Optimization-Based Model Checking and Trace Synthesis for Complex STL Specifications

Sota Sato[1,3(✉)], Jie An[1,4(✉)], Zhenya Zhang[1,2(✉)], and Ichiro Hasuo[1,3(✉)]

[1] National Institute of Informatics, Tokyo, Japan
{sotasato,jiean,hasuo}@nii.ac.jp
[2] Kyushu University, Fukuoka, Japan
zhang@ait.kyushu-u.ac.jp
[3] SOKENDAI (The Graduate University for Advanced Studies), Tokyo, Japan
[4] Institute of Software, Chinese Academy of Sciences, Beijing, China

Abstract. Techniques of light-weight formal methods, such as monitoring and falsification, are attracting attention for quality assurance of cyber-physical systems. The techniques require formal specs, however, and writing right specs is still a practical challenge. Commonly one relies on *trace synthesis*—i.e. automatic generation of a signal that satisfies a given spec—to examine the meaning of a spec. In this work, motivated by 1) complex STL specs from an automotive safety standard and 2) the struggle of existing tools in their trace synthesis, we introduce a novel trace synthesis algorithm for STL specs. It combines the use of MILP (inspired by works on controller synthesis) and a *variable-interval encoding* of STL semantics (previously studied for SMT-based STL model checking). The algorithm solves model checking, too, as the dual of trace synthesis. Our experiments show that only ours has realistic performance needed for the interactive examination of STL specs by trace synthesis.

1 Introduction

Safety and quality assurance of *cyber-physical systems (CPSs)* is an important and multifaceted problem. The pervasiveness and safety-critical nature of CPSs makes the problem imminent and pressing; at the same time, the problem comes with very different flavors in different application domains, calling for different solutions. For example, in the aerospace domain, full formal verification all the way up from the codebase seems feasible [33]. Such is a luxury that the automotive domain may not afford, however, because of short product cycles, dependence on third-party (thus black-box) components, heterogeneous environmental uncertainties, and fierce competition (thus tight budget).

The authors are supported by ERATO HASUO Metamathematics for Systems Design Project (No. JPMJER1603), the START Grant No. JPMJST2213, the ASPIRE grant No. JPMJAP2301, JST. S.S. is supported by KAKENHI No. 23KJ1011, JSPS. Z.Z. is supported by JSPS KAKENHI Grant No. JP23K16865 and No. JP23H03372.

A. Gurfinkel and V. Ganesh (Eds.): CAV 2024, LNCS 14683, pp. 282–306, 2024.
https://doi.org/10.1007/978-3-031-65633-0_13

The above limitations in the automotive domain point, in the formal methods terms, to the *absence of white-box system models*. This has led to the flourish of *light-weight formal methods*, such as monitoring [8], runtime verification, and hybrid system falsification [16]. These are logic-based methods that operate on *formal specifications*, often given in *signal temporal logic (STL)* [24]. These methods give up comprehensive guarantee due to the absence of white-box system models; yet their values in practical usage scenarios are widely acknowledged.

Trace Synthesis and Model Checking. In this paper, we are motivated by some automotive instances of the *trace synthesis* problem: it asks to synthesize an execution trace σ of a system $\mathcal{M}$ that satisfies a given STL specification φ. There are two major approaches to trace synthesis for CPSs.

One common approach is via *hybrid system falsification* [16]: here, we try many input signals τ for $\mathcal{M}$, iteratively modifying them in the direction of satisfying φ; the quantitative *robust semantics* of STL [17] serves as an objective function that allows hill-climbing optimization. It is notable that the system model $\mathcal{M}$ can be *black-box*: we do not need to know its internal working; it is enough to compute the execution trace $\mathcal{M}(\tau)$ under given input τ. Falsification has attracted a lot of interest especially in the automotive domain; see e.g. [16].

We take the other approach to trace synthesis, namely as the *dual of the model checking problem*. Here model checking decides if, under *any* input τ, the execution trace $\mathcal{M}(\tau)$ satisfies φ. Our choice of this approach may be puzzling—it requires a white-box model $\mathcal{M}$, but it is rare in the automotive domain.

Analyzing Specifications (Rather Than Models). Our choice of the model checking approach to trace synthesis comes from the following basic scope of the paper: *we use trace synthesis to analyze the quality of specifications (specs)*.

This is in stark contrast with many falsification tools whose scope is analyzing *models*. There, a model $\mathcal{M}$ is extensive and complex (typically a Simulink model of an actual product), and counterexample traces are used for "debugging" $\mathcal{M}$.

In this paper, instead, a model $\mathcal{M}$ is simple and white-box (it can even be the trivial model, where the input and output are the same), but a spec φ tends to be complex. One typical usage scenario for our framework is when φ is a *normative rule*—such as a law, a traffic rule, or a property required in an international standard—in which case φ is imposed on many different systems (e.g. different vehicle models). Then $\mathcal{M}$ should be a simple overapproximation of a variety of systems, rather than a detailed system model.

Another typical usage scenario of our framework is an early "requirement development" phase of the *V-model* of the automotive system design. Here, engineers fix specs that pin down the later development efforts, in which those specs get refined and realized. They want to confirm that the specs are sensible (e.g. there is no mutual conflict) and faithful to their intentions. Since a system is yet to be developed, a system model $\mathcal{M}$ cannot be detailed.

Motivating Example. More specifically, the current work is motivated by the work [30] on formalizing disturbance scenarios in the ISO 34502 standard for automated driving vehicles. There, a vehicle dynamics model is simple (the scenarios should apply to different vehicle models—see above), but STL formulas are complex. It is observed that existing algorithms have a hard time handling the complexity of specs (see §6 for experiments). This motivated our current technical development, namely a trace synthesis algorithm that exploits *white-box models* and *MILP optimization* for efficiency.

Fig. 1. Rear-end near collision

The following example illustrates the challenge encountered in [30].

Example 1.1 (rear-end near collision). We would like to express, in STL, a *rear-end near collision* scenario for two cars. It refers to those driving situations where a rear car $\mathrm{Car_r}$ comes too close to a front car $\mathrm{Car_f}$. We assume a single-lane setting (Fig. 1), so we can ignore lateral dynamics.

Consider the following STL formulas. Here, $x_\mathrm{f}, v_\mathrm{f}, a_\mathrm{f}$ are the variables for the position, velocity, and acceleration of $\mathrm{Car_f}$; the other variables are for $\mathrm{Car_r}$.

$$\begin{array}{rcl} \texttt{danger} & :\equiv & x_\mathrm{f} - x_\mathrm{r} \leq 10 \\ \texttt{dyn\_inv} & :\equiv & x_\mathrm{f} - x_\mathrm{r} \geq 0 \wedge 2 \leq v_\mathrm{f} \leq 27 \wedge 2 \leq v_\mathrm{r} \leq 27 \\ \texttt{trimming} & :\equiv & (\Diamond \texttt{danger}) \Rightarrow \big((\Box_{[0,0.2]} a_\mathrm{r} \geq 0.5)\, \mathcal{U}\, \texttt{danger}\big) \\ \texttt{RNC1} & :\equiv & \Box(\texttt{dyn\_inv} \wedge \texttt{trimming}) \wedge \Diamond_{[0,9]} \Box_{[0,1]} \texttt{danger} \end{array} \tag{1}$$

The last formula `RNC1` formalizes rear-end near collision; in particular, its subformula $\Diamond_{[0,9]}\Box_{[0,1]}\texttt{danger}$ requires that `danger` occurs within 9 s and persists for at least one second.

The formula `RNC1` comes with two auxiliary conditions: `dyn_inv` and `trimming`. We shall now exhibit their content and why they are needed. In fact, these conditions arose naturally in the course of *trace synthesis*, the problem of our focus.

Specifically, in [30], we conducted trace synthesis repeatedly in order to 1) *illustrate* the meaning of STL specifications and 2) *confirm* that they reflect informal intentions. The generated traces were animated for graphical illustration. This workflow is much like in the tool STLInspector [31].

The formula `dyn_inv` imposes basic constraints on the dynamics of the cars. In the trace synthesis in [30], without this basic constraint, we obtained a number of nonsensical example traces in which a car warps and instantly passes the other, drives much faster than the legal maximum, and so on.

The formula `trimming` requires $\mathrm{Car_r}$ to accelerate until `danger` occurs. It was added to limit a generated trace to an interesting part. For example, a trace can have `danger` only after a 8-s pacific journey; animating this whole trace can easily bore users. The condition trims such a trace to the part where $\mathrm{Car_r}$ is accelerating towards `danger`.

The dynamics model used in [30] is the following simple one:

$$\dot{x_{\mathrm{f}}} = v_{\mathrm{f}},\ \dot{v_{\mathrm{f}}} = a_{\mathrm{f}}; \qquad \dot{x_{\mathrm{r}}} = v_{\mathrm{r}},\ \dot{v_{\mathrm{r}}} = a_{\mathrm{r}}. \tag{2}$$

This relates x, v and a in the spec (1). The double integrator model is certainly simplistic, but it suffices the purpose in [30] of illustrating and confirming specs.

Remark 1.2. In [30], after illustrating and confirming STL specs through trace synthesis, the final goal was to use them for monitoring actual driving data. Neither the dynamics model (2) nor the condition `dyn_inv` is really relevant to monitoring—actual driving data should comply with them anyway. In contrast, `trimming` is important, in order to extract only relevant parts of the data.

Technical Solution: MILP-Based Trace Synthesis. We present a novel trace synthesis algorithm. Note that it also solves the dual problem, namely STL model checking. It originates from two recent lines of work: MILP-based optimal control [14,28,29] and SMT-based STL model checking [7,23,34].

The controller synthesis techniques in [14,28,29] exploit *mixed-integer linear programming (MILP)* for efficiency. The optimal control problem that they solve can be specialized to our trace synthesis problem (detailed discussions come later). But we found their capability of handling complex specs (as in Ex. 1.1) limited, largely because of their *constant-interval encoding* to MILP.

We solve this challenge by our novel *variable-interval encoding* of the STL semantics to MILP. It is inspired by the *stable partitioning* technique introduced in [7]: the technique is used in [7,23,34] for *logical* encoding towards SMT-based model checking; we use it for *numerical* encoding to MILP. This way we will solve the *bounded* trace synthesis problem—in the sense that variability of the truth values of the relevant formulas is bounded—much like in [7,23,34]. For our MILP encoding, however, we need special care since MILP does not accommodate strict inequalities (partitions such as $\dots, (\gamma_{i-1}, \gamma_i), \{\gamma_i\}, (\gamma_i, \gamma_{i+1}), \dots$ in [7] cannot be expressed). We therefore use a novel technique called *δ-stable partitioning*.

Overall, our algorithm works as follows. We assume that a system model $\mathcal{M}$ can be MILP-encoded, either exactly or approximately. Some model families are discussed in §5. This assumption, combined with our key technique of *variable-interval MILP encoding* of STL, reduces trace synthesis to an MILP problem, which we solve by Gurobi Optimizer [20]. We conduct experimental evaluation to confirm the scalability of our algorithm, especially for complex specs (§6).

Our algorithm is *anytime* (i.e. *interruptible*): even if the budget runs out in the course of optimization, a best-effort result (the trace that is the closest to a solution so far) is obtained. A similar benefit is there in case there is no execution trace σ that satisfies the spec φ: we obtain a trace σ' that is the closest to satisfy φ. Accommodation of *parameters* is another advantage thanks to our use of MILP; we exploit it for *parameter mining* for PSTL formulas. See §3.

Both controller synthesis techniques [14,28,29] and SMT-based model checking techniques [7,23,34] can be used for trace synthesis. The methodological differences are discussed later in §1; experimental comparison is made in §6.

Contributions and Organization. We summarize our contributions.

- We introduce an optimization-based algorithm for bounded trace synthesis for STL specs. It assumes that a system model is white-box and MILP-encodable; it also solves the dual problem (namely bounded model checking).
- As a key element, we introduce a *variable-interval* encoding of STL to MILP.
- MILP encodings of some system models, notably rectangular hybrid automata and double integrator dynamics (suited for the automotive domain).
- We experimentally confirm scalability of our algorithm, especially for complex specs. Comparison is made with MILP-based optimal control [14], SMT-based model checking [34], and optimization-based falsification [11,37].
- Through the algorithm, case studies and experiments, we argue for the importance and feasibility of *spec analysis* for CPSs.

After exhibiting preliminaries on STL and stable partitioning in §2, we formulate our problems (bounded trace synthesis, model checking, etc.) in §3. In §4 we present a novel *variable-interval MILP encoding* of STL; in §5 we discuss MILP encoding of a few families of models. Our main algorithm combines these two encodings. In §6 we present experiment results.

Related Work I: Optimal STL Control with MILP. The works [14,28,29] inspire our use of MILP for STL. Their problem is *optimal controller synthesis under STL constraints*, i.e. to find an input signal τ to a system model $\mathcal{M}$ so that 1) the output signal $\mathcal{M}(\tau)$ satisfies a given STL spec φ and 2) it optimizes $J(\mathcal{M}(\tau))$, where J is a given objective function. This problem subsumes our problem of trace synthesis, by taking a constant function as J.

The algorithms in [14,28,29] reduce their problem to MILP by a *constant-interval encoding* of the robust semantics [13,17] of STL (an enhanced encoding is presented in [22]). Specifically, their system model is discrete-time dynamics $x(t + \Delta t) = f_{\mathrm{d}}(x(t), u(t), w(t))$ with a constant interval Δt.

In contrast, in our *variable-interval* encoding (§4), continuous time is discretized into the intervals $\dots, (\gamma_{i-1}, \gamma_i), \{\gamma_i\}, (\gamma_i, \gamma_{i+1}), \dots$ where the end points γ_i are also variables in MILP. This is advantageous not only in modeling precision but also in scalability: for system models that are largely continuous, constant-interval discretization incurs more integer variables in MILP, hampering the performance of MILP solvers. See §6 for experimental comparison.

Related Work II: SMT-Based STL Model Checking. Our key technical element (a variable-interval MILP encoding of STL) uses the idea of stable partitioning from [7,23,34]. They solve bounded STL model checking, and also its dual (trace synthesis). The main difference is the class of system models $\mathcal{M}$ accommodated. SMT solvers accommodate more theories than MILP solving, and thus allows encoding of a greater class of models. In contrast, by restricting the model class to MILP-encodable, our algorithm benefits speed and scalability (MILP is faster than SMT). Iterative optimization in MILP also makes our algorithm an anytime one. Native support of parameter synthesis is another plus.

Other Related Work. *Optimization-based falsification* has its root in the quantitative robust semantics of STL [13,17]; the successful combination with stochastic optimization metaheuristics has made falsification an approach of both scientific and industrial interest. See the ARCH competition report [16] for state-of-the-art. Falsification is most of the time thought of as *search-based testing*; therefore, unlike the model checking approach, the absence of counterexamples is usually not proved. Exceptions are [25,35] where they strive for probabilistic guarantees for such absence.

The current work is motivated by the observation that falsification solvers often struggle in trace synthesis for complex STL specs, even if a system model is simple. It is known that specs with more connectives pose a performance challenge, and many countermeasures are proposed, including [2] (for temporal operators) and [36,37] (for Boolean connectives).

2 Preliminaries

We let $\mathbb{N}, \mathbb{R}$ denote the sets of natural numbers and reals, respectively; $\mathbb{R}_{\geq 0}$ denotes an obvious subset. The set $\overline{\mathbb{R}} = \mathbb{R} \cup \{-\infty, \infty\}$ is that of extended reals. The set $\mathbb{B} = \{\top, \bot\}$ is for Boolean truth values. The *powerset* of a set X is denoted by $\wp(X)$. An *interval* is a subset of $\mathbb{R}_{\geq 0}$ of the form (a, b), $[a, b)$, $(a, b]$, or $[a, c]$, where $a < b$ and $a \leq c$. Therefore a singleton $\{a\}$ is an interval.

Definition 2.1 (linear predicate $p$and $[\![p]\!], \pi_p$). Given a set V of variables, a *(closed) linear predicate* is a function $p\colon \mathbb{R}^V \to \mathbb{B}$ defined as follows, using some $c \in \mathbb{R}^V$ and $b \in \mathbb{R}$: $p(x) = \top$ if and only if $c^\top x + b \geq 0$. We write $[\![p]\!]$ for the closed half-space $\{x \mid p(x) = \top\} \subseteq \mathbb{R}^V$.

For the above p, we define a function $\pi_p(x)\colon \mathbb{R}^V \to \mathbb{R}$ by $\pi_p(x) \coloneqq c^\top x + b$. This is understood as the degree of satisfaction (or violation, if negative) of a linear predicate p by $x \in \mathbb{R}^V$. Indeed, $\pi_p(x)$ is the (signed) Euclidean distance to the boundary of $[\![p]\!]$, assuming that the Euclidean norm of c is $\|c\| = 1$.

Definition 2.2 (signal). Let V be a finite set of variables and T a positive real. A *signal* over V with a time horizon T is a function $\sigma : [0, T] \to \mathbb{R}^V$. We write $\mathbf{Signal}_V^T$ for the set of all signals over V with time horizon T, or simply $\mathbf{Signal}_V$ when T is clear from the context.

If necessary, the domain $[0, T]$ of σ can be extended to $\mathbb{R}_{\geq 0}$ by setting $\sigma(t) \coloneqq \sigma(T)$ for all $t > T$. This allows us to define the notion of *t-postfix*, which will serve as the basis of the STL semantics (§2.1). Precisely, the *t-postfix* of σ is a signal σ^t defined by $\sigma^t(t') \coloneqq \sigma(t + t')$. The domain of σ^t can be chosen freely but we set it to $[0, T]$ for consistency.

Definition 2.3 (system model, trace set $\mathcal{L}(\mathcal{M})$). Let V, V' be finite sets of variables. A *system model* $\mathcal{M}$ from V' to V with a time horizon T is a function $\mathcal{M}\colon \mathbf{Signal}_{V'}^T \to \wp(\mathbf{Signal}_V^T)$. The *trace set* $\mathcal{L}(\mathcal{M}) \coloneqq \bigcup_{\tau \in \mathbf{Signal}_{V'}^T} \mathcal{M}(\tau)$ is the set of all output signals of $\mathcal{M}$ where an input signal τ can vary.

We allow system models to be nondeterministic (note the the powerset construction $\wp$); the models in §1 were deterministic for simplicity. A special case of the above is when $V' = \emptyset$, that is, when $\mathcal{M}$ does not have any input. In this case, a system model $\mathcal{M}$ can be identified with a subset $\mathcal{L}(\mathcal{M}) \subseteq \mathbf{Signal}_V$.

Example 2.4 ($\mathcal{M}_{\mathrm{RNC}}$). The dynamics model in Ex. 1.1 is formalized as a system model $\mathcal{M}_{\mathrm{RNC}}$ whose input variables (in V') are $a_{\mathrm{f}}, v_{\mathrm{f}}^{\mathrm{init}}, x_{\mathrm{f}}^{\mathrm{init}}, a_{\mathrm{r}}, v_{\mathrm{r}}^{\mathrm{init}}, x_{\mathrm{r}}^{\mathrm{init}}$, and output variables (in V) are $a_{\mathrm{f}}, v_{\mathrm{f}}, x_{\mathrm{f}}, a_{\mathrm{r}}, v_{\mathrm{r}}, x_{\mathrm{r}}$. Here, the input is acceleration rates $(a_{\mathrm{f}}, a_{\mathrm{r}})$ and the initial values of velocities and positions (modeled using signals $v_{\mathrm{f}}^{\mathrm{init}}$ etc. for convenience). The time horizon T of $\mathcal{M}$ represents its simulation time; here we set $T = 20$. Given an input signal τ, the output $\mathcal{M}(\tau)$ is a singleton $\mathcal{M}(\tau) = \{\sigma\}$, and σ is determined by the ODE (2). Specifically, $\sigma(t)(a_{\mathrm{f}}) = \tau(t)(a_{\mathrm{f}})$, $\sigma(t)(v_{\mathrm{f}}) = \tau(0)(v_{\mathrm{f}}^{\mathrm{init}}) + \int_0^t \tau(t')(a_{\mathrm{f}})\,\mathrm{d}t'$, and so on.

2.1 Signal Temporal Logic

Definition 2.5 (signal temporal logic (STL)). In STL, an *atomic proposition* over a variable set V is represented as $p :\equiv (f(\vec{w}) \geq 0)$, where $f : \mathbb{R}^V \to \mathbb{R}$ is a function that maps a V-dimensional vector $\vec{w}$ to a real. The syntax of an STL formula φ (over V) is defined as follows: $\varphi :\equiv p \mid \bot \mid \top \mid \neg\varphi \mid \varphi_1 \vee \varphi_2 \mid \varphi_1 \wedge \varphi_2 \mid \Diamond_I \varphi \mid \Box_I \varphi \mid \varphi_1 \,\mathcal{U}_I\, \varphi_2 \mid \varphi_1 \,\mathcal{R}_I\, \varphi_2$, where I is a nonsingular closed time interval, and $\Diamond_I, \Box_I, \mathcal{U}_I, \mathcal{R}_I$ are temporal operators *eventually*, *always*, *until* and *release*. Implication is defined: $\varphi_1 \Rightarrow \varphi_2 :\equiv \neg\varphi_1 \vee \varphi_2$. We write temporal operators without the subscript I when $I = [0, \infty]$ (e.g., $\Diamond$). Note that we do not lose generality by restricting the inequality in $p :\equiv (f(\vec{w}) \geq 0)$. Indeed, $\leq, <, >$ can be encoded using (a combination of) $-f$ and $\neg$.

The set $\mathrm{Sub}(\varphi)$ collects all subformulas of an STL formula φ; the set $\mathrm{AP}(\varphi)$ collects all atomic propositions α occurring in φ.

Proposition 2.6. Every STL formula has a formula in the *negation normal form (NNF)*—i.e. one in which negation $\neg$ appears only in front of atomic propositions—that is semantically equivalent. □

Assumption 2.7. We assume that each atomic proposition p is a linear predicate (Def. 2.1), that is, $f(x) = c^\top x + b$ with some $c \in \mathbb{R}^V, b \in \mathbb{R}$ in each $p :\equiv (f(\vec{w}) \geq 0)$.

The Boolean semantics $\sigma \models \varphi$ and robust semantics $[\![\sigma, \varphi]\!] \in \overline{\mathbb{R}}$ of STL are standard. See [32, Appendix A].

PSTL is a parametric extension of STL. It is from [4]; see also [9]. Its definition is in [32, Appendix A]. The semantics of PSTL formula is defined naturally by fixing $\vec{u}, \vec{v}$; see Prob. 3.3 for the specific forms we use.

2.2 Finite Variability

The satisfiability checking problem for STL—this is equivalent to the model checking problem under the trivial (identity) system model—is already

EXPSPACE-complete [3]. To ease computational complexity, *bounded model checking* has been a common approach [23,26]. Its main idea is to bound the number of time-points at which the truth value of each subformula can vary.

Definition 2.8 (finite variability [27]). A (finite) *partition* $\mathcal{P}$ of an interval $D \subseteq \mathbb{R}$ is a sequence $\mathcal{P} = (J_i)_{i=1}^N$ of nonempty and mutually disjoint intervals such that $\bigcup_{i=1}^N J_i = D$, and $\sup(J_i) \leq \inf(J_{i'})$ for any $i < i'$. A Boolean signal $q \colon \mathbb{R}_{\geq 0} \to \mathbb{B}$ is *constant* on an interval $J \subseteq \mathbb{R}_{\geq 0}$ if $q(t) = q(t')$ for any $t, t' \in J$. We say $q(t)$ has N*-bounded variability* if there exists a partition $\mathcal{P}$ of $[0, \infty)$ and $q(t)$ is constant on every interval $J \in \mathcal{P}$.

Let $\sigma \colon [0, T] \to \mathbb{R}^V$ be a signal and φ be an STL formula over V. We say that σ has the N*-bounded variability* with respect to φ if the Boolean (truth value) signal $t \mapsto (\sigma^t \models \varphi)$ has the N-bounded variability. We say σ is *finitely variable* with respect to φ if it has the N-bounded variability for some N.

Finally, we say σ has the *hereditary* N*-bounded variability* with respect to φ if, for each subformula $\psi \in \mathrm{Sub}(\varphi)$, σ has the N-bounded variability with respect to ψ. We write N*-HBV* for the hereditary N-bounded variability.

Lemma 2.9 ([7]). Let φ be an STL formula. A signal σ has the N-HBV with respect to φ for some N if and only if it is finitely variable with respect to each atomic proposition $p \in \mathrm{AP}(\varphi)$ occurring in φ. □

The following is the basis of bounded model checking in [7,23].

Definition 2.10 (stable partition). Let σ be a signal, φ be an STL formula, and $\mathcal{P}$ be a partition of $[0, T]$ such that every $J \in \mathcal{P}$ is singular or open. Intuitively, $\mathcal{P}$ looks like $\{\gamma_0\}, (\gamma_0, \gamma_1), \{\gamma_1\}, (\gamma_1, \gamma_2), \ldots, \{\gamma_N\}$. We say $\mathcal{P}$ is a *stable partition* for σ and φ if $t \mapsto \sigma^t \models \psi$ is constant on J for each $J \in \mathcal{P}$, $\psi \in \mathrm{Sub}(\varphi)$.

3 Problem Formulation

We formulate our problems and discuss their mutual relationship. The next problem is studied in [7,23,34].

Problem 3.1 (bounded STL model checking). Given an STL formula φ (over V), a system model $\mathcal{M}$ (from V' to V) with time horizon T, and a variability bound $N \in \mathbb{N}$, decide if the following is true or not: $\sigma \models \varphi$ holds for an arbitrary trace $\sigma \in \mathcal{L}(\mathcal{M})$ (cf. Def. 2.3) that has the hereditary N-bounded variability (N-HBV) with respect to φ.

The following is the dual of Prob. 3.1, and is our main scope.

Problem 3.2 (bounded STL trace synthesis). Given $\varphi, \mathcal{M}, T$ and N as in Prob. 3.1, find a trace $\sigma \in \mathcal{L}(\mathcal{M})$ such that 1) σ has the N-HBV with respect to φ and 2) $\sigma \models \varphi$ holds, or prove that such σ does not exist.

Prob. 3.2 resembles the *falsification problem* [17]: given $\mathcal{M}$ (that can be black-box) and φ', find a *counterexample input* τ such that $\mathcal{M}(\tau) \not\models \varphi'$. The emphases and the settings are often different though; see §1.

The following is a special case of the *STL parameter mining* problem; see e.g. [9, § 3.5]. Recall from [32, Def. A.3] that $\varphi_{\vec{u},\vec{v}}$ instantiates parameters $\vec{p}, \vec{q}$ in φ with real values $\vec{u}, \vec{v}$ from the domains P, Q, respectively.

Problem 3.3 (bounded existential parameter mining). Let φ be a PSTL formula over parameters $(\vec{p}, \vec{q})$, and $\mathcal{M}, T$ and N be as in Prob. 3.1. Find the set $\left\{ (\vec{u}, \vec{v}) \in P \times Q \,\middle|\, \sigma \models \varphi_{\vec{u},\vec{v}} \text{ for some } \sigma \in \mathcal{L}(\mathcal{M}) \text{ that has the } N\text{-HBV wrt. } \varphi \right\}$.

In §6, we study a further special case where there is only one parameter p and the goal is to find the maximum p in the above set.

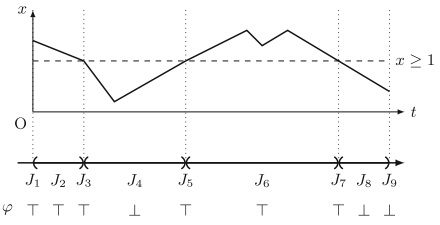

Fig. 2. A stable partition (cf. [7]) for σ and $\varphi :\equiv x \geq 1$. The symbols $\top$ and $\bot$ denote the (constant) truth value of φ each interval J_i.

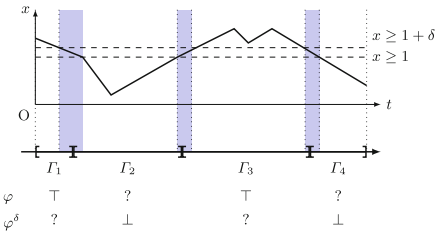

Fig. 3. A δ-stable partition (Def. 4.7) for σ and φ. Here $\varphi^\delta \equiv (x \geq 1+\delta)$. $\top$ and $\bot$ are much like in Fig. 2; the symbol ? indicates that the truth value is not necessarily constant. In some regions (shaded), $\sigma^t \models \varphi$ is true but $\sigma^t \models \varphi^\delta$ is not.

4 Variable-Interval Encoding of STL to MILP

4.1 δ-Stable Partitions

We shall adapt the idea of stable partitioning [7], reviewed in §2.2, to the current MILP setting. A major difference we need to address is that SMT is symbolic while MILP is numerical: most MILP solvers do not distinguish $<$ from $\leq$ and do not accommodate strict inequalities. See e.g. [20].

In order to address this difference, we develop a theory of *δ-stable partitions*. Here is its outline. Firstly, we replace partitions $\ldots, (\gamma_{i-1}, \gamma_i), \{\gamma_i\}, (\gamma_i, \gamma_{i+1}), \ldots$ used in [7] (see also Def. 2.10) with new "partitions" $\ldots, [\gamma_{i-1}, \gamma_i], [\gamma_i, \gamma_{i+1}], \ldots$. The latter can be expressed only using $\leq$; but they have overlaps (at γ_i). The original stability notion (see §2.2) does not fit the new partition notion—it requires "constantly true" or "never true," and prohibits overlaps. Therefore we introduce *δ-stability*; it requires either "constantly true" or "never *robustly* true."

Example 4.1. Let σ be a continuous signal. Suppose that a sequence $\mathcal{P} = (J_i)_{i=1}^{M}$ is a stable partition for σ and an STL formula φ, as illustrated in Fig. 2.

In this paper, since MILP solvers do not accommodate strict inequalities, we are forced to use closed intervals; see $\Gamma_1, \dots, \Gamma_4$ in Fig. 3. Notice that the truth value of the formula φ not constant in Γ_2 or Γ_4. To regain stability, we introduce the *δ-tightening* φ^δ of the formula φ with some $\delta > 0$ (Def. 4.4); here $\varphi^\delta \equiv (x \geq 1 + \delta)$. Then the truth value of φ^δ (instead of φ) is constantly false in Γ_2 and Γ_4, that is, φ is "never δ-robustly true" in Γ_2 and Γ_4.

Definition 4.2 (timed state sequence). A *time sequence* of $[0, T]$ is a sequence $\Gamma = (0 = \gamma_0 < \cdots < \gamma_N = T)$. Such a time sequence induces a "partition of $[0, T]$ with singular overlaps," namely $\Gamma = \big([\gamma_{i-1}, \gamma_i]\big)_{i=1}^{N}$. We identify it with the original time sequence, writing Γ_i for the interval $[\gamma_{i-1}, \gamma_i]$.

Given a time sequence, a *timed state sequence* over V is a sequence $\varsigma = \big((x_0, \gamma_0), \dots, (x_N, \gamma_N)\big)$, where $x_0, \dots, x_N$ in $\mathbb{R}^V$.

In MILP, it is efficient to represent signals as (continuous) *piecewise-linear signals*, so that values within an interval can be deduced by linear interpolation.

Definition 4.3 (piecewise-linear signal). Given a timed state sequence $\varsigma = ((x_0, \gamma_0), \dots, (x_N, \gamma_N))$, the signal $\varsigma^{\mathrm{pwl}} \colon [0, \gamma_N] \to \mathbb{R}^V$ is defined by the following linear interpolation: $\varsigma^{\mathrm{pwl}}(t) \coloneqq (1 - \lambda)x_{i-1} + \lambda x_i$ if $\gamma_{i-1} \leq t \leq \gamma_i$ (where $\lambda = \frac{1}{\gamma_i - \gamma_{i-1}}(t - \gamma_{i-1})$).

In this paper, a *piecewise-linear signal* is a signal of the form ς^{pwl} for some timed state sequence ς. Note that it is continuous everywhere, and is linear everywhere except for only finitely many points. Obviously, ς^{pwl} is finitely variable with respect to any linear predicate p (Def. 2.1).

Definition 4.4 (δ-tightening of linear predicates). Let $\delta > 0$ be a positive real and p be a linear predicate defined by $p(x) = \top \iff c^\top x + b \geq 0$. The *$\delta$-tightening* of p is a linear predicate defined by $p^\delta(x) = \top \iff c^\top x + b \geq \delta$.

Note that p^δ is stronger than p, i.e., $[\![p^\delta]\!] \subsetneq [\![p]\!]$. We further extend the concept of δ-tightening for general STL formulas in NNF (cf. Prop. 2.6). Let p^- be the linear predicate defined by $p^-(x) = \top \iff -c^\top x - b \geq 0$.

Definition 4.5 (δ-tightening of STL formulas in NNF). Let φ be an STL formula in NNF. The *δ-tightening* φ^δ of φ is the STL formula obtained from φ by replacing all occurrences of atomic predicates p (resp. $\neg p$) by p^δ (resp. $(p^-)^\delta$).

The δ-tightening construction is related to robust semantics [32, Def. A.2].

Proposition 4.6. Let σ be a signal, φ be an STL formula in NNF, and $\delta > 0$. Then $\sigma \models \varphi^\delta$ implies $[\![\sigma, \varphi]\!] \geq \delta$. □

Since the closed halfspace $[\![p^-]\!]$ coincides with the closure of the open halfspace $\mathbb{R}^V \setminus [\![p]\!]$, the robust semantics is not affected by the difference between p^- and $\neg p$. For simplicity, in the following, we assume that any STL formula in NNF does not contain negation, i.e., $\neg p$ is replaced by a new atomic proposition p^-.

We are ready to define *δ-stability*.

Definition 4.7 (δ-stability). Let φ be an STL formula over V in NNF, $\sigma \in \mathbf{Signal}_V^T$ be a signal, and $\Gamma = (\gamma_0, \dots, \gamma_N)$ be a time sequence (Def. 4.2) with $\gamma_N = T$. We say Γ is *δ-stable* for σ and φ if, for each $i \in [1, N]$ and each subformula $\psi \in \mathrm{Sub}(\varphi)$, either of the following holds: 1) $\sigma^t \models \psi$ for each $t \in \Gamma_i$, or 2) $\sigma^t \not\models \psi^\delta$ for each $t \in \Gamma_i$.

In the above definition, in each interval Γ_i, a subformula ψ is either 1) always true or 2) never robustly true. The two conditions are not mutually exclusive—both hold if $\sigma^t \models \psi \wedge \neg\psi^\delta$ for all $t \in \Gamma_i$.

The next notion of conservative valuation records which of 1) and 2) is true in each interval. It conservatively approximates the actual truth of φ (Fig. 3).

Definition 4.8 (conservative valuation). Let φ be an STL formula in NNF, and $\Gamma = (\gamma_0, \dots, \gamma_N)$ be a time sequence . A *valuation* of φ in Γ is a function $\Theta : \mathrm{Sub}(\varphi) \times [1, N] \to \mathbb{B}$ that assigns, to each subformula and index of the intervals of Γ, a Boolean truth value. Let σ be a signal with a time horizon $T = \gamma_N$. We say that Θ is a *conservative valuation* of φ in Γ on σ (up to δ) if 1) $\Theta(\psi, i) = \top$ implies that, for each $t \in \Gamma_i$, $\sigma^t \models \psi$ holds; and 2) $\Theta(\psi, \Gamma_i) = \bot$ implies, for each $t \in \Gamma_i$, $\sigma^t \not\models \psi^\delta$.

We simply write $\langle\psi\rangle_i$ for $\Theta(\psi, i)$ when Θ is clear from context.

Suppose there exists a conservative valuation Θ of an STL formula φ in a time sequence Γ on a signal σ up to δ. Then Γ is δ-stable for σ and φ.

We shall argue in §4.2 that, for each piecewise-linear signal σ (Def. 4.3), an STL formula φ, there is a time sequence Γ in which φ is δ-stable on σ. We start with a special case where φ is an atomic proposition p.

Definition 4.9. Let $x, x' \in \mathbb{R}^V$, and p be a linear predicate on V. We say (x, x') is a *δ-crossing pair* with respect to p if $x \in [\![p^\delta]\!]$ and $x' \notin [\![p^\delta]\!]$ (cf. Def. 2.1), or vice versa. A δ-crossing pair is *stationary* if $x \in [\![p]\!]$ and $x' \in [\![p]\!]$.

Lemma 4.10. Let p be a linear predicate and σ be a piecewise-linear signal. There is a time sequence $\Gamma = (\gamma_0, \dots, \gamma_N)$ such that, for any $i \in [1, N]$, 1) σ is linear in the interval $[\gamma_{i-1}, \gamma_i]$, and 2) if $(\sigma(\gamma_{i-1}), \sigma(\gamma_i))$ is a δ-crossing pair, it is stationary. It follows that there is a conservative valuation Θ of p in Γ on σ.

Proof. The lemma argues that, whenever σ enters or leaves $[\![p^\delta]\!]$, it has to do so via $[\![p]\!] \setminus [\![p^\delta]\!]$. See Fig. 4. This can be enforced by adding suitable points to Γ, exploiting continuity of σ (Def. 4.3) and the intermediate value theorem. □

We note another advantage of δ-stable partitions: the number of invervals is roughly halved compared to (original) stable partitions (see Figs. 2 and 3). This advantage may be exploited also in SMT-based approaches [7] for scalability.

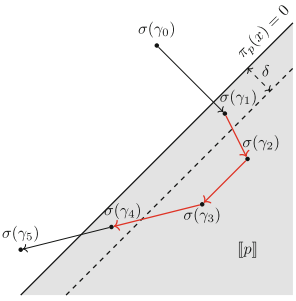

Fig. 4. A conservative valuation Θ of a linear predicate p on σ. The red segments are assigned $\top$ by Θ. (Color figure online)

4.2 Variable-Interval MILP Encoding

Our MILP encoding of STL relies on the constructs in §4.1. For the purpose of trace synthesis for an STL formula φ, our basic strategy is to search for 1) a time sequence $\Gamma = (\gamma_0, \dots, \gamma_N)$ (i.e. a "partition," see Def. 4.2) and 2) a valuation $\Theta : \mathrm{Sub}(\varphi) \times [1, N] \to \mathbb{B}$, such that

- Θ is *consistent* in the sense that the truth values assigned to subformulas comply with the STL semantics (§2.1);
- Θ is *fulfilling* in the sense that it assigns $\top$ to the top-level formula φ in Γ_1 (the first interval); and
- Θ is *realizable* in the sense that there is a piecewise-linear trace $\sigma \in \mathcal{L}(\mathcal{M})$ of $\mathcal{M}$ that *yields* Θ. That is, precisely, Θ must be a conservative valuation of φ in Γ on σ (Def. 4.8).

The entities Γ, Θ we search for are expressed as MILP variables, and the above three conditions are expressed as MILP constraints. We describe these MILP variables and constraints in the rest of the section. The constraints expressing $\sigma \in \mathcal{L}(\mathcal{M})$ require system model encoding and are thus deferred to later sections.

Variables. We use the following MILP variables. Their collection is denoted by $\mathbf{Var}(\varphi, N)$. Here $N \in \mathbb{N}$ is a constant for variability bound (Prob. 3.2).

- Real-valued variables $\{\gamma_0, \dots, \gamma_N\}$ for a time sequence Γ.
- Boolean variables $\{\langle\psi\rangle_i \mid 1 \le i \le N, \psi \in \mathrm{Sub}(\varphi)\}$ for the value $\Theta(\psi, i)$ of a valuation Θ that we search for.
- Real-valued variables $\{x_{i,v} \mid 0 \le i \le N, v \in V\}$ for the values of a piecewise-linear trace $\sigma \in \mathcal{L}(\mathcal{M})$.
- Boolean variables $\{\zeta_i^p, \zeta_i^{\delta,p} \mid 0 \le i \le N, p \in \mathrm{AP}(\varphi)\}$ for the truth values of p and p^δ at time γ_i. These variables are used to detect crossing pairs (Def. 4.9).
- Real-valued variables $\{S_i^\psi \mid 0 \le i \le N, \Box_I \psi \in \mathrm{Sub}(\varphi)\}$. This auxiliary variable records for how long ψ has been true before γ_i.
- Real-valued variables $\{P_i^\psi \mid 0 \le i \le N, \Diamond_I \psi \in \mathrm{Sub}(\varphi)\}$. This auxiliary variable records for how long ψ has been false before γ_i.

By an *assignment* we refer to a function $\boldsymbol{v} \colon \mathbf{Var}(\varphi, N) \to \mathbb{R}$ such that $\boldsymbol{v}(y) \in \{0, 1\}$ for each Boolean variable y. The MILP problem is to find an assignment $\boldsymbol{v}$ that optimizes an objective under given constraints.

Notation 4.11. In what follows, as a notational convention, we simply write a variable y for the value $\boldsymbol{v}(y)$ when the assignment $\boldsymbol{v}$ is clear from context. We further write ς for the timed state sequence composed of the time sequence $\{\gamma_0, \ldots, \gamma_N\}$ and the trace values $\{x_{j,v} \mid 0 \leq j \leq N, v \in V\}$.

Note that, in this paper, we encode the *Boolean* semantics of STL [32, Def. A.1], unlike [28,29] where the *robust* semantics is encoded in a constant-interval manner. The combination of variable-interval encoding and quantitative robust semantics is future work; for example, a quantitative extension of δ-stable partitions (§4.1) seems quite nontrivial.

Shorthands for Propositional Connectives. We use standard shorthands for Boolean connectives in MILP constraints (such as $\neg A, A \wedge B$ where A, B are Boolean variables). See [32, Appendix B] for the formal encodings.

Realizability Constraints: Traces and Atomic Propositions. We need to constrain $\gamma_0, \ldots, \gamma_N$ to be a time sequence (Def. 4.2), using some constant $\varepsilon > 0$ and letting $\cdots \geq \varepsilon$ stand for $\cdots > 0$.

$$\gamma_0 = 0, \quad \gamma_N = T, \quad \gamma_i - \gamma_{i-1} \geq \varepsilon \quad \text{for all } i \in [1, N] \tag{3}$$

For each i and $p \in \mathrm{AP}(\varphi)$ (say p is defined by $c^\top x + b \geq 0$), the variables $\zeta_i^p, \zeta_i^{\delta,p}$ are constrained as follows,

$$\begin{array}{ll} \zeta_i^p = 1 \;\Rightarrow\; c^\top x_i + b \geq 0 & \zeta_i^p = 0 \;\Rightarrow\; c^\top x_i + b \leq -\varepsilon \\ \zeta_i^{\delta,p} = 1 \;\Rightarrow\; c^\top x_i + b \geq \delta & \zeta_i^{\delta,p} = 0 \;\Rightarrow\; c^\top x_i + b \leq \delta - \varepsilon \end{array} \tag{4}$$

Moreover, we impose the following to ensure that Γ is the one in Lem. 4.10:

$$\zeta_i^{\delta,p} = 0 \wedge \zeta_{i+1}^{\delta,p} = 1 \;\Rightarrow\; \zeta_i^p = 1, \quad \zeta_i^{\delta,p} = 1 \wedge \zeta_{i+1}^{\delta,p} = 0 \;\Rightarrow\; \zeta_{i+1}^p = 1 \tag{5}$$

Under constraints (3) to (5), Γ is δ-stable for ς^{pwl} (cf. Def. 4.3) and p, by Lem. 4.10. By the definition of δ-stability, we can now constrain the variable $\langle p \rangle_i$ by $\langle p \rangle_i = \zeta_{i-1}^{p,\delta} \vee \zeta_i^{p,\delta}$ for each i and $p \in \mathrm{AP}(\varphi)$.

Remark 4.12. Note that ε must be chosen to be small enough for the completeness of the encoding (Thm. 4.18). Thereafter we assume that, given a piecewise-linear signal σ and an STL formula φ, ε is small enough to find a δ-stable partition for σ and φ, and we omit ε from the constraints for simplicity.

Consistency Constraints I: Boolean Connectives. We can directly encode conjunction $\bigwedge_{j=1}^m \psi_j$ in STL by recursively applying the shorthand $\wedge$ in [32, Appendix B]: $\langle \bigwedge_{j=1}^m \psi_j \rangle_i = \langle \psi_1 \rangle_i \wedge \langle \bigwedge_{j=2}^m \psi_j \rangle_i$ for each $i \in [1, N]$. It is known that the following alternative encoding avoids auxiliary variables $\langle \bigwedge_{j=k}^m \psi_j \rangle_i$ (where k varies): for each $i \in [1, N]$, $\langle \bigwedge_{j=1}^m \psi_j \rangle_i \geq 1 - m + \sum_{j=1}^m \langle \psi_j \rangle_i$ and $\langle \bigwedge_{j=1}^m \psi_j \rangle_i \leq \langle \psi_j \rangle_i$. An encoding for disjunction is given similarly: $\langle \bigvee_{j=1}^m \psi_j \rangle_i \leq \sum_{j=1}^m \langle \psi_j \rangle_i$, $\langle \bigvee_{j=1}^m \psi_j \rangle_i \geq \langle \psi_j \rangle_i$.

Consistency Constraints II: Unbounded Temporal Modalities. For temporal operators with $I = [0, \infty)$, the following encodings are straightforward.

$$\begin{array}{ll} \langle \psi_1 \mathcal{U}\ \psi_2 \rangle_i = \langle \psi_2 \rangle_i \vee (\langle \psi_1 \mathcal{U}\ \psi_2 \rangle_{i+1} \wedge \langle \psi_1 \rangle_i), & \\ \langle \psi_1 \mathcal{R}\ \psi_2 \rangle_i = \langle \psi_2 \rangle_i \wedge (\langle \psi_1 \mathcal{R}\ \psi_2 \rangle_{i+1} \vee \langle \psi_1 \rangle_i) & \text{for each } i \in [1, N-1], \\ \langle \psi_1 \mathcal{U}\ \psi_2 \rangle_N = \langle \psi_2 \rangle_N, \quad \langle \psi_1 \mathcal{R}\ \psi_2 \rangle_N = \langle \psi_2 \rangle_N & \text{for } i = N. \end{array} \tag{6}$$

The encodings for $\Diamond, \Box$ are special cases.

Consistency Constraints III: Bounded Temporal Modalities. This is the most technically involved part. The challenge here is that the stability for $\Box_{[a,b]}\psi$ is not guaranteed by the stability for ψ (similarly for $\Diamond_{[a,b]}\psi$). Therefore we need additional MILP constraints for the stability for $\Box_{[a,b]}\psi$.

Our encoding is inspired by the results from [26]; ours is simpler thanks to our theory in §4.1 where intervals are all closed.

Recall that we use the variables S_i^ψ, P_i^ψ for this purpose. We focus on $\Box_{[a,b]}\psi$; the encoding of $\Diamond_{[a,b]}\psi$ is similar. The constraints on S_i^ψ are as follows.

$$\begin{array}{ll} S_0^\psi = 0, \qquad \langle \psi \rangle_i = 0 \ \Rightarrow\ S_i^\psi = 0, & \\ \langle \psi \rangle_i = 1 \ \Rightarrow\ S_i^\psi \geq S_{i-1}^\psi + (\gamma_i - \gamma_{i-1}) & \text{for each } i \in [1, N]. \end{array}$$

It follows that, for any non-negative real number $L \in [0, \gamma_j)$, we have $S_j^\psi \leq L$ if and only if there exists $k \in [1, j]$ such that $\langle \psi \rangle_k = 0$ and $\gamma_j - \gamma_k \leq L$.

We proceed to the constraints that describe the relationship between S_i^ψ and the semantics of $\Box_I \psi$. Suppose $\Gamma = (\gamma_0, \dots, \gamma_N)$ is δ-stable for a signal σ and ψ. Let us write $\gamma_{N+1} = \infty$ and $\langle \psi \rangle_{N+1} = \langle \psi \rangle_N$ for simplicity.

We consider consistency for the positive and negative cases separately. For the positive one (i.e. $\langle \Box_{[a,b]}\psi \rangle_i = 1$), the following observation is used.

Proposition 4.13. Let $\varphi \equiv \Box_I \psi$ be an STL formula in NNF, and Θ be a conservative valuation of ψ in $\Gamma = (\gamma_0, \dots, \gamma_N)$ on a signal σ. Given $i \in [1, N]$, suppose $(\Gamma_i + I) \cap (\gamma_{j-1}, \gamma_j] \neq \emptyset$ implies $\langle \psi \rangle_j = 1$ for each $j \in [i, N+1]$. Then $\sigma^t \models \varphi$ holds for any $t \in \Gamma_i$. □

Prop. 4.13 leads to the following MILP constraint:

$$\neg \langle \varphi \rangle_i \vee (\gamma_i + b \leq \gamma_{j-1}) \vee (\gamma_{i-1} + a > \gamma_j) \vee \langle \psi \rangle_j \quad \text{for each } i \in [1, N], j \in [i, N+1].$$

The constraint itself does not follow the MILP format; we can nevertheless express it in MILP using an auxiliary Boolean variable Z_f. Specifically, an inequality $f(x) \geq 0$ in a disjunctive constraint is constrained by $Z_f = 1 \Rightarrow f(x) \geq 0$.

For the consistency in the negative case (i.e. $\langle \Box_{[a,b]}\psi \rangle_i = 0$), the counterpart of Prop. 4.13 also involves S_j^ψ. See below; it leads to an MILP constraint much like Prop. 4.13 does.

Proposition 4.14. Suppose φ, σ, Γ, and Θ are as in Prop. 4.13. For any $t \in \Gamma_i$, $\sigma^t \not\models \varphi^\delta$ holds if the following conditions are satisfied for each $j \in [i, N]$:

$$\begin{cases} S^\psi_j \le b - a & \text{if } \gamma_j \in (\gamma_{i-1} + b, \gamma_i + b), \\ S^\psi_j \le \gamma_j - \gamma_i - a & \text{if } \gamma_i + b \in [\gamma_{j-1}, \gamma_j], \\ S^\psi_N \le \max(0, \gamma_N - \gamma_i - a) & \text{if } \gamma_i + b > \gamma_N. \end{cases} \tag{7}$$

Proof. Let $j_t \in [i, N+1]$ be the unique index such that $t + b \in [\gamma_{j_t - 1}, \gamma_{j_t})$. When $j_t \le N$ and $\gamma_{j_t} < \gamma_i + b$, we have $\gamma_{j_t} \in (\gamma_{i-1} + b, \gamma_i + b)$ and by assumption $S^\psi_{j_t} \le b - a$. There is $k \in [1, j_t]$ such that $\langle \psi \rangle_k = 0$ and $\gamma_k \ge \gamma_{j_t} - b + a > t + a$. We obtain $\Gamma_k \cap (t + [a, b]) \neq \emptyset$ and then $\sigma^t \not\models \varphi^\delta$ holds. The other cases can be checked in a similar manner. □

Remark 4.15. For Prop. 4.13, the converse of the statement does not hold. This is because $\sigma^t \models \psi$ does not guarantee $\langle \psi \rangle_i := \Theta(\psi, i) = 1$ where $t \in \Gamma_i$—we allow $\langle \psi \rangle_i = 0$ when $\sigma^t \models \psi \wedge \neg \psi^\delta$. It is similar for Prop. 4.14. However, this does not affect the completeness of the encoding (Thm. 4.18): while the converse of Prop. 4.13 does not hold for *fixed* Γ, in our workflow we also search for Γ, in which case it is easily shown that the MILP constraints derived from Prop. 4.13 are complete. The same is true for Prop. 4.14.

The remaining cases ($\varphi \equiv \psi_1 \, \mathcal{U}_I \, \psi_2$ and $\varphi \equiv \psi_1 \, \mathcal{R}_I \, \psi_2$) can be reduced to the cases for $\Box_I$ and $\Diamond_I$. It is by the rewriting techniques shown in [12]:

$$\psi_1 \, \mathcal{U}_{[a,b]} \, \psi_2 \quad \sim \quad \Diamond_{[a,b]} \psi_2 \wedge \Box_{[0,a]} (\psi_1 \mathcal{U} \, \psi_2), \tag{8}$$

$$\psi_1 \, \mathcal{R}_{[a,b]} \, \psi_2 \quad \sim \quad \Box_{[a,b]} \psi_2 \vee \Diamond_{[0,a]} (\psi_1 \mathcal{R} \, \psi_2). \tag{9}$$

These equivalences hold in both Boolean and robust semantics.

Correctness of Encoding. Let $\mathbf{Enc_{STL}}(\varphi, N, T, \delta)$ denote the polyhedron defined by the above MILP constraints. It is correct in the following sense; see also the goal we announced in the beginning of §4.2. Its proof is by induction on φ.

Lemma 4.16. Let φ be an STL formula in NNF, $N \in \mathbb{N}$, $T > 0$ and $\delta > 0$. Given an assignment $\boldsymbol{v} \colon \mathbf{Var}(\varphi, N) \to \mathbb{R}$ that lies in $\mathbf{Enc_{STL}}(\varphi, N, T, \delta)$, let Γ, ς be the time sequence and the timed state sequence determined by $\boldsymbol{v}$, and define a valuation Θ by $\Theta(\psi, i) := \langle \psi \rangle_i$ (cf. Def. 4.8). Then Θ is a conservative valuation of φ in Γ on the signal ς^{pwl}. □

We define $\mathbf{Enc}(\varphi, \mathcal{M}, N, T, \delta)$ by the intersection of $\mathbf{Enc_{STL}}(\varphi, N, T, \delta)$, the MILP encoding $\mathbf{Enc_{model}}(\mathcal{M}, N, T)$ of a system model $\mathcal{M}$, and $\langle \varphi \rangle_1 = 1$.

Theorem 4.17 (soundness). Let φ be an STL formula in NNF, $\mathcal{M}$ be a model with a time horizon T, $N \in \mathbb{N}$ and $\delta > 0$. If an assignment $\boldsymbol{v}$ lies in $\mathbf{Enc}(\varphi, \mathcal{M}, N, T, \delta)$, the induced ς^{pwl} has $\varsigma^{\mathrm{pwl}} \in \mathcal{L}(\mathcal{M})$ and $[\![\varsigma^{\mathrm{pwl}}, \varphi]\!] \ge 0$. □

Theorem 4.18 (completeness). Assume the setting of Thm. 4.17. If there is piecewise-linear $\sigma \in \mathcal{L}(\mathcal{M})$ such that $[\![\sigma, \varphi]\!] \ge \delta$, there is an assignment $\boldsymbol{v}$ that lies in $\mathbf{Enc}(\varphi, \mathcal{M}, N, T, \delta)$ for some $N \in \mathbb{N}$. □

5 System Models and Their MILP Encoding

We introduce the MILP encoding $\mathbf{Enc}_{\mathbf{model}}(\mathcal{M}, N, T)$ for some families of models $\mathcal{M}$. We introduce an exact encoding for *rectangular hybrid automata (RHAs)*, and an approximate one for *HAs with closed-form solutions*. We also introduce a refinement of the latter—it is more precise and efficient—restricting to *double integrator dynamics*. The last is useful for automotive examples such as Ex. 1.1.

We defer the discussion of RHAs for the space reason; see [32, Appendix C]. We thus focus on the other two families.

5.1 HAs with Closed-Form Solutions

Here we are interested in hybrid automata (HAs) whose continuous flow dynamics at each control mode has a closed-form solution. The basic idea is simple and it is illustrated in Fig. 5, where the solution $f(t)$ of dynamics (blue) is approximated by a piecewise linear function (red). Such MILP encoding is standard; see e.g. [5].

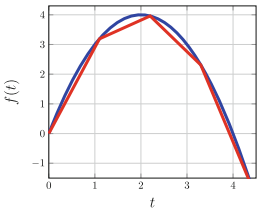

Fig. 5. MILP encoding of $f(t)$

We formalize this intuition. Firstly, to accommodate input signals $\tau \in \mathbf{Signal}_{V'}$ (Def. 2.3), we extend the HA definition so that some variables x^{in} can be designated to be *input variables*. This means that there are no ODEs whose left-hand side is $\dot{x}^{\mathrm{in}}$, and that the variable updates associated with mode transitions never change x^{in}.

Then the above "closed-form solution" assumption on an HA $\mathcal{H}$ is precisely described as follows. Let $\vec{x^{\mathrm{in}}} = (x_1^{\mathrm{in}}, \ldots, x_k^{\mathrm{in}})$ enumerate $\mathcal{H}$'s input variables, and $\vec{x} = (x_1, \ldots, x_l)$ enumerate its other variables. We assume that, for the flow dynamics at each control mode u, there is a *closed-form solution*

$$\vec{x}(t) \;=\; f_u(t, \vec{x^{\mathrm{in}}}, \vec{x_0}) \qquad \text{such that, for each } t_0 \in \mathbb{R}_{\geq 0},\ f_u(t_0, \vec{x^{\mathrm{in}}}, \vec{x}_0) \text{ is a linear function over the variables } \vec{x^{\mathrm{in}}}, \vec{x}_0. \tag{10}$$

Here, the variable t is the elapsed time since the arrival at the current control mode u; the variables $\vec{x^{\mathrm{in}}}$ refer to the input variables (their values are assumed to be constant within the same mode); and the variables $\vec{x_0}$ refer to the *initial* values of $\vec{x}$ on the arrival at u. The assumption holds in many examples, such as polynomial dynamics.

Let us motivate the assumption. A closed-form solution f_u helps precision: in piecewise linear approximation such as in Fig. 5, errors do not accumulate over time; in contrast, if a closed-form solution is not given, our alternative will be numerical integration e.g. by the Euler method, where errors accumulate. The linearity assumption in (10) is there for MILP encoding; see below.

Our approximate MILP encoding poses the closed-form solution assumption and follows the intuition of Fig. 5. Specifically, 1) it fixes a constant $\Delta t \in \mathbb{R}_{\geq 0}$ as

a sampling interval; 2) it obtains a family $\big(\, f_u(k{\cdot}\Delta t, \vec{x^{\text{in}}}, \vec{x}_0)\,\big)_k$ of linear functions over the variables $\vec{x^{\text{in}}}, \vec{x}_0$; and 3) the value of $\vec{x}$ at the elapsed time t is expressed by the linear interpolation

$$\tfrac{(k+1)\Delta t - t}{\Delta t} f_u(k\Delta t, \vec{x^{\text{in}}}, \vec{x}_0) + \tfrac{t - k\Delta t}{\Delta t} f_u\big((k+1)\Delta t, \vec{x^{\text{in}}}, \vec{x}_0\big), \tag{11}$$

where k is such that $k\Delta t \le t \le (k+1)\Delta t$. This encoding of flow dynamics is combined with the HA structure, much like in [32, Appendix C], yielding an approximate MILP encoding of the whole HA.

The above encoding has two sources of numerical errors. One is *linear interpolation*. Errors caused by it are illustrated in Fig. 5 as the vertical margin between blue and red.

The other source is *binary expansion* [18,19], a standard MILP technique for encoding bilinear functions. Indeed, in (11), $t, \vec{x^{\text{in}}}, \vec{x}_0$ are all continuous variables in MILP, and the expression (11) can contain their products. The linearity assumption in (10) has been posed to restrict (11) to bilinear.

5.2 HAs with Double Integrator Dynamics

Our next focus is a special case of the model family of §5.1, where each continuous flow is *double integrator dynamics*. This is important because 1) it gets rid of one of the two error sources in §5.1, namely linear interpolation, by the *trapezoidal rule*, and 2) it can be used for many automotive dynamics models (cf. Ex. 1.1).

The *trapezoidal rule* is a basic technique in numerical integration [6], where $\int_a^b g(t)\,\mathrm{d}t$ is approximated by $(b-a)\frac{g(a)+g(b)}{2}$. For double integrator dynamics, we apply the trapezoidal rule to the velocity v, and it is exact since v's evolution is linear. This allows us to express the position x in the bilinear form $x = t \cdot \frac{v_0+v}{2}$, using the variables t (elapsed time), v_0 (initial velocity), and v (current velocity). Thus we can dispose of the sampling points and their interpolation (11) in §5.1.

We exploit this encoding for our automotive case studies such as Ex. 1.1.

6 Implementation and Experiments

We implemented, in Python, our MILP encodings of the STL semantics (§4) and two model families, namely RHAs [32, Appendix C] and double integrator dynamics (§5.2; multiple modes are not supported since our benchmarks do not need them). The hyperparameter δ in our encoding is fixed at 0.1 for all benchmarks. The resulting MILP constraints are solved by Gurobi Optimizer [20]. This prototype implementation is called *STLts*—STL trace synthesizer.

Our experiments are designed to address the following research questions.

RQ1 Assess the effect of variability bounds N (Prob. 3.2) on the performance.
RQ2 Compare the performance of STLts with optimization-based falsification, and with SMT-based model checking.
RQ3 Assess the performance of STLts for real-world complex scenarios.

RQ4 Assess the performance of STLts in parameter mining (Prob. 3.3).

We used three classes of benchmarks: *rear-end near collision (RNC)*, *navigation (NAV)*, and *disturbance scenarios in ISO 34502 (ISO)*. In each class, we have multiple STL specs, resulting in benchmarks such as RNC1, RNC2, etc.

Rear-End Near Collision (RNC1–3). As discussed in Ex. 1.1, these automotive benchmarks are simplifications of the ISO benchmarks below. The spec `RNC1` is presented in Ex. 1.1. The system model (2) (see also $\mathcal{M}_{\text{RNC}}$ in Ex. 2.4) is double integrator dynamics (§5.2) and is shared by the benchmarks RNC1–3.

The other two specs `RNC2`, `RNC3` are defined as follows, using formulas in (1):

$$\begin{aligned} \texttt{RNC2} &:\equiv \big(\Box(x_{\text{f}} - x_{\text{r}} \geq 0)\big) \wedge \\ &\qquad \Diamond_{[0,9]}\big((\Box_{[0,1]}\texttt{danger}) \wedge (\Box_{[0,1]} a_{\text{r}} \geq 1) \wedge (\Diamond_{[1,5]}\neg\texttt{danger})\big) \\ \texttt{trimming2} &:\equiv (\Diamond\texttt{danger}) \Rightarrow \big((\Box_{[0,1]} a_{\text{r}} \geq 1)\, \mathcal{U}\, \texttt{danger}\big) \\ \texttt{RNC3} &:\equiv \Box(\texttt{dyn\_inv} \wedge \texttt{trimming2}) \wedge \Diamond_{[0,9]}\Box_{[0,1]}\texttt{danger} \end{aligned} \tag{12}$$

Navigation (NAV1–2). Here we use a system model that adapts NAV-2 from [15]. The latter is a standard example of an RHA [32, Appendix C], used e.g. in [10].

Our system model $\mathcal{M}_{\text{NAV}}$ is an RHA that describes the motion of a point robot in a 2×2 grid where each region has a rectangular vector field, with a time horizon $T = 40$. See Fig. 6. We have 4 regions $\ell_1, \ldots, \ell_4$, each associated with rectangular bounds for $\dot{x}, \dot{y}$ and invariants; besides, we set an unsafe region `unsafeR` ($x \in [9, 10]$) and a goal region `goalR` ($x \in [4, 6] \wedge y \in [2, 5]$). The robot starts from an initial position (x_0, y_0) where $x_0 \in [0, 3] \wedge y_0 = 0$.

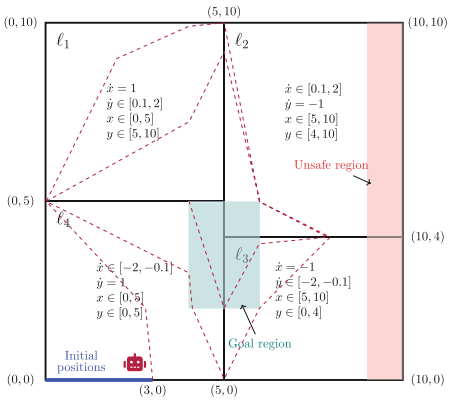

Fig. 6. The RHA $\mathcal{M}_{\text{NAV}}$ for NAV1–2

We consider two specs: `NAV1` $:\equiv \Diamond(\Box_{[0,3]}((x, y) \in \texttt{goalR})) \wedge \Box(x \notin \texttt{unsafeR})$ and `NAV2` $\equiv \Box((x, y) \in \ell_3 \rightarrow \Diamond_{[0,3]}(x, y) \in \ell_4)$. `NAV1` is almost a standard reach-avoid constraint, but it additionally requires the *persistence* to the goal region for three seconds. Such specifications are not accommodated in many control and model checking frameworks specialized in reach-avoid constraints (see e.g. [10]). `NAV2` is a *response specification*—the trigger (being in ℓ_3) must be responded by moving to ℓ_4 within a three-second deadline. Such specs are common in manufacturing; see e.g. [36].

ISO 34502 Disturbance Scenarios for Automated Driving (ISO1, ISO3, ..., ISO8). These benchmarks motivated the current work. As discussed in §1 (see Ex. 1.1), we obtained in [30] complex STL specs as the formalization of the

disturbance scenarios in the ISO 34502 standard, but in our illustration efforts by trace synthesis, we found that existing techniques such as optimization-based falsification struggle.

In our experiments, the system model is similar to $\mathcal{M}_{\text{RNC}}$ (Ex. 1.1 and 2.4), while lateral dynamics is added and the time horizon is 10 time units here. As for specs, we use seven STL specs $\texttt{ISO1}, \texttt{ISO3}, \ldots, \texttt{ISO8}$; these are obtained in [30] as the formalization of the *disturbance scenarios No. 1,3,...,8* in the ISO 34502 standard for automated driving vehicles. See Table 1. Scenario No. 2 was omitted in [30] since it involves three vehicles; we omit Scenarios No. 9–24 since they are the same with No. 1–8 except in the road shape.

Table 1. Disturbance scenarios in the ISO 34502 standard. Table from [21]

Road sector	Subject-vehicle behaviour	Cut in	Cut out	Acceleration	Deceleration (Stop)
Main roadway	Lane keep	No.1	No.2	No.3	No.4
	Lane change	No.5	No.6	No.7	No.8
Merge zone	Lane keep	No.9	No.10	No.11	No.12
	Lane change	No.13	No.14	No.15	No.16
Departure zone	Lane keep	No.17	No.18	No.19	No.20
	Lane change	No.21	No.22	No.23	No.24

Specifically, the specs $\texttt{ISO}i$ follow the common format shown below [30]:

$$\begin{aligned} \texttt{ISO}i &\equiv \texttt{initSafe} \wedge \texttt{disturb}_i, \\ \texttt{disturb}_i &\equiv \texttt{initialCondition}_i \wedge \texttt{behaviourSV}_i \wedge \texttt{behaviourPOV}_i \end{aligned}$$

where SV refers to the subject ("ego") vehicle and POV refers to the principal other vehicle. The component formulas $\texttt{initialCondition}_i$, $\texttt{behaviourSV}_i$ and $\texttt{behaviourPOV}_i$ vary for different scenarios (No. i). Going into their definitions are beyond the scope of this paper; we highlight $\texttt{ISO5}$ as an example to demonstrate the complexity of the specs $\texttt{ISO}i$.

$$\begin{aligned} \texttt{initialCondition}_5 &\equiv \top \qquad\qquad \texttt{behaviourSV}_5 \equiv \texttt{leavingLane}(\texttt{SV}, L) \\ \texttt{behaviourPOV}_5 &\equiv \texttt{cutIn}(\texttt{POV}, \texttt{SV}) \\ \texttt{leavingLane}(a, L) &\equiv \texttt{atLane}(a, L) \wedge \Diamond(\neg \texttt{atLane}(a, L)) \\ \texttt{cutIn}(\texttt{POV}, \texttt{SV}, L) &\equiv \neg \texttt{sameLane}(\texttt{POV}, \texttt{SV}, L) \wedge \Diamond\big(\texttt{danger}(\texttt{SV}, \texttt{POV}) \\ &\quad \wedge \Diamond_{[0,\texttt{minDanger}]}(\texttt{sameLane}(\texttt{SV}, \texttt{POV}, L) \wedge \texttt{aheadOf}(\texttt{SV}, \texttt{POV}))\big) \\ \texttt{danger}(\texttt{SV}, \texttt{POV}) &\equiv \Box_{[0,\texttt{minDanger}]} \texttt{rssViolation}(\texttt{SV}, \texttt{POV}) \end{aligned} \tag{13}$$

The formulas not defined here are suitably defined atomic propositions.

Experiment Settings. Our implementation *STLts* is compared with the following tools: 1) a widely used optimization-based falsification tool *Breach* [11]; 2) another falsification tool *ForeSee* [1,37] that emphasizes optimized treatment of Boolean connectives in STL; 3) an MILP-based STL optimal control tool *bluSTL* [14]; and 4) *STLmc*, an SMT-based bounded STL model checker [34].

The experiments were conducted on an Amazon EC2 c4.4xlarge instance (2.9 GHz Intel Xeon E502666 v3, 30.0GB RAM) running Ubuntu Server 20.04.

RQ1: the Effect of the Variability Bound N. There is an obvious trade-off about the choice of a variability bound N (Prob. 3.2): bigger N means the search is more extensive, but it incurs greater computational cost.

This tendency is confirmed in our experiments; the result for the `ISO6` benchmark is in Fig. 7 for illustration. Here, synthesis was successful for $N = 4$ for the first time.

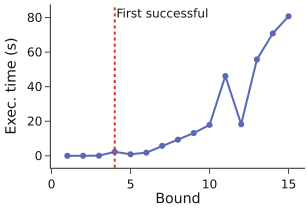

Fig. 7. Execution time of STLts for different var. bd. N, on `ISO6`

We also observe in the figure that computational cost is low when trace synthesis is unsuccessful. This suggests the following strategy: we start with small N and increment it if trace synthesis is unsuccessful. We might waste time by trying too small N's; but the wasted time should be small.

Experimental Results, Overview. Our experimental results are in summarized in Table 2, where the best performers are highlighted by color.

We explain the missing entries. In (∗), the tool is not applicable due to the nondeterminism of the benchmark. In (†), we did not conduct experiments since the performance comparison with STLts is already clear with simpler `RNC` benchmarks. In (‡), bluSTL does not support multiple control modes. (¶) is because bluSTL (at least its implementation available to us) does not support the until $\mathcal{U}$ modality.

Table 2. Experimental results for trace synthesis, showing execution time (seconds). (N) for STLts is the first successful bound. Timeout (t/o) is 600 s.

	STLts		Breach	ForeSee	bluSTL	STLmc
`RNC1`	0.1	(3)	59.4	546.8	(¶)	t/o
`RNC2`	0.3	(4)	9.3	104.3	14.3	t/o
`RNC3`	0.1	(3)	81.3	197.4	(¶)	t/o
`NAV1`	32.5	(17)	(∗)	(∗)	(‡)	16.5
`NAV2`	2.1	(11)				10.0
`ISO1`	0.4	(3)	8.9	t/o		
`ISO3`	0.2	(2)	t/o	t/o		
`ISO4`	0.4	(2)	t/o	t/o		
`ISO5`	9.9	(4)	31.2	435.8	(†)	(†)
`ISO6`	2.4	(4)	t/o	58.9		
`ISO7`	0.6	(3)	33.6	187.2		
`ISO8`	1.5	(3)	38.8	t/o		

Overall, our STLts is clearly the best performer in all benchmarks but one. The other tools time out, or takes tens of seconds. For our motivation of *illustrating* STL specs by trace synthesis in close interaction with users, tens of seconds is prohibitively long. The results adequately demonstrate satisfactory performance of our algorithm, in trace synthesis for complex STL specs.

RQ2: Comparison with Other Approaches. A summary of comparison is in Table 3. The comparison with optimization-based falsification tools is as we expected—their struggle with complex specs motivated this work (§1). Boolean connectives in STL specs have been found problematic in falsification: this is called the *scale problem* [36,37]. The results in Table 2 show that our benchmark specs are even beyond the capability of ForeSee, a tool that incorporates Monte Carlo tree search to specifically handle the scale problem. After all, one can say

Table 3. Comparison of our approach (STLts) with baselines (Breach, ForeSee, bluSTL, STLmc). Highlited cells represent positive features.

Feature	**STLts**	**Breach/ForeSee**	**bluSTL**	**STLmc**
Trace synthesis for analyzing specs	● Successful in all benchmarks with large STL formulas	◑ Good for falsifying models but not good with large STL formulas	- Timeout in most of benchmarks	◔ Timeout except for linear dynamics
Model checking	◑ Complete up to N and δ	-	◑ Control synthesis with guarantee	● Complete up to N
Parameter mining	◑ By MILP	-	◑ By MILP	◔ By binary search
Continuous STL semantics	● Variable-interval encoding	- Discretized	- Discretized	● Variable-interval encoding
Accommodated class of nonlinear dynamics	MILP-encodable, can be nondeterministic	Black-box, deterministic	MILP-encodable, can be nondeterministic	SMT-encodable, can be nondeterministic

● = full support; ◑ = partial support; ◔ = very limited support; - = not supported.

that falsification tools are aimed at complex *models*, while our STLts is aimed at complex *specs*.

STLmc has a similar ("dual") scope and utilizes a similar technique (stable partitioning) to our STLts; the main difference is that STLmc is SMT-based while STLts is MILP-based. Therefore STLts accommodates a smaller class of models, but it can be faster on them exploiting numeric optimization. Table 2 suggests the advantage of STLts for common STL specs in manufacturing.

RQ3: Performance in Real-World Scenarios. For this RQ, we refer to STLts's performance on the ISO benchmarks. Illustrating the specs `ISO`i by trace synthesis is a real-world problem about safety standards for automated driving (§1), and Table 2 shows that STLts has sufficient performance and scalability to handle complex specs there (see (13)).

RQ4: Performance in Parameter Mining. We conducted parameter mining experiments with the `ISO8` benchmark. Its specification has a subformula `fasterThan`(SV, POV, p) that requires that SV's velocity is bigger than POV's by at least a parameter p. We used STLts to solve Prob. 3.3, that is, to find the maximum p for which a satisfying trace exists.

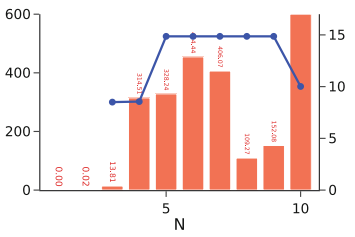

Fig. 8. STLts for parameter synthesis. Red is execution time (axis left, seconds); blue is the maximum p (axis right). (Color figure online)

Figure 8 shows the results with varying variability bound N. Parameter mining is generally more expensive than trace synthesis. This is because the former has a nontrivial objective function (namely p in this example), while

the latter does not (it is thus a constraint satisfaction problem). We observe the optimization with $N \geq 10$ resulted in a timeout. The tendency, much like in trace synthesis, is that the result (max p) improves but execution time gets larger as N becomes bigger (there are some exceptions such as $N = 8, 9$ though). Taking the same strategy as above (incrementing N), it takes roughly 10 min to obtain a largely converged value (~ 14.9 for the maximum p). Overall, we believe this is a realistic performance for practical usage.

References

1. ForeSee falsification solver (2021). https://github.com/choshina/ForeSee
2. Akazaki, T., Hasuo, I.: Time robustness in MTL and expressivity in hybrid system falsification. In: Kroening, D., Păsăreanu, C.S. (eds.) CAV 2015. LNCS, vol. 9207, pp. 356–374. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-21668-3_21
3. Alur, R., Feder, T., Henzinger, T.A.: The benefits of relaxing punctuality. J. ACM **43**(1), 116–146 (1996). https://doi.org/10.1145/227595.227602
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. Asghari, M., Fathollahi-Fard, A.M., Mirzapour Al-e hashem, S.M.J., Dulebenets, M.A.: Transformation and linearization techniques in optimization: a state-of-the-art survey. Mathematics **10**(2), 283 (2022). https://doi.org/10.3390/math10020283
6. Atkinson, K.E.: An Introduction to Numerical Analysis. Wiley, New York, second edn. (1989). http://www.worldcat.org/isbn/0471500232
7. Bae, K., Lee, J.: Bounded model checking of signal temporal logic properties using syntactic separation. Proc. ACM Program. Lang. **3**(POPL), 51:1–51:30 (2019). https://doi.org/10.1145/3290364
8. Bartocci, E., et al.: Specification-based monitoring of cyber-physical systems: a survey on theory, tools and applications. In: Bartocci, E., Falcone, Y. (eds.) Lectures on Runtime Verification. LNCS, vol. 10457, pp. 135–175. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-75632-5_5
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. Bu, L., Frehse, G., Kundu, A., Ray, R., Shi, Y., Zaffanella, E.: Arch-comp22 category report: Hybrid systems with piecewise constant dynamics and bounded model checking. In: Frehse, G., Althoff, M., Schoitsch, E., Guiochet, J. (eds.) Proceedings of 9th International Workshop on Applied Verification of Continuous and Hybrid Systems (ARCH22). EPiC Series in Computing, vol. 90, pp. 44–57. EasyChair (2022). https://doi.org/10.29007/lnzf
11. Donzé, A.: Breach, a toolbox for verification and parameter synthesis of hybrid systems. In: Touili, T., Cook, B., Jackson, P. (eds.) CAV 2010. LNCS, vol. 6174, pp. 167–170. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-14295-6_17
12. Donzé, A., Ferrère, T., Maler, O.: Efficient robust monitoring for STL. In: Sharygina, N., Veith, H. (eds.) CAV 2013. LNCS, vol. 8044, pp. 264–279. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-39799-8_19

13. Donzé, A., Maler, O.: Robust satisfaction of temporal logic over real-valued signals. In: Chatterjee, K., Henzinger, T.A. (eds.) FORMATS 2010. LNCS, vol. 6246, pp. 92–106. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-15297-9_9
14. Donzé, A., Raman, V.: BluSTL: controller synthesis from signal temporal logic specifications. In: ARCH14-15. 1st and 2nd International Workshop on Applied veRification for Continuous and Hybrid Systems, pp. 160–150. https://doi.org/10.29007/g39q
15. Duggirala, P.S., Mitra, S.: Abstraction refinement for stability. In: 2011 IEEE/ACM Second International Conference on Cyber-Physical Systems, pp. 22–31. IEEE (2011). https://doi.org/10.1109/ICCPS.2011.24
16. Ernst, G., et al.: ARCH-COMP 2021 category report: falsification with validation of results. In: Frehse, G., Althoff, M. (eds.) 8th International Workshop on Applied Verification of Continuous and Hybrid Systems (ARCH21), Brussels, Belgium, July 9, 2021. EPiC Series in Computing, vol. 80, pp. 133–152. EasyChair (2021). https://doi.org/10.29007/XWL1
17. Fainekos, G.E., Pappas, G.J.: Robustness of temporal logic specifications for continuous-time signals. Theoret. Comput. Sci. **410**(42), 4262–4291 (2009). https://doi.org/10.1016/j.tcs.2009.06.021
18. Glover, F.: Improved linear integer programming formulations of nonlinear integer problems. Manag. Sci. **22**, 455–460 (1975). https://doi.org/10.1287/mnsc.22.4.455
19. Gupte, A., Ahmed, S., Cheon, M.S., Dey, S.: Solving mixed integer bilinear problems using MILP formulations. SIAM J. Optim. **23**(2), 721–744 (2013). https://doi.org/10.1137/110836183
20. Gurobi Optimization, LLC: Gurobi Optimizer Reference Manual (2023). https://www.gurobi.com
21. Road vehicles - Test scenarios for automated driving systems - Scenario based safety evaluation framework. Standard, International Organization for Standardization, Geneva, CH (2022)
22. Kurtz, V., Lin, H.: A more scalable mixed-integer encoding for metric temporal logic. IEEE Control. Syst. Lett. **6**, 1718–1723 (2022). https://doi.org/10.1109/LCSYS.2021.3132839
23. Lee, J., Yu, G., Bae, K.: Efficient SMT-based model checking for signal temporal logic. In: 2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE), pp. 343–354 (2021). https://doi.org/10.1109/ASE51524.2021.9678719
24. Maler, O., Nickovic, D.: Monitoring temporal properties of continuous signals. In: Lakhnech, Y., Yovine, S. (eds.) FORMATS/FTRTFT -2004. LNCS, vol. 3253, pp. 152–166. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-30206-3_12
25. Pedrielli, G., et al.: Part-X: A family of stochastic algorithms for search-based test generation with probabilistic guarantees. IEEE Trans. Autom. Sci. Eng. 1–22 (2023). https://doi.org/10.1109/TASE.2023.3297984
26. Prabhakar, P., Lal, R., Kapinski, J.: Automatic trace generation for signal temporal logic. In: 2018 IEEE Real-Time Systems Symposium (RTSS), pp. 208–217. IEEE, Nashville, TN (2018). https://doi.org/10.1109/RTSS.2018.00038
27. Rabinovich, A.M.: On the decidability of continuous time specification formalisms. J. Log. Comput. **8**(5), 669–678 (1998). https://doi.org/10.1093/logcom/8.5.669
28. Raman, V., Donzé, A., Maasoumy, M., Murray, R.M., Sangiovanni-Vincentelli, A.L., Seshia, S.A.: Model predictive control with signal temporal logic specifications. In: 53rd IEEE Conference on Decision and Control, CDC 2014, Los Angeles,

CA, USA, December 15-17, 2014, pp. 81–87. IEEE (2014). https://doi.org/10.1109/CDC.2014.7039363
29. Raman, V., Donzé, A., Sadigh, D., Murray, R.M., Seshia, S.A.: Reactive synthesis from signal temporal logic specifications. In: Proceedings of the 18th International Conference on Hybrid Systems: Computation and Control, pp. 239–248. ACM, Seattle Washington (2015). https://doi.org/10.1145/2728606.2728628
30. Reimann, J., et al.: Temporal logic formalisation of ISO 34502 critical scenarios: modular construction with the RSS safety distance. In: Proceedings of the 39th ACM/SIGAPP Symposium on Applied Computing (SAC 2024) to appear (2024). arXiv:2403.18764
31. Roehm, H., Heinz, T., Mayer, E.C.: STLInspector: STL Validation with Guarantees. In: Majumdar, R., Kunčak, V. (eds.) CAV 2017. LNCS, vol. 10426, pp. 225–232. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-63387-9_11
32. Sato, S., An, J., Zhang, Z., Hasuo, I.: Optimization-based model checking and trace synthesis for complex STL specifications (extended version) (2024). available on arXiv
33. Souyris, J., Wiels, V., Delmas, D., Delseny, H.: Formal verification of avionics software products. In: Cavalcanti, A., Dams, D.R. (eds.) FM 2009. LNCS, vol. 5850, pp. 532–546. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-05089-3_34
34. Yu, G., Lee, J., Bae, K.: Stlmc: Robust STL model checking of hybrid systems using SMT. In: Shoham, S., Vizel, Y. (eds.) Computer Aided Verification - 34th International Conference, CAV 2022, Haifa, Israel, August 7-10, 2022, Proceedings, Part I. LNCS, vol. 13371, pp. 524–537. Springer (2022). https://doi.org/10.1007/978-3-031-13185-1_26
35. Zhang, Z., Arcaini, P.: Gaussian process-based confidence estimation for hybrid system falsification. In: Huisman, M., Păsăreanu, C., Zhan, N. (eds.) FM 2021. LNCS, vol. 13047, pp. 330–348. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-90870-6_18
36. Zhang, Z., Hasuo, I., Arcaini, P.: Multi-armed bandits for Boolean connectives in hybrid system falsification. In: Dillig, I., Tasiran, S. (eds.) CAV 2019. LNCS, vol. 11561, pp. 401–420. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-25540-4_23
37. Zhang, Z., Lyu, D., Arcaini, P., Ma, L., Hasuo, I., Zhao, J.: Effective hybrid system falsification using monte carlo tree search guided by QB-robustness. In: Silva, A., Leino, K.R.M. (eds.) CAV 2021. LNCS, vol. 12759, pp. 595–618. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-81685-8_29