STLMC: Robust STL Model Checking of Hybrid Systems Using SMT

Geunyeol Yu, Jia Lee, and Kyungmin Bae(✉)

Pohang University of Science and Technology,
Pohang, Korea
kmbae@postech.ac.kr

Abstract. We present the STLMC model checker for signal temporal logic (STL) properties of hybrid systems. The STLMC tool can perform STL model checking up to a robustness threshold for a wide range of hybrid systems. Our tool utilizes the refutation-complete SMT-based bounded model checking algorithm by reducing the robust STL model checking problem into Boolean STL model checking. If STLMC does not find a counterexample, the system is guaranteed to be correct up to the given bounds and robustness threshold. We demonstrate the effectiveness of STLMC on a number of hybrid system benchmarks.

1 Introduction

Signal temporal logic (STL) [31] has emerged as a popular property specification formalism for hybrid systems. STL formulas describe linear-time properties of continuous real-valued signals. Because hybrid systems exhibit both discrete and continuous behaviors, STL provides a convenient and expressive way to specify important requirements of hybrid systems. STL has a vast range of applications on hybrid systems, including automotive systems [26], robotics [24,40], medical systems [36], IoT [7], smart cities [30], etc.

Due to the infinite-state nature of hybrid systems with continuous dynamics, most techniques and tools for analyzing STL properties focus on monitoring and falsification. These techniques analyze concrete samples of signals obtained by simulating hybrid automata to monitor the system's behavior [13,15,32] or find counterexamples [1,37,43], often combined with stochastic optimization. To this end, STL monitoring and falsification use quantitative semantics that defines the *robustness degree* to indicate how well the formula is satisfied. However, these methods cannot be used to guarantee correctness.

Recently, several STL model checking techniques have been proposed for hybrid systems [3,29,35]. In particular, the SMT-based bounded model checking algorithms [3,29] are refutation-complete, i.e., they can guarantee correctness up to given bounds. However, these techniques are based on the Boolean semantics of STL instead of quantitative semantics. This is a limitation for hybrid systems as small perturbations of signals can cause the system to violate the properties *verified* by Boolean STL model checking. Moreover, there exists no tool with a convenient user interface implementing STL model checking techniques.

S. Shoham and Y. Vizel (Eds.): CAV 2022, LNCS 13371, pp. 524–537, 2022.
https://doi.org/10.1007/978-3-031-13185-1_26

This paper presents the STLMC tool for robust STL model checking of hybrid systems. Our tool can verify that, up to given bounds, the robustness degree of an STL formula φ is greater than a *robustness threshold* $\epsilon > 0$ for all possible behaviors of the system. We reduce the robust STL model checking problem to Boolean STL model checking using ϵ-*strengthening* (perturbing the problem by ϵ to make it harder to be true), first proposed in [21] for first-order logic and extended to STL. We then apply the refutation-complete bounded model checking algorithm [3,29] to build the SMT encoding of the resulting Boolean STL model checking problem, which can be solved using SMT solvers.

Apart from the robust STL model checking method, STLMC also implements several techniques to improve the usability and scalability of the tool:

- STLMC implements a generic interface to connect with various SMT solvers, such as Z3 [12], Yices2 [17], and dReal [22]. Since dReal can (approximately) deal with nonlinear ordinary differential equations (ODEs), STLMC can also support hybrid systems with nonlinear ODE dynamics.
- STLMC implements parallelized two-step SMT solving to improve scalability. Instead of directly solving the complex encoding with ODEs, we first obtain a *discrete abstraction* without ODEs and find satisfying scenarios. We then check the *discrete refinements* of such scenarios using dReal in parallel.
- STLMC provides a visualization command to draw counterexample signals and robustness degrees. Such graphs intuitively explain why the robustness degree of the formula is greater than a given threshold, and thus greatly help in analyzing counterexamples and debugging hybrid systems.

We demonstrate the effectiveness of the STLMC tool on a number of hybrid system benchmarks— including linear, polynomial, and ODE dynamics— and nontrivial STL properties. The tool is available at https://stlmc.github.io.

2 Background: Robust STL Model Checking

Hybrid Automata. Hybrid systems are often formalized as *hybrid automata* [25], defined as a tuple $H = (Q, X, \mathit{init}, \mathit{inv}, \mathit{jump}, \mathit{flow})$. A set of modes Q specifies discrete states. A set of real-valued variables $X = \{x_1, ..., x_l\}$ gives continuous states. A pair $\langle q, \vec{v} \rangle$ of mode $q \in Q$ and vector $\vec{v} \in \mathbb{R}^l$ constitutes a state of H. An initial condition $\mathit{init}(q, \vec{v})$ defines a set of initial states. An invariant condition $\mathit{inv}(q, \vec{v})$ defines a set of valid states. A jump condition $\mathit{jump}(q, \vec{v}, q', \vec{v'})$ defines a discrete transition from $\langle q, \vec{v} \rangle$ to $\langle q', \vec{v'} \rangle$. A flow condition $\mathit{flow}(q, \vec{v}, \vec{v}_t, t)$ defines a continuous evolution of X's values from $\vec{v}$ to $\vec{v}_t$ over time t in mode q.

A *signal* σ represents a continuous execution of a hybrid automaton H, given by a function $[0, \tau) \to Q \times \mathbb{R}^l$ with a time bound $\tau > 0$. A signal σ is called a *trajectory* of a hybrid automaton H, written $\sigma \in H$, if σ describes a valid behavior of H: formally, there exists a sequence of times $0 = t_0 < t_1 < ... < \tau$ such that: (i) $\sigma(t_0)$ is an initial state by *init*; (ii) for $i \geq 1$, H's state evolves from $\sigma(t_i)$ according to *flow*, while satisfying *inv*, for each time interval $[t_{i-1}, t_i)$; and (iii) for $i \geq 1$, a discrete transition occurs by *jump* at each time point t_i.

Signal Temporal Logic. Signal temporal logic (STL) is widely used to specify properties of hybrid systems [31]. The syntax of STL is defined by:

$$\varphi ::= p \mid \neg\varphi \mid \varphi \wedge \varphi \mid \varphi \, \mathbf{U}_I \, \varphi$$

where p denotes state propositions, and $I \subseteq \mathbb{R}_{\geq 0}$ is any interval of nonnegative real numbers. Examples of state propositions include relational expressions of the form $f(\vec{x}) \geq 0$ over variables X with a real-valued function $f : \mathbb{R}^l \to \mathbb{R}$. Other common Boolean and temporal operators can be derived by equivalences: e.g., $\varphi \vee \varphi' \equiv \neg(\neg\varphi \wedge \neg\varphi')$, $\Diamond_I \, \varphi \equiv \top \, \mathbf{U}_I \, \varphi$, $\Box_I \, \varphi \equiv \neg \Diamond_I \neg\varphi$, etc.

We consider a quantitative semantics of STL based on *robustness degrees* [15]. The semantics of a state proposition p is defined as a function $p : Q \times \mathbb{R}^l \to \overline{\mathbb{R}}$ that assigns to a state the degree to which p is true, where $\overline{\mathbb{R}} = \mathbb{R} \cup \{-\infty, \infty\}$. Specifically, the robustness degree of a state proposition $f(\vec{x}) \geq 0$ is the value of $f(\vec{x})$. E.g., the robustness degree of $x \geq 4$ is the value of $x - 4$ at a given state. The robustness degree of an STL formula can be defined as follows [15], where a time bound τ of a signal is explicitly taken into account.[1]

Definition 1. *Given an STL formula φ, a signal $\sigma : [0, \tau) \to \mathbb{R}^l$, and a time $t \in [0, \tau)$, the robustness degree $\rho_\tau(\varphi, \sigma, t) \in \overline{\mathbb{R}}$ is defined inductively by:*[2]

$$\begin{aligned}
\rho_\tau(p, \sigma, t) &= p(\sigma(t)) \\
\rho_\tau(\neg\varphi, \sigma, t) &= -\rho_\tau(\varphi, \sigma, t) \\
\rho_\tau(\varphi_1 \wedge \varphi_2, \sigma, t) &= \min(\rho_\tau(\varphi_1, \sigma, t), \rho_\tau(\varphi_2, \sigma, t)) \\
\rho_\tau(\varphi_1 \, \mathbf{U}_I \, \varphi_2, \sigma, t) &= \sup_{t' \in (t+I) \cap [0,\tau)} \min(\rho_\tau(\varphi_2, \sigma, t'), \inf_{t'' \in [t,t']} \rho_\tau(\varphi_1, \sigma, t''))
\end{aligned}$$

The robust STL model checking problem is to determine if the robustness degree of an STL formula φ is always greater than a given robustness threshold $\epsilon > 0$ for all possible trajectories of a hybrid automaton H.

Definition 2 (Robust STL Model Checking). *For a time bound $\tau > 0$, an STL formula φ is satisfied at time $t \in [0, \tau)$ on a hybrid automaton H with respect to a robustness threshold $\epsilon > 0$ iff for every trajectory $\sigma \in H$, $\rho_\tau(\varphi, \sigma, t) > \epsilon$.*

A Running Example. Consider two rooms interconnected by an open door. The temperature x_i of each room, $i = 0, 1$, changes depending on the heater's mode $q_i \in \{\mathsf{On}, \mathsf{Off}\}$ and the temperature of the other room. The continuous dynamics of x_i can be specified as the following ODEs, where K_i, h_i, c_i, d_i are determined by the size of the room, the heater's power, and the size of the door [2,19,25]:

$$\dot{x_i} = \begin{cases} K_i(h_i - (c_i x_i - d_i x_{1-i})) & (\mathsf{On}) \\ -K_i(c_i x_i - d_i x_{1-i}) & (\mathsf{Off}), \end{cases}$$

[1] C.f., in the Boolean semantics of STL [29,31], the satisfaction of an STL formula is defined as a Boolean value (i.e., true or false).

[2] The Minkowski sum of intervals I and J is denoted by $I + J$. For a singular interval, $\{t\} + I$ is written as $t + I$. We write $\sup_{a \in A} g(a)$ and $\inf_{a \in A} g(a)$ to denote the least upper bound and the greatest lower bound of the set $\{g(a) \mid a \in A\}$, respectively.

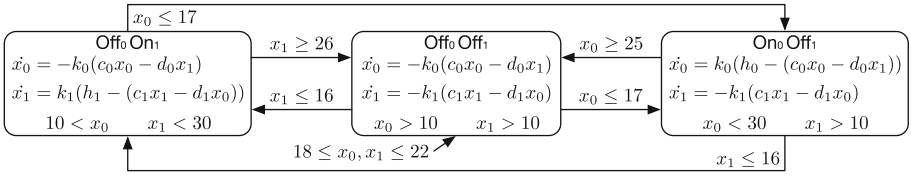

Fig. 1. A hybrid automaton for the networked thermostats.

Figure 1 shows a hybrid automaton of our networked thermostat controllers. Initially, both heaters are off and the temperatures are between 18 and 22. The jumps between modes then define a control logic to keep the temperatures within a certain range using only one heater. We are interested in robust model checking of nontrivial STL properties, such as:

ϕ_1**:** $\Diamond_{[0,15]}(x_0 \geq 14\ \mathbf{U}_{[0,\infty)}\ x_1 \leq 19)$: at some moment in the first 15 s, x_1 is less than or equal to 19; until then, x_0 is greater than or equal to 14.

ϕ_2**:** $\Box_{[2,4]}(x_0 - x_1 \geq 4 \rightarrow \Diamond_{[3,10]}\, x_0 - x_1 \leq -3)$: between 2 and 4 s, whenever $x_0 - x_1 \geq 4$, $x_0 - x_1 \leq -3$ holds within 10 s after 3 s.

3 The STLMC Model Checker

The STLMC tool can model check STL properties of hybrid automata, given three parameters $\epsilon > 0$ (robustness threshold), $\tau > 0$ (time bound), and $N \in \mathbb{N}$ (discrete bound). STLMC provides an expressive input format to easily specify a wide range of hybrid automata. STLMC also provides a visualization command to give an intuitive description of counterexamples.

3.1 Input Format

The input format of STLMC, inspired by dReach [28], consists of five sections: variable declarations, mode definitions, initial conditions, state propositions, and STL properties. Mode and continuous variables define discrete and continuous states of hybrid automata. Mode definitions specify flow, jump, and invariant conditions. STL formulas can also include user-defined state propositions.

Figure 2 shows the input model of the hybrid automaton described in the running example above. Constants are introduced with the `const` keyword. Two mode variables `on0` and `on1` denote the heaters' modes. Continuous variables `x0` and `x1` are declared with domain intervals. There are three "mode blocks" that specify the three modes in Fig. 1 and their invariant, flow, and jump conditions.

In mode blocks, a `mode` component includes a set of logic formulas over mode variables. An `inv` component contains a set of logic formulas over continuous variables. A `flow` component can include ODEs over continuous variables. A `jump` component contains a set of jump conditions of the form *guard* => *reset*, where *guard* and *reset* are logic formulas over mode and continuous variables, and "primed" variables denote states after the jump has occurred.

```
const k0 = 0.015;          const k1 = 0.045;
const h0 = 100;            const h1 = 200;
const c0 = 0.98;           const c1 = 0.97;
const d0 = 0.01;           const d1 = 0.03;

int on0;                   int on1;
[10, 35] x0;               [10, 35] x1;

{ mode: on0 = 0;           on1 = 1;
  inv:  10 < x0; x1 < 30;
  flow: d/dt[x0] = - k0 * (c0 * x0 - d0 * x1);
        d/dt[x1] = k1 * (h1 - (c1 * x1 - d1 * x0));
  jump: x0 <= 17 => (and (on0' = 1) (on1' = 0)
                         (x0' = x0) (x1' = x1));
        x1 >= 26 => (and (on1' = 0) (on0' = on0)
                         (x0' = x0) (x1' = x1));
}
{ mode: on0 = 1;           on1 = 0;
  inv:  x0 < 30; x1 > 10;
  flow: d/dt[x0] = k0 * (h0 - (c0 * x0 - d0 * x1));
        d/dt[x1] = - k1 * (c1 * x1 - d1 * x0);
  jump: x1 <= 16 => (and (on0' = 0) (on1' = 1)
                         (x0' = x0) (x1' = x1));
    x0 >= 25 => (and (on0' = 0) (on1' = on1)
                     (x0' = x0) (x1' = x1));
}
{ mode: on0 = 0;           on1 = 0;
  inv:  x0 > 10; x1 > 10;
  flow: d/dt[x0] = - k0 * (c0 * x0 - d0 * x1);
        d/dt[x1] = - k1 * (c1 * x1 - d1 * x0);
  jump:
    x0 <= 17 => (and (on0' = 1) (on1' = on1)
                     (x0' = x0) (x1' = x1));
    x1 <= 16 => (and (on1' = 1) (on0' = on0)
                     (x0' = x0) (x1' = x1));
}

init: on0 = 0;  18 <= x0;  x0 <= 22;
      on1 = 0;  18 <= x1;  x1 <= 22;

proposition:
  [p1]: x0 - x1 >= 4;     [p2]: x0 - x1 <= -3;

goal:
  [f1]: <>[0,15](x0 >= 14 U[0, inf) x1 <= 19);
  [f2]: [][2, 4](p1 -> <>[3, 10] p2);
```

Fig. 2. An input model example

STL properties are declared in the `goal` section, and "named" propositions are declared in the `proposition` section. State propositions are arithmetic and relational expressions over mode and continuous variables. For example, in Fig. 2, the STL formula `f1` contains two state propositions $x_0 \geq 14$ and $x_1 \leq 19$, and the formula `f2` contains the user-defined propositions `p1` and `p2`.

3.2 Command Line Options

STLmc provides a command-line interface with various options in Table 1. The options `-two-step` and `-parallel` enable the two-step solving optimization in Sect. 4.3. STLmc supports three SMT solvers to choose from based on continuous dynamics: Z3 [12] and Yices2 [17] can deal with linear and polynomial dynamics (solutions of ODEs are linear functions or polynomials), and dReal [22] can approximately deal with ODE dynamics with Lipschitz-continuous ODEs.

A discrete bound N limits the number of mode changes and *variable points* at which the truth value of some STL subformula changes. This is a distinctive parameter of STL model checking that cannot typically be derived from a time bound τ or the maximal number of jumps (say, m). E.g., for any positive natural number $n \in \mathbb{N}$, consider the function $y(t) = \sin(\frac{\pi}{\tau} \cdot n \cdot t)$; the state proposition $y > 0$ has $n - 1$ variable points even if there is no mode change ($m = 0$).[3]

For the input model in Fig. 2, the following command found a counterexample of the formula `f2` at bound 2 with respect to $\epsilon = 2$ in 15 s using dReal:

[3] This example also hints that STL model checking can be arbitrary complex even for one mode; τ and m cannot limit such model checking computation, whereas N can limit the computation involving *both* discrete and continuous behaviors.

Table 1. Some command line options for STLMC.

Option	Explanation	Option	Explanation
`-bound`⟨N⟩	a discrete bound	`-two-step`	enable two-step solving
`-time-bound`⟨τ⟩	a time bound	`-parallel`	parallel two-step solving
`-threshold`⟨ϵ⟩	a robustness threshold	`-visualize`	generate visualization data
`-solver`⟨Name⟩	`z3`, `yices`, or `dreal`	`-goal`	goals to be checked

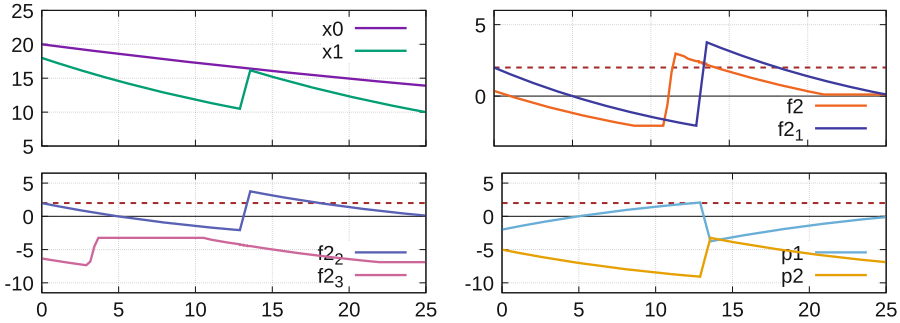

Fig. 3. Visualization of a counterexample (horizontal dotted lines denote $\epsilon = 2$).

```
$./stlmc ./therm.model -bound 5 -time-bound 25 -threshold 2 \
         -goal f2 -solver dreal -two-step -parallel -visualize
result: counterexample found at bound 2 (14.70277s)
```

Similarly, the following command verified the formula `f1` up to bounds $N = 5$ and $\tau = 25$ with respect to $\epsilon = 0.5$ in 819 s using dReal:

```
$./stlmc ./therm.model -bound 5 -time-bound 25 -threshold 0.5 \
         -goal f1 -solver dreal -two-step -parallel
result : True (818.73110s)
```

STLMC provides a command to visualize counterexamples for robust STL model checking. It can generate images representing counterexample trajectories and robustness degrees. Figure 3 shows the visualization graphs, showing the values of variables or robustness degrees over time, generated for the formula $\mathtt{f2} = \Box_{[2,4]}(x_0 - x_1 \geq 4 \rightarrow \Diamond_{[3,10]}(x_0 - x_1 \leq -3))$ with the subformulas:

$$\mathtt{f2}_1 = x_0 - x_1 \geq 4 \rightarrow \Diamond_{[3,10]}(x_0 - x_1 \leq -3) \qquad \mathtt{f2}_2 = \neg(x_0 - x_1 \geq 4)$$
$$\mathtt{f2}_3 = \Diamond_{[3,10]}(x_0 - x_1 \leq -3) \qquad \mathtt{p}_1 = x_0 - x_1 \geq 4 \qquad \mathtt{p}_2 = x_0 - x_1 \leq -3$$

The robustness degree of `f2` is less than ϵ at time 0, since the robustness degree of $\mathtt{f2}_1$ goes below ϵ in the interval $[2, 4]$, which is because both the degrees of $\mathtt{f2}_2$ and $\mathtt{f2}_3$ are less than ϵ in $[2, 4]$. The robustness degree of $\mathtt{f2}_3$ is less than ϵ in $[2, 4]$, since the robustness degree of $\mathtt{p}_2$ is less than ϵ in $[5, 14] = [2, 4] + [3, 10]$.

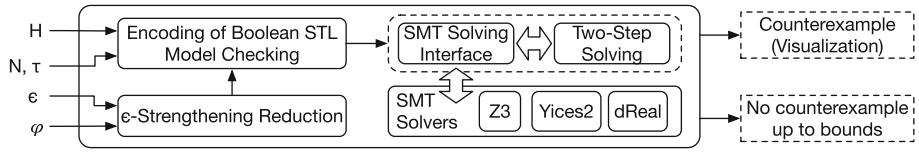

Fig. 4. The STLMC architecture

4 Algorithms and Implementation

Figure 4 shows the architecture of the STLMC tool. The tool first reduces robust STL model checking into Boolean STL model checking using ϵ-strengthening. It then applies an existing SMT-based STL model checking algorithm [3,29]. The satisfiability of the SMT encoding can be checked directly using an SMT solver or using the two-step solving algorithm to improve the performance for ODE dynamics. Our tool is implemented in around 9,500 lines of Python code.

4.1 Reduction to Boolean STL Model Checking

As usual for model checking, robust STL model checking is equivalent to finding a counterexample. Specifically, an STL formula φ is not satisfied on a hybrid automata H with respect to a robustness threshold $\epsilon > 0$ iff there exists a counterexample for which the robustness degree of $\neg\varphi$ is greater than or equal to $-\epsilon$. (Formally, $\neg(\forall \sigma \in H.\ \rho_\tau(\varphi, \sigma, t) > \epsilon)$ iff $\exists \sigma \in H.\ \rho_\tau(\neg\varphi, \sigma, t) \geq -\epsilon$.)

Consider a state proposition $x < 0$. Its robust model checking is equivalent to finding a counterexample $\sigma \in H$ with $\rho_\tau(x \geq 0, \sigma, t) \geq -\epsilon$, which is equivalent to $\rho_\tau(x \geq -\epsilon, \sigma, t) \geq 0$. Observe that $x \geq -\epsilon$ is *weaker* than $x \geq 0$ by ϵ. The notion of ϵ-weakening is first introduced in [21] for first-order logic, and we extend the definitions of ϵ-weakening and ϵ-strengthening to STL as follows.

Definition 3. *The ϵ-weakening $\varphi^{-\epsilon}$ and ϵ-strengthening $\varphi^{+\epsilon}$ of φ are defined as follows: $(p^{-\epsilon})(s) = p(s) - \epsilon$ and $(p^{+\epsilon})(s) = p(s) + \epsilon$ for a state s, and:*

$$(\neg\varphi)^{-\epsilon} \equiv \neg(\varphi^{+\epsilon}) \quad (\varphi_1 \wedge \varphi_2)^{-\epsilon} \equiv \varphi_1^{-\epsilon} \wedge \varphi_2^{-\epsilon} \quad (\varphi_1 \,\mathbf{U}_I\, \varphi_2)^{-\epsilon} \equiv \varphi_1^{-\epsilon} \,\mathbf{U}_I\, \varphi_2^{-\epsilon}$$
$$(\neg\varphi)^{+\epsilon} \equiv \neg(\varphi^{-\epsilon}) \quad (\varphi_1 \wedge \varphi_2)^{+\epsilon} \equiv \varphi_1^{+\epsilon} \wedge \varphi_2^{+\epsilon} \quad (\varphi_1 \,\mathbf{U}_I\, \varphi_2)^{+\epsilon} \equiv \varphi_1^{+\epsilon} \,\mathbf{U}_I\, \varphi_2^{+\epsilon}$$

Finding a counterexample of φ for robust STL model checking can be reduced to finding a counterexample of the ϵ-strengthening $\varphi^{+\epsilon}$ for Boolean STL model checking. The satisfaction of φ by the Boolean STL semantics [29,31] is denoted by $\sigma, t \models_\tau \varphi$. We have the following theorem (see our report [42] for details).

Theorem 1. *(1) $\exists \sigma \in H.\ \sigma, t \models_\tau \neg(\varphi^{+\epsilon})$ implies $\exists \sigma \in H.\ \rho_\tau(\neg\varphi, \sigma, t) \geq -\epsilon$, and (2) $\forall \sigma \in H.\ \sigma, t \not\models_\tau \neg(\varphi^{+\epsilon})$ implies $\forall \sigma \in H.\ \rho_\tau(\varphi, \sigma, t) \geq \epsilon$.*

As a consequence, a counterexample of $\varphi^{+\epsilon}$ for Boolean STL model checking is also a counterexample of φ for robust STL model checking. If there is no counterexample of $\varphi^{+\epsilon}$ for Boolean STL model checking, then φ is satisfied on H with respect to any robustness threshold $0 < \epsilon' < \epsilon$. It is worth noting that φ may not be satisfied on H with respect to ϵ itself.

4.2 Boolean STL Model Checking Algorithm

For Boolean STL model checking, there exist *refutationally complete* bounded model checking algorithms [3,29] with two bound parameters: τ for the time domain, and N for the number of mode changes and variable points. A time point t is a variable point if a truth value of φ's subformula changes at t. The algorithms build an SMT encoding $\Psi^{N,\tau}_{H,\neg\varphi}$ of Boolean STL model checking:

Theorem 2. *[3, 29]* $\Psi^{N,\tau}_{H,\neg\varphi}$ *is satisfiable iff there is a counterexample trajectory* $\sigma \in H$*, with at most* N *variable points and mode changes, such that* $\sigma, t \not\models_\tau \varphi$*.*

For hybrid automata with polynomial continuous dynamics, the satisfiability of the encoding Ψ can be precisely determined using standard SMT solvers, including Z3 [12] and Yices2 [17]. For ODE dynamics, the satisfiability of Ψ is undecidable in general, but there exist specialized solvers, such as dReal [22] and iSAT-ODE [18], that can approximately determine the satisfiability.

To support various SMT solvers, the implementation of STLmc utilizes a generic wrapper interface based on the SMT-LIB standard [5]. Therefore, if it follows SMT-LIB, a new SMT solver can be easily integrated with our tool. Moreover, STLmc can also detect the most suitable solver for a given input model; e.g., if the model has ODE dynamics, then the tool chooses dReal.

The encoding Ψ includes universal quantification over time, e.g., because of invariant conditions. Several SMT solvers (including Z3 and Yice2) support these $\exists\forall$-conditions but at high computational costs [27]. For polynomial dynamics, we implement the encoding method [10] to simplify $\exists\forall$-conditions to quantifier-free formulas. For ODE dynamics, dReal natively supports $\exists\forall$-conditions [23].

4.3 Two-Step Solving Algorithm

To reduce the complexity of ODE dynamics, we propose a two-step solving algorithm in Algorithm 1, inspired by the lazy SMT solving approach [38]:

1. We obtain the *discrete abstraction* of the encoding Ψ by substituting the flow and invariant conditions with Boolean variables. We then enumerate a satisfying *scenario* π, a conjunction of literals, where π implies Ψ.
2. For each scenario π, we check the satisfiability of its *discrete refinement* with the flow and invariant conditions using dReal. If any refinement is satisfiable, we obtain a counterexample; otherwise, there is no counterexample.

We also implement a simple method to avoid redundant scenarios by minimizing a scenario. A scenario $\pi = l_1 \wedge \cdots \wedge l_m$ is minimal if $(\neg l_i \wedge \bigwedge_{j\neq i} l_j) \rightarrow \Psi$— one literal in π is false— is not valid. To minimize a scenario π, we use a dual propagation approach [33]. Since π implies Ψ, $\pi \wedge \neg\Psi$ is unsatisfiable. We compute the unsatisfiable core of $\pi \wedge \neg\Psi$ using Z3 to extract a minimal scenario from π.

We parallelize the two-step solving algorithm by running the satisfiability checking of refinements in parallel. If any of such refinements is satisfied and a counterexample is found, then all other jobs are terminated. If all refinements, checking in parallel, are unsatisfiable, then there is no counterexample. As shown in Sect. 5, it greatly improves the performance for the ODE cases in practice.

Algorithm 1: Two-Step SMT Solving Algorithm

Input: Hybrid automaton H, STL formula φ, threshold ϵ, bounds τ and N

```
for k = 1 to N do
    Ψ̄ ← abstraction of the encoding Ψ^{k,τ}_{H,¬(φ^{+ε})} without flow and inv;
    while checkSat(Ψ̄) is Sat do
        π ← a minimal satisfying scenario;
        π̂ ← the refinement of π with flow and inv;
        if checkSat(π̂) is Sat then
            return counterexample(result.satAssignment);
        Ψ̄ ← Ψ̄ ∧ ¬π;
return True;
```

5 Experimental Evaluation

We evaluate the effectiveness of the STLMC model checker using a number of hybrid system benchmarks and nontrivial STL properties.[4] We use the following models, adapted from existing benchmarks [2,6,19,20,25,34]: load management for two batteries (Bat), two networked water tank systems (Wat), autonomous driving of two cars (Car), a railroad gate (Rail), two networked thermostats (Thm), a spacecraft rendezvous (Space), navigation of a vehicle (Nav), and a filtered oscillator (Oscil). We use a modified model with either linear, polynomial, or ODE dynamics to analyze the effect of different continuous dynamics. For each model, we use three STL formulas with nested temporal operators. More details on the benchmark models can be found in the longer report [42].

We measure the SMT encoding size and execution time for robust STL model checking, up to discrete bound $N = 20$ for linear models, $N = 10$ for polynomial models, and $N = 5$ for ODEs models, with a timeout of 60 min. We use different time bounds τ and robustness thresholds ϵ for different models, since τ and ϵ depend on each model. As an underlying SMT solver, we use Yices for linear and polynomial models, and dReal for ODE models with a precision $\delta = 0.001$. We run both direct SMT solving (`1-step`) and two-step SMT solving (`2-step`). We use 25 cores for parallelizing the two-phase solving algorithm. We have run all experiments on Intel Xeon 2.8 GHz with 256 GB memory.

The experimental results are summarized in Table 2, where $|\Psi|$ denotes the size of the SMT encoding Ψ (in thousands) as the number of connectives in Ψ. For the model checking results, $\top$ indicates that the tool found no counterexample up to bound N, and $\bot$ indicates that the tool found a counterexample at bound $k \leq N$. For the algorithms (Alg.), we write one of the results with a better

[4] For reachability properties, STLMC has a similar performance to other SMT-based tools, because STLMC uses the same SMT encoding. Indeed, our previous work [29] shows that the underlying algorithm used for STLMC has comparable performance to other tools for reachability properties. Nonetheless, our companion report [42] also includes some experimental results comparing STLMC with four reachability analysis tools (HyComp [9], SpaceEx [20], Flow* [8], and dReach [28]).

Table 2. Robust Bounded Model Checking of STL (Time in seconds)

Dyn.	Model	τ	STL formula	ϵ	$\lvert\Psi\rvert$	Time	Result	k	Alg.	$\#\pi$
Linear (N = 20)	Car	40	$(\Diamond_{[3,5]}\, p_1)\, \mathbf{U}_{[2,10]}\, p_2$	0.1	2.5	7.6	$\bot$	5	1-step	-
			$\Box_{[3,10]}(\Diamond_{[5,15]}\, p_1)$	0.5	10.8	559.2	$\top$	-	1-step	-
			$(\Box_{[2,5]}\, p_1)\, \mathbf{R}_{[0,10)}\, p_2$	1.0	2.5	7.8	$\bot$	5	1-step	-
	Wat	20	$\Box_{[1,3]}(p_1\, \mathbf{R}_{[1,10]}\, p_2)$	2.5	18.8	25.1	$\top$	-	1-step	-
			$(\Diamond_{[1,10)}\, p_1)\, \mathbf{U}_{[2,5]}\, p_2$	0.1	1.9	4.3	$\bot$	4	1-step	-
			$\Diamond_{[4,10]}(p_1 \rightarrow \Box_{[2,5]}\, p_2)$	0.01	11.2	16.3	$\top$	-	1-step	-
	Bat	30	$\Diamond_{[4,10]}(p_1 \rightarrow \Box_{[4,10]}\, p_2)$	0.1	12.9	119.5	$\top$	-	1-step	-
			$(\Diamond_{[1,5]}\, p_1)\, \mathbf{R}_{[5,20]}\, p_2$	3.5	2.8	6.0	$\bot$	5	1-step	-
			$\Box_{[4,14]}(p_1 \rightarrow \Diamond_{[0,10]}\, p_2)$	0.1	3.8	44.6	$\bot$	8	1-step	-
Poly (N = 10)	Thm	10	$(\Box_{[2,10]}\, p_1)\, \mathbf{U}_{[1,4]}\, p_2$	0.5	2.0	4.4	$\bot$	4	1-step	-
			$\Diamond_{[0,5]}(p_1 \rightarrow \Box_{[2,5)}\, p_2)$	0.1	3.9	5.0	$\top$	-	1-step	-
			$\Diamond_{[0,10]}(p_1\, \mathbf{R}_{[2,4]}\, p_2)$	1.0	5.7	6.3	$\top$	-	1-step	-
	Car	15	$\Box_{[0,4]}(p_1 \rightarrow \Diamond_{[2,5]}\, p_2)$	0.5	2.2	5.5	$\bot$	5	1-step	-
			$(\Diamond_{[0,4]}\, p_1)\, \mathbf{U}_{[0,5]}\, p_2$	2.0	1.7	4.7	$\bot$	3	1-step	-
			$\Diamond_{[0,3]}(p_1\, \mathbf{U}_{[0,5]}\, p_2)$	0.1	7.3	7.7	$\top$	-	1-step	-
	Rail	20	$\Diamond_{[0,5]}(p_1\, \mathbf{U}_{[1,8]}\, p_2)$	1.0	2.3	3.0	$\bot$	5	1-step	-
			$\Diamond_{[0,4]}(p_1 \rightarrow \Box_{[2,10]}\, p_2)$	5.0	3.8	3.8	$\top$	-	1-step	-
			$(\Box_{[0,5)}\, p_1)\, \mathbf{U}_{[2,10]}\, p_2$	4.0	1.9	2.7	$\bot$	4	1-step	-
ODE (N = 5)	Thm	25	$\Diamond_{[0,15]}(p_1\, \mathbf{U}_{[0,\infty)}\, p_2)$	0.5	1.2	818.7	$\top$	-	2-step	3,580
			$\Box_{[2,4]}(p_1 \rightarrow \Diamond_{[3,10]}\, p_2)$	2.0	0.7	14.7	$\bot$	2	2-step	91
			$\Box_{[0,10]}(p_1\, \mathbf{R}_{[0,\infty)}\, p_2)$	2.0	1.2	161.7	$\bot$	4	2-step	279
	Space	5	$\Box_{[0,2]}(p_1 \rightarrow \Diamond_{[0,3]}\, p_2)$	1.5	0.8	278.3	$\bot$	2	2-step	79
			$\Diamond_{[2,3]}(\Box_{[1,2]}\, p_1)$	0.1	1.1	37.0	$\bot$	3	2-step	138
			$\Diamond_{[0,4]}(p_1\, \mathbf{U}_{[0,\infty]}\, p_2)$	0.5	1.3	716.8	$\top$	-	2-step	2,681
	Oscil	8	$\Diamond_{[0,3]}(p_1\, \mathbf{R}_{[0,\infty)}\, p_2)$	0.1	1.5	108.9	$\top$	-	2-step	326
			$\Diamond_{[2,5]}(\Box_{[0,3]}\, p_1)$	1.0	1.2	192.8	$\bot$	3	2-step	601
			$(\Box_{[1,3]}\, p_1)\, \mathbf{R}_{[2,5]}\, p_2$	0.1	1.8	112.1	$\bot$	3	2-step	258
	Nav	10	$\Diamond_{[2,4]}(p_1 \rightarrow \Box_{[1,5]}\, p_2)$	3.0	1.2	399.3	$\bot$	3	2-step	1,388
			$\Diamond_{[2,4]}(\Box_{[3,6]}\, p_1)$	2.0	1.1	332.2	$\bot$	3	2-step	1,213
			$\Diamond_{[1,5]}(p_1\, \mathbf{R}_{[0,\infty)}\, p_2)$	1.0	1.4	749.6	$\top$	-	2-step	2,411

performance. For the `2-step` case, we also write the number of minimal scenarios generated ($\#\pi$). Actually, two-step SMT solving timed out for all linear and polynomial models, and direct SMT solving timed out for all ODE models.

As shown in Table 2, our tool can perform robust model checking of nontrivial STL formulas for hybrid systems with different continuous dynamics. The cases of ODE models generally take longer than the cases of linear and polynomial models, because of the high computational costs for ODE solving. Nevertheless, our parallelized two-step SMT solving method works well and all model checking analyses are finished before the timeout. In contrast, for linear and polynomial models with a larger discrete bound $N \geq 10$, direct SMT solving is usually effective but the two-step SMT solving method is not. There are too many scenarios, and the scenario generation does not terminate within 60 min. Therefore, the two algorithms implemented in our tool are complementary.

6 Related Work

There exist many tools for falsifying STL properties of hybrid systems, including Breach [14], S-talrio [1], and TLTk [11]. STL falsification techniques are based on STL monitoring [13,32], and often use stochastic optimization techniques, such as Ant-Colony Optimization [1], Monte-Carlo tree search [43], deep reinforcement learning [41], and so on. These techniques are often quite useful for finding counterexamples in practice, but, as mentioned, cannot be used to verify STL properties of hybrid systems.

There exist many tools for analyzing reachability properties of hybrid systems based on reachable-set computation, including C2E2 [16], Flow* [8], Hylaa [4], and SpaceEx [20]. They can be used to guarantee the correctness of invariant properties of the form $p \rightarrow \Box_I q$, but cannot verify general STL properties. In contrast, STLmc uses a refutation-complete bounded STL model checking algorithm to verify general STL properties, including complex ones.

Our tool is also related to SMT-based tools for analyzing hybrid systems, including dReach [28], HyComp [9], and HybridSAL [39]. These techniques also focus on analyzing invariant properties of hybrid systems, but some SMT-based tools, such as HyComp, can verify LTL properties of hybrid systems. Unlike STLmc, they cannot deal with general STL properties of hybrid systems.

7 Concluding Remarks

We have presented the STLmc tool for robust bounded model checking of STL properties for hybrid systems. STLmc can verify that, up to given bounds, the robustness degree of an STL formula φ is always greater than a given robustness threshold for all possible behaviors of a hybrid system. STLmc also provides a convenient user interface with an intuitive counterexample visualization.

Our tool leverages the reduction from robust model checking to Boolean model checking, and utilizes the refutation-complete SMT-based Boolean STL model checking algorithm to guarantee correctness up to given bounds and find subtle counterexamples. STLmc can deal with hybrid systems with (nonlinear) ODEs using dReal. We have shown using various hybrid system benchmarks that STLmc can effectively analyze nontrivial STL properties.

Future work includes extending our tool with other hybrid system analysis methods, such as reachable-set computation, besides SMT-based approaches.

Acknowledgments. This work was supported in part by the National Research Foundation of Korea (NRF) grants funded by the Korea government (MSIT) (No. 2021R1A5A1021944 and No. 2019R1C1C1002386).

References

1. Annpureddy, Y., Liu, C., Fainekos, G., Sankaranarayanan, S.: S-TaLiRo: a tool for temporal logic falsification for hybrid systems. In: Abdulla, P.A., Leino, K.R.M. (eds.) TACAS 2011. LNCS, vol. 6605, pp. 254–257. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-19835-9_21

2. Bae, K., Gao, S.: Modular SMT-based analysis of nonlinear hybrid systems. In: Proceedings FMCAD, pp. 180–187. IEEE (2017)
3. Bae, K., Lee, J.: Bounded model checking of signal temporal logic properties using syntactic separation. Proc. ACM Program. Lang. **3**(POPL), 1–30 (2019). 51
4. Bak, S., Duggirala, P.S.: HYLAA: a tool for computing simulation-equivalent reachability for linear systems. In: Proceedings of the HSCC, pp. 173–178. ACM (2017)
5. Barrett, C., Fontaine, P., Tinelli, C.: The SMT-LIB Standard: Version 2.5. Technical report, Department of Computer Science, University of Iowa (2015). www.SMT-LIB.org
6. Chan, N., Mitra, S.: Verifying safety of an autonomous spacecraft rendezvous mission. In: Proceedings of the ARCH. EPiC Series in Computing, vol. 48. EasyChair (2017)
7. Chen, G., Liu, M., Kong, Z.: Temporal-logic-based semantic fault diagnosis with time-series data from industrial Internet of Things. IEEE Trans. Industr. Electron. **68**(5), 4393–4403 (2020)
8. Chen, X., Ábrahám, E., Sankaranarayanan, S.: Flow*: an analyzer for non-linear hybrid systems. In: Sharygina, N., Veith, H. (eds.) CAV 2013. LNCS, vol. 8044, pp. 258–263. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-39799-8_18
9. Cimatti, A., Griggio, A., Mover, S., Tonetta, S.: HyComp: an SMT-based model checker for hybrid systems. In: Baier, C., Tinelli, C. (eds.) TACAS 2015. LNCS, vol. 9035, pp. 52–67. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-46681-0_4
10. Cimatti, A., Mover, S., Tonetta, S.: A quantifier-free SMT encoding of non-linear hybrid automata. In: Proceedings of the FMCAD, pp. 187–195. IEEE (2012)
11. Cralley, J., Spantidi, O., Hoxha, B., Fainekos, G.: TLTk: a toolbox for parallel robustness computation of temporal logic specifications. In: Deshmukh, J., Ničković, D. (eds.) RV 2020. LNCS, vol. 12399, pp. 404–416. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-60508-7_22
12. de Moura, L., Bjørner, N.: Z3: an efficient SMT solver. In: Ramakrishnan, C.R., Rehof, J. (eds.) TACAS 2008. LNCS, vol. 4963, pp. 337–340. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-78800-3_24
13. Deshmukh, J.V., Donzé, A., Ghosh, S., Jin, X., Juniwal, G., Seshia, S.A.: Robust online monitoring of signal temporal logic. Formal Methods Syst. Des. **51**(1), 5–30 (2017). https://doi.org/10.1007/s10703-017-0286-7
14. 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
15. 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
16. Duggirala, P.S., Mitra, S., Viswanathan, M., Potok, M.: C2E2: a verification tool for stateflow models. In: Baier, C., Tinelli, C. (eds.) TACAS 2015. LNCS, vol. 9035, pp. 68–82. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-46681-0_5
17. Dutertre, B.: Yices 2.2. In: Biere, A., Bloem, R. (eds.) CAV 2014. LNCS, vol. 8559, pp. 737–744. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-08867-9_49

18. Eggers, A., Ramdani, N., Nedialkov, N.S., Fränzle, M.: Improving the SAT modulo ODE approach to hybrid systems analysis by combining different enclosure methods. Softw. Syst. Model. **14**(1), 121–148 (2015). https://doi.org/10.1007/s10270-012-0295-3
19. Fehnker, A., Ivančić, F.: Benchmarks for hybrid systems verification. In: Alur, R., Pappas, G.J. (eds.) HSCC 2004. LNCS, vol. 2993, pp. 326–341. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-24743-2_22
20. 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.) CAV 2011. LNCS, vol. 6806, pp. 379–395. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-22110-1_30
21. Gao, S., Avigad, J., Clarke, E.M.: Delta-decidability over the reals. In: 2012 27th Annual IEEE Symposium on Logic in Computer Science, pp. 305–314. IEEE (2012)
22. Gao, S., Kong, S., Clarke, E.M.: dReal: an SMT solver for nonlinear theories over the reals. In: Bonacina, M.P. (ed.) CADE 2013. LNCS (LNAI), vol. 7898, pp. 208–214. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-38574-2_14
23. Gao, S., Kong, S., Clarke, E.M.: Satisfiability modulo ODEs. In: Proceedings of the FMCAD, pp. 105–112. IEEE (2013)
24. Goldman, R.P., Bryce, D., Pelican, M.J.S., Musliner, D.J., Bae, K.: A hybrid architecture for correct-by-construction hybrid planning and control. In: Rayadurgam, S., Tkachuk, O. (eds.) NFM 2016. LNCS, vol. 9690, pp. 388–394. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-40648-0_29
25. Henzinger, T.: The theory of hybrid automata. In: Inan, M.K., Kurshan, R.P. (eds.) Verification of Digital and Hybrid Systems. NATO ASI Series, vol. 170, pp. 265–292. Springer, Heidelberg (2000). https://doi.org/10.1007/978-3-642-59615-5_13
26. Jin, X., Deshmukh, J.V., Kapinski, J., Ueda, K., Butts, K.: Powertrain control verification benchmark. In: Proceedings of the HSCC. ACM (2014)
27. Jovanović, D., de Moura, L.: Solving non-linear arithmetic. In: Gramlich, B., Miller, D., Sattler, U. (eds.) IJCAR 2012. LNCS (LNAI), vol. 7364, pp. 339–354. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-31365-3_27
28. Kong, S., Gao, S., Chen, W., Clarke, E.: dReach: δ-reachability analysis for hybrid systems. In: Baier, C., Tinelli, C. (eds.) TACAS 2015. LNCS, vol. 9035, pp. 200–205. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-46681-0_15
29. Lee, J., Yu, G., Bae, K.: Efficient SMT-based model checking for signal temporal logic. In: Proceedings of the ASE, pp. 343–354. IEEE (2021)
30. Ma, M., Bartocci, E., Lifland, E., Stankovic, J., Feng, L.: SaSTL: spatial aggregation signal temporal logic for runtime monitoring in smart cities. In: Proceedings of the ICCPS, pp. 51–62. IEEE (2020)
31. 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
32. Ničković, D., Lebeltel, O., Maler, O., Ferrère, T., Ulus, D.: AMT 2.0: qualitative and quantitative trace analysis with extended signal temporal logic. In: Beyer, D., Huisman, M. (eds.) TACAS 2018. LNCS, vol. 10806, pp. 303–319. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-89963-3_18
33. Niemetz, A., Preiner, M., Biere, A.: Turbo-charging lemmas on demand with don't care reasoning. In: Proceedings of the FMCAD, pp. 179–186. IEEE (2014)

34. Raisch, J., Klein, E., Meder, C., Itigin, A., O'Young, S.: Approximating automata and discrete control for continuous systems — two examples from process control. In: Antsaklis, P., Lemmon, M., Kohn, W., Nerode, A., Sastry, S. (eds.) HS 1997. LNCS, vol. 1567, pp. 279–303. Springer, Heidelberg (1999). https://doi.org/10.1007/3-540-49163-5_16
35. Roehm, H., Oehlerking, J., Heinz, T., Althoff, M.: STL model checking of continuous and hybrid systems. In: Artho, C., Legay, A., Peled, D. (eds.) ATVA 2016. LNCS, vol. 9938, pp. 412–427. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-46520-3_26
36. Roohi, N., Kaur, R., Weimer, J., Sokolsky, O., Lee, I.: Parameter invariant monitoring for signal temporal logic. In: Proceedings of the HSCC, pp. 187–196. ACM (2018)
37. Sankaranarayanan, S., Fainekos, G.: Falsification of temporal properties of hybrid systems using the cross-entropy method. In: Proceedings of the HSCC, pp. 125–134 (2012)
38. Sebastiani, R.: Lazy satisfiability modulo theories. J. Satisfiability Boolean Model. Comput. **3**(3–4), 141–224 (2007)
39. Tiwari, A.: HybridSAL relational abstracter. In: Madhusudan, P., Seshia, S.A. (eds.) CAV 2012. LNCS, vol. 7358, pp. 725–731. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-31424-7_56
40. Xu, Z., Belta, C., Julius, A.: Temporal logic inference with prior information: an application to robot arm movements. IFAC-PapersOnLine **48**(27), 141–146 (2015)
41. Yamagata, Y., Liu, S., Akazaki, T., Duan, Y., Hao, J.: Falsification of cyber-physical systems using deep reinforcement learning. IEEE Trans. Softw. Eng. **47**(12), 2823–2840 (2020)
42. Yu, G., Lee, J., Bae, K.: Robust STL model checking of hybrid systems using SMT (2022). https://stlmc.github.io/assets/files/stlmc-techrep.pdf
43. 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