# What Is Formal Verification Without Specifications? A Survey on Mining LTL Specifications

Daniel Neider[1,2] and Rajarshi Roy[3(✉)]

[1] TU Dortmund University, Dortmund, Germany
[2] Center for Trustworthy Data Science and Security, University Alliance Ruhr, Dortmund, Germany
[3] Department of Computer Science, University of Oxford, Oxford, UK
rajarshi.roy@cs.ox.ac.uk

**Abstract.** Virtually all verification techniques using formal methods rely on the availability of a formal specification, which describes the design requirements precisely. However, formulating specifications remains a manual task that is notoriously challenging and error-prone. To address this bottleneck in formal verification, recent research has thus focussed on automatically generating specifications for formal verification from examples of (desired and undesired) system behavior. In this survey, we list and compare recent advances in mining specifications in Linear Temporal Logic (LTL), the de facto standard specification language for reactive systems. Several approaches have been designed for learning LTL formulas, which address different aspects and settings of specification design. Moreover, the approaches rely on a diverse range of techniques such as constraint solving, neural network training, enumerative search, etc. We survey the current state-of-the-art techniques and compare them for the convenience of the formal methods practitioners.

## 1 Introduction

Formal methods refer to the discipline of computer science that employs mathematically rigorous techniques to ensure the safe behavior of software, hardware, and cyber-physical systems. There have been countless success stories of formal methods, ranging over several application domains such as communication systems [23,47], railway transportation [5,6], aerospace [20,27], and operating systems [40,70], to name but a few. We refer the reader to the exceptional textbook by Baier and Katoen [7] for a comprehensive introduction.

However, there is an important catch with verification techniques: they assume the availability of functional and usable specifications that precisely describe the design requirements. This assumption is often unrealistic as designing specifications, which had been primarily a manual task, proves not only to be tedious but also error-prone. Consequently, the availability of formal specifications is widely regarded as a major bottleneck in formal methods [2,11,67].


N. Jansen et al. (Eds.): Principles of Verification: Cycling the Probabilistic Landscape, LNCS 15262, pp. 109–125, 2025.
https://doi.org/10.1007/978-3-031-75778-5_6

To overcome this limitation, recent efforts have focused on developing methods that can automatically generate specifications from examples of desired and undesired system behavior. Notably, a significant body of research has emerged that concentrates on *learning* specifications in Linear Temporal Logic (LTL). This focus on LTL is due to its dual benefits: mathematical precision and interpretability. The latter has recently become of increasing interest as it facilitates the application of LTL beyond formal verification to areas such as reinforcement learning [48], planning [72], and other AI-related domains.

This survey provides a comprehensive overview of the diverse body of work focused on learning LTL specifications. In the past decade, researchers have tackled this task from multiple perspectives, spanning various settings and methodologies. As summarized in Table 1, these efforts can be differentiated based on their learning setup, methodology, and the guarantees they provide, offering a nuanced understanding of the field.

We specifically focus on approaches addressing the *passive learning problem*, where the objective is to learn concise LTL formulas that accurately capture user-provided examples of a system's behavior. These examples typically consist of two categories: positive (desirable) and negative (undesirable) system behaviors. However, in many practical scenarios, it is unrealistic to assume the availability of perfectly labeled examples for both classes. Real-world data is often noisy, making it challenging to accurately classify the data. Furthermore, in many safety-critical domains, such as autonomous vehicles and medical devices, obtaining negative examples may be infeasible or even risky (potentially harming humans). As a result, some approaches have explored less conventional settings, including noisy data and scenarios where only positive examples are available.

The approaches summarized in Table 1 share a common thread: they employ search strategies to navigate the space of possible LTL formulas, although these strategies differ substantially in their methodology. Some approaches leverage off-the-shelf constraint solvers (e.g., SAT and SMT solvers) by carefully encoding the learning problem in propositional or first-order logic, while others employ specialized enumeration techniques tailored to the task. A third category of approaches harnesses advances in deep learning to identify promising LTL formulas. As a result, the search strategies, considered LTL fragments, and theoretical guarantees vary substantially.

The remainder of this paper compares and contrasts the underlying principles of the aforementioned approaches, categorized into three distinct groups based on their search strategies. These categories comprise constraint-based, enumeration-based, and neural-network-based methods, which will be explored in Sects. 3, 4, and 5, respectively. Section 2 provides the required background on LTL and the learning problem we consider.

## 2 Preliminaries

**System Executions and Words.** Informal methods, executions or trajectories of systems are typically formalized as sequences of symbols from a finite non-empty set $\Sigma$, known as alphabet. We refer to such sequences as *words* over $\Sigma$.

**Table 1.** Comparison of Related Works and Their Features. 'Pos' and 'Neg' refer to positive and negative examples, respectively; 'SAT', 'MaxSAT', 'MILP' refer to satisfiability, maximum satisfiability, and mixed-integer linear programming, respectively.

| Work | Primary techniques | LTL Fragments | Classification | Input Data | Guarantees |
|---|---|---|---|---|---|
| Neider and Gavran [52], Riener [64] | SAT | full LTL | perfect | Pos+Neg | sound, complete, minimal |
| Camacho and Mcllraith [15] | SAT | full LTL | perfect | Pos+Neg | sound, complete, minimal |
| Raha et al. [60] | enumerative search, dynamic programming | comb. of directed LTL | perfect/noisy | Pos+Neg | sound |
| Arif et al. [3] | Syntax-Guided Synthesis | past-time LTL | perfect | Pos+Neg | sound, complete, minimal |
| Gaglione et al. [26] | MaxSAT | full LTL | noisy | Pos+Neg | sound, complete, minimal |
| Chou et al. [18] | MILP, counterexample-guided | full LTL | perfect | Pos only | sound, complete |
| Ghiorzi et al. [30] | enumerative search | full LTL | perfect | Pos+Neg | sound, complete, minimal |
| Ielo et al. [38] | Answer Set Programming | full LTL | perfect | Pos+Neg | sound, complete, minimal |
| Roy et al. [66] | SAT, counterexample-guided | full LTL | perfect | Pos only | sound, complete, language minimal |
| Valizadeh et al. [69] | enumerative search, GPU acceleration | full LTL | perfect/noisy | Pos+Neg | sound |
| Luo et al. [48] | Graph Neural Network | full LTL | noisy | Pos+Neg | No |
| Wan et al. [72] | Graph Neural Network | full LTL | noisy | Pos+Neg | No |

A word $w = a_0 a_1 \ldots$, where $a_i \in \Sigma$, can be either finite or infinite, depending on whether the execution it represents is finite or infinite. The set of infinite words over $\Sigma$ is denoted by $\Sigma^\omega$, while the set of finite words is denoted by $\Sigma^*$. Given a word $w = a_1 a_2 \ldots$ in $\Sigma^*$ or $\Sigma^\omega$, we let $w[i] \coloneqq a_i$ denote symbol of $w$ at position $i$ and $w[i:] \coloneqq a_i a_{i+1} \ldots$ the suffix starting from the starting from position $i$. The length $|w|$ of a word $w$ is the number of its symbols. In particular, the empty word, denoted by $\varepsilon$, has length zero.

**Linear Temporal Logic (LTL).** The logic LTL [58] is the de facto standard for reasoning about executions, or sequences of events, of reactive systems. Typically, specific events in a system are abstracted using a set $\mathcal{P}$ of propositions, which represent events or properties of interest in the system under consideration. A system execution is then modeled by a word over the alphabet $\Sigma = 2^{\mathcal{P}}$, capturing the propositions that hold true at specific time points along the system's execution.

Given a set $\mathcal{P}$ of propositions, the syntax of LTL formulas is defined inductively using the grammar

$$\varphi \coloneqq p \in \mathcal{P} \mid \neg\varphi \mid \varphi \vee \varphi \mid \mathbf{X}\,\varphi \mid \varphi\,\mathbf{U}\,\varphi,$$

where $\neg$ (not) and $\vee$ (or) are Boolean operators, while $\mathbf{X}$ (neXt) and $\mathbf{U}$ (Until) are so-called future-time temporal operators. Several derived Boolean operators, including $\wedge$ (conjunction) and $\rightarrow$ (implication), as well as temporal operators, such as $\mathbf{F}$ (Eventually), $\mathbf{G}$ (Always), $\mathbf{W}$ (Weak Until), and $\mathbf{R}$ (Release), are often

added as syntactic sugar. Additionally, some works also incorporate past-time temporal operators, including **P** (Previously) and **S** (Since), which are past-time analogs of **X** and **U**, respectively.

To define the semantics of LTL, one usually uses a "model relation", denoted by $\models$, which captures when the suffix of a word $w \in (2^{\mathcal{P}})^\omega$ starting at a position $\in \mathbb{N}$ satisfies an LTL formula $\varphi$. Formally, this relation is given as follows:

$$\begin{aligned}
w, i \models p &\text{ if and only if } p \in w[i] \\
w, i \models \neg\varphi &\text{ if and only if } w, i \not\models \varphi \\
w, i \models \varphi_1 \vee \varphi_2 &\text{ if and only if } w, i \models \varphi_1 \text{ or } w, i \models \varphi_2 \\
w, i \models \mathbf{X}\,\varphi &\text{ if and only if } w, i+1 \models \varphi \\
w, i \models \varphi_1 \,\mathbf{U}\, \varphi_2 &\text{ if and only if } w, j \models \varphi_2 \text{ for some } j \geq i \text{ and} \\
&\qquad\qquad w, k \models \varphi_1 \text{ for each } i \leq k < j
\end{aligned}$$

If the entire word starting at position 0 satisfies $\varphi$ (i.e., $w, 0 \models \varphi$), we simply write $w \models \varphi$ and say that $w$ satisfies $\varphi$.

Traditionally, LTL has been interpreted over infinite words, but there is a growing interest in interpreting LTL over finite words [32], particularly in artificial intelligence applications. To reflect this shift, it is sufficient to adapt the model relation slightly, taking into account the end of a word in the operators **X** and **U**. More precisely, we modify the model relation for a finite word $w \in (2^{\mathcal{P}})^*$ as follows:

$$\begin{aligned}
w, i \models \mathbf{X}\,\varphi &\text{ if and only if } i < |w| - 1 \text{ and } w, i+1 \models \varphi \\
w, i \models \varphi_1 \,\mathbf{U}\, \varphi_2 &\text{ if and only if } u, j \models \varphi_2 \text{ for some } |w-1| \geq j \geq i \text{ and} \\
&\qquad\qquad u, k \models \varphi_1 \text{ for each } i \leq k < j
\end{aligned}$$

The size of an LTL formula $\varphi$, denoted by $|\varphi|$, is defined as the number of its unique subformulas. For example, the size of $\varphi := (p\,\mathbf{U}\,\mathbf{G}\,q) \vee \mathbf{X}(\mathbf{G}\,q)$ is six since it contains six unique subformulas: $(p\,\mathbf{U}\,\mathbf{G}\,q) \vee \mathbf{X}(\mathbf{G}\,q)$, $p\,\mathbf{U}\,\mathbf{G}\,q$, $\mathbf{X}(\mathbf{G}\,q)$, $\mathbf{G}\,q$, $p$, and $q$.

When learning LTL formulas, it is useful to have a canonical representation. A common approach is to use a so-called *syntax directed acyclic graph (DAG)*, which is a syntax tree that merges common subformulas. Figure 1 illustrates the distinction between a syntax tree (Fig. 1a) and a syntax DAG (Fig. 1b) for the formula $\varphi := (p\,\mathbf{U}\,\mathbf{G}\,q) \vee \mathbf{X}(\mathbf{G}\,q)$. Notably, the size of a formula and the number of nodes in its syntax DAG coincide, whereas the number of nodes in the syntax tree can be exponentially larger.

**Passive Learning of LTL Formulas.** With the necessary groundwork established, we now turn our attention to the central task of this survey: learning LTL formulas from examples. To this end, we assume that the examples of desired and undesired system executions are bundled in a *sample*, denoted by $\mathcal{S}$. In the standard setting for passive learning of LTL formulas, this sample takes the


(a) Syntax Tree (b) Syntax DAG

**Fig. 1.** Representations of LTL formula $\varphi = (p \,\mathbf{U}\, \mathbf{G}\, q) \vee \mathbf{X}(\mathbf{G}\, q)$

form of a pair $\mathcal{S} = (P, N)$, comprising two sets of words: $P$, consisting of positive examples, and $N$, comprising negative examples, where $P \cap N = \emptyset$.

A crucial concept in this setting is that of *consistency*, where an LTL formula $\varphi$ is said to be consistent with the sample $\mathcal{S}$ if it satisfies two conditions: firstly, every word $u \in P$ must satisfy $\varphi$ (i.e., $u \models \varphi$), and secondly, every word $v \in N$ must not satisfy $\varphi$ (i.e., $v \not\models \varphi$). This definition allows us to define the passive learning task formally.

**Definition 1 (Passive Learning of LTL formulas).** *Given a sample $\mathcal{S} = (P, N)$, compute a minimal LTL formula $\varphi$ that is consistent with $\mathcal{S}$.*

A crucial aspect of the above definition is the minimality requirement for the prospective LTL formula, as emphasized by Neider and Gavran [52]. Notably, the problem becomes trivial if this size restriction is relaxed: for any $u \in P$ and $v \in N$, one can construct a formula $\varphi_{u,v}$ that captures the first symbol where $u$ and $v$ differ using a sequence of **X**-operators and a suitable propositional formula, ensuring that $u \models \varphi_{u,v}$ and $v \not\models \varphi_{u,v}$. Then, the conjunction of these formulas, $\bigwedge_{u \in P} \bigvee_{v \in N} \varphi_{u,v}$, is consistent with the sample $\mathcal{S}$. However, a formula that simply enumerates the differences in the positive and negative examples suffers from overfitting and fails to generalize the temporal patterns. Furthermore, the resulting formula can become excessively complex, thereby compromising its interpretability.

While the above problem states the standard passive learning problem, in practice, a perfect sample may not always be available. The sample may be noisy (involving misclassifications, sensor reading errors, etc.) or incomplete (e.g., only positive examples are available). Therefore, several works adapt the passive learning problem to accommodate these scenarios.

## 3 Constraint-Based Approaches

As hinted at in the introduction, constraint-based approaches leverage off-the-shelf solvers to search for prospective LTL formulas. These solvers employ a wide range of technologies, including solvers for satisfiability (SAT) [8,51] and maximum satisfiability (MaxSAT) [12] for propositional logic, Inductive Logic Programming (ILP) [29], Mixed Integer Linear Programming (MILP) [53], and Syntax-Guided Synthesis (SyGuS) [1].

At the heart of constraint-based approaches lies the idea of translating the learning problem into one or several satisfiability problems within a suitable logical framework (e.g., SAT or SMT). Although the solver technologies may differ, the underlying logical encodings of the learning problem exhibit striking similarities across most works. In particular, most approaches separate the encoding of the syntax of the prospective formula from its semantics, allowing for a flexible and modular formulation of the search problem. This modularity enables the customization of the search to accommodate specific requirements, such as targeting a particular subclass of formulas or satisfying a specific subset of examples.

To further illustrate this approach, let us consider one of the pioneering works in this category, the paper by Neider and Gavran [52], which leverages SAT solving. The core of their approach revolves around constructing a series $(\Phi_n^{\mathcal{S}})_{n=1,2,\ldots}$ of propositional formulas that facilitate the search for prospective LTL formulas of increasing size $n$. Specifically, each formula $\Phi_n^{\mathcal{S}}$ satisfies two crucial properties: (i) it is satisfiable if and only if there exists an LTL formula of size $n$ that is consistent with $\mathcal{S}$ and (ii) a satisfying assignment of $\Phi_n^{\mathcal{S}}$ contains sufficient information to construct such an LTL formula. By incrementally increasing the value of $n$ until $\Phi_n^{\mathcal{S}}$ becomes satisfiable, one can obtain an LTL formula that is guaranteed to be minimal and consistent with $\mathcal{S}$.

The formula $\Phi_n^{\mathcal{S}}$ is constructed as a conjunction of two subformulas, $\Phi_n^{\mathcal{S}} = \Phi_n^{DAG} \wedge \Phi_n^{con}$. The subformula $\Phi_n^{DAG}$ encodes the syntax DAG of the prospective LTL formula and encompasses a range of constraints, ensuring that fundamental properties of a syntax DAG of an LTL formula are satisfied (e.g., each node being labelled by a unique LTL operator and each node having at most two children). On the other hand, the subformula $\Phi_n^{con}$ ensures that the the positive examples satisfy the prospective LTL formula while the negative ones violate it. To this end, the formula $\Phi_n^{con}$ encodes of the semantics of LTL on the given positive and negative words similar to Bounded Model Checking [19]. They rely on the SAT solver Z3 [51] for their implementation.

Riener [64] expanded upon the work of Neider and Gavran, streamlining the SAT encoding by preprocessing the possible syntactic structures of LTL formulas, formalized as *partial DAGs*. By precomputing and storing partial DAGs, which are not yet labeled with LTL operators, Riener's approach enables a more efficient search by decomposing the search space according to the underlying DAG structure. This innovation potentially enables parallelization, thereby accelerating the encoding process and reducing computational complexity.

Camacho and Mcllraith [15] propose a SAT encoding similar to that of Neider and Gavran, leveraging Alternating Finite Automata (AFA), a type of finite state acceptors for words. Since the definition of AFA are distinct from LTL, both approaches seem different at the first glance. However, it is well established that counter-free AFA, a specific subclass of AFA, are equivalent to LTL in terms of their expressive power. Furthermore, the syntactic structure and semantic interpretation of counter-free AFA, as demonstrated by Camacho and Mcllraith [15, Theorem 1, Property 1], show a striking resemblance to those of LTL, ultimately leading to an encoding that is almost identical to that of Neider and Gavran. They rely on the SAT solver Pycosat [10] for their implementation.

Arif et al. [3] elevate the SAT-based encoding of Neider and Gavran to a syntax-guided synthesis (SyGUS) framework. Such frameworks, employed in programming synthesis, inherently support various search heuristics, including symmetry breaking, rewrite rules, and others. Notably, in contrast to previous work that translates LTL semantics to SAT in a straightforward manner [52], Arif et al. rely on a bit-vector arithmetic-based encoding. They use the CVC4SY solver [63] for their implementation.

Ielo et al. [38] elevate the SAT-based encoding of Neider and Gavran to the Answer Set Programming (ASP) framework, a declarative programming paradigm that allows for defining problems in terms of rules and constraints. They devise two formulations of the passive learning problem in ASP, as an abduction problem and as a context-dependent learning problem. They rely ASP solvers such as CLINGO and ILASP [28] in their implementation.

*Learning from Noisy Data.* To accommodate noisy data, a relaxation of the requirement for the generated formula to be consistent with all examples is necessary. This relaxed consistency criterion is often expressed using metrics of misclassification, such as the loss function $l(\mathcal{S}, \varphi) = \frac{\sum_{u \in P}[u \not\models \varphi] + \sum_{v \in N}[v \models \varphi]}{|P|+|N|}$, where $\mathcal{S} = (P, N)$ and the Iverson bracket $[\ ]$ maps true statements to 1 and false to 0. This loss quantifies the proportion of examples misclassified by the formula $\varphi$ and closely mimic standard loss functions used in statistical machine learning.

To learn minimal LTL formulas that minimize the above loss function, Gaglione et al. [26] propose translating the problem into a Maximum Satisfiability (MaxSAT) instance, mirroring the techniques employed by Neider and Gavran [52] and Riener [64] for propositional logic. MaxSAT extends the classical satisfiability problem of propositional logic, allowing for the definition of hard constraints (mandatory clauses) and soft constraints (optional clauses). The solution to a MaxSAT problem is a variable assignment that satisfies all hard constraints and as many soft constraints as possible. Gaglione et al. capitalize on this technology by designating all clauses in $\Phi_n^{DAG}$ as hard constraints and selected clauses in $\Phi_n^{con}$ as soft constraints. As a result, they obtain a minimal LTL formula that minimizes the specified loss function. They rely on the MaxSAT solving capabilities of Z3 [12] for their implementation.

In fact, by following a similar method, almost all of the constraint-based approaches can potentially be extended to noisy settings if the solver employed allows such relaxations.

*Learning from Positive Examples Only.* The problem of learning from positive examples only is a special case of the one-class learning task, where only one class of inputs (positive or negative) is available. This problem frequently arises in AI applications, particularly in the context of explainability, where one seeks to infer the behavior of an autonomous agent from observational data.

Unlike learning from noisy data, extending constraint-based approaches to learn from only positive examples is not straightforward. The primary reason

for this is that learning LTL formulas from positive examples is an inherently ill-posed problem. Given a set of positive examples $P$, the smallest LTL formula that is consistent with $P$ is the trivial formula *true*, which is satisfied by any word. Clearly, this formula is too general and does not provide any insights into the underlying (temporal) patterns in the examples.

To address this challenge, Roy et al. [66] propose strongness—or specificity—as an additional optimization parameter besides the size of the formula. In particular, the authors solve formulate a learning task wherein, given a set $P$ of positive examples and a size bound $n > 0$, the goal is to learn an LTL formula $\varphi$ that satisfies the following three conditions: (i) each $w \in P$ satisfies $\varphi$, (ii) $\varphi$ has size at most $n$, and (iii) there exists no other formula with the former two properties that implies $\varphi$. To tackle this problem, Roy et al. employ a counterexample-guided inductive synthesis loop [1], which leverages negative examples to guide the learning algorithm towards a most specific LTL formula. In each iteration of the loop, the authors utilize one of the aforementioned SAT-based methods to construct a consistent LTL formula. This formula is then analyzed, and if necessary, used to generate a new negative example that directs the search towards a more specific formula. This iterative process continues until no formulas more specific than the current formula can be found, at which point the algorithm terminates.

Chou et al. [18] propose learning LTL formulas from positive examples of high-dimensional data. They rely on domain specific non-convex cost functions involving the positive examples and LTL formulas to ensure that the prospective formula tightly describes the examples. To search for the prospective formula, they provide a joint encoding of the cost functions and the syntax DAG of LTL formulas using Mixed Integer Linear Programming (MILP). They rely on the MILP solver IPOPT [71] for their implementation.

## 4 Enumeration-Based Approaches

The constraint-based approaches discussed in Sect. 3 provide a systematic method for learning arbitrary LTL formulas. However, the performance of these approaches is limited by the capabilities of the underlying solvers. The search techniques typically employed in the solvers are not optimized for learning LTL formulas, thus often leading to bottlenecks in scalability.

As a result, recent works have started exploring alternative search strategies that are tailored to navigate through the search space of LTL formulas efficiently. These approaches search through relevant/interesting LTL formulas in a more targeted manner that results in scalability, typically at the expense of the minimality of the learned formulas.

A prominent example of this approach is Scarlet, a tool developed by Raha et al. [60,61] that detects and accumulates common temporal patterns in a given sample. For instance, analyzing a sample consisting of a positive word $u = \{p\}\{p\}\{q\}\{p\}\{r\}\{p\}$ and a negative word $v = \{p\}\{p\}\{r\}\{p\}\{q\}$, Scarlet extracts the formula $\mathbf{F}(q \wedge \mathbf{F}(r))$, which captures the order in which the propositions $q$ and $r$ appear. By employing dynamic programming, the tool identifies

a large number of such patterns, which are then translated into a simple yet expressive LTL fragment named directed LTL. Scarlet then combines a suitable selection of directed LTL formulas to construct a consistent formula using a novel procedure called Boolean subset cover. Unlike constraint-based approaches, Scarlet's search strategy integrates syntax and semantics computations in a single, unified process, resulting in a more efficient and effective method for learning LTL formulas.

Another notable example is the highly parallelized algorithm developed by Valizadeh et al. [69], which is designed to leverage the processing power of Graphics Processing Units (GPUs). Their approach comprises two pivotal procedures: *relaxed unique checks* (RUCs) and *divide and conquer* (D&C). The RUCs procedure performs a bottom-up search through the syntax of LTL formulas, eliminating redundant formulas that exhibit the same behavior on the given sample. Since this procedure is resource-intensive, it cannot be easily extended to large samples. To mitigate this, the D&C procedure partitions the sample into smaller, manageable subsets on which RUCs can be applied in parallel. The resulting formulas can then be combined using Boolean combinations to generate one consistent LTL formula. Internally, Valizadeh et al.'s approach employs bit-vectors to encode the semantics of LTL formulas, which can be highly efficiently implemented on GPUs. By exploiting the parallel processing capabilities of GPUs, the authors achieve a significant speedup, making their approach perhaps the most scalable one of all.

Ghiorzi et al. [30] propose a range of heuristics to expedite the enumeration of LTL formulas. Inspired by Riener [64], the authors first employ an enumeration strategy based on partial DAGs to navigate the search space quickly. Then, they utilize LTL rewrite rules to eliminate equivalent and redundant formulas, leveraging rules such as $\varphi \wedge \neg\varphi \equiv \mathit{false}$ and $\neg\, \mathbf{F}\, \varphi \equiv \mathbf{G}(\neg\varphi)$. Additionally, the authors efficiently eliminate tautologies and contradictions by alternately checking the satisfaction of enumerated LTL formulas on positive and negative examples.

## 5 Neural Network-Based Approaches

Recent research also focuses on leveraging the optimized training capabilities of neural networks to achieve scalability in the LTL learning process. However, due to the inherent uncertainty of neural network training, these approaches lack theoretical guarantees regarding the consistency of the learned LTL formulas. Nonetheless, they can produce reasonably good LTL formulas from large, typically noisy datasets.

The current approaches specifically exploit Graph Neural Networks (GNNs) [36] to learn LTL formulas. GNNs are a powerful neural architecture that learns vector representations of vertex and edge features, typically called embeddings. More formally, GNNs define a message-passing process between vertices in a graph, where each vertex aggregates information from its neighbors to update its own representation. This process is repeated several times, allowing the model to learn complex patterns and relationships between vertex and edge features.

The critical insight to understanding the connection of LTL and GNNs is to view a word $w = a_1 a_2 \dots a_n$ as a linear graph $v_1 \to v_2 \cdots \to v_n$ with $n$ nodes. This representation allows associating a feature vector $x_i$ to each node $v_i$ that tracks the satisfaction of the different subformulas of a prospective LTL formula $\varphi$ when evaluated at the $i$-position of an example (see the definition of the model relation on Page 4). By leveraging message passing, the satisfaction of the entire formula $\varphi$ can be computed by aggregating the feature vectors of nodes $v_j$ with $j > i$ according to the semantics of LTL.

Luo et al. [48] built upon the insight of representing words as linear graphs to train a GNN on a sample $\mathcal{S} = (P, N)$. Subsequently, the authors use the learned network weights to extract an LTL formula that closely approximates the behavior of the GNN on the given sample.

Although Luo et al.'s work pioneered the use of GNNs for learning LTL formulas, it suffers from the limitation that the extracted LTL formula might accurately capture the behavior of the trained GNN. Wan et al. [72] address this shortcoming by introducing an enhanced architecture. In particular, the authors devise a faithful encoding of the LTL semantics within the GNN architecture, achieved through parametric constraints on the network weights. This innovative encoding ensures that a consistent LTL formula can always be reliably extracted from the trained GNN.

## 6 Other Settings

Our discussion thus far has centered around the classical passive learning problem for LTL as defined in Definition 1. However, several variants of this problem have been explored, each presenting unique challenges. In this section, we discuss three such extensions, highlighting their distinct characteristics and proposed solutions.

### 6.1 Mining LTL Based on Templates

A key property of Definition 1 is that it makes no restrictions on the syntactic structure of an LTL formula as long as this formula is consistent with the given sample. In practice, however, users sometimes want to incorporate domain knowledge into the learning process or must confine the solutions to specific LTL fragments. Unfortunately, the methods discussed in Sects. 3 to 5 do not offer this level of fine-grained control and, thus, cannot be applied in such situations.

To address this limitation, researchers have proposed methods that allow users to provide a partial formula, typically referred to as a template or sketch, where parts can be omitted (typically indicated by a question mark). In this framework, the task of a learning algorithm is then to infer the missing part of a template (or sketch) so that the completed formula is consistent with a given set of examples. Notable examples operating in this setting are the methods by Lemieux et al. [44] and Lutz et al. [49]. The former approach considers templates, called *property types*, in which question marks serve as placeholders for missing

propositions. By contrast, the latter approach introduces so-called *sketches* in which placeholders can represent missing operators and even entire subformulas.

For example, Li et al. [45] have developed a method to mine LTL specifications in the GR(1) fragment of LTL, using templates that include $\mathbf{G}\,\mathbf{F}?$, $\mathbf{G}(? \rightarrow \mathbf{X}?)$, and others. Similarly, Shah et al. [68] have focused on conjunctive LTL formulas built from a limited set of typical temporal properties collected by Dwyer et al. [22]. Another notable example is the work by Kim et al. [39], which considers a set of interpretable LTL templates originating in software system development and seeks to infer formulas robust to noise in the input data.

### 6.2 Mining LTL from Natural Language

One of the significant barriers to adopting temporal logic in practice is the limited expertise of practitioners and engineers in this area [34,37]. As a result, they often prefer to specify their requirements in natural language, which is more intuitive and accessible to them. Several research efforts have focused on bridging this gap by automatically extracting LTL formulas from natural language descriptions. Early approaches [24,31,33,43,55] achieved this by efficiently parsing English sentences to translate them into LTL and other temporal logic formulas. The advent of data-driven techniques has led to the development of neural-network-based methods [17,35,56] that rely on human-labeled pairs of natural language descriptions and corresponding logic formulas. More recently, researchers have begun to leverage the impressive natural language understanding capabilities of Large Language Models to enhance the translation capabilities further [21,25, 46,57], offering promise for more effective and efficient property specification.

### 6.3 Logics Beyond LTL

The widespread adoption of continuous-time logics, such as Signal Temporal Logic (STL), in the context of cyber-physical systems has spawned a significant body of research focused on learning specifications in STL. In fact, a comprehensive survey by Bartocci et al. [9] is dedicated entirely to this problem. Most of these works concentrate on learning formulas with a specific syntactic structure [13] or identifying time intervals for given STL formulas [4,41,42]. A handful of works also tackle the more general passive learning problem, where the goal is to learn STL formulas of arbitrary structure [50,54].

In addition to linear-time properties, there exist several works focusing on learning branching-time properties in Computation Tree Logic (CTL). For instance, Chan [16] addresses the problem of completing simple CTL templates, while Wasylkowski and Zeller [73] investigate inferring operational preconditions for Java methods in CTL. Recent research by Pommellet et al. [59] and Bordais et al. [14] demonstrate that constraint-based techniques can be used to learn not only CTL but also Alternating-time Temporal Logics (ATL), which extends CTL for multi-agent systems.

Apart from LTL, STL and CTL, there is also research on learning formulas in other temporal logics, such as Metric Temporal Logic (MTL) [62], and the Property Specification Language (PSL) [65].

## 7 Conclusion

This survey provides a comprehensive overview of the diverse research efforts focused on learning specifications in temporal logic, with a particular emphasis on Linear Temporal Logic. We systematically compared and contrasted these works based on their search strategies to navigate the vast space of possible formulas. Some approaches leverage off-the-shelf solvers, while others propose customized enumeration techniques or exploit advances in deep learning to facilitate the learning process. By synthesizing the strengths and limitations of these approaches, we aim to provide a roadmap for future research in this exciting and rapidly evolving field.

**Acknowledgements.** Rajarshi Roy acknowledges partial funding by the ERC under the European Union's Horizon 2020 research and innovation programme (grant agreement No.834115, FUN2MODEL) and Daniel Neider acknowledges funding by Deutsche Forschungsgemeinschaft (DFG) (grant number 459419731).

## References

1. Alur, R., et al.: Syntax-guided synthesis. In: Formal Methods in Computer-Aided Design, FMCAD 2013, Portland, OR, USA, 20–23 October 2013, pp. 1–8. IEEE (2013). https://ieeexplore.ieee.org/document/6679385/
2. Ammons, G., Bodík, R., Larus, J.R.: Mining specifications. In: Launchbury, J., Mitchell, J.C. (eds.) Conference Record of POPL 2002: The 29th SIGPLAN-SIGACT Symposium on Principles of Programming Languages, Portland, OR, USA, 16–18 January 2002, pp. 4–16. ACM (2002). https://doi.org/10.1145/503272.503275
3. Arif, M.F., Larraz, D., Echeverria, M., Reynolds, A., Chowdhury, O., Tinelli, C.: SYSLITE: syntax-guided synthesis of PLTL formulas from finite traces. In: FMCAD, pp. 93–103. IEEE (2020)
4. Asarin, E., Donzé, A., Maler, O., Nickovic, D.: Parametric identification of temporal properties. In: Khurshid, S., Sen, K. (eds.) RV 2011. LNCS, vol. 7186, pp. 147–160. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-29860-8_12
5. Bacherini, S., Fantechi, A., Tempestini, M., Zingoni, N.: A story about formal methods adoption by a railway signaling manufacturer. In: Misra, J., Nipkow, T., Sekerinski, E. (eds.) FM 2006. LNCS, vol. 4085, pp. 179–189. Springer, Heidelberg (2006). https://doi.org/10.1007/11813040_13
6. Badeau, F., Amelot, A.: Using B as a high level programming language in an industrial project: Roissy VAL. In: Treharne, H., King, S., Henson, M., Schneider, S. (eds.) ZB 2005. LNCS, vol. 3455, pp. 334–354. Springer, Heidelberg (2005). https://doi.org/10.1007/11415787_20
7. Baier, C., Katoen, J.: Principles of Model Checking. MIT Press, Cambridge (2008)

8. Barrett, C., et al.: CVC4. In: Gopalakrishnan, G., Qadeer, S. (eds.) CAV 2011. LNCS, vol. 6806, pp. 171–177. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-22110-1_14
9. Bartocci, E., Mateis, C., Nesterini, E., Nickovic, D.: Survey on mining signal temporal logic specifications. Inf. Comput. **289**(Part), 104957 (2022). https://doi.org/10.1016/J.IC.2022.104957
10. Biere, A.: Picosat essentials. J. Satisf. Boolean Model. Comput. **4**(2-4), 75–97 (2008). https://doi.org/10.3233/SAT190039
11. Bjørner, D., Havelund, K.: 40 years of formal methods. In: Jones, C., Pihlajasaari, P., Sun, J. (eds.) FM 2014. LNCS, vol. 8442, pp. 42–61. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-06410-9_4
12. Bjørner, N.S., Phan, A.: $\nu$z - maximal satisfaction with Z3. In: Kutsia, T., Voronkov, A. (eds.) 6th International Symposium on Symbolic Computation in Software Science, SCSS 2014, Gammarth, La Marsa, Tunisia, 7–8 December 2014. EPiC Series in Computing, vol. 30, pp. 1–9. EasyChair (2014). https://doi.org/10.29007/JMXJ
13. Bombara, G., Vasile, C.I., Penedo, F., Yasuoka, H., Belta, C.: A decision tree approach to data classification using signal temporal logic. In: Proceedings of the 19th International Conference on Hybrid Systems: Computation and Control, HSCC 2016, pp. 1–10. Association for Computing Machinery, New York (2016). https://doi.org/10.1145/2883817.2883843
14. Bordais, B., Neider, D., Roy, R.: Learning branching-time properties in CTL and ATL via constraint solving. CoRR **abs/2406.19890** (2024). https://doi.org/10.48550/ARXIV.2406.19890
15. Camacho, A., McIlraith, S.A.: Learning interpretable models expressed in linear temporal logic. In: ICAPS, pp. 621–630. AAAI Press (2019)
16. Chan, W.: Temporal-logic queries. In: Emerson, E.A., Sistla, A.P. (eds.) CAV 2000. LNCS, vol. 1855, pp. 450–463. Springer, Heidelberg (2000). https://doi.org/10.1007/10722167_34
17. Cherukuri, H., Ferrari, A., Spoletini, P.: Towards explainable formal methods: from LTL to natural language with neural machine translation. In: Gervasi, V., Vogelsang, A. (eds.) REFSQ 2022. LNCS, vol. 13216, pp. 79–86. Springer, Heidelberg (2022). https://doi.org/10.1007/978-3-030-98464-9_7
18. Chou, G., Ozay, N., Berenson, D.: Learning temporal logic formulas from suboptimal demonstrations: theory and experiments. Auton. Robots **46**(1), 149–174 (2022). https://doi.org/10.1007/S10514-021-10004-X
19. Clarke, E.M., Biere, A., Raimi, R., Zhu, Y.: Bounded model checking using satisfiability solving. Formal Methods Syst. Des. **19**(1), 7–34 (2001). https://doi.org/10.1023/A:1011276507260
20. Cofer, D., Miller, S.: DO-333 certification case studies. In: Badger, J.M., Rozier, K.Y. (eds.) NFM 2014. LNCS, vol. 8430, pp. 1–15. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-06200-6_1
21. Cosler, M., Hahn, C., Mendoza, D., Schmitt, F., Trippel, C.: nl2spec: interactively translating unstructured natural language to temporal logics with large language models. In: Enea, C., Lal, A. (eds.) CAV 2023, Part II. LNCS, vol. 13965, pp. 383–396. Springer, Heidelberg (2023). https://doi.org/10.1007/978-3-031-37703-7_18
22. Dwyer, M.B., Avrunin, G.S., Corbett, J.C.: Property specification patterns for finite-state verification. In: Ardis, M.A., Atlee, J.M. (eds.) Proceedings of the Second Workshop on Formal Methods in Software Practice, Clearwater Beach, Florida, USA, 4–5 March 1998, pp. 7–15. ACM (1998). https://doi.org/10.1145/298595.298598

23. Fecko, M.A., et al.: A success story of formal description techniques: Estelle specification and test generation for MIL-STD 188–220. Comput. Commun. **23**(12), 1196–1213 (2000)
24. Finucane, C., Jing, G., Kress-Gazit, H.: LTLMoP: experimenting with language, temporal logic and robot control. In: 2010 IEEE/RSJ International Conference on Intelligent Robots and Systems, Taipei, Taiwan, 18–22 October 2010, pp. 1988–1993. IEEE (2010). https://doi.org/10.1109/IROS.2010.5650371
25. Fuggitti, F., Chakraborti, T.: NL2LTL - a Python package for converting natural language (NL) instructions to linear temporal logic (LTL) formulas. In: Proceedings of the Thirty-Seventh AAAI Conference on Artificial Intelligence and Thirty-Fifth Conference on Innovative Applications of Artificial Intelligence and Thirteenth Symposium on Educational Advances in Artificial Intelligence, AAAI 2023/IAAI 2023/EAAI 2023. AAAI Press (2023). https://doi.org/10.1609/aaai.v37i13.27068
26. Gaglione, J.-R., Neider, D., Roy, R., Topcu, U., Xu, Z.: Learning linear temporal properties from noisy data: a MaxSAT-based approach. In: Hou, Z., Ganesh, V. (eds.) ATVA 2021. LNCS, vol. 12971, pp. 74–90. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-88885-5_6
27. Gario, M., Cimatti, A., Mattarei, C., Tonetta, S., Rozier, K.Y.: Model checking at scale: automated air traffic control design space exploration. In: Chaudhuri, S., Farzan, A. (eds.) CAV 2016. LNCS, vol. 9780, pp. 3–22. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-41540-6_1
28. Gebser, M., Kaminski, R., Kaufmann, B., Schaub, T.: Answer Set Solving in Practice. Synthesis Lectures on Artificial Intelligence and Machine Learning, Morgan & Claypool Publishers (2012). https://doi.org/10.2200/S00457ED1V01Y201211AIM019
29. Gelfond, M., Lifschitz, V.: Classical negation in logic programs and disjunctive databases. New Gener. Comput. **9**(3/4), 365–386 (1991). https://doi.org/10.1007/BF03037169
30. Ghiorzi, E., Colledanchise, M., Piquet, G., Bernagozzi, S., Tacchella, A., Natale, L.: Learning linear temporal properties for autonomous robotic systems. IEEE Robotics Autom. Lett. **8**(5), 2930–2937 (2023). https://doi.org/10.1109/LRA.2023.3263368
31. Ghosh, S., Elenius, D., Li, W., Lincoln, P., Shankar, N., Steiner, W.: ARSENAL: automatic requirements specification extraction from natural language. In: NASA Formal Methods, NFM (2016). https://doi.org/10.1007/978-3-319-40648-0_4
32. Giacomo, G.D., Vardi, M.Y.: Linear temporal logic and linear dynamic logic on finite traces. In: Rossi, F. (ed.) IJCAI 2013, Proceedings of the 23rd International Joint Conference on Artificial Intelligence, Beijing, China, 3–9 August 2013, pp. 854–860. IJCAI/AAAI (2013). http://www.aaai.org/ocs/index.php/IJCAI/IJCAI13/paper/view/6997
33. Giannakopoulou, D., Pressburger, T., Mavridou, A., Rhein, J., Schumann, J., Shi, N.: Formal requirements elicitation with FRET. In: International Conference on Requirements Engineering: Foundation for Software Quality, REFSQ (2020). http://ceur-ws.org/Vol-2584/PT-paper4.pdf
34. Greenman, B., Saarinen, S., Nelson, T., Krishnamurthi, S.: Little tricky logic: misconceptions in the understanding of LTL. Art Sci. Eng. Program. **7**(2) (2023)
35. Hahn, C., Schmitt, F., Kreber, J.U., Rabe, M.N., Finkbeiner, B.: Teaching temporal logics to neural networks. In: 9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, 3–7 May 2021. OpenReview.net (2021). https://openreview.net/forum?id=dOcQK-f4byz

36. Hamilton, W.L., Ying, R., Leskovec, J.: Representation learning on graphs: methods and applications. IEEE Data Eng. Bull. **40**(3), 52–74 (2017). http://sites.computer.org/debull/A17sept/p52.pdf
37. Holzmann, G.J.: The logic of bugs. In: SIGSOFT FSE, pp. 81–87. ACM (2002)
38. Ielo, A., Law, M., Fionda, V., Ricca, F., De Giacomo, G., Russo, A.: Towards ILP-based LTLF passive learning. In: Bellodi, E., Lisi, F.A., Zese, R. (eds.) ILP 2023. LNCS, vol. 14363, pp. 30–45. Springer, Heidelberg (2023). https://doi.org/10.1007/978-3-031-49299-0_3
39. Kim, J., Muise, C., Shah, A., Agarwal, S., Shah, J.: Bayesian inference of linear temporal logic specifications for contrastive explanations. In: IJCAI, pp. 5591–5598. ijcai.org (2019)
40. Klein, G., et al.: seL4: formal verification of an operating-system kernel. Commun. ACM **53**(6), 107–115 (2010)
41. Kong, Z., Jones, A., Belta, C.: Temporal logics for learning and detection of anomalous behavior. IEEE Trans. Autom. Control **62**(3), 1210–1222 (2017). https://doi.org/10.1109/TAC.2016.2585083
42. Kong, Z., Jones, A., Medina Ayala, A., Aydin Gol, E., Belta, C.: Temporal logic inference for classification and prediction from data. In: Proceedings of the 17th International Conference on Hybrid Systems: Computation and Control, HSCC 2014, pp. 273–282. Association for Computing Machinery, New York (2014). https://doi.org/10.1145/2562059.2562146
43. Kress-Gazit, H., Fainekos, G.E., Pappas, G.J.: Translating structured English to robot controllers. Adv. Robot. **22**(12), 1343–1359 (2008)
44. Lemieux, C., Beschastnikh, I.: Investigating program behavior using the texada LTL specifications miner. In: Cohen, M.B., Grunske, L., Whalen, M. (eds.) 30th IEEE/ACM International Conference on Automated Software Engineering, ASE 2015, Lincoln, NE, USA, 9–13 November 2015, pp. 870–875. IEEE Computer Society (2015). https://doi.org/10.1109/ASE.2015.94
45. Li, W., Dworkin, L., Seshia, S.A.: Mining assumptions for synthesis. In: MEMOCODE, pp. 43–50. IEEE (2011)
46. Liu, J.X., et al.: Lang2LTL: translating natural language commands to temporal robot task specification. CoRR **abs/2302.11649** (2023). https://doi.org/10.48550/ARXIV.2302.11649
47. Lowe, G.: Breaking and fixing the Needham-Schroeder public-key protocol using FDR. Softw. Concepts Tools **17**(3), 93–102 (1996)
48. Luo, W., Liang, P., Du, J., Wan, H., Peng, B., Zhang, D.: Bridging LTLf inference to GNN inference for learning LTLf formulae. In: AAAI, pp. 9849–9857. AAAI Press (2022)
49. Lutz, S., Neider, D., Roy, R.: Specification sketching for linear temporal logic. In: André, É., Sun, J. (eds.) ATVA 2023, Part II. LNCS, vol. 14216, pp. 26–48. Springer, Cham (2023). https://doi.org/10.1007/978-3-031-45332-8_2
50. Mohammadinejad, S., Deshmukh, J.V., Puranic, A.G., Vazquez-Chanlatte, M., Donzé, A.: Interpretable classification of time-series data using efficient enumerative techniques. In: HSCC 2020: 23rd ACM International Conference on Hybrid Systems: Computation and Control, Sydney, New South Wales, Australia, 21–24 April 2020, pp. 9:1–9:10. ACM (2020). https://doi.org/10.1145/3365365.3382218
51. de Moura, L., Bjørner, N.: Z3: an efficient SMT solver. In: Ramakrishnan, C.R., Rehof, J. (eds.) TACAS 2008. LNCS, vol. 4963, pp. 337–340. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-78800-3_24

52. Neider, D., Gavran, I.: Learning linear temporal properties. In: Bjørner, N.S., Gurfinkel, A. (eds.) 2018 Formal Methods in Computer Aided Design, FMCAD 2018, Austin, TX, USA, October 30 - November 2, 2018, pp. 1–10. IEEE (2018). https://doi.org/10.23919/FMCAD.2018.8603016
53. Nemhauser, G.L., Wolsey, L.A.: Integer and Combinatorial Optimization. Wiley Interscience Series in Discrete Mathematics and Optimization. Wiley (1988). https://doi.org/10.1002/9781118627372
54. Nenzi, L., Silvetti, S., Bartocci, E., Bortolussi, L.: A robust genetic algorithm for learning temporal specifications from data. In: McIver, A., Horvath, A. (eds.) QEST 2018. LNCS, vol. 11024, pp. 323–338. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-99154-2_20
55. Nikora, A.P., Balcom, G.: Automated identification of LTL patterns in natural language requirements. In: ISSRE 2009, 20th International Symposium on Software Reliability Engineering, Mysuru, Karnataka, India, 16–19 November 2009, pp. 185–194. IEEE Computer Society (2009). https://doi.org/10.1109/ISSRE.2009.15
56. Oh, Y., Patel, R., Nguyen, T., Huang, B., Pavlick, E., Tellex, S.: Planning with state abstractions for non-Markovian task specifications. In: Bicchi, A., Kress-Gazit, H., Hutchinson, S. (eds.) Robotics: Science and Systems XV, University of Freiburg, Freiburg im Breisgau, Germany, 22–26 June 2019 (2019). https://doi.org/10.15607/RSS.2019.XV.059
57. Pan, J., Chou, G., Berenson, D.: Data-efficient learning of natural language to linear temporal logic translators for robot task specification. In: IEEE International Conference on Robotics and Automation, ICRA 2023, London, UK, May 29 - June 2, 2023, pp. 11554–11561. IEEE (2023). https://doi.org/10.1109/ICRA48891.2023.10161125
58. Pnueli, A.: The temporal logic of programs. In: 18th Annual Symposium on Foundations of Computer Science, Providence, Rhode Island, USA, 31 October - 1 November 1977, pp. 46–57. IEEE Computer Society (1977). https://doi.org/10.1109/SFCS.1977.32
59. Pommellet, A., Stan, D., Scatton, S.: SAT-based learning of computation tree logic. In: Benzmüller, C., Heule, M.J.H., Schmidt, R.A. (eds.) IJCAR 2024, Part I. LNCS, vol. 14739, pp. 366–385. Springer, Cham (2024). https://doi.org/10.1007/978-3-031-63498-7_22
60. Raha, R., Roy, R., Fijalkow, N., Neider, D.: Scalable anytime algorithms for learning fragments of linear temporal logic. In: Fisman, D., Rosu, G. (eds.) TACAS 2022. LNCS, vol. 13243, pp. 263–280. Springer International Publishing, Cham (2022). https://doi.org/10.1007/978-3-030-99524-9_14
61. Raha, R., Roy, R., Fijalkow, N., Neider, D.: Scarlet: scalable anytime algorithms for learning fragments of linear temporal logic. J. Open Source Softw. **9**(93), 5052 (2024). https://doi.org/10.21105/JOSS.05052
62. Raha, R., Roy, R., Fijalkow, N., Neider, D., Pérez, G.A.: Synthesizing efficiently monitorable formulas in metric temporal logic. In: Dimitrova, R., Lahav, O., Wolff, S. (eds.) VMCAI 2024. LNCS, vol. 14500, pp. 264–288. Springer, Cham (2024). https://doi.org/10.1007/978-3-031-50521-8_13
63. Reynolds, A., Barbosa, H., Nötzli, A., Barrett, C., Tinelli, C.: CVC4SY: smart and fast term enumeration for syntax-guided synthesis. In: Dillig, I., Tasiran, S. (eds.) CAV 2019, Part II. LNCS, vol. 11562, pp. 74–83. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-25543-5_5
64. Riener, H.: Exact synthesis of LTL properties from traces. In: FDL, pp. 1–6. IEEE (2019)

65. Roy, R., Fisman, D., Neider, D.: Learning interpretable models in the property specification language. In: IJCAI, pp. 2213–2219. ijcai.org (2020)
66. Roy, R., Gaglione, J., Baharisangari, N., Neider, D., Xu, Z., Topcu, U.: Learning interpretable temporal properties from positive examples only. CoRR **abs/2209.02650** (2022)
67. Rozier, K.Y.: Specification: the biggest bottleneck in formal methods and autonomy. In: Blazy, S., Chechik, M. (eds.) VSTTE 2016. LNCS, vol. 9971, pp. 8–26. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-48869-1_2
68. Shah, A., Kamath, P., Shah, J.A., Li, S.: Bayesian inference of temporal task specifications from demonstrations. In: NeurIPS, pp. 3808–3817 (2018)
69. Valizadeh, M., Fijalkow, N., Berger, M.: LTL learning on GPUS. In: Gurfinkel, A., Ganesh, V. (eds.) CAV 2024. LNCS, vol. 14683, pp. 209–231. Springer, Cham (2024). https://doi.org/10.1007/978-3-031-65633-0_10
70. Verhulst, E., de Jong, G.: OpenComRTOS: an ultra-small network centric embedded RTOS designed using formal modeling. In: Gaudin, E., Najm, E., Reed, R. (eds.) SDL 2007. LNCS, vol. 4745, pp. 258–271. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-74984-4_16
71. Wächter, A., Biegler, L.T.: On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. Math. Program. **106**(1), 25–57 (2006). https://doi.org/10.1007/S10107-004-0559-Y
72. Wan, H., Liang, P., Du, J., Luo, W., Ye, R., Peng, B.: End-to-end learning of LTLF formulae by faithful LTLF encoding. In: AAAI, pp. 9071–9079. AAAI Press (2024)
73. Wasylkowski, A., Zeller, A.: Mining temporal specifications from object usage. Autom. Softw. Eng. **18**(3–4), 263–292 (2011)