Using Four-Valued Signal Temporal Logic for Incremental Verification of Hybrid Systems

Florian Lercher(✉) [0009−0007−7202−5417] and
Matthias Althoff[0000−0003−3733−842X]

School for Computation, Information and Technology,
Technical University of Munich, 85748 Garching, Germany
florian.lercher@tum.de, althoff@tum.de

Abstract. Hybrid systems are often safety-critical and at the same time difficult to formally verify due to their mixed discrete and continuous behavior. To address this issue, we propose a novel incremental verification algorithm for hybrid systems based on online monitoring techniques and reachability analysis. To this end, we develop a four-valued semantics for signal temporal logic that allows us to distinguish two types of uncertainty: one arising from set-based evaluation and another one from the incremental nature of our algorithm. Using these semantics to continuously update the verification verdict, our verification algorithm is the first to run alongside the reachability analysis of the system to be verified. This makes it possible to stop the reachability analysis as soon as we obtain a conclusive verdict. We demonstrate the usefulness of our novel approach by several experiments.

Keywords: Hybrid systems verification · Many-valued temporal logic · Online verification.

1 Introduction

Hybrid systems are a powerful modeling concept, as they can exhibit both continuous and discrete dynamics. As such, they are applicable in many contexts, including autonomous vehicles, power systems, robotics, and systems biology. As the typical application areas indicate, hybrid systems are often safety-critical and thus require formal verification. Specifications for the formal verification of hybrid systems are often formalized using signal temporal logic (STL) [26], which is evaluated on real-valued signals over continuous time. STL *monitoring* algorithms can determine whether a concrete execution of a system satisfies an STL specification [8]. By considering the *reachable set*, i.e., the set of states reached by at least one execution, rather than single executions, monitoring algorithms can be adapted to verify a specification for all executions [22, 35]. While [22, 35] focus on offline monitoring algorithms, we adapt an online algorithm for incremental verification. This allows us to stop the computation of the reachable set as soon as the specification can be verified or falsified. Thus, we can use our novel method online to, e.g., verify motion plans of autonomous vehicles [3].

This version of the contribution has been accepted for publication, after peer review (when applicable) but is not the Version of Record and does not reflect post-acceptance improvements, or any corrections. The Version of Record is available online at: http://dx.doi.org/10.1007/978-3-031-65633-0_12

1.1 Related Work

Online Monitoring of Real-Time Temporal Logics: The paper that originally introduced STL also presents an offline monitoring algorithm propagating satisfaction signals of atomic subformulas up the syntax tree of the specification [26]. Later, [27] calls this method *offline marking* and adapts it for online monitoring. The new procedure, called *incremental marking*, essentially performs offline marking for each new observation and discards the already propagated parts of the signals. The tool AMT [28] implements both algorithms. Other online monitoring approaches for real-time logics rely on translating the formula to timed [10,17] or untimed [20] automata. The algorithm in [34] rewrites the monitored metric temporal logic formula to represent remaining constraints whenever an observation is made. For robust monitoring, [15] adapts incremental marking to quantitative semantics of STL [16].

Many-Valued Semantics of Temporal Logics: In the context of monitoring, [10] employs three-valued semantics for linear temporal logic (LTL) and timed LTL to handle uncertainty due to finite traces. To obtain a more expressive monitoring result for finite traces, [9] extends this to a four-valued semantics that distinguishes *presumably* true or false finite traces. The authors of [13] use a five-valued semantics of LTL to deal with uncertainties arising from finite traces and race conditions in parallel systems. Most closely related to our approach are [22, 35], which employ three-valued semantics for verification of hybrid systems. Based on reachable sets, previous work constructs three-valued satisfaction signals for the atomic predicates of an STL formula implicitly [22] or explicitly [35]. The third truth value indicates that both satisfying and violating states are reachable. To decide whether the specification is met, these are propagated akin to offline marking. Moreover, [35] employs statically determined *masks* to evaluate atomic predicates only where they are relevant; masking is an orthogonal approach to our proposed incremental verification.

Hybrid Systems Verification: Besides the aforementioned approaches based on three-valued semantics of STL [22,35], there are other verification methods using only the usual two truth values. The authors of [32] introduce a variant of STL called *reachset temporal logic* that is interpreted directly over the reachable set. They provide a sound transformation of STL into their logic, which is complete if all intervals in the STL formula range from 0 to a globally fixed time step. In [7], the authors propose a syntactic separation procedure for STL, which splits a formula into subformulas referring to disjoint time intervals. Based on the separated formula, they use satisfiability modulo theories (SMT) techniques to search for counterexamples bounded in length and the number of value changes; the SMT encoding is improved in [25]. Finally, there are deductive verification approaches that adapt dynamic logic suitable for software verification into *differential dynamic logic* suitable for hybrid systems [29, 30]. Differential dynamic logic has been augmented with a fragment of STL to derive temporal properties [1] and assumption-commitment reasoning to handle parallel hybrid systems [12].

1.2 Contributions

We propose a novel algorithm for verifying STL specifications on hybrid systems based on reachability analysis. Following the idea of the incremental marking procedure for STL monitoring [27, Sec. 3.2], our algorithm runs alongside the reachability analysis. As soon as the reachability algorithm determines the reachable set for new time steps, our algorithm uses the new information to update its verdict on specification satisfaction. Thus, we can terminate the reachability analysis as soon as we obtain a conclusive verdict. The theoretical foundation of our algorithm is a novel four-valued semantics for STL. The two new truth values handle uncertainty arising from set-based (sets might contain both states satisfying and violating a predicate) and incremental (the entire reachable set over time is not immediately available) computation.

This paper is organized as follows: After discussing preliminaries and our problem statement in Sec. 2, we give an overview of our solution concept in Sec. 3. We present our four-valued semantics for STL in Sec. 4, followed by the novel incremental verification algorithm in Sec. 5. In Sec. 6, we apply a prototype implementation to systems occurring in autonomous driving and systems biology before coming to a conclusion in Sec. 7.

2 Preliminaries and Problem Statement

After introducing the necessary interval operations, we establish the required truth values. We then define signals as functions over time and briefly discuss set-based reachability analysis of hybrid systems. Finally, we recapitulate the syntax and Boolean semantics of STL before providing our problem statement.

2.1 Intervals

We work with intervals over $\mathbb{R}$, admitting ∞ and $-\infty$ as endpoints if the interval is open. The *left-closure* $\mathrm{cl}_\mathtt{l}(I)$ of an interval I always includes its left endpoint, except if the endpoint is infinite (e.g., $\mathrm{cl}_\mathtt{l}((a,b]) = [a,b]$ if $a \neq -\infty$). Analogously, the *right-closure* $\mathrm{cl}_\mathtt{r}(I)$ always includes the right endpoint.

For sets $\mathcal{A}$ and $\mathcal{B}$, their *Minkowski sum* $\mathcal{A} \oplus \mathcal{B}$ is $\{a + b \mid a \in \mathcal{A}, b \in \mathcal{B}\}$. We will write $a \oplus \mathcal{B}$ instead of $\{a\} \oplus \mathcal{B}$. We also use $\mathcal{A} \oplus (-\mathcal{B})$ for *back shifting* [26], where $-\mathcal{B} := \{-b \mid b \in \mathcal{B}\}$. If $\mathcal{A}$ and $\mathcal{B}$ are intervals, so are $\mathcal{A} \oplus \mathcal{B}$ and $-\mathcal{B}$.

2.2 Truth Values

We use the values $\mathbb{B} := \{\top, \bot\}$ to denote *truth* $\top$ and *falsehood* $\bot$. By extending the semantics of the usual Boolean connectives to handle a third value $\dashv_1$ denoting *unknown*, we can indicate that a statement could be true or false. This results in a three-valued propositional logic, such as that of Kleene [23]. For uncertainty arising from incremental computations, we add a fourth value $\dashv_2$ to denote *inconclusive*, indicating that the statement is either true, false, or unknown. In

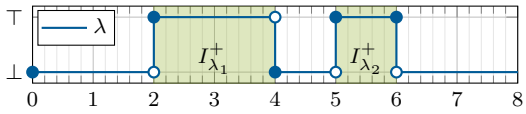


Fig. 1. A Boolean signal λ and its unitary decomposition $\{\lambda_1, \lambda_2\}$

other words, $\dashv_1$ means "we know that we don't know," while $\dashv_2$ means "we don't know whether we don't know." We define the sets of truth values $\mathbb{U}_1 := \mathbb{B} \cup \{\dashv_1\}$ and $\mathbb{U}_2 := \mathbb{U}_1 \cup \{\dashv_2\}$. Moreover, we introduce the *truth order* $\sqsubseteq_\mathtt{t}$, where $v \sqsubseteq_\mathtt{t} v'$ for $v, v' \in \mathbb{U}_2$ means that v is "less true" than v'. Thus, we define $\bot \sqsubseteq_\mathtt{t} \dashv_1 \sqsubseteq_\mathtt{t} \top$ and $\bot \sqsubseteq_\mathtt{t} \dashv_2 \sqsubseteq_\mathtt{t} \top$; $\dashv_1$ and $\dashv_2$ are incomparable.

2.3 Signals

Let us fix $\mathbb{R}_{\geq 0}$ as our *time domain*. A *signal* over the domain $\mathcal{D}$, or $\mathcal{D}$-signal for short, is a function $\sigma : \mathbb{R}_{\geq 0} \to \mathcal{D}$. A *partial* $\mathcal{D}$-signal $\tilde{\sigma} : \mathcal{T} \to \mathcal{D}$ is only defined over a subset $\mathcal{T} \subseteq \mathbb{R}_{\geq 0}$ of the time domain. We refer to signals over $\mathbb{B}$, $\mathbb{U}_1$, and $\mathbb{U}_2$ as *logical signals*; in particular, *Boolean signals* are logical signals over $\mathbb{B}$. We adopt the following naming convention for logical signals: λ indicates Boolean signals, Λ indicates $\mathbb{U}_1$-signals, and $\tilde{\Lambda}$ indicates $\mathbb{U}_2$-signals.

A Boolean signal λ is *unitary* if there is one contiguous interval $I^+_\lambda \subseteq \mathbb{R}_{\geq 0}$ such that $\lambda(t) = \top$ for all $t \in I^+_\lambda$ and $\lambda(t) = \bot$ everywhere else [26]. Every Boolean signal can be represented as a disjunction of unitary signals, as shown in Fig. 1 [26]. In this work, we require this *unitary decomposition* to be minimal, i.e., the number of involved unitary signals must be minimal.

2.4 Reachability Analysis of Hybrid Systems

The literature provides numerous methods for describing hybrid systems. Our verification method is independent of the chosen description method as long as the system model is amenable to set-based reachability analysis (see [4] for an overview). Given the mixed continuous and discrete *state space* $\mathcal{X}$ of the hybrid system $\mathcal{H}$, an *execution* of $\mathcal{H}$ is a signal $\xi : \mathbb{R}_{\geq 0} \to \mathcal{X}$.

We are interested in the *reachable set* of the system $\mathcal{H}$, i.e., the set of all states that are part of at least one execution of $\mathcal{H}$. Formally, the reachable set of $\mathcal{H}$ is a signal $\mathcal{R} : \mathbb{R}_{\geq 0} \to 2^\mathcal{X}$ given by

$$\mathcal{R}(t) := \{\xi(t) \mid \xi \text{ is an execution of } \mathcal{H}\}.$$

Since determining the exact reachable set is often computationally infeasible, tools like CORA [2], JuliaReach [11], and SpaceEx [18] typically return a discrete-time overapproximation when performing reachability analysis. To this end, they represent $\mathcal{R}$ as a sequence of sets so that the set $\mathcal{R}_I$ for the time interval I subsumes $\bigcup_{t \in I} \mathcal{R}(t)$. Our verification algorithm assumes that this sequence is incrementally computed for consecutive time intervals, as is the case with the tools

mentioned. To handle Taylor model representations (e.g., as used by Flow* [14]), a preprocessing step would be required to obtain a sequence of sets.

2.5 Signal Temporal Logic with Boolean Semantics

Suppose $\mathcal{AP}$ is a fixed set of atomic predicates, where each predicate is a function $a : \mathcal{X} \to \mathbb{B}$. An STL *formula* φ over $\mathcal{AP}$ is constructed according to the grammar

$$\varphi ::= \mathit{true} \mid a \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \,\mathbf{U}_I\, \varphi_2,$$

where $a \in \mathcal{AP}$ and I is an interval over $\mathbb{R}_{\geq 0}$ with rational endpoints [26]. We use the common abbreviations $\varphi_1 \vee \varphi_2 := \neg(\neg\varphi_1 \wedge \neg\varphi_2)$, $\mathbf{F}_I\, \varphi := \mathit{true}\, \mathbf{U}_I\, \varphi$ (finally), and $\mathbf{G}_I\, \varphi := \neg\, \mathbf{F}_I\, \neg\varphi$ (globally). Note that we define *true* as basic syntax rather than introducing it as an abbreviation for $a \vee \neg a$, because the law of excluded middle does not transfer well to the four-valued semantics we define later.

In Boolean semantics, an STL formula φ is interpreted over an execution $\xi : \mathbb{R}_{\geq 0} \to \mathcal{X}$ to obtain a yes-or-no answer whether ξ satisfies φ [26, 27]. We define the Boolean *satisfaction signal* $[\![\varphi]\!]_\xi : \mathbb{R}_{\geq 0} \to \mathbb{B}$ of φ over ξ inductively as

$$\begin{aligned}
[\![\mathit{true}]\!]_\xi(t) &:= \top, \\
[\![a]\!]_\xi(t) &:= a(\xi(t)), \\
[\![\neg\varphi]\!]_\xi(t) &:= \neg[\![\varphi]\!]_\xi(t), \\
[\![\varphi_1 \wedge \varphi_2]\!]_\xi(t) &:= [\![\varphi_1]\!]_\xi(t) \wedge [\![\varphi_2]\!]_\xi(t), \\
[\![\varphi_1 \,\mathbf{U}_I\, \varphi_2]\!]_\xi(t) &:= \begin{cases} \top & \text{if } \exists t' \in t \oplus I : [\![\varphi_2]\!]_\xi(t') = \top \\ & \qquad \text{and } \forall t'' \in (t, t') : [\![\varphi_1]\!]_\xi(t'') = \top \\ \bot & \text{otherwise} \end{cases},
\end{aligned}$$

where $a \in \mathcal{AP}$. The value of the satisfaction signal at time t indicates whether φ holds at t. Thus, an execution ξ *satisfies* φ, denoted by $\xi \models \varphi$, if and only if $[\![\varphi]\!]_\xi(0) = \top$. A hybrid system $\mathcal{H}$ *satisfies* φ, written as $\mathcal{H} \models \varphi$, if $\xi \models \varphi$ for all executions ξ of $\mathcal{H}$. Note that we use the strict until semantics from [27], which does not require φ_1 to hold at t or t', unlike the version in [26,35]. This semantics is more expressive, as we can recover the until from [26,35] as $\varphi_1 \wedge \varphi_1\, \mathbf{U}_I (\varphi_1 \wedge \varphi_2)$.

To exclude Zeno behavior, we limit ourselves to executions ξ such that, for all $a \in \mathcal{AP}$, the Boolean signal given by point-wise application of a to ξ is of *finite variability*. That is, it changes its value only a finite number of times in any finite time interval [6, Sec. 2.3.5]. This is a common assumption in related work [17,26,27,35], albeit not always under this name; we refer the reader to [26, Sec. 4] and [27, Sec. 4] for a discussion.

2.6 Problem Statement

Given an STL formula φ and a hybrid system $\mathcal{H}$, we want to determine whether $\mathcal{H} \models \varphi$ based on the reachable set of $\mathcal{H}$. As reachability analysis is often incremental, we need to interpret φ over a reachable set that is only known for

some time intervals to form a preliminary verification verdict. Formally, this means we want to define and compute a $\mathbb{U}_2$-satisfaction signal $[\![\varphi]\!]_{\tilde{\mathcal{R}}} : \mathbb{R}_{\geq 0} \to \mathbb{U}_2$ over a partial reachable set $\tilde{\mathcal{R}} : \mathcal{T} \to 2^{\mathcal{X}}$, where $\mathcal{T} \subseteq \mathbb{R}_{\geq 0}$. If our preliminary verdict is inconclusive due to partial knowledge of the reachable set, we aim to efficiently update our verdict as soon as more information becomes available. If the final verdict turns out to be $\dashv_1$, we need to refine our overapproximation of the reachable set to verify or falsify the specification.

This paper focuses on the four-valued semantics and the efficient update of the preliminary verdict. We only briefly discuss refinements of the overapproximation, as this often amounts to tuning the parameters of the reachability analysis. Moreover, it is a common step in verification approaches based on reachable sets [22, 32, 35]. An automatic refinement technique for the reachset temporal logic approach [32] is presented in [24].

3 Basic Idea and Solution Concept

As a motivating example, consider the intentionally simple dynamical system given by the differential equation $\dot{x} = u$, where the input u lies somewhere in $[0.9, 1.1]$ and initially $x \in [-0.5, 0.5]$. Suppose we want to verify that x eventually becomes larger than 1 within the next five seconds, which we formalize in STL as $\varphi := \mathbf{F}_{[0,5]}\, x > 1$. To this end, we exploit that the reachable set $\mathcal{R}$ of our system encloses all executions of the system. Thus, if we can prove that there exists a $t \in [0, 5]$ such that $x > 1$ for all states $x \in \mathcal{R}(t)$, we have shown φ for all executions of our system.

Recall that the reachable set is usually computed incrementally for consecutive time intervals. We are thus dealing with a partial reachable set $\tilde{\mathcal{R}}$, which is only determined for a subset of the time domain. For example, a reachability algorithm might first compute the reachable set for up to 1 s (top left in Fig. 2) and then continue to determine the reachable set in the next 0.8 s (top right).

If we are able to interpret our specification φ over such partial reachable sets, we can re-evaluate its satisfaction with every newly determined time interval and terminate the algorithm once we obtain a conclusive result. To this end, we derive a set-based version $\hat{a} : 2^{\mathcal{X}} \to \mathbb{U}_1$ of every atomic predicate $a \in \mathcal{AP}$ so that

$$\hat{a}(\mathcal{X}') := \begin{cases} \top & \text{if } \mathcal{X}' \neq \emptyset \text{ and } \mathcal{X}' \subseteq [\![a]\!] \\ \bot & \text{if } \mathcal{X}' \neq \emptyset \text{ and } \mathcal{X}' \cap [\![a]\!] = \emptyset \\ \dashv_1 & \text{otherwise} \end{cases}$$

for $\mathcal{X}' \subseteq \mathcal{X}$, where $[\![a]\!] := \{x \in \mathcal{X} \mid a(x) = \top\}$ denotes the set of states satisfying a. This enables us to construct a $\mathbb{U}_2$-satisfaction signal for atomic predicates over $\tilde{\mathcal{R}}$ by assigning $\dashv_2$ at times where the reachable set has not yet been determined. As shown in the second row of Fig. 2, this signal becomes $\dashv_1$ as soon as the reachable set starts intersecting with our atomic predicate and changes to $\top$ once it lies fully inside. Thus, $\dashv_1$ means that we do not know whether a predicate is

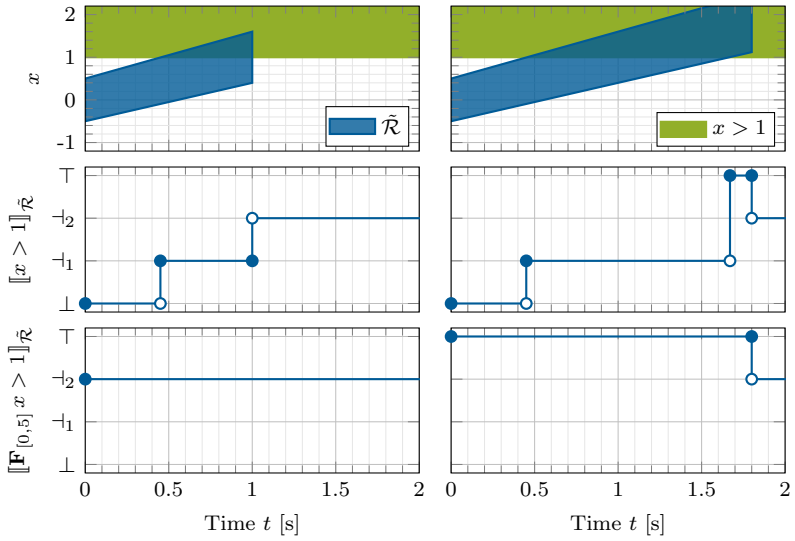


Fig. 2. Reachable set $\tilde{\mathcal{R}}$ and $\mathbb{U}_2$-satisfaction signals of $x > 1$ and $\mathbf{F}_{[0,5]}\, x > 1$ for the simple example system after reachability analysis for up to 1 s (left) and 1.8 s (right)

true or false since the reachable set contains satisfying and violating states; $\dashv_2$ means that we do not know as we have not yet computed the set for this time.

Now that we have satisfaction signals for the atomic predicates, it remains to combine them in order to obtain satisfaction signals for compound formulas. For this, we develop operators that preserve the intended meaning of our two uncertain values $\dashv_1$ and $\dashv_2$ in Sec. 4. The third row of Fig. 2 shows the resulting satisfaction signal $[\![\varphi]\!]_{\tilde{\mathcal{R}}}$ for our example specification: After reachability analysis for up to 1 s, the verification verdict is inconclusive, since $[\![\varphi]\!]_{\tilde{\mathcal{R}}}(0) = \dashv_2$. Once we update the satisfaction signal to incorporate information about the next 0.8 s, we can verify that φ holds and terminate early. In Sec. 5, we use ideas of the incremental marking procedure [27] to perform this update efficiently.

4 Four-Valued Signal Temporal Logic

To compute the $\mathbb{U}_2$-satisfaction signal for an STL formula φ with respect to a partial reachable set, we proceed by structural recursion on φ. The base case for *true* is clear, and we treat atomic predicates as described in Sec. 3. For compound formulas, i.e., negation, conjunction, and until, we combine the recursively computed satisfaction signals of their subformulas using suitable operators on signals. Instead of defining these operators directly on $\mathbb{U}_2$-signals, we under- and over-approximate a $\mathbb{U}_2$-signal using $\mathbb{U}_1$-signals. Similarly, we represent $\mathbb{U}_1$-signals by Boolean signals. Utilizing this representation, we can use the negation, conjunction, and until operators defined for Boolean signals to combine $\mathbb{U}_2$-satisfaction

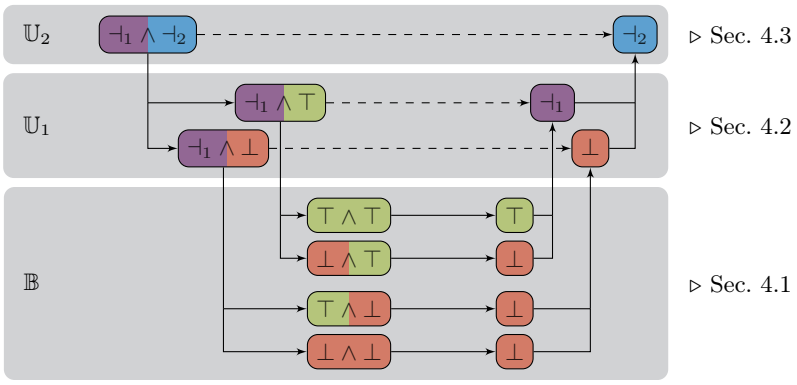


Fig. 3. Evaluating $\dashv_1 \wedge \dashv_2$ in $\mathbb{U}_2$-semantics: On the way down, we under- and overapproximate the input values with respect to the truth order by replacing the uncertain value of the current layer with $\bot$ and $\top$. In the Boolean layer, we evaluate the conjunction as usual. To move back up, we check whether the results for both approximations agree, and assign the appropriate uncertain value if this is not the case.

signals. Fig. 3 shows this concept simplified to the propositional case, in which we are dealing with truth values instead of logical signals.

4.1 Computing Boolean Satisfaction Signals

To define the operators for combining Boolean satisfaction signals, we closely follow the procedure from [27]. For negation and conjunction, we lift $\neg$ and $\wedge$ from Boolean values to Boolean signals by point-wise application [27, Sec. 3.1.1]. Unlike [27], we handle until directly instead of expressing it as a combination of untimed until and timed finally, thus avoiding the rather involved specification rewriting of [27, Lem. 1]. To this end, we adapt the method from [26, Sec. 3] for arbitrary intervals and strict semantics.

We first define an until operator that works only for unitary Boolean signals and generalize it later using the unitary decomposition. Unitary signals have the helpful property that $\lambda(t) = \top = \lambda(t')$ for times $t \leq t'$ implies $\lambda(t'') = \top$ for all $t'' \in [t, t']$. Given unitary signals λ_1, λ_2 and an interval I over $\mathbb{R}_{\geq 0}$, we define the unitary until $\overline{\mathbf{U}}_I$ so that $(\lambda_1 \,\overline{\mathbf{U}}_I\, \lambda_2)(t) = \top$ if and only if $t \in I_1 \cup I_2$, where

$$I_1 := \left[\left(I^+_{\lambda_2} \cap \mathrm{cl}_{\mathrm{r}}(I^+_{\lambda_1})\right) \oplus (-(I \setminus \{0\}))\right] \cap \mathrm{cl}_{\mathrm{l}}(I^+_{\lambda_1}), \quad I_2 := \begin{cases} I^+_{\lambda_2} & \text{if } 0 \in I \\ \emptyset & \text{otherwise} \end{cases}. \tag{1}$$

Here, $I \setminus \{0\}$ is always an interval, since $I \subseteq \mathbb{R}_{\geq 0} = [0, \infty)$. We prove that $\overline{\mathbf{U}}_I$ implements the until semantics for unitary signals.

Lemma 1. *Suppose* λ_1 *and* λ_2 *are* unitary *Boolean signals, and* I *is an interval over* $\mathbb{R}_{\geq 0}$*. For all* $t \in \mathbb{R}_{\geq 0}$*, we have*

$$(\lambda_1 \,\overline{\mathbf{U}}_I\, \lambda_2)(t) = \begin{cases} \top & \text{if } \exists t' \in t \oplus I : \lambda_2(t') = \top \text{ and } \forall t'' \in (t,t') : \lambda_1(t'') = \top \\ \bot & \text{otherwise} \end{cases}.$$

Proof. Let I_1 and I_2 be given as in (1) so that $(\lambda_1 \,\overline{\mathbf{U}}_I\, \lambda_2)(t) = \top$ if and only if $t \in I_1 \cup I_2$. For an arbitrary $t \in \mathbb{R}_{\geq 0}$, we first prove that I_1 treats the case $t' > t$:

$$\begin{aligned} t \in I_1 &\iff t \in \left[\left(I^+_{\lambda_2} \cap \mathrm{cl}_{\mathbf{r}}(I^+_{\lambda_1})\right) \oplus (-(I \setminus \{0\}))\right] \cap \mathrm{cl}_{\mathbf{l}}(I^+_{\lambda_1}) \\ &\iff \exists t' \in t \oplus (I \setminus \{0\}) : t' \in I^+_{\lambda_2} \text{ and } t' \in \mathrm{cl}_{\mathbf{r}}(I^+_{\lambda_1}) \text{ and } t \in \mathrm{cl}_{\mathbf{l}}(I^+_{\lambda_1}) \\ &\iff \exists t' \in t \oplus (I \setminus \{0\}) : \lambda_2(t') = \top \text{ and } \forall t'' \in (t,t') : \lambda_1(t'') = \top. \end{aligned}$$

For the second step, observe that $\mathcal{A} \oplus (-\mathcal{B}) = \{x \mid \exists a \in x \oplus \mathcal{B} : a \in \mathcal{A}\}$. The last equivalence uses that λ_1 is unitary: If λ_1 is $\top$ both immediately after t, i.e., $t \in \mathrm{cl}_{\mathbf{l}}(I^+_{\lambda_1})$, and immediately before t', i.e., $t' \in \mathrm{cl}_{\mathbf{r}}(I^+_{\lambda_1})$, the signal must also be $\top$ throughout (t,t'), since $I^+_{\lambda_1}$ is contiguous. For the converse, note that $\emptyset \subsetneq (t,t') \subseteq I^+_{\lambda_1}$ implies $t \in \mathrm{cl}_{\mathbf{l}}(I^+_{\lambda_1})$ and $t' \in \mathrm{cl}_{\mathbf{r}}(I^+_{\lambda_1})$. If $0 \notin I$, we are done, as $I = I \setminus \{0\}$ and $I_2 = \emptyset$. Otherwise, I_2 handles the case $t' = t$, where $t \oplus \{0\} = \{t\}$ and the universal quantifier is vacuously satisfied:

$$\begin{aligned} t \in I_2 &\iff t \in I^+_{\lambda_2} \\ &\iff \lambda_2(t) = \top \\ &\iff \exists t' \in t \oplus \{0\} : \lambda_2(t') = \top \text{ and } \forall t'' \in (t,t') : \lambda_1(t'') = \top. \end{aligned}$$ □

Using that all Boolean signals admit a unitary decomposition, we generalize $\overline{\mathbf{U}}_I$ to Boolean signals λ_1 and λ_2 that are not necessarily unitary. We define

$$(\lambda_1 \,\mathbf{U}_I\, \lambda_2)(t) := \bigvee_{1 \leq i \leq n_1} \bigvee_{1 \leq j \leq n_2} (\lambda_{1,i} \,\overline{\mathbf{U}}_I\, \lambda_{2,j})(t),$$

where $t \in \mathbb{R}_{\geq 0}$ and $\{\lambda_{k,1}, \ldots, \lambda_{k,n_k}\}$ is the unitary decomposition of λ_k for $k \in \{1,2\}$. We prove that this is a general implementation of the until semantics.

Lemma 2. *Let* λ_1 *and* λ_2 *be Boolean signals and* I *be an interval over* $\mathbb{R}_{\geq 0}$*. For all* $t \in \mathbb{R}_{\geq 0}$*, we have*

$$(\lambda_1 \,\mathbf{U}_I\, \lambda_2)(t) = \begin{cases} \top & \text{if } \exists t' \in t \oplus I : \lambda_2(t') = \top \text{ and } \forall t'' \in (t,t') : \lambda_1(t'') = \top \\ \bot & \text{otherwise} \end{cases}.$$

In particular, we obtain $[\![\varphi_1 \,\mathbf{U}_I\, \varphi_2]\!]_\xi = [\![\varphi_1]\!]_\xi \,\mathbf{U}_I [\![\varphi_2]\!]_\xi$*.*

Proof. Observe that the second statement is a specialization of the first by the Boolean semantics of STL. Let $t \in \mathbb{R}_{\geq 0}$ be arbitrary. If $(\lambda_1 \,\mathbf{U}_I\, \lambda_2)(t) = \top$, there must be i and j so that $(\lambda_{1,i} \,\overline{\mathbf{U}}_I\, \lambda_{2,j})(t) = \top$, and we can immediately conclude

with Lem. 1. Conversely, let $t' \in t \oplus I$ so that $\lambda_2(t') = \top$ and $\lambda_1(t'') = \top$ for all $t'' \in (t, t')$. Since $\lambda_2(t') = \top$, there exists j such that $\lambda_{2,j}(t') = \top$. As the unitary decomposition of λ_1 is minimal, the union of two or more of the $I^+_{\lambda_{1,i}}$ cannot yield a contiguous interval (otherwise, we could merge them for a smaller decomposition). Hence, there exists i such that $(t, t') \subseteq I^+_{\lambda_{1,i}}$, or, in other words, $\lambda_{1,i}(t'') = \top$ for all $t'' \in (t, t')$. Applying Lem. 1 again concludes the proof. □

4.2 Computing Three-Valued Satisfaction Signals

To compute the $\mathbb{U}_1$-satisfaction signal of an STL formula over a reachable set, we represent $\mathbb{U}_1$-signals using Boolean signals and then reuse the techniques from Sec. 4.1. Recall from Sec. 2.2 that $\dashv_1$ means that we do not know whether a statement is true or false. Thus, every $\mathbb{U}_1$-signal Λ induces a set of Boolean signals, called *refinements* of Λ, in which the occurrences of $\dashv_1$ are replaced with $\top$ or $\bot$. Formally, a Boolean signal λ *refines* Λ, denoted as $\lambda \prec \Lambda$, if $\Lambda(t) \neq \dashv_1$ implies $\lambda(t) = \Lambda(t)$ for all $t \in \mathbb{R}_{\geq 0}$. Since the set of refinements is unique for each $\mathbb{U}_1$-signal Λ, we use it to represent Λ. We argue that the two special refinements

$$\lfloor \Lambda \rfloor(t) := \begin{cases} \bot & \text{if } \Lambda(t) = \dashv_1 \\ \Lambda(t) & \text{otherwise} \end{cases} \quad \text{and} \quad \lceil \Lambda \rceil(t) := \begin{cases} \top & \text{if } \Lambda(t) = \dashv_1 \\ \Lambda(t) & \text{otherwise} \end{cases}$$

adequately characterize this set. Lifting the truth order $\sqsubseteq_{\mathtt{t}}$ to a partial order on logical signals by point-wise application, we find that the Boolean signal λ refines Λ if and only if $\lfloor \Lambda \rfloor \sqsubseteq_{\mathtt{t}} \lambda \sqsubseteq_{\mathtt{t}} \lceil \Lambda \rceil$. Hence, $\lfloor \Lambda \rfloor$ *underapproximates* the refinements of Λ, while $\lceil \Lambda \rceil$ *overapproximates* them. We can recover $\Lambda(t)$ as $\lfloor \Lambda \rfloor(t) \sqcup_1 \lceil \Lambda \rceil(t)$, where $v \sqcup_1 v'$ with $v, v' \in \mathbb{B}$ yields v if and only if $v = v'$ and $\dashv_1$ otherwise.

The operator $\mathbf{U}_I$ on Boolean signals is *monotone*, i.e., we have $\lambda_1 \,\mathbf{U}_I\, \lambda_2 \sqsubseteq_{\mathtt{t}} \lambda_1' \,\mathbf{U}_I\, \lambda_2'$ given that $\lambda_i \sqsubseteq_{\mathtt{t}} \lambda_i'$ for $i \in \{1, 2\}$. Intuitively, this means that if we set the inputs to $\top$ at more time points, the output signal will also be $\top$ more often. Thus, $\lfloor \Lambda_1 \rfloor \,\mathbf{U}_I\, \lfloor \Lambda_2 \rfloor$ is a faithful underapproximation of $\lambda_1 \,\mathbf{U}_I\, \lambda_2$, given that $\lambda_i \prec \Lambda_i$ for $i \in \{1, 2\}$. Similarly, $\lceil \Lambda_1 \rceil \,\mathbf{U}_I \lceil \Lambda_2 \rceil$ is an overapproximation. Fig. 4 visualizes this for a derived finally operator $\mathbf{F}_I\, \lambda := \lambda_\top \,\mathbf{U}_I\, \lambda$, where $\lambda_\top$ is $\top$ everywhere. To show monotonicity of $\mathbf{U}_I$, we apply Lem. 2 and use that $\lambda_i(t) = \top$ implies $\lambda_i'(t) = \top$. The operator $\wedge$ is also monotone. In contrast, $\neg$ is *antitone*, i.e., $\lambda \sqsubseteq_{\mathtt{t}} \lambda'$ implies $\neg\lambda' \sqsubseteq_{\mathtt{t}} \neg\lambda$. So, $\neg\lceil \Lambda \rceil$ is an underapproximation, while $\neg\lfloor \Lambda \rfloor$ is an overapproximation. We define the operators $\neg$, $\wedge$, and $\mathbf{U}_I$ on $\mathbb{U}_1$-signals such that they recover a $\mathbb{U}_1$-signal from these over- and underapproximations:

$$\begin{aligned} (\neg\Lambda)(t) &:= (\neg\lfloor \Lambda \rfloor)(t) \sqcup_1 (\neg\lceil \Lambda \rceil)(t), \\ (\Lambda_1 \wedge \Lambda_2)(t) &:= (\lfloor \Lambda_1 \rfloor \wedge \lfloor \Lambda_2 \rfloor)(t) \sqcup_1 (\lceil \Lambda_1 \rceil \wedge \lceil \Lambda_2 \rceil)(t), \\ (\Lambda_1 \,\mathbf{U}_I\, \Lambda_2)(t) &:= (\lfloor \Lambda_1 \rfloor \,\mathbf{U}_I \lfloor \Lambda_2 \rfloor)(t) \sqcup_1 (\lceil \Lambda_1 \rceil \,\mathbf{U}_I \lceil \Lambda_2 \rceil)(t). \end{aligned}$$

Finally, we define the $\mathbb{U}_1$-satisfaction signal $[\![\varphi]\!]_{\mathcal{R}} : \mathbb{R}_{\geq 0} \to \mathbb{U}_1$ of an STL formula φ with respect to a reachable set $\mathcal{R} : \mathbb{R}_{\geq 0} \to 2^{\mathcal{X}}$ using our operators. For all $t \in \mathbb{R}_{\geq 0}$, we define

$$[\![\mathit{true}]\!]_{\mathcal{R}}(t) := \top \qquad \text{and} \qquad [\![a]\!]_{\mathcal{R}}(t) := \hat{a}(\mathcal{R}(t)),$$

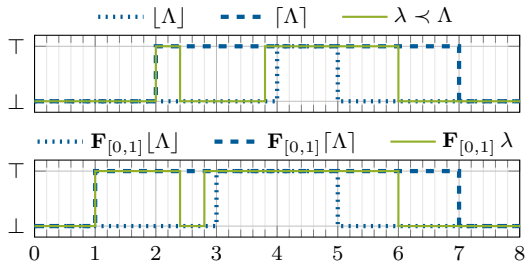


Fig. 4. Top: All refinements λ of a $\mathbb{U}_1$-signal Λ lie between the underapproximation $\lfloor\Lambda\rfloor$ and the overapproximation $\lceil\Lambda\rceil$. Bottom: After applying the monotone finally operator to all three Boolean signals, the refinement is still between the approximations. We omit the markers at the jumps, as they are irrelevant to the point of this example.

where $a \in \mathcal{AP}$ and $\hat{a}$ is defined as in Sec. 3. Moreover, we define

$$\begin{aligned}
[\![\neg\varphi]\!]_{\mathcal{R}} &:= \neg[\![\varphi]\!]_{\mathcal{R}},\\
[\![\varphi_1 \wedge \varphi_2]\!]_{\mathcal{R}} &:= [\![\varphi_1]\!]_{\mathcal{R}} \wedge [\![\varphi_2]\!]_{\mathcal{R}},\\
[\![\varphi_1 \,\mathbf{U}_I\, \varphi_2]\!]_{\mathcal{R}} &:= [\![\varphi_1]\!]_{\mathcal{R}} \,\mathbf{U}_I [\![\varphi_2]\!]_{\mathcal{R}}.
\end{aligned}$$

To relate this to the Boolean semantics, we show that $[\![\varphi]\!]_\xi$ refines $[\![\varphi]\!]_{\mathcal{R}}$, if $\mathcal{R}$ covers the execution ξ. We say a reachable set $\mathcal{R}$ *covers* an execution ξ, denoted as $\xi \prec \mathcal{R}$, if $\xi(t) \in \mathcal{R}(t)$ for all $t \in \mathbb{R}_{\geq 0}$.

Theorem 1. *Suppose $\mathcal{R}$ is a reachable set and φ an STL formula. For every execution ξ covered by $\mathcal{R}$, we have $[\![\varphi]\!]_\xi \prec [\![\varphi]\!]_{\mathcal{R}}$. In other words, if $[\![\varphi]\!]_{\mathcal{R}}(t) = v$, we have $[\![\varphi]\!]_\xi(t) = v$ for all $\xi \prec \mathcal{R}$, $v \in \mathbb{B}$, and $t \in \mathbb{R}_{\geq 0}$.*

Proof. We proceed by structural induction on φ. Let $t \in \mathbb{R}_{\geq 0}$ be arbitrary.

Base Cases: The case for $\varphi = \mathit{true}$ is straightforward, as *true* always evaluates to $\top$ in Boolean and $\mathbb{U}_1$-semantics. For an atomic predicate $a \in \mathcal{AP}$, we find

$$\hat{a}(\mathcal{R}(t)) = \top \iff \mathcal{R}(t) \neq \emptyset \text{ and } \mathcal{R}(t) \subseteq [\![a]\!] \implies \forall \xi \prec \mathcal{R} : a(\xi(t)) = \top,$$

and thus $[\![a]\!]_{\mathcal{R}}(t) = \top \implies \forall \xi \prec \mathcal{R} : [\![a]\!]_\xi(t) = \top$. The argument for $\bot$ is similar.

Direct Semantics: Turning to the inductive cases, we notice that our operators on $\mathbb{U}_1$-signals depend on their Boolean counterparts. This makes them easy to implement, but difficult to handle in proofs. Therefore, we first show that they

adhere to the following direct semantics that work without this dependency:[1]

$$(\neg \Lambda)(t) = \begin{cases} \top & \text{if } \Lambda(t) = \bot \\ \bot & \text{if } \Lambda(t) = \top \\ \dashv_1 & \text{otherwise} \end{cases},$$

$$(\Lambda_1 \wedge \Lambda_2)(t) = \begin{cases} \top & \text{if } \Lambda_1(t) = \top \text{ and } \Lambda_2(t) = \top \\ \bot & \text{if } \Lambda_1(t) = \bot \text{ or } \Lambda_2(t) = \bot \\ \dashv_1 & \text{otherwise} \end{cases}, \tag{2}$$

$$(\Lambda_1 \,\mathbf{U}_I\, \Lambda_2)(t) = \begin{cases} \top & \text{if } \exists t' \in t \oplus I : \Lambda_2(t') = \top \text{ and } \forall t'' \in (t, t') : \Lambda_1(t'') = \top \\ \bot & \text{if } \forall t' \in t \oplus I : \Lambda_2(t') = \bot \text{ or } \exists t'' \in (t, t') : \Lambda_1(t'') = \bot \\ \dashv_1 & \text{otherwise} \end{cases}.$$

Proof of the Direct Semantics: First, consider the case where $(\Lambda_1 \,\mathbf{U}_I\, \Lambda_2)(t)$ is $\top$. Recalling that $v \sqcup_1 v'$ only yields $\top$ if $v = \top = v'$, we derive

$$\begin{aligned} (\Lambda_1 \,\mathbf{U}_I\, \Lambda_2)(t) = \top &\iff (\lfloor \Lambda_1 \rfloor \,\mathbf{U}_I \lfloor \Lambda_2 \rfloor)(t) = \top = (\lceil \Lambda_1 \rceil \,\mathbf{U}_I \lceil \Lambda_2 \rceil)(t) \\ &\iff \forall \lambda_1 \prec \Lambda_1 : \forall \lambda_2 \prec \Lambda_2 : (\lambda_1 \,\mathbf{U}_I\, \lambda_2)(t) = \top. \end{aligned}$$

The second equivalence is due to the monotonicity of $\mathbf{U}_I$ over Boolean signals: If $\lambda_i \prec \Lambda_i$, we know that $\lfloor \Lambda_i \rfloor \sqsubseteq_\mathtt{t} \lambda_i \sqsubseteq_\mathtt{t} \lceil \Lambda_i \rceil$ for $i \in \{1, 2\}$. Thus, we also have $\lfloor \Lambda_1 \rfloor \,\mathbf{U}_I \lfloor \Lambda_2 \rfloor \sqsubseteq_\mathtt{t} \lambda_1 \,\mathbf{U}_I\, \lambda_2 \sqsubseteq_\mathtt{t} \lceil \Lambda_1 \rceil \,\mathbf{U}_I \lceil \Lambda_2 \rceil$. Hence, $(\lambda_1 \,\mathbf{U}_I\, \lambda_2)(t)$ must be $\top$ due to the antisymmetry of $\sqsubseteq_\mathtt{t}$. The converse is clear, as $\lfloor \Lambda_i \rfloor$ and $\lceil \Lambda_i \rceil$ are particular refinements of Λ_i. We continue our derivation by applying Lem. 2 and find

$$\begin{aligned} \ldots &\iff \forall \lambda_1 \prec \Lambda_1 : \forall \lambda_2 \prec \Lambda_2 : \\ &\qquad \exists t' \in t \oplus I : \lambda_2(t') = \top \text{ and } \forall t'' \in (t, t') : \lambda_1(t'') = \top \\ &\iff \exists t' \in t \oplus I : \Lambda_2(t') = \top \text{ and } \forall t'' \in (t, t') : \Lambda_1(t'') = \top. \end{aligned}$$

To explain the forward direction of the last equivalence, we consider the refinements $\lfloor \Lambda_1 \rfloor$ and $\lfloor \Lambda_2 \rfloor$. Instantiating the universal quantifiers, we find that

$$\exists t' \in t \oplus I : \lfloor \Lambda_2 \rfloor(t') = \top \text{ and } \forall t'' \in (t, t') : \lfloor \Lambda_1 \rfloor(t'') = \top.$$

Since $\lfloor \Lambda_i \rfloor(t) = \top$ if and only if $\Lambda_i(t) = \top$, this establishes the forward direction. For $(\Lambda_1 \,\mathbf{U}_I\, \Lambda_2)(t) = \bot$, we argue analogously, and then the case for $\dashv_1$ follows by elimination. The reasoning for the remaining operators is similar.

[1] These semantics arise naturally from the Boolean semantics in Sec. 2.5 by making the Boolean "otherwise" case explicit and adding a new "otherwise" to handle $\dashv_1$.

Inductive Cases: We are now equipped to prove the inductive cases for our main statement. Using the direct semantics (2), we exemplarily show the case for until:

$$\begin{aligned}
[\![\varphi_1 \,\mathbf{U}_I\, \varphi_2]\!]_{\mathcal{R}}(t) = \top \iff\ & \exists t' \in t \oplus I : [\![\varphi_2]\!]_{\mathcal{R}}(t') = \top \\
& \quad \text{and } \forall t'' \in (t, t') : [\![\varphi_1]\!]_{\mathcal{R}}(t'') = \top \\
\implies\ & \exists t' \in t \oplus I : (\forall \xi \prec \mathcal{R} : [\![\varphi_2]\!]_{\xi}(t') = \top) && \text{(IH)} \\
& \quad \text{and } \forall t'' \in (t, t') : \forall \xi \prec \mathcal{R} : [\![\varphi_1]\!]_{\xi}(t'') = \top \\
\implies\ & \forall \xi \prec \mathcal{R} : \exists t' \in t \oplus I : [\![\varphi_2]\!]_{\xi}(t') = \top && (*) \\
& \quad \text{and } \forall t'' \in (t, t') : [\![\varphi_1]\!]_{\xi}(t'') = \top \\
\iff\ & \forall \xi \prec \mathcal{R} : [\![\varphi_1 \,\mathbf{U}_I\, \varphi_2]\!]_{\xi}(t) = \top.
\end{aligned}$$

To derive $[\![\varphi_1 \,\mathbf{U}_I\, \varphi_2]\!]_{\mathcal{R}}(t) = \bot \implies \forall \xi \prec \mathcal{R} : [\![\varphi_1 \,\mathbf{U}_I\, \varphi_2]\!]_{\xi}(t) = \bot$, we argue similarly. Note that the step marked with $(*)$ is not an equivalence since we need to swap an existential and a universal quantifier. Intuitively, this means that the Boolean semantics allow us to choose the time when φ_2 becomes true for each refinement individually, while we have to choose the same time for all refinements in the $\mathbb{U}_1$-semantics. A similar step is necessary in the $\bot$ case of conjunction, where we have to choose which subformula is false. □

4.3 Computing Four-Valued Satisfaction Signals

In the previous section, we used two Boolean signals to over- and underapproximate the refinements of a $\mathbb{U}_1$-signal. This enabled us to reuse the operators defined on Boolean signals for computing a $\mathbb{U}_1$-satisfaction signal of an STL formula with respect to a reachable set. Following the same pattern, we can over- and underapproximate the refinements of a $\mathbb{U}_2$-signal by two $\mathbb{U}_1$-signals to compute $\mathbb{U}_2$-satisfaction signals over partial reachable sets. Hence, many concepts, definitions, and proofs are analogous to Sec. 4.2, so we only sketch them here.

Given a $\mathbb{U}_2$-signal $\tilde{\Lambda}$, the $\mathbb{U}_1$-signal Λ *refines* $\tilde{\Lambda}$, denoted by $\Lambda \prec \tilde{\Lambda}$, if $\tilde{\Lambda}(t) \neq \dashv_2$ implies $\Lambda(t) = \tilde{\Lambda}(t)$ for all $t \in \mathbb{R}_{\geq 0}$. We define the *underapproximation* $\lfloor \tilde{\Lambda} \rfloor$ and the *overapproximation* $\lceil \tilde{\Lambda} \rceil$ analogously to Sec. 4.2, i.e., by replacing $\dashv_2$ with $\bot$ and $\top$, respectively. Again, we have $\lfloor \tilde{\Lambda} \rfloor \sqsubseteq_{\mathtt{t}} \Lambda \sqsubseteq_{\mathtt{t}} \lceil \tilde{\Lambda} \rceil$ if and only if $\Lambda \prec \tilde{\Lambda}$. Moreover, we can reconstruct $\tilde{\Lambda}(t)$ as $\lfloor \tilde{\Lambda} \rfloor(t) \sqcup_2 \lceil \tilde{\Lambda} \rceil(t)$. Here, $v \sqcup_2 v'$ with $v, v' \in \mathbb{U}_1$ is v if and only if $v = v'$ and $\dashv_2$ otherwise.

To define the operators for negation, conjunction, and until on $\mathbb{U}_2$-signals, we first show that the $\mathbb{U}_1$-operators are monotone or antitone.

Lemma 3. *The operators $\wedge$ and $\mathbf{U}_I$ on $\mathbb{U}_1$-signals are monotone; $\neg$ is antitone.*

Proof. Using the direct semantics (2) of the operators shown in the proof of Thm. 1 and case distinction, the proof is straightforward. We exemplarily consider the case $(\Lambda_1 \,\mathbf{U}_I\, \Lambda_2)(t) = \dashv_1$, where we need to show $\dashv_1 \sqsubseteq_{\mathtt{t}} (\Lambda_1' \,\mathbf{U}_I\, \Lambda_2')(t)$ given $\Lambda_i \sqsubseteq_{\mathtt{t}} \Lambda_i'$ for $i \in \{1, 2\}$. From the direct semantics, we know

$$\exists t' \in t \oplus I : \Lambda_2(t') \neq \bot \text{ and } \forall t'' \in (t, t') : \Lambda_1(t'') \neq \bot.$$

Since Λ'_i can only be $\bot$ where Λ_i is also $\bot$, the same statement holds for Λ'_1 and Λ'_2. Thus, $(\Lambda'_1 \,\mathbf{U}_I\, \Lambda'_2)(t)$ cannot be $\bot$. Consequently, it must be either $\dashv_1$ or $\top$, and we know that $\dashv_1 \sqsubseteq_\mathtt{t} \top$. □

Due to Lem. 3, it is justified to define the operators $\neg$, $\wedge$, and $\mathbf{U}_I$ on $\mathbb{U}_2$-signals like in Sec. 4.2 using $\sqcup_2$ instead of $\sqcup_1$. With these, we can define the *satisfaction signal* $[\![\varphi]\!]_{\tilde{\mathcal{R}}} : \mathbb{R}_{\geq 0} \to \mathbb{U}_2$ of an STL formula φ with respect to a partial reachable set $\tilde{\mathcal{R}} : \mathcal{T} \to 2^{\mathcal{X}}$, where $\mathcal{T} \subseteq \mathbb{R}_{\geq 0}$. For atomic formulas *true* and $a \in \mathcal{AP}$, we define

$$[\![\mathit{true}]\!]_{\tilde{\mathcal{R}}}(t) := \top \qquad \text{and} \qquad [\![a]\!]_{\tilde{\mathcal{R}}}(t) := \begin{cases} \hat{a}(\tilde{\mathcal{R}}(t)) & \text{if } t \in \mathcal{T} \\ \dashv_2 & \text{otherwise} \end{cases}$$

for all $t \in \mathbb{R}_{\geq 0}$. The satisfaction signal for compound formulas is defined analogously to Sec. 4.2 using our operators. Finally, we state the equivalent of Thm. 1 to relate $\mathbb{U}_2$- and $\mathbb{U}_1$-satisfaction signals: We show that $[\![\varphi]\!]_{\mathcal{R}}$ refines $[\![\varphi]\!]_{\tilde{\mathcal{R}}}$, if $\mathcal{R}$ is an extension of $\tilde{\mathcal{R}}$. Given $\tilde{\mathcal{R}} : \mathcal{T} \to 2^{\mathcal{X}}$ with $\mathcal{T} \subseteq \mathbb{R}_{\geq 0}$, we say that the reachable set $\mathcal{R}$ *extends* $\tilde{\mathcal{R}}$, denoted as $\mathcal{R} \prec \tilde{\mathcal{R}}$, if $\mathcal{R}(t) = \tilde{\mathcal{R}}(t)$ for all $t \in \mathcal{T}$.

Theorem 2. *Suppose $\tilde{\mathcal{R}}$ is a partial reachable set and φ an STL formula. For every reachable set $\mathcal{R}$ that extends $\tilde{\mathcal{R}}$, we have $[\![\varphi]\!]_{\mathcal{R}} \prec [\![\varphi]\!]_{\tilde{\mathcal{R}}}$. In other words, if $[\![\varphi]\!]_{\tilde{\mathcal{R}}}(t) = v$, we have $[\![\varphi]\!]_{\mathcal{R}}(t) = v$ for all $\mathcal{R} \prec \tilde{\mathcal{R}}$, $v \in \mathbb{U}_1$, and $t \in \mathbb{R}_{\geq 0}$.*

Sketch of proof. The proof proceeds by structural induction on φ. For the most part, it is analogous to the proof of Thm. 1, except that we now have to consider an additional case for $v = \dashv_1$. In the base case for $\varphi = a$ with $a \in \mathcal{AP}$, we derive:

$$\begin{aligned} [\![a]\!]_{\tilde{\mathcal{R}}}(t) = \dashv_1 &\iff t \in \mathcal{T} \text{ and } \hat{a}(\tilde{\mathcal{R}}(t)) = \dashv_1 \\ &\implies \forall \mathcal{R} \prec \tilde{\mathcal{R}} : \hat{a}(\mathcal{R}(t)) = \dashv_1 \\ &\iff \forall \mathcal{R} \prec \tilde{\mathcal{R}} : [\![a]\!]_{\mathcal{R}}(t) = \dashv_1, \end{aligned}$$

where the partial reachable set $\tilde{\mathcal{R}}$ is defined over $\mathcal{T} \subseteq \mathbb{R}_{\geq 0}$. For $\top$ and $\bot$, we argue similarly. For the inductive cases with compound formulas, we can determine direct semantics for our operators on $\mathbb{U}_2$-signals similar to those in the proof of Thm. 1. Like (2), they follow the pattern of making the "otherwise" case of the $\mathbb{U}_1$-semantics explicit and introducing a new "otherwise" case to handle $\dashv_2$. As for Thm. 1, the crucial points in the proof are those where we need to use a statement about all refinements of a $\mathbb{U}_2$-signal $\tilde{\Lambda}$ to infer something about $\tilde{\Lambda}$ itself. In particular, we need variations of the following property

$$\forall \Lambda \prec \tilde{\Lambda} : \exists t \in \mathbb{R}_{\geq 0} : \Lambda(t) \in \mathcal{U} \iff \exists t \in \mathbb{R}_{\geq 0} : \tilde{\Lambda}(t) \in \mathcal{U},$$

where $\mathcal{U} \subsetneq \mathbb{U}_1$. To prove the forward direction, we choose a value $v \in \mathbb{U}_1 \setminus \mathcal{U}$, which must exist since $\mathcal{U}$ is a proper subset of $\mathbb{U}_1$. We consider the refinement $\Lambda_v \prec \tilde{\Lambda}$ in which all occurrences of $\dashv_2$ are replaced by v and find that $\Lambda_v(t) \in \mathcal{U}$ if and only if $\tilde{\Lambda}(t) \in \mathcal{U}$ to establish the forward direction. Using the direct semantics, we show the inductive cases analogously to the proof of Thm. 1. □

5 Incremental Verification of Hybrid Systems

Thms. 1 and 2 provide a sound, but incomplete, method of proving or disproving that a hybrid system $\mathcal{H}$ satisfies an STL specification φ. We note that incompleteness is unavoidable to some extent since the reachability problem for hybrid systems is undecidable in general [19, Sec. 4]. We summarize the verification method in the following corollary.

Corollary 1. *Let φ be an STL formula and $\tilde{\mathcal{R}}$ a partial reachable set. Let $\mathcal{R}$ be an extension of $\tilde{\mathcal{R}}$. If $[\![\varphi]\!]_{\tilde{\mathcal{R}}}(0)$ is*

- *$\top$, we have $\xi \models \varphi$ for all executions ξ covered by $\mathcal{R}$.*
- *$\bot$, we have $\xi \not\models \varphi$ for all executions ξ covered by $\mathcal{R}$.*
- *$\dashv_1$, the result is unknown. Based on the reachable set $\mathcal{R}$, we cannot make a statement about all covered executions.*
- *$\dashv_2$, the result is inconclusive. The partial reachable set $\tilde{\mathcal{R}}$ does not contain enough information to support a claim about all its extensions.*

If $\tilde{\mathcal{R}}$ is a partial overapproximation of the reachable set of a hybrid system $\mathcal{H}$, $[\![\varphi]\!]_{\tilde{\mathcal{R}}}(0) = \top$ implies $\mathcal{H} \models \varphi$, while $[\![\varphi]\!]_{\tilde{\mathcal{R}}}(0) = \bot$ implies $\mathcal{H} \not\models \varphi$.

The last claim above holds because there must exist some extension of $\tilde{\mathcal{R}}$ covering all executions of the system by definition of the reachable set.

5.1 Incremental Verification Algorithm

Alg. 1 implements the approach outlined in Cor. 1. It incrementally computes a $\mathbb{U}_2$-satisfaction signal of the specification φ as the reachability analysis of the hybrid system $\mathcal{H}$ progresses. At its core, the algorithm alternates between computing the reachable states of the system, which allows it to observe the satisfaction or violation of predicates in new time intervals, and propagating these observations up the syntax tree of φ to update the satisfaction signal. The algorithm terminates as soon as the satisfaction signal provides a conclusive verdict. Below, we explain Alg. 1 in more detail.

Before entering the main loop, we initialize the reachability analysis of $\mathcal{H}$ and construct the syntax tree of φ. We assume that φ is written without syntactic sugar. Each node n of the syntax tree stores the $\mathbb{U}_2$-satisfaction signal of the subformula of φ that its subtree represents. Initially, the satisfaction signal, which we refer to as $n.signal$, is $\dashv_2$ on the entire time domain, except for nodes representing *true*, where it is $\top$ everywhere.

In the main loop, we first compute one step of the reachability analysis, which yields the set of states $\mathcal{R}_I$ that the system can reach in the time interval I. For each node representing an atomic predicate $a \in \mathcal{AP}$, we determine $\hat{a}(\mathcal{R}_I)$ based on the observed set (function Eval) and update its satisfaction signal during I accordingly. Afterward, we propagate the new observations up the syntax tree. Note that all occurring signals are guaranteed to be of finite variability as long as the computed sequence of reachable sets does not exhibit Zeno behavior. If

Algorithm 1 Incremental Verification

Input: STL formula φ, hybrid system $\mathcal{H}$, time horizon t_h
Output: Verdict from $\mathbb{U}_2$ on whether $\mathcal{H}$ satisfies φ

```
1: reach ← InitializeReachabilityAnalysis(H, t_h)
2: tree ← SyntaxTree(φ)
3: while ¬reach.Done() do                          ▷ Did we reach t_h?
4:     R_I ← reach.NextStep()                      ▷ Reachable states in time interval I
5:     for ap ∈ tree.aps do                        ▷ Iterate nodes representing atomic predicates
6:         v ← ap.Eval(R_I)                        ▷ Set-based predicate evaluation to v ∈ U_1
7:         ap.signal.Set(I, v)
8:     end for
9:     tree.root.Propagate()                       ▷ See Alg. 2
10:    verdict ← tree.root.signal(0)
11:    if verdict ≠ ⊣_2 then                       ▷ Is the verdict conclusive?
12:        return verdict
13:    end if
14: end while
15: return tree.root.signal(0)
```

we obtain a conclusive verdict on the satisfaction of the top-level formula φ at time 0, we return the verdict early. Otherwise, we continue until the reachable set is determined up to a given time horizon t_h. After reaching t_h, we return the current verdict, even if it is the inconclusive $\dashv_2$.

Alg. 2 propagates new observations up the syntax tree. It closely follows the incremental marking procedure of Maler and Ničković [27, Alg. 2]. The algorithm traverses the syntax tree in post-order and applies the operators developed in Sec. 4.3 to update the satisfaction signal of each node based on the satisfaction signals of its children (function Combine).

Since we only admit future connectives, the satisfaction of a formula at time t depends only on the truth values of its subformulas at times $t' \geq t$ [27, Sec. 3.2]. To exploit this, every node n stores a time interval $n.irr$ of the form $[0, t)$ or $[0, t]$, indicating the prefix of $n.signal$ that is irrelevant for updates of the parent node. Initially, $n.irr$ is empty. After updating the signal of n, we also need to revise the irrelevant prefixes of its children. To this end, we find the largest interval I containing 0 such that $n.signal$ has a conclusive value, i.e., not $\dashv_2$, everywhere in $I \setminus n.irr$; if $n.signal(0) = \dashv_2$ and $0 \notin n.irr$, we return an empty interval (function ConclusiveInterval). We exclude $n.irr$ from consideration because $n.signal$ itself is irrelevant at these times. Then, we can drop the irrelevant prefix from memory by overwriting it with $\dashv_2$.

Remark 1 (Propagation Frequency). If we conduct the reachability analysis with a small time step size, we obtain numerous observations. To reduce the overhead incurred by signal propagation, we can accumulate the observations of several reachability steps and then propagate them all at once. However, in doing so,

Algorithm 2 Signal Propagation

Input: Syntax tree node n

```
1: function n.Propagate()
2:     if n.IsLeaf() then
3:         return
4:     end if
5:     for c ∈ n.children do
6:         c.Propagate()
7:     end for
8:     Δ ← n.Combine({c.signal | c ∈ n.children})
9:     n.signal.Merge(Δ)        ▷ Overwrite n.signal with Δ where Δ is not ⊣2
10:    for c ∈ n.children do
11:        c.irr ← ConclusiveInterval(n.signal, n.irr)
12:        c.signal.Set(c.irr, ⊣2)        ▷ Drop the irrelevant prefix from memory
13:    end for
14: end function
```

we might compute more reachability steps than required to reach a conclusive verdict. Choosing a propagation frequency is thus a trade-off similar to the one mentioned in [27, Sec. 5.3]. In the extreme case, where we never propagate before reaching the time horizon, we obtain an offline method similar to [22, 35].

5.2 Refinement via Branching the Reachability Analysis

Since we use the reachable set as the basis for verification, our approach works best if all executions of the system under scrutiny behave roughly similarly. The intuition for this is given at the end of the proof of Thm. 1: To verify that $\varphi_1 \mathbf{U}_I \varphi_2$ holds, we require that all executions covered by the reachable set satisfy the eventuality φ_2 at the same time. However, hybrid systems can have executions with vastly different behavior, e.g., due to discrete transitions changing the continuous dynamics. Moreover, the system behavior might strongly depend on the initial state. For these systems, our algorithm would often return $\dashv_1$, as the executions covered by the reachable set do not synchronize as required.

The underlying problem is that we compute just one reachable set to cover all system executions. If these executions have significant differences, the reachable set also covers many additional *spurious* executions that are infeasible according to the system dynamics. To alleviate this issue, we can perform the reachability analysis in multiple *branches* so that each branch only covers similar executions. For example, we could start a new branch whenever a discrete transition occurs. In addition, we could partition the set of initial states so that we analyze initial states that lead to vastly different behavior in separate branches.

To adapt Alg. 1 for several branches, we clone the syntax tree whenever a new branch starts. We then process each branch using its copy of the syntax

tree and combine the verdicts. If the verdicts are conclusive for all branches, we merge them using $\sqcup_1$, i.e., we return $\top$ or $\bot$ if all branches agree on the verdict, and $\dashv_1$ otherwise. If the analysis is inconclusive for at least one branch after reaching the time horizon t_h, the combined verdict is also $\dashv_2$. In this case, we need to extend the time horizon only for the inconclusive branches.

6 Evaluation

We implemented our algorithm in MATLAB using CORA[2] [2] for reachability analysis. First, we demonstrate the capabilities and limitations of our method on a simple hybrid system. Then, we apply it to autonomous driving and systems biology. For all experiments, our algorithm is configured to accumulate 20 observations before propagation. Note that CORA continues the reachability analysis in a new branch (cf. Sec. 5.2) after the system takes a discrete transition.

6.1 Bouncing Ball

First, let us consider a bouncing ball, which is a classic example of a hybrid system. The bouncing ball has two state variables: height h and velocity v. It accelerates under the influence of gravity and bounces back up once it hits the ground ($h = 0$). Bouncing is modeled as a discrete transition that reduces the velocity and flips its sign (see, e.g., [33, Sec. 2.2.3] for a full derivation). Initially, we have $h \in [0.95, 1.05]$ and $v \in [-0.05, 0.05]$, which means that the first bounce happens after about 0.5 s. The time horizon for our verification algorithm is $t_\mathrm{h} := 2.5\,\mathrm{s}$ and the reachability analysis uses a time step size of 0.01 s.

Tab. 1 summarizes the results of our experiments. The *future reach* of a formula is the amount of time it maximally looks into the future [21, Sec. 3]. Thus, approaches like [22, 35] need to perform reachability analysis up to the future reach of the specification. We can successfully verify the first two specifications and falsify the third while terminating the reachability analysis well before their future reach. Even though the fourth property also holds for the system (recall that the ball bounces after about 0.5 s), we cannot verify it since the reachable set is not sufficiently accurate. However, as $\dashv_1$ is already returned after 0.7 s, we know early that we need to refine the reachable set. After refining the reachability analysis by using a time step size of 0.002 s, we can also prove this specification. The final property demonstrates the drawbacks of accumulating observations: While we could reject it already after observing the initial set, our algorithm only returns $\bot$ after 20 time steps.

For the last column of Tab. 1, we applied the verification algorithm by Roehm et al. [32] to the bouncing ball. Since this algorithm requires the reachable set for the entire future reach of the specification in order to start the verification, it is not applicable to the first property. We cannot compute the reachable set for an infinite time horizon here, as no fixed point is detected. As shown by the fourth

[2] https://cora.in.tum.de

Table 1. Application of our approach and the approach from [32] to the bouncing ball

		Our approach		[32]
Specification	Future reach	Verdict	Termination	Verdict
$\mathbf{F}_{[0.2,\infty)}\, h < 0.5$	∞	$\top$	0.4 s	N/A
$\mathbf{F}_{[0,0.1]}(v < 0\, \mathbf{U}_{[0,1]}\, h < 0.25)$	1.1 s	$\top$	0.6 s	$\top$
$\mathbf{F}_{[0,1]}(h < 0.1 \wedge \mathbf{G}_{[0,2]}\, h < 0.3)$	3.0 s	$\bot$	1.3 s	$\bot$
$\mathbf{F}_{[0,0.1]}(v < 0\, \mathbf{U}_{[0,1]}\, h < 0.01)$	1.1 s	$\dashv_1$	0.6 s	$\bot$
$\mathbf{G}_{[0,1]}\, h < 0.1$	1.0 s	$\bot$	0.2 s	$\bot$

specification, [32] returns $\bot$ whenever it fails to verify a property. Therefore, in contrast to our algorithm, it does not distinguish between insufficient accuracy of the reachable set and actual falsification of the property by the system. For the other properties, its verdict is the same as ours.

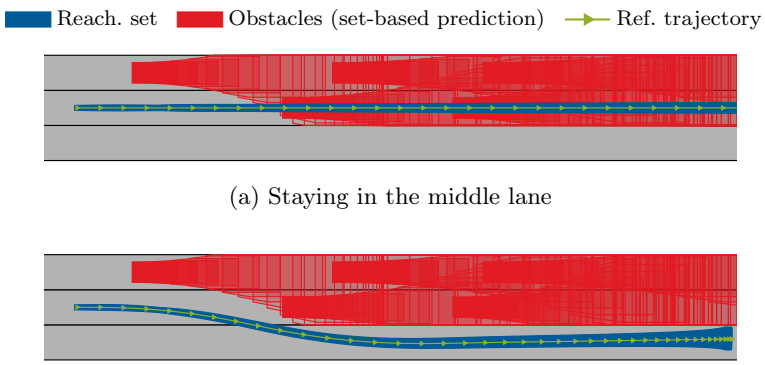
6.2 Autonomous Driving

Next, we examine an application for autonomous driving in the context of the CommonRoad[3] [5] framework. In our example scenario, an autonomous vehicle is driving in the middle lane of an interstate with another vehicle in front indicating to change from the left to the middle lane (see Fig. 5; CommonRoad scenario ID: `ZAM_HW-1_1_S-1`). Suppose the motion planner has determined two reference trajectories that the autonomous vehicle could follow for the next five seconds: one where it stays in its current lane and another where it changes to the right lane. We want to verify that the autonomous vehicle avoids collisions with other vehicles for the entire planned trajectory, even if it cannot precisely follow the trajectory due to disturbances. Moreover, it should eventually enter the right lane and stay there for at least 1 s. We formalize these requirements in STL as $(\mathbf{G}_{[0,4]}(x, y) \notin \mathcal{O}) \wedge \mathbf{F}_{[0,4]}\, \mathbf{G}_{[0,1]}(x, y) \in \mathcal{L}$, where (x, y) is the position of the autonomous vehicle, $\mathcal{O}$ is the area occupied by other vehicles according to a set-based prediction, and $\mathcal{L}$ is the right lane. Here, obtaining a verdict as soon as possible is particularly important since the autonomous vehicle has limited time to decide which trajectory to follow. With our algorithm, the vehicle can quickly reject the "stay" trajectory after computing the reachable set up to 1 s. Then, it can spend the remaining time on verifying the "change" trajectory, which yields the verdict $\top$ after performing reachability analysis up to 4 s. For the reachability analysis, we adopted a kinematic single-track model [31, Sec. 2.2] of the vehicle and assumed it tracks the reference trajectory using a P controller.

6.3 Genetic Oscillator

Finally, we consider the 9-dimensional genetic oscillator example (state variables $x_1, \ldots, x_9$) from [35, Sec. 5]. In [35], the authors verified the specifi-

[3] https://commonroad.in.tum.de



(a) Staying in the middle lane

(b) Changing to the right lane

Fig. 5. Reachable set projected to the position domain for two reference trajectories

cation $\mathbf{G}_{[0,1]}(a_1 \vee \mathbf{G}_{[3,3.5]}\, a_2)$, where $a_1 := x_6 - 1 > 0$ and $a_2 := 0.032 - 125^2(x_4 - 0.003)^2 - 3(x_6 - 0.5)^2 > 0$. We could not verify the original property since the reachable sets computed by CORA were not accurate enough, resulting in the verdict $\dashv_1$. After slightly relaxing the property by using $a_2' := 0.04 - 125^2(x_4 - 0.003)^2 - 3(x_6 - 0.5)^2 > 0$ instead of a_2, verification succeeded. The reachability analysis was stopped after 4.5 s, matching the future reach of the formula. If we change the specification to $\mathbf{G}_{[0,2]}(a_1 \vee \mathbf{G}_{[0,0.5]}\, a_2')$, we can reject it early after computing the reachable set for up to 1.3 s.

7 Conclusion

We proposed an incremental STL verification algorithm for hybrid systems based on reachability analysis and a four-valued semantics for STL. Due to its incremental nature, our algorithm can run alongside the reachability analysis and continuously update its verdict. Consequently, it can stop the computation of the reachable set as soon as the verdict becomes conclusive. This makes our approach particularly worthwhile for high-dimensional systems, for which reachability analysis is computationally expensive. The evaluation of our prototype showed promising results across several application domains.

Acknowledgments. The authors gratefully acknowledge funding from the German Research Foundation (DFG) under grant numbers AL 1185/20-1 and GRK 2428. Moreover, they thank Benedikt Seidl for implementing the system models used for the evaluation in CORA.

Disclosure of Interests. The authors have no competing interests to declare that are relevant to the content of this article.

References

1. Ahmad, H., Jeannin, J.B.: A program logic to verify signal temporal logic specifications of hybrid systems. In: Proc. of the Int. Conf. on Hybrid Systems: Computation and Control (HSCC). pp. 1–11 (2021). https://doi.org/10.1145/3447928.3456648
2. Althoff, M.: An introduction to CORA 2015. In: Proc. of the 1st and 2nd Workshop on Applied Verification for Continuous and Hybrid Systems. pp. 120–151 (2015). https://doi.org/10.29007/zbkv
3. Althoff, M., Dolan, J.M.: Online verification of automated road vehicles using reachability analysis. IEEE Transactions on Robotics **30**(4), 903–918 (2014). https://doi.org/10.1109/TRO.2014.2312453
4. Althoff, M., Frehse, G., Girard, A.: Set propagation techniques for reachability analysis. Annual Review of Control, Robotics, and Autonomous Systems **4**(1), 369–395 (2021). https://doi.org/10.1146/annurev-control-071420-081941
5. Althoff, M., Koschi, M., Manzinger, S.: CommonRoad: Composable benchmarks for motion planning on roads. In: Proc. of the IEEE Intelligent Vehicles Symp. (IV). pp. 719–726 (2017). https://doi.org/10.1109/IVS.2017.7995802
6. Alur, R., Feder, T., Henzinger, T.A.: The benefits of relaxing punctuality. Journal of the ACM **43**(1), 116–146 (1996). https://doi.org/10.1145/227595.227602
7. Bae, K., Lee, J.: Bounded model checking of signal temporal logic properties using syntactic separation. Proceedings of the ACM on Programming Languages **3**(POPL), 51:1–51:30 (2019). https://doi.org/10.1145/3290364
8. Bartocci, E., Deshmukh, J., Donzé, A., Fainekos, G., Maler, O., Ničković, D., Sankaranarayanan, S.: Specification-based monitoring of cyber-physical systems: A survey on theory, tools and applications. In: Bartocci, E., Falcone, Y. (eds.) Lectures on Runtime Verification: Introductory and Advanced Topics, pp. 135–175 (2018). https://doi.org/10.1007/978-3-319-75632-5_5
9. Bauer, A., Leucker, M., Schallhart, C.: Comparing LTL semantics for runtime verification. Journal of Logic and Computation **20**(3), 651–674 (2010). https://doi.org/10.1093/logcom/exn075
10. Bauer, A., Leucker, M., Schallhart, C.: Runtime verification for LTL and TLTL. ACM Transactions on Software Engineering and Methodology **20**(4), 14:1–14:64 (2011). https://doi.org/10.1145/2000799.2000800
11. Bogomolov, S., Forets, M., Frehse, G., Potomkin, K., Schilling, C.: JuliaReach: A toolbox for set-based reachability. In: Proc. of the Int. Conf. on Hybrid Systems: Computation and Control (HSCC). pp. 39–44 (2019). https://doi.org/10.1145/3302504.3311804
12. Brieger, M., Mitsch, S., Platzer, A.: Dynamic logic of communicating hybrid programs (2023). https://doi.org/10.48550/arXiv.2302.14546
13. Chai, M., Schlingloff, B.H.: Online monitoring of distributed systems with a five-valued LTL. In: Proc. of the IEEE Int. Symp. on Multiple-Valued Logic (ISMVL). pp. 226–231 (2014). https://doi.org/10.1109/ISMVL.2014.47
14. Chen, X., Ábrahám, E., Sankaranarayanan, S.: Flow*: An analyzer for non-linear hybrid systems. In: Sharygina, N., Veith, H. (eds.) Computer Aided Verification (CAV). pp. 258–263 (2013). https://doi.org/10.1007/978-3-642-39799-8_18
15. Deshmukh, J.V., Donzé, A., Ghosh, S., Jin, X., Juniwal, G., Seshia, S.A.: Robust online monitoring of signal temporal logic. Formal Methods in System Design **51**(1), 5–30 (2017). https://doi.org/10.1007/s10703-017-0286-7

16. Donzé, A., Maler, O.: Robust satisfaction of temporal logic over real-valued signals. In: Chatterjee, K., Henzinger, T.A. (eds.) Formal Modeling and Analysis of Timed Systems (FORMATS). pp. 92–106 (2010). https://doi.org/10.1007/978-3-642-15297-9_9
17. Ferrère, T., Maler, O., Ničković, D., Pnueli, A.: From real-time logic to timed automata. Journal of the ACM **66**(3), 19:1–19:31 (2019). https://doi.org/10.1145/3286976
18. Frehse, G., Le Guernic, C., Donzé, A., Cotton, S., Ray, R., Lebeltel, O., Ripado, R., Girard, A., Dang, T., Maler, O.: SpaceEx: Scalable verification of hybrid systems. In: Gopalakrishnan, G., Qadeer, S. (eds.) Computer Aided Verification (CAV). pp. 379–395 (2011). https://doi.org/10.1007/978-3-642-22110-1_30
19. Henzinger, T.A.: What's decidable about hybrid automata? Journal of Computer and System Sciences **57**(1), 94–124 (1998). https://doi.org/10.1006/jcss.1998.1581
20. Ho, H.M., Ouaknine, J., Worrell, J.: Online monitoring of metric temporal logic. In: Bonakdarpour, B., Smolka, S.A. (eds.) Runtime Verification (RV). pp. 178–192 (2014). https://doi.org/10.1007/978-3-319-11164-3_15
21. Hunter, P., Ouaknine, J., Worrell, J.: Expressive completeness for metric temporal logic. In: Proc. of the ACM/IEEE Symp. on Logic in Computer Science (LICS). pp. 349–357 (2013). https://doi.org/10.1109/LICS.2013.41
22. Ishii, D., Yonezaki, N., Goldsztejn, A.: Monitoring temporal properties using interval analysis. IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences **E99-A**(2), 442–453 (2016). https://doi.org/10.1587/transfun.E99.A.442
23. Kleene, S.C.: On notation for ordinal numbers. The Journal of Symbolic Logic **3**(4), 150–155 (1938). https://doi.org/10.2307/2267778
24. Kochdumper, N., Bak, S.: Fully automated verification of linear time-invariant systems against signal temporal logic specifications via reachability analysis. Nonlinear Analysis: Hybrid Systems **53**, 101491 (2024). https://doi.org/10.1016/j.nahs.2024.101491
25. Lee, J., Yu, G., Bae, K.: Efficient SMT-based model checking for signal temporal logic. In: Proc. of the IEEE/ACM Int. Conf. on Automated Software Engineering (ASE). pp. 343–354 (2021). https://doi.org/10.1109/ASE51524.2021.9678719
26. Maler, O., Ničković, D.: Monitoring temporal properties of continuous signals. In: Lakhnech, Y., Yovine, S. (eds.) Formal Techniques, Modelling and Analysis of Timed and Fault-Tolerant Systems (FORMATS/FTRTFT). pp. 152–166 (2004). https://doi.org/10.1007/978-3-540-30206-3_12
27. Maler, O., Ničković, D.: Monitoring properties of analog and mixed-signal circuits. International Journal on Software Tools for Technology Transfer **15**(3), 247–268 (2013). https://doi.org/10.1007/s10009-012-0247-9
28. Ničković, D., Lebeltel, O., Maler, O., Ferrère, T., Ulus, D.: AMT 2.0: Qualitative and quantitative trace analysis with extended signal temporal logic. International Journal on Software Tools for Technology Transfer **22**(6), 741–758 (2020). https://doi.org/10.1007/s10009-020-00582-z
29. Platzer, A.: Differential dynamic logic for hybrid systems. Journal of Automated Reasoning **41**(2), 143–189 (2008). https://doi.org/10.1007/s10817-008-9103-8
30. Platzer, A.: Logical Foundations of Cyber-Physical Systems. Springer International Publishing (2018). https://doi.org/10.1007/978-3-319-63588-0
31. Rajamani, R.: Vehicle Dynamics and Control. Springer (2012). https://doi.org/10.1007/978-1-4614-1433-9

32. Roehm, H., Oehlerking, J., Heinz, T., Althoff, M.: STL model checking of continuous and hybrid systems. In: Artho, C., Legay, A., Peled, D. (eds.) Automated Technology for Verification and Analysis (ATVA). pp. 412–427 (2016). https://doi.org/10.1007/978-3-319-46520-3_26
33. van der Schaft, A., Schumacher, H.: An Introduction to Hybrid Dynamical Systems. Springer (2000). https://doi.org/10.1007/BFb0109998
34. Thati, P., Roşu, G.: Monitoring algorithms for metric temporal logic specifications. Electronic Notes in Theoretical Computer Science **113**, 145–162 (2005). https://doi.org/10.1016/j.entcs.2004.01.029
35. Wright, T., Stark, I.: Property-directed verified monitoring of signal temporal logic. In: Deshmukh, J., Ničković, D. (eds.) Runtime Verification (RV). pp. 339–358 (2020). https://doi.org/10.1007/978-3-030-60508-7_19