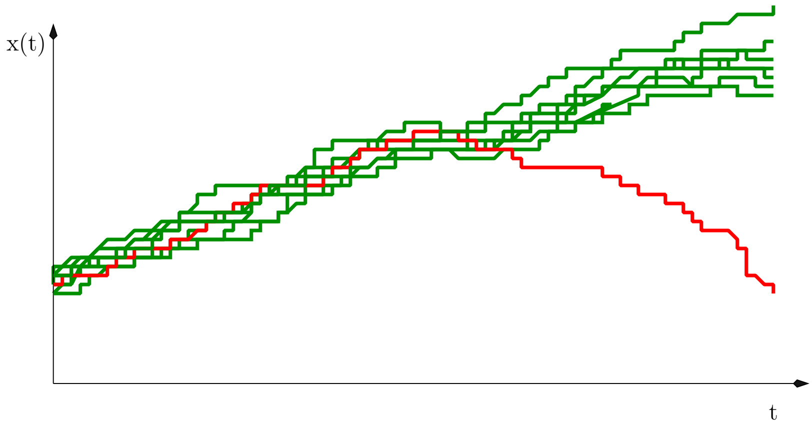


Figure 2.7: Example of nominal (green) and anomalous (red) signals. (For interpretation of the colors in the figure(s), the reader is referred to the web version of this article.)

classical optimization methods (e.g., simulated annealing). Figure 2.6 illustrates the DAG refinement iteration on a simple example. If the resulting rSTL formula correctly classifies all positive examples and only a fraction of the negative examples above some high threshold, the algorithm stops and the candidate specification is selected. Otherwise, the high-cost node is pruned from the DAG and the DAG is expanded to new candidate formulas. The expansion is done by combining nodes that create a specification with higher order with respect to their parent(s) node(s). The procedure is then repeated. This procedure is extended to an online learning setting in [KJB17].

2.4.2 Mining STL Specifications for Anomaly Detection

We summarize in this section a procedure for unsupervised passive learning of STL specifications [JKB14]. This procedure adapts the supervised learning approach described in Section 2.4.1 to the setting where signals are not labeled. The aim of this procedure is to learn a specification that characterizes *nominal* behaviour of a system. Any signal that violates the mined specification is then considered to be *anomalous*.

The specification mining procedure related to the *inference* fragment of STL is introduced in [JKB14]. Inference STL (iSTL) removes from the rSTL fragment the restriction that top-level formulas must contain a Boolean implication, and is thus defined as follows:

$$\varphi ::= \mathbf{F}_{[0,T)}\varphi$$

where φ has the form of cause or effect formulas (2.2) in Section 2.4.1.

The specification mining procedure inherits several ideas from the algorithm described in Section 2.4.1, including the formulation of the mining as an optimization problem and the search of candidate formulas in the DAG representing the partial order of specifications. However, there are two key differences from the previous work.

First, since the signals are not labeled, the procedure makes two assumptions to be able to distinguish between nominal and anomalous behavior: (i) anomalous signals are rare and with low probability, and (ii) the signals associated to nominal behaviors are qualitatively different from the anomalous ones. Figure 2.7 depicts these assumptions. Second, the objective function is formalized using one-class support vector machines (SVM). This method moves the optimization problem to a higher-dimensional space that increases the separation between nominal and anomalous behaviors. The objective function is designed to simultaneously

- minimize the number of signals classified as anomalous – this is consistent with the first assumption,
- maximize the separation between nominal and anomalous behavior – this is consistent with the second assumption, and
- penalize, using a tightness metric, the number of signals that satisfy with large robustness the specification – to avoid trivial specifications such as **true** that is satisfied by all signals.

2.4.3 Mining STL Specifications using Grids

In this section, we present another passive learning approach for mining STL specifications using a *grid-based* discretization of the time and value domain [VIB+17].

We sketch the mining procedure, also illustrated in Figure 2.8, for the special case of one-dimensional signals:

- *Step 1:* Discretize the two-dimensional time-value space into a rectangular grid, with granularity regulated by the user.
- *Step 2:* For every signal, mark all the cells in the grid that it covers (intersects).
- *Step 3:* Cluster the signals according to the similarity of cells that they cover, where the exact notion of similarity is defined by the user.
- *Step 4:* For each cluster, map all the cells covered by the signals in that cluster.
- *Step 5:* Translate the set of cells associated to each cluster to an STL formula that characterizes the cluster.
- *Step 6:* Build the overall STL formula as disjunction of the cluster STL formulas.

The main consideration to take when using this procedure is the granularity of the grid, which affects the tightness of the resulting STL formula. Small granularity may result in a large over-fitting STL formula, while large granularity may result in an STL formula that accepts too many signals and provides a poor characterization of the signals.

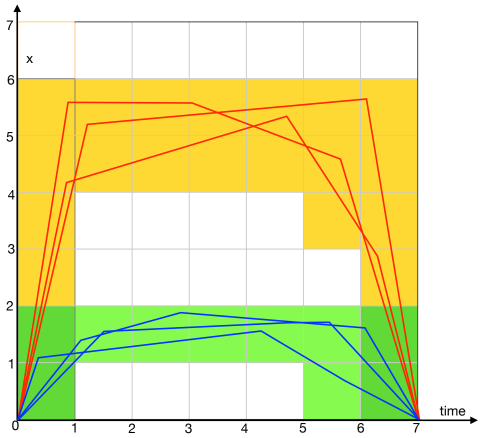


Figure 2.8: The STL formula which describes the yellow cluster and therefore the included red signals is: $\varphi_1 = \mathbf{G}_{[0,7]}(x \leq 6) \wedge \mathbf{G}_{[0,1]}(x > 0) \wedge \mathbf{G}_{[1,5]}(x > 4) \wedge \mathbf{G}_{[5,6]}(x > 3) \wedge \mathbf{G}_{[6,7]}(x > 0)$. The one for the green cluster (blue signals) is $\varphi_2 = \mathbf{G}_{[0,7]}(x \leq 2) \wedge \mathbf{G}_{[0,1]}(x > 0) \wedge \mathbf{G}_{[1,5]}(x > 1) \wedge \mathbf{G}_{[5,7]}(x > 0)$. The final formula is obtained by disjunction between the two STL formulas: $\varphi = \varphi_1 \vee \varphi_2$. This example is taken from [VIB+17].

Translating the set of cells to an STL formula works as follows. For every column i in the grid (time steps), we identify the bottom-most j and the top-most k rows (value steps) such that the cells (i, j) and (i, k) are in the set. We then describe this column using the STL formula $\mathbf{G}_{[t_1,t_2]}(x \geq s_1 \wedge x \leq s_2)$, where t_1 and t_2 are the start and the end times of column i, s_1 is the minimum value of the row j and s_2 is the maximum value of the row k. The overall STL specification is the conjunction of formulas obtained for each column.

We finally note that the generalization of the mining procedure from one-dimensional to multi-dimensional signals is straight-forward. The presented procedure is applied to each component of the multi-dimensional signal and the resulting formulas from each component are combined using conjunction.

2.4.4 Mining STL Specifications with Decision Trees

Decision trees present an alternative for mining STL specifications. We review in this section three methods based on decision trees for inferring STL formulas: (i) offline supervised learning from positive and negative examples, (ii) online supervised learning, and (iii) offline unsupervised learning.

Offline Supervised Learning

In this section, we present an approach for passive learning of STL specifications from positive and negative examples using *binary decision trees* [BVP+16].

The objective is the same as in Section 2.4.1 and consists in finding an STL specification that minimizes the mis-classification rate. The specification mining procedure makes use of a key insight – the possibility to map a fragment of STL to decision trees, and more specifically to *binary* decision trees. This enables using classical methods on decision trees to solve the problem – first, build a tree that classifies signals, and then translate the tree to an STL formula.

The procedure takes three parameters as input: (i) a set of PSTL template specifications that are allowed to be used for generating the final specification, (ii) an *impurity* measure that defines the criteria for preferring one PSTL template over another, and (iii) a stopping criterion. Two classes of primitive PSTL formulas are proposed in the original paper [BVP+16] – first-level primitives of the form $\mathbf{F}_I(x \circ s)$ or $\mathbf{G}_I(x \circ s)$, and second-level primitives of the form $\mathbf{F}_I\mathbf{G}_J(x \circ s)$ or $\mathbf{G}_I\mathbf{F}_J(x \circ s)$, where $\circ \in \{\leq, \geq\}$. The authors propose canonical measures, including information, Gini and mis-classification gain, as possible impurity measures. These measures are adapted to better generalize unseen data by defining boundaries that separate two classes of signals as much as possible based on the robustness interpretation of the STL specifications. Finally, the proposed termination conditions are based on the percentage of signals belonging to the same class associated to a node in the decision tree and the depth of the tree.

The decision tree is constructed recursively in three major steps:

- *Step 1:* Check if the terminal conditions are satisfied on the current node.
- *Step 2:* If yes, create a leaf node, associate it to the label that gives the best classification and stop the current recursion iteration. If not, create a non-terminal node and associate it with the STL formula that best splits the data reaching that node.
- *Step 3:* Partition the data that reach the current node according to whether they satisfy or violate its associate specification and build accordingly two sub-trees. Apply the same steps to each sub-tree.

Step 2 requires associating the new node to the STL formula that "best" splits the data. The STL formulas are chosen among all instantiations of PSTL specification templates defined by the user. The notion of best refers to the optimization of the impurity measure, also given by the user.

The remaining step consists in translating the binary decision tree to the overall STL formula. This is also done recursively on the structure of the tree. The leave nodes with positive (negative) labels are associated to the formula **true** (**false**). The STL formula associated to a tree with the STL formula φ in the root is the disjunction between two conjunctions: (i) the conjunction of φ and the STL formula associated to the left child sub-tree, and (ii) the conjunction between $\neg\varphi$ and the STL formula associated to the right child sub-tree. An example of the procedure is depicted in Figure 2.9.

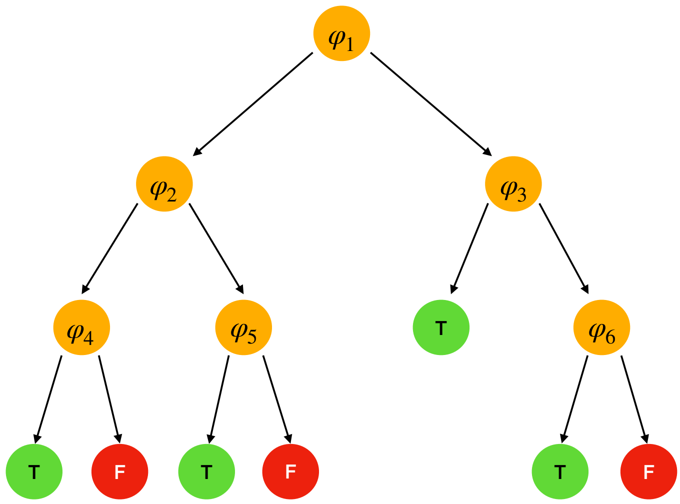


Figure 2.9: Example of transformation from binary tree to STL formula as presented in [BVP+16]. The inferred formula in this case is: $\varphi_{\text{tree}} = \big(\varphi_1 \wedge ((\varphi_2 \wedge \varphi_4) \vee (\neg\varphi_2 \wedge \varphi_5))\big) \vee \big(\neg\varphi_1 \wedge (\varphi_3 \vee (\neg\varphi_3 \wedge \varphi_6))\big)$. The STL formulas φ_i (for $i \in \{1, \ldots, 6\}$) represent the instantiations of PSTL primitives associated to the corresponding nodes.

The decision tree generated by the above procedure can be in some cases very large with high classification accuracy on the training data. Such a tree may be over-fitting the training data and may not give explainable insights about the data. The dimensions of the constructed tree can be reduced using a pruning algorithm [BB21]. The main idea is to equip the tree with a cost that takes into account the percentage of mis-classified data and the width of the tree. A sequence of trees with decreasing size is constructed by varying the weights that are given to the mis-classification rate and the width of the tree in the cost. Finally, it is possible to choose among the obtained trees picking the one that has the minimum classification error on the validation set.

Online Supervised Learning

This section describes a procedure that extends the supervised mining approach based on decision trees [BVP+16] presented in the previous section from offline to online learning [BB18]. In the online learning scenario, data is not fully available prior to training, but rather arrives continuously over time. The aim of the online mining procedure is an efficient on-the-fly generation and refinement of the STL specification according to the new observed data.

The overall structure of the mining procedure is similar to its offline counterpart. That is, the decision tree classifies the observed labeled signals. The candidate specifications belong to a set of primitive formulas and the combination thereof. Finally, an impurity measure is used to define a criterion for selecting the primitive formula that splits data well with respect to their labels.

The key difference of the online mining method is its mechanism to update the learned STL specification with the arrival of new signals. There is a trivial solution to the online mining problem that consists in applying the offline procedure at each new arrival of data. This solution is nevertheless highly inefficient and expensive.

The main difficulty for an efficient update of a decision tree happens when a new signal causes the change of the best primitive formula associated to a node, according to the used impurity measure. To make the online learning procedure efficient, decision tree updates of this kind are postponed to when the new STL formula is reasonably certain to hold in the future, a decision based on a probabilistic condition.

Offline Unsupervised Learning

This section presents a procedure for offline unsupervised mining of STL specifications using decision trees [BB17]. In contrast to the other approaches in this section, this procedure clusters signals according to their temporal logic properties and then describes and discriminates the groups by appropriate STL formulas. In this setting, the input data does not need to be labeled since the clustering problem is an unsupervised problem that require to work uniquely on raw signals. The output of the algorithm is a hierarchical tree where each node verifies if a temporal property of the data is satisfied or not and each leaf represents a cluster that is mapped to an STL formula.

The mining procedure is similar to its supervised offline counterpart. However, the interpretation of the decision tree is different. The decision tree provides a hierarchical representation of signal clusters. The initial node contains the entire set of signals. When moving down the tree hierarchy, each node presents a refined set of similar signals. The leaf nodes represent the actual signal clusters. Each node also has an associated STL specification that characterizes the set of signals represented by the node. This STL formula aims to maximize discrimination between the signals associated to that node, and all the other signals.

The discrimination is achieved using a *homogeneity* measure that quantifies the distance between two signals (e.g. Euclidean Distance or Dynamic Time Warping).

2.4.5 Mining STL Specifications by Enumeration

This section presents a procedure for learning STL specification classifiers by combining enumerative search of monotonic PSTL templates with robustness-based instantiation of PSTL parameters [MDP$^+$20b].

The procedure is summarized in Algorithm 2.1 and consists of two major interleaved steps:

Algorithm 2.1: Enumerative mining approach presented in [MDP+20b]

Input: Maximum length max_length, acceptable MCR δ, set of traces $\mathcal{S}$

Output: STL formula with low MCR (if it is found)

$l \leftarrow 1$
$D \leftarrow \emptyset$
while $l \leq max\_length$ **do**
  $\Phi \leftarrow$ Enumerate(l)
  foreach $\varphi \in \Phi$ **do**
    $s_\varphi \leftarrow$ ComputeSignature(φ)
  end
  while $\Phi \setminus D \neq \emptyset$ **do**
    pick $\varphi \in \Phi \setminus D$
    $is\_new \leftarrow$ true
    foreach $\psi \in D$ **do**
      if $s_\varphi = s_\psi$ **then**
        $D \leftarrow D \cup \{\varphi\}$
        $is\_new \leftarrow$ false
        break
      end
    end
    if is_new **then**
      $\boldsymbol{v} \leftarrow$SelectValuation($\varphi, \mathcal{S}$)
      if MCR($\varphi_{\boldsymbol{v}}$) $\leq \delta$ **then**
        return $\varphi_{\boldsymbol{v}}$
      else
        $D \leftarrow D \cup \{\varphi\}$
      end
    end
  end
  $l \leftarrow l + 1$
end
return *no STL formula found*

- *Step 1:* systematic enumeration of PSTL templates ordered by their size (function *Enumerate* in line 4), and
- *Step 2:* logical classification by instantiation of each template to an STL formula by minimizing its mis-classification rate.

The above procedure stops as soon as an STL formula with low mis-classification rate is found. Due to the exploration of specification templates in the increasing order of their size, the resulting STL formula is guaranteed to be the smallest one that satisfies the criterion.

Logical classification for every PSTL specification in the enumeration is computationally expensive with the risk to repeat the parameter instantiation for many equivalent PSTL formulas. To detect and remove equivalent, and hence redundant PSTL formulas before their logical classification is performed, the notion of signature is used to approximately check PSTL specifications for equivalence. The signature of a PSTL formula corresponds to a matrix containing the robustness values with respect to different traces and parameter valuations (this is what function *ComputeSignature* computes in line 6). Finding the parameters of a PSTL formula that minimizes mis-classification rate is done using an adaptation of the binary search algorithm [ADMN11] that approximates the satisfaction boundary of the formula (function *SelectValuation* in line 19). An instance of the binary search algorithm is illustrated in Figure 2.10.

The above procedure is adapted to learn the environment assumption in a CPS model [MDP20a]. Following the observation that if the input traces satisfy this assumption, then the corresponding output traces satisfy the output requirement, the aim is to learn an STL classifier that partitions the input signals into the ones that result in the satisfaction, and the others that result in the violation of the system requirements. The robustness-driven method to learn parameter values is replaced by a decision-tree based approach.

2.4.6 Mining STL Specifications from Positive Examples Guided by Robustness Metrics

This section presents an incremental procedure for passively mining STL specifications from positive examples. This procedure extends the approach that infers parameters in STL specification templates guided by tightness metrics [JTS+17] presented in Section 2.3.3 with the ability to also learn the specification structure from data [JTS+19]. The main objective is to learn the tightest STL formula that is satisfied by all positive examples.

The core of the method is to build the formula structure incrementally, constructing in each iteration a more complex STL formula. The overall approach is summarized in Algorithm 2.2. The algorithm takes as input a set of traces and a set of candidate PSTL predicate templates (i.e. parametric inequalities over the signals such as $f(x) \geq k$, where x is a signal and k a parameter). The procedure is iterative, and in each iteration it

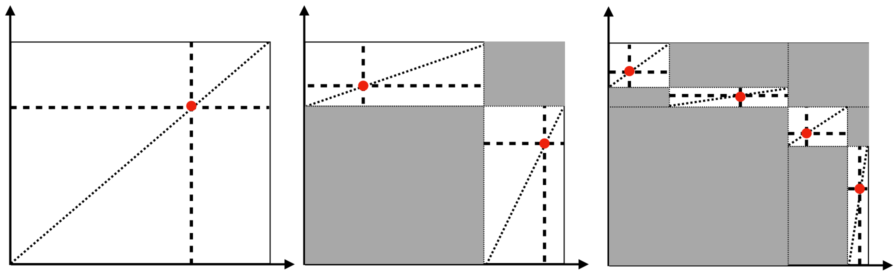


Figure 2.10: By leveraging the monotonicity of the PSTL formula, the adaptation of the binary search algorithm consists in recursively dividing the parameter space into sub-regions containing the formula's satisfaction boundary. Each red point is the valuation on the diagonal of the respective hyper-box that instantiates an STL formula with robustness value equal to zero. Depending on the monotonicity direction, two of the generated sub-regions are excluded from further search (grey boxes) because they only contain either all valid or invalid valuations, hence no points on the satisfaction boundary. The procedure is then iteratively repeated on the remaining sub-regions.

updates a set Ψ of candidate specifications. This set is initiated with simple formulas of the form $\mathbf{F}_I p$, $\mathbf{G}_I p$ and $p\mathbf{U}_I q$, where p and q are PSTL predicate templates and $I = [a, b]$ is a parameterized time interval (function *InitializeSet* in line 1). An iteration of the procedure consists of the following steps:

- *Step 1:* apply the parameter inference procedure guided by the tightness metric to all the PSTL specifications in the set Ψ (function *ParameterInference* in line 5) and create the corresponding set of STL formulas Φ.
- *Step 2:* filter the resulting STL formulas according to their robustness w.r.t. the positive examples (function *Filter* in line 8). The filtering uses a selection heuristic, which initially keeps the set of formulas diverse (keeping even some formulas with negative robustness), then it gradually favors only formulas which are tightly satisfied by the set of traces.
- *Step 3:* if no STL formula remains in Φ after filtering, the procedure is not able to find an STL specification that is consistent with the positive examples and the procedure stops.
- *Step 4:* if there is only one remaining STL formula in Φ after filtering, the procedure returns this as output specification and stops.
- *Step 5:* augment the set Ψ by nesting existing sub-formulas with another temporal ($\mathbf{F}_I$, $\mathbf{G}_I$ or $\mathbf{U}_I$) operator, and combining the existing sub-formulas with disjunction (function *Augment* in line 14).

Algorithm 2.2: Mining procedure in [JTS+19]

Input: Set of traces $\mathcal{S}$, set of PSTL predicate templates P
Output: STL formula (if it is found)

$\Psi \leftarrow$InitializeSet(P)
while *true* **do**
  $\Phi \leftarrow \emptyset$
  foreach $\varphi \in \Psi$ **do**
    $\varphi_{\boldsymbol{v}} \leftarrow$ ParameterInference(φ)
    $\Phi \leftarrow \Phi \cup \{\varphi_{\boldsymbol{v}}\}$
  end
  $\Phi \leftarrow$Filter(Φ)
  if $|\Phi| = 0$ **then**
    return *no STL formula found*
  else if $|\Phi| = 1$ **then**
    return $\varphi_v \in \Phi$
  else
    $\Psi \leftarrow$Augment(Ψ)
  end
end

2.4.7 Mining STL Specifications with Genetic Algorithms

Genetic algorithms can be used to tackle the problem of automatically inferring an STL formula that acts as a two-class classifier among two sets of signals, giving also insights on what distinguishes the two sets. The traces in the training data set needs to have been previously labeled as either desired or anomalous trajectories of the system under study.

Genetic algorithms take inspiration from the process of natural selection among species. In the specification mining applications, the term *individual* refers to a candidate formula. The definition of three aspects characterizes a particular genetic algorithm: 1) fitness function (indicating how an individual is suitable to survive, i.e., how good a formula discriminates between the sets of positive and negative examples), 2) crossover operator (how a new individual is generated starting from two others, i.e., how formulas are combined to have new ones), and 3) mutation operator (what random changes may happen to an individual formula).

There are two similar approaches that use genetic algorithms for mining STL formulas. The first approach is a supervised learning algorithm for mining an STL specification classifier by combining an Evolutionary Algorithm (EA) to learn the structure of the formula with a Gaussian Process Upper Confidence Bound Algorithm (GP-UCB) to infer its optimal parameters [NSBB18]. The approach in [NSBB18] considers the full STL specification language and not just specific fragments. Another genetic algorithm to mine specifications is presented in [SG19, AG20]. The second approach is a pure genetic algorithm approach and restricts its attention to the past STL (ptSTL) fragment, which forbids the use of future temporal operators. Since these two approaches are similar, we sketch in more detail the former approach only.

The procedure takes as input two labeled datasets: the set of positive and the set of negative examples. The procedure is described in Algorithm 2.3 and starts with the generation of a set of templates via conjunction of predicates or application of temporal operators to predicates (function *InitializeTemplateSet* in line 1). The rest of the algorithm consists of the structure mining and the parameter inference steps, which are combined in a compositional manner. In every iteration of the procedure:

- *Step 1:* EA updates the set of candidate specification templates,
- *Step 2:* For each specification template, GP-UCB is used to find the parameters that optimize its discrimination between the sets of positive and negative examples.

Algorithm 2.3: Genetic algorithm to mine specifications [NSBB18]

Input: Set of positive examples $\mathcal{S}_p$, set of negative examples $\mathcal{S}_n$, fitness function F, discrimination function D, maximum number of iterations N

Output: STL formula discriminating $\mathcal{S}_p$ from $\mathcal{S}_n$

```
1  Ψ ←InitializeTemplateSet
2  Φ ← ParameterInference(Ψ, D, S_p, S_n)
3  for i = 1, ..., N do
4      Φ_s ←Sample(Φ, F, S_p, S_n)
5      Ψ ← Crossover(Φ_s) or Mutation(Φ_s)
6      Φ_s ← ParameterInference(Ψ, D, S_p, S_n)
7      Φ ←Sample(Φ ∪ Φ_s, F, S_p, S_n)
8  end
9  φ ← SelectFormula(Φ)
10 return φ
```

These two steps require four main ingredients: (i) the fitness function, (ii) the discrimination function, (iii) the cross-over operation, and (iv) the mutation operator.

The fitness function is used to sample the formulas from the set of candidate templates that will generate new formulas or will be passed to the following iteration (function *Sample* in lines 4 and 7, respectively). This function is defined as the sum of two terms: one that penalizes long-size formulas and the other that favors formulas with higher discrimination power. The latter is defined as a *discrimination* function that captures how well an STL formula discriminates between two sets of examples. It assigns to the formula the difference between the average robustness of positive and the average robustness of negative examples, divided by the sum of the respective standard deviations. This is the

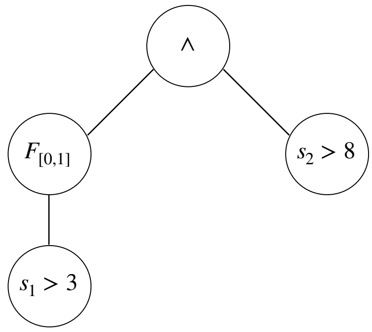


Figure 2.11: Syntax tree of the formula $\varphi = \mathbf{F}_{[0,1]}(s_1 > 3) \wedge (s_2 > 8)$.

function that GB-UCP aims to maximize in order to find the optimal parameters for each formula template (function *ParameterInference* in lines 2 and 6).

Considering the representation of a formula as a syntax tree (see an example in Figure 2.11), the cross-over operator works as follows: given two formulas (the parents), it randomly selects one subtree from each formula and switches them, generating two new formulas (the children). Finally, the mutation operator takes as input a formula and it changes one random node with one random formula (either predicates or temporal operators **F**, **G** and **U** applied to predicates). In every iteration, the cross-over operator and the mutation operator are mutually exclusive and randomly selected according to their predefined probabilities (line 5).

After a maximum number of iterations, the algorithm returns the formula in the set of candidate specifications that have the maximum discrimination power according to the discrimination function (function *SelectFormula* in line 9).

2.5 Related Work

This tutorial focuses on revisiting methods for learning formal specifications in Signal Temporal Logic [MN04] (STL). However, there are other approaches to learn different formal specification languages and models. In the following, we briefly report some of them. For a more general overview we refer the reader to a recent interesting survey [WCTZ20].

Multi-valued specification languages STL with quantitative semantics is a multi-valued logic assuming values in $\mathbb{R} \cup \{-\infty, \infty\}$ and allowing to measure how far is a behavior from satisfying or violating a specification. There are other branches of multi-valued logic such as fuzzy logic [Zad88] and probabilistic logic [Nil86]. In contrast to STL, fuzzy logic is normalized to the interval $[0, 1]$ of truth values and aims to represent vagueness and imprecise information. Probabilistic logic allows to reason with uncertain knowledge by having the truth values of sentences be probabilities between 0 and 1.

Learning Invariants Temporal invariants are properties that remain unchanged during the system's executions. In STL such property would correspond to learn the template $\mathbf{G}\varphi$, where φ is a constraint expressed over the system's variables (for example $x \leq y + 1$) that the system will always satisfy. A popular tool to learn such invariants is DAIKON [EPG+07] and it is employed in CPSDEBUG [BMM+21] to support failure explanation in CPS models.

Learning LTL LTL describes temporal properties of reactive systems. Texada [LPB15] is a popular approach for learning LTL specifications from temporal templates by instantiating the template with the propositions that are consistent with the observed data. Huang and Cleaveland [HC20] propose an automata-based approach for inferring LTL formula from a specification template and a set of data streams. Neider and Gavran [NG18] introduce two passive approaches for learning LTL - the first one consists of a sequence of satisfiability problems in propositional Boolean logic that generate the smallest LTL specification that is consistent with the observations, while the second one combines the SAT-driven learning with learning decision trees. The second approach allows to scale to large-size scenarios while dropping the minimality requirement. Riener [Rie19] improves on the SAT-based learning method by using topology structures to partition the search space into small sub-problems. Gaglione et al. [GNR+21] extend the LTL mining method from Neider and Gavran by enabling inference of concise LTL specifications in the presence of noisy data by using MaxSMT.

Learning state-machines State-machine models such as finite state automata provide an alternative approach to express the system's behavior. They consist of a finite set of states and a transition function determining how to switch from a state to another given a certain input symbol. Dana Angluin proposed in [Ang87] the first learning method for deterministic finite automata called L* algorithm. This algorithm can be formulated as a cooperative game between two players: a learner and a teacher. The aim of the learner is to infer, by formulating some queries and reasoning about the obtained answers, a deterministic finite automaton modelling an unknown language L. The teacher acts instead as an oracle that can answer queries formulated by the learner. There are generally two types of queries: *membership* and *equivalence* queries. The first type of queries consists in asking whether a certain word belongs to the unknown language, while the second type of queries asks whether a candidate language is equivalent to the unknown language. The family of approaches based on this interaction between a learner and a teacher are also called *active learning methods* [SHM11, HS18, WCTZ20]. The main prerequisite of these methods is the existence of a teacher, but this is not always feasible. An alternative approach that does not require a teacher is given in ALERGIA algorithm [CO94] where a prefix tree acceptor is built starting from a set of observed execution traces. Approaches that do not require the interaction with a teacher are called *passive learning methods* [WCTZ20]. More recently, many of these concepts have been further extended to learn more expressive and complex state machine models such as timed automata [PMM17], linear hybrid automata [SHSZ19, MRBF15], and hybrid automata with inputs and outputs [MRBF15].

Learning Regular Expressions *Regular expressions* describe the regular languages accepted by deterministic finite automata and can be used to specify in a concise way patterns of symbols. A regular expression can be automatically generated from a deterministic finite automata using Kleene's algorithm [HU79, Kle51]. A method for mining timed regular expressions [ACM02], from traces was developed in [NCJF18, NF19]. *Shape expressions* [NQF$^+$21] are regular expressions where the symbols are parametrized functions (linear, exponential, sinosuoidal, etc) describing shapes that characterize the features of a signal. Recent works [BDG$^+$20, BDM$^+$21] provide an automated approach to learn linear shape expressions from a set of positive examples and it is supported by ShapeIt [BDM$^+$21].

Learning model checking Learning model checking is a new interesting framework introduced in [BGKN22] that consists in predicting the probability of satisfaction (or the robustness value) of an STL specification with respect to a model M, given the satisfaction values of a set of independently selected STL properties. The key insight is to learn a linear model that predicts the satisfaction values of STL formulas in a feature space, namely a higher dimensional space than the original space of data and where it is easier to work on. As commonly used in machine learning, the kernel trick is exploited to perform an efficient learning that does not require the explicit computation of the feature space: it is indeed sufficient to use the similarity between pairs of STL formulas. The similarity between two formulas is defined as the scalar product of the functions that compute the robustness values of the formulas by varying the trajectories. The authors mention the possibility of using their new approach for specification mining purposes: the search would then be shifted from the discrete space of formula templates to a continuous space where distances depend on semantic similarities.

Data Mining in Temporal Series Mining patterns in temporal series has been the subject of intense and prolific research activities for at least three decades. As discussed in [KK03] the main tasks in temporal series data mining are *indexing*, *clustering*, *classification* and *segmentation*. *Indexing* [KCMP01] consists in matching times series in a database that are the closest according to a similarity measure to a given query time series. *Clustering* [KCMP01, KL88] is a task that groups times series according to a similarity measure, while *classification* [KCMP01, KL88] is a supervised machine learning task that takes in input an unlabeled time series and determines one or more classes to which the time series belongs. Finally the goal of *segmentation* [KP98, SZ96] is to compute from an initial time series, another time series with less data points approximating the original one. This last task is general functional to efficiently perform the other three. Mining STL specifications can be considered as *clustering* and *classification* tasks, while monitoring an STL formula over time series is similar to the *indexing* task. However, one main difference is that we can leverage a data-efficient, concise and human-understandable symbolic language to represent the sets of time series we are interested in. This provides a suitable tool not only to classify/cluster time series, but also to explain the temporal relations characterizing the essence of the desired time series pattern. Furthermore, the use of STL opens to the possibility of using all the machinery available for monitoring, testing, and control synthesis.

2.6 Conclusion and Future Directions

This paper evidences the active and vivid research in the domain of mining STL specifications. This field connects the worlds of machine learning and formal methods, exploiting many potential synergies between the two. We observe that both template-based and template-free mining approaches have relevant practical applications and receive considerable attention from the research community. Despite many different angles used to tackle this problem, we see several unexplored opportunities for research, that we briefly discuss in this section.

Model-based active mining We notice that only one reviewed approach uses the system model in the specification mining process. We think that this is an interesting and not sufficiently explored direction to pursue, especially when using specification mining to support the CPS design. More specifically, the model can be used to generate new behaviors and thus actively refine the mined specification, providing in every iteration a more precise insight in the model properties.

Mining input/output relations All the approaches considered in this paper assume that the outcome is a flat specification that characterizes a language. In CPS, the systems are dominantly *open* and have a clear input and output signature – the system continuously receives inputs and reacts to them, generating outputs. For many applications, mining an input/output relation would be a very valuable outcome.

Mining local patterns The methods reviewed in this paper focus on learning specifications from the entire signals. A different view on the mining problem consists in finding repeating anomalous patterns within a single behavior. This more local mining problem has been extensively studied in the time series community with the ideas of shapelets [UBK15] and motifs [YZU$^+$16] but has received no attention so far in the context of STL.

Explainable AI Specification mining is a form of explainable artificial intelligence, since a formal specification provides a-priori an interpretable artefact. However, this corner on learning specifications has not been sufficiently studied, and there is little work that investigates, for instance, whether STL specifications can provide meaningful additional insight into opaque black-box components, such as deep neural networks.

Acknowledgements This work has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 956123 and it is partially funded by the TU Wien-funded Doctoral College for SecInt: Secure and Intelligent Human-Centric Digital Technologies. The authors acknowledge TU Wien Bibliothek for financial support through its Open Access Funding Programme.

CHAPTER 3

Mining Specification Parameters for Multi-Class Classification

Reference. The content of this chapter has been published in [ABM+23]: *Edgar A. Aguilar, Ezio Bartocci, Cristinel Mateis, Eleonora Nesterini, Dejan Ničković. 2023.* Mining Specification Parameters for Multi-class Classification. *In: Katsaros, P., Nenzi, L. (eds) Runtime Verification. RV 2023. Lecture Notes in Computer Science, vol 14245. Springer, Cham.* https://doi.org/10.1007/978-3-031-44267-4_5

Abstract. We present a method for mining parameters of temporal specifications for signal classification. Given a parametric formula and a set of labeled traces, we find one parameter valuation for each class and use it to instantiate the specification template. The resulting formula characterizes the signals in a class by discriminating them from signals of other classes. We propose a two-step approach: first, for each class, we approximate its validity domain, which is the region of the valuations that render the formula satisfied. Second, we select from each validity domain the valuation that maximizes the distance from the validity domain of other classes. We provide a statistical guarantee that the selected parameter valuation is at a bounded distance from being optimal. Finally, we validate our approach on three case studies from different application domains.

3.1 Introduction

Formal specifications are crucial during system design, verification, and operation. They allow a precise and unambiguous exchange of requirements between engineering teams, enable rigorous verification of critical system components, and translate into test oracles and runtime monitors during system execution.

Cyber-physical systems (CPS) include physical parts whose properties are only partially known at design time. Generally, an engineer may expect a system to act a certain way without knowing the exact numerical values that characterize the system's behavior. For instance, an engineer may know that the system response to a step input is an overshoot while being unaware of its delay from the input or its amplitude. We can express this partial knowledge about the intended property using a parameterized specification template. Characterizing the actual parameters is a challenging task that requires the help of an automated tool.

Related work. Specification mining [BMNN22] is an emerging area of research in the field of CPS motivated by the need to analyze and explain the massive amount of data gathered from these systems while interacting with the environment. The vast majority of the available approaches employ Signal Temporal Logic (STL) [MN04] as the formal specification language for mining requirements in CPS [XJ18, BGKN22]. Parametric Signal Temporal Logic (PSTL) [ADMN11] is a parametric extension of STL in which parameters replace both threshold constants in numerical predicates and time bounds in temporal operators. Some works address the problem of learning the optimal parameters for a given candidate template formula [ADMN11, HDF18, BBS14a], while others learn both the structure and the parameters [JTS+19,NSBB18,BBS+14b,BGKN22]. These approaches focus on learning an STL formula using only positive examples of trajectories [JTS+19] or positive and negative examples [BBS14a, NSBB18].

In this paper, we broaden the applicability of STL specification mining to the multi-class classification problem: we aim to find the set of parameters for given PSTL template formulas that can discriminate multiple classes of trajectories. We assume a set of labeled traces and characterize each class of traces with an STL formula. A final classifier leverages STL quantitative semantics to evaluate how robustly a trace satisfies/violates each specification and consequently predicts the trace's label. Recently, also other papers [MDP+20b, LTLT22] considered a similar task. However, both works limit the space of mined specifications to STL fragments, reducing the resulting expressiveness. Moreover, they are both template-free methods; namely, they mine both the formula template and its numerical predicates. Such an approach is a double-edged sword: on the one hand, template-free methods do not require prior knowledge, but, on the other hand, they do not allow embedding information in the procedure and thus steer the search toward specific properties that are relevant in a particular application. Conversely, we aim to tackle the complementary task of mining parameters without any restrictions on STL grammar.

Our contribution. In this paper, we propose a family of methods for mining PSTL parameters for multi-class classification. We assume a PSTL template associated with each labeled class of signals and choose the parameters in each template to maximize discrimination from the signals with the other labels. We devise a two-step approach to this problem. For each labeled class of signals, we first approximate two validity domains, one for the signals belonging to that class and one for all other signals. The *validity domain* is the region of the parameter space that instantiates satisfied specifications from

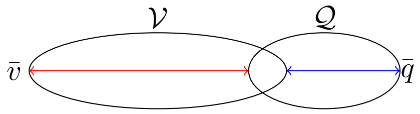


Figure 3.1: The Hausdorff distance D_H between sets $\mathcal{V}$ and $\mathcal{Q}$ is defined as: $D_H(\mathcal{V}, \mathcal{Q}) := \max\{D(\mathcal{V}, \mathcal{Q}), D(\mathcal{Q}, \mathcal{V})\}$. In the above example, $D(\mathcal{V}, \mathcal{Q}) = d(\bar{v}, \mathcal{Q})$, $D(\mathcal{Q}, \mathcal{V}) = d(\bar{q}, \mathcal{V})$, and $D_H(\mathcal{V}, \mathcal{Q}) = D(\mathcal{V}, \mathcal{Q})$.

a given template. In the second step, we select a parameter instantiation within the first validity domain and maximize its distance from the other one. We return both the set of instantiated STL specifications (one for each class) and a classifier that combines them. The former helps gain insights into what discriminates one class from the others. At the same time, the latter is essential to performing the classification task and, in particular, to classifying new unlabeled traces.

One of the main advantages of the proposed methods is that they do not require any assumptions regarding the monotonicity of parameters. Monotonicity is a common restriction in the specification mining literature because it significantly simplifies the geometry of the resulting validity domains and thus leads to a much more tractable problem. Additionally, we show that our selection of parameter valuations has a statistical guarantee of being optimal. We implement our approach in a publicly available prototype tool and evaluate it on three case studies from different application domains.

3.2 Preliminaries

A *time series* $\boldsymbol{x} = (t_1, x_1), (t_2, x_2), \ldots, (t_l, x_l)$ is a finite sequence of (time, value) pairs, where $t_1 = 0$, $t_i < t_{i+1}$ for every $i \in \{1, \ldots, l-1\}$, and $x_i \in \mathbb{R}^k$ is a vector of real values. We denote by $|\boldsymbol{x}| = l$ the length of $\boldsymbol{x}$. We will abuse notation, and also denote by $\boldsymbol{x}$ the signal $\mathbb{R}_{\geq 0} \to \mathbb{R}^k$, such that $\boldsymbol{x}(t) = x_i$ if $t \in [t_i, t_{i+1})$ with $i < l$, or $\boldsymbol{x}(t) = x_l$ if $t \geq t_l$.

The distance between a point $\boldsymbol{v} \in \mathbb{R}^k$ and a (non-empty) set $\mathcal{Q} \subset \mathbb{R}^k$ is defined as:

$$d(\boldsymbol{v}, \mathcal{Q}) = \inf\{\|\boldsymbol{v} - \boldsymbol{q}\| \mid \boldsymbol{q} \in \mathcal{Q}\}, \tag{3.1}$$

where $\|\cdot\|$ is the Euclidean norm in $\mathbb{R}^k$. To compute the distance from one set to another, we consider the so-called *directed Hausdorff distance* defined by:

$$D(\mathcal{V}, \mathcal{Q}) = \sup\{d(\boldsymbol{v}, \mathcal{Q}) \mid \boldsymbol{v} \in \mathcal{V}\}. \tag{3.2}$$

We observe that function D is not a proper distance because it is not necessarily symmetric. Intuitively, the function computes the longest distance from a point in $\mathcal{V}$ and its closest point in $\mathcal{Q}$. Fig. 3.1 depicts how function D works and how it is related to the Hausdorff distance.

Signal Temporal Logic (STL) [MN04] is a specification language used to express continuous temporal properties over real-valued signals. The syntax of STL is given by the grammar:

$$\varphi := \mathbf{true} \mid f(\boldsymbol{x}) > 0 \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \, \mathbf{U}_I \, \varphi_2$$

where **true** is the Boolean true constant, $f(\boldsymbol{x}) > 0$ is an atomic proposition with $f\colon \mathbb{R}^k \to \mathbb{R}$, $\neg$ is the Boolean negation, $\wedge$ is the Boolean conjunction, and $\mathbf{U}_I$ is the *until* operator defined over the interval I in $\mathbb{R}_{\geq 0}$. The interval I is generally omitted if $I = [0, \infty)$. The *finally* **F** and *globally* **G** operators can be derived as follows: $\mathbf{F}_I\varphi := \mathbf{true}\,\mathbf{U}_I\varphi$ and $\mathbf{G}_I\varphi := \neg\mathbf{F}_I\neg\varphi$.

The quantitative semantics of STL [DFM13] is defined in terms of the *robustness function* ρ that has as arguments: the STL specification φ, the signal $\boldsymbol{x}$ and the time t. The value $\rho(\varphi, \boldsymbol{x}, t)$ is defined inductively as follows:

$$\begin{aligned}
\rho(\mathbf{true}, \boldsymbol{x}, t) &= +\infty \\
\rho(f(\boldsymbol{x}) > 0, \boldsymbol{x}, t) &= f(\boldsymbol{x}(t)) \\
\rho(\neg\varphi, \boldsymbol{x}, t) &= -\rho(\varphi, \boldsymbol{x}, t) \\
\rho(\varphi_1 \wedge \varphi_2, \boldsymbol{x}, t) &= \min\left(\rho(\varphi_1, \boldsymbol{x}, t), \rho(\varphi_2, \boldsymbol{x}, t)\right) \\
\rho(\varphi_1 \, \mathbf{U}_I \, \varphi_2, \boldsymbol{x}, t) &= \sup_{t' \in t \oplus I} \Big(\min\big(\rho(\varphi_2, \boldsymbol{x}, t'), \inf_{t'' \in (t, t')} (\rho(\varphi_1, \boldsymbol{x}, t''))\big)\Big)
\end{aligned}$$

where $\oplus$ denotes the Minkowski sum. We indicate $\rho(\varphi, \boldsymbol{x}, 0)$ by $\rho(\varphi; \boldsymbol{x})$. The STL quantitative semantics is *sound*, i.e., when $\rho(\varphi, \boldsymbol{x}, t) > 0$ the signal $\boldsymbol{x}$ satisfies φ, while when $\rho(\varphi, \boldsymbol{x}, t) < 0$ it violates it.

A *PSTL template* is an STL formula where some temporal or magnitude values are replaced by parameter symbols [ADMN11]. If m is the number of different parameter symbols in a PSTL formula, we indicate by $P = \{p_1, \ldots, p_m\}$ the set of parameter symbols and with $\mathcal{W}$ the *parameter space*, that is the set of values that parameter symbols may assume. In our case, $\mathcal{W}$ is a closed hyperbox in $\mathbb{R}^m$:

$$\mathcal{W} = \prod_{i=1}^{m} [a_i, b_i], \tag{3.3}$$

where, for every $i \in \{1, \ldots, m\}$, a_i and b_i are scalar values in $\mathbb{R}$ and $b_i > a_i$. A *parameter valuation* $\boldsymbol{v} \in \mathcal{W}$ maps a PSTL formula φ into the STL formula $\varphi_{\boldsymbol{v}}$ obtained from φ by replacing its parameter symbols with $\boldsymbol{v}$. For example, given the template $\varphi = \mathbf{G}_{[T_1, T_2]}(x \geq k)$, where T_1 and T_2 are temporal parameters and k is a magnitude parameter, if $\boldsymbol{v} = (0, 2, 3.5)$, then the resulting STL formula $\varphi_{\boldsymbol{v}}$ is $\varphi_{\boldsymbol{v}} = \mathbf{G}_{[0,2]}(x \geq 3.5)$. With a little abuse of notation, we will occasionally write that a signal $\boldsymbol{x}$ satisfies/violates a parameter valuation $\boldsymbol{v}$ to mean that $\boldsymbol{x}$ satisfies/violates the specification $\varphi_{\boldsymbol{v}}$.

Given a signal $\boldsymbol{x}$ and a PSTL formula φ, the *validity domain* $\mathcal{V}(\boldsymbol{x}, \varphi) \subseteq \mathcal{W}$ is defined as the set of all parameter valuations $\boldsymbol{v}$ that generate STL formulas $\varphi_{\boldsymbol{v}}$ satisfied by the signal $\boldsymbol{x}$. In the example, if the signal $\boldsymbol{x}$ satisfies $\varphi_{\boldsymbol{v}} = \mathbf{G}_{[0,2]}(x \geq 3.5)$, then $\boldsymbol{v} \in \mathcal{V}(\boldsymbol{x}, \varphi)$.

The validity domain of a finite set $\mathcal{S}$ of signals corresponds to the intersection of all validity domains of signals in the set $\mathcal{S}$. In other words, $\boldsymbol{v} \in \mathcal{V}(\mathcal{S}, \varphi)$ if and only if $\varphi_{\boldsymbol{v}}$ is satisfied by all signals in $\mathcal{S}$.

3.3 Problem Description

Hypotheses. We assume that two classes of time-series are given: $\mathcal{S}_1$ and $\mathcal{S}_2$. The restriction to two classes is for the sake of a simpler description: in Section 3.4.3, we show how the proposed method is extended to an arbitrary number of classes. In addition, the user provides a PSTL template for each class of signals. We first assume the template φ is the same for both classes and then, in Section 3.4.2, we present the generalization to different templates. Finally, we assume that the lower/upper bounds on parameter values are available, so the values of a_i and b_i in Eq. (3.3) are known for all $i = 1, \ldots, m$.

Goal. Our goal is to discriminate classes $\mathcal{S}_1$ and $\mathcal{S}_2$ using formula φ. To do so, we need to find two parameters valuations $\boldsymbol{v}_1^*$ and $\boldsymbol{v}_2^*$ for φ such that they characterize the two sets of time series and *best* discriminate between them, where *best* refers to the following optimality criteria. We denote by $\mathcal{V}_i$ the validity domain $\mathcal{V}(\mathcal{S}_i, \varphi)$ of class $\mathcal{S}_i$ with respect to φ whenever φ is clear from the context.

Definition 1. An *optimal* parameter valuation $\boldsymbol{v}_1^*$ for $\mathcal{S}_1$ with respect to $\mathcal{S}_2$ is such that (i) $\boldsymbol{v}_1^* \in \mathcal{V}_1$, and (ii) $d(\boldsymbol{v}, \mathcal{V}_2) \leq d(\boldsymbol{v}_1^*, \mathcal{V}_2)$ for all $\boldsymbol{v} \in \mathcal{V}_1$, where d is the point-set distance defined in (3.1).

The first condition ensures that all time series in $\mathcal{S}_1$ satisfy the STL formula $\varphi_{\boldsymbol{v}_1^*}$, while the second one forces $\boldsymbol{v}_1^*$ to be the point in $\mathcal{V}_1$ that is the *furthest* from the other class satisfaction region. In other words, the first condition makes $\varphi_{\boldsymbol{v}_1^*}$ a characterization of time series in $\mathcal{S}_1$, while the second condition renders $\varphi_{\boldsymbol{v}_1^*}$ one of the *best* instantiations of φ for discriminating $\mathcal{S}_1$ from $\mathcal{S}_2$ (the optimal point is not necessarily unique). The definition of $\boldsymbol{v}_2^*$ for $\mathcal{S}_2$ is analogous.

In some extreme cases, we observe that the given definition of an optimal point is meaningless because the Hausdorff distance between the two validity domains is zero (for example, if the two validity domains are identical or one validity domain is a subset of the other). This eventuality may happen when the template φ is not suitable for discriminating the two classes; in this case, there is no choice of the parameter valuations that would help with the classification. The only reasonable possibility is that the user changes the PSTL template. Section 3.5 shows an example of this instance.

Once our procedure identifies the parameter valuations, we need to perform the classification. We remind that the validity domain of a class is the intersection of the validity domains of all signals belonging to that class. It follows that a signal $\boldsymbol{x}$ belonging to $\mathcal{S}_2$ does not necessarily violate $\varphi_{\boldsymbol{v}}$ with $\boldsymbol{v} \in \mathcal{V}_1 \setminus \mathcal{V}_2$. As a consequence, it may happen that $\boldsymbol{x}$ satisfies both $\varphi_{\boldsymbol{v}_1^*}$ and $\varphi_{\boldsymbol{v}_2^*}$. Hence, we cannot simply use the Boolean satisfaction of $\varphi_{\boldsymbol{v}_1^*}$ and $\varphi_{\boldsymbol{v}_2^*}$ to predict the signal's label. For this reason, we use the quantitative semantics of STL and classify a signal $\boldsymbol{x}$ in class 1 if $\rho(\varphi_{\boldsymbol{v}_1^*};\boldsymbol{x}) \geq \rho(\varphi_{\boldsymbol{v}_2^*};\boldsymbol{x})$ and in class 2 otherwise, being ρ the robustness function defined in Section 3.2.

3.4 Our Approach

Algorithm 3.1: Validity domain approximation with *FixedBin*

Input: PSTL template φ, set of signals $\mathcal{S}$, parameter space $\mathcal{W}$, number of trials N
Output: Approximated validity domain $\hat{\mathcal{V}}$

$\hat{\mathcal{V}} \leftarrow \emptyset$
$\mathcal{C} \leftarrow$ cell_partition$(\mathcal{W})$
foreach $C \in \mathcal{C}$ **do**
    for $i = 1, \ldots, N$ **do**
        $\boldsymbol{v} \leftarrow$ uniform(C)
        flag $\leftarrow$ true
        foreach $\boldsymbol{x} \in \mathcal{S}$ **do**
            if $\boldsymbol{x} \not\models \varphi_{\boldsymbol{v}}$ **then**
                flag $\leftarrow$ false
                break
            end
        end
        if flag **then**
            $\hat{\mathcal{V}} \leftarrow \hat{\mathcal{V}} \cup C$
            break
        end
    end
end
return $\hat{\mathcal{V}}$

Algorithm 3.2: Validity domain approximation using *AdaBin*.

Input: PSTL template φ, set of signals $\mathcal{S}$, cell C, approx. validity domain $\hat{\mathcal{V}}$, # trials N, threshold on the concentration of satisfied parameter valuation h
Output: New approximation for $\hat{\mathcal{V}}$

counter $\leftarrow 0$
for $i = 1, \ldots, N$ **do**
    $\boldsymbol{v} \leftarrow$ uniform(C)
    flag $\leftarrow$ true
    foreach $\boldsymbol{x} \in \mathcal{S}$ **do**
        if $\boldsymbol{x} \not\models \varphi_{\boldsymbol{v}}$ **then**
            flag $\leftarrow$ false
            break
        end
    end
    if flag **then** counter $\leftarrow$ counter $+1$
end
if $\frac{\text{counter}}{N} \geq h$ **then return** $\hat{\mathcal{V}} \cup C$
else if counter $= 0$ **then return** $\hat{\mathcal{V}}$
else if size$(C) = \boldsymbol{g}$ **then return** $\hat{\mathcal{V}} \cup C$
else
    $C_1, \ldots, C_{2^m} \leftarrow$split$(C)$
    for $j = 1, \ldots, 2^m$ **do**
        $\hat{\mathcal{V}} \leftarrow$ Algorithm3.2$(\varphi, \mathcal{S}, C_j, \hat{\mathcal{V}}, N, h)$
    end
end
return $\hat{\mathcal{V}}$

We propose a two-step approach to address the problem of Section 3.3: for each class, we compute (i) an over-approximation $\hat{\mathcal{V}}$ of its validity domain $\mathcal{V}$, and (ii) an approximation of the *optimal* point in the validity domain. For the approximation of the validity domains, we divide the parameter space $\mathcal{W}$ into a lattice of cells. We present two methods: one in which the cell resolution is fixed and one in which it is adaptive. We first consider the case in which the same PSTL template is associated with the two different classes of signals (Sections 3.4.1). Then, we extend our approach to allow two different PSTL templates (Section 3.4.2) and an arbitrary number of classes (Section 3.4.3). The proofs are shown in the Appendix.

3.4.1 Binary Satisfaction

Fixed Granularity with Binary Satisfaction (FixedBin). We first describe how we compute the approximation $\hat{\mathcal{V}}$ of a validity domain $\mathcal{V}$ (Algorithm 3.1). The parameter space $\mathcal{W}$ is divided into a partition $\mathcal{C}$ of axis-aligned closed-hyperboxes having all the same size that we call *cells* (line 2). Each cell is then evaluated separately, and included in the over-approximation $\hat{\mathcal{V}}$ only if a parameter valuation $\boldsymbol{v}$ inside the cell is found such that the STL formula $\varphi_{\boldsymbol{v}}$ is satisfied by all time series in the class under study. If, after N trials, no valid parameter valuation is found, then the cell is not included in $\hat{\mathcal{V}}$.

We observe that $\hat{\mathcal{V}}$ is intended to be an over-approximation of $\mathcal{V}$. However, it is possible that a cell $C \subseteq \mathcal{W}$ is not included in $\hat{\mathcal{V}}$ even though there exists a valuation $\boldsymbol{v} \in C$ in the cell that is also included in $\mathcal{V}$. This can happen because we decide whether C shall be included in $\hat{\mathcal{V}}$ based on a finite number N of samples in C. It follows that the choice of N in Algorithm 3.1 affects the likelihood of a cell that has non-empty intersection with the true validity domain to be included in the over-approximation, as stated by Lemma 1.

Lemma 1 (Choice of N)**.** *Let $\mathcal{V}$ be a validity domain and C a cell not included in the (intended) over-approximation $\hat{\mathcal{V}}$ computed with N random samples per cell. If $N \geq \log_{1-p_0}(\alpha)$ with p_0, $\alpha \in (0,1)$, then the fraction of points in C that are contained in $\mathcal{V}$ is smaller than p_0 with confidence $1-\alpha$.*

As a concrete example, let us set $p_0 = 0.01$ and $\alpha = 0.05$. In this setting, we need to examine $N \geq \log_{(1-p_0)} \alpha = \log_{0.99}(0.05) = 298$ parameter valuations to conclude with confidence 95% that the proportion of satisfied parameter valuations in the cell is less than 1%.

Once we have computed both $\hat{\mathcal{V}}_1$ and $\hat{\mathcal{V}}_2$, we select the point $\hat{\boldsymbol{v}}_1$ in $\hat{\mathcal{V}}_1$ such that $\hat{\boldsymbol{v}}_1$ and $\hat{\mathcal{V}}_2$ have maximum distance. In other words, we select $\hat{\boldsymbol{v}}_1$ such that $d(\hat{\boldsymbol{v}}_1, \hat{\mathcal{V}}_2) = D(\hat{\mathcal{V}}_1, \hat{\mathcal{V}}_2)$, where the functions d and D are defined in (3.1) and (3.2), respectively. To prove the existence of such a point $\hat{\boldsymbol{v}}_1$, we need to show that $\hat{\mathcal{V}}_1$ is a compact set, so that the supremum in definition (3.2) is in fact a maximum. Since the cells used to approximate $\mathcal{V}_1$ are closed-hyperboxes, $\hat{\mathcal{V}}_1$ is also a closed set (it is finite union of closed sets). Moreover, $\hat{\mathcal{V}}_1$ is a bounded set because it is a subset of the parameter space $\mathcal{W}$ that is bounded by hypothesis. Finally, being a closed and bounded set in $\mathbb{R}^m$, $\hat{\mathcal{V}}_1$ is compact. The same reasoning is applied to the choice of $\hat{\boldsymbol{v}}_2$ in $\hat{\mathcal{V}}_2$.

The following theorem relates the selected parameter valuation to an optimal one, showing that the difference of their respective distances to the other class validity domain is upper-bounded by the cells' size. Hence, it can be arbitrarily reduced.

Theorem 1 (Optimality)**.** *Let* $i \in \{1,2\}$*,* $\mathcal{V}_i$ *be the validity domain for class* $\mathcal{S}_i$*, and* $\hat{\mathcal{V}}_i$ *be the approximation of* $\mathcal{V}_i$ *using* $N \geq \log_{1-p_0}(\alpha)$ *random samples per cell and the cell resolution given by vector* $\boldsymbol{g} \in \mathbb{R}^m_{\geq 0}$*. Let* $\boldsymbol{v}_i^*$ *be an optimal parameter valuation for class* i *and* $\hat{\boldsymbol{v}}_i$ *the parameter valuation found with FixedBin. Then, the difference of the distances from* $\hat{\boldsymbol{v}}_i$ *and from* $\boldsymbol{v}_i^*$ *to the other class validity domain is bounded by the cell resolution:*

$$|d(\hat{\boldsymbol{v}}_i, \mathcal{V}_{(i \, mod \, 2)+1}) - d(\boldsymbol{v}_i^*, \mathcal{V}_{(i \, mod \, 2)+1})| \leq \|\boldsymbol{g}\| \tag{3.4}$$

with probability $1 - p_0$ *and confidence* $1 - \alpha$*.*

In practice, the values of p_0 and α are chosen freely by the user, affecting the number of random samples N required to guarantee (3.4): the smaller p_0 and α are, the greater N must be.

Finally, the complexity of the algorithm is reported in the following theorem.

Theorem 2 (Complexity)**.** *Let* n *be the total number of signals and let* m *be the number of different parameter symbols in the PSTL template* φ*. Let* $\boldsymbol{g} \in \mathbb{R}^m_{\geq 0}$ *be the vector that contains the minimal sizes of the cells and let the parameter space be* $\mathcal{W} = \prod_{i=1}^m [a_i, b_i]$ *as in Eq.* (3.3)*. Then, the total number* L *of monitors is exponential in* m *and linear in* n*. In particular,*

$$L \leq \left(\max_{i=1,\ldots,m} \left\lceil \frac{b_i - a_i}{g_i} \right\rceil \right)^m \cdot (N \cdot n) = \mathcal{O}(M^m) \cdot \mathcal{O}(n),$$

where N *is the maximum number of parameter valuations to be evaluated in every cell and* M *is a suitable positive number.*

Adaptive Granularity with Binary Satisfaction (AdaBin). We present an alternative to compute over-approximations of validity domains using an adaptive cell resolution that exploits binary search (Algorithm 3.2).

The algorithm starts considering an initial empty approximation of the validity domain ($\hat{\mathcal{V}} \leftarrow \emptyset$), and the whole parameter space as a single cell ($C \leftarrow \mathcal{W}$). One cell at a time is then evaluated with three possible outcomes: (i) the cell is added to the approximation of the validity domain; (ii) the cell is not added to the approximation of the validity domain; (iii) the cell is split. The first two scenarios happen, respectively, when the cell presents a high concentration h of satisfied parameter valuations (line 13), or when no satisfied parameter valuations are found (line 14). Otherwise, reminding that m is the number of parameter symbols in the template φ, the cell is split into 2^m new cells generated by dividing each edge into two (at its middle point). When the minimal cell size is reached (line 15), we perform (i) if at least one satisfied parameter valuation is found. The minimal cell resolution $\boldsymbol{g}$ has to be reached by doing the same amount of splits in all parameter dimensions. Recalling the representation of parameter space $\mathcal{W}$ in Eq. (3.3), it follows that there must exist a natural number k such that

$$b_i - a_i = 2^k \cdot g_i \quad \forall i \in \{1, \ldots, m\}. \tag{3.5}$$

AdaBin is introduced to speed up the approximation of the validity domains by employing the finest cell resolution only in the most uncertain areas of the parameter space, such as those on the validity domains boundaries. On the other hand, *AdaBin* may be less accurate than *FixedBin*, and the guarantee expressed by Theorem 1 does not apply. However, we observe that a reduction in the accuracy of the approximation of the validity domain does not necessarily result in a decrease in the classification accuracy.

Monotonic variant. The *monotonic variant* of the previous methods is introduced to speed up the computation of the validity domain approximations in case the parametric STL template is monotonic with respect to all its parameters. A PSTL formula φ is *monotonically increasing* (or *decreasing*) with respect to its parameter p_i, if, for every pair of valuations w_i and w_i' such that $w_i \leq w_i'$ (or $w_i \geq w_i'$) and for every signal $\boldsymbol{x}$, $\rho(\varphi_{\boldsymbol{v},w_i};\boldsymbol{x}) \leq \rho(\varphi_{\boldsymbol{v},w_i'};\boldsymbol{x})$, where $\boldsymbol{v}$ corresponds to a generic valuation of the other parameters in φ.

For a PSTL formula φ that is monotonic with respect to all its parameters, we define the *lowest point* and the *highest point* of a cell C. The cells that we use to approximate the parameter space are closed hyperboxes in $\mathbb{R}^m$; hence, they are representable as $C = \prod_{i=1}^{m}[q_i, Q_i]$, where q_i, $Q_i \in \mathbb{R}$ and $q_i < Q_i$ for every $i \in \{1, \dots, m\}$. The *lowest point* in C is the point that is the most likely to be violated, namely the vector in $\mathbb{R}^m$ whose i-th component is q_i if φ is increasing with respect to parameter p_i, or Q_i if φ is decreasing with respect to p_i. Conversely, the *highest point* of a cell C is the point which is most likely to be satisfied, being the vector in $\mathbb{R}^m$ whose i-th component is Q_i if φ is increasing with respect to parameter p_i, or q_i if φ is decreasing with respect to p_i.

The following equivalences are the bases of the monotonic variant methods, allowing for a complete evaluation of a cell through the study of only these two special parameter valuations: (i) all parameter valuations in a cell are violated if and only if its *highest point* is violated, and (ii) all parameter valuations in a cell are satisfied if and only if its *lowest point* is satisfied.

Remark 1. *If the PSTL template is monotonic with respect to all its parameters, then* (3.4) *holds deterministically because the statistical guarantee in Lemma 1 is not needed for the evaluation of the cell if highest and lowest points are studied.*

3.4.2 Different templates

The following extension should be applied when the user wants to describe the two classes of signals with different PSTL templates. Let φ_1 represent the temporal behavior of time series in class $\mathcal{S}_1$ and φ_2 for class $\mathcal{S}_2$. The parameter valuation $\hat{\boldsymbol{v}}_1$ is chosen in $\hat{\mathcal{V}}(\mathcal{S}_1, \varphi_1)$ to maximize the distance from $\hat{\mathcal{V}}(\mathcal{S}_2, \varphi_1)$ and is used to instantiate the PSTL template φ_1 for the characterization of $\mathcal{S}_1$. Analogously, $\hat{\boldsymbol{v}}_2$ will instantiate φ_2 to characterize $\mathcal{S}_2$ by discriminating $\hat{\mathcal{V}}(\mathcal{S}_2, \varphi_2)$ from $\hat{\mathcal{V}}(\mathcal{S}_1, \varphi_2)$. In other words, the computation of $\hat{\boldsymbol{v}}_1$ and $\hat{\boldsymbol{v}}_2$ is carried out separately by applying one of the previous methods twice: once as if the only given PSTL formula were φ_1, and the other time as if the only given PSTL formula were φ_2.

3.4.3 Multiple classes

In this section, we describe the generalization of our approach to the more general problem where the number of classes of time series is $k > 2$. Let us suppose to have a dataset of time series labeled with k values. The symbol $\mathcal{S}_i$ represents the class of time series labeled by i, where $i \in \{1, 2, \ldots, k\}$. The validity domain (with respect to a given PSTL formula φ) is represented by $\mathcal{V}_i$. We approximate all $\mathcal{V}_i$ separately with one of the two proposed methods. Then, for every label $i \in \{1, \ldots, k\}$, the parameter valuation $\hat{\boldsymbol{v}}_i$ is chosen in $\hat{\mathcal{V}}_i$ to maximize the distance from the union set of the other validity domains. Therefore, $\hat{\boldsymbol{v}}_i = \operatorname{argmax}_{\boldsymbol{v} \in \hat{\mathcal{V}}_i} d\Big(\boldsymbol{v}, \bigcup_{\substack{j=1 \\ j \neq i}}^{k} \hat{\mathcal{V}}_j\Big)$, where d is the point-set distance defined in (3.1).

Finally, for the classification purpose, in analogy with the binary case, we classify a signal $\boldsymbol{x}$ into the class i such that $i = \operatorname{argmax}_{j=1,\ldots,k} \rho(\varphi_{\hat{\boldsymbol{v}}_j}; \boldsymbol{x})$.

3.5 Application

We implemented our approach in the publicly available tool MiniPaSTeL[1] (Mining Parameters for Signal Temporal Logic), leveraging the RTAMT tool [NY20] for monitoring STL formulas, and applied it to three case studies. We provide the scripts and the seeds to reproduce the results we present. We run the experiments on a MacBook Pro with 16 GB RAM and M1 processor. For each application, we report the set of mined STL specifications and the misclassification rate (MCR) achieved in the classification. Moreover, each case study focuses on seeking an answer to a research question:

RQ1 [Applicability] *Does the need for a PSTL template as input limit the applicability of our approach?* Our method requires the user to provide the PSTL template, but such a template may not be available or may be inadequate to discriminate between different classes. In such cases, we can interleave our approach with the human investigation to refine guesses for the PSTL template. The Aircraft Elevator case study illustrates how we can use the method to support the user in finding a suitable specification.

RQ2 [Performance Evaluation] *How does our method compare to state-of-the-art?* We compare our approach to existing ones on the Naval dataset showing that it is comparable to or outperforms the state-of-the-art in terms of MCR. Moreover, we study how the MCR and the execution time vary with the number of training samples in the dataset.

[1] https://github.com/eleonoranesterini/MiniPaSTeL

RQ3 [Explainability of the Results] *Can the user exploit the results to infer new insights into the traces?* The choice of parameter values in existing specification mining methods is generally carried out by applying off-the-shelf optimization techniques that minimize a given objective function without providing a general understanding of the values selection. Conversely, our method provides a visual understanding of how the parameters are selected by plotting (projections of) the validity domains. Thanks to that, it is possible to explain (unexpected) results, as shown in the Parking case study.

3.5.1 Aircraft Elevator

To answer **RQ1**, we consider the Simulink model of the Aircraft Elevator Control System [GM05], whose output signals report the position over time of the left elevator of the aircraft. We generated time series of two classes, $\mathcal{S}_1$ and $\mathcal{S}_2$, by varying the parameters of the system input function: Fig. 3.2a depicts one representative signal per class.

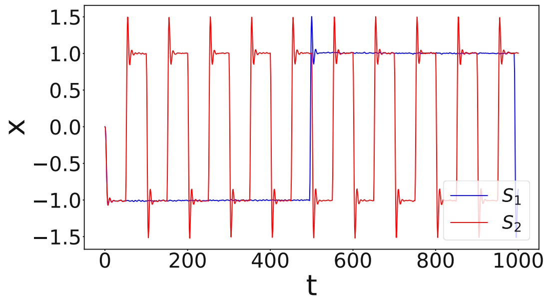
We explore the workflow in which an engineer uses our approach to explore manually crafted templates and find one that discriminates well between two classes of signals.

By analyzing the shape of the time series in Fig. 3.2a, we observe that the signals from both classes are characterized by oscillations around the same values, but with different frequencies. In particular, the red one has a higher frequency than the blue one. To leverage this difference, we may design a specification that imposes that signals oscillate around the higher (unknown) value p for a certain (unknown) amount of time T. For each class, we learn the values p and T. We expect the signals of the two classes stabilizing around the same value with the interval of time for the time series in $\mathcal{S}_1$ longer than for time series in $\mathcal{S}_2$. Thus, we consider the following PSTL template:

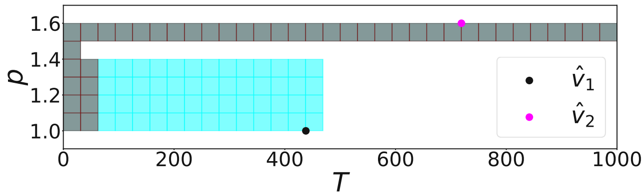
$$\varphi = \mathbf{G}\Big((x \geq p) \rightarrow (\mathbf{G}_{[0,T]}|x - p| < 0.5)\Big).$$

The formula φ expresses the behavior of signal x that remains "close" to the magnitude parameter p for T time units, immediately after the signal has become greater than p. We decided to manually set the notion of "closeness" to indicate a distance smaller than the value 0.5. This makes the validity domains 2-dimensional, giving the reader a better visual understanding of how the proposed methods work.

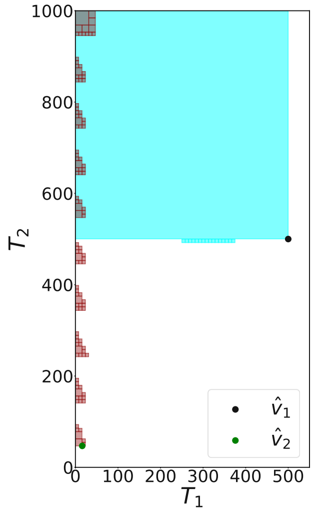
The parameter space is given by $\mathcal{W} = [0, 1000] \times [-1.6, 1.6]$. We impose the minimal sizes of cells in the parameter space to be $\boldsymbol{g} = (31.25, 0.1)$ to satisfy (3.5) for $k = 5$. We then apply *FixedBin* method to compute the parameter valuations $\hat{\boldsymbol{v}}_1$ and $\hat{\boldsymbol{v}}_2$. $\hat{\mathcal{V}}_1$, $\hat{\mathcal{V}}_2$, $\hat{\boldsymbol{v}}_1$, and $\hat{\boldsymbol{v}}_2$ are shown in Fig. 3.2b: $\hat{\mathcal{V}}_1$ is represented by the cyan region, $\hat{\mathcal{V}}_2$ by the red one (not visible because entirely subsumed by $\hat{\mathcal{V}}_1$), while the gray color represents their overlapping. The use of *AdaBin* method leads to the same approximations with the only difference that cells do not have all the same (minimal) size.



(a) The blue time series depicts one representative element in $\mathcal{S}_1$, the red one in $\mathcal{S}_2$.



(b) Approximations of validity domains for φ (cyan for $\mathcal{S}_1$ and red for $\mathcal{S}_2$) using *FixedBin*.



(c) Approximations of validity domains for ψ (cyan for $\mathcal{S}_1$ and red for $\mathcal{S}_2$) using *AdaBin*.

Figure 3.2: Time series and validity domains for the Aircraft Elevator case study.

The mined parameter valuations are $\hat{\boldsymbol{v}}_1 = (438, 1.0)$ and $\hat{\boldsymbol{v}}_2 = (719, 1.6)$ for $\mathcal{S}_1$ and $\mathcal{S}_2$, respectively, yielding the following specifications:

$$\mathcal{S}_1 : \varphi_{\hat{\boldsymbol{v}}_1} = \mathbf{G}((x \geq 1.0) \rightarrow (\mathbf{G}_{[0,438]}|x - 1.0| < 0.5))$$
$$\mathcal{S}_2 : \varphi_{\hat{\boldsymbol{v}}_2} = \mathbf{G}((x \geq 1.6) \rightarrow (\mathbf{G}_{[0,719]}|x - 1.6| < 0.5)).$$

We note that $\hat{\boldsymbol{v}}_2$ belongs not only to $\hat{\mathcal{V}}_2$, but also to $\hat{\mathcal{V}}_1$. From Fig. 3.2b, we observe that this is inevitable: since $\hat{\mathcal{V}}_2$ is a subset of $\hat{\mathcal{V}}_1$, every parameter valuation $\boldsymbol{v} \in \hat{\mathcal{V}}_2$ is also in $\hat{\mathcal{V}}_1$, resulting in $d(\boldsymbol{v}, \hat{\mathcal{V}}_1) = 0$. Hence, the chosen φ cannot be instantiated to discriminate $\mathcal{S}_2$ from $\mathcal{S}_1$. This observation suggests the PSTL formula should be changed to better discriminate between the two sets.

We then observe that φ requires the signal to remain close to p for at least T time units without specifying any condition about when the signal has to fall. Therefore, $\mathcal{V}_1$ includes $\mathcal{V}_2$ because the satisfaction of *high* values of T implies the satisfaction of *low* values of T. Hence, we change the PSTL formula φ by adding the requirement that the signal has to move away from the value around which it oscillates after T_2 time units. Furthermore, to remain in the 2-dimensional parameter space for visualization reasons, we replace the parameter value p with the scalar value 1.1. As a result, the new PSTL template we define is:

$$\psi = \mathbf{G}\Big((x \geq 1.1) \rightarrow (\mathbf{G}_{[0,T_1]}|x - 1.1| < 0.5 \land \mathbf{G}_{[T_2,T_2+T_1]}|x - 1.1| > 0.5)\Big),$$

where T_1 is just a different symbol to indicate the parameter T in φ. In this case, both parameters T_1 and T_2 are temporal parameters, so the parameter space becomes $\mathcal{W} = [0, 1000] \times [0, 1000]$.

The approximations of validity domains with respect to the new template ψ are shown in Fig. 3.2c (this plot is done with *AdaBin* to show what the approximation with adaptive cell size looks like). We see that no validity domain subsumes the other one any longer, so it is possible to find parameter valuations discriminating one set of time series from the other one. The mined values are $\hat{\boldsymbol{v}}_1 = (500, 500)$ and $\hat{\boldsymbol{v}}_2 = (16, 47)$.

By studying the performances of φ and ψ as classifiers, we confirm the outcomes of the manual analysis carried out with the visualization of the validity domains. As expected, φ performs very poorly, with a MCR of 0.5 because all signals in $\mathcal{S}_1$ are misclassified in class 2; conversely, ψ achieves a MCR of 0.

To summarize, template-based techniques are usually employed when an engineer expects a certain behavior from the system and wants to extrapolate its numerical values. Nevertheless, this case study shows how our approach can be used not only to mine the desired parameters but also to help the engineer refine the PSTL template, providing a visual understanding of the differences between two signal classes. This observation opens up to processes in which the algorithm's executions and human investigation are interleaved: the engineer steers the search while the algorithm refines it.

3.5.2 Naval Surveillance

We use the naval surveillance dataset [KJB17] to compare the performance of our methods to existing ones (**RQ2**). Since, to the best of our knowledge, there are no existing works focusing on parameter mining for STL to tackle the classification task, the only meaningful comparisons that are possible are with methods mining both the template and the parameters of the STL formula.

The dataset contains 2000 2-dimensional signals describing the evolution over time of the coordinates of ships approaching a harbor; each signal contains 61 sample points. Fig. 3.3 depicts the projection of some examples of the signal data set to the 2-dimensional spatial coordinates. The signals are partitioned into three different classes:

- *green* signals represent nominal behaviors in which ships arrive from the sea, get through the passage between the island and the peninsula, and reach the harbor.
- *red* signals describe anomalous trajectories of ships deviating to the island before heading towards the harbor (behavior relatable to human trafficking activities).
- *blue* signals depict ships that approach the passage between the island and the peninsula but that, at a certain point, turn around and go back to the open sea; this scenario could be connected to terrorist activities.

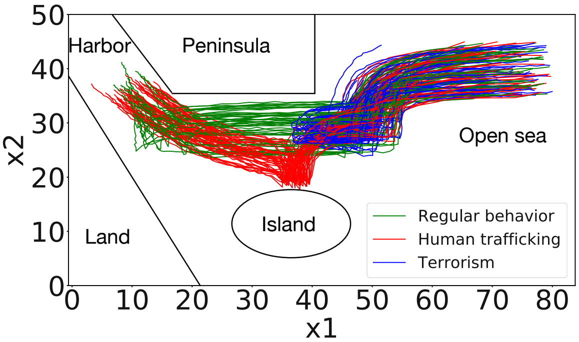


Figure 3.3: Traces representing ship trajectories. The Figure has been redrawn from [KJB17].

This dataset was mainly used for anomaly detection purposes, namely with the goal of discriminating the normal behavior from the two anomalous ones [KJB17, BVP+16, NSBB18]. Similarly to authors in [MDP+20b], we are rather interested in characterizing the three classes by discriminating each class from the other two. We consider the PSTL template mined in [NSBB18]: $\varphi = ((x_2 > p_2)\, \mathbf{U}_{[T_1,T_2]}\, (x_1 \leq p_1))$, where x_1 and x_2 are variables that refer to the first and second coordinate of the signals, respectively. For each one of the three classes, we aim to find the parameter valuations for p_1, p_2, T_1 and T_2 that produce the most appropriate instantiation of φ.

We observe that φ is monotonic with respect to all its parameters (increasing with respect to T_2 and p_1, and decreasing with respect to T_1 and p_2). Hence, we can apply the monotonic variant of our methods. The set of specifications generated using 150 training samples (50 for each class) is:

$$\begin{aligned}
\text{Class 1 (green): } \varphi_{\hat{\boldsymbol{v}}_1} &= (x_2 > 22.44)\, \mathbf{U}_{[110,150]}\, (x_1 \leq 42.0)\\
\text{Class 2 (red): } \varphi_{\hat{\boldsymbol{v}}_2} &= (x_2 > 24.25)\, \mathbf{U}_{[20,75]}\, (x_1 \leq 42.0)\\
\text{Class 3 (blue): } \varphi_{\hat{\boldsymbol{v}}_3} &= (x_2 > 29.69)\, \mathbf{U}_{[110,300]}\, (x_1 \leq 66.37).
\end{aligned}$$

We evaluate the classification performance of our method for each class by considering the misclassification rate (MCR), namely the number of misclassified signals over the total number of signals. MCR for class i is defined as $\mathrm{MCR}_i := \frac{|\mathrm{FN}_i|+|\mathrm{FP}_i|}{\sum_{j=1}^{k}|\mathcal{S}_j|}$, where FN_i and FP_i correspond to the false negatives and false positives of class i, respectively.

Table 3.1 summarizes the results of average MCRs on the testing set over five different runs (different training samples are drawn at every run), the number of training and testing samples used and the computational time. We use Table 3.1 to first study how

Table 3.1: Comparison of MCR values (average and standard deviation, when available) with previous works on testing set, number of training and testing samples and computational time.

	# samples (train - test)	MCR_1	MCR_2	MCR_3	Time (hours)
FixedBin	30 - 1970	0.04 ± 0.02	0.003 ± 0.003	0.04 ± 0.02	0.78 ± 0.02
FixedBin	90 - 1910	0.008 ± 0.008	0.002 ± 0.002	0.006 ± 0.006	1.71 ± 0.05
FixedBin	150 - 1850	0.02 ± 0.01	0.003 ± 0.002	0.02 ± 0.01	2.53 ± 0.07
AdaBin	150 - 1850	0.01 ± 0.02	0.002 ± 0.001	0.01 ± 0.02	3.7 ± 0.3
[BVP+16]	150 - 1850	0.008 ± 0.004	-	-	0.0001 ± 0.0001
[MDP+20b]	150 - 1850	0.009 ± 0.01	0.01 ± 0.005	0.08 ± 0.07	0.53 ± 0.01
[NSBB18]	150 - 1850	0.52 ± 0.1	-	-	0.043 ± 0.001

our method scales when varying the number of training samples and then to compare our approach with previous works in the literature.

As stated by Theorem 2, the complexity of *FixedBin* increases linearly with the increasing of the number of traces in the dataset; this is consistent with the experimental results reported in the first three rows of Table 3.1 and depicted in Figure 3.6 (see Appendix). In terms of MCR, we would expect the performance to improve when increasing the number of traces in the training set; we observe that this trend is confirmed (*FixedBin* achieves better results with 150 signals than with 30), but not in a monotonic way, as *FixedBin* with 90 samples outperforms the other two.

To make a fair comparison with the existing works from the literature, we installed the tools presented in [BVP+16, MDP+20b, NSBB18] on our laptop and ran them with the same number of training and testing samples. For [NSBB18] we could not reproduce the results reported in the paper and the MCR we obtained is very poor; conversely, the results achieved for [BVP+16, MDP+20b] are consistent with the values presented in the respective papers.

Our misclassification rates in classifying class 1 for *FixedBin* and *AdaBin* are slightly greater but still comparable with MCR_1 of [BVP+16, MDP+20b], while, for the second and third classes, our methods outperform the only method that tackled the multi-class classification problem. We remark that [MDP+20b] addresses the multi-class classification problem as multiple binary classification problems. This means that three different classifiers are trained: one to distinguish class 1 from classes 2 and 3 together, a second for class 2 versus classes 1 and 3, and a third for class 3 versus classes 1 and 2. This is a common approach when tackling the multi-class classification problem, but, if no rules about how to combine the classifiers are presented, the prediction of the label of a new trace might be ambiguous: a trace might be unintentionally classified in several classes at the same time. Conversely, we combine different classifiers into a unique one discriminating among the three classes simultaneously.

Since both of our methods approximate the entire validity domain before selecting the

Figure 3.4: Example of images extracted from the onboard camera of the vehicle. For illustration purpose, four pedestrians were placed in each scene. Left: scenario of a clear day with adult pedestrian. Right: scenario with foggy night and child.

parameter values, the computational time is higher compared to that of other tools. Nevertheless, the approaches we propose can be parallelized because the most expensive part corresponds to the computation of the different validity domains that are independent of each other. For *FixedBin*, even different cells within the same validity domain are computed independently. Surprisingly, we observe that *AdaBin* performs slower than *FixedBin*, despite evaluating a smaller number of parameters (12138 versus 65536). The reason for that might be attributed to the recursive implementation of *AdaBin*, which is generally less efficient than an iterative one (as used in *FixedBin*). However, it is important to note that this result is specific to the monotonic variants, where only one parameter is monitored for every cell. Therefore, in this setting, skipping the evaluation of one cell does not accelerate the overall process as much as in the general case.

3.5.3 Parking Scenario

In this case study, a vehicle is driving through a parking lot when a pedestrian walks out from behind a SUV onto the path of the ego vehicle. More specifically, the pedestrian walks in the x-direction at a speed of 1.5m/s, while the vehicle drives in an orthogonal y-direction with a constant velocity in the range [5.55, 11.11] mps (20-40 kph). The vehicle is equipped with an automatic emergency braking (AEB) function which brakes as hard as possible to avoid a collision with the pedestrian. We simulate the scenario with CARLA 9.13, an open-source photo-realistic driving simulator for autonomous vehicle research [DRC+17]. An RGB camera is mounted on top of the vehicle and is equipped with YOLOv7, a state of the art image detection deep neural network [WBL22]. We use the YOLOv7 model off-the-shelf and trigger an AEB procedure when there is an output with the label “person” with over 0.94 confidence probability (the threshold was set to completely eliminate false positives).

Since the AEB functionality is triggered through visual data, we have 3 different visual variations of the scenario: Time of day {day, night}, Clarity {clear, foggy}, Pedestrian type {adult, child}. For each combination, we collected 200 batches of data (coordinates of the position, velocity and acceleration of the car and the pedestrian). Fig. 3.4 shows two images extracted from the onboard camera for further illustration.

Suppose we are an engineer who wants to characterize the dynamics of two classes of data belonging to two opposite scenarios: an adult crossing the parking lot during the day versus a child pedestrian at night. Since each episode might end either with the vehicle braking and reaching a complete stop before hitting the pedestrian or with the vehicle crashing into the pedestrian, we start our analysis by providing the outcome of the episodes (crash or no crash) with an intuitive robustness measure. In particular, in the case of a non-crashing episode, we study the distance $d(t)$ of the car and the pedestrian over time. If we registered no crash, the car managed to stop in time, so $d(t)$ remains strictly positive throughout the whole episode. The greater the distance is, the safer the episode is (and vice versa). We translate the study of the distance between car and pedestrian as a function of the car's velocity as the following PSTL template:

$$\varphi_1 = \mathbf{G}(v_c(t) < v_{\text{no-crash}}) \rightarrow \mathbf{G}(d(t) \geq d_{\text{no-crash}}),$$

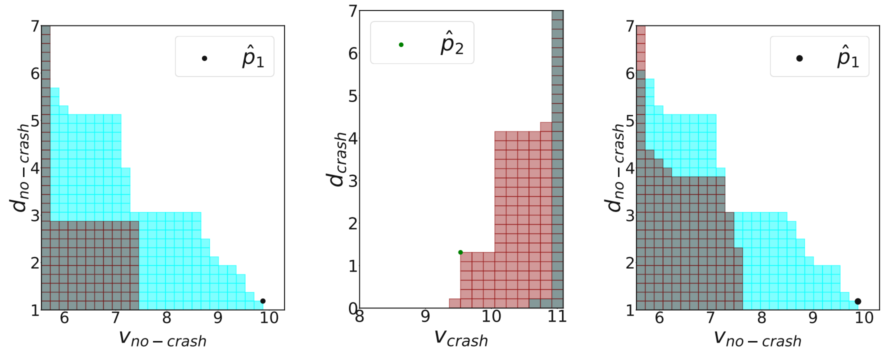
where $v_c(t)$ indicates the car's velocity at time t, while $v_{\text{no-crash}}$ and $d_{\text{no-crash}}$ are the parameters to be mined.

Conversely, we can quantify the robustness of a crashing scenario by studying how strong the impact on the pedestrian is. To do so, we can either consider the car's velocity at the moment of the crash or the distance the car would have required to stop after passing the crashing point. We opt for the latter option so that both the two measures of robustness are distances. Knowing the velocity v and the acceleration a (which is a deceleration) at the last frame before the crash, we apply basic notions of physics and estimate the distance D covered by the car before getting to a complete stop after passing the pedestrian's y-position using the following relationship: $D = \frac{v^2}{2a}$. The greater D is, the stronger the impact is, and, consequently, the unsafer the scenario is. To study the value of D as a function of the car's velocity, we consider the following PSTL template:

$$\varphi_2 = \mathbf{F}(v_c(t) > v_{\text{crash}}) \rightarrow (D \geq d_{\text{crash}}),$$

where $v_c(t)$ represents the car's velocity at time t, while v_{crash} and d_{crash} are the parameters to learn.

The quantification of the robustness of crashing and non-crashing episodes led to two PSTL templates. What would the engineer need to do now? Pick just one of the two templates? If our approach discriminated two classes using only one formula (as is always the case in the literature), the answer would be yes, but our approach allows different classes to be associated with different templates (Section 3.4.2). Guided by common sense, we judge the scenario taking place during the day and with the adult as pedestrian to be *safer* than the one by night with the child pedestrian since the greater visibility should allow the car to identify the pedestrian in advance (we will see later on that our method's output verifies the validity of this assumption). Since φ_1 describes a non-crashing episode, we associate it to the *safer* scenario - day and adult - whose class



(a) Cyan: $\hat{\mathcal{V}}(\mathcal{S}_1, \varphi_1)$; Red: $\hat{\mathcal{V}}(\mathcal{S}_2, \varphi_1)$.

(b) Cyan: $\hat{\mathcal{V}}(\mathcal{S}_1, \varphi_2)$; Red: $\hat{\mathcal{V}}(\mathcal{S}_2, \varphi_2)$.

(c) Cyan: $\hat{\mathcal{V}}(\mathcal{S}_1, \varphi_1)$; Red: $\hat{\mathcal{V}}(\bar{\mathcal{S}}_2, \varphi_1)$.

Figure 3.5: Left and Middle: Approximation of validity domains for $\mathcal{S}_1$ in cyan and $\mathcal{S}_2$ in red (gray color is given by their overlapping) with respect to φ_1 (Left) and φ_2 (Middle). Right: Approximation of validity domains of $\mathcal{S}_1$ and $\bar{\mathcal{S}}_2$ with respect to φ_1.

of data we indicate by $\mathcal{S}_1$. Conversely, the crashing episode template φ_2 is used for the *unsafer* scenario - night and child - indicated by $\mathcal{S}_2$.

We run the experiments with *FixedBin* (in 6.2 minutes) learning from the 80% of the dataset. Fig. 3.5a depicts the approximated validity domains of $\mathcal{S}_1$ and $\mathcal{S}_2$ with respect to φ_1: parameter $\hat{\boldsymbol{p}}_1$ is chosen in $\mathcal{V}(\mathcal{S}_1, \varphi_1)$ to maximize the distance from $\mathcal{V}(\mathcal{S}_2, \varphi_1)$. Similarly, Fig. 3.5b represents the validity domains with respect to φ_2, where $\hat{\boldsymbol{p}}_2$ is extracted in $\mathcal{V}(\mathcal{S}_2, \varphi_2)$. The learned specifications are:

$$\mathcal{S}_1 : \varphi_{1,\hat{\boldsymbol{p}}_1} = \mathbf{G}(v_c(t) < 9.87) \rightarrow \mathbf{G}(d(t) \geq 1.18)$$
$$\mathcal{S}_2 : \varphi_{2,\hat{\boldsymbol{p}}_2} = \mathbf{F}(v_c(t) > 9.53) \rightarrow (D \geq 1.31).$$

We combine $\varphi_{1,\hat{\boldsymbol{p}}_1}$ and $\varphi_{2,\hat{\boldsymbol{p}}_2}$ to predict the classification of the unseen set of traces (20% of the dataset) and we achieve a MCR of 0.037. In this application, all other tools perform worse: the MCR for [BVP$^+$16] is 0.13, for [NSBB18] is 0.5, while [MDP$^+$20b] returns failure after more than one hour and half of computation. Moreover, to answer **RQ3**, we observe that, apart from the classification itself, our approach provides additional insights into the case study thanks to the computation of the entire validity domains. For instance, by observing Fig. 3.5 (Left and Middle), the engineer gets the proof that $\mathcal{S}_2$ actually describes an *unsafer* scenario than $\mathcal{S}_1$ (for every velocity value, $d_{\text{no-crash}}$ is smaller for $\mathcal{S}_2$, while d_{crash} is greater). In addition, the engineer finds out that the safety on non-crashing scenarios remains unchanged by varying the velocity in the range $[6, 7.5]$ mps ($d_{\text{no-crash}}$ remains, indeed, around 5 meters for the day-adult scenario and 3 meters for the night-child one), but it falls abruptly when the velocity overcomes the threshold of 7.5 mps ($\approx$ 27 kph). Finally, the knowledge of the entire validity domains allows the engineer to easily add restrictions on the parameters to be learned (e.g., increasing the lower bound of $d_{\text{no-crash}}$ from one meter to two meters to enhance safety).

So far, we compared the scenarios of an adult pedestrian crossing the parking lot during the day versus a child pedestrian at night, both with a clear sky. What would happen if the latter scenario had foggy weather instead? Our human experience would consider the fog as a factor that decreases visibility and, consequently, increases the danger. To check this hypothesis, we run new experiments with the second class of data $\bar{\mathcal{S}}_2$ representing traces collected in a foggy night and with a child pedestrian. The MCR value we obtain now is 0.15, significantly greater than before (the other tools achieved the following MCRs: 0.11 for [BVP$^+$16], 0.5 for [NSBB18], failure for [MDP$^+$20b]). Without any additional insights, we would not be able to understand the reason behind this unintuitive result: we would have expected a decrease in the MCR due to a larger gap in safety between the two classes caused by the presence of fog. By comparing Fig. 3.5a with Fig. 3.5c, we observe that the addition of fog produces the expansion of the validity domain of $\bar{\mathcal{S}}_2$ with respect to φ_1, meaning that the vehicle manages to stop further from the pedestrian than in the limpid scenario. Consequently, the foggy scenario is actually safer and therefore less distinguishable from $\mathcal{S}_1$. By manually inspecting the CARLA simulations, we confirmed this analysis by noticing that the fog added some visual noise masking the pedestrian, but in fact also accentuated their contour. The latter effect made the pedestrians more easily recognizable for YOLO. In conclusion, thanks to the insights provided by our method, we discovered that our human intuition for which the fog renders a pedestrian less visible does not hold in the CARLA simulation environment.

3.6 Conclusion and Future Work

We presented a new approach to mine parameters of arbitrary PSTL templates to perform multi-class classification tasks over time series data. We proposed two variants of the mining procedure, providing a statistical guarantee of optimality for the mined parameters. We validated the applicability of our approach in three case studies by demonstrating how to leverage our method's outputs to refine the PSTL template or infer new insights into the data and by comparing our performances with the state-of-the-art.

The approach proposed in this paper is passive - the specification is mined with respect to a set of existing signals. We plan to explore the active learning approach and devise a testing strategy that can help inferring a representative specification that explains well the examples.

Acknowledgments This work has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 956123 and it is partially funded by the TU Wien-funded Doctoral College for SecInt: Secure and Intelligent Human-Centric Digital Technologies.

3.7 Appendix

Lemma 1 (Choice of N)**.** *Let $\mathcal{V}$ be a validity domain and C a cell not included in the (intended) over-approximation $\hat{\mathcal{V}}$ computed with N random samples per cell. If $N \geq \log_{1-p_0}(\alpha)$ with p_0, $\alpha \in (0,1)$, then the fraction of points in C that are contained in $\mathcal{V}$ is smaller than p_0 with confidence $1-\alpha$.*

Proof. Let us suppose to compute the over-approximation $\hat{\mathcal{V}}$ of the validity domain $\mathcal{V}$ by adding a cell to $\hat{\mathcal{V}}$ if and only if at least one of the parameter valuations inside the cell belongs to the real validity domain. In applying this technique, we need a stopping criterion when no satisfied parameter valuations are found in a cell. This criterion is related to the number of parameter valuations to be examined before classifying a cell as not part of the validity domain.

With this goal in mind, let us briefly describe the *sequential probability ratio test*, introduced by Wald in [Wal92]. The main idea is to perform the classical hypothesis testing with a number of samples not fixed in advance and that is shown to be minimal. Let H_0 denote the null hypothesis, H_1 the alternative hypothesis, α and β be the upper bounds for error of Type I and II, respectively. The test consists in iteratively drawing a sample and evaluating the corresponding likelihood ratio, given by the likelihood under the alternative hypothesis divided by the likelihood under the null hypothesis.

- If the value of the ratio is greater than the quantity $A(\alpha,\beta) = \frac{1-\beta}{\alpha}$, the alternative hypothesis H_1 is accepted and the test stops;
- If the value of the ratio is smaller than the value $B(\alpha,\beta) = \frac{\beta}{1-\alpha}$, the null hypothesis H_0 is accepted and the test stops;
- If none of the previous conditions is satisfied, a new sample is drawn and the procedure is repeated.

In our case, we use the *sequential probability ratio test* to compute the minimum number N of unsatisfied parameter valuations that we need to evaluate in a cell, before concluding with confidence $1-\alpha$ that the fraction of satisfied parameter valuations of the cell is smaller than p_0, for a chosen value of $p_0 \in (0,1)$. If, while doing the N evaluations, we find one satisfied parameter valuation, the *sequential probability ratio test* is not required anymore, since we are sure there exists at least one satisfied parameter valuation in the cell and, therefore, we can add the cell to the approximation of the validity domain.

Let p be the unknown true fraction of satisfied parameter valuations of the cell. The null hypothesis (that we aim to reject) is that the value of p is greater than a predefined threshold $p_0 \in (0,1)$, namely $H_0 : p \geq p_0$. On the other hand, the alternative hypothesis (that we aim to accept) is that the proportion of satisfied parameter valuations in the cell under study is zero, namely $H_1 : p = 0$. The region $(0, p_0)$ is called *indifference region* and it must be non-empty in order to freely choose α and β. If H_2 is the hypothesis that neither H_0 nor H_1 holds, i.e., $H_2 : p \in (0, p_0)$, then the following results are guaranteed [YS02]:

$$\begin{cases} Pr[H_0 \text{ holds} \mid \text{accept } H_1] & \leq \alpha, \\ Pr[H_1 \text{ holds} \vee H_2 \text{ holds} \mid \text{accept } H_1] & \geq 1 - \alpha. \end{cases}$$

In the following, a sample s corresponds to a parameter valuation and its value is 1 if the parameter valuation renders the formula satisfied, 0 otherwise. P indicates the Bernoulli probability distribution. When r samples have been drawn and all of them are unsatisfied, the likelihood ratio is given by:

$$\frac{p_{1,r}}{p_{0,r}} = \frac{P\left(s_1 = 0, \ldots, s_r = 0 \mid p = 0\right)}{\max\limits_{p_0 \leq \bar{p} \leq 1} P\left(s_1 = 0, \ldots, s_r = 0 \mid p = \bar{p}\right)} =$$
$$\frac{1}{\max\limits_{p_0 \leq \bar{p} \leq 1} \binom{r}{0} \bar{p}^0 (1 - \bar{p})^{(r-0)}} = \frac{1}{(1 - p_0)^r}.$$

In order to satisfy $\frac{p_{1,r}}{p_{0,r}} \geq A$, it is therefore sufficient to choose $N = r \geq \log_{(1-p_0)}\left(\frac{1}{A}\right) = \log_{(1-p_0)}\left(\frac{\alpha}{1-\beta}\right)$. We can set β (i.e., the error of Type II) to zero because it corresponds to the situation in which H_1 holds but we have accepted H_0. However, we never accept H_0 since we stop the sequential test and include a cell to the over-approximation as soon as we find one satisfied parameter valuation. This is due to the fact that we do not need to accept $p \geq p_0$ to include a cell in $\hat{\mathcal{V}}$, but it is enough that one satisfied valuation is found.

In conclusion, the evaluation of a cell is carried out by evaluating at most $\log_{(1-p_0)}(\alpha)$ parameter valuations. □

Before proving Theorem 1, let us consider some preliminary results.

Lemma 2. *Triangle inequality holds for function D, namely $\forall \mathcal{V}_1, \mathcal{V}_2, \mathcal{V}_3$ such that $\mathcal{V}_i$ is a bounded set in $\mathbb{R}^m$ for $i = 1, 2, 3$,*

$$D(\mathcal{V}_1, \mathcal{V}_2) \leq D(\mathcal{V}_1, \mathcal{V}_3) + D(\mathcal{V}_3, \mathcal{V}_2).$$

The result is known but we provide a sketch of the proof for completeness sake.

Proof. Given $\varepsilon > 0$, for each $\boldsymbol{v}_1$ in $\mathcal{V}_1$ there must exist a point $\hat{\boldsymbol{v}}_2$ in $\mathcal{V}_2$ such that $dist(\boldsymbol{v}_1, \hat{\boldsymbol{v}}_2) \leq D(\mathcal{V}_1, \mathcal{V}_2) + \varepsilon$. This is due to the fact that, for each $\boldsymbol{v}_1$ in $\mathcal{V}_1$,

$$\inf_{\boldsymbol{v}_2 \in \mathcal{V}_2} dist(\boldsymbol{v}_1, \boldsymbol{v}_2) = d(\boldsymbol{v}_1, \mathcal{V}_2) \leq D(\mathcal{V}_1, \mathcal{V}_2).$$

So, for every $\varepsilon > 0$,

$$D(\mathcal{V}_1, \mathcal{V}_2) + \varepsilon \geq \inf_{\boldsymbol{v}_2 \in \mathcal{V}_2} dist(\boldsymbol{v}_1, \boldsymbol{v}_2) + \varepsilon \geq dist(\boldsymbol{v}_1, \hat{\boldsymbol{v}}_2)$$

for a certain $\hat{\boldsymbol{v}}_2$ in $\mathcal{V}_2$. Therefore, given $\varepsilon_1, \varepsilon_2 > 0$, for each $\boldsymbol{v}_1$ in $\mathcal{V}_1$ there exist $\hat{\boldsymbol{v}}_2$ in $\mathcal{V}_2$ and $\hat{\boldsymbol{v}}_3$ in $\mathcal{V}_3$ such that:

$$dist(\boldsymbol{v}_1, \hat{\boldsymbol{v}}_2) \leq dist(\boldsymbol{v}_1, \hat{\boldsymbol{v}}_3) + dist(\hat{\boldsymbol{v}}_3, \hat{\boldsymbol{v}}_2) \leq D(\mathcal{V}_1, \mathcal{V}_3) + \varepsilon_1 + D(\mathcal{V}_3, \mathcal{V}_2) + \varepsilon_2,$$

where the first inequality holds because *dist* is the Euclidean distance. Hence,

$$\inf_{\boldsymbol{v}_2 \in \mathcal{V}_2} dist(\boldsymbol{v}_1, \boldsymbol{v}_2) \leq dist(\boldsymbol{v}_1, \hat{\boldsymbol{v}}_2) \leq D(\mathcal{V}_1, \mathcal{V}_3) + D(\mathcal{V}_3, \mathcal{V}_2) + \varepsilon_1 + \varepsilon_2.$$

Since the previous inequality holds for any $\boldsymbol{v}_1 \in \mathcal{V}_1$, the following inequality is also satisfied

$$D(\mathcal{V}_1, \mathcal{V}_2) = \sup_{\boldsymbol{v}_1 \in \mathcal{V}_1} \inf_{\boldsymbol{v}_2 \in \mathcal{V}_2} dist(\boldsymbol{v}_1, \boldsymbol{v}_2) \leq D(\mathcal{V}_1, \mathcal{V}_3) + D(\mathcal{V}_3, \mathcal{V}_2) + \varepsilon_1 + \varepsilon_2.$$

Since the inequality holds for arbitrary $\varepsilon_1, \varepsilon_2 > 0$, the thesis is proven. □

Lemma 3. *$D(\hat{\mathcal{V}}_i, \mathcal{V}_i) \leq \|\boldsymbol{g}\|$, for $i = 1, 2$.*

Proof. For simplicity of notation, let us consider $i = 1$. Let us pick a generic point $\boldsymbol{v} \in \hat{\mathcal{V}}_1$; we want to prove that $d(\boldsymbol{v}, \mathcal{V}_1) \leq \|\boldsymbol{g}\|$. If $\boldsymbol{v} \in \mathcal{V}_1$, then $d(\boldsymbol{v}, \mathcal{V}_1) = 0 \leq \|\boldsymbol{g}\|$. Otherwise, $\boldsymbol{v} \in \hat{\mathcal{V}}_1 \setminus \mathcal{V}_1$. Since $\boldsymbol{v} \in \hat{\mathcal{V}}_1$, the whole cell $C_{\boldsymbol{v}}$ to which $\boldsymbol{v}$ belongs to is included in $\hat{\mathcal{V}}_1$ ($\boldsymbol{v}$ may belong to several cells if it is a point on the border; in this case, at least one of these cells must be included in $\hat{\mathcal{V}}_1$). According to the procedure applied in our methods, a cell is included in the approximation $\hat{\mathcal{V}}_1$ of the validity domain $\mathcal{V}_1$ only if at least one of its point belongs to $\mathcal{V}_1$. Let us name $\boldsymbol{w}$ the point in $C_{\boldsymbol{v}}$ that belongs to $\mathcal{V}_1$. Since the maximum distance between two points in the same cell having size $\boldsymbol{g}$ is $\|\boldsymbol{g}\|$, it follows that $dist(\boldsymbol{v}, \boldsymbol{w}) \leq \|\boldsymbol{g}\|$. Therefore,

$$d(\boldsymbol{v}, \mathcal{V}_1) = \inf\{dist(\boldsymbol{v}, \boldsymbol{u}) \mid \boldsymbol{u} \in \mathcal{V}_1\} \leq dist(\boldsymbol{v}, \boldsymbol{w}) \leq \|\boldsymbol{g}\|.$$

We proved that for all $\boldsymbol{v} \in \hat{\mathcal{V}}_1$, $d(\boldsymbol{v}, \mathcal{V}_1) \leq \|\boldsymbol{g}\|$. This implies that $\sup\{d(\boldsymbol{v}, \mathcal{V}_1) \mid \boldsymbol{v} \in \hat{\mathcal{V}}_1\} \leq \|\boldsymbol{g}\|$. The conclusion holds because the left hand side of the previous inequality corresponds to the definition of $D(\hat{\mathcal{V}}_1, \mathcal{V}_1)$. □

Theorem 1 (Optimality)**.** *Let $i \in \{1, 2\}$, $\mathcal{V}_i$ be the validity domain for class $\mathcal{S}_i$, and $\hat{\mathcal{V}}_i$ be the approximation of $\mathcal{V}_i$ using $N \geq \log_{1-p_0}(\alpha)$ random samples per cell and the cell resolution given by vector $\boldsymbol{g} \in \mathbb{R}^m_{\geq 0}$. Let $\boldsymbol{v}_i^*$ be an optimal parameter valuation for class i and $\hat{\boldsymbol{v}}_i$ the parameter valuation found with FixedBin. Then, the difference of the distances from $\hat{\boldsymbol{v}}_i$ and from $\boldsymbol{v}_i^*$ to the other class validity domain is bounded by the cell resolution:*

$$|d(\hat{\boldsymbol{v}}_i, \mathcal{V}_{(i \bmod 2)+1}) - d(\boldsymbol{v}_i^*, \mathcal{V}_{(i \bmod 2)+1})| \leq \|\boldsymbol{g}\| \tag{3.4}$$

with probability $1 - p_0$ and confidence $1 - \alpha$.

Proof. Let us first prove $d(\boldsymbol{v}_1^*, \mathcal{V}_2) - d(\hat{\boldsymbol{v}}_1, \mathcal{V}_2) \leq \|\boldsymbol{g}\|$. We observe that:

$$d(\boldsymbol{v}_1^*, \mathcal{V}_2) \leq D(\mathcal{V}_1, \mathcal{V}_2) \leq D(\mathcal{V}_1, \hat{\mathcal{V}}_1) + D(\hat{\mathcal{V}}_1, \hat{\mathcal{V}}_2) + D(\hat{\mathcal{V}}_2, \mathcal{V}_2), \tag{3.6}$$

where the last inequality is obtained by the double application of triangle inequality for function D (Lemma 2).

By computing $\hat{\mathcal{V}}_1$ with $N \geq \log_{1-p_0}(\alpha)$ random samples per cell, we can deduce from Lemma 1 that the probability that $\hat{\mathcal{V}}_1$ is actually an over-approximation of $\mathcal{V}_1$ is $1 - p_0$ with confidence $1 - \alpha$; hence, $D(\mathcal{V}_1, \hat{\mathcal{V}}_1) = 0$ with probability $1 - p_0$ and confidence $1 - \alpha$. In addition, $D(\hat{\mathcal{V}}_2, \mathcal{V}_2) \leq \|\boldsymbol{g}\|$ for Lemma 3. Moreover, since $D(\hat{\mathcal{V}}_1, \hat{\mathcal{V}}_2) = \sup\{d(\boldsymbol{v}, \hat{\mathcal{V}}_2) \mid \boldsymbol{v} \in \hat{\mathcal{V}}_1\}$, for every $\varepsilon > 0$, there must exist $\boldsymbol{v} \in \hat{\mathcal{V}}_1$ such that $D(\hat{\mathcal{V}}_1, \hat{\mathcal{V}}_2) \leq d(\boldsymbol{v}, \hat{\mathcal{V}}_2) + \varepsilon$. We remind also that $\hat{\boldsymbol{v}}_1$ was chosen in $\hat{\mathcal{V}}_1$ such that $d(\boldsymbol{v}, \hat{\mathcal{V}}_2) \leq d(\hat{\boldsymbol{v}}_1, \hat{\mathcal{V}}_2)$ for every $\boldsymbol{v} \in \hat{\mathcal{V}}_1$. Proceeding from (3.6), we obtain therefore:

$$\begin{aligned}
&d(\boldsymbol{v}_1^*, \mathcal{V}_2) \leq D(\hat{\mathcal{V}}_1, \hat{\mathcal{V}}_2) + D(\hat{\mathcal{V}}_2, \mathcal{V}_2) \leq d(\boldsymbol{v}, \hat{\mathcal{V}}_2) + \varepsilon + \|\boldsymbol{g}\| \leq \\
&d(\hat{\boldsymbol{v}}_1, \hat{\mathcal{V}}_2) + \varepsilon + \|\boldsymbol{g}\| \leq d(\hat{\boldsymbol{v}}_1, \mathcal{V}_2) + D(\mathcal{V}_2, \hat{\mathcal{V}}_2) + \varepsilon + \|\boldsymbol{g}\| = \\
&d(\hat{\boldsymbol{v}}_1, \mathcal{V}_2) + \varepsilon + \|\boldsymbol{g}\|,
\end{aligned}$$

where the last equality holds in probability since $D(\mathcal{V}_2, \hat{\mathcal{V}}_2) = 0$, as previously described for $\mathcal{V}_1$. Since $d(\boldsymbol{v}_1^*, \mathcal{V}_2) - d(\hat{\boldsymbol{v}}_1, \mathcal{V}_2) \leq \|\boldsymbol{g}\| + \varepsilon$ holds for every ε, the thesis is proven.

Let us now prove that $d(\hat{\boldsymbol{v}}_1, \mathcal{V}_2) - d(\boldsymbol{v}_1^*, \mathcal{V}_2) \leq \|\boldsymbol{g}\|$.

$$\begin{aligned}
&d(\hat{\boldsymbol{v}}_1, \mathcal{V}_2) - d(\boldsymbol{v}_1^*, \mathcal{V}_2) \leq d(\hat{\boldsymbol{v}}_1, \mathcal{V}_2) - D(\mathcal{V}_1, \mathcal{V}_2) + \varepsilon \underset{\text{Def of } D \text{ in (3.2)}}{\leq} \\
&D(\hat{\mathcal{V}}_1, \mathcal{V}_2) - D(\mathcal{V}_1, \mathcal{V}_2) + \varepsilon \underset{\text{Lemma 2}}{\leq} \\
&D(\hat{\mathcal{V}}_1, \mathcal{V}_1) + \cancel{D(\mathcal{V}_1, \mathcal{V}_2)} - \cancel{D(\mathcal{V}_1, \mathcal{V}_2)} + \varepsilon = \\
&D(\hat{\mathcal{V}}_1, \mathcal{V}_1) + \varepsilon \underset{\text{Lemma 3}}{\leq} \|\boldsymbol{g}\| + \varepsilon
\end{aligned}$$

where the first inequality holds for every $\varepsilon > 0$ for definition of supremum and for the choice of $\boldsymbol{v}_1^*$: $D(\mathcal{V}_1, \mathcal{V}_2) - \varepsilon \leq d(\boldsymbol{v}, \mathcal{V}_2) \leq d(\boldsymbol{v}_1^*, \mathcal{V}_2)$. □

Theorem 2 (Complexity)**.** *Let n be the total number of signals and let m be the number of different parameter symbols in the PSTL template φ. Let $\boldsymbol{g} \in \mathbb{R}^m_{\geq 0}$ be the vector that contains the minimal sizes of the cells and let the parameter space be $\mathcal{W} = \prod_{i=1}^m [a_i, b_i]$ as in Eq. (3.3). Then, the total number L of monitors is exponential in m and linear in n. In particular,*

$$L \leq \left(\max_{i=1,\ldots,m} \left\lceil \frac{b_i - a_i}{g_i} \right\rceil \right)^m \cdot (N \cdot n) = \mathcal{O}(M^m) \cdot \mathcal{O}(n),$$

where N is the maximum number of parameter valuations to be evaluated in every cell and M is a suitable positive number.

Proof. The total number c of cells needed to approximate $\mathcal{W}$ is given by:

$$c = \prod_{i=1}^{m} \left\lceil \frac{b_i - a_i}{g_i} \right\rceil \leq \left(\max_{i=1,\ldots,m} \left\lceil \frac{b_i - a_i}{g_i} \right\rceil \right)^m, \tag{3.7}$$

which is therefore exponential in the number of parameters m and does not depend on the number of time series n. In every cell, the total number l of monitored formulas is proportional to the number of time series n. In particular, $l \leq N \cdot n$, where N is the maximum number of parameter valuations to be evaluated in every cell (the choice of N is done according to Lemma 1). In conclusion, from $L \leq c \cdot l \leq c \cdot N \cdot n$ and the replacement of c with (3.7), the thesis is proven. □

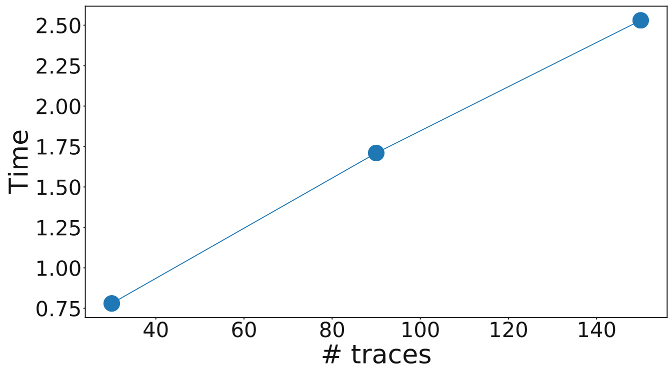


Figure 3.6: Computational time (hours) as a function of the number of traces in the training set for the *Naval* case study.

The linear dependence of the complexity with respect to the number of signals n stated by Theorem 2 is consistent with the results obtained in the *Naval* case study (Section 3.5.2), whose values are reported in Table 3.1 and depicted in Figure 3.6.

CHAPTER 4

Mining Specifications for Predictive Safety Monitoring

Reference. The content of this chapter has been published in [NBG+25]: *Eleonora Nesterini, Ezio Bartocci, Alessio Gambi, Dejan Nickovic, Sanjit A. Seshia, and Hazem Torfah. 2025.* Mining Specifications for Predictive Safety Monitoring. *In Proceedings of the ACM/IEEE 16th International Conference on Cyber-Physical Systems (with CPS-IoT Week 2025) (ICCPS '25). Association for Computing Machinery, New York, NY, USA, Article 6, 1–11.* `https://doi.org/10.1145/3716550.3722021`

Abstract. Safety-critical autonomous systems must reliably predict unsafe behavior to take timely corrective actions. Safety properties are often defined over variables that are not directly observable at runtime, making prediction and detection of violations hard. We present a new approach for learning interpretable monitors characterized by concise Signal Temporal Logic (STL) formulas that can predict safety property violations from the observable sensor data. We train these monitors from synthetic, possibly highly unbalanced data generated in a simulation environment. Our specification mining procedure combines a grammar-based method and two novel ensemble techniques. Our approach outperforms the existing solutions by enhancing accuracy and explainability, as demonstrated in two autonomous driving case studies.

4.1 Introduction

The deployment of autonomous Cyber-Physical Systems (CPS) has significantly increased in recent years in safety-critical domains. Applications such as autonomous driving (AD) leverage recent advancements in machine learning (ML) to enable driverless vehicle operation, offering many potential benefits, including reduced traffic accidents, enhanced comfort, and reduced traffic congestion. Given the complex environments in which these safety-critical systems operate and the brittleness of ML components, it is crucial to accurately detect potential safety risks and allow for timely countermeasures to prevent unsafe outcomes like collisions [SSS22].

The detection of potential safety risks in CPS is best managed by runtime monitors that continuously evaluate system safety [BFFR18]. An accurate specification that captures the safety requirements is key to effective runtime monitoring. Formal specifications allow developers to precisely define these requirements and offer the crucial advantage of enabling automated monitor generation. However, a significant bottleneck in constructing and integrating such monitors arises because safety specifications are often defined over system-level variables that the autonomous system cannot directly observe during operation. The perfect knowledge of geographical boundaries, roads' type and physical attributes, or the ground-truth distance between the autonomous vehicle and the other traffic participants are features relevant for detecting safety violations in an autonomous driving system that, in practice, are not directly available to the autonomous vehicle.

In order to serve as a practical means to increase the safety of autonomous CPS, the monitors must rely only on variables observable from within the CPS. Variables like sensor data (e.g., images) and internal controller inputs and outputs are generally available; however, using them to design effective safety monitors is challenging because their connection to safety properties is not straightforward, i.e., there is no one-to-one relationship between them.

This paper addresses this challenge by introducing a novel framework for mining safety monitor specifications over observable data. Given a traditional system-level specification represented as a formula in Signal Temporal Logic (STL) [MN04], our framework learns another STL specification that approximates the validity of the original system-level specification solely using valuations of observable variables. Our framework implements a data-driven approach that collects data from system simulations, evaluates it using the predefined system-level specification, and derives new specifications based on observable variables using a grammar-based specification mining process. This novel specification mining process allows CPS developers to encode prior knowledge and tailor the target specifications to their applications. Recognizing that data-driven approaches may not lead to generating perfectly accurate monitors, our mining process solves an optimization problem to minimize false positive (unwarranted alarms) and false negative (missed violations) rates.

To learn monitors that detect impending safety property violations *before* they occur, thus enabling corrective actions to prevent unsafe scenarios, we extend observable-level validity labels with future system-level predictions following the work by Torfah et al. [TXJ$^+$22, TJS$^+$23]. However, to address the main shortcomings of previous works, namely the generated monitors' lack of interpretability and (over-)sensitivity to training data, we leverage STL formulas to learn concise specifications and use quantitative STL semantics (i.e., robustness) to combine them into a single monitor capable of dealing with variance in the training data.

We evaluate our approach in two representative case studies in the autonomous driving domain and show that our "robustness-based" ensemble of STL monitors outperforms existing approaches while maintaining a compact, interpretable structure. Although our evaluation focuses on autonomous driving systems, the proposed method is general and can be applied to other autonomous and safety-critical CPS.

To summarize, the main contributions of this paper are:

1. A novel data-driven approach for mining accurate and explainable predictive STL safety monitors from the observable variables only.
2. As a central component of the above approach, a grammar-based method for mining STL formulas that discriminate between positive and negative examples, even in the case of highly unbalanced datasets;
3. Two new ensemble methods that leverage the STL quantitative semantics;
4. Extensive experimental analysis that compares our approach to the state-of-the-art (in terms of monitors learning and ensemble techniques) and studies its sensitivity to key dataset characteristics (imbalance, size, and prediction horizon).

We release the implementation of our approach along with all the data, scripts, and instructions to reproduce the experiments and analyze the data in the publicly accessible repository *ObSTLearn*[1].

4.2 Related Work

Recent work [TXJ+22] introduced the problem of learning observable monitors to predict the safety of system's executions. However, the decision-tree approach employed by the authors produced trees with hundreds of nodes that are hard to interpret. Since explainability is essential in safety-critical domains, we look for monitors in the form of STL specifications, suitable to express temporal properties of CPS in an unambiguous yet human-understandable manner. *Specification mining* [BMNN22] is the research field that focuses on automatically inferring properties of a system from its executions. STL mining methods are generally categorized as either *template-based* [ADMN11] or *template-free* [BVP+16, MDP+20b], depending on whether only parameter values or both the formula structure and parameter values must be learned from data, respectively. Recent works [PMN21, IBMN22, BMNN23] mitigated this distinction by enabling the definition of an STL (sub)grammar of admissible formulas and hence allowing for the embedding of a flexible amount of knowledge in the mining process. Existing grammar-based methods aim to learn formulas to (tightly) describe only a single set of positive examples [JTS+19]. Conversely, the predictive monitoring problem addressed in this paper requires discriminating between sets of positive and negative examples [BVP+16,

[1] https://github.com/eleonoranesterini/ObSTLearn.git

MDP+20b]. In addition, in this context, negative executions result from system faults or safety issues and are much rarer than favorable executions, leading to significantly imbalanced datasets.

For these reasons, we devised and implemented an extension of a recently-proposed grammar-based approach [BMNN23] to learn from both positive and negative examples, even in highly imbalanced settings. Furthermore, we propose two ensemble methods that leverage STL quantitative semantics to improve the accuracy of multiple mined formulas inspired by promising insights in the literature. In particular, authors in [TJS+23] proved that classical ensemble methods, such as Majority Voting, reduce the misclassification rate exponentially with the number of monitors. Other works [IBMN22, AVBB22] have exploited this ensemble method to enhance the classification performances of STL formulas. However, to the best of our knowledge, ensemble methods that leverage the intrinsic characteristics of STL (such as its quantitative semantics) have not been explored yet.

Other predictive monitoring techniques were proposed in [DHF14, MSBF21] where the authors used a Simulink model [DHF14] or an ML model [MSBF21] of the CPS to predict what will happen in the future. However, in both cases, the monitoring properties are assumed to be given, while in our approach they are learned from systems executions.

4.3 STL Background

We define a *trace* of $m \in \mathbb{N}$ real-valued variables and length $n \in \mathbb{N}$ as the finite sequence of the (time,value) pairs $x = (t_1, x_1), \dots, (t_n, x_n)$, where $t_1 = 0$, $t_i < t_{i+1}\ \forall i \in \{1, \dots, n-1\}$, and $x_i \in \mathbb{R}^m$. For a trace x of length n, the *prediction horizon* $h \in \mathbb{N}$ with $h < n$ is the number of time steps ahead of the final step at which we aim to make our prediction. Hence, we will consider the trace x truncated by removing its last h time steps, which we denote by $x^{-h} := (t_1, x_1), (t_2, x_2), \dots, (t_{n-h}, x_{n-h})$.

We adopt Signal Temporal Logic (STL) [MN04] as our specification language. Its syntax is defined by the following grammar:

$$\varphi := \mathbf{true} \mid f(x) > 0 \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1\, \mathbf{U}_I\, \varphi_2$$

where $f(x) > 0$ is an atomic proposition ($f \colon \mathbb{R}^n \to \mathbb{R}$), $\neg$ and $\wedge$ are, respectively, the Boolean negation and conjunction, and $\mathbf{U}_I$ is the *until* temporal operator defined over the dense interval of time I in $\mathbb{R}_{\geq 0}$ - the interval I is generally omitted when $I = [0, \infty)$. The *finally* $\mathbf{F}$ and *globally* $\mathbf{G}$ operators are derived as follows: $\mathbf{F}_I\varphi := \mathbf{true}\,\mathbf{U}_I\varphi$ and $\mathbf{G}_I\varphi := \neg\mathbf{F}_I\neg\varphi$.

STL quantitative semantics was introduced in [DM10] and is defined in terms of the robustness function ρ that maps an STL specification φ, a trace x, and time t, to the value $\rho(\varphi, x, t)$ representing how far is x from the satisfaction boundary of φ at time t. We write $\rho(\varphi; x)$ to indicate $\rho(\varphi, x, 0)$. Notably, the STL quantitative semantics is *sound*:

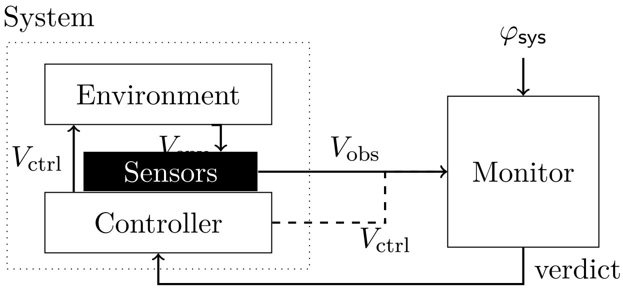


Figure 4.1: A general monitoring architecture in a CPS. This Figure has been redrawn from [JST25].

when $\rho(\varphi; x) > 0$ the trace x satisfies φ (indicated by $x \models \varphi$), whereas when $\rho(\varphi; x) < 0$ the trace x violates φ ($x \not\models \varphi$).

A Parametric STL (PSTL) formula is an STL formula whose temporal and quantitative numerical values are replaced by parameter symbols [ADMN11]. As common in the literature, our learning process will be split into (i) exploring the space of PSTL formulas; and (ii) instantiation of the candidate PSTL formula into a concrete STL formula by choosing appropriate values for its parameter symbols.

4.4 Problem Statement

In Figure 4.1, we present a general monitoring architecture in a closed-loop CPS. The task of the monitor is to issue a verdict regarding the satisfaction of a specification of interest. The monitor can rely only on measurements received via a designated sensor interface to issue its verdict. This means that the monitor deduces information about the system's state (specifically, the environment) using only valuations of an observable feature space provided by the sensor measurements and may have only partial observability of the system's current state.

Formally, we distinguish between two sets of variables: the set of system-level variables V_{sys}, and the set of observable variables V_{obs}. The sets V_{sys} include variables defining both the actual state of the environment V_{env} and control V_{ctrl}. The set V_{obs} includes variables that capture the sensors' measurements.

In the case of autonomous car driving on a city road, the set V_{sys} might include variables such as the absolute coordinates of the *ego car* (namely, the self-driving vehicle under study), the distance between the ego car and other agents, the characteristics of objects on the road. In contrast, V_{obs} might include information collected by sensors mounted on the car, such as images collected by a front-facing camera, radar data, etc., and sometimes also part of V_{ctrl} defining the state of the internal control, e.g., velocity, steering, etc.

Let Σ_{sys} and Σ_{obs} define the sets of valuations of the set of variables V_{sys} and V_{obs}, respectively. The system behavior corresponds to a set $S_{\mathsf{sys}} \subseteq \Sigma^*_{\mathsf{sys}}$ of finite traces over

Σ_{sys}, which we denote as system-level traces, while its executions observable to the monitor are defined as a set $S_{\mathsf{obs}} \subseteq \Sigma^*_{\mathsf{obs}}$ of traces over Σ_{obs}, denoted as observable traces. Considering the partiality on the monitor side, different system-level traces may induce the same observable trace. We assume, however, that a system-level trace always induces the same observable trace[2]. Consequently, the (unknown) relation between the system-level behaviors and the corresponding observations can be expressed by a mapping $\Omega\colon S_{\mathsf{sys}} \to S_{\mathsf{obs}}$.

The system-level specification φ_{sys} is known and defined as a set of traces over valuations of the variables in V_{sys}. It is used to determine whether an execution of S is *safe* or *unsafe*: a system behavior $\sigma \in S_{\mathsf{sys}}$ is considered *safe* if $\sigma \models \varphi_{\mathsf{sys}}$, and *unsafe* otherwise (for consistency with the notation of STL, we use the symbol $\models$ for generic specifications to indicate membership in the set, i.e. $\in$). Since not every variable in φ_{sys} may be directly observable by our monitor, our goal is to translate φ_{sys} to a specification φ_{obs} defined over variables in V_{obs} that captures the validity of φ_{sys} on the observation level. Ideally, we want to find a specification $\varphi_{\mathsf{obs}} \subseteq \Sigma^*_{\mathsf{obs}}$ such that, for every observable trace $x \in S_{\mathsf{obs}}$, $x \models \varphi_{\mathsf{obs}}$ if and only if, $\forall \sigma \in \Omega^{-1}(x), \sigma \models \varphi_{sys}$. Such a formulation of the problem is however idealized and may not be feasible in practice [TXJ+22]. First, this formulation is too conservative. An observation trace is excluded if it can be mapped back to any system-level trace that violates the system-level specification. In practice, this will mostly lead to a specification that rejects most observation traces. A more practical definition of the problem takes into consideration the occurrence rate of system-level traces and optimizes a specification for traces that are more frequent. Furthermore, depending on the (syntactic) class of specifications over the observation space from which φ_{obs} is to be chosen, we may not always find an exact match to φ_{sys}. Yet, we might want to search for the optimal specification within this class. To address these challenges, we change our formulation to define a quantitative optimization problem that mines optimal solutions.

Before introducing our quantitative formulation of the mining problem, we point to two key points:

- in practice, one may want to bias the search toward higher false positive rates to reduce the rate of false negatives. In our formulation, we encode this using specific cost measures for the false positives and false negatives.
- monitors need to be predictive, i.e., they should determine the violation/satisfaction of a specification many steps in advance. To this end, we define our problem in terms of a prediction horizon h, which refers to the violation/satisfaction of a specification h time steps before the end of the execution.

With that, we can now define the problem of mining temporal properties for predictive monitoring as follows.

[2]We assume a setting with deterministic sensors. We consider the setting of noisy sensors to be an extension of our approach and leave it for future work.

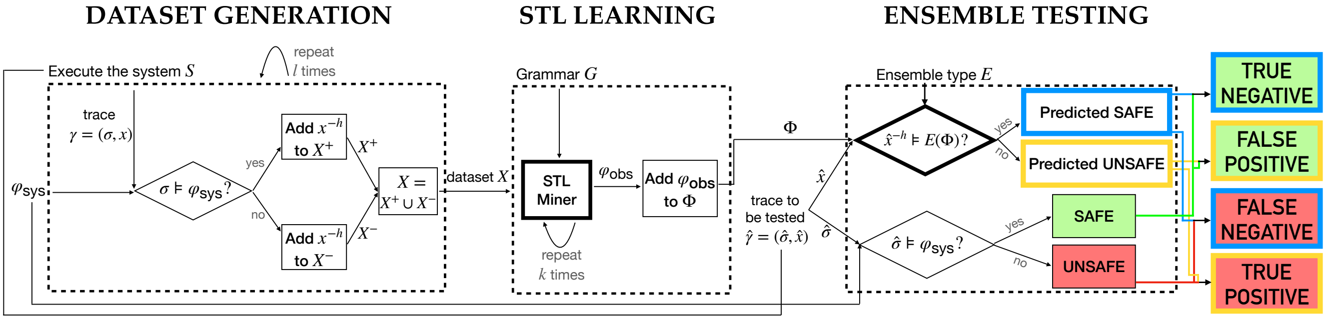


Figure 4.2: Diagram representing the three main steps of our approach.

Problem 1. *Given a specification* $\varphi_{sys} \subseteq \Sigma^*_{sys}$*, a prediction horizon* h*, and a grammar* G *with variables in* V_{obs}*, find a specification* $\varphi_{obs} \in L(G)$*, such that:*

$$\begin{aligned}
&\varphi_{obs} \in \operatorname*{argmin}_{\psi \in L(G)} \\
&\mu_1(\{\Omega(\sigma) \mid \sigma \in S_{sys} \wedge \sigma \models \varphi_{sys} \wedge (\Omega(\sigma))^{-h} \not\models \psi\}) + \\
&\mu_2(\{\Omega(\sigma) \mid \sigma \in S_{sys} \wedge \sigma \not\models \varphi_{sys} \wedge (\Omega(\sigma))^{-h} \models \psi\}),
\end{aligned} \tag{4.1}$$

where $\mu_1, \mu_2 \colon \mathcal{P}(S_{obs}) \to \mathbb{R}^+$ *are appropriate measures of cost.*

We define the argument of μ_1 as our set of *false positives*: safe executions that the learned specification predicts to be unsafe. Conversely, the argument of μ_2 will be our set of *false negatives*: unsafe executions misclassified as safe.

In the next section, we introduce a framework for solving the problem for specifications given in terms of STL formulas.

4.5 Methodology

In this section, we describe our approach for mining φ_{obs} from Eq. (4.1) as an STL formula. Figure 4.2 summarizes our three main steps: the dataset generation (sec 4.5.1), the grammar-based STL learning (sec 4.5.2), and the building of the robustness-based ensembles (sec 4.5.3).

4.5.1 Dataset Generation

We structure the set of system executions as a labeled dataset in the following way. For each execution, we collect both the system-level behavior $\sigma \in S_{\mathsf{sys}}$ and the corresponding observations $\Omega(\sigma) = x \in S_{\mathsf{obs}}$ to create the trace $\gamma = (\sigma, x)$. Then, we monitor σ against the system-level specification φ_{sys}: if $\sigma \models \varphi_{\mathsf{sys}}$, we add the trimmed observation x^{-h} to the set of safe examples X^+; whereas, if $\sigma \not\models \varphi_{\mathsf{sys}}$, we add x^{-h} to the set of unsafe examples X^-. The input to our mining method will then be the dataset of size l (defined by the user) containing the trimmed labeled observations of our system: $X = X^+ \cup X^-$, where, for each $z \in X$, we keep the information whether z is a positive ($z \in X^+$) or a negative ($z \in X^-$) example[3]. With this notation, given a candidate STL specification φ, the corresponding set of *false positives* correspond to $F_p(\varphi) := \{z \in X^+ \mid z \not\models \varphi\}$, while the set of *false negatives* to $F_n(\varphi) = \{z \in X^- \mid z \models \varphi\}$. Analogously, the *true positives* are $T_p(\varphi) := \{z \in X^+ \mid z \models \varphi\}$, and the *true negatives* $T_n(\varphi) := \{z \in X^- \mid z \not\models \varphi\}$.

4.5.2 Grammar-Based Mining of STL Monitors

We adopt a grammar-based approach for mining STL properties, inspired by [BMNN23]. In that previous work, the user chooses a (sub)-grammar that defines the set of admissible (e.g., STL) specifications, which are then learned from a set of positive examples. This grammar-based customization allows to encode prior knowledge, making the inferred specifications more application-specific. Our work advances this approach by addressing the need for binary classification, which is essential for distinguishing safe executions from unsafe ones. We therefore introduce a procedure that mines STL specification classifiers from both positive and negative examples. Our method integrates [BMNN23] with a Simulated Annealing [KGV83] approach with the goal of solving the optimization problem defined in Eq.(4.1) and finding an approximation φ^* for φ_{obs} (we will detail our choice of the cost measures in section 15).

Algorithm 4.1 outlines the main steps in our procedure. We start by uniformly randomly sampling a PSTL formula template among all the possible ones admitted by the grammar G and having the user-defined $l_{\max}$ maximum length. We then instantiate its parameters by computing an approximation of the cost function minimum with the off-the-shelf Powell method [Pow64] implemented in the SciPy [JOP+01] Python library. We store the resulting formula as our initial optimal solution φ^* and enter a loop to refine the candidate formula. This loop ends either when a satisfying STL formula is found (i.e., with cost below a given threshold ε) or after a maximum number of iterations. As standard in Simulated Annealing algorithms, iterations in this loop are characterized by the variable temperature (*temp*), which regulates the typical trade-off between exploration and exploitation throughout the execution of the program. In particular, higher values of *temp* are employed in the initial phases to favor exploration by allowing for bigger changes in the candidate formula and higher probabilities of accepting mutations that worsen its cost. During the execution of the program, the number of iterations with the same value of *temp* increases, while *temp* decreases to gradually shift the goal toward exploitation (characterized by smaller changes that have lower probability to be accepted when increasing the cost).

We apply changes to the candidate formula φ_{par} using mutations [BMNN23]: the PSTL formula is represented as a syntax tree and one of its subtrees is randomly sampled and replaced by a new subtree generated randomly and such that the resulting formula is still

[3]We recall that the set X may be unbalanced, with $|X^+| \gg |X^-|$.

admitted by the grammar G and has maximum length $l_{\max}$. The function *ApplyChanges* we propose here differs from its previous version in two aspects: (i) The value of the temperature variable is affecting the length of the subtree to be removed: for smaller values of *temp* (i.e., toward the final sets of iterations), only smaller subtrees are allowed to be removed; (ii) The new subtree does not have to be of the same length as the older one.

We instantiate the parameters values for the mutated PSTL formula ψ_{par} and update the candidate formula φ to ψ with probability $p = \exp(-\frac{\text{Cost}(\psi,X)-\text{Cost}(\varphi,X)}{temp})$ if ψ has higher cost than φ, and with probability 1 otherwise. Note that p decreases when increasing the cost of ψ or when decreasing *temp*. Such a choice is adopted to favor exploration in the initial phases of the algorithm (when *temp* is high), while, at the same time, discouraging very poor formulas ψ. Finally, at every iteration, the best formula φ^* is updated if and only if a new formula with lower cost is found.

Algorithm 4.1: Grammar-based Simulated Annealing for mining an STL classifier

Input: Set of labeled traces X, STL (sub)grammar G, maximum formula length $l_{\max}$, initial temperature *temp*, cost threshold ε, maximum number of iterations $N_{\max}$

Output: STL formula φ^* generated from G classifying traces in X

$\varphi_{\text{par}} \leftarrow \text{SampleInitialFormula}(G, l_{\max})$
$\varphi \leftarrow \text{InstantiateParameters}(\varphi_{\text{par}}, X)$
$\varphi^* \leftarrow \varphi$
$i \leftarrow 0$
while ***True*** **do**
    for $n_{temp} = 1, \ldots, \mathit{ComputeNumbIter}(temp)$ **do**
        if $(\mathit{Cost}(\varphi^*, X) \leq \varepsilon) \vee (i \geq N_{\max})$ **then return** φ^*
        $\psi_{\text{par}} \leftarrow \text{ApplyChanges}(\varphi_{\text{par}}, G, temp, l_{\max})$
        $\psi \leftarrow \text{InstantiateParameters}(\psi_{\text{par}}, X)$
        if $\mathit{UpdateCandidate}(\varphi, \psi, X, temp)$ **then** $\varphi \leftarrow \psi$
        if $\mathit{Cost}(\psi, X) < \mathit{Cost}(\varphi^*, X)$ **then** $\varphi^* \leftarrow \varphi$
        $i \leftarrow i + 1$
    end
    $temp \leftarrow \text{DecreaseTemp}(temp)$
end

Objective Function

For a given labeled dataset $X = X^+ \cup X^-$ and an STL specification φ, we define the cost $\mathit{Cost}(\varphi, X)$ according to Eq. (4.1) as the sum of a cost on the false positives and a cost on the false negatives, namely $\mathit{Cost}(\varphi, X) := \mu_1(F_p(\varphi)) + \mu_2(F_n(\varphi))$. We want to define the cost measures μ_1 and μ_2 to account for not only the ratio of misclassified traces, but also the *degrees* of such errors in terms of their robustness values.

Formally, we define $\mu_1, \mu_2 \colon \mathcal{P}(S_{\mathsf{obs}}) \to \mathbb{R}^+$ as, for any $Y \in \mathcal{P}(S_{\mathsf{obs}})$, $\mu_1(Y) := \mu(Y, |X^+|)$ and $\mu_2(Y) := \mu(Y, |X^-|)$ where the auxiliary function $\mu \colon \mathcal{P}(S_{\mathsf{obs}}) \times \mathbb{N} \to \mathbb{R}^+$ is defined as

$$\mu(Y, n) := \frac{|Y|}{n} + \frac{1}{2} \cdot \frac{\sum_{y \in Y} |\rho(\varphi; y)|}{|Y|} + 2 \cdot \mathbb{1}_{\left\{\frac{|Y|}{n} > 0.7\right\}}. \tag{4.2}$$

The first term in Eq. (4.2) is standard in binary classification as it is used to evaluate the ratio of misclassified traces. Indeed, when computing $\mu_1(F_p(\varphi)) = \mu(F_p(\varphi), |X^+|)$, this term corresponds to the *ratio of false positives* $\frac{|F_p(\varphi)|}{|X^+|}$, namely the number of traces incorrectly classified as safe over the total number of safe traces. Analogously, the first addend in $\mu_2(F_n(\varphi))$ computes the *ratio of false negatives*.

The second addend corresponds to the half average absolute robustness value of the traces in the input set Y. We interpret the robustness as a confidence measure of the classifier: hence, we would like such a confidence to be low whenever a misclassification occurs (i.e., when $Y = F_p(\varphi)$ or $Y = F_n(\varphi)$). In order to give every variable in φ the same weight, we assume the dataset X had been previously normalized such that each variable V_{obs} varies between 0 and 1. Consequently, it is possible to prove recursively on the definition of ρ that the fraction takes values in $[0, 1]$. We halve this value to place less emphasis on this term relative to others.

Finally, in the third addend, the indicator function, which has value 1 if $\frac{|Y|}{n} > 0.7$ and 0 otherwise, is used to penalize high ratios of misclassified traces. The value of 2 is chosen to prevail over the other two terms (whose sum can be maximum 1.5) in case the condition $\frac{|Y|}{n} \leq 0.7$ is violated. This term introduces a significant jump discontinuity in the objective function, but is fundamental for dealing with highly imbalanced datasets because it avoids that all traces are predicted in the most numerous class.

4.5.3 Robustness-based STL Ensembles

To improve the classification performance, we run Algorithm 4.1 several times and learn a set Φ of k STL formulas (in general, the randomness in our procedure will produce different formulas even for the same training dataset). To predict the safety of the (possibly unseen) execution $\hat{\gamma} = (\hat{\sigma}, \hat{x})$, we evaluate its truncated observation $\hat{x}^{-h}$ with respect to each monitor and combine their verdicts into a single one. To do so, we propose two new voting criteria that leverage the STL quantitative semantics: *Total Robustness Voting* and *Largest Robustness Voting*.

For the *Total Robustness Voting* (TRV), we compute the sum of the robustness values of the formulas in Φ: $R := \sum_{\varphi \in \Phi} \rho(\varphi; \hat{x}^{-h})$. If R is positive, the ensemble TRV will predict that the trace $\hat{\gamma}$ is safe; otherwise, it predicts it to be unsafe. This ensemble method can be seen as an extension of *majority voting* that, instead, assigns each monitor a binary verdict (either 1 or -1). Conversely, we weight each verdict by the confidence level associated with that prediction, as quantified by the robustness function.

The *Largest Robustness Voting* (LRV) selects the STL specification $\bar{\varphi}$ in the set Φ with the largest absolute robustness value: $\bar{\varphi} = \text{argmax}_{\varphi \in \Phi} |\rho(\varphi; \hat{x}^{-h})|$. Then, if $\hat{x}^{-h} \models \bar{\varphi}$, the trace is predicted to be safe, and negative otherwise.

4.6 Evaluation

To assess whether the ensembles of STL formulas that we mine from the observable variables are effective predictive safety monitors and study how sensitive they are to dataset characteristics, we investigate the following research questions:

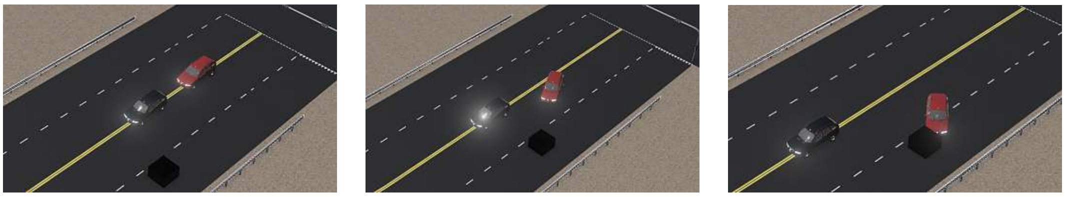
RQ1. [Predictive Performance] *Can our method learn accurate, efficient, and explainable safety predictor monitors?* We use our monitors in safety-critical applications, hence it is essential to evaluate both the accuracy of their predictions and the ease of interpreting their meaning. In addition, it is important to assess the cost-effectiveness of our approach.

Metrics: We use standard metrics for assessing the quality of binary classification systems. Given a binary classifier and a labeled dataset, we compute the corresponding number of true positives tp (correctly predicted safety violations), true negatives tn (correctly predicted safe behaviors), false positives fp (unwarranted alarms), and false negatives fn (missed violations). Consequently, we define:

$$\textbf{Accuracy} := \frac{tp + tn}{tp + tn + fp + fn}; \quad \textbf{Precision} := \frac{tp}{tp + fp};$$
$$\textbf{Recall} := \frac{tp}{tp + fn}; \quad \textbf{F1-Score} := 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{(\text{Precision} + \text{Recall})}$$

Accuracy is the proportion of correctly classified samples; however, since this measure is unreliable for imbalanced datasets, we also consider Precision and Recall. Precision corresponds to the classifier's ability to raise an alert *only* for unsafe executions, whereas Recall is the ability to raise an alert for *all* unsafe executions. The F1-Score, instead, combines them using harmonic mean. All the metrics defined above vary between 0 and 1, with higher values corresponding to better predictors. We assess the explainability of the mined monitors through a manual qualitative analysis and by evaluating their size, as monitors with fewer formulas, conditions, and predicates are intuitively easier to read and analyze. As detailed in section 4.6.1, we contextualize the results by comparing them against those achieved by existing approaches.

RQ2. [Voting Criterion] *Is robustness a suitable voting criterion?* We propose two new voting criteria (TRV and LRV) for STL specifications that leverage robustness values. Since robustness-based voting criteria were never studied before, investigating whether they lead to better predictions than existing criteria is important.



(a) 150 time steps before the end of the execution: the ego car correctly follows the lead.

(b) 100 time steps before the end of the execution: the ego car deviates its trajectory toward the box.

(c) End of the execution: the ego car crashes against the box.

Figure 4.3: Snapshots of three key moments in an unsafe execution of the Lead-Follower case study.

Metrics: We use the metrics defined in RQ1 (i.e., Accuracy, Recall, Precision, and F1-Score) to evaluate whether using robustness as a voting criterion improves predictors' performance.

RQ3. [Sensitivity Analysis] *How sensitive is our method to dataset characteristics?* We use passive learning to infer predictive STL safety monitors from labeled scenario executions. It follows that the training dataset is fixed for the mining task, and we cannot add new training examples. Therefore, the training dataset can have an important influence on the learning process, and we must study its impact on the accuracy of our predictions. More specifically, we study the robustness of the mining process with respect to the size of the dataset, the choice of features, and the degree of imbalance between the safe and unsafe labels in the data. Studying the sensitivity of our predictors concerning the training dataset is important to understand their practical applicability to safety, where nominal cases are much more common than critical ones.

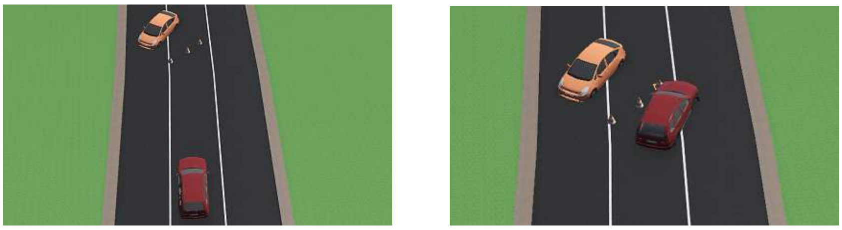
Metrics: We use F1-Score to study how the classification performance varies with different characteristics of the training dataset. We use computational time to study the scalability of our method and ensure it remains effective under different conditions.

4.6.1 Experimental Setting

In this section, we report on the two case studies used in the evaluation and the state-of-art approaches selected as baselines to contextualize our results. Next, we report on the dataset, hyperparameter settings, and other setups.

Case Studies. Our evaluation considers the Lead-Follower (LF) and the Traffic Cones (TC) case studies involving an autonomous vehicle under test (the *ego* car) that must solve a critical task.

The **Lead-Follower** (Figure 4.3) case study was originally presented by Yalcinkaya et al. [YTFS23] in the context of compositional analysis of autonomous systems. This case



(a) Ego car performing *lane keeping*.

(b) Ego car performing *lane changing* too late and hitting one traffic cone.

Figure 4.4: Snapshots from an unsafe execution in TC.

study involves a (black) leading car programmed to drive across intersections and the ego car that must follow it. The ego car autonomously steers by coupling a vision-based component with a controller. The vision-based component analyses the images captured by a front-facing camera mounted on the ego car to estimate the heading difference between the lead and the ego car. The controller acts on the steering wheel to minimize such a difference. As illustrated in Figure 4.3, the ego vehicle is challenged by dark objects (e.g., a black cube) on the road that might cause the vision-based component to wrongly estimate the position of the lead car and eventually crash. To disambiguate critical situations like the one depicted in Figure 4.3 or situations in which the ego car loses sight of the lead vehicle from correct ones, we rely on the following system-level specification: $\varphi_{\mathsf{sys}} := \mathbf{G}((d > 5) \wedge (d < 15))$, where d represents the Euclidean distance between the lead and the ego car. The specification φ_{sys} is violated either when the distance between two vehicles becomes too small ($d \leq 5$) or too large ($d \geq 15$). The first case represents a clear safety issue. The second case indicates that the ego is not able to follow anymore the lead vehicle. While this type of violation does not necessarily represent a safety risk, it arises in situations where the ego vehicle hits an obstacle.

The **Traffic Cone** (Figure 4.4) case study was originally proposed to showcase VerifAI [DFG+19], a platform for the formal design and analysis of learning-enabled cyber-physical systems. According to the original setup, a broken car is on the left side of a three-lane road, and three traffic cones in front of it signal the issue to oncoming traffic. The ego car is approaching in the middle lane and must move to the rightmost lane to avoid the obstacle. The ego car implements vision-based lane-keeping and collision avoidance. To keep the lane, it estimates the distance to the lane center by analyzing images collected from a front-facing camera and adjusting the steering angle accordingly. Simultaneously, it performs object detection using a pre-trained convolutional neural network (CNN) to detect the presence of traffic cones and initiate the lane-changing maneuver if the estimated distance from the traffic cones falls below 15 meters. We consider safe executions those cases in which the ego car detects the traffic cones and changes lanes before touching them. Formally, we rely on the following system-level specification to identify safe executions: $\varphi_{\mathsf{sys}} := \mathbf{G}(d > 0.75)$, where d indicates the Euclidean distance between the ego car and the traffic cones' center, and 0.75 is the radius of standard traffic cones.

Baselines for the Evaluation. To contextualize the results achieved by our STL predictors (TRV and LRV), we compare their results against those of existing works from the literature. Specifically, we consider *MV-dt* by Torfah et al. [TJS+23], who defined a majority voting ensemble over Decision Trees, and two template-free STL miners for binary classification: *STL-dt* by Bombara et al. [BVP+16], based on decision trees, and *STL-enum* by Mohammadinejad et al. [MDP+20b], which learns STL formulas by enumerating PSTL formulas.

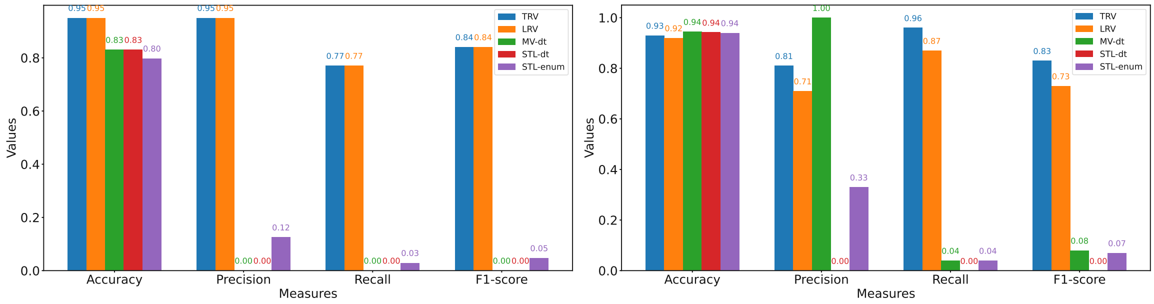
Dataset. To collect the training dataset, we instantiated the LF and TC scenarios randomly. For LF, we randomly placed a colored box on the side of the road; the box's color and position were sampled randomly. Likewise, we randomly sampled the initial coordinates and orientations of the two cars. For TC, we randomly configured the color and rotation of the broken car, the initial velocity, position, and heading of the ego car; additionally, we fuzzed the initial position of the traffic cones.

For each case study, we collected 2000 executions to train the safety predictors and 400 to test them. Unsafe simulations are rare in both scenarios, so the resulting datasets are highly imbalanced: for LF approximately 15% of the traces are unsafe, whereas for TC, unsafe traces account for only 5% of the traces.

Since we did not constrain the scenario execution, traces do not have a fixed length. For instance, the average trace length is 1748 simulation steps in the Lead-Follower case study and 377 simulation steps in the Traffic-Cone case study. *MV-dt*, *STL-dt*, and *STL-enum* require the traces to be all of the same length. Therefore, we pre-processed their training datasets by applying a sliding window of length j, thus generating $|x|-j+1$ new segments for each trace x used for training. We labeled each segment independently depending on whether it satisfied the safety specification φ_{sys}.

Hyperparameters. We mined 10 STL formulas for every ensemble Φ , so each specification was learned from a different *batch* of $\frac{2000}{10} = 200$ training traces. Given the randomness intrinsic to Algorithm 4.1, we repeated every experiment 10 times. We fixed the maximum formula length $l_{\max}$ to 7 (measured as the sum of Boolean/temporal operators and predicates) , the grammar G to the until-free fragment of STL whose predicates are defined over variables V_{obs}, the cost threshold ε to 0.05, and the maximum number of iterations $N_{\max}$ to 50. Finally, we set the prediction horizon h to 100 simulation steps to balance the trade-off between capturing deviations from the desired behavior (which requires more observations) and allowing enough time for corrective actions before a potential violation (which requires prompt intervention).

Software and Hardware Setup. We used Scenic [FKD+22] to define the distributions over the initial conditions, VerifAI [DFG+19] to randomly sample among them and launch the corresponding execution, Webots [Web23] to simulate the system, and RTAMT [NY20]



(a) Lead-Follower (b) Traffic Cones

Figure 4.5: Average values for accuracy, precision, recall, and F1-Score for the two variants of our approach (TRV and LRV), and three methods from the literature: *MV-dt* [TJS+23], *STL-dt* [BVP+16], and *STL-enum* [MDP+20b].

to monitor STL formulas and compute their robustness. We ran the experiments on a non-dedicated cluster with three Intel(R) Xeon(R) Gold 6230R CPU @ 2.10GHz and Ubuntu 20.04 nodes.

4.6.2 RQ1 - Performance Evaluation

To answer RQ1, we learned the monitors using the selected training data, predicted the label of the test data, and computed the accuracy, precision, recall, and F1-Score achieved by the two versions of our approach, TRV and LRV, and the three baselines, *MV-dt*, *STL-dt*, and *STL-enum*. We repeated the experiments 10 times to account for the intrinsic randomness of TRV and LRV in training. We report the average values across the runs in Figure 4.5. Notably, since the three baselines work on trace segmented, we computed their performance metrics as follows: we predicted the label of a trace x by evaluating the monitor against all segments generated from x and classifying x as unsafe if and only if at least one of its segments is predicted unsafe. To select the appropriate value for the sliding window length (j), we conducted a preliminary evaluation varying the value of j among 15, 150, 300 (for TC) and 1500 (for LF). We selected the largest value such that predictors were able to complete training and produced results. In summary, for *MV-dt* we set $j = 15$ for both LF and TC, whereas for *STL-dt* and *STL-enum* we set $j = 1500$ for LF and $j = 300$ for TC [4].

In the LF case study (Figure 4.5a), both TRV and LRV outperformed the existing works on all four metrics. In particular, we notice that although the accuracy of all the methods is comparable, there is a significant difference between the Precision, Recall, and F1-Score of TRV and LRV compared to the three baselines. We explain this remarkable difference in terms of the imbalance of the dataset. Since the majority of the training data are safe, *MV-dt* and *STL-dt* classify all traces as safe; this, in turn, inflates the accuracy but reduces the other metrics. *STL-enum* is slightly more robust with 2 detected true

[4] For *STL-enum* we had to reduce the size of the training batches from 200 to 50.

Table 4.1: Comparison of computational time required to mine an ensemble of 10 monitors (in hours).

	TRV	LRV	MV-dt	STL-dt	STL-enum
LF	16.0 ± 2.2	16.0 ± 2.2	0.97	$1 \cdot 10^{-4}$	1.8
TC	1.3 ± 0.2	1.3 ± 0.2	0.14	$9 \cdot 10^{-5}$	0.5

positives (out of 69); consequently, its precision and recall are greater than zero. However, these values remain too low to be satisfactory. Conversely, TRV and LRV detected 53 (out of 69) true positives on average, so they achieved higher precision, recall, and F1-Score. We make similar observations for the Traffic Cones use case (Fig. 4.5b): although all five methods achieved similar accuracy, they have significantly different distributions of misclassified traces and F1-Score values, with TRV and LRV outperforming the others.

A possible explanation for the poor performance of the three baselines is that they are driven by the misclassification rate, which does not effectively account for the possible imbalances of the dataset. In particular, in our experiments, we observed that *STL-dt* and *STL-enum* returned the first candidate formula they considered: **true** for the former, and a predicate $(p \geq l_p)$ stating that a variable p is greater than its lower bound l_p on the first time step for the latter. These predictors accepted the first candidate formulas because they were satisfied by (almost) all traces in the training dataset, thus achieving a low misclassification rate. On the contrary, our proposed objective function did not lead to cluster all traces into a single class because it strongly penalizes high ratios of false negatives and false positives. However, by doing so, we introduced non-continuity in the objective function, thus making it more challenging to optimize. This drawback is reflected in the high computational time required by our methods, as summarized in Table 4.1. Notably, the significant discrepancy in computational times between the two case studies arises from the longer executions of LF scenarios compared to TC ones (approximately four times longer). We remark that the times reported in Table 4.1 refer to the training phase of the STL monitors. In particular, these learning costs do not impact the time required for making predictions at runtime, which depends on the chosen STL monitor and the number of monitors in the ensemble. Therefore, in practice, this high computational cost does not limit the applicability of our approach in real-time scenarios.

In terms of explainability, we note that the formulas returned by *STL-dt* (**true**) and *STL-enum* $(p \geq l_p)$ are certainly straightforward to interpret, but useless in practice. The trees learned by *MV-dt*, instead, contain between one and two hundred nodes for the Lead-Follower case study and 30 to 120 nodes for the Traffic-Cone case study, making any interpretation of these monitors challenging.

In contrast, our mined STL formulas have a bounded size by construction, making them significantly easier to parse. We remark that short specifications are not necessarily interpretable if they are too abstract. However, our grammar-based approach helps mitigate this risk, as the user can filter the admissible basic predicates in the grammar according to their explainable relevance to the specific application. As a result, the mined formulas can be easily understood, thus providing precious insights into the characterization of safety. To illustrate this point, we report below an example of mined formulas for each case study.

$$\mathbf{F}_{[321:+\infty]}\mathbf{G}(n\_black\_pix < 37.6) \tag{4.3}$$
$$\mathbf{G}\neg\mathbf{G}(\mathbf{F}(n\_orange\_pix < 0) \vee size\_sec\_bb < 373.9) \tag{4.4}$$

Formula 4.3 states that after at least 321 simulation steps, the number of black pixels in the image (n_black_pix) must remain below 37.6 (effectively, below 38); this formula matches our intuition that, in safe executions, the follower should not get too close to any particular black object. Although Formula 4.4 does not look intuitive, we note that it can be simplified using trivial equivalence rules. For instance, using the equivalence $\mathbf{F}\varphi = \neg\mathbf{G}\neg\varphi$, it becomes:

$$\mathbf{GF}(\mathbf{G}(n\_orange\_pix \geq 0) \wedge size\_sec\_bb \geq 373.9).$$

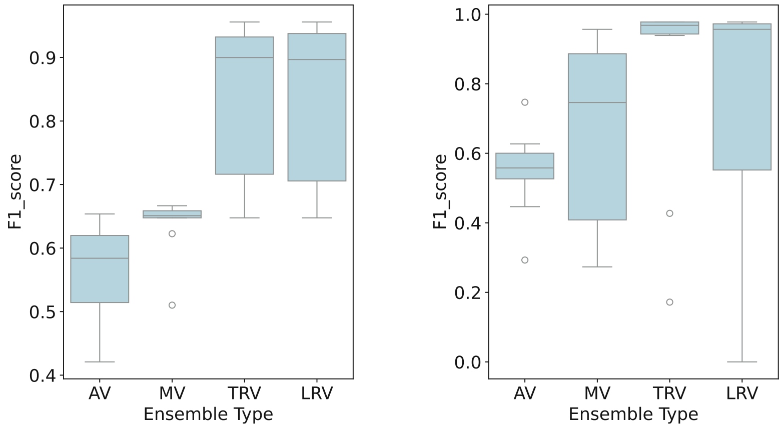
Since the number of orange pixels (n_orange_pix) in an image cannot be negative, we can further simplify the formula to obtain $\mathbf{GF}(size\_sec\_bb \geq 373.9)$, which succinctly captures the lower bound on the size of the second biggest bounding box ($size\_sec\_bb$) returned by the CNN in the final part of the execution. In practice, Formula 4.4 is violated in unsafe executions likely because, toward the end of the execution, the ego car has not yet correctly identified all three traffic cones. As a result, the second biggest bounding box - if even existing - is too small.

The safety monitors generated by our method from imbalanced data outperform the monitors learned from the same data by the existing techniques. Our monitors are also easier to interpret than the decision tree-based methods, although they have a significantly higher computational price.

4.6.3 RQ2 - Voting Criterion

To address RQ2, we compare the performance of our robustness-based ensembles, TRV and LRV, against the traditional voting mechanism of majority voting (MV). For a given set Φ of STL formulas and the trace for which we want to predict safety, MV evaluates the binary outcome of each monitor (safe vs unsafe) and chooses the prediction voted by the majority of the formulas as the final outcome. For completeness, we also compare the performances of TRV and LRF against the average prediction of each individual monitor (AV): in this case, we evaluate each prediction separately without ensembling them and then we take the average value for each performance metric. Thus, AV does not represent a proper voting criterion, but it is still relevant as baseline. For a fair comparison, we use the same set Φ of 10 STL formulas learned via Algorithm 4.1 for all voting mechanisms.

Figure 4.6 plots the distribution of F1-Score over 10 runs of the four ensemble techniques on the two case studies. In the Lead-Follower scenario (Figure 4.6a), we observe that



(a) Lead-Follower (b) Traffic Cones

Figure 4.6: Boxplots of F1-Score values for four kinds of ensemble: AV, MV and the robustness-based TRV and LRV.

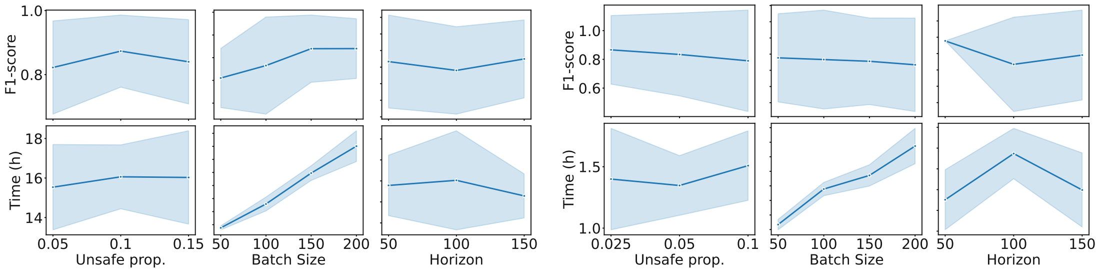
TRV and LRV performed equally well and outperformed MV, while MV outperformed AV. The Traffic Cone case study (Fig. 4.6b) confirms the superior performance of the TRV ensemble. While LRV shows a similar median, its distribution has much greater variability, indicating that LRV's predictions are less reliable. We, therefore, performed a more quantitative analysis to get a better understanding of these results. More specifically, we assessed the statistical significance of the results using the Mann-Whitney U test and measured the strength of the significance using Vargha and Delaney's effect size, expressed with labels according to Kitchenham et al.'s classification [KMB+17]. This analysis confirmed that LRV achieved significantly better performance than MV and AV with *medium* effect size.

> The TRV and LRV ensemble methods, which leverage the robust semantics of STL, significantly increase the prediction accuracy of the safety monitors compared to classical ensemble and averaging methods.

4.6.4 RQ3 - Dataset Characteristics

To evaluate how variations in key characteristics of the training dataset affect the performance of our predictors, we conducted three additional sets of experiments modifying:

- *Imbalance between positive and negative samples.* In the LF case study, we reduced the number of negative samples from 15% to 10% and 5%, whereas in the TC case study, we incremented it from 5% to 10% and then reduced it to 2.5%.
- *Dataset size.* Large training sets usually result in better prediction performance at the price of longer training times and higher costs. We were interested in



(a) Lead-Follower (b) Traffic Cones

Figure 4.7: F1-Score and computational time (in hours) achieved by TRV when varying the proportion of unsafe traces, batch size, or prediction horizon. The line indicates the average values over 10 runs, while the shaded regions their confidence intervals.

understanding how much our predictors' performance drops when using smaller training datasets that entail shorter training procedures. Therefore, we reduced the batch size from 200 to 150, 100, and 50 resulting in training sets of 1500, 1000, and 500 traces.

- *Prediction horizon (h).* We recall that our safety monitors are predictive, i.e., they are trained to detect potential safety violations h steps in advance. The higher the predictive horizon, the more time the system has to take corrective action and avoid safety violations. However, the more difficult it is for the monitor to make an accurate prediction. We studied the effect of the prediction horizon size on the accuracy of our safety monitors. The prediction horizon directly impacts the dataset since we trim h steps from every trace. We experimented with h values set to 50, 100, and 150.

We focused on the best-performing ensemble, i.e., TRV, and investigated how the above dataset characteristics impact its performance. We report in Figure 4.7 the results of our experiments in terms of F1-Score (performance, top-row) and computational time (training effort, bottom row) across 10 runs. The solid lines in the plots mark average values, while the shaded regions identify the confidence intervals around them. In both the Lead-Follower (Figure 4.7a) and Traffic Cones (Figure 4.7b) case studies, we observe that TVR's performance and training time were stable against variations in the imbalance, suggesting the ensemble is not sensitive to them. As expected, training time increased with the size of the training dataset (bottom row, middle plots). However, although the performance increased with larger training datasets in the Lead-Follower case study, this was different in the Traffic Cones. This observation is due to the restrained parameter space and, consequently, the minor variability of concrete scenarios in the Traffic Cones case study. Even a small dataset of simulations generated from this case study sufficiently represents the system.

Finally, when extending the prediction horizon to 150 simulation steps, our method maintains a comparably high F1-Score despite the increased difficulty of earlier predictions.

As expected, the F1-score remains high also for the shorter 50-step horizon since predicting criticality is easier when closer to a potential crash. It is particularly interesting to notice that our method achieves perfect prediction performance in the Traffic Cones case study, suggesting that this time interval may be too short to avoid a crash.

Our mining approach is generally robust to changes in the level of imbalance in the training data and its size, which makes it suitable for practical applications. However, the choice of prediction horizon is application-dependent and may affect the quality of the predictions. Nevertheless, our method has shown stability under significant variations in h.

4.6.5 Threats to Validity

We identified the following main threats that might affect the validity of our conclusions.

Internal Validity Our method can be sensitive to two sources of uncertainty: (i) the stochastic nature of our mining algorithm and (ii) the variability of the training data. To address this, we repeated each experiment 10 times and analyzed the robustness of our algorithm with respect to dataset characteristics. Since our method relies on passive learning, it may be necessary to collect more data to obtain accurate monitors. We mitigated this risk by considering sufficiently large datasets (2000 randomly generated traces) and by studying the accuracy in function of the dataset size.

External Validity Our experiments encounter a limitation in the generalizability of the results, given that we have applied our approach to one domain with synthetically generated datasets. To some extent, we mitigated this threat by selecting two different tasks for the case studies. We plan the application of our method to other cyber-physical domains in future work. We believe that, even though real-world environments may contain more noise and uncertainty, the classification performance of STL monitors will remain satisfactory, as they capture declarative and abstract system properties without overfitting the data. This claim is supported by the experimental results shown in Figure 4.7, where the F1-scores remain highly stable across different case studies and dataset characteristics. Conversely, we expect scalability of computational time to more complex scenarios to be influenced by both the number and length of traces in the dataset, as demonstrated by the variability in the results of the two case studies. Nevertheless, we aim to optimize the current implementation and introduce syntactic rules to infer the satisfaction of STL formulas, thereby skipping unnecessary monitoring steps and accelerating the overall procedure.

Conclusion Validity The accuracy of our conclusions may be affected by the selection of basic predicates and observable variables. We argue that minimal expertise and knowledge are needed for a meaningful mining process. To mitigate this risk, we had the same setup with the other related methods, and, besides selecting the basic predicates, we left the search for STL formulas unrestricted.

4.7 Conclusion and Future Work

We introduced the first approach to learning predictive and explainable monitors for safety in the form of STL specifications from observable variables. Our approach's uniqueness lies in (i) a novel grammar-based method for distinguishing between highly imbalanced sets of positive and negative traces, and (ii) two robustness-based ensemble techniques. Experimental results demonstrated the advantages of our method over the state-of-the-art in two automotive case studies.

In future work, we plan to extend the mining method from passive to active learning by generating new traces that will allow us to refine our inferred specifications and increase their prediction accuracy. With the foreseen active mining approach, we can also provide statistical guarantees about the inferred monitors.

We noted that this paper focuses on autonomous driving applications and safety requirements. However, we foresee applying our framework to other classes of cyber-physical systems and properties defined over system-level variables that are not directly observable by the monitors. We plan to explore case studies from other domains, broadening our framework's applicability.

Finally, we have seen that our mining approach does not optimize the syntax of the specifications. We intend to define syntactic translation rules that will allow us to further simplify the mined specifications before presenting them to the user, making them even more accessible to interpret. We also plan to conduct a more systematic evaluation of explainability, in which users not involved in the grammar definition will examine the mined formulas and asses their understandability.

Acknowledgment This project has received funding from the Austrian FFG ICT of the Future program under grant agreement No 894789 (FATE project), and from the TU Wien-funded Doctoral College for SecInt: Secure and Intelligent Human-Centric Digital Technologies. This work was also partly supported by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation, and by the UC Berkeley iCyPhy center. The Austrian Marshall Plan Foundation funded the Marshall Plan Scholarship, which supported Nesterini's research visit to the University of California Berkeley. Finally, the authors acknowledge TU Wien Bibliothek for financial support through its Open Access Funding Programme.

CHAPTER 5

Mining Hyperproperties using Temporal Logics

Reference. The content of this chapter has been published in [BMNN23]: *Ezio Bartocci, Cristinel Mateis, Eleonora Nesterini, and Dejan Ničković. 2023.* Mining Hyperproperties using Temporal Logics. *ACM Trans. Embed. Comput. Syst. 22, 5s, Article 156 (October 2023), 26 pages.* `https://doi.org/10.1145/3609394`

Abstract. Formal specifications are essential to express precisely systems, but they are often difficult to define or unavailable. Specification mining aims to automatically infer specifications from system executions. The existing literature mainly focuses on learning properties defined on single system executions. However, many system characteristics, such as security policies and robustness, require relating two or more executions, and hence cannot be captured by properties. Hyperproperties address this limitation by allowing simultaneous reasoning about multiple executions with quantification over system traces. In this paper, we propose an effective approach for mining Hyper Signal Temporal Logic (HyperSTL) specifications. Our approach is based on the syntax-guided synthesis framework and allows users to control the amount of prior knowledge embedded in the mining procedure. To the best of our knowledge, this is the first mining method for hyperproperties that does not require a pre-defined template as input and allows for quantifier alternation. We implemented our approach and demonstrated its applicability and versatility in several case studies where we showed that we can use the same method to mine specifications both with and without templates, but also to infer subsets of HyperSTL, including STL, HyperLTL, LTL and non-temporal specifications.

5.1 Introduction

The development of complex systems typically starts from a collection of requirements. Their role is central in the lifecycle of software and embedded systems. In the early stages of the design, specification languages provide the framework to formalize requirements and exchange information between development teams rigorously and unambiguously: formal requirements can precisely capture the expected system behavior. Engineers use these requirements to verify safety-critical properties over a system's model: for example, to test [BMNY23] and debug [BMM+21] its implementation, and to localize the root causes of faults [BFMN18]. Finally, formal specifications can be used to synthesize monitors [BFFR18, BDD+18] that observe the system during its operation and detect violations of requirements, alerting the user or even taking corrective action.

Despite their importance, formal specifications are often partially available. Specification mining [BMNN22] refers to a collection of methods that infer potential system properties by observing its executions. The mainstream approaches in specification mining generate formulas in Linear Temporal Logic [Pnu77] (LTL) and its continuous-time real-valued extensions such as Signal Temporal Logic [MN04] (STL). A recent survey in [BMNN22] summarized the main techniques for learning STL properties, classifying them according to different characteristics. There are two main approaches to mine specifications [BMNN22]: in *template-based* mining [ADMN11] (see also the tool Texada [LPB15, LB15] for LTL), the specification skeleton is given to the mining procedure, which aims at completing the template with missing elements in a way that is consistent with the observed system executions. The missing parts in the template can be propositions or timing and amplitude parameters. In contrast, *template-free* mining infers the entire structure of the formula [JTS+19, NSBB18, MDP+20b, BVP+16], including its parameters.

Most approaches for mining requirements focus on learning trace-based properties from a set of traces. However, some critical system characteristics, such as the system robustness or information-flow security requirements, require simultaneous reasoning about multiple traces and cannot be specified as trace properties. Hyperproperties [CS08] overcome this restraint by allowing specification of properties of trace sets. HyperLTL [CFK+14] and HyperSTL [NKJ+17b] are popular specification languages designed to express an essential class of hyperproperties that extend LTL and STL with trace quantifiers respectively.

This paper considers the problem of mining hyperproperties as HyperSTL/HyperLTL formulae. Only a few works in the literature address this problem and only for a very restricted class of hyperproperties. In particular, none of the existing works allows for quantifier alternation. However, in the literature, there are hyperproperties that require quantifier alternation. For example, the information-flow security policies of *noninterference* and of *generalized noninterference* [CS08] state that for each pair of traces there must exist another trace sharing the same values of secret input variables with the first trace and the same values of public output variables with the second trace. The key idea is that users cannot infer secret information from observing the system's outputs.

Related Work In [SFA17], the authors proposed an approach to learning the relationship between inputs and outputs of different executions of one or more (non-temporal) functions allowing only one possible (universally quantified) template. Another related work is [RMS20] where the authors present a template-based approach to mine HyperLTL specifications. Our technique differs from this method because it allows for quantifier alternation, while [RMS20] relies on enumerating candidate formulas without this feature. In [FHT19], the authors propose an extension of the L* algorithm [Ang87] that actively mines the universally-safe fragment of HyperLTL. This fragment consists of HyperLTL formulas that are universally-quantified and until-free. Thus, this method supports a specification language with limited expressivity compared to our approach.

Our contributions We propose a new procedure for mining hyperproperties from a fixed set of execution traces. The goal is to learn (hyper)properties that this set of positive examples satisfies. The main contributions of our approach are:

- Our work is the first to learn hyperproperties [CS08], expressed in HyperSTL (including formulas with quantifier alternations), STL, LTL, and HyperLTL as special cases. Thus, our approach represents a step forward to the recent state-of-the-art in [BMNN22].
- We have integrated the *syntax-guided synthesis* (SyGuS) [ABJ$^+$13] in the process of mining temporal logic formulas to guide the search in the space of candidate formulas. More specifically, in addition to the execution traces, our mining procedure takes a grammar that defines the search space of valid specifications as additional input. We chose HyperSTL, an expressive specification language for hyperproperties, as the most general template. The grammar can also be restrained, e.g. by bounding the number of trace quantifiers, restricting the use of logical and temporal operators, or fixing the predicate parameters. This flexible mechanism for constraining the space of candidate formulas has multiple advantages. First, it allows the engineer to fine-tune the addition of domain knowledge that helps to infer useful formulas while not over-constraining the search. Second, to the best of our knowledge, this approach is the first one that unifies template-free and template-based mining requirements in temporal logics (the closest related attempt is the work in [NG20] that can handle only quantifier-free first-order logic formulas). Finally, it allows the same method to infer versatile system characteristics such as the correctness properties and the security hyperproperties. In the case of HyperSTL we perform both the parameter and structural syntheses.
- We have extensively evaluated our approach on multiple case studies learning different specifications, including LTL, STL, HyperLTL, HyperSTL, and Hyper-propositional formulas. Furthermore, we compare our results with those obtained from two other state-of-the-art tools: Texada [LPB15, LB15] and Hyper-Miner [RMS20].

- We introduced two heuristics to improve the efficiency of the mining process: (i) we leverage the intrinsic properties of quantifiers to avoid checking all the possible tuples of traces; (ii) we use the correctness properties of HyperSTL/STL to minimize the number of monitoring checks, thus speeding up the overall mining process. As the monitoring of HyperLTL/HyperSTL properties represents an important step in our mining procedure, these heuristics play a crucial step in the applicability of our approach. Authors in [FHST20] proposed an automata-based method to monitor temporal hyperproperties, which is not applicable in our setting as it does not support either HyperSTL or alternating quantifiers.

Outline In Sec. 5.2, we provide the necessary background on HyperSTL [NKJ$^+$17b]. In Sec. 5.3, we present the main steps of our approach to mine HyperLTL formulas, while in Sec. 5.4, we extend it to learn also HyperSTL specifications. Sec. 5.5 provides an extensive assessment of our approach to different case studies, comparing the results obtained with our method with those obtained using other tools. We conclude and discuss future work in Sec. 5.6.

5.2 Preliminaries

Definition 1 (Trace and Set of Traces [NKJ$^+$17b])**.** Let $\mathbb{T} \subseteq \mathbb{R}_{\geq 0}$ be the time domain. A *trace* with q real-valued variables and j Boolean variables $\omega = (t_1, \omega_1), (t_2, \omega_2), \ldots, (t_k, \omega_k)$ is a finite sequence of (time, value) pairs, where $t_i \in \mathbb{T}$, $t_i < t_{i+1}$ for every $i \in \{1, \ldots, k-1\}$, and $\omega_i \in \mathbb{D} = \mathbb{R}^q \times \mathbb{B}^j$ is a vector of real and Boolean values. We denote by $|\omega| = k$ the length of ω. We indicate by $\omega[v]$ the projection of trace ω on variable v. We will slightly abuse notation and also denote by $\omega[v]$ the signal $\mathbb{T} \to \mathbb{R}$ or $\mathbb{T} \to \mathbb{B}$ (depending on whether v is a real or Boolean variable), such that $\omega[v](t) = \omega_i[v]$ if $t \in [t_i, t_{i+1})$ with $i < k$, or $\omega[v](t) = \omega_k[v]$ if $t \geq t_k$. Given a set of traces T and a set of *trace variables* Π, we denote by $\Pi_T : \Pi \to T$ a *trace variable assignment*[1] that maps each trace variable to a concrete trace.

Definition 2 (HyperSTL [NKJ$^+$17b])**.** HyperSTL is a hyper-temporal logic interpreted over real- and Boolean-valued traces that is captured by the following syntax:

$$\begin{array}{rcl} \psi & := & \exists \pi.\psi \mid \forall \pi.\psi \mid \varphi \\ \varphi & := & \pi[b] \mid f(\pi[x_1], \ldots, \pi[x_q]) < c \mid \neg\varphi \mid \varphi_1 \vee \varphi_2 \mid \varphi_1 \mathbf{U}_I \varphi_2 \mid \varphi_1 \mathbf{S}_I \varphi_2 \end{array}$$

where X is a set of real-valued variables, B is a set of Boolean variables, the trace variable $\pi \in \Pi$ ranges over the set $B \cup X$ of variables, $b \in B$, $x_1, \ldots, x_q \in X$, $f : \mathbb{R}^q \to \mathbb{R}$ is a real-valued function, I is an interval in $\mathbb{R}_{\geq 0} \cup +\infty$, and $c \in \mathbb{R}$.

[1]In this work, we assume the *trace variable assignment* is an injective function associating different trace variables with different concrete traces.

Given $t \in \mathbb{T}$, we define the semantics of HyperSTL in terms of a *robustness* function ρ:

$$
\begin{array}{lcl}
\rho(\exists\pi.\psi, \Pi_T, t) & = & \max_{\omega\in T} \rho(\psi, \Pi_T[\pi \rightarrow \omega], t) \\
\rho(\forall\pi.\psi, \Pi_T, t) & = & \min_{\omega\in T} \rho(\psi, \Pi_T[\pi \rightarrow \omega], t) \\
\rho(\neg\varphi, \Pi_T, t) & = & -\rho(\varphi, \Pi_T, t) \\
\rho(\varphi_1 \vee \varphi_2, \Pi_T, t) & = & \max(\rho(\varphi_1, \Pi_T, t), \rho(\varphi_2, \Pi_T, t)) \\
\rho(\varphi_1 \mathbf{U}_I \varphi_2, \Pi_T, t) & = & \sup_{t'\in(t\oplus I)\cap[0,k]} \min(\inf_{t''\in[t,t')} \rho(\varphi_1, \Pi_T, t''), \rho(\varphi_2, \Pi_T, t')) \\
\rho(\varphi_1 \mathbf{S}_I \varphi_2, \Pi_T, t) & = & \sup_{t'\in(t\ominus I)\cap[0,k]} \min(\inf_{t''\in(t',t]} \rho(\varphi_1, \Pi_T, t''), \rho(\varphi_2, \Pi_T, t')) \\
\rho(f(\pi[x_1], \ldots, \pi[x_q]) < c, \Pi_T, t) & = & c - f(\Pi_T(\pi)[x_1](t), \ldots, \Pi_T(\pi)[x_q](t)) \\
\rho(\pi[b], \Pi_T, t) & = & +\infty \text{ if } \Pi_T(\pi)[b](t), -\infty \text{ otherwise}
\end{array}
$$

where $\oplus$ an $\ominus$ denote the Minkowski sum and difference, respectively. We note that we can interpret HyperSTL both over dense and discrete-time em(in this case we consider the interval in the discrete domain), and that the latter interpretation will be used notably when considering HyperLTL and LTL fragments.

From this basic definition of HyperSTL, we can derive the other operators as usual: $\mathbf{true} = \varphi \vee \neg\varphi$, $\mathbf{false} = \neg\mathbf{true}$, $\varphi_1 \wedge \varphi_2 = \neg(\neg\varphi_1 \vee \neg\varphi_2)$, $\varphi_1 \rightarrow \varphi_2 = \neg\varphi_1 \vee \varphi_2$, $\mathbf{F}_I\varphi = \mathbf{true}\ \mathbf{U}_I\varphi$, $\mathbf{G}_I\varphi = \neg\mathbf{F}_I\neg\varphi$, $\mathbf{O}_I = \mathbf{true}\ \mathbf{S}_I\varphi$, $\mathbf{H}_I = \neg\mathbf{O}_I\neg\varphi$. The untimed temporal operators **U**, **S**, **F**, **G**, **O** and **H** are obtained by setting the time interval I to $[0, \infty)$. The *next* and *previous* operators are then (in the discrete-time interpretation of the logic) equivalent to $\mathbf{X}\varphi = \mathbf{F}_{[1,1]}\varphi$ and $\mathbf{Y}\varphi = \mathbf{O}_{[1,1]}\varphi$.

By appropriate restrictions of the HyperSTL syntax, we can also obtain HyperLTL, STL and LTL specification languages. HyperLTL is obtained by restricting the temporal operators to untimed ones only. The STL (LTL) formulas correspond to HyperSTL/HyperLTL variants of the form $\forall\pi.\varphi(\pi)$.

Proposition 1 (Soundness [NKJ+17b]). *Given an HyperSTL formula ψ and a trace variable assignment Π_T, when $\rho(\psi, \Pi_T, t) > 0$ the formula is satisfied, while if $\rho(\psi, \Pi_T, t) < 0$ the formula is violated.*

Proposition 2 (Correctness of STL fragment [DFM13]). *Given an STL formula expressed as the HyperSTL fragment φ (without trace quantifiers) and two traces ω and ω' over the same time domain, if ω at time t satisfies φ and $\|\omega - \omega'\|_\infty < \rho(\varphi, \omega, t)$, then also ω' satisfies φ at time t.*

Proposition 3 (Parametric formula (adapted from [ADMN11])). *An HyperSTL formula φ is* parametric *if at least one of its constants (in the numerical predicates or time bounds of the temporal operator intervals) is replaced by a parameter variable. Given a parametric formula φ with a parameter p, we denote by $\varphi[p \rightarrow v]$ the instantiation of the parameter p in φ with the constant v. A formula φ is said to be* monotonically increasing *in its parameter p_i if, for every pair of parameter valuations v_i and v_i' such that $v_i \leq v_i'$ and for every trace ω, the following condition on the robustness values is satisfied:*

$$\rho(\varphi[p_i \rightarrow v_i], \omega, t) \leq \rho(\varphi[p_i \rightarrow v_i']), \omega, t).$$

Analogously, φ is said to be monotonically decreasing *if the previous condition holds for each pair of parameter valuations v_i and v_i' such that $v_i \geq v_i'$. Monotonicity can be automatically checked using polarity rules in [ADMN11].*

5.3 Mining HyperLTL

In this section, we propose a method for mining HyperLTL that leverages the stochastic search of the syntax-guided synthesis (SyGuS) framework [ABJ+13] developed in the program synthesis research field. In the original setting, SyGuS aims to find a program that satisfies a given correctness requirement among all the acceptable programs defined by the grammar. Similarly, we desire to learn a (hyper)property consistent with a set of positive examples chosen from a group of admissible formulas expressed by the user-defined grammar.

This grammar-based approach to specification mining brings many advantages. By allowing the restriction of the grammar, it admits both template-free and template-based mining. In the latter case, the user can control the number of restrictions the grammar imposes. In this section, we assume HyperLTL as the baseline grammar and consider any additional constraints as a step toward building a template.

We illustrate how we can use basic knowledge about the system under investigation to restrict HyperLTL and steer the search toward meaningful characteristics. For many systems, we know their interface and can distinguish between input and output variables. Many significant properties define the temporal relations between the input and the output behavior. The subclass of HyperLTL that characterizes the input/output relations holding throughout the whole execution can be expressed as follows:

$$\begin{array}{lcl} \psi & := & \exists \pi.\psi \mid \forall \pi.\psi \mid \gamma \\ \gamma & := & \mathbf{G}\varphi[I] \rightarrow \mathbf{G}\varphi[O] \\ \varphi[Y] & := & \pi[b] \mid \neg\varphi[Y] \mid \varphi_1[Y] \wedge \varphi_2[Y] \end{array} \tag{5.1}$$

where B is the set of Boolean variables partitioned into input $I \subset B$ and output $O \subseteq B$ variables, $Y \subset B$ is a subset of variables and $b \in Y$. We can further refine the grammar to restrict the scope of mining to the important hyperproperty of *observational determinism*, stating that the system appears deterministic with respect to its inputs:

$$\forall \pi \forall \pi' \quad \mathbf{G}(\bigwedge_{\mathrm{in} \in I} \pi[\mathrm{in}] = \pi'[\mathrm{in}]) \rightarrow \mathbf{G}(\bigwedge_{\mathrm{out} \in O} \pi[\mathrm{out}] = \pi'[\mathrm{out}]). \tag{5.2}$$

In Section 5.3.1, we describe the algorithm for mining HyperLTL from a set of traces. We propose a measure to quantify the satisfaction of temporal hyperproperties in Section 5.3.2 and an efficient monitoring technique in Sections 5.3.3.

5.3.1 Syntax-Guided Synthesis for Mining Hyperproperties

In Algorithm 5.1, we describe our adaptation of the SyGuS approach [ABJ+13] for mining HyperLTL. The procedure takes as input a grammar G that is a subset of HyperLTL, a set of traces (positive examples) T and the target length l of the mined specification[2].

The algorithm first uniformly samples one candidate specification ψ from the set of all hyperproperties generated by G having m quantifiers and size l (Line 1). Then, ψ is associated with a score s (Line 2), which quantifies the quality of the candidate formula for the dataset T. The score is defined as $s = \exp(0.5 \cdot F(\psi, T))$, where 0.5 is a smoothing value and F is any *fitness function* that gives a (strictly) positive number if ψ is satisfied by T, and a number smaller or equal to zero if it is violated. We propose a concrete fitness function in Section 5.3.2. Alternatively, the Boolean satisfaction returned by a monitoring algorithm (as the one that we propose in Section 5.3.3) can be used as *fitness function.* If ψ is consistent with T, i.e. its score is greater than 1, then ψ is a valid solution and we are done. Else, we iteratively mutate ψ with Algorithm 5.2 (detailed later on) until we either find a variant consistent with T or exceed the maximum number of iterations $N_{\max}$ (Lines 4-13). For the mutated formula ψ', we compute the so-called *Metropolis-Hastings acceptance ratio*, defined as $M(\psi, \psi') = \min\left(1, \frac{\text{score}(\psi')}{\text{score}(\psi)}\right)$. $M(\psi, \psi')$ corresponds to the probability for ψ' to replace the current candidate ψ. The key idea is to always accept changes that improve the score of the current formula, while changes that decrease the score have a certain probability of being rejected (they are not necessarily rejected to allow for exploration). In the limit, the Metropolis-Hastings procedure is guaranteed to sample formulas with probability proportional to their scores. Consequently, this procedure will eventually draw satisfied formulas (if they exist). However, for practical reasons, we impose a stopping condition after $N_{\max}$ iterations, after which we consider the learning process failed. In our experiments, $N_{\max}$ is set to 500.

We now describe the mutation function, summarized in Algorithm 5.2. Given the current specification candidate ψ, we randomly apply one of two equally-probable changes: (i) we modify the quantifiers (Lines 1-4), or (ii) we change the formula structure (Lines 7-13). In the first case, we replace a randomly selected sequence of consecutive quantifiers with a newly sampled sequence of quantifiers of the same length generated according to the grammar rules G. The number of quantifiers must remain unaltered to avoid adding/removing quantified trace variables in the formula structure φ. In the second case, we modify the quantifier-free part of the current formula φ by manipulating its syntax tree $\mathcal{T}_\varphi$; an example of such transformation is depicted in Figure 5.1. We uniformly randomly select one of its nodes (e.g., the *until* node in Figure 5.1) and consider the subtree $\mathcal{S}$ originated by this node (highlighted in red in the figure). From the grammar G, we sample one formula γ having as length the number of nodes in $\mathcal{S}$ (3 in the example) and transform γ into its syntax-tree $\mathcal{T}_\gamma$. Then, we replace the subtree $\mathcal{S}$ inside the tree of the original formula $\mathcal{T}_\varphi$ with $\mathcal{T}_\gamma$ (an example for $\mathcal{T}_\gamma$ is given by the green nodes in Figure 5.2)

[2]The length is affected only by the number of propositions and the operators, but not by the number of quantifiers.

and translate the resulting tree into the formula φ'. Finally, the output specification ψ' has the same quantifiers as the original specification ψ and φ' as quantifier-free formula.

Algorithm 5.1: Stochastic search for HyperLTL mining

Input: HyperLTL (sub)grammar G, set of traces T, formula length l, number of quantifiers m, maximum number of iterations $N_{\max}$
Output: HyperLTL formula ψ generated from G and satisfied by T

```
1  ψ ← sample_hyperproperty(G, l, m)
2  s ← score(ψ, T)
3  i ← 0
4  while s ≤ 1 do
5      if i ≥ N_max then
6          return failure
7      end
8      i ← i + 1
9      ψ' ← mutate(ψ, G)
10     s' ← score(ψ', T)
11     p ← min(1, s'/s)
12     if uniform(0, 1) ≤ p then
13         (s, ψ) ← (s', ψ')
14     end
15 end
16 return ψ
```

Algorithm 5.2: Function mutate(ψ, G)

Input: HyperLTL formula candidate $\psi = Q_1\pi_1 \dots Q_m\pi_m\, \varphi(\pi_1, \dots, \pi_m)$, HyperLTL (sub)grammar G
Output: Mutated formula ψ'

```
if uniform(0, 1) < 0.5 then
    i ← uniform({1, 2, ..., m})
    Q'_i, Q'_{i+1} ... Q'_m ← sample_quantifiers(G, m − i + 1)
    ψ' ← Q_1π_1 ... Q'_iπ_i ... Q'_mπ_m φ(π_1, ..., π_m)
end
else
    T_φ ← formula_to_tree(φ(π_1, ..., π_m))
    S ← select_subtree(T_φ)
    γ ← sample_formula(G, length(S))
    T_γ ← formula_to_tree(γ)
    T'_φ ← replace(T_φ, S, T_γ)
    φ'(π_1, ..., π_m) ← tree_to_formula(T'_φ)
    ψ' ← Q_1π_1 ... Q_mπ_m φ'(π_1, ..., π_m)
end
return ψ'
```

We observe that, in this formulation of the algorithm, the length l and the number of quantifiers m remain fixed during the whole execution. Since, in general, the user will be interested in a set of hyperproperties satisfied by the set of data rather than in a single one, a natural approach is to use different lengths and numbers of quantifiers in different runs, either in a systematic, incremental way or by sampling the values of l and m in intervals of interest.

In terms of complexity, the monitoring process (and consequently the computation of the score - Lines 2 and 10) represents the only expensive step. As we will detail in Sections 5.3.2 and 5.3.3, we require up to $\frac{|T|!}{(|T|-m)!}$ calls to the LTL monitor to compute the score we propose. In the worst-case scenario, Algorithm 5.1 ends after $N_{\max}$ score evaluations, hence yielding an overall computational complexity of $N_{\max} \cdot \frac{|T|!}{(|T|-m)!}$.

Table 5.1: Example of satisfaction (✓) or violation (×) of $\varphi(\pi, \pi')$ by pairs of traces in T.

π \ π'	ω_1	ω_2	ω_3	ω_4
ω_1	-	✓	✓	×
ω_2	×	-	✓	✓
ω_3	✓	×	-	✓
ω_4	×	✓	✓	-

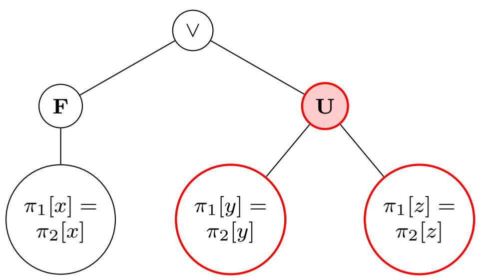


Figure 5.1: Syntax tree for the quantifier-free formula: $\varphi(\pi_1, \pi_2) = \mathbf{F}(\pi_1[x] = \pi_2[x]) \vee ((\pi_1[y] = \pi_2[y])\mathbf{U}(\pi_1[z] = \pi_2[z]))$.

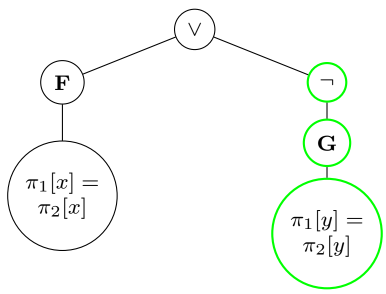


Figure 5.2: Syntax tree for the quantifier-free formula: $\varphi'(\pi_1, \pi_2) = \mathbf{F}(\pi_1[x] = \pi_2[x]) \vee \neg\mathbf{G}(\pi_1[y] = \pi_2[y])$.

5.3.2 Fitness Function

In this section, we propose a *fitness* function $F(\psi, T)$ for quantifying the degree of satisfaction/violation of a HyperLTL specification ψ with respect to T. Consider a HyperLTL formula ψ of the form $Q_1\pi_1 \ldots Q_m\pi_m\, \varphi(\pi_1, \ldots, \pi_m)$. We define $F(\psi, T)$ as the minimum number of m-tuples of traces in T whose satisfaction with respect to φ should be changed in order to change the satisfaction of ψ by T. Note that $F(\psi, T)$ is positive when ψ is satisfied by T, and negative otherwise.

We illustrate this concept with an example. Let $\psi = \forall\pi\exists\pi'\, \varphi(\pi, \pi')$ where $\varphi(\pi, \pi')$ is an LTL formula whose predicates involve the trace variables π and π'. Let $T = \{\omega_1, \omega_2, \omega_3, \omega_4\}$ and let Table 5.1 represent the Boolean satisfaction (✓) or violation (×) of $\varphi(\pi, \pi')$ when trace variables π and π' are replaced by concrete pairs of traces in T.

The quantifiers in ψ impose that for every row in the table (i.e., $\forall\pi$) there is at least one column (i.e., $\exists\pi'$) with the satisfaction symbol ✓. It is evident from Table 5.1 that T satisfies ψ because in every row there are at least two columns marked with ✓. Consequently, at least two changes from ✓ to × are required to violate ψ. For this reason, we have that $F(\psi, T) = 2$.

We now generalize this example and present the algorithm for computing F. We separate the computation of F into two parts: *positive* fitness F^+, which gives a positive value when T satisfies ψ and zero otherwise, and *negative* fitness F^-, which gives negative

Algorithm 5.3: Computation of the *positive* fitness value $F^+(\psi, T)$

Input: HyperLTL formula $\psi = Q_1\pi_1 \ldots Q_m\pi_m\, \varphi(\pi_1, \ldots, \pi_m)$, set of data T with $|T| = n$;
Output: $F^+(\psi, T)$ *Positive* fitness of ψ w.r.t. T;

foreach $(\omega_1, \ldots, \omega_m) \in T^m$ *s.t.* $\omega_i \neq \omega_j$ *if* $i \neq j$ **do**
    if $\varphi(\omega_1, \ldots, \omega_m)$ *holds* **then** $O(\psi; \omega_1, \ldots, \omega_m) \leftarrow 1$
    else $O(\psi; \omega_1, \ldots, \omega_m) \leftarrow 0$
end
for $k = m, m-1, \ldots, 2$ **do**
    foreach $(\omega_1, \ldots, \omega_{k-1}) \in T^{k-1}$ *s.t.* $\omega_i \neq \omega_j$ *if* $i \neq j$ **do**
        if $Q_k = \exists$ **then** $O(\psi; \omega_1, \ldots, \omega_{k-1}) \leftarrow \sum_{l=1}^{n} O(\varphi; \omega_1, \ldots, \omega_{k-1}, \omega_l)$
        else if $Q_k = \forall$ **then**
         $O(\psi; \omega_1, \ldots, \omega_{k-1}) \leftarrow \min_{l=1,\ldots,n} O(\varphi; \omega_1, \ldots, \omega_{k-1}, \omega_l)$
    end
end
if $Q_1 = \exists$ **then**
    $F^+(\psi, T) \leftarrow \sum_{l=1}^{n} O(\varphi; \omega_l)$
end
else if $Q_1 = \forall$ **then**
    $F^+(\psi, T) \leftarrow \min_{l=1,\ldots,n} O(\varphi; \omega_l)$
end
return $F^+(\psi, T)$

value when T violates ψ and zero otherwise. Since the two cases are symmetric, we present only the procedure for computing F^+ in Algorithm 5.3.

Let $n = |T|$ and $\psi = Q_1\pi_1 \ldots Q_m\pi_m\, \varphi(\pi_1, \ldots, \pi_m)$. In the first step, we evaluate for every m-tuple $(\omega_1, \ldots, \omega_m)$ of traces in T whether it satisfies the $\varphi(\pi_1, \ldots, \pi_m)$ formula, in which every trace variable π_i is instantiated with a concrete trace ω_i. This corresponds to a standard LTL membership check. The membership check results are stored in an m-dimensional tensor O. We then iteratively reduce the dimensions of O until we obtain a scalar value, corresponding to the final outcome of Algorithm 5.3. To do so, we associate each $\exists$ quantifier with the sum operator and each $\forall$ quantifier with the minimum operator. Then, starting from the innermost quantifier, we apply its corresponding operator to the vectors of values in O sharing all but the last components. In the example of Table 5.1, since $Q_2 = \exists$, we first sum the number of ✓ in each row, obtaining the vector $O = (2, 2, 2, 2)$. Then, being $Q_1 = \forall$, we take the minimum value in O, which is 2 and corresponds to the final value of $F^+(\psi, T)$.

If F^+ is equal to zero, the hyperproperty is violated, and we need to compute the *negative* fitness F^- to quantify how robust the violation is. The computation of F^- is symmetric to Algorithm 5.3.

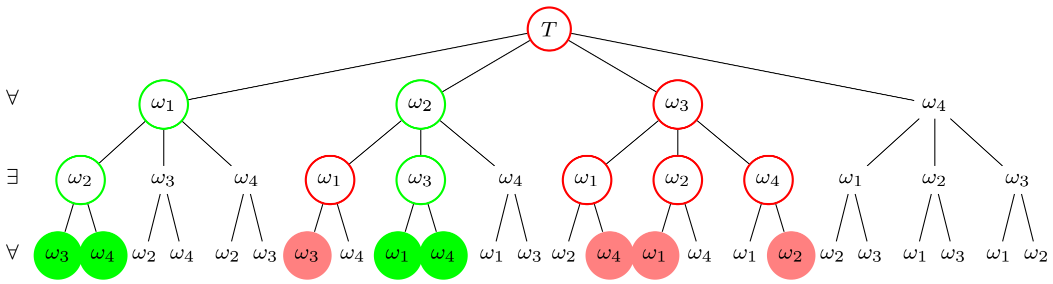


Figure 5.3: Tree representation of the 3-tuples of traces for the dataset $T = \{\omega_1, \omega_2, \omega_3, \omega_4\}$. The evaluation of $\psi = \forall\pi\exists\pi'\forall\pi''\, \varphi(\pi, \pi', \pi'')$ is represented by the color green to mean satisfaction and red for violation.

We observe that Algorithm 5.3 has high computational complexity – it requires $\frac{n!}{(n-m)!} = n \cdot (n-1) \ldots (n-m+1)$ membership checks. Hence, the algorithm has the complexity of $\mathcal{O}(n^m)$, requiring an exponential number of LTL membership checks in the number of the quantifiers.

5.3.3 Using Quantifier Properties for Efficient Monitoring

The goal of the efficient monitoring algorithm we propose is to reduce the practical complexity of Algorithm 5.3 by avoiding unnecessary LTL membership checks. Given a HyperLTL formula $\psi = Q_1\pi_1 \ldots Q_m\pi_m\, \varphi(\pi_1, \ldots, \pi_m)$ and a set T of traces, we devise a method early-stopping criteria based on the semantics of the quantifiers. Our monitoring approach is summarized in Algorithm 5.4.

As a preliminary step, we first use a tree $\mathcal{T}$ to enumerate the tuples of traces in T that have m non-repeated components (we assume that different trace variables refer to different execution traces, but slight changes are sufficient to adapt our approach if this hypothesis does not hold). An example for the dataset $T = \{\omega_1, \omega_2, \omega_3, \omega_4\}$ is depicted in Figure 5.3. We call an *ancestor* of a node p any node encountered when climbing the tree from p to the root, and a *sibling* of a node p any node that shares the same parent as p. The root node of $\mathcal{T}$ is always given by the whole dataset T (we consider it as the node with depth 0) and has $|T| = n$ children nodes, each one representing a trace in T. These nodes make up depth level 1 in the tree and each of them has $n-1$ children, one for each trace in T different from the trace of the node itself. By iterating this process and only allowing a node to have children different from its ancestors, we obtain a tree of depth m and with $\frac{n!}{(n-m)!}$ leaves. Each leaf can be interpreted as an m-tuple of traces, obtained as the sequence of its ancestors ordered from the oldest to the youngest. For example, in Figure 5.3, the leftmost leaf is interpreted as the 3-tuple $(\omega_1, \omega_2, \omega_3)$, while the rightmost leaf as $(\omega_4, \omega_3, \omega_2)$. We conclude the preliminary step by associating each depth level i of the tree with the quantifier Q_i in ψ for all $i = 1, \ldots, m$.

We now describe how Algorithm 5.4 works. The key idea is to minimize the number of evaluations of the property with respect to different tuples of traces by introducing early

Algorithm 5.4: Efficient Monitoring for Temporal Hyperproperties

Input: HyperLTL formula $\psi = Q_1\pi_1 \ldots Q_m\pi_m\, \varphi(\pi_1, \ldots, \pi_m)$, m-tuples of traces of T organized in a tree T;

Output: Boolean outcome: *satisfaction* or *violation* of ψ with respect to T;

```
1  node ← SampleLeafNode(root(T))
2  E(node) ← Evaluate(node, φ)
3  depth ← m
4  while depth > 0 do
5      if (Q_depth = ∀ ∧ E(node) = unsat) ∨ (Q_depth = ∃ ∧ E(node) = sat) then
6          E(Parent(node)) ← E(node)
7          node ← Parent(node)
8          depth ← depth − 1
9      end
10     else if NotEvaluatedSiblings(node) ≠ ∅ then
11         node ← SampleLeafNode(Sample(NotEvaluatedSiblings(node)))
12         E(node) ← Evaluate(node, φ)
13         depth ← m
14     end
15     else if NotEvaluatedSiblings(node) = ∅ then
16         node ← Parent(node)
17         if Q_depth = ∀ then E(node) ← sat
18         else if Q_depth = ∃ then E(node) ← unsat
19         depth ← depth − 1
20     end
21 end
22 return E(root(T))
```

stops as soon as a satisfied witness for the $\exists$ quantifier or a violation for the $\forall$ is found. In particular, we start by evaluating the satisfaction of one uniformly sampled tuple of traces (the function *SampleLeafNode*(x) in Line 1 returns one leaf node of the subtree having as root the node x and the function *Evaluate* in Line 2 calls the LTL monitoring library to compute whether the corresponding tuple of traces satisfies the quantifier-free formula φ). In Figure 5.3, we color a node with green to indicate satisfaction and with red to indicate violation. Depending on the quantifier associated with the depth level of the current node (i.e., m for the first iteration), we might infer the satisfaction or violation of the parent node. In particular (Line 5), if the current quantifier is $\forall$ and the current node is violated, we also associate its parent node with violation since, regardless of the evaluation of its siblings, we already found a witness of the violation. Conversely, if the quantifier is $\exists$ and the node is satisfied, we consider its parent node as satisfied as well, as the current node already demonstrates that at least one satisfied tuple exists. In Figure 5.3, we represent the inferred satisfaction and violation of the nodes by coloring only their contour to underline the fact that their evaluation did not require a call to the LTL monitoring library. If it is not yet possible to infer the satisfaction/violation of the parent node, we proceed by evaluating the satisfaction of one of the siblings of the current node that has not been studied yet (Lines 10-13). When all the siblings have been studied, we can finally infer the status of the parent node: violated if the quantifier is $\exists$ and all siblings are violated, or satisfied if the quantifier is $\forall$ and all the siblings are satisfied (Lines 15-19).

Figure 5.3 shows an example of the execution of Algorithm 5.4. In this case, the 3-tuples of traces that have been evaluated to conclude the violation of ψ with respect to T are 8 out of 24. Nevertheless, we observe that, in the worst-case, there are $\frac{n!}{(n-m)!}$ LTL membership checks required, provoking an exponential complexity in the number of quantifiers m.

5.4 Extension to HyperSTL and Parameter Synthesis

In this section, we describe the extension of the mining procedure for HyperLTL (presented in Section 5.3) to HyperSTL. HyperSTL provides two new features that need to be taken into account: bounded temporal operators and numeric predicates over real-valued variables. When the HyperSTL (timing and amplitude) parameters are fixed and part of the template, Algorithm 5.1 can be used without any adaptation. However, in many common scenarios, this is not the case, as only a range of admissible values for each parameter is known.

We partition the mining procedure into two separate steps: (i) mining the structure of the candidate formula (Algorithm 5.5), and (ii) inferring the parameters appearing in the formula. In this work, we restrict the first step to finding monotonic parametric HyperSTL formula templates. For this reason, we combine Algorithm 5.1 with rejection sampling (Lines 1-4 and 12-15) to ensure that the candidate parametric template ψ_p is monotonic with respect to all its parameters. We automatically check monotonicity using the polarity rules described in [ADMN11]. Whenever a monotonic formula template is found, we instantiate it (Lines 6 and 17) by replacing every parameter symbol with the value in the respective admissible interval that is most likely to be satisfied, namely the upper bound for monotonically increasing parameters and the lower bound for the decreasing ones. Let us consider for example $\psi_p = \forall\pi\exists\pi'.\mathbf{G}_{[0,p_1]}(\pi[x]+\pi'[x] \leq p_2)$ where p_1 varies in the interval $I_1 = [10, 30]$ and p_2 in $I_2 = [0.5, 3]$. We observe that ψ_p is monotonically decreasing in p_1 and monotonically increasing in p_2. Hence, the resulting concrete hyperproperty $\psi = \text{instantiate\_formula}(\psi_p, [I_1, I_2])$ will be $\psi = \forall\pi\exists\pi'.\mathbf{G}_{[0,10]}(\pi[x] + \pi'[x] \leq 3)$.

The evaluation of the score is analogous to the HyperLTL case, with the only difference that the quantifier-free STL formulas require an STL monitor instead of an LTL monitor. When we obtain a candidate HyperSTL formula that is satisfied by the dataset, we perform the second step of the procedure and refine the formula's parameter values.

The parameter synthesis procedure consists in the natural extension to hyperproperties of the combination of monitoring and binary search proposed in [ADMN11]. The goal is

Algorithm 5.5: Mining HyperSTL formula structure

Input: HyperSTL (sub)grammar G, set of traces T, formula length l, number of quantifiers m, maximum number of iterations $N_{\max}$, list of intervals I for the parameters in G

Output: HyperSTL formula ψ generated from G and satisfied by T

$mono \leftarrow$ false
while $mono = false$ **do**
    $\psi_p \leftarrow \text{sample\_hyperproperty}(G, l, m)$
    $mono \leftarrow \text{check\_monotonicity}(\psi_p)$
end
$\psi \leftarrow \text{instantiate\_formula}(\psi_p, I)$
$s \leftarrow \text{score}(\psi, T)$
$i \leftarrow 0$
while $s \leq 1$ **do**
    if $i \geq N_{\max}$ **then return** *failure*
    $i \leftarrow i + 1$
    $mono \leftarrow$ false
    while $mono = false$ **do**
        $\psi'_p \leftarrow \text{mutate}(\psi, G)$
        $mono \leftarrow \text{check\_monotonicity}(\psi'_p)$
    end
    $\psi' \leftarrow \text{instantiate\_formula}(\psi'_p, I)$
    $s' \leftarrow \text{score}(\psi', T)$
    if $uniform(0,1) \leq \min\left(1, \frac{s'}{s}\right)$ **then**
        $(s, \psi, \psi_p) \leftarrow (s', \psi', \psi'_p)$
    end
end
return ψ_p

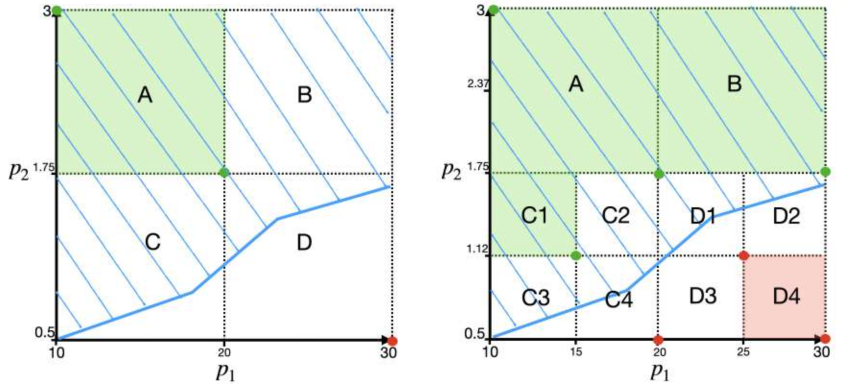
to find parameter values that provide a tight satisfaction of the hyperproperty, meaning that if their values change slightly, the overall satisfaction outcome of the hyperproperty changes too. The reason for such a choice is to find interesting hyperproperties that do not overgeneralize the features of the given dataset. To achieve this goal, we leverage the monotonicity of ψ_p to approximate the region of parameter values that render ψ_p satisfied by the set of traces T and choose the final parameter values as one point on the satisfaction boundary. We describe in more detail such refinement procedure by presenting its application to the previous example.

We denote by k the number of parameter symbols in the formula ($k = 2$ in the example) and by P the *parameter space*, namely the Cartesian product of the intervals of admissible values for the parameters in ψ_p ($P = [10, 30] \times [0.5, 3]$). The procedure is iteratively repeated on hyper-rectangles in P, starting with the whole P as the first hyper-rectangle.

We first consider the point l that is the *least likely to be satisfied* in the current hyper-rectangle, namely the lower bound for monotonically increasing parameters and upper bound for decreasing ones (in the example, $l = [30, 0.5]$). We instantiate ψ_p with l, obtaining $\psi_p[l] = \forall\pi\exists\pi'.\mathbf{G}_{[0,30]}(\pi[x] + \pi'[x] \leq 0.5)$ and we monitor it against the set of traces T. If $\psi_p[l]$ is satisfied - Condition (i) - we could infer the satisfaction of the whole hyper-rectangle. Indeed, since ψ_p is monotonically increasing in p_2, for any other value $l_2 \geq 0.5$, the formula $\psi_p[30, l_2]$ would be satisfied as well, as any values of $\pi[x]$ and $\pi'[x]$ that realize $\pi[x] + \pi'[x] \leq 0.5$ would also realize $\pi[x] + \pi'[x] \leq l_2$. In an analogous way, since ψ_p is monotonically decreasing in p_1, we can deduce the satisfaction of $\psi_p[l_1, 0.5]$ for any $l_1 \leq 30$. Conversely, if $\psi_p[l]$ is violated, we monitor $\psi_p[M]$, being M the point that is *most likely to be satisfied*. If $\psi_p[M]$ is violated - Condition (ii) - we can infer the violation of the entire hyper-rectangle with symmetrical considerations (in the first iteration, this cannot happen because we apply such procedure to the output of Algorithm 5.5, which is guaranteed to be satisfied in at least one point). If none of the conditions (i) and (ii) is realized, we consider the midpoint q on each parameter dimension (for P, $q = [20, 1.75]$), monitor $\psi_p[q]$ and apply the previous reasoning on monotonicity to infer either the satisfaction or violation of one of the 2^k generated hyper-rectangles. Figure 5.4a depicts such process: the blue striped area represents the exact (unknown) satisfaction region, the green area (hyper-rectangle A) is the inferred satisfied region and the white area corresponds to the yet unknown region. The same procedure is then applied to the remaining $2^k - 1 (= 3)$ generated hyper-rectangles (B, C and D in Figure 5.4a) and iteratively repeated. Figure 5.4b depicts the status of the satisfaction region approximation after the second iteration of the binary search algorithm: hyper-rectangle B satisfies condition (i); region C violates condition (i) because its *least satisfiable point* $[20, 0.5]$ is violated and violates condition (ii) because its *most satisfiable point* $[0.5, 1.75]$ is satisfied (such point is shared with the satisfied region A, hence does not require explicit monitoring). C is therefore divided into four regions by its midpoint $[15, 1.125]$. Similarly, also D needs to be partitioned.

The procedure ends either after the area of the unknown region falls below a predefined threshold or after a maximum number of iterations N_b (in our experiments, $N_b = 10$). The final parameter values correspond to a point sampled on the border of the approximated satisfaction region.

We discuss now the complexity of the proposed approach. Starting from the parameter synthesis, we observe that, given a parametric HyperSTL formula with k parameter symbols, the j-th iteration of the binary search algorithm needs at most $3 \cdot (2^k - 1)^{j-1}$ HyperSTL monitors in the worst-case scenario of all hyper-rectangles requiring to be partitioned. Hence, since N_b is the maximum number of iterations, the whole procedure requires up to $3 \cdot \sum_{j=1}^{N_b} (2^k - 1)^{j-1} = 3 \cdot \frac{(2^k-1)^{N_b}-1}{2^k-2}$ HyperSTL monitors. As stated in Sections 5.3.2 and 5.3.3, each HyperSTL monitor corresponds to up to $\frac{|T|!}{(|T|-m)!}$ calls to the STL monitor, hence the parameter refinement requires at most $\frac{|T|!}{(|T|-m)!} \cdot 3 \cdot \frac{(2^k-1)^{N_b}-1}{2^k-2}$ STL membership checks. By summing this value with the complexity of Algorithm 5.5 (equal to the complexity of Algorithm 5.1), we conclude that, in the worst-case scenario,



(a) Approximation of satisfaction region after one binary search iteration.

(b) Approximation of satisfaction region after two binary search iterations.

Figure 5.4: Example of approximation (green area) of the satisfaction region (blue striped area) of ψ_p through binary search algorithm. The white region represents the unknown region, while the red one is the (known) violated region. Colored points represent monitored parameter values (green for satisfaction and red for violation).

our approach needs

$$C = \frac{|T|!}{(|T|-m)!} \cdot \left(N_{\max} + 3\frac{(2^K-1)^{N_b}-1}{2^K-2} \right)$$

calls to the STL monitor, being K the total number of parameter symbols in the grammar G.

We observe that our restriction to monotonic formulas is common in the STL specification mining literature, as monotonicity holds for many relevant properties and, at the same time, it renders much more tractable the approximation of the formula's satisfaction boundary.

5.4.1 Using Robustness for Efficient Monitoring

In this section, we use the correctness property of STL robustness function (Proposition 2) to reduce the number of STL monitoring calls during the evaluation of an HyperSTL formula. Every time we need to call the STL monitor for a tuple of traces (Algorithm 5.4 Line 12), we first check if there exists another already evaluated tuple that differs from the current tuple for just one trace and such that the different traces have distance smaller than the robustness value of the evaluated tuple. If so, we can skip the evaluation of the current tuple, inferring it from the other tuple. To clarify this statement, let us consider an example. Let us suppose we have already monitored the tuple of traces $\Omega = (\omega_1, \omega_2, \omega_3)$ with respect to the quantifier-free formula $\varphi(\pi_1, \pi_2, \pi_3)$ and that Ω satisfies φ with (positive) robustness value $\hat{\rho}$. Whenever we need to monitor another tuple of traces that differs from Ω for only one trace, for example $\Omega' = (\omega_1, \omega_2, \omega_4)$, we compute the infinity-norm distance between the two traces that are different; in this case, $g = \|\omega_3 - \omega_4\|_\infty$. If $g < \hat{\rho}$, we leverage Proposition 2 to infer that also Ω' satisfies φ, skipping its monitoring. The analogous reasoning applies if Ω violates φ with (negative) robustness $\bar{\rho}$: if $g < |\bar{\rho}|$, then Ω' violates φ as well. The vice versa does not hold and requires the explicit monitoring of Ω' with respect to φ.

In terms of complexity, we observe that, similarly to the case of the efficient monitoring algorithm, the proposed heuristic effectively reduces the number of STL monitors in practice, but the worst-case remains unchanged.

5.5 Evaluation

In this section, we start by defining our research questions. We then describe our experimental setup, our case studies and experimental results.

5.5.1 Research Questions

We aim to address these research questions:

RQ1[Performance Evaluation] *To what extent are the mined specifications a good characterization of the system properties according to the following metrics?* We introduce several case studies where traces are generated by different systems and run systematic experiments with two different settings: free and constrained grammar. We analyze the results regarding the ratio of false positive examples (i.e., ratio of mined formulas holding on random traces that do not come from the system), the ratio of false negative examples (i.e., ratio of mined formulas that do not hold on new traces coming from the system), the variability of the mined formulas, and computational time. We show how embedding knowledge through restrictions on the space of admissible formulas generally improves all the mentioned performance metrics.

RQ2[Ability to mine useful specifications] *How useful are the specifications that we infer from the traces?* We perform a qualitative analysis of the mined specifications. More specifically, for each subclass of HyperSTL covered by our case studies, we present and discuss learned formulas representing (i) interesting , (ii) trivial or non-relevant, or (iii) incorrect properties of the system.

RQ3[Flexibility of the mining procedure] *Can we effectively use the same method to mine not only HyperSTL properties but also its subsets, such as HyperLTL, STL, LTL, and even non-temporal properties?* To answer this question, we select several case studies, each requiring a different type of specifications: (i) addition and trigonometric (non-temporal) functions, (ii) LTL temporal tester transducers and dining philosophers, (iii) serial adder and microcontroller systems, and (iv) autonomous parking valet scenario.

RQ4[Scalability] *How does our method scale in terms of the complexity of the formulas and concerning different grammar restrictions?* We analyze the computation time our approach needs when varying the length of the target formula and the constraints imposed on the grammar. We also evaluate the effectiveness of our monitoring algorithm.

RQ5[Comparison with the state-of-the-art] *How does our approach compare to other existing specification mining methods?* To answer this research question, we first identify two appropriate specification mining tools, mainly Texada [LB15] for template-based learning of LTL and HyperMiner [RMS20] for template-based learning of HyperLTL formulas. Since these tools can mine only subsets of specification languages handled by our approach, we select case studies where the comparison is possible and appropriate. We also study whether our more flexible method can infer additional specifications after relaxing the restrictions on the grammar imposed by the other tools.

5.5.2 Experimental Setup and Case Studies

We implemented our approach in the publicly available tool HyTeM[3] (Hyper-Temporal Miner). We also provide the scripts and the seeds to reproduce the results we present. We run the experiments on a scientific cluster based on Intel(R) Xeon(R) Gold 6230R CPU @ 2.10GHz and Ubuntu 20.04.

In the following, we describe the case studies on which we evaluate our method and the relative data generation. For each case study, we also identify one essential and relevant property of the system-under-test, which we call *target formula*. In some experiments, we will restrict the grammar to steer the mining procedure towards learning such property.

Addition Function takes as input two integers x and y and outputs another integer $z = x + y$. Hence, a trace w is stateless and consists of a single valuation (x, y, z). We generated a dataset of 100 traces by sampling 50 random pairs of positive integers (x, y) in the range $[0, 1000]$ and for each such pair computing two traces (x, y, z) and (y, x, z). The target formula for this case study is represented by the *commutativity*:

$$\forall \pi \forall \pi'. (\pi[x] = \pi'[y] \wedge \pi[y] = \pi'[x]) \rightarrow (\pi[z] = \pi'[z]).$$

Trigonometric Functions that we consider in this case study are $\sin x$, $\cos x$, $\arcsin x$ and $\arccos x$, where the input x is a real value bounded to the interval $(-1, 1)$. We generated a dataset of 100 stateless traces of the form $(x, \sin x, \cos x, \arcsin x, \arccos x)$ where x is uniformly sampled from the interval $(-1, 1)$. In this case, one interesting hyperproperty is represented by the *increasing monotonicity* of the arcsin function:

$$\forall x \forall y. (x < y) \leftrightarrow (\arcsin x < \arcsin y).$$

An *LTL Temporal Tester* T_φ for an LTL formula φ is a transducer that realizes the semantics of φ. It reads a (potentially multi-dimensional) trace w and outputs another trace u such that for every $i \geq 0$, $u[i] = 1$ if and only if $(w, i) \models \varphi$. In this case study, we selected five LTL formulas $\mathbf{X}p$, $\mathbf{XX}p$, $\mathbf{G}p$, $\mathbf{F}p$ and $p\mathbf{U}q$. For each formula, we created

[3] https://github.com/eleonoranesterini/HyTeM

a temporal tester in the form of a Mealy machine and collected a dataset of 50 traces, where each trace is a sequence of 1000 valuations generated by randomly exploring the underlying temporal tester. In other words, a temporal tester reproduces the outcome of the monitoring process of a specific LTL specification and we use temporal testers to generate traces that satisfy predetermined LTL operators. For example, for the *next* operator, given a Boolean trace w, we use $T_{\mathbf{X}}$ to generate the Boolean trace u such that, at every time step, $u \leftrightarrow \mathbf{X}(w)$. Hence, for each LTL formula φ, the target formula is the characterization of the temporal tester via the equivalence with the original specification, that is: $\forall\pi.\mathbf{G}(\pi[u] \leftrightarrow \varphi(\pi[w]))$.

Dining Philosophers is a well-known concurrency problem. It consists of philosophers sitting at a round table and sharing one chopstick with each one of the two philosophers sitting next to them. Each philosopher can do exactly one of the following three actions: thinking, being hungry or eating. Since two chopsticks are required to be able to eat, two adjacent philosophers cannot eat at the same time. We consider the log trace of one solution of the dining philosophers' problem for five philosophers, available in the repository of the Texada specification mining tool[4]. Every trace is a sequence of 1088 5-tuple valuations encoding the state of five philosophers. We consider as our target formula one of the templates studied in [LB15]:

$$\forall\pi.\mathbf{G}(\pi[\text{Predicate}_{i,j}] \rightarrow \neg\pi[\text{Predicate}_{i',j'}]), \tag{5.3}$$

where $\pi[\text{Predicate}_{i,j}]$ stands for "*philosopher i performing action j*", with $i \in \{1, 2, \ldots, 5\}$ and $j \in \{\text{thinking, hungry, eating}\}$ in trace π.

Serial Adder with Carry-On is a sequential machine in which the output of the transducer corresponds to the element-wise binary addition with carry-on of two given inputs. Hence, a trace w of the serial adder with carry-on is a sequence of valuations of the form (x, y, c, z), where x and y denote the inputs, c denotes the carry-on bit, z denotes the output, and for every time step $i \geq 1$,

$$w[z][i] = (w[x][i] + w[y][i] + w[c][i-1]) \mod 2,$$

where $w[c][0] = 0$ and, for $i \geq 1$,

$$w[c][i] = \begin{cases} 0 & \text{if } (w[x][i] + w[y][i] + w[c][i-1] \leq 1), \\ 1 & \text{otherwise.} \end{cases}$$

We generated a dataset of 50 traces of 1000 time steps each by randomly creating the sequences of x and y inputs. In this case, the target formula is the *observational determinism* of variable z with respect to x and y:

$$\forall\pi\forall\pi'.\Big(\mathbf{G}(\pi[x] = \pi'[x] \wedge \pi[y] = \pi'[y]) \rightarrow \mathbf{G}(\pi[z] = \pi'[z])\Big).$$

[4] https://github.com/ModelInference/texada/tree/master/traces/dining-philosophers

Figure 5.5: Example of the scene as seen from the camera of the autonomous car. To illustrate how the pedestrian crosses the parking lot, we depicted four pedestrians, although only one is the actual scenario.

Microcontroller is a system-on-chip (SoC) design studied in the context of mining hyperproperties [RMS20]. We use the publicly available dataset[5] that consists of 100 traces generated from a microcontroller design, where each trace consists of a sequence of 15884 35-variable valuations. Since we expect the variables to affect each other deterministically, the target formula for this case study is the *observational determinism* (5.2).

Autonomous Driving Parking is a case study involving an autonomous car driving through a parking lot with a pedestrian that walks out from behind a car onto the path of the vehicle. The car is equipped with an RGB camera and uses YOLOv7, a state-of-the-art image detection deep neural network [WBL22], to detect the pedestrian. As soon as the pedestrian is recognized, the car brakes as hard as possible to avoid the collision. We simulated such scenario with CARLA 9.13 [DRC+17] and collected a sequence of 185 samples containing the coordinates of the velocity v of the car and its distance d to the pedestrian. For each execution, the car's target velocity before braking is fixed to a value sampled between 5.55 and 11.11 mps ($20 - 40$ kph). The length of each execution is of 171 time units (representing 0.05 seconds each). Figure 5.5 shows an example of an image from the vehicle's onboard camera. For this case study, the target formula is the *robustness* hyperproperty:

$\forall\pi\forall\pi'.\mathbf{G}(\|\pi[\text{in}] - \pi'[\text{in}]\| < \varepsilon_1 \rightarrow \|\pi[\text{out}] - \pi'[\text{out}]\| < \varepsilon_2)$. This hyperproperty relates the magnitude ε_1 of the perturbations in the input variables *in* (in this case study, the velocity) with the magnitude ε_2 of the corresponding changes in the output variables *out* (the distance).

Experimental Setup. For each case study, we perform the following steps:

- *Training data generation:* We generate three sets of 50 traces (L_1, L_2, L_3) from the system.

[5] `https://github.com/skmuduli92/TraceExamplesCustomCov/tree/51e3e73d0e507ab1ac44e85a0510118202f73c74`

- *Positive test data generation:* We generate another three sets of 50 traces (P_1, P_2, P_3) from the system.
- *Negative test data generation:* We generate three sets of random 50 traces (N_1, N_2, N_3), hence *not* generated from the system.
- *Mining:* We learn three sets of 50 specifications: S_1, S_2, and S_3 from L_1, L_2 and L_3, and another two sets of specifications S_1' and S_1'' from L_1. We repeat the mining procedure twice, using (i) an unrestricted grammar with up to three quantifiers, and (ii) a template for the target formula.
- *Evaluation:* Since the mined properties cannot be formally verified against the system, the quality of the learned formulas is assessed though sampling and monitoring. More specifically, we evaluate the procedure according to five criteria: (i) the ratio of false positives, (ii) the ratio of false negatives, (iii) the variability of the mined formulas induced by the randomness of the procedure on the same training dataset, (iv) the variability induced by the differences between datasets, and (v) the computational time.

The *ratio of false positives* (RFP) is the proportion of mined formulas that are satisfied by a set of traces not generated from the same system. A high value of RFP (close to 1) indicates that the mined formulas might be too generic and not specific to the given dataset. Similarly, the *ratio of false negatives* (RFN) is the proportion of mined formulas that are violated by a set of traces generated from the same system. A small value of RFN (close to 0) means that the mined hyperproperties do not overfit the learning dataset, as they are valid also on an unseen set of traces. We formalize RFP and RFN as functions of a generic set Q of formulas as follows:

$$\mathrm{RFP}(Q) = \frac{|\{\varphi \mid \varphi \in Q \wedge N \models \varphi\}|}{|Q|}, \quad \mathrm{RFN}(Q) = \frac{|\{\varphi \mid \varphi \in Q \wedge P \not\models \varphi\}|}{|Q|}.$$

We quantify the degree of *variability* V_{same} of the sets of specifications mined using the same set of traces (S_1, S_1', S_1'') and the *variability* V_{diff} of the sets of formulas when different sets of traces are adopted (S_1, S_2, S_3). The two quantities vary between 0 and 1, where the former value indicates that the three sets are all identical and the latter one that they are completely different, not sharing any formula. By comparing V_{same} with V_{diff}, we aim at studying whether the randomness in the output specifications is intrinsic to the mining process (if $V_{\text{same}} \approx V_{\text{diff}}$) or it depends on the training dataset (if $V_{\text{same}} << V_{\text{diff}}$). V_{same} and V_{diff} are defined as:

$$V_{\text{same}} = \frac{|S \setminus S_1| + |S \setminus S_1'| + |S \setminus S_1''|}{6 \cdot \mathrm{average}(|S_1|, |S_1'|, |S_1''|)}, \quad V_{\text{diff}} = \frac{|\hat{S} \setminus S_1| + |\hat{S} \setminus S_2| + |\hat{S} \setminus S_3|}{6 \cdot \mathrm{average}(|S_1|, |S_2|, |S_3|)},$$

where $S = (S_1 \cup S_1' \cup S_1'')$ and $\hat{S} = (S_1 \cup S_2 \cup S_3)$. In terms of computational time, we set a *runout* after 72 hours for the total computation (i.e., five learning runs of 50 formulas each).

Table 5.2: Summary of the experimental results on eleven case studies.

Case Study	Setting	Target formula	RFP	RFN	V_{same}	V_{diff}	Time (hours)	#Formulas
Addition	Constr.	Commutativity	0.5 ± 0.3	0.5 ± 0.35	1	0.95	2.14 ± 1.77	50 ± 0
	Free		0.72 ± 0.04	0.09 ± 0.07	0.97	0.98	0.53 ± 0.19	50 ± 0
Trigonometry	Constr.	Monotonicity	0.11 ± 0.02	0.02 ± 0.02	0.95	0.98	0.018 ± 0.002	50 ± 0
	Free		0.85 ± 0.009	0.04 ± 0.03	1	0.99	0.03 ± 0.02	50 ± 0
Tester Always	Constr.	Equiv. Always	0.19 ± 0.03	0.0 ± 0.0	0.62	0.64	0.05 ± 0.04	50 ± 0
	Free		0.78 ± 0.01	0.02 ± 0.01	0.98	0.99	10.8 ± 6.8	50 ± 0
Tester Eventually	Constr.	Equiv. Eventually	0.18 ± 0.04	0.0 ± 0.0	0.67	0.63	0.05 ± 0.03	50 ± 0
	Free		0.65 ± 0.02	0.02 ± 0.02	1	1	4.8 ± 3.2	50 ± 0
Tester Next	Constr.	Equiv. Next	0.28 ± 0.06	0.0 ± 0.0	0.74	0.69	0.12 ± 0.05	50 ± 0
	Free		0.96 ± 0.01	0.01 ± 0.009	0.97	0.98	6.3 ± 2.7	50 ± 0
Tester NextNext	Constr.	Equiv. NextNext	0.23 ± 0.03	0.0 ± 0.0	0.75	0.75	0.14 ± 0.04	48 ± 2
	Free		0.98 ± 0.01	0.006 ± 0.009	0.97	1	7.6 ± 5.0	50 ± 0
Tester Until	Constr.	Equiv. Until	0.69 ± 0.07	0.0 ± 0.0	0.7	0.64	0.23 ± 0.05	50 ± 0
	Free		0.96 ± 0.0	0.04 ± 0.0	1	1	9.6 ± 3.8	50 ± 0
Philosophers	Constr.	Template in [LB15]	0.54 ± 0.08	0.2 ± 0.1	1	1	0.07 ± 0.04	50 ± 0
	Free		0.72 ± 0.02	0.18 ± 0.04	1	1	0.8 ± 0.9	50 ± 0
Serial Adder	Constr.	Obs Det	1.0 ± 0.0	0.0 ± 0.0	0.42	0.45	0.39 ± 0.12	29 ± 2
	Free		0.91 ± 0.01	0.006 ± 0.009	0.98	0.97	4.9 ± 3.6	50 ± 0
Microcontroller	Constr.	Obs Det	0.98 ± 0.01	0.04 ± 0.06	1	1	10 ± 4	50 ± 0
	Free		-	-	-	-	runout	-
AD Parking	Constr.	Robustness	0.0 ± 0.0	0.12 ± 0.09	0.28	0.41	1.25 ± 0.29	30 ± 2
	Free		0.28 ± 0.05	0.05 ± 0.01	0.99	0.95	12.85 ± 2.348	50 ± 0

The described experiments and evaluations are repeated twice for each case study: one with free grammar (with the only limitation of having at most 3 quantifiers and a maximum formula length of 7) and one with a restricted grammar to steer the search towards the *target formula* of each case study. In particular, we impose the following grammars:

- $\forall\pi\forall\pi'.\varphi_1(\pi,\pi') \rightarrow \varphi_2(\pi,\pi') \,|\, \varphi_1(\pi,\pi') \leftrightarrow \varphi_2(\pi,\pi')$ with $\varphi_1(\pi,\pi')$ and $\varphi_2(\pi,\pi')$ Boolean combinations of predicates defined over trace variables π and π' for the *Addition Function* and the *Trigonometric Functions*;
- $\forall\pi.\mathbf{G}(\varphi(\pi))$, with φ a generic LTL specification for the *Temporal Testers* and the *Philosophers*;
- Grammar defined by Eq. (5.1) but with two universal quantifiers for the *Serial Adder* and the *Microcontroller*;
- $\forall\pi\forall\pi'.\mathbf{G}(\varphi_1(\pi,\pi') \rightarrow \varphi_2(\pi,\pi'))$, where $\varphi_1(\pi,\pi')$ and $\varphi_2(\pi,\pi')$ are two generic STL templates defined over the trace variables π and π' for the *AD Parking*.

5.5.3 Experimental Results

We present here the experimental results that we have carried out, providing an answer to the research questions.

Quantitative Analysis

For each of the 11 case studies, Table 5.2 reports the average and standard deviation over RFP(S_i) for $i = 1, 2, 3$ as the ratio of false positives RFP and over RFN(S_i) as the ratio of false negatives RFN, the variability of the formulas mined on the same trace set V_{same} and on different trace sets V_{diff} and the average and standard deviation of the computational time needed to learn a set of specifications. Before analyzing the table, we point out that we could not generate traces for the *Philosophers* case study as the available dataset consists of a single trace log. Hence, for only this case study, the free setting imposes the use of just one quantifier. Furthermore, to reproduce the experimental setup adopted for the other case studies, we divided the available log trace in three sub-traces of the same length corresponding to $L_1 = P_3$, $L_2 = P_1$, $L_3 = P_2$ such that L_1, L_2 and L_3 are disjoint sets and so it is each pair (L_i, P_i) for $i = 1, 2, 3$. Moreover, we note that, although the target number of formulas is set to 50 for all executions of the tool, it was not always possible in the constrained setting to learn so many formulas due to the limited number of valid specifications admitted by the restricted grammar. For this reason, the last column of Table 5.2 reports the average and standard deviation of the number of mined formulas over the five learning runs.

From Table 5.2, we first observe that the ratio of false positives RFP is quite high in the free setting of all case studies (from a minimum of 0.28 till a maximum of 0.98), meaning that the learned formulas are probably overgeneralizing the characteristics of the training datasets because they are satisfied also by randomly generated sets of traces. Accordingly, the ratio of false negatives RFN is small, as the mined formulas are satisfied also by the other sets of traces generated by the same system used for the training datasets. If we compare these outcomes with the corresponding ones obtained in the constrained setting, we see that both RFP and RFN improve significantly: from the drop-off of RFP we deduce that the mined specifications are now much tighter and they impose stricter requirements. Since the corresponding value of RFN does not increase but actually decreases, we can conclude that the achieved strictness is not an overfitting of the training data, as the mined formulas are also satisfied by unseen sets of positive examples. Therefore, as expected, the presented results show that the embedding of knowledge into the mining process through restrictions on the grammar is helpful in steering the search towards hyperproperties that are consistent with the system that generated the data, without overgeneralizing the system's properties.

By studying the variability, we observe that the values of V_{same} and V_{diff} for each setting of each case study are always extremely similar. This reveals that the variability in the mined formulas is not significantly affected by the employment of different training sets, rather than by the randomness intrinsic in the learning process. The variability values reported in Table 5.2 are often very high (close to the maximum value of 1), which is undesirable because it implies that different runs may produce highly different results. However, we remark that: (i) The equivalence between different formulas in our study is syntactic and not semantics, hence, for example, $\psi_1 = \forall\pi.(\pi[x] = \pi[y])$ and $\psi_2 = \forall\pi.(\pi[y] = \pi[x])$ are considered as two different specifications. Therefore, it is very likely that the specifications are more homogeneous than they appear and the reported variability values should be considered as upper bounds for the actual semantic variability. (ii) In all case studies (apart from the *Addition* and the *Philosophers*), the constrained setting produces much less variable results than the corresponding free setting. This reveals that one reason behind the high variability in the free learning is the high number of existing satisfied formulas, among which our method draws randomly. Indeed, when the grammar is restricted and the consequent number of admissible formulas is reduced, the variability decreases effectively.

Finally, we observe that the free setting is consistently more computational expensive than the constrained one due to the presence of hyperproperties with three quantifiers. In this case, the monitoring process becomes extremely costly, as the number of LTL/STL monitors in the worst case scenario is exponential in the number of quantifiers. In particular, we registered a *runout* for the free learning in the *Microcontroller* case study. This application is indeed the most expensive one for the high number of variables (35) and the length of the traces (15884 time units).

Qualitative Analysis

In this section, we conduct a qualitative analysis of some of the specifications mined by our approach. For each subclass of HyperSTL, we report (i) one example of interesting specification (possibly different from the target formula of the relative case study); (ii) a trivial or non-relevant formula; (iii) an incorrect formula, which is valid on the training set, but not on the system in general. The variable names refer to the notation introduced for the corresponding case study. The complete list of mined formulas is available online[3].

Hyper-proposition: *Trigonometry*

$$\forall x \forall y.(\sin x > \sin y) \leftrightarrow (\arcsin x > \arcsin y) \tag{5.4}$$

$$\exists x \forall y.\neg(\arccos y > x \leftrightarrow \arccos y \geq \arccos x) \tag{5.5}$$

$$\forall x \forall y.(x \leq \sin y) \leftrightarrow (x < \sin y) \tag{5.6}$$

Specification (5.4) represents an interesting unexpected hyperproperty stating that sin and arcsin are monotonically related in the interval $(-1, 1)$, while (5.5) is not particularly relevant as it just affirms that a possible equivalence does not hold. Finally, formula (5.6) is not valid in general as in the case $x = \sin y$ the proposed equivalence is actually false.

LTL: *Always Temporal Tester*

$$\forall \pi.\mathbf{G}(\pi[u] \rightarrow (\mathbf{F}\pi[w])) \tag{5.7}$$

$$\forall \pi.\mathbf{G}(\pi[w]\mathbf{U}(\pi[u] \leftrightarrow \pi[u])) \tag{5.8}$$

$$\forall \pi.\neg\mathbf{XXG}\pi[w] \tag{5.9}$$

Reminding that, at each time step t, $\pi[u[t]] = 1$ if and only if $\mathbf{G}(\pi[w], t)$ holds, we observe that the LTL formula (5.7) informs us that the satisfaction of the temporal operator *globally* $\mathbf{G}$ implies the satisfaction of *finally* $\mathbf{F}$, which is an important property for LTL semantics. Conversely, (5.8) is a trivial specification as the second term in the *until* $\mathbf{U}$ operator is a tautology, while (5.9) is satisfied by the training dataset but, in general, it may exist a trace $\hat{\pi}$ such that, starting (at least) from the third time step, its variable w is always 1.

HyperLTL: *Serial Adder*

$$\forall\pi\forall\pi'.\mathbf{G}(\pi[x] = \pi'[x] \wedge \pi[y] = \pi'[y]) \rightarrow (\mathbf{G}(\pi[z] = \pi'[z])) \tag{5.10}$$

$$\forall\pi\exists\pi'.\pi[x] = \pi'[x] \tag{5.11}$$

$$\forall\pi\forall\pi'.\mathbf{G}(\pi[y] \neq \pi'[y]) \rightarrow \mathbf{G}(\pi[z] = \pi'[z]) \tag{5.12}$$

Formula (5.10) is extremely relevant as it represents the *observational determinism* of variable z with respect to the two input variables x and y: if two traces agree on the same values for x and y, they have to agree also on the output variable z. We consider the HyperLTL specification (5.11) trivial because it states that for every trace there exists another starting with the same value on the x variable. Since x is Boolean, such information is not very informative. Finally, (5.12) is incorrect, as the Serial Adder can output non-identical traces (z variable), even though their second inputs y are always different.

STL/HyperSTL: *AD Parking*

$$\forall\pi.\mathbf{G}(\pi[v] < 12.75) \tag{5.13}$$

$$\forall\pi\forall\pi'\forall\pi''.(\mathbf{G}(|\pi[d] - \pi'[d]| < 0.1 \vee \pi[d] > 50))\mathbf{U}(\pi''[v] < 0.1) \tag{5.14}$$

$$\exists\pi\forall\pi'.\mathbf{G}(|\pi[d] - \pi'[d]| < 12.57) \tag{5.15}$$

Specification (5.13) is an interesting STL property defining the upper bound of the car's velocity in all executions; such value is consistent with the (supposed unknown) set up of the case study where the car's target velocity varies in the interval $[5.55, 11.11]$ mps. To get a tighter bound on the maximum velocity, the user can decrease the threshold for the uncertainty region area in the parameter synthesis phase (in this application, the threshold was set to one tenth of the input parameter interval). (5.14) is not an interesting HyperSTL formula as at the beginning of every episode the car starts from a standstill, so the velocity is below 0.1 mps. Consequently, the second term for the *until* operator is immediately satisfied and the condition in the first term is irrelevant. To conclude, (5.15) is valid for the traces in the learning dataset, but in principle it is not a requirement of the case study set up, so it may exist a trace whose variable d assumes a value whose distance from the d-values of the other traces is greater than 12.57 meters.

To summarize and to answer **RQ2**, we remark that the presence of incorrect formulas represents the main inevitable limitation of the *passive learning* approaches: the absence of guarantees for the mined properties to hold in general for the system that generated the data. For this reason, a manual inspection of the learned formulas is required. Nevertheless, the mined specifications remain useful to infer new interesting insights into the given set of traces.

Table 5.3: The applicability of the specification mining procedure.

Case Study	Specification
Addition	Hyper-propositional
Trigonometry	Hyper-propositional
Temporal Tester	LTL
Dining Philosophers	LTL
Serial Adder with Carry-On	HyperLTL
Microcontroller	HyperLTL
AD Parking	STL/HyperSTL

Flexibility of the Mining Procedure

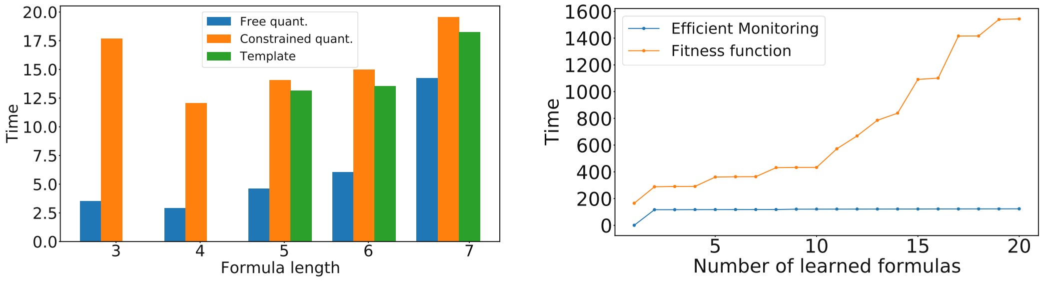
The experimental results show that our mining procedure can be effectively used to infer useful specifications. We have seen that different case studies have different requirements. For instance, certain systems operate in a stateless manner, while others are sequential. Some have properties that can be described based on single executions, while others necessitate reasoning about multiple executions. We have demonstrated that we can effectively use our procedure in a flexible manner to address this large variety of requirements. Table 5.3 summarizes this observation and shows that we were able to use our approach to mine (i) stateless hyperproperties, (ii) LTL, (iii) HyperLTL, (iv) STL, and (v) HyperSTL. To the best of our knowledge, we also developed the first approach that is able to mine properties with quantifier alternation. This flexibility is achieved by putting appropriate restrictions on the grammar that is explored by our procedure. To summarize, we give a positive answer to the research question **RQ3**.

Scalability and Computational Efficiency

In this section, we study the computational efficiency of our mining procedure and how it scales. Consequently, we perform several experiments and report the results.

We first perform an experiment in the Serial Adder with Carry On case study to analyze the time required to mine HyperLTL formulas. Figure 5.6a shows the comparison of the average time (over 50 runs) needed to learn one formula for fixed formula lengths (from 3 to 7). The plot distinguishes among three different setups: the grammar only imposes the employment of two quantifiers (*free quant.*), the grammar is restricted to the $\forall\forall$ quantifiers (*constrained quant.*) and the search is limited to the input-output relation template given by Eq. (5.1) but with two universal quantifiers (*template*). For all formula lengths, *constrained quant.* requires an amount of time notably greater than the *free* setup. There are at least two explanations for this effect: (i) Intuitively, it is much easier for a formula to hold for a dataset when existential quantifiers are used rather than when only universal quantifiers are employed. In formulas,

$$\forall\pi\forall\pi'\,\varphi(\pi,\pi') \rightarrow \forall\pi\exists\pi'\,\varphi(\pi,\pi')\,,\ \exists\pi\forall\pi'\,\varphi(\pi,\pi') \rightarrow \exists\pi\exists\pi'\,\varphi(\pi,\pi').$$



(a) Comparison of average time (seconds) needed for different grammar restrictions to learn one formula using Algorithm 5.4 when varying the formula length.

(b) Comparison of cumulative time (minutes) needed in the constrained setting to learn 20 different formulas using the *fitness* function versus the effective monitoring algorithm.

Figure 5.6: Analysis of time required by our method in the different setups of the Serial Adder case study.

(ii) With our effective monitoring technique (Algorithm 5.4), it is faster to monitor formulas with existential quantifiers rather than universal ones. Indeed, in the former case, as soon as a witness satisfying the specification is found, the monitoring algorithm stops. Conversely, for the $\forall\forall$ quantifiers, all the pairs of traces need to be evaluated to conclude that the overall hyperproperty is satisfied by the dataset[6]. From Figure 5.6a we can also conclude that, as expected, for increasing formula lengths (and consequent complexity), the average time increases as well. Such increase is monotonic in all three settings, with the only exception of length 3 for *constrained quant.* An explanation could be that the relative space of satisfied formulas is very limited and, therefore, it is more time-consuming to sample several formulas in it. Finally, we observe that the *template* setup is comparable to the *constrained quant.* one in terms of time.

The experiments described so far were all carried out using the effective monitoring algorithm (Algorithm 5.4) to compute the score of the candidate hyperproperties. In the next experiment, we use instead the *fitness* function (Algorithm 5.3). Figure 5.6b depicts the comparison in terms of time between the two monitoring algorithms to learn 20 formulas in the constrained-quantifiers scenario, namely when the quantifiers are set to $\forall\forall$. The *fitness* function requires a quantity of time that increases exponentially with the number of learned formulas, while the effective monitoring approach remains almost constant (the mining of the first formulas is the most expensive because at the beginning our method enumerates all admissible formulas for the sampled length; such enumeration is stored and reused in the remaining learning process). The reason for the disparity between *fitness* function and monitoring algorithm is that, for every candidate hyperproperty, the *fitness* function always requires the evaluation of all pairs of traces (that, in this case, are $50 \cdot 49 = 2450$) to return a quantitative value representing the satisfaction or violation of the formula. Conversely, since the effective monitoring algorithm only returns a Boolean outcome, it stops as soon as a violation is detected

[6]We note that the situation is opposite if we are interested in monitoring the property violations.

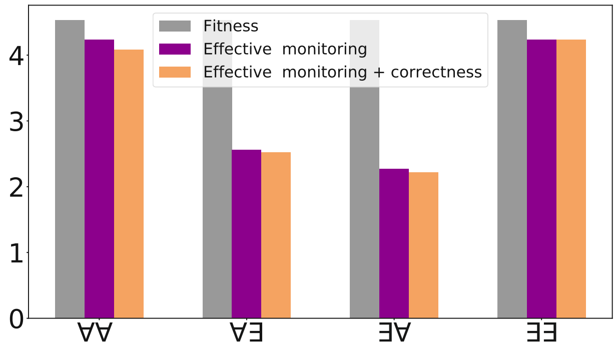


Figure 5.7: Average number (logarithmic scale) of pairs of traces monitored in the AD Parking case study.

and monitors 2450 pairs of traces only in the worst case scenarios. Consequently, it is evident that the *fitness* function does not scale properly and it is hardly usable for learning purposes. However, it is very helpful for a different task: the ranking of the learned formulas. In other words, we can use the *fitness* function to establish which ones in the set of learned formulas are the most interesting ones, in the sense of being the tightest satisfied for the dataset under study. For example, we compute the *fitness* value F for specifications (5.10) and (5.11) and obtain 1 and 23, respectively. These quantitative values confirm our previous manual inspection: formula (5.11) appears quite generic, while the $\forall\forall$ specification (5.10) represents a more specific characterization of the set of traces. Hence, the *fitness* function should be leveraged to replace (or, at least, support) the role played by the user in analyzing the outcomes of the learning method.

Finally, we analyze the performance of the monitoring algorithm (Algorithm 5.4) and its improvement for HyperSTL formulas based on the STL correctness proposition that we introduced in Section 5.4.1. We consider the AD Parking application and compare the number of tuples of traces that have to be monitored to evaluate formulas with two quantifiers for (i) a brute-force monitoring algorithm (e.g., the *fitness* function) that always evaluates all the tuples of traces (in this case, $185 \cdot 184 = 34040$); (ii) the effective monitoring algorithm; (iii) the effective monitoring algorithm with the STL correctness proposition improvement. We sample 80 HyperSTL specifications (20 for each pair of quantifiers), both satisfied and violated by the parking scenario dataset. Figure 5.7 depicts the average number in a logarithmic scale of the number of pair of traces evaluated by each monitoring algorithm. Although in the worst-case scenario the effective monitoring algorithm needs to evaluate all pairs of traces, we observe that, in practice, such number is much smaller and drops of two orders of magnitude in the formulas with alternation of quantifiers. The improvement in alternation-free hyperproperties is more restrained because the satisfied universally-quantified formulas and the violated existentially-quantified ones inevitably require the evaluation of all pairs of traces.

Finally, we observe that the improvement based on the STL robustness correctness proposition further enhances the monitoring algorithm performances, although the most evident speed-up is achieved by the efficient monitoring.

As the answer to **RQ4**, we can observe that we could successfully apply our method to many case studies with reasonable computational effort (Table 5.2). The limitation bottleneck remains the number of monitoring checks that need to be done and that is exponential in the number of formula quantifiers. This limitation is in practice partially addressed by the two heuristics that explain intrinsic robustness and quantifiers properties to avoid many calls to the LTL/STL monitor. Since such heuristics cannot be applied to the *fitness* function, its computation remains significantly slow and hardly applicable to large datasets. Nevertheless, it represents a valid help for the user when the learned formulas need to be analyzed.

Comparison with the State-of-the-Art

We note that the other existing methods in the literature can only mine subsets of specifications that can be handled by our approach. We therefore restrict the grammar accordingly to make the comparison meaningful.

Comparison with Texada. Texada [LPB15, LB15] is a tool for mining LTL formulas from fixed user-defined templates, where only propositional variables can be inferred.

We run experiments for the *Philosophers* case study reproducing the experimental setting in [LB15]. More specifically, we first restrict the grammar to template (5.3) and learn all the intended LTL formulas reported in [LB15] in 4.6 seconds, while Texada required 0.035 seconds.. Such properties express the intrinsic requirements of the Dining Philosophers problem: (i) Two adjacent philosophers cannot eat at the same time (e.g., $\mathbf{G}(\textit{philosopher 1 is eating} \rightarrow \neg\textit{philosopher 2 is eating})$) and (ii) Each philosopher can only perform one action at a time (e.g., $\mathbf{G}(\textit{philosopher 1 is thinking} \rightarrow \neg\textit{philosopher 1 is hungry}))$. In the second step, we restrict the grammar to the other template reported in [LB15]: $\mathbf{F}(\text{Predicate}_{i,j} \wedge \text{Predicate}_{i',j'})$ and effectively mine the five LTL formulas representing the intended property: each pair of non-adjacent philosophers is allowed to eat at the same time. Our method needed 1.08s, while Texada 0.044s.

Finally, to infer new insights from the traces, we run a new experiment with the less restrictive grammar of the form $\forall\pi.\mathbf{G}\varphi(\pi)$, where $\varphi(\pi)$ is an arbitrary LTL formula. The time required to learn 50 formulas for this run is 133s. Among the other mined properties, we find formulas in the form of template (5.3), meaning that our approach can successfully learn this property even without imposing its template. Moreover, we learn another interesting formula: $\gamma = \mathbf{G}(\textit{philosopher 2 is eating} \rightarrow \neg\mathbf{X}(\textit{philosopher 2 is hungry}))$. In words, γ states that, for the second philosopher, it is not possible to change state from eating to being hungry. Inspired by this result, we repeat the experiments by imposing the template of γ and, in 7.6s, we discover that, for any philosopher, the only allowed changes of actions are: *thinking* $\rightarrow$ *hungry* $\rightarrow$ *eating* $\rightarrow$ *thinking*. This final outcome shows that a template-based approach, on the one hand, allows for a focused search on known features, but that, on the other hand, it is not capable of inferring new knowledge.

This is one limitation of Texada that we overcome by allowing the user to run experiments both with and without prior knowledge.

Comparison with HyperMiner. HyperMiner [RMS20] is a template-based alternation-free approach to learn HyperLTL specifications from traces, targeting mainly systems-on-chip (SoC) design. The authors show the applicability of their approach on several SoC designs, including the *Microcontroller* case study considered in this paper. HyperMiner is run using two formula templates: the observational determinism and a similar deterministic dependency that has to hold for each time step. In [RMS20], authors report they need nearly 10 minutes to learn 239 formulas. In this experiment, we qualitatively compare our approach to HyperMiner, by demonstrating that we are able to infer useful specifications beyond the above two templates. Indeed, contrary to [RMS20], our method can learn interesting formulas with quantifier alternation and without a predetermined template. Some examples of such formulas mined with free grammar include:

$$\exists\pi\forall\pi'.(\pi[x] = \pi'[x])$$
$$\forall\pi\forall\pi'\exists\pi''.(\pi[y] = \pi''[y]) \leftrightarrow (\pi'[z] = \pi''[z])$$
$$\forall\pi\forall\pi'\forall\pi''.(\pi'[p] = \pi''[p])\mathbf{U}\pi[q] = \pi'[q]$$

where the variables x is *mem_intrfc_inc_pc*, y is *mem_intrfc_op2_buff*, z is *mem_intrfc_ddat_ir*, p is *mem_intrfc_pcs_source*, and q is *mem_intrfc_imm_r*.

We then consider the experiments with grammar constrained to equations (5.1) and verify that the observational determinism relationship holds between many pairs of variables. In some cases, a variable appears a deterministic function of two variables (instead of only one), such as

$$\forall\pi\forall\pi'.\mathbf{G}\Big(\pi[r] = \pi'[r] \wedge \pi[s] = \pi'[s]\Big) \rightarrow \mathbf{G}\Big(\pi[t] = \pi'[t]\Big),$$

where variables r, s and t correspond to the design variables *mem_intrfc_imm2_r*, *decoder_state* and *mem_intrfc_dmem_wait*. We remark that such hyperproperty could not be learned by [RMS20], as it only supports the one-variable observational determinism template. However, the time required by our approach to learn 50 formulas in this setting is significantly higher: around three hours.

To summarize, our method can reproduce the results of other mining tools while providing much more flexibility and expressiveness. More specifically, we are unaware of any different approach capable of covering both template-based and template-free mining, with arbitrary control of domain knowledge that can be embedded between these two extremes. As expected, our approach is quite slower because its broad applicability prevents it from benefiting from optimizations used by the other tools for specific templates expressed in a fixed specification language. Overall, the presented experiments give a positive answer to **RQ5**.

5.6 Conclusion and Future Work

We introduced the first method that mines HyperSTL and quantifier-alternated HyperLTL specifications from system executions. The method leverages the syntax-guided synthesis allowing the user to embed as much prior knowledge as they want through the definition of the input grammar.

In summary, the main novelties of the paper are:

- Adaptation of SyGuS stochastic search approach to the mining of HyperLTL (Algorithm 5.1) and HyperSTL (Algorithm 5.5);
- Introduction of two heuristics to improve the monitoring of hyperproperties: the efficient monitoring algorithm (Algorithm 5.4) based on the quantifiers semantics and the robustness-based HyperSTL monitoring improvement (Section 5.4.1), which relies on the correctness property of STL quantitative semantics;
- Introduction of the fitness function described by Algorithm 5.3 to quantify the degree of satisfaction/violation of hyperproperties. Although its expensive monitoring renders impractical its implementation in the mining process, it can be leveraged to help the user in assessing the quality of the learned formulas;
- Extensive evaluation on several case studies characterized by different specification languages and analysis of the impact of the domain-knowledge on the quality of the mined formulas.

In future works, we aim to address the limitation of the passive mining approach by developing an active learning method that can steer the generation of new traces from the system. Finally, we are interested in improving the exploration of the formula space by employing different fitness functions, such as the STL robustness, to drive the search.

Acknowledgments This work has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 956123 and it is partially funded by the TU Wien-funded Doctoral College for SecInt: Secure and Intelligent Human-Centric Digital Technologies. The authors acknowledge TU Wien Bibliothek for financial support through its Open Access Funding Programme. Finally, the authors would like to thank Leonardo Mariani for his insightful comments and suggestions on the earlier draft of the paper.

CHAPTER 6

Mining Shape Expressions with ShapeIt

Reference. The content of this chapter has been published in [BDM+21]: *Ezio Bartocci, Jyotirmoy Deshmukh, Cristinel Mateis, Eleonora Nesterini, Dejan Ničković, Xin Qin. 2021.* Mining Shape Expressions with ShapeIt. *In: Calinescu, R., Păsăreanu, C.S. (eds) Software Engineering and Formal Methods. SEFM 2021. Lecture Notes in Computer Science, vol 13085. Springer, Cham.* *https://doi.org/10.1007/978-3-030-92124-8_7*

Abstract. We present ShapeIt, a tool for mining specifications of cyber-physical systems (CPS) from their real-valued behaviors. The learned specifications are in the form of *linear shape expressions*, a declarative formal specification language suitable to express behavioral properties over real-valued signals. A linear shape expression is a regular expression composed of parameterized lines as atomic symbols with symbolic constraints on the line parameters. We present here the architecture of our tool along with the different steps of the specification mining algorithm. We also describe the usage of the tool demonstrating its applicability on several case studies from different application domains.

6.1 Introduction

Specification mining [NSBB18, BBS14a, WCTZ20] is the process of inferring likely system properties from observing its execution and the behavior of its environment. This is an emerging research field that supports the engineering of cyber-physical systems (CPS) where computational units are tightly embedded with physical entities such as sensors and actuators controlling a physical process. CPS often operate (autonomously) in sophisticated and unpredictable environments.

In this context, mined properties can be used to complete existing incomplete or outdated specifications, to understand essential properties of black-box components (e.g., machine learning components) and to automate difficult tasks such as fault-localization [BFMN18, JDDS15], failure explanation [BMM+19] and falsification analysis [BDD+18]. The symbolic and declarative nature of formal specification languages provide an high-level and abstract framework that facilitates generalisation. Furthermore, mined specifications are re-usable, data-efficient, compositional and closer to human understanding.

In this paper, we present ShapeIt, a tool for automatic mining formal specifications from positive examples of time-series data encoding system behaviors or a discrete-time trace of the value of a particular system variable. ShapeIt uses *Linear Shape Expressions* (LSEs) [NQF+19], a recent introduced declarative formalism suitable to express expected behaviors over noisy real-valued signals. A linear shape expression is a regular expression composed of parameterized lines as atomic symbols with symbolic constraints on the line parameters.

Given a set of time-series and a maximum error threshold, ShapeIt implements the specification mining procedure [BDG+20] consisting of three steps: (1) **segmentation** of time-series into an optimal piecewise-linear approximation, (2) **abstraction** and **clustering** of linear segments into a finite set of symbols, where each symbol represent a set of similar lines, and (3) **learning** of linear shape expressions from the sequences of symbols generated in the previous step.

In the rest of the paper, we present the specification language and the architecture of the tool. We also show the usage of our tool, demonstrating the applicability to several different examples of time-series taken from the literature. The code of our tool is publicly available at: `https://www.doi.org/10.5281/zenodo.5569447`.

6.2 Shape Expressions

Linear shape expressions (LSE) [NQF+19] are regular expressions defined over parameterized *linear atomic shapes*, where a linear atomic shape is uniquely determined by three parameters: slope a, (relative) offset b and duration d. LSEs can have additional constraints over these parameters. We use the following syntax to define the fragment of LSEs supported by ShapeIt.

$$\begin{array}{lcl} \mathsf{shape} & := & \mathsf{line}(a, b, d) \mid \mathsf{shape}_1 + \mathsf{shape}_2 \mid \mathsf{shape}_1 \,.\, \mathsf{shape}_2 \mid (\mathsf{shape})* \\ \mathsf{cst} & := & x \text{ in } [\mathbf{c1},\ \mathbf{c2}] \mid \mathsf{cst}_1 \text{ and } \mathsf{cst}_2 \\ \mathsf{SE} & := & \mathsf{shape} : \mathsf{cst} \end{array}$$

where **c1** and **c2** are rational constants such that $\mathbf{c1} \leq \mathbf{c2}$.

A LSE SE consists of two main components, a regular expression shape that captures the qualitative aspect of the specification, and a constraint cst imposed on the LSE parameters. Shape expressions are evaluated against finite signals – sequences of (time, value) pairs. The semantics of LSE is defined in terms of a *noisy match* relation. We say that a signal is a ν-noisy match of a linear atomic shape, if there exists an ideal line segment with some slope a, relative offset b and duration d such that (1) a, b and d satisfy the constraint cst, and (2) the mean square error (MSE) between the signal segment and the ideal line segment is smaller than or equal to ν. This definition is inductively lifted to arbitrary LSEs. In essence, a signal is a ν-noisy match of an arbitrary LSE if there exists a sequence of linear atomic shapes with instantiated parameters such that: (1) the sequence is consistent with the qualitative (regular expression) part of the LSE, (2) the instantiated parameters satisfy the LSE constraint, and (3) the signal can be split into the sequence with the same number of segments, such that each signal segment is a ν-noisy match of its corresponding atomic shape. The formal syntax and semantics of shape expressions are presented in [BDG+20].

6.3 ShapeIt Architecture, Methods and Implementation

The architecture of ShapeIt is depicted in Figure 6.1. The tool consists of five components: (1) segmentation, (2) abstraction, (3) clustering, (4) automata learning and (5) translation from automata to regular expressions. ShapeIt is implemented in Python 3 with the use of external Python and Java libraries.

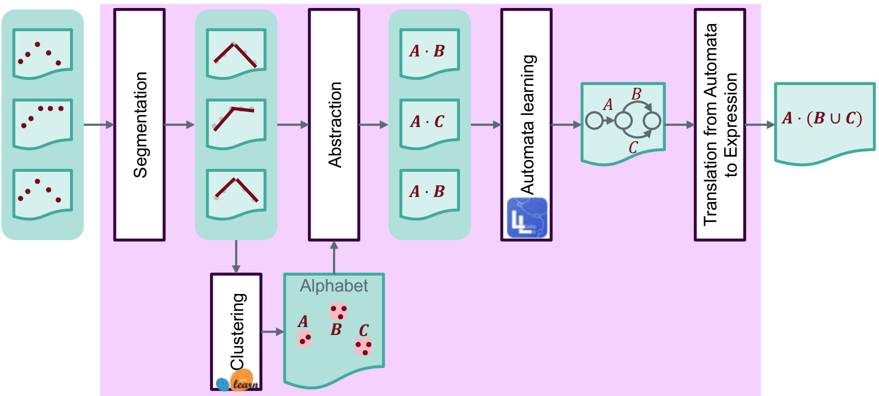


Figure 6.1: Overview of ShapeIt workflow.

Segmentation module implements the piecewise-linear approximation algorithm with quadratic complexity from [BDG+20] that given a time series and a mean square error (MSE) threshold computes the minimal sequence of segments such that for each segment of data, its linear regression MSE is below the threshold. The input of this module is a set of time-series and the output is a set of line segment sequences, where each line segment is characterized by slope, relative offset and duration parameters.

Abstraction and clustering module takes as input the set of line segments (computed by the segmentation module) and uses the k-Means clustering implementation from the scikit-learn library[1] to group lines with similar parameters. The user specifies a threshold on the derivative of the Within-Cluster-Sum-of-Squares (WCSS) error measure to determine the optimal number of clusters. The tool defines a finite alphabet in which each letter is associated to a different cluster. Each letter is also assigned the minimal bounding cube that contains all the points in its corresponding cluster. Each line segment is mapped to a letter in the alphabet, resulting in a set of finite words.

Automata learning module applies the Regular Positive and Negative Inference (RPNI) algorithm for passive learning from positive examples, implemented in the Java learnlib library[2], to infer a deterministic finite automaton (DFA) from a set of finite words. The integration of the Java library in our Python implementation is done using the JPype library.[3]

DFA to shape expressions module implements the algorithm for translating DFAs to regular expressions using the state elimination method. The NetworkX library[4] is used to represent and manipulate DFAs during the translation.

6.4 Evaluation

In the following, we evaluate the applicability of ShapeIt[5] to find temporal patterns over different time-series datasets stored in the UCR Time Series Classification Archive [CKH+15]. Our experiments run on a Notebook Dell Latitude 5320, Intel Quad-Core i7-1185G7 (3,00 GHz/Turbo 4,80 GHz), RAM 32 GB. ShapeIt software components run on Python version 3.8.8 and on Java version 16.0.2. For all the experiments we set to 10 the threshold on the derivative of the WCSS error discussed in Section 6.3.

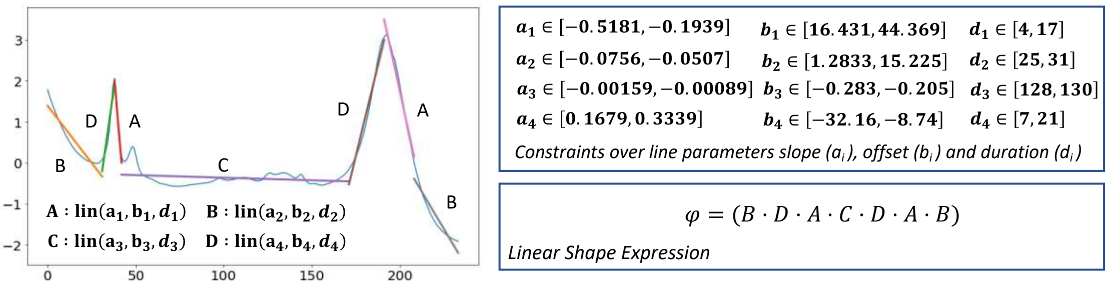


Figure 6.2: (Left) An example of piece-wise linear approximation of a trace in *Wine dataset* with $\varepsilon_{\max} = 0.05$ (Right) Generated Linear Shape Expression.

[1] https://scikit-learn.org/stable/
[2] https://learnlib.de/
[3] https://jpype.readthedocs.io/en/latest/
[4] https://networkx.org/
[5] commit in the repository used: d92341998d66615cf6a9c4f3bcc419df4cd988b6

Table 6.1: Computational cost of ShapeIt.

# traces	$\lvert w \rvert$	$t_s(s)$	$t_c(s)$	$t_l(s)$	$t_{\text{total}}(s)$
1	10	$2.205 \cdot 10^{-4}$	$1.100 \cdot 10^{-6}$	$3.499 \cdot 10^{-4}$	$5.724 \cdot 10^{-4}$
1	100	$7.173 \cdot 10^{-2}$	$4.968 \cdot 10^{-3}$	$3.727 \cdot 10^{-4}$	$7.707 \cdot 10^{-2}$
1	234	$4.227 \cdot 10^{-1}$	$5.195 \cdot 10^{-3}$	$4.319 \cdot 10^{-4}$	$4.283 \cdot 10^{-1}$
10	10	$1.993 \cdot 10^{-3}$	$4.932 \cdot 10^{-3}$	$4.175 \cdot 10^{-4}$	$7.281 \cdot 10^{-3}$
10	100	$7.232 \cdot 10^{-1}$	$5.114 \cdot 10^{-3}$	$7.976 \cdot 10^{-4}$	$7.183 \cdot 10^{-1}$
10	234	4.353	$1.176 \cdot 10^{-2}$	$1.537 \cdot 10^{-3}$	4.366
57	10	$1.21 \cdot 10^{-2}$	$7.594 \cdot 10^{-3}$	$6.122 \cdot 10^{-4}$	$2.039 \cdot 10^{-2}$
57	100	4.110	$2.954 \cdot 10^{-2}$	$2.188 \cdot 10^{-2}$	4.161
57	234	$2.934 \cdot 10$	$2.983 \cdot 10^{-2}$	$4.201 \cdot 10^{-3}$	$2.937 \cdot 10$

Table 6.2: Sensitivity w.r.t. $\varepsilon_{\max}$.

$\varepsilon_{\max}$	φ	# clusters
0.05	$B \cdot\ D \cdot A \cdot C \cdot D \cdot A \cdot B$	4
0.1	$F \cdot\ E \cdot G \cdot I \cdot H \cdot F$	5
0.5	$K \cdot (L + M)$	3

Wine dataset. This dataset [CKH+15] consists of 111 traces, representing the food spectrograph of two kinds of wine. We consider only one class of wine data, containing 57 traces of length 234 samples (Fig. 6.2 shows one example).

By setting the maximum error threshold $\varepsilon_{\max}$ to 0.05 (a little insight into how the learned specification varies depending on this value can be found in Table 6.2), ShapeIt obtains an alphabet of four letters, each one describing a set of segments characterized by the values of slope, relative offset and duration reported in Fig. 6.2.

The concatenation of letters D and A represents the peaks that appear in the shape (see Figure 6.2), in which D describes the rising part (with positive slope) and A the decreasing one (with negative slope). Letter C represents the approximately constant part of the trace that separates the two peaks, while B describes the two extremes (they are both decreasing segments but less steep than the ones that come after the peaks' maxima).

In this particular application the values of slopes would be able to distinguish the different letters by their own: the intervals of slopes are indeed disjoint. The same happens for the relative offset but not for the duration.

ShapeIt generates an LSE specification (see Fig. 6.2) that captures the two main peaks of the trace, but it is not able to recognize the little one that comes immediately after the first peak. The maximum error threshold $\varepsilon_{\max}$ should be reduced if one is interested in detecting also this little curve.

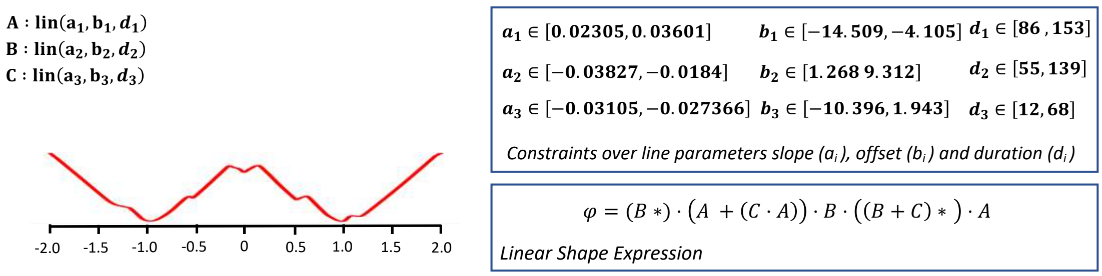


Figure 6.3: (Left) Example of time series generated from the fish contour (see Fig. 20 of [UXKL06] for more details). (Right) Generated Linear Shape Expression.

In Table 6.1, we report the time (expressed in seconds) required by the tool to complete the three different phases: segmentation (t_s), clustering (t_c) and automata learning (t_l). In the last column, t_{total} summarizes the total time needed. Varying the number of traces and their lengths, we can observe that almost always the segmentation represents the most expensive part of the computation, while the clustering and the automata learning can be both considered negligible in terms of computation time. The only exceptions are the two cases in which the total number of traces is 1 or 10 with traces long only 10: the values of t_s, t_c and t_l are comparable since the segmentation is very fast due to the low number of samples to approximate.

In Table 6.2, we compare the specifications learned varying the maximum error threshold $\varepsilon_{\max}$ from 0.05 to 0.5. The number of clusters does not decrease monotonically when increasing the maximum error allowed in the segmentation, while the specifications become shorter and therefore have less explanatory power.

Fish dataset. This data set [CKH+15] is composed by 350 time series representing the shape of seven different species of fishes (chinook salmon, winter coho, brown trout, Bonneville cutthroat, Colorado River cutthroat trout, Yellowstone cutthroat and mountain whitefish). Starting from 50 images for each class, Lee *et al.* in [DJAS+08] generated the data set leveraging a novel technique that transforms the contour of the fish into a time series (see Fig. 6.3 on the left) using a turn-angle function as illustrated in [UXKL06]. Setting to 0.05 the maximum threshold error, with ShapeIt we are able to learn a specification (Fig. 6.3 on the right) from 26 shapes of the same species of fish, each one containing 463 samples.

The concatenation of letters B and A represents the predominant shape in the traces: the triangular repeating behavior where, in particular, A describes the rising part and B the descending one (see Figure 6.3). Letter C is instead used to symbolize the noisy parts, both with positive and negative slope, that eventually separates these longer segments. The choice operator (+) represents the possibility to have multiple symbols or expressions in different time series. Finally, the Kleene star ($*$) is used to indicate that a symbol or an expression can appear zero or more times.

The learned specification provides insights about the relevant shapes in the time series data, displaying them in an human understandable language and therefore offering interpretability to the user. In this example, referring to the fish image in Fig. 6.3 on the left, we can associate the concatenation of letters B and A in the specification to the upper contour of the fish silhouette that is starting from the head and is ending with the tail. Since the same concatenation is then repeated in the specification, we can infer that the contour of the lower part of the fish is not significantly different from the upper one. Finally, letter C can be interpreted as the presence of a big fin that interrupts the predominant lines described by letters A and B.

6.5 Conclusion and Future Work

In this paper, we presented ShapeIt, a tool for mining specifications that describe the behaviors of CPS. The tool requires a set of real-valued signals generated by the system under study as input and it returns as output the specification that better summarize the properties of the traces in the form of linear shape expression. ShapeIt is structured in three phases: segmentation (approximating the traces with segments), abstraction and clustering (grouping lines with similar parameters) and automata learning (learning a DFA from words). Two additional values are needed as inputs to regulate the first two processes: a threshold expressing the maximum error allowed by the approximation and a threshold for the WCSS error to find an optimal number of clusters. We demonstrated the applicability of our tool over two different case studies (*Wine* and *Fish*) but other datasets are present in our repository. These data can be used as well to do experiments and gain confidence with ShapeIt.

As possible future works, we are interested in exploring and learning more general Shape Expressions (not necessarily linear ones), probably gaining explanatory power at the cost of an increasing computation time. We will also study how to automatize the tuning of the two thresholds required by the tool for the segmentation and the clustering phases. In this paper, the segmentation tool finds automatically the optimal number of segments to be used for the approximation, given a maximum error allowed. However it has already been developed to work in the other way round: receiving the number of required segments as input and then finding the approximation that provides the minimum error. It will be therefore interesting to exploit this feature to embed some domain knowledge (in the form of number of segments) in the specification mining process. A step forward will be adding the possibility to set constraints to the parameters of the lines. Finally, an other direction of work could be trying to generalize the tool in order to make it able to handle online processes instead of only offline ones.

Acknowledgments This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 956123 and it is partially funded by the TU Wien-funded Doctoral College for SecInt: Secure and Intelligent Human-Centric Digital Technologies.

Bibliography

[ABJ+13] Rajeev Alur, Rastislav Bodik, Garvit Juniwal, Milo M. K. Martin, Mukund Raghothaman, Sanjit A. Seshia, Rishabh Singh, Armando Solar-Lezama, Emina Torlak, and Abhishek Udupa. Syntax-guided synthesis. In *Proc. of FMCAD 2013*, pages 1–8. IEEE, 2013.

[ABM+23] Edgar A. Aguilar, Ezio Bartocci, Cristinel Mateis, Eleonora Nesterini, and Dejan Ničković. Mining specification parameters for multi-class classification. In Panagiotis Katsaros and Laura Nenzi, editors, *Runtime Verification*, pages 86–105, Cham, 2023. Springer Nature Switzerland.

[ACH+10] C. Ackermann, R. Cleaveland, S. Huang, A. Ray, C. P. Shelton, and E. Latronico. Automatic requirement extraction from test cases. In *Proc. of RV 2010*, volume 6418 of *LNCS*, pages 1–15. Springer, 2010.

[ACM02] Eugene Asarin, Paul Caspi, and Oded Maler. Timed regular expressions. *J. ACM*, 49(2):172–206, 2002.

[AD22] Arvind Adimoolam and Thao Dang. Safety Verification of Networked Control Systems by Complex Zonotopes. *Leibniz Transactions on Embedded Systems*, 8(2):01:1–01:22, 2022.

[ADMN11] Eugene Asarin, Alexandre Donzé, Oded Maler, and Dejan Nickovic. Parametric identification of temporal properties. In *Proc. of RV 2011*, volume 7186 of *LNCS*, pages 147–160. Springer, 2011.

[AFH96] Rajeev Alur, Tomás Feder, and Thomas A. Henzinger. The benefits of relaxing punctuality. *J. ACM*, 43(1):116–146, 1996.

[AG20] Sertaç Kagan Aydin and Ebru Aydin Gol. Synthesis of monitoring rules with STL. *J. Circuits Syst. Comput.*, 29(11):2050177:1–2050177:26, 2020.

[AGD17] Mohammad Al Khatib, Antoine Girard, and Thao Dang. Stability verification and timing contract synthesis for linear impulsive systems using reachability analysis. *Nonlinear Analysis: Hybrid Systems*, 25:211–226, 2017.

[AHFU14] Houssam Abbas, Bardh Hoxha, Georgios Fainekos, and Koichi Ueda. Robustness-guided temporal logic testing and verification for stochastic cyber-physical systems. In *The 4th Annual IEEE International Conference on Cyber Technology in Automation, Control and Intelligent*, pages 1–6, 2014.

[ALFS11] Yashwanth Annpureddy, Che Liu, Georgios Fainekos, and Sriram Sankaranarayanan. S-TaLiRo: A tool for temporal logic falsification for hybrid systems. In *Proc. of TACAS 2011*, volume 6605 of *LNCS*, pages 254–257. Springer, 2011.

[AMF14] Houssam Abbas, Hans Mittelmann, and Georgios Fainekos. Formal property verification in a conformance testing framework. In *2014 Twelfth ACM/IEEE Conference on Formal Methods and Models for Codesign (MEMOCODE)*, pages 155–164, 2014.

[Ang87] Dana Angluin. Learning regular sets from queries and counterexamples. *Information and Computation*, 75(2):87–106, 1987.

[AVBB22] Erfan Aasi, Cristian Ioan Vasile, Mahroo Bahreinian, and Calin Belta. Classification of time-series data using boosted decision trees. In *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1263–1268, 2022.

[BB17] Giuseppe Bombara and Calin Belta. Signal clustering using temporal logics. In *Proc. of RV 2017*, volume 10548 of *LNCS*, pages 121–137. Springer, 2017.

[BB18] Giuseppe Bombara and Calin Belta. Online learning of temporal logic formulae for signal classification. In *Proc. of ECC 2018*, pages 2057–2062. IEEE, 2018.

[BB21] Giuseppe Bombara and Calin Belta. Offline and online learning of signal temporal logic formulae using decision trees. *ACM Trans. Cyber Phys. Syst.*, 5(3):22:1–22:23, 2021.

[BBL+20] Ezio Bartocci, Luca Bortolussi, Michele Loreti, Laura Nenzi, and Simone Silvetti. MoonLight: A lightweight tool for monitoring spatio-temporal properties. In *Proc. of RV 2020*, volume 12399 of *LNCS*, pages 417–428. Springer, 2020.

[BBS14a] Ezio Bartocci, Luca Bortolussi, and Guido Sanguinetti. Data-driven statistical learning of temporal logic properties. In *Proc. of FORMATS 2014*, volume 8711 of *LNCS*, pages 23–37. Springer, 2014.

[BBS+14b] Sara Bufo, Ezio Bartocci, Guido Sanguinetti, Massimo Borelli, Umberto Lucangelo, and Luca Bortolussi. Temporal logic based monitoring of assisted ventilation in intensive care patients. In *Proc. of ISoLA 2014*, volume 8803 of *LNCS*, pages 391–403. Springer, 2014.

[BDD+18] 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 - Introductory and Advanced Topics*, volume 10457 of *LNCS*, pages 135–175. Springer, 2018.

[BDG+20] Ezio Bartocci, Jyotirmoy Deshmukh, Felix Gigler, Cristinel Mateis, Dejan Nickovic, and Xin Qin. Mining shape expressions from positive examples. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.*, 39(11):3809–3820, 2020.

[BDM+21] Ezio Bartocci, Jyotirmoy Deshmukh, Cristinel Mateis, Eleonora Nesterini, Dejan Nickovic, and Xin Qin. Mining shape expressions with ShapeIt. In *Proc. of SEFM 2021*, volume 13085 of *LNCS*, pages 110–117. Springer, 2021.

[BFFR18] Ezio Bartocci, Yliès Falcone, Adrian Francalanza, and Giles Reger. Introduction to runtime verification. In *Lectures on Runtime Verification - Introductory and Advanced Topics*, volume 10457 of *LNCS*, pages 1–33. Springer, 2018.

[BFMN18] Ezio Bartocci, Thomas Ferrère, Niveditha Manjunath, and Dejan Nickovic. Localizing faults in simulink/stateflow models with STL. In Maria Prandini and Jyotirmoy V. Deshmukh, editors, *Proc. of HSCC 2018*, pages 197–206. ACM, 2018.

[BGKN22] Luca Bortolussi, Giuseppe Maria Gallo, Jan Křetínský, and Laura Nenzi. Learning model checking and the kernel trick for signal temporal logic on stochastic processes. In *Proc. of TACAS 2022*, volume 13243 of *LNCS*, pages 281–300. Springer, 2022.

[BK08] Christel Baier and Joost-Pieter Katoen. *Principles of Model Checking*. The MIT Press, 2008.

[BMM+19] Ezio Bartocci, Niveditha Manjunath, Leonardo Mariani, Cristinel Mateis, and Dejan Nickovic. Automatic failure explanation in CPS models. In *Proc. of SEFM 2019*, volume 11724 of *LNCS*, pages 69–86. Springer, 2019.

[BMM+21] Ezio Bartocci, Niveditha Manjunath, Leonardo Mariani, Cristinel Mateis, and Dejan Nickovic. CPSDebug: Automatic failure explanation in CPS models. *Int. J. Softw. Tools Technol. Transf.*, 23(5):783–796, 2021.

[BMNN22] Ezio Bartocci, Cristinel Mateis, Eleonora Nesterini, and Dejan Nickovic. Survey on mining signal temporal logic specifications. *Inf. Comput.*, 289(Part):104957, 2022.

[BMNN23] Ezio Bartocci, Cristinel Mateis, Eleonora Nesterini, and Dejan Ničković. Mining hyperproperties using temporal logics. *ACM Trans. Embed. Comput. Syst.*, 22(5s), 2023.

[BMNY23] Ezio Bartocci, Leonardo Mariani, Dejan Ničković, and Drishti Yadav. Property-based mutation testing. In *Proc. of ICST 2023*, pages 222–233, 2023.

[BVP+16] Giuseppe Bombara, Cristian Ioan Vasile, Francisco Penedo, Hirotoshi Yasuoka, and Calin Belta. A decision tree approach to data classification using signal temporal logic. In *Proc. of HSCC 2016*, pages 1–10. ACM, 2016.

[CFK+14] Michael R. Clarkson, Bernd Finkbeiner, Masoud Koleini, Kristopher K. Micinski, Markus N. Rabe, and César Sánchez. Temporal logics for hyperproperties. In *Proc. of POST 2014*, volume 8414 of *LNCS*, pages 265–284. Springer, 2014.

[CKH+15] Yanping Chen, Eamonn Keogh, Bing Hu, Nurjahan Begum, Anthony Bagnall, Abdullah Mueen, and Gustavo Batista. The UCR time series classification archive, July 2015.

[CO94] Rafael C. Carrasco and José Oncina. Learning stochastic regular grammars by means of a state merging method. In *Proc. of ICGI 94*, volume 862 of *LNCS*, pages 139–152. Springer, 1994.

[CS08] Michael R. Clarkson and Fred B. Schneider. Hyperproperties. In *2008 21st IEEE Computer Security Foundations Symposium*, pages 51–65, 2008.

[CTUB13] Yushan Chen, Jana Tumova, Alphan Ulusoy, and Calin Belta. Temporal logic robot control based on automata learning of environmental dynamics. *The International Journal of Robotics Research*, 32(5):547–565, 2013.

[Dae10] Walter Daelemans. Colin de la higuera: Grammatical inference: learning automata and grammars: Cambridge university press, 2010, iv + 417 pages. *Machine Translation*, 24:291–293, 12 2010.

[DFG+19] Tommaso Dreossi, Daniel J. Fremont, Shromona Ghosh, Edward Kim, Hadi Ravanbakhsh, Marcell Vazquez-Chanlatte, and Sanjit A. Seshia. VerifAI: A toolkit for the formal design and analysis of artificial intelligence-based systems. In *Proc of CAV 2019*, volume 11561 of *LNCS*, pages 432–442. Springer, 2019.

[DFM13] Alexandre Donzé, Thomas Ferrère, and Oded Maler. Efficient robust monitoring for STL. In *Proc. of CAV 2013*, volume 8044 of *LNCS*, pages 264–279. Springer, 2013.

[DHF14] Adel Dokhanchi, Bardh Hoxha, and Georgios Fainekos. On-line monitoring for temporal logic robustness. In *Proc. of RV 2014: the 5th International Conference on Runtime Verification*, volume 8734 of *LNCS*, pages 231–246. Springer, 2014.

[DJAS+08] Lee Dah-Jye, James Archibald, Robert Schoenberger, Aaron Dennis, and Dennis Shiozawa. *Contour Matching for Fish Species Recognition and Migration Monitoring*, volume 122, pages 183–207. 2008.

[DM10] Alexandre Donzé and Oded Maler. Robust satisfaction of temporal logic over real-valued signals. In *Proc. of FORMATS 2010*, volume 6246 of *LNCS*, pages 92–106. Springer, 2010.

[Don10] Alexandre Donzé. Breach, a toolbox for verification and parameter synthesis of hybrid systems. In Tayssir Touili, Byron Cook, and Paul Jackson, editors, *Proc. of CAV 2010*, volume 6174 of *LNCS*, pages 167–170. Springer, 2010.

[DRC+17] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. CARLA: An open urban driving simulator. In *Proc. of CoRL*, volume 78 of *PMLR*, pages 1–16, 2017.

[DS19] Jyotirmoy V. Deshmukh and Sriram Sankaranarayanan. *Formal Techniques for Verification and Testing of Cyber-Physical Systems*, pages 69–105. Springer International Publishing, Cham, 2019.

[EA02] Oded Maler Eugene Asarin, Paul Caspi. Timed regular expressions. *J. ACM*, 49(2):172–206, March 2002.

[EPG+07] Michael D. Ernst, Jeff H. Perkins, Philip J. Guo, Stephen McCamant, Carlos Pacheco, Matthew S. Tschantz, and Chen Xiao. The Daikon system for dynamic detection of likely invariants. *Sci. Comput. Program.*, 69(1-3):35–45, 2007.

[FA22] Nicole Fronda and Houssam Abbas. Differentiable inference of temporal logic formulas. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 41(11):4193–4204, 2022.

[FHST20] Bernd Finkbeiner, Christopher Hahn, Marvin Stenger, and Leander Tentrup. Efficient monitoring of hyperproperties using prefix trees. *Int. J. Softw. Tools Technol. Transf.*, 22(6):729–740, 2020.

[FHT19] Bernd Finkbeiner, Lennart Haas, and Hazem Torfah. Canonical representations of k-safety hyperproperties. In *Proc. of CSF 2019*, pages 17–1714. IEEE 32nd, 2019.

[FKD+22] Daniel J. Fremont, Edward Kim, Tommaso Dreossi, Shromona Ghosh, Xiangyu Yue, Alberto L. Sangiovanni-Vincentelli, and Sanjit A. Seshia.

Scenic: A language for scenario specification and data generation. *Machine Learning Journal*, 112:3805—-3849, 2022.

[FLGD+11] Goran Frehse, Colas Le Guernic, Alexandre Donzé, Scott Cotton, Rajarshi Ray, Olivier Lebeltel, Rodolfo Ripado, Antoine Girard, Thao Dang, and Oded Maler. Spaceex: Scalable verification of hybrid systems. In Ganesh Gopalakrishnan and Shaz Qadeer, editors, *Computer Aided Verification*, pages 379–395, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[FMRS18] Yliès Falcone, Leonardo Mariani, Antoine Rollet, and Saikat Saha. *Runtime Failure Prevention and Reaction*, pages 103–134. Springer International Publishing, Cham, 2018.

[FP06] Georgios E. Fainekos and George J. Pappas. Robustness of temporal logic specifications. In *Proc. of FATES 2006 and RV 2006*, volume 4262 of *LNCS*, pages 178–192. Springer, 2006.

[FP09] Georgios E. Fainekos and George J. Pappas. Robustness of temporal logic specifications for continuous-time signals. *Theor. Comput. Sci.*, 410(42):4262–4291, 2009.

[FR75] Jeanne Ferrante and Charles Rackoff. A decision procedure for the first order theory of real addition with order. *SIAM J. Comput.*, 4(1):69–76, 1975.

[FR09] François Fages and Aurélien Rizk. From model-checking to temporal logic constraint solving. In *Proc. of CP 2009*, volume 5732 of *LNCS*, pages 319–334. Springer, 2009.

[GM05] Jason Ghidellaand and Pieter Mosterman. Requirements-based testing in aircraft control design. In *AIAA Modeling and Simulation Technologies Conference and Exhibit*, pages 1–11, 08 2005.

[GNR+21] Jean-Raphaël Gaglione, Daniel Neider, Rajarshi Roy, Ufuk Topcu, and Zhe Xu. Learning linear temporal properties from noisy data: A MaxSAT-Based approach. In *Proc. of ATVA 2021*, volume 12971 of *LNCS*, pages 74–90. Springer, 2021.

[HC20] Samuel Huang and Rance Cleaveland. Temporal-logic query checking over finite data streams. In *Proc. of FMICS 2020*, volume 12327 of *LNCS*, pages 252–271. Springer, 2020.

[HDF18] Bardh Hoxha, Adel Dokhanchi, and Georgios E. Fainekos. Mining parametric temporal logic properties in model-based design for cyber-physical systems. *STTT*, 20(1):79–93, 2018.

[HHY+23] Navid Hashemi, Bardh Hoxha, Tomoya Yamaguchi, Danil Prokhorov, Georgios Fainekos, and Jyotirmoy Deshmukh. A neurosymbolic approach to the verification of temporal logic properties of learning-enabled control systems. In *Proceedings of the ACM/IEEE 14th International Conference on Cyber-Physical Systems (with CPS-IoT Week 2023)*, ICCPS '23, page 98–109, New York, NY, USA, 2023. Association for Computing Machinery.

[HS18] Falk Howar and Bernhard Steffen. Active automata learning in practice. In *Machine Learning for Dynamic Software Analysis: Potentials and Limits: International Dagstuhl Seminar 16172*, volume 11026 of *LNCS*, pages 123–148. Springer, 2018.

[HU79] John E. Hopcroft and Jeff D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley Publishing Company, 1979.

[IBMN22] Patrick Indri, Alberto Bartoli, Eric Medvet, and Laura Nenzi. One-shot learning of ensembles of temporal logic formulas for anomaly detection in cyber-physical systems. In *Genetic Programming*, volume 13223 of *LNCS*, pages 34–50. Springer, 2022.

[JDDS13] Xiaoqing Jin, Alexandre Donzé, Jyotirmoy V. Deshmukh, and Sanjit A. Seshia. Mining requirements from closed-loop control models. In *Proc. of HSCC 2013*, pages 43–52. ACM, 2013.

[JDDS15] Xiaoqing Jin, Alexandre Donzé, Jyotirmoy V. Deshmukh, and Sanjit A. Seshia. Mining requirements from closed-loop control models. *IEEE TCAD*, 34(11):1704–1717, 2015.

[JKB14] Austin Jones, Zhaodan Kong, and Calin Belta. Anomaly detection in cyber-physical systems: A formal methods approach. In *53rd IEEE Conference on Decision and Control*, pages 848–853, 2014.

[JOP+01] Eric Jones, Travis Oliphant, Pearu Peterson, et al. SciPy: Open source scientific tools for Python, 2001.

[JST25] Sebastian Junges, Sanjit A. Seshia, and Hazem Torfah. Active learning of runtime monitors under uncertainty. In Nikolai Kosmatov and Laura Kovács, editors, *Integrated Formal Methods*, pages 297–306, Cham, 2025. Springer Nature Switzerland.

[JTS+17] Susmit Jha, Ashish Tiwari, Sanjit A. Seshia, Tuhin Sahai, and Natarajan Shankar. TeLEx: Passive STL learning using only positive examples. In *Proc. of RV 2017*, volume 10548 of *LNCS*, pages 208–224. Springer, 2017.

[JTS+19] Susmit Jha, Ashish Tiwari, Sanjit A. Seshia, Tuhin Sahai, and Natarajan Shankar. TeLEx: Learning signal temporal logic from positive examples using tightness metric. *Formal Methods in System Design*, 54(3):364–387, 2019.

[KCMP01] Eamonn J. Keogh, Kaushik Chakrabarti, Sharad Mehrotra, and Michael J. Pazzani. Locally adaptive dimensionality reduction for indexing large time series databases. In *Proc. of ACM SIGMOD 2001*, pages 151–162. ACM, 2001.

[KDJ+16] James Kapinski, Jyotirmoy V. Deshmukh, Xiaoqing Jin, Hisahiro Ito, and Ken Butts. Simulation-based approaches for verification of embedded control systems: An overview of traditional and advanced modeling, testing, and verification techniques. *IEEE Control Systems Magazine*, 36(6):45–64, 2016.

[KGV83] Scott Kirkpatrick, Daniel Gelatt, and Mario P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.

[KJB17] Zhaodan Kong, Austin Jones, and Calin Belta. Temporal logics for learning and detection of anomalous behavior. *IEEE Transactions on Automatic Control*, 62(3):1210–1222, 2017.

[KJMA+14] Zhaodan Kong, Austin Jones, Ana Medina Ayala, Ebru Aydin Gol, and Calin Belta. Temporal logic inference for classification and prediction from data. In *Proc. of HSCC 2014*, pages 273—-282. ACM, 2014.

[KK03] Eamonn J. Keogh and Shruti Kasetty. On the need for time series data mining benchmarks: A survey and empirical demonstration. *Data Min. Knowl. Discov.*, 7(4):349–371, 2003.

[KL88] Dennis F. Kibler and Pat Langley. Machine learning as an experimental science. In *Proc. of EWSL 1988*, pages 81–92. Pitman Publishing, 1988.

[Kle51] S. C. Kleene. *Representation of Events in Nerve Nets and Finite Automata*. RAND Corporation, Santa Monica, CA, 1951.

[KMB+17] Barbara Kitchenham, Lech Madeyski, David Budgen, Jacky Keung, Pearl Brereton, Stuart Charters, Shirley Gibbs, and Amnart Pohthong. Robust statistical methods for empirical software engineering. *Empirical Softw. Engg.*, 22(2):579–630, apr 2017.

[Koy90] Ron Koymans. Specifying real-time properties with metric temporal logic. *Real Time Syst.*, 2(4):255–299, 1990.

[KP98] Eamonn J. Keogh and Michael J. Pazzani. An enhanced representation of time series which allows fast and accurate classification, clustering and relevance feedback. In *Proc. of KDD 98*, pages 239–243. AAAI Press, 1998.

[LAP20] Karen Leung, Nikos Arechiga, and Marco Pavone. Back-propagation through signal temporal logic specifications: Infusing logical structure into gradient-based methods. In *Proc. of WAFR 2021*, volume 17 of *Springer Proceedings in Advanced Robotics*, pages 432–449. Springer, 2020.

[LB15] Caroline Lemieux and Ivan Beschastnikh. Investigating program behavior using the texada LTL specifications miner. In *Proc. of ASE*, pages 870–875, 2015.

[LG10] Colas Le Guernic and Antoine Girard. Reachability analysis of linear systems using support functions. *Nonlinear Analysis: Hybrid Systems*, 4(2):250–262, 2010. IFAC World Congress 2008.

[LPB15] Caroline Lemieux, Dennis Park, and Ivan Beschastnikh. General LTL specification mining (T). In *Proc. of ASE 2015*, pages 81–92. IEEE, 2015.

[LTLT22] Alexis Linard, Ilaria Torre, Iolanda Leite, and Jana Tumova. Inference of multi-class stl specifications for multi-label human-robot encounters. In *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1305–1311, 2022.

[LW93] Rüdiger Loos and Volker Weispfenning. Applying linear quantifier elimination. *Comput. J.*, 36(5):450–462, 1993.

[Mac67] James MacQueen. Some methods for classification and analysis of multivariate observations. In *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability, Volume 1: Statistics*, volume 5, pages 281–298. University of California press, 1967.

[MDP20a] Sara Mohammadinejad, Jyotirmoy V. Deshmukh, and Aniruddh Gopinath Puranic. Mining environment assumptions for cyber-physical system models. In *Proc. of ICCPS 2020*, pages 87–97. IEEE, 2020.

[MDP+20b] 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 *Proc. of HSCC 2020*, pages 9:1–9:10. ACM, 2020.

[MN04] Oded Maler and Dejan Nickovic. Monitoring temporal properties of continuous signals. In *Proc. of FORMATS and FTRTFT 2024*, volume 3253 of *LNCS*, pages 152–166. Springer, 2004.

[MRBF15] Ramy Medhat, S. Ramesh, Borzoo Bonakdarpour, and Sebastian Fischmeister. A framework for mining hybrid automata from input/output traces. In *Proc. of EMSOFT 2015*, pages 177–186. IEEE, 2015.

[MSBF21] Meiyi Ma, John A. Stankovic, Ezio Bartocci, and Lu Feng. Predictive monitoring with logic-calibrated uncertainty for cyber-physical systems. *ACM Trans. Embed. Comput. Syst.*, 20(5s), 2021.

[NBB+22] Laura Nenzi, Ezio Bartocci, Luca Bortolussi, Simone Silvetti, and Michele Loreti. MoonLight: A lightweight tool for monitoring spatio-temporal properties. *Int. J. Softw. Tools Technol. Transf.*, page to appear, 2022.

[NBG+25] Eleonora Nesterini, Ezio Bartocci, Alessio Gambi, Dejan Nickovic, Sanjit A. Seshia, and Hazem Torfah. Mining specifications for predictive safety monitoring. In *Proceedings of the ACM/IEEE 16th International Conference on Cyber-Physical Systems (with CPS-IoT Week 2025)*, ICCPS '25, New York, NY, USA, 2025. Association for Computing Machinery.

[NCJF18] Apurva Narayan, Greta Cutulenco, Yogi Joshi, and Sebastian Fischmeister. Mining timed regular specifications from system traces. *ACM Trans. Embed. Comput. Syst.*, 17(2):46:1–46:21, 2018.

[NF19] Apurva Narayan and Sebastian Fischmeister. Mining time for timed regular specifications. In *Proc. of SMC 2019*, pages 63–69. IEEE, 2019.

[NG18] Daniel Neider and Ivan Gavran. Learning linear temporal properties. In *Proc. of FMCAD 2018*, pages 1–10. IEEE, 2018.

[NG20] Daniel Neider and Bishwamittra Ghosh. Probably approximately correct explanations of machine learning models via syntax-guided synthesis. 2020.

[Nil86] Nils J. Nilsson. Probabilistic logic. *Artif. Intell.*, 28(1):71–87, 1986.

[NKJ+17a] Luan Viet Nguyen, James Kapinski, Xiaoqing Jin, Jyotirmoy V. Deshmukh, Ken Butts, and Taylor T. Johnson. Abnormal Data Classification Using Time-Frequency Temporal Logic. In *Proc. of HSCC 2017*, pages 237–242. ACM, 2017.

[NKJ+17b] Luan Viet Nguyen, James Kapinski, Xiaoqing Jin, Jyotirmoy V. Deshmukh, and Taylor T. Johnson. Hyperproperties of real-valued signals. In *Proc. of MEMOCODE 2017*, pages 104–113. ACM, 2017.

[NQF+19] Dejan Nickovic, Xin Qin, Thomas Ferrère, Cristinel Mateis, and Jyotirmoy V. Deshmukh. Shape expressions for specifying and extracting signal features. In *Proc. of RV*, volume 11757 of *LNCS*, pages 292–309, 2019.

[NQF+21] Dejan Nickovic, Xin Qin, Thomas Ferrère, Cristinel Mateis, and Jyotirmoy Deshmukh. Specifying and detecting temporal patterns with shape expressions. *Int. J. Softw. Tools Technol. Transf.*, 23(4):565–577, 2021.

[NSBB18] Laura Nenzi, Simone Silvetti, Ezio Bartocci, and Luca Bortolussi. A robust genetic algorithm for learning temporal specifications from data. In *Proc. of QEST 2018*, volume 11024 of *LNCS*, pages 323–338. Springer, 2018.

[NSF+10] Truong Nghiem, Sriram Sankaranarayanan, Georgios Fainekos, Franjo Ivancic, Aarti Gupta, and George J. Pappas. Monte-carlo techniques for falsification of temporal properties of non-linear hybrid systems. In *Proc. of HSCC 2010*, pages 211–220. ACM, 2010.

[NWP02] Tobias Nipkow, Markus Wenzel, and Lawrence C. Paulson, editors. *5. The Rules of the Game*, pages 67–104. Springer Berlin Heidelberg, Berlin, Heidelberg, 2002.

[NY20] Dejan Nickovic and Tomoya Yamaguchi. RTAMT: online robustness monitors from STL. In *Proc. of ATVA 2020*, volume 12302 of *LNCS*, pages 564–571. Springer, 2020.

[ORS92] Sam Owre, John Rushby, and Natarajan Shankar. PVS: A prototype verification system. In Deepak Kapur, editor, *Automated Deduction—CADE-11*, pages 748–752, Berlin, Heidelberg, 1992. Springer Berlin Heidelberg.

[Oza16] Necmiye Ozay. An exact and efficient algorithm for segmentation of arx models. In *2016 American Control Conference (ACC)*, pages 38–41, 2016.

[PMM17] Fabrizio Pastore, Daniela Micucci, and Leonardo Mariani. Timed k-tail: Automatic inference of timed automata. In *Proc. of ICST 2017*, pages 401–411. IEEE, 2017.

[PMN21] Federico Pigozzi, Eric Medvet, and Laura Nenzi. Mining road traffic rules with signal temporal logic and grammar-based genetic programming. *Applied Sciences*, 11(22), 2021.

[Pnu77] Amir Pnueli. The temporal logic of programs. In *Proc. of SFCS 1977*, pages 46–57. IEEE, 1977.

[Pow64] Michael J.D. Powell. An efficient method for finding the minimum of a function of several variables without calculating derivatives. In *The Computer Journal*, volume 7, pages 155–162. 1964.

[QXZ+22] Xin Qin, Yuan Xia, Aditya Zutshi, Chuchu Fan, and Jyotirmoy V. Deshmukh. Statistical verification of cyber-physical systems using surrogate models and conformal inference. In *2022 ACM/IEEE 13th International Conference on Cyber-Physical Systems (ICCPS)*, pages 116–126, 2022.

[Rie19] Heinz Riener. Exact synthesis of LTL properties from traces. In *Proc. of FDL 2019*, pages 1–6. IEEE, 2019.

[RMS20] Mayank Rawat, Sujit Kumar Muduli, and Pramod Subramanyan. Mining hyperproperties from behavioral traces. In *Proc. VLSI-SOC 2020*, pages 88–93. IEEE, 2020.

[RRF20] Matthieu Renard, Antoine Rollet, and Yliès Falcone. Runtime enforcement of timed properties using games. *Form. Asp. Comput.*, 32(2–3):315–360, July 2020.

[SFA17] Calvin Smith, Gabriel Ferns, and Aws Albarghouthi. Discovering relational specifications. In *Proc. of ESEC/FSE 2017*, pages 616–626. ACM, 2017.

[SG19] Irmak Saglam and Ebru Aydin Gol. Cause mining and controller synthesis with STL. In *Proc. of CDC 2019*, pages 4589–4594. IEEE, 2019.

[SHM11] Bernhard Steffen, Falk Howar, and Maik Merten. Introduction to active automata learning from a practical perspective. In *Proc. of SFM 2011*, volume 6659 of *LNCS*, pages 256–296. Springer, 2011.

[SHSZ19] Miriam García Soto, Thomas A. Henzinger, Christian Schilling, and Luka Zeleznik. Membership-based synthesis of linear hybrid automata. In *Proc. of CAV 2019*, volume 11561 of *LNCS*, pages 297–314. Springer, 2019.

[SSS22] Sanjit A. Seshia, Dorsa Sadigh, and S. Shankar Sastry. Toward verified artificial intelligence. *Communications of the ACM*, 65(7):46–55, 2022.

[SZ96] Hagit Shatkay and Stanley B. Zdonik. Approximate queries and representations for large data sequences. In *Proc. of the ICDE 1996*, pages 536–545. IEEE, 1996.

[TJS+23] Hazem Torfah, Aniruddha Joshi, Shetal Shah, S. Akshay, Supratik Chakraborty, and Sanjit A. Seshia. Learning monitor ensembles for operational design domains. In *Proc of RV 2023*, volume 14245 of *LNCS*, pages 271–290. Springer, 2023.

[TXJ+22] Hazem Torfah, Carol Xie, Sebastian Junges, Marcell Vazquez-Chanlatte, and Sanjit A. Seshia. Learning monitorable operational design domains for assured autonomy. In *Proc of ATVA 2022*, volume 13505 of *LNCS*, pages 3–22. Springer, 2022.

[TYML+20] Hoang-Dung Tran, Xiaodong Yang, Diego Manzanas Lopez, Patrick Musau, Luan Viet Nguyen, Weiming Xiang, Stanley Bak, and Taylor T. Johnson. Nnv: The neural network verification tool for deep neural networks and learning-enabled cyber-physical systems. In Shuvendu K. Lahiri and Chao Wang, editors, *Computer Aided Verification*, pages 3–17, Cham, 2020. Springer International Publishing.

[UBK15] Liudmila Ulanova, Nurjahan Begum, and Eamonn J. Keogh. Scalable clustering of time series with U-Shapelets. In *Proc. of SDM 2015*, pages 900–908. SIAM, 2015.

[Ulu19] Dogan Ulus. Online monitoring of metric temporal logic using sequential networks. *CoRR*, abs/1901.00175, 2019.

[UXKL06] Ken Ueno, Xiaopeng Xi, Eamonn Keogh, and Dah-jye Lee. Anytime classification using the nearest neighbor algorithm with applications to stream mining. In *Sixth International Conference on Data Mining (ICDM'06)*, pages 623–632, 2006.

[VDJS17] Marcell Vazquez-Chanlatte, Jyotirmoy V. Deshmukh, Xiaoqing Jin, and Sanjit A. Seshia. Logical clustering and learning for time-series data. In *Proc. of CAV 2017*, volume 10426 of *LNCS*, pages 305–325. Springer, 2017.

[VGD+18] Marcell Vazquez-Chanlatte, Shromona Ghosh, Jyotirmoy V. Deshmukh, Alberto L. Sangiovanni-Vincentelli, and Sanjit A. Seshia. Time-series learning using monotonic logical properties. In *Proc. of RV 2018*, volume 11237 of *LNCS*, pages 389–405. Springer, 2018.

[VIB+17] Prashant Vaidyanathan, Rachael Ivison, Giuseppe Bombara, Nicholas A. DeLateur, Ron Weiss, Douglas Densmore, and Calin Belta. Grid-based temporal logic inference. In *Proc. of CDC 2017*, pages 5354–5359. IEEE, 2017.

[Wal92] A. Wald. *Sequential Tests of Statistical Hypotheses*, pages 256–298. Springer New York, New York, NY, 1992.

[WBL22] Chien-Yao Wang, Alexey Bochkovskiy, and Hong-Yuan Mark Liao. YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors. 2022.

[WCTZ20] Fujun Wang, Zining Cao, Lixing Tan, and Hui Zong. Survey on learning-based formal methods: Taxonomy, applications and possible future directions. *IEEE Access*, 8:108561–108578, 2020.

[Web23] Webots, 2023. Open-source Mobile Robot Simulation Software.

[XJ18] Z. Xu and A. A. Julius. Census signal temporal logic inference for multiagent group behavior analysis. *IEEE Transactions on Automation Science and Engineering*, 15(1):264–277, 2018.

[XTJ18] Weiming Xiang, Hoang-Dung Tran, and Taylor T. Johnson. Output reachable set estimation and verification for multilayer neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 29(11):5777–5783, 2018.

[YHF12] Hengyi Yang, Bardh Hoxha, and Georgios E. Fainekos. Querying parametric temporal logic properties on embedded systems. In *Proc. of ICTSS 2012*, volume 7641 of *LNCS*, pages 136–151. Springer, 2012.

[YS02] Håkan L. S. Younes and Reid G. Simmons. Probabilistic verification of discrete event systems using acceptance sampling. In *Proc. of CAV 2002*, volume 2404 of *LNCS*, page 223–235. Springer, 2002.

[YTFS23] Beyazit Yalcinkaya, Hazem Torfah, Daniel J. Fremont, and Sanjit A. Seshia. Compositional simulation-based analysis of ai-based autonomous systems for markovian specifications. In *Proc of RV 2023*, volume 14245 of *LNCS*, pages 191–212. Springer, 2023.

[YZU+16] Chin-Chia Michael Yeh, Yan Zhu, Liudmila Ulanova, Nurjahan Begum, Yifei Ding, Hoang Anh Dau, Diego Furtado Silva, Abdullah Mueen, and Eamonn J. Keogh. Matrix profile I: all pairs similarity joins for time series: A unifying view that includes motifs, discords and shapelets. In *Proc. of ICDM 2016*, pages 1317–1322. IEEE, 2016.

[Zad88] Lotfi A. Zadeh. Fuzzy logic. *Computer*, 21(4):83–93, 1988.

Overview of Generative AI Tools Used

I have used OpenAI's ChatGPT-4o-mini as a writing assistant. Specifically, I have employed this tool to enhance the clarity of my writing by polishing text, suggesting contextually appropriate synonyms, and correcting grammatical mistakes. This very paragraph was generated by ChatGPT in response to my request to transform my informal description of its usage into a formal acknowledgment. However, no other text in this thesis was generated by ChatGPT; all content, analysis, and argumentation are entirely my own.

Additionally, ChatGPT contributed to debugging the *.tex* file of this thesis, some parts of the Python code for the tool *ObSTLearn* (Section 1.5.3), and translating the English version of the abstract into the German *Kurzfassung*.

List of Figures

List of Tables

List of Algorithms