

# 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

**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) [33] 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 [28], robotics [26, 42], medical systems [38], IoT [8], smart cities [32], 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 [14, 16, 34] or find counterexamples [2, 39, 44], 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 [4, 31, 37]. In particular, the SMT-based bounded model checking algorithms [4, 31] 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.

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  $\varphi$  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  $\epsilon$ -*strengthening* (perturbing the problem by  $\epsilon$  to make it harder to be true), first proposed in [23] for first-order logic and extended to STL. We then apply the refutation-complete bounded model checking algorithm [4, 31] 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, STL<sub>MC</sub> also implements several techniques to improve the usability and scalability of the tool:

- STL<sub>MC</sub> implements a generic interface to connect with various SMT solvers, such as Z3 [13], Yices2 [18], and dReal [24]. Since dReal can (approximately) deal with nonlinear ordinary differential equations (ODEs), STL<sub>MC</sub> can also support hybrid systems with nonlinear ODE dynamics.
- STL<sub>MC</sub> 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.
- STL<sub>MC</sub> 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 STL<sub>MC</sub> 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>.

The rest of this paper is organized as follows. Section 2 explains some background on robust STL model checking of hybrid systems using STL<sub>MC</sub>. Section 3 presents the STL<sub>MC</sub> tool and language. Section 4 presents the algorithm and implementation of STL<sub>MC</sub>. Section 5 shows the experimental results. Section 6 discusses the related work. Finally, Section 7 presents some concluding remarks.

## 2 Background

### 2.1 Hybrid Automata

Hybrid automata are widely used for formalizing cyber-physical systems (CPSs) that exhibit both discrete and continuous behaviors. In a hybrid automaton  $H$ , a set of *modes*  $Q$  specifies discrete states, and a finite set of real-valued *variables*  $X = \{x_1, \dots, x_l\}$  specifies continuous states. A state of  $H$  is a pair  $\langle q, \vec{v} \rangle \in Q \times \mathbb{R}^l$  of mode  $q$  and real-valued vector  $\vec{v}$ . An initial condition  $init(\langle q, \vec{v} \rangle)$  defines a set of initial states. An invariant condition  $inv(\langle q, \vec{v} \rangle)$  defines a set of valid states. A flow condition  $flow(\langle q, \vec{v} \rangle, t, \langle q, \vec{v}_t \rangle)$  defines a *continuous evolution* of variables  $X$  from a value  $\vec{v}$  to  $\vec{v}_t$  over duration  $t$  in mode  $q$ . A jump condition  $jump(\langle q, \vec{v} \rangle, \langle q', \vec{v}' \rangle)$  defines a discrete transition from state  $\langle q, \vec{v} \rangle$  to  $\langle q', \vec{v}' \rangle$ .

**Definition 1.** A hybrid automaton is a tuple  $H = (Q, X, init, inv, jump, flow)$ .

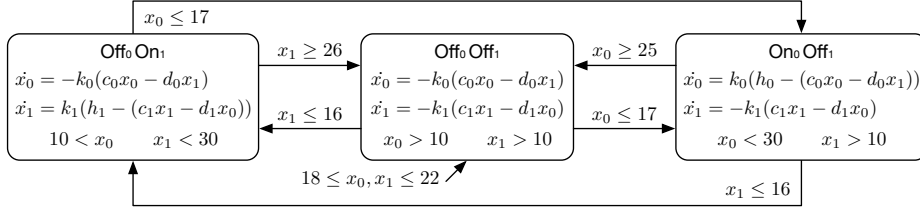


Fig. 1: A hybrid automaton for the networked thermostats.

A signal  $\sigma$  represents a continuous execution of a hybrid automaton  $H$ , given by a function  $[0, \tau) \rightarrow Q \times \mathbb{R}^l$  with a time bound  $\tau > 0$ . A signal  $\sigma$  is called *bounded* if  $\tau < \infty$ . A signal  $\sigma$  is called a *trajectory* of a hybrid automaton  $H$ , written  $\sigma \in H$ , if  $\sigma$  describes a valid behavior of  $H$ .

**Definition 2.** For a hybrid automaton  $H$ , a signal  $\sigma : [0, \tau) \rightarrow Q \times \mathbb{R}^l$  is a trajectory of  $H$ , written  $\sigma \in H$ , if there exist a sequence of modes  $q_1, q_2, q_3, \dots$  and of times  $0 = t_0 < t_1 < \dots < \tau$  such that:

- the initial condition holds at time  $t_0$ : i.e.,  $\text{init}(\sigma(t_0))$  holds;
- for  $i \geq 1$ , the values of  $X$  change from  $\sigma(t_{i-1})$  for time  $t_i - t_{i-1}$  by the flow condition, satisfying the invariant condition: i.e., for any  $t \in [t_{i-1}, t_i)$ :

$$\text{flow}(\sigma(t_{i-1}), t - t_{i-1}, \sigma(t)) \quad \text{and} \quad \text{inv}(\sigma(t))$$

- for  $i \geq 1$ , a discrete jump happens at time  $t_i$ : i.e., for some  $s_i \in Q \times \mathbb{R}^l$ :

$$\text{flow}(\sigma(t_{i-1}), t_i - t_{i-1}, s_i) \quad \text{and} \quad \text{jump}(s_i, \sigma(t_i))$$

*Example 1.* There are two rooms connected by an open door. The temperature  $x_i$  of each room  $i \in \{0, 1\}$  is controlled by each thermostat, depending on the heater's mode  $q_i \in \{\text{On}, \text{Off}\}$  and the other room's temperature. The continuous dynamics of  $x_i$  can be given as ODEs as follows [3, 27]:

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

where  $K_i, h_i, c_i, d_i$  are constants depending on the size of the room, the heater's power, and the size of the door. Figure 1 shows a hybrid automaton of our 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.

## 2.2 Signal Temporal Logic

Signal temporal logic (STL) is widely used to specify properties of hybrid systems [33]. 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 an 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  $f : \mathbb{R}^l \rightarrow \mathbb{R}$ . Other common boolean and temporal operators can be derived by equivalences:

$$\begin{aligned} \varphi \vee \varphi' &\equiv \neg(\neg\varphi \wedge \neg\varphi'), & \diamond_I \varphi &\equiv \top \mathbf{U}_I \varphi, \\ \square_I \varphi &\equiv \neg \diamond_I \neg\varphi, & \varphi \mathbf{R}_I \varphi' &\equiv \neg((\neg\varphi) \mathbf{U}_I (\neg\varphi')) \end{aligned}$$

*Example 2.* For Example 1, we consider the following STL formulas:

- $\phi_1$ :  $\diamond_{[0,15]}(x_0 \geq 14 \mathbf{U}_{[0,\infty)} x_1 \leq 19)$ : at some moment in the first 15 seconds,  $x_1$  is less than or equal to 19; until then,  $x_0$  is greater than or equal to 14.  
 $\phi_2$ :  $\square_{[2,4]}(x_0 - x_1 \geq 4 \rightarrow \diamond_{[3,10]} x_0 - x_1 \leq -3)$ : between 2 and 4 seconds, whenever  $x_0 - x_1 \geq 4$ ,  $x_0 - x_1 \leq -3$  holds within 10 seconds after 3 seconds.  
 $\phi_3$ :  $\square_{[0,10]}(x_0 > 23 \mathbf{R}_{[0,\infty)} x_0 - x_1 \geq 4)$ : for the first 10 seconds,  $x_0$  is greater than 23; release then,  $x_0 - x_1 \geq 4$ .

The Minkowski sum of two intervals  $I$  and  $J$  is denoted by  $I + J$ . E.g.,  $[a, b] + [c, d] = [a + c, b + d]$ . For a singular interval  $\{t\}$ , the set  $\{t\} + I$  is often 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.

We consider a quantitative semantics of STL based on *robustness degrees* [16]. The semantics of a proposition  $p$  is specified as a function  $p : Q \times \mathbb{R}^l \rightarrow \overline{\mathbb{R}}$ , where  $\overline{\mathbb{R}} = \mathbb{R} \cup \{-\infty, \infty\}$ , that assigns to a state of  $H$  the degree to which  $p$  is true. For a state proposition of the form  $f(\vec{x}) \geq 0$ , the robustness degree is the value of  $f(\vec{x})$  at a given state. E.g., the robustness degree of proposition  $x_0 - x_1 \geq 4$  in Example 2 is the value of  $x_0 - x_1 - 4$  at a given state.

The robustness degree of an STL formula can be defined as follows [16], where a time bound  $\tau$  of a signal is explicitly taken into account.

**Definition 3.** *Given an STL formula  $\varphi$ , a signal  $\sigma : [0, \tau) \rightarrow \mathbb{R}^l$ , and a time  $t \in [0, \tau)$ , the robustness degree  $\rho_\tau(\varphi, \sigma, t) \in \overline{\mathbb{R}}$  is defined inductively by:*

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

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

**Definition 4 (Robust STL Model Checking).** *For a time bound  $\tau > 0$ , an STL formula  $\varphi$  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  $\sigma \in H$ ,  $\rho_\tau(\varphi, \sigma, t) > \epsilon$ .*

### 2.3 STL Boolean Semantic

We also introduce the Boolean semantics of STL, which was originally proposed as the semantics of STL in [33]. We consider the following definition from [31] that explicitly considers a time bound  $\tau > 0$ .

**Definition 5.** *Given an STL formula  $\varphi$ , a signal  $\sigma : [0, \tau) \rightarrow \mathbb{R}^l$ , and a time  $t \in [0, \tau)$ , the satisfaction of  $\varphi$ , denoted by  $\sigma, t \models_{\tau} \varphi$  is inductively defined by:*

$$\begin{aligned} \sigma, t \models_{\tau} p & \quad \text{iff } t < \tau \text{ and } p(\sigma(t)) = \top \\ \sigma, t \models_{\tau} \neg\varphi & \quad \text{iff } \sigma, t \not\models_{\tau} \varphi \\ \sigma, t \models_{\tau} \varphi_1 \wedge \varphi_2 & \quad \text{iff } \sigma, t \models_{\tau} \varphi_1 \text{ and } \sigma, t \models_{\tau} \varphi_2 \\ \sigma, t \models_{\tau} \varphi_1 \mathbf{U}_I \varphi_2 & \quad \text{iff } \exists t' \in (t + I) \cap [0, \tau). \sigma, t' \models_{\tau} \varphi_2, \forall t'' \in [t, t']. \sigma, t'' \models_{\tau} \varphi_1 \end{aligned}$$

The following theorem shows the obvious relationship between the Boolean semantics and the quantitative semantics. If the robustness degree of  $\varphi$  is strictly positive (resp., negative), then  $\varphi$  is true (resp., false).

**Theorem 1.** [16] *For an STL formula  $\varphi$ , a signal  $\sigma$ , and a time  $t \in [0, \tau)$ ,  $\rho_{\tau}(\varphi, \sigma, t) > 0$  implies  $\sigma, t \models_{\tau} \varphi$ , and  $\rho_{\tau}(\varphi, \sigma, t) < 0$  implies  $\sigma, t \not\models_{\tau} \varphi$ .*

## 3 The STLMC Tool

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 Modeling language

The model input format of STLMC, inspired by dReach [30], consists of five sections: variable declarations, mode definitions, initial conditions, state propositions, and STL properties. Mode and continuous variables specify discrete and continuous states of hybrid automata. Mode definitions specify flow, jump, and invariant conditions. STL formulas can also include user-defined state propositions.

**Variable Declarations.** STLMC uses mode and continuous variables to specify discrete and continuous states of a hybrid automaton. *Mode variables* define a set of discrete modes  $Q$ , and *continuous variables* define a set of real-valued variables  $X$ .<sup>1</sup> We can also declare named constants whose values do not change.

Mode variables are declared with one of three types: `bool` variables with `true` and `false` values, `int` variables with integer values, and `real` variables with real values. E.g., the following declares a `bool` variable `b` and an `int` variable `i`:

<sup>1</sup> An assignment to mode and continuous variables corresponds to a single state.

```
bool b;   int i;
```

Continuous variables are declared with domain intervals. Domains can be any intervals of real numbers, including open, closed, and half-open intervals. E.g., the following declares two continuous variable  $x$  and  $y$ :

```
[0, 50] x;   (-1.1, 1) y;
```

Finally, constants are introduced with the `const` keyword. Constants can have Boolean, integer, or rational values. For example, the following declares a constant `k1` with a rational value 0.015:

```
const k1 = 0.015;
```

**Mode Definitions.** In STL<sub>MC</sub>, *mode blocks* define mode, jump, invariant, and flow conditions for a group of modes in a hybrid automaton. Multiple model blocks can be declared, and each mode block consists of four components:

```
{
  mode: ...
  inv: ...
  flow: ...
  jump: ...
}
```

A mode component contains a semicolon-separated set of Boolean conditions over mode variables. The conjunction of these conditions represents a set of modes.<sup>2</sup> E.g., the following represents a set of two modes  $\{(true, 1), (true, 2)\}$ , provided there are two mode variables `b` (of `bool` type) and `i` (of `int` type):

```
b = true;   i > 0;   i < 3;
```

An `inv` component contains a semicolon-separated set of Boolean formulas over continuous variables. The conjunction of these conditions then represents the invariant condition for each mode in the mode block. For example, the following declares an invariant condition for two continuous variables  $x$  and  $y$ :

```
x < 30;   y > - 0.5;
```

A `flow` component contains either a system of ordinary differential equations (ODEs) or a closed-form solution of ODEs. In STL<sub>MC</sub>, a system of ODEs over continuous variables  $x_1, \dots, x_n$  is written as a semicolon-separated equation of the following form, where  $e_i$  denotes an expression over  $x_1, \dots, x_n$ :

<sup>2</sup> We assume that the mode conditions of different mode blocks cannot be satisfied at the same time, which can be automatically detected by our tool.

```

d/dt[x1] = e1(x1, ..., xn) ;
...
d/dt[xn] = en(x1, ..., xn) ;

```

A closed-form solution of ODEs is written as a set of continuous functions, parameterized by a time variable  $t$  and the initial values  $x_1(0), \dots, x_n(0)$ :

```

x1(t) = e1(t, x1(0), ..., xn(0)) ;
...
xn(t) = en(t, x1(0), ..., xn(0)) ;

```

A jump component contains a set of jump conditions  $guard \Rightarrow reset$ , where  $guard$  and  $reset$  are Boolean conditions over mode and continuous variables. We use “primed” variables to denote states after jumps have occurred. E.g., the following defines a jump with four variables  $b$ ,  $i$ ,  $x$ , and  $y$  (declared above):

```

(and (i = 1) (x > 10)) => (and (b' = false) (i = 3) (x' = x) (y' = 0));

```

**Initial Conditions.** In the `init` section, an initial condition is declared as a set of Boolean formulas over mode and continuous variables. Similarly, the conjunction of these conditions represents a set of initial modes. E.g., the following shows an initial condition with variables  $b$ ,  $i$ ,  $x$ , and  $y$ :

```

init: (and (not b) (i = 0) (19.9 <= x) (y = 0));

```

**STL Properties.** In the `goal` section, STL properties are declared with labels. State propositions are arithmetic and relational expressions over mode and continuous variables, and labels can be omitted. For example, the following declares the first STL formula in Example 2 with label `f2`:

```

f2: [] [2, 4] ((x0 - x1 >= 4) -> <> [3, 10] (x0 - x1 <= -3));

```

To make it easy to write repeated propositions, “named” state propositions can be declared in the `proposition` section. For example, the above STL formula can be rewritten using two propositions `p1` and `p2` as follows:

```

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

goal:
[f2]: [] [2, 4] (p1 -> <> [3, 10] p2);

```

Figure 2 shows the input model of the hybrid automaton described in the running of Example 1. Constants are introduced with the `const` keyword. Two mode variables `on0` and `on1` denote the heaters’ modes. Continuous variables `x0`

<pre> 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 &lt; x0; x1 &lt; 30;  flow: d/dt[x0] = - k0 * (c0 * x0 - d0 * x1);        d/dt[x1] = k1 * (h1 - (c1 * x1 - d1 * x0));  jump: x0 &lt;= 17 =&gt; (and (on0' = 1) (on1' = 0)                   (x0' = x0) (x1' = x1));        x1 &gt;= 26 =&gt; (and (on1' = 0) (on0' = on0)                   (x0' = x0) (x1' = x1)); }  {mode: on0 = 1;      on1 = 0;  inv:  x0 &lt; 30; x1 &gt; 10;  flow: d/dt[x0] = k0 * (h0 - (c0 * x0 - d0 * x1));        d/dt[x1] = - k1 * (c1 * x1 - d1 * x0);  jump: x1 &lt;= 16 =&gt; (and (on0' = 0) (on1' = 1)                   (x0' = x0) (x1' = x1)); } </pre>	<pre> x0 &gt;= 25 =&gt; (and (on0' = 0) (on1' = on1)               (x0' = x0) (x1' = x1)); } {mode: on0 = 0;      on1 = 0;  inv:  x0 &gt; 10; x1 &gt; 10;  flow: d/dt[x0] = - k0 * (c0 * x0 - d0 * x1);        d/dt[x1] = - k1 * (c1 * x1 - d1 * x0);  jump:    x0 &lt;= 17 =&gt; (and (on0' = 1) (on1' = on1)                  (x0' = x0) (x1' = x1));    x1 &lt;= 16 =&gt; (and (on1' = 1) (on0' = on0)                  (x0' = x0) (x1' = x1)); }  init: on0 = 0; 18 &lt;= x0; x0 &lt;= 22;       on1 = 0; 18 &lt;= x1; x1 &lt;= 22;  proposition: [p1]: x0 - x1 &gt;= 4;   [p2]: x0 - x1 &lt;= -3;  goal: [f1]: &lt;&gt;[0,15](x0 &gt;= 14 U[0, inf] x1 &lt;= 19); [f2]: [][2, 4](p1 -&gt; &lt;&gt;[3, 10] p2); </pre>
--	--

Fig. 2: An input model example

and  $x_1$  are declared with domain intervals. There are three “mode blocks” that specify the three modes in Fig. 1 and their invariant, flow, 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  $guard \Rightarrow reset$ , where  $guard$  and  $reset$  are logic formulas over mode and continuous variables, and “primed” variables denote states after the jump has occurred.

STL properties are declared in the `goal` section, and “named” propositions are declared in the `proposition` section. State propositions can include arithmetic and relational expressions over mode and continuous variables. For example, in Fig 2, the STL formula `f1` contains two state propositions  $x_0 \leq 26$  and  $x_0 \geq 17$ , and the formula `f2` contains the user-defined propositions `p1` and `p2`.

### 3.2 Configuration

A *configuration file* is defined in a `.cfg` file. The configuration file specifies the analysis parameters for STLMC and underlying solvers. Analysis parameters for STLMC are listed in Table 1. Fig. 3 shows a configuration example.

Analysis parameters for STLMC (Table 1) are defined within curly braces introduced with the `common` keyword. There are two mandatory parameters and eight optional parameters. When mandatory parameters are not set, STLMC throws an exception.

Two parameters, `bound` and `time-bound`, are required parameters because the STLMC tool solves the robust STL model checking problem upto two bound



```

# STLMC configuration
common {
  # mandatory arguments
  # bound =
  # time-bound =

  threshold = 0.01          # positive rational number
  solver = auto              # dreal, yices, z3
  time-horizon = "time-bound" # restrict time duration
  goal = "all"               # STL formula labels
  two-step = "false"        # on
  parallel = "false"         # on
  visualize = "false"        # on
  verbose = "false"          # print verbose messages
}

# underlying solver
z3      { logic = "QF_NRA" # QF_LRA }
yices   { logic = "QF_NRA" # QF_LRA }
dreal {
  precision = 0.001          # positive rational number
  ode-order = 5              # natural number
  ode-step = 0.001           # positive rational number
  executable-path = "../dreal" # dreal executable path
}

```

Fig. 3: A configuration example

parameters. A bound limits the number of mode changes and the number of *variable points*—at which the truth value of some STL subformula changes—in trajectories. A time bound bounds the time domain of trajectories. A threshold is a robustness threshold that bound the robustness degree of an STL formula. A time horizon  $T$  limits the maximum time duration of single modes in trajectories. If an STL formula is declared with a label, only that formula can be analyzed by providing the label as an argument to the `goal`. The parameters `two-step` and `parallel` enable the (parallelized) two-step optimization. When the `visualize` is set, extra data for visualization is generated.

We can choose different SMT solvers by setting an argument to the `solver`. The STLMC tool currently supports three SMT solvers: Z3 [13], Yices2 [18] and dReal [24]. The underlying solver is chosen depending on the flow conditions of a hybrid automaton. Z3 and Yices2 can deal with linear and polynomial functions, and dReal can deal with nonlinear ODEs—which is undecidable in general for hybrid automata—approximately up to a given precision  $\delta > 0$ .

Analysis parameters for each solver are defined within curly braces introduced with the `solver_name` keyword. For `z3` and `yices2` there is only one parameter, `logic`, which sets the background logic for SMT solving. There are four param-

Name	Explanation	Default
<code>bound</code>	a discrete bound $N \in \mathbb{N}$	-
<code>time-bound</code>	a time bound $\tau \in \mathbb{Q}^+$	-
<code>threshold</code>	a robustness threshold $\epsilon \in \mathbb{Q}^+$	0.01
<code>solver</code>	an SMT solver to be used ( <code>z3/yices/dreal</code> )	<code>auto</code>
<code>time-horizon</code>	a mode duration bound	$\tau$
<code>goal</code>	a list of STL goals to be analyzed	all
<code>two-step</code>	use the two-step optimization	disabled
<code>parallel</code>	parallelize the two-step optimization	disabled
<code>visualize</code>	generate extra visualization data	disabled
<code>verbose</code>	print more information of the execution	disabled

Table 1: STLMC configuration parameters.

eters for the `dreal3` solver. A `precision` sets  $\delta$  for the framework of  $\delta$ -complete decision procedures. A `ode-order` sets an maximum order of taylor expansion of ordinary differential equations. A `ode-step` sets a time step for numerical calculation. A `executable-path` defines a path to the `dreal` executable file.

For flexibility in setting analysis parameters, three levels of configurations are provided: (1) default configuration, (2) model configuration, and (3) model specification configuration. The model specific configuration inherits the model configuration, and the model configuration inherits the default configuration.

The STLMC tool provides a predefined configuration file, named `default.cfg`, in the top directory. The default configuration sets all parameters to their default values except for mandatory arguments (Refer Fig. 3)

The model configuration specifies some analysis parameters related to a specific model. For example, we want to analyze the input model in Fig. 1 at bound 5 and time bound 25 with respect to robustness threshold  $\epsilon = 0.1$ . Then, we only need to change the following three parameters: `bound`, `time-bound`, and `threshold`. Figure 4 shows the model configuration that specifies these partial parameters. Other parameters such as `goal`, `parallel`, etc, inherit the values defined in `default.cfg`.

Furthermore, people may want to run a model using a different configuration for each formula. People can instantiate a model specific configurations to instantiate some analysis parameters in a model configuration. For the same example above, we want to analyze a formula `f2` with respect to  $\epsilon = 2$ . Then, we only need to change the following two parameters: `goal` and `threshold`. Figure 5 shows the model specific configuration that specifies these partial parameters. Other parameters such as `bound`, `time-bound`, etc, inherit the values defined in a model configuration.

### 3.3 Command Line interface

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

```
# STLMc configuration
common { bound = 5
         time-bound = 25
         threshold = 0.1 }
```

Fig. 4: A model configuration example

```
# STLMc configuration
common { goal = f2
         threshold = 2 }
```

Fig. 5: A model specific configuration example

Sec. 4.3. STLMC supports three SMT solvers to choose from based on continuous dynamics: Z3 [13] and Yices2 [18] can deal with linear and polynomial dynamics (solutions of ODEs are linear functions or polynomials), and dReal [24] 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  $\tau$  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 ( $m = 0$ ).<sup>3</sup>

The tool takes a model file, configuration files, and all parameters in the default configuration. The model file argument is required and other configuration arguments are optional.

```
./stlmc [path to model file] \
        -default-cfg [path to default config file]\
        -model-cfg [path to model config file] \
        -model-specific-cfg [path to model specific config file] \
        -bound [int] ...
```

If the `-default-cfg` is not provided, the tool uses the `default.cfg` in the top directory. If the `-model-cfg` is not given, the tool looks for a configuration file `<model_file>.cfg` in the path to model file.<sup>4</sup> If some parameters in the configuration are given as command-line options, those parameters override the values specified in the configuration files.

<sup>3</sup> This example also hints that STL model checking can be arbitrary complex even for one mode;  $\tau$  and  $m$  cannot limit such model checking computation, whereas  $N$  can limit the computation involving *both* discrete and continuous behaviors.

<sup>4</sup> If there is no matched model configuration file, the model configuration inherits the default configuration.

*Example 3.* Consider the input model in Fig. 2 (`therm.model`), the input model configuration in Fig. 4 (`them-model.cfg`), and the input model specification configuration in Fig. 5 (`them-model-spec.cfg`). The following command found a counterexample of the formula `f2` at bound 5 with respect to robustness threshold  $\epsilon = 2$  in 15 seconds using `dReal`.

```

$./stlmc ./therm.model -model-cfg therm-model.cfg \
        -model-specific-cfg therm-model-spec.cfg \
        -solver drealm -two-step -parallel -visualize
goal: []_[2.0,4.0] (p1 -> (<>_[3.0,10.0] p2))
result: counterexample found at bound 2 (14.70277 seconds)

```

The following command verifies the formula `f1` up to bounds  $N = 5$  and  $\tau = 25$  with respect to robustness threshold  $\epsilon = 0.5$  in 819 seconds using `dReal`.

```

$./stlmc ./therm.model -bound 5 -time-bound 30 -threshold 1 \
        -goal f1 -solver drealm -two-step -parallel
goal : (<>_[0.0,3.0] ((x0 >= 13) U_[0.0,inf) (x1 <= 22)))
result : True (818.73110 seconds)

```

### 3.4 Visualization

The STL<sub>MC</sub> tool provides a script to visualize counterexamples for robust STL model checking. The visualization script takes a counterexample file and a visualization configuration file and returns graphs representing counterexample trajectories and robustness degrees.

```

./stlmc-vis [path to counterexample file] \
        -cfg [path to configuration file]

```

The counterexample file is specified in a `.counterexample` file. The file is created when there is a counterexample and the `visualize` option is set. The configuration file contains following things: (1) continuous variables, (2) STL subformula labels, (3) output format, and (4) group of variables. The file contains variable information of continuous variables and STL subformulas. The STL<sub>MC</sub> tool supports "html" and "pdf" as output formats. The states of variables in a group are plotted on one graph. Only variables of the same type can be grouped together.<sup>5</sup> Variables not assigned to a group are grouped together according to their type. Figure 6 shows a visualization configuration file of our thermostat controllers.

*Example 4.* Consider the counterexample file for `f2` in Example 3 (`therm.model`) and the visualization configuration file in Fig. 6 (`therm_vis.cfg`). The following command generate a PDF image. in Fig. 7.

```

./stlmc-vis therm.counterexample -cfg therm_vis.cfg

```

<sup>5</sup> For example, if there is a group  $(x_0, f_2)$ , the tool will throw an error because the types of  $x_0$  and  $f_2$  are real and bool, respectively.

```

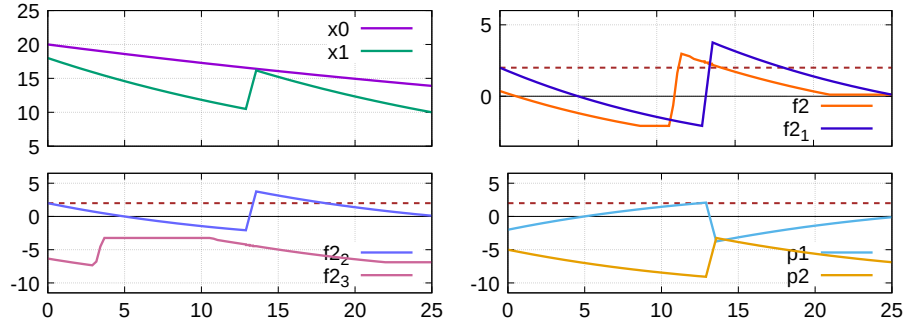
{
  # continuous variables: x0, x1
  # STL subformula labels:
  # f2 --> [][2, 4]((x0 - x1 >= 4) -> <>[3,10](x0 - x1 <= -3))
  # f2_1 --> (x0 - x1 >= 4) -> <>[3,10](x0 - x1 <= -3)
  # f2_2 --> not (x0 - x1 >= 4)
  # f2_3 --> <>[3,10](x0 - x1 <= -3)
  # p_1 --> x0 - x1 >= 4
  # p_2 --> x0 - x1 <= -4

  # output = pdf # html
  group { (x0, x1), (f2, f2_1) (f2_2, f2_3) (p_1, p_2) }
}

```

Fig. 6: A visualization configuration example

The robustness degree of  $f_2$  is less than  $\epsilon$  at time 0, since the robustness degree of  $f_{2_1}$  goes below  $\epsilon$  in the interval  $[2, 4]$ , which is because both the degrees of  $f_{2_2}$  and  $f_{2_3}$  are less than  $\epsilon$  in  $[2, 4]$ . The robustness degree of  $f_{2_3}$  is less than  $\epsilon$  in  $[2, 4]$ , since the robustness degree of  $p_2$  is less than  $\epsilon$  in  $[5, 14] = [2, 4] + [3, 10]$ .

Fig. 7: Visualization of a counterexample (horizontal dotted lines denote  $\epsilon = 2$ ).

## 4 Algorithms and Implementation

This section explains the algorithms and implementation of our STLMC tool. STLMC takes a hybrid automaton  $H$ , an STL formula  $\varphi$ , a variable point bound  $N$ , time bound  $\tau$ , and a threshold  $\epsilon$ . The tool first reduces robust STL model checking into Boolean STL model checking using  $\epsilon$ -strengthening. It then applies an existing SMT-based STL model checking algorithm [4, 31]. 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. We can easily see that an STL formula  $\varphi$  is not satisfied in a hybrid automata  $H$  with respect to a robustness threshold  $\epsilon > 0$  iff there is a trajectory that the robustness degree of  $\neg\varphi$  is greater than or equal to  $-\epsilon$ .

**Corollary 1.** *For a hybrid automaton  $H$ , a time  $t \in [0, \tau)$ , and a robustness threshold  $\epsilon > 0$ , an STL formula  $\varphi$  is not satisfied at  $t$  in  $H$  with respect to  $\epsilon$  iff there exists a trajectory  $\sigma \in H$  such that  $\rho_\tau(\neg\varphi, \sigma, t) \geq -\epsilon$ .*

*Proof.* By definition,  $\varphi$  is satisfied at  $t$  in  $H$  with respect to  $\epsilon$  iff for any trajectory  $\sigma \in H$ ,  $\rho_\tau(\varphi, \sigma, t) > \epsilon$  holds. Notice that  $\neg(\forall \sigma \in H. \rho_\tau(\varphi, \sigma, t) > \epsilon)$  iff  $\exists \sigma \in H. \rho_\tau(\varphi, \sigma, t) \leq \epsilon$ . The formula is equal to  $\exists \sigma \in H. -\rho_\tau(\varphi, \sigma, t) \geq -\epsilon$ . Thus,  $\exists \sigma \in H. \rho_\tau(\neg\varphi, \sigma, t) \geq -\epsilon$  by definition.  $\square$

Our goal is to reduce finding a counterexample of robust STL model checking into finding a counterexample of Boolean STL model checking. Let us first a simple case, say, a state proposition  $x \geq 0$ . Its robust model checking with respect to  $\epsilon$  is equivalent to finding a counterexample  $\sigma \in H$  with  $\rho_\tau(-x > 0, \sigma, t) \geq -\epsilon$  by Corollary 1, which is equivalent to  $\rho_\tau(-x > -\epsilon, \sigma, t) \geq 0$ . Notice that  $-x > -\epsilon$  can be obtained by *weakening*  $-x \geq 0$  by  $\epsilon$ .

The notions of  $\epsilon$ -weakening and  $\epsilon$ -strengthening are first introduced in [23] for first-order formulas. In this paper, we extend the definitions of  $\epsilon$ -weakening and  $\epsilon$ -strengthening to STL formulas as follows.

**Definition 6.** *The  $\epsilon$ -weakening  $\varphi^{-\epsilon}$  and  $\epsilon$ -strengthening  $\varphi^{+\epsilon}$  of  $\varphi$  are defined as follows:  $(p^{-\epsilon})(s) = p(s) - \epsilon$  and  $(p^{+\epsilon})(s) = p(s) + \epsilon$  for a state  $s$ , and:*

$$\begin{aligned} (\neg\varphi)^{-\epsilon} &\equiv \neg(\varphi^{+\epsilon}) & (\varphi_1 \wedge \varphi_2)^{-\epsilon} &\equiv \varphi_1^{-\epsilon} \wedge \varphi_2^{-\epsilon} & (\varphi_1 \mathbf{U}_I \varphi_2)^{-\epsilon} &\equiv \varphi_1^{-\epsilon} \mathbf{U}_I \varphi_2^{-\epsilon} \\ (\neg\varphi)^{+\epsilon} &\equiv \neg(\varphi^{-\epsilon}) & (\varphi_1 \wedge \varphi_2)^{+\epsilon} &\equiv \varphi_1^{+\epsilon} \wedge \varphi_2^{+\epsilon} & (\varphi_1 \mathbf{U}_I \varphi_2)^{+\epsilon} &\equiv \varphi_1^{+\epsilon} \mathbf{U}_I \varphi_2^{+\epsilon} \end{aligned}$$

The following lemmas state that relationships between the quantitative semantics of  $\varphi$  and the Boolean semantics of  $\varphi^{+\epsilon}$  and  $\varphi^{-\epsilon}$ .

**Lemma 1.** *For a signal  $\sigma$  and an STL property  $\varphi$ :*

$$\sigma, t \not\models_\tau \varphi^{+\epsilon} \Rightarrow \rho_\tau(\varphi, \sigma, t) \leq \epsilon \quad \text{and} \quad \sigma, t \models_\tau \varphi^{-\epsilon} \Rightarrow \rho_\tau(\varphi, \sigma, t) \geq -\epsilon$$

*Proof.* The proof is by structural induction on  $\varphi$ .

For  $\varphi = p$ , if  $\sigma, t \not\models_\tau p^{+\epsilon}$ , then  $p^{+\epsilon}(\sigma(t)) = \perp$  by definition. If  $\sigma, t \models_\tau p^{-\epsilon}$ , then  $p^{-\epsilon}(\sigma(t)) = \top$  by definition. There are two possible propositions forms,  $f(\vec{x}) \geq 0$  and  $f(\vec{x}) > 0$ . For  $p = f(\vec{x}) \geq 0$ ,  $\epsilon$ -strengthening  $p^{+\epsilon}$  of  $p$  is  $f(\vec{x}) \geq \epsilon$  and  $\epsilon$ -weakening  $p^{-\epsilon}$  of  $p$  is  $f(\vec{x}) \geq -\epsilon$  by definition.

Suppose  $\sigma, t \not\models_\tau p^{+\epsilon}$ . Then,  $f(\sigma(t)) < \epsilon$  is satisfied. The robustness degree  $\rho_\tau(f(\vec{x}) \geq 0, \sigma, t)$  is  $f(\sigma(t))$  by definition. Therefore,  $\rho_\tau(f(\vec{x}) \geq 0, \sigma, t) < \epsilon$ . If  $\rho_\tau(f(\vec{x}) \geq 0, \sigma, t) < \epsilon$ , then  $\rho_\tau(f(\vec{x}) \geq 0, \sigma, t) \leq \epsilon$ .

Suppose  $\sigma, t \models_{\tau} p^{-\epsilon}$ . Then,  $f(\sigma(t)) \geq -\epsilon$  is satisfied. The robustness degree  $\rho_{\tau}(f(\vec{x}) \geq 0, \sigma, t)$  is  $f(\sigma(t))$  by definition. Therefore,  $\rho_{\tau}(f(\vec{x}) \geq 0, \sigma, t) \geq -\epsilon$ .

For  $p = f(\vec{x}) > 0$ ,  $\epsilon$ -strengthening  $p^{+\epsilon}$  of  $p$  is  $f(\vec{x}) > \epsilon$  and  $\epsilon$ -weakening  $p^{-\epsilon}$  of  $p$  is  $f(\vec{x}) > -\epsilon$  by definition. Suppose  $\sigma, t \not\models_{\tau} p^{+\epsilon}$ . Then,  $f(\sigma(t)) \leq \epsilon$  is satisfied. Since  $\rho_{\tau}(f(\vec{x}) > 0, \sigma, t) = f(\sigma(t))$ ,  $\rho_{\tau}(f(\vec{x}) > 0, \sigma, t) \leq \epsilon$ .

Suppose  $\sigma, t \models_{\tau} p^{-\epsilon}$ . Then,  $f(\sigma(t)) > -\epsilon$  is satisfied. Thus,  $\rho_{\tau}(f(\vec{x}) > 0, \sigma, t) > -\epsilon$ , since  $\rho_{\tau}(f(\vec{x}) > 0, \sigma, t) = f(\sigma(t))$  by definition. The  $\epsilon$ -weakening of  $p$  is  $f(\vec{x}) > -\epsilon$  by definition. Thus,  $\rho_{\tau}(f(\vec{x}) > 0, \sigma, t) \geq -\epsilon$ .

For  $\varphi = \neg\phi$ ,  $\sigma, t \not\models_{\tau} (\neg\phi)^{+\epsilon}$  iff  $\sigma, t \not\models_{\tau} \neg(\phi^{-\epsilon})$ , since  $(\neg\phi)^{+\epsilon} \equiv \neg(\phi^{-\epsilon})$  by definition. Thus,  $\sigma, t \not\models_{\tau} \neg(\phi^{-\epsilon})$  iff  $\sigma, t \models_{\tau} (\phi^{-\epsilon})$ .

If  $\sigma, t \models_{\tau} (\phi^{-\epsilon})$ , then  $\rho_{\tau}(\phi, \sigma, t) \geq -\epsilon$  by induction hypothesis. By definition,  $\rho_{\tau}(\phi, \sigma, t) \geq -\epsilon$  iff  $-\rho_{\tau}(\phi, \sigma, t) \leq \epsilon$ . Thus,  $\rho_{\tau}(\neg\phi, \sigma, t) \leq \epsilon$ , since  $-\rho_{\tau}(\phi, \sigma, t) = \rho_{\tau}(\neg\phi, \sigma, t)$  by definition.

Suppose  $\sigma, t \models_{\tau} (\neg\phi)^{-\epsilon}$ . Then,  $\sigma, t \models_{\tau} \neg(\phi^{+\epsilon})$  is also satisfied, since  $(\neg\phi)^{-\epsilon} \equiv \neg(\phi^{+\epsilon})$  by definition. Thus,  $\sigma, t \models_{\tau} \neg(\phi^{+\epsilon})$  iff  $\sigma, t \not\models_{\tau} (\phi^{+\epsilon})$ .

If  $\sigma, t \not\models_{\tau} (\phi^{+\epsilon})$ , then  $\rho_{\tau}(\phi, \sigma, t) \leq \epsilon$  by induction hypothesis. By definition,  $\rho_{\tau}(\phi, \sigma, t) \leq \epsilon$  iff  $-\rho_{\tau}(\phi, \sigma, t) \geq -\epsilon$ . Thus,  $\rho_{\tau}(\neg\phi, \sigma, t) \geq -\epsilon$ , since  $-\rho_{\tau}(\phi, \sigma, t) = \rho_{\tau}(\neg\phi, \sigma, t)$  by definition.

For  $\varphi = \varphi_1 \wedge \varphi_2$ ,  $\sigma, t \not\models_{\tau} (\varphi_1 \wedge \varphi_2)^{+\epsilon}$  iff  $\sigma, t \not\models_{\tau} \varphi_1^{+\epsilon} \vee \sigma, t \not\models_{\tau} \varphi_2^{+\epsilon}$  by definition. by induction hypothesis Then, by induction hypothesis,

$$\rho_{\tau}(\varphi_1, \sigma, t) \leq \epsilon \vee \rho_{\tau}(\varphi_2, \sigma, t) \leq \epsilon.$$

Thus,  $\rho_{\tau}(\varphi_1 \wedge \varphi_2, \sigma, t) \leq \epsilon$ , since  $\rho_{\tau}(\varphi_1 \wedge \varphi_2, \sigma, t) = \min(\rho_{\tau}(\varphi_1, \sigma, t), \rho_{\tau}(\varphi_2, \sigma, t))$  and  $\rho_{\tau}(\varphi_1, \sigma, t) \leq \epsilon \vee \rho_{\tau}(\varphi_2, \sigma, t) \leq \epsilon$ .

Suppose  $\sigma, t \models_{\tau} (\varphi_1 \wedge \varphi_2)^{-\epsilon}$ . Then,  $\sigma, t \models_{\tau} \varphi_1^{-\epsilon} \wedge \sigma, t \models_{\tau} \varphi_2^{-\epsilon}$  by definition. Thus, by induction hypothesis,  $\rho_{\tau}(\varphi_1, \sigma, t) \geq -\epsilon \wedge \rho_{\tau}(\varphi_2, \sigma, t) \geq -\epsilon$ . Therefore,  $\rho_{\tau}(\varphi_1 \wedge \varphi_2, \sigma, t) \geq -\epsilon$ , since  $\rho_{\tau}(\varphi_1 \wedge \varphi_2, \sigma, t) = \min(\rho_{\tau}(\varphi_1, \sigma, t), \rho_{\tau}(\varphi_2, \sigma, t))$  and  $\rho_{\tau}(\varphi_1, \sigma, t) \geq -\epsilon \wedge \rho_{\tau}(\varphi_2, \sigma, t) \geq -\epsilon$ .

For  $\varphi = \varphi_1 \mathbf{U}_I \varphi_2$ , suppose  $\sigma, t \not\models_{\tau} (\varphi_1 \mathbf{U}_I \varphi_2)^{+\epsilon}$ . Then, for all time points  $t' \in (t + I) \cap [0, \tau)$ ,  $\sigma, t' \not\models_{\tau} \varphi_2^{+\epsilon} \vee \exists t'' \in [t, t'], \sigma, t'' \not\models_{\tau} \varphi_1^{+\epsilon}$  by definition. Therefore, by induction hypothesis,

$$\forall t' \in (t + I) \cap [0, \tau). (\rho_{\tau}(\varphi_2, \sigma, t') \leq \epsilon \vee \exists t'' \in [t, t'], \rho_{\tau}(\varphi_1, \sigma, t'') \leq \epsilon)$$

If  $\exists t'' \in [t, t'], \rho_{\tau}(\varphi_1, \sigma, t'') \leq \epsilon$ , then  $\inf_{t'' \in [t, t']} \rho_{\tau}(\varphi_1, \sigma, t'') \leq \epsilon$ . In summary,

$$\forall t' \in (t + I) \cap [0, \tau), (\rho_{\tau}(\varphi_2, \sigma, t') \leq \epsilon \vee \inf_{t'' \in [t, t']} \rho_{\tau}(\varphi_1, \sigma, t'') \leq \epsilon).$$

Since  $\forall t' \in t + I \cap [0, \tau), (\min(\rho_{\tau}(\varphi_2, \sigma, t'), \inf_{t'' \in [t, t']} \rho_{\tau}(\varphi_1, \sigma, t'')) \leq \epsilon)$

$$\sup_{t' \in t + I \cap [0, \tau)} \min(\rho_{\tau}(\varphi_2, \sigma, t'), \inf_{t'' \in [t, t']} \rho_{\tau}(\varphi_1, \sigma, t'')) \leq \epsilon.$$

Therefore,  $\rho_{\tau}(\varphi, \sigma, t) \leq \epsilon$  by definition.

Suppose  $\sigma, t \models_{\tau} (\varphi_1 \mathbf{U}_I \varphi_2)^{-\epsilon}$ . Then, there is a time point  $t' \in (t + I) \cap [0, \tau)$  such that  $\sigma, t' \models_{\tau} \varphi_2^{-\epsilon} \wedge \forall t'' \in [t, t'], \sigma, t'' \models_{\tau} \varphi_1^{-\epsilon}$  by definition. Then,

$\rho_\tau(\varphi_2, \sigma, t') \geq -\epsilon \wedge \forall t'' \in [t, t'], \rho_\tau(\varphi_1, \sigma, t'') \geq -\epsilon$  by induction hypothesis. If  $\forall t'' \in [t, t'], \rho_\tau(\varphi_1, \sigma, t'') \geq -\epsilon$ , then  $\inf_{t'' \in [t, t']} \rho_\tau(\varphi_1, \sigma, t'') \geq -\epsilon$ . In summary,

$$\exists t' \in (t + I) \cap [0, \tau), (\rho_\tau(\varphi_2, \sigma, t') \geq -\epsilon \wedge \inf_{t'' \in [t, t']} \rho_\tau(\varphi_1, \sigma, t'') \geq -\epsilon).$$

Since  $\exists t' \in t + I \cap [0, \tau)$ ,  $\min(\rho_\tau(\varphi_2, \sigma, t'), \inf_{t'' \in [t, t']} \rho_\tau(\varphi_1, \sigma, t'')) \geq -\epsilon$ ,

$$\sup_{t' \in t + I \cap [0, \tau)} \min(\rho_\tau(\varphi_2, \sigma, t'), \inf_{t'' \in [t, t']} \rho_\tau(\varphi_1, \sigma, t'')) \geq -\epsilon.$$

Therefore,  $\rho_\tau(\varphi, \sigma, t) \geq -\epsilon$  by definition.  $\square$

**Lemma 2.** For a signal  $\sigma$  and an STL property  $\varphi$ :

$$\sigma, t \models_\tau \varphi^{+\epsilon} \Rightarrow \rho_\tau(\varphi, \sigma, t) \geq \epsilon \quad \text{and} \quad \sigma, t \not\models_\tau \varphi^{-\epsilon} \Rightarrow \rho_\tau(\varphi, \sigma, t) \leq -\epsilon$$

*Proof.* The proof is by structural induction on  $\varphi$ .

For  $\varphi = p$ , if  $\sigma, t \models_\tau p^{+\epsilon}$ , then  $p^{+\epsilon}(\sigma(t)) = \top$  by definition. If  $\sigma, t \not\models_\tau p^{-\epsilon}$ , then  $p^{-\epsilon}(\sigma(t)) = \perp$  by definition. There are two possible state propositions forms,  $f(\vec{x}) \geq 0$  and  $f(\vec{x}) > 0$ . For  $p = f(\vec{x}) \geq 0$ ,  $\epsilon$ -strengthening  $p^{+\epsilon}$  of  $p$  is  $f(\vec{x}) \geq \epsilon$  and  $\epsilon$ -weakening  $p^{-\epsilon}$  of  $p$  is  $f(\vec{x}) \geq -\epsilon$  by definition.

Suppose  $\sigma, t \models_\tau p^{+\epsilon}$ . Then,  $f(\sigma(t)) \geq \epsilon$  is satisfied. The robustness degree  $\rho_\tau(f(\vec{x}) \geq 0, \sigma, t)$  is  $f(\sigma(t))$  by definition. Therefore,  $\rho_\tau(f(\vec{x}) \geq 0, \sigma, t) \geq \epsilon$ .

Suppose  $\sigma, t \not\models_\tau p^{-\epsilon}$ . Then,  $f(\sigma(t)) < -\epsilon$  is satisfied. The robustness degree  $\rho_\tau(f(\vec{x}) \geq 0, \sigma, t)$  is  $f(\sigma(t))$  by definition. Therefore,  $\rho_\tau(f(\vec{x}) \geq 0, \sigma, t) < -\epsilon$ .

If  $\rho_\tau(f(\vec{x}) \geq 0, \sigma, t) < -\epsilon$ , then  $\rho_\tau(f(\vec{x}) \geq 0, \sigma, t) \leq -\epsilon$ .

For  $p = f(\vec{x}) > 0$ ,  $\epsilon$ -strengthening  $p^{+\epsilon}$  of  $p$  is  $f(\vec{x}) > \epsilon$  and  $\epsilon$ -weakening  $p^{-\epsilon}$  of  $p$  is  $f(\vec{x}) > -\epsilon$  by definition. Suppose  $\sigma, t \models_\tau p^{+\epsilon}$ . Then,  $f(\sigma(t)) > \epsilon$  is satisfied. Thus,  $\rho_\tau(f(\vec{x}) > 0, \sigma, t) > \epsilon$ , since  $\rho_\tau(f(\vec{x}) > 0, \sigma, t) = f(\sigma(t))$  by definition. If  $\rho_\tau(f(\vec{x}) > 0, \sigma, t) > \epsilon$ , then  $\rho_\tau(f(\vec{x}) > 0, \sigma, t) \geq \epsilon$ .

Suppose  $\sigma, t \not\models_\tau p^{-\epsilon}$ . Then,  $f(\sigma(t)) \leq -\epsilon$  is satisfied. Thus,  $\rho_\tau(f(\vec{x}) > 0, \sigma, t) \leq -\epsilon$ , since  $\rho_\tau(f(\vec{x}) > 0, \sigma, t) = f(\sigma(t))$ .

For  $\varphi = \neg\phi$ , suppose  $\sigma, t \models_\tau (\neg\phi)^{+\epsilon}$ . Then,  $\sigma, t \not\models_\tau \phi^{-\epsilon}$  by definition, since  $(\neg\varphi)^{+\epsilon} \equiv \neg(\varphi^{-\epsilon})$ . If  $\sigma, t \not\models_\tau \phi^{-\epsilon}$  is satisfied, then  $\rho_\tau(\phi, \sigma, t) \leq -\epsilon$  by induction hypothesis. Thus,  $\rho_\tau(\phi, \sigma, t) \leq -\epsilon$  iff  $\rho_\tau(\neg\phi, \sigma, t) \geq \epsilon$ , since  $\rho_\tau(\neg\phi, \sigma, t) = -\rho_\tau(\phi, \sigma, t)$  by definition.

Suppose  $\sigma, t \not\models_\tau (\neg\phi)^{-\epsilon}$ . Then,  $\sigma, t \models_\tau \phi^{+\epsilon}$ , since  $(\neg\phi)^{-\epsilon} \equiv \neg(\phi^{+\epsilon})$  by definition. If  $\sigma, t \models_\tau \phi^{+\epsilon}$  is satisfied, then  $\rho_\tau(\phi, \sigma, t) \geq \epsilon$  by induction hypothesis.  $\rho_\tau(\phi, \sigma, t) \geq \epsilon$  iff  $\rho_\tau(\neg\phi, \sigma, t) \leq -\epsilon$ , since  $\rho_\tau(\neg\phi, \sigma, t) = -\rho_\tau(\phi, \sigma, t)$ .

For  $\varphi = \varphi_1 \wedge \varphi_2$ , suppose  $\sigma, t \models_\tau (\varphi_1 \wedge \varphi_2)^{+\epsilon}$ . Then,  $\sigma, t \models_\tau \varphi_1^{+\epsilon} \wedge \sigma, t \models_\tau \varphi_2^{+\epsilon}$  by definition. Thus, by induction hypothesis,

$$\rho_\tau(\varphi_1, \sigma, t) \geq \epsilon \wedge \rho_\tau(\varphi_2, \sigma, t) \geq \epsilon.$$

Therefore,  $\rho_\tau(\varphi_1 \wedge \varphi_2, \sigma, t) = \min(\rho_\tau(\varphi_1, \sigma, t), \rho_\tau(\varphi_2, \sigma, t)) \geq \epsilon$ .



Suppose  $\sigma, t \not\models_{\tau} (\varphi_1 \wedge \varphi_2)^{-\epsilon}$ . Then,  $\sigma, t \not\models_{\tau} \varphi_1^{-\epsilon} \vee \sigma, t \not\models_{\tau} \varphi_2^{-\epsilon}$  by definition. Thus,  $\rho_{\tau}(\varphi_1, \sigma, t) \leq -\epsilon \vee \rho_{\tau}(\varphi_2, \sigma, t) \leq -\epsilon$  by induction hypothesis. Then,  $\min(\rho_{\tau}(\varphi_1, \sigma, t), \rho_{\tau}(\varphi_2, \sigma, t)) \leq -\epsilon$ . Therefore,  $\rho_{\tau}(\varphi_1 \wedge \varphi_2) \leq -\epsilon$ .

For  $\varphi = \varphi_1 \mathbf{U}_I \varphi_2$ , suppose  $\sigma, t \models_{\tau} (\varphi_1 \mathbf{U}_I \varphi_2)^{+\epsilon}$ . Then, there is a time point  $t' \in (t + I) \cap [0, \tau)$  such that  $\sigma, t' \models_{\tau} \varphi_2^{+\epsilon} \wedge \forall t'' \in [t, t'], \sigma, t'' \models_{\tau} \varphi_1^{+\epsilon}$  by definition. Therefore, by induction hypothesis,

$$\rho_{\tau}(\varphi_2, \sigma, t') \geq \epsilon \wedge \forall t'' \in [t, t'], \rho_{\tau}(\varphi_1, \sigma, t'') \geq \epsilon.$$

If  $\forall t'' \in [t, t'], \rho_{\tau}(\varphi_1, \sigma, t'') \geq \epsilon$ , then  $\inf_{t'' \in [t, t']} \rho_{\tau}(\varphi_1, \sigma, t'') \geq \epsilon$ . In summary,

$$\exists t' \in (t + I) \cap [0, \tau), (\rho_{\tau}(\varphi_2, \sigma, t') \geq \epsilon \wedge \inf_{t'' \in [t, t']} \rho_{\tau}(\varphi_1, \sigma, t'') \geq \epsilon).$$

Since  $\exists t' \in t + I \cap [0, \tau)$ ,  $\min(\rho_{\tau}(\varphi_2, \sigma, t'), \inf_{t'' \in [t, t']} \rho_{\tau}(\varphi_1, \sigma, t'')) \geq \epsilon$ ,

$$\sup_{t' \in t + I \cap [0, \tau)} \min(\rho_{\tau}(\varphi_2, \sigma, t'), \inf_{t'' \in [t, t']} \rho_{\tau}(\varphi_1, \sigma, t'')) \geq \epsilon.$$

Therefore,  $\rho_{\tau}(\varphi, \sigma, t) \geq \epsilon$  by definition.

Suppose  $\sigma, t \not\models_{\tau} (\varphi_1 \mathbf{U}_I \varphi_2)^{-\epsilon}$ . Then, for all time points  $t' \in (t + I) \cap [0, \tau)$ ,  $\sigma, t' \not\models_{\tau} \varphi_2^{-\epsilon} \vee \exists t'' \in [t, t'], \sigma, t'' \not\models_{\tau} \varphi_1^{-\epsilon}$  by definition. Therefore,

$$\forall t' \in (t + I) \cap [0, \tau), (\rho_{\tau}(\varphi_2, \sigma, t') \leq -\epsilon \vee \exists t'' \in [t, t'], \rho_{\tau}(\varphi_1, \sigma, t'') \leq -\epsilon)$$

by induction hypothesis. If  $\exists t'' \in [t, t'], \rho_{\tau}(\varphi_1, \sigma, t'') \leq -\epsilon$ , then

$$\inf_{t'' \in [t, t']} \rho_{\tau}(\varphi_1, \sigma, t'') \leq -\epsilon. \text{ In summary,}$$

$$\forall t' \in (t + I) \cap [0, \tau), (\rho_{\tau}(\varphi_2, \sigma, t') \leq -\epsilon \vee \inf_{t'' \in [t, t']} \rho_{\tau}(\varphi_1, \sigma, t'') \leq -\epsilon).$$

Since  $\forall t' \in t + I \cap [0, \tau)$ ,  $(\min(\rho_{\tau}(\varphi_2, \sigma, t'), \inf_{t'' \in [t, t']} \rho_{\tau}(\varphi_1, \sigma, t'')) \leq -\epsilon)$ ,

$$\sup_{t' \in t + I \cap [0, \tau)} \min(\rho_{\tau}(\varphi_2, \sigma, t'), \inf_{t'' \in [t, t']} \rho_{\tau}(\varphi_1, \sigma, t'')) \leq -\epsilon.$$

Therefore,  $\rho_{\tau}(\varphi, \sigma, t) \leq -\epsilon$  by definition.  $\square$

According to above lemmas, we can reduce finding a counterexample for robust STL model checking of an STL formula  $\varphi$  into finding a counterexample for Boolean STL model checking of the  $\epsilon$ -strengthening  $\varphi^{+\epsilon}$ .

**Theorem 2.** *For a hybrid automaton  $H$ , an STL formula  $\varphi$ , a time  $t$ , a time bound  $\tau$ , and an arbitrary positive real value  $\epsilon$ , the following properties are hold:*

- (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  $\forall \sigma \in H. \rho_{\tau}(\varphi, \sigma, t) \geq \epsilon$ .

*Proof.* (1) Suppose there is a trajectory  $\sigma \in H$ .  $\sigma, t \models_{\tau} \neg(\varphi^{+\epsilon})$ . Then, the robustness degree  $\rho_{\tau}(\varphi, \sigma, t) \leq \epsilon$  by Lemma 1. If the robustness degree  $\rho_{\tau}(\varphi, \sigma, t)$  is less than or equal to  $\epsilon$ , then  $-\rho_{\tau}(\varphi, \sigma, t) \geq -\epsilon$ . It is equivalent to  $\rho_{\tau}(\neg\varphi, \sigma, t) \geq -\epsilon$ , since  $\rho_{\tau}(\neg\varphi, \sigma, t) = -\rho_{\tau}(\varphi, \sigma, t)$  by definition.

(2) Suppose all trajectories  $\forall \sigma \in H$ .  $\sigma, t \not\models_{\tau} \neg(\varphi^{+\epsilon})$ . The formula  $\sigma, t \not\models_{\tau} \neg(\varphi^{+\epsilon})$  is equivalent to  $\sigma, t \models_{\tau} \varphi^{+\epsilon}$  by definition. If a trajectory satisfies  $\varphi^{+\epsilon}$  at time  $t$ , then the robustness degree  $\rho_{\tau}(\varphi, \sigma, t) \geq \epsilon$  by Lemma 2.  $\square$

As a consequence, a counterexample of  $\varphi^{+\epsilon}$  for Boolean STL model checking is also a counterexample for robust STL model checking of  $\varphi$  with respect to a robustness threshold  $\epsilon$ . If there is no counterexample of  $\varphi^{+\epsilon}$  for Boolean STL model checking, then it is guaranteed that the robustness degree of  $\varphi$  is always greater than or equal to  $\epsilon$ . Therefore, for any positive robustness threshold  $0 < \epsilon' < \epsilon$ ,  $\varphi$  is satisfied at  $t$  in  $H$  with respect to  $\epsilon'$ . It is worth noting that  $\varphi$  may not be satisfied with respect to  $\epsilon$  itself.

## 4.2 Boolean STL Model Checking Algorithm

For Boolean STL model checking, there exist *refutationally complete* bounded model checking algorithms [4, 31] with two bound parameters:  $\tau$  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  $\varphi$ 's subformula changes at  $t$ . The algorithms build an SMT encoding  $\Psi_{H, \neg\varphi}^{N, \tau}$  of Boolean STL model checking:

**Theorem 3.** [4, 31]  $\Psi_{H, \neg\varphi}^{N, \tau}$  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  $\Psi$  can be precisely determined using standard SMT solvers, including Z3 [13] and Yices2 [18]. For ODE dynamics, the satisfiability of  $\Psi$  is undecidable in general, but there exist specialized solvers, such as dReal [24] and iSAT-ODE [19], that can approximately determine the satisfiability.

To support various SMT solvers, the implementation of STL<sub>MC</sub> utilizes a generic wrapper interface based on the SMT-LIB standard [6]. Therefore, if it follows SMT-LIB, a new SMT solver can be easily integrated with our tool. Moreover, STL<sub>MC</sub> 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  $\Psi$  includes universal quantification over time, e.g., because of invariant conditions. Several SMT solvers (including Z3 and Yices2) support these  $\exists\forall$ -conditions but at high computational costs [29]. For polynomial dynamics, we implement the encoding method [11] to simplify  $\exists\forall$ -conditions to quantifier-free formulas. For ODE dynamics, dReal natively supports  $\exists\forall$ -conditions [25].

## 4.3 Two-Step Solving Algorithm

The computation cost of dReal for ODEs is highly expensive. To reduce the complexity, we propose a two-step SMT solving algorithm, summarized in Algorithm 1, inspired by the lazy SMT solving approach [40]:

---

**Algorithm 1: Two-Step SMT Solving Algorithm**


---

**Input:** Hybrid automaton  $H$ , STL formula  $\varphi$ , threshold  $\epsilon$ , bounds  $\tau$  and  $N$

```

1 for  $k = 1$  to  $N$  do
2    $\bar{\Psi} \leftarrow$  abstraction of the encoding  $\Psi_{H, \neg(\varphi+\epsilon)}^{k, \tau}$  without flow and inv;
3   while checkSat( $\bar{\Psi}$ ) is Sat do
4      $\pi \leftarrow$  a minimal satisfying scenario;
5      $\hat{\pi} \leftarrow$  the refinement of  $\pi$  with flow and inv;
6     if checkSat( $\hat{\pi}$ ) is Sat then
7       return counterexample(result.satAssignment);
8      $\bar{\Psi} \leftarrow \bar{\Psi} \wedge \neg\pi$ ;
9 return True;
```

---

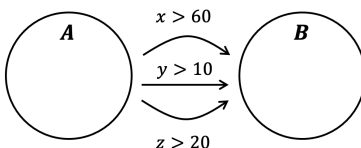


Fig. 8: Jump conditions between two modes

1. We obtain the *discrete abstraction* of the encoding  $\Psi$  by substituting the flow and invariant conditions with Boolean variables. We then enumerate a satisfying *scenario*  $\pi$ , a conjunction of literals, where  $\pi$  implies  $\Psi$ .
2. For each scenario  $\pi$ , 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.

A naive implementation of the two-step solving algorithm may generate many redundant scenarios for hybrid automata.

For example, in Figure 8, suppose there are two modes A and B, jump conditions from A to B are  $x > 60$ ,  $y > 10$ , and  $z > 20$ . There are 7 possible scenarios from these conditions for a single jump. However, when one jump, say  $x > 60$ , is taken, the guards for the other jumps, i.e.,  $y > 10$  and  $z > 20$ , are not important. It suffices to consider only 3 scenarios, namely,  $x > 60$ ,  $y > 10$ , and  $z > 20$ .

We implement a simple algorithm to minimize scenarios. A conjunction of literals  $\pi = l_1 \wedge \dots \wedge l_m$  is a minimal scenario, if  $\pi' = l_1 \wedge \dots \wedge l_{i-1} \wedge l_{i+1} \wedge \dots \wedge l_m$  is not a scenario for any  $i$ . The conjunction  $\pi'$  is not a scenario when  $\pi' \rightarrow \Psi$  is not valid. It means  $\pi$  is the minimal scenario, if  $\pi' \wedge \neg\Psi$  is satisfied.

To minimize a scenario  $\pi$ , we apply a dual propagation approach [35]. Because  $\pi$  implies  $\Psi$ , the formula  $\pi \wedge \neg\Psi$  is unsatisfiable. A conjunction of literals in the unsatisfiable core of  $\pi \wedge \neg\Psi$  is a minimal scenario. We evaluate the unsatisfiable core of  $\pi \wedge \neg\Psi$  using Z3. To obtain the unsatisfiable core, we encode the negation of the encoding  $\Psi$  in a separate dual solver under the assumptions of a set of literals in  $\pi$  using *assert\_and\_track* in Z3. To get a minimal unsatisfiable core, we use the option *smt.core.minimize true* for Z3.

#### 4.4 Implementation

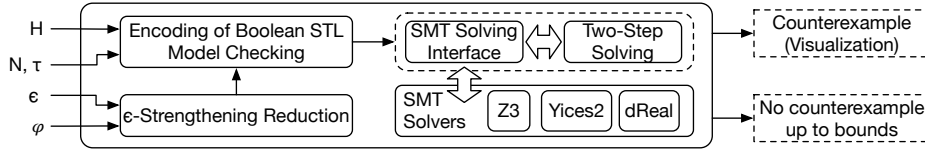


Fig. 9: The STLMC architecture

Figure 9 shows the overall architecture of STLMC. The given STL formula  $\varphi$  is first translated into the  $\epsilon$ -strengthening of the negated formula  $\neg(\varphi^{+\epsilon})$ . The SMT encoding  $\Psi_{H, \neg(\varphi^{+\epsilon})}^{k, \tau}$  for  $1 \leq k \leq N$  is then built using the STL bounded model checking algorithm [4, 31]. The satisfiability of  $\Psi_{H, \neg(\varphi^{+\epsilon})}^{k, \tau}$  can be checked directly using an SMT solver or using the two-step optimization algorithm. Our tool is implemented in around 9,500 lines of Python code.

*Simplifying Universal Quantification.* The encoding  $\Psi_{H, (\neg\varphi)^{-\epsilon}}^{k, \tau}$  includes universal quantification over time. Such  $\exists\forall$ -conditions are supported by several solvers but involve high computational costs. STLMC implements the following methods to simplify universal quantification. For polynomial hybrid automata, we use the quantifier-free encoding [11] to encode  $\exists\forall$ -conditions as quantifier-free formulas. For ODE dynamics, dReal natively supports  $\exists\forall$ -conditions [25]. For invariant STL properties of the form  $p \rightarrow \Box_I q$ , we reduce redundant universal quantifiers by reducing the model checking problem into the reachability [31].

*Parallelizing Two-step Solving.* We parallelize the two-step solving algorithm for ODEs. As mentioned, the first phase generates abstract scenarios, and the second phase checks the feasibility of the scenarios. We simply run the feasibility checking of different scenarios in parallel. If any of such scenario is satisfied, then a counterexample is found and all other jobs are terminated. If all scenarios, checking in parallel, are unsatisfiable, then there is no counterexample. As shown in Sec 5, it greatly improves the performance for the ODE cases in practice.

*Supporting Various SMT Solvers.* We implement a generic wrapper interface based on the SMT-LIB standard [6] to support various SMT solvers. Therefore, it is easy to extend our tool with a new SMT solvers, if it follows SMT-LIB. 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. Given an input model, STLMC can detect the most suitable solvers. If the model has ODE dynamics and Z3 is selected, the tool raises an exception. Also, when the model has linear dynamics and the dReal is chosen, STLMC gives a warning to use Z3 or Yices instead of dReal.

## 5 Experimental Evaluation

We evaluate the effectiveness of the STLMC model checker using a number of hybrid system benchmarks and nontrivial STL properties. We use the following models, adapted from existing benchmarks [3, 7, 20, 22, 27, 36]: 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**), docking of spacecraft (**Space**), navigation of a vehicle (**Nav**), and a filtered oscillator (**Oscil**). The models and the experimental results are available at <https://doi.org/10.5281/zenodo.6620846>.

Section 5.1 shows the experiments on robust STL model checking. Section 5.2 compares the performance of STLMC for invariant properties with the existing reachability analysis tools for hybrid automata. We have run all experiments on Intel Xeon 2.8GHz with 256 GB memory.

### 5.1 STL Model Checking of Hybrid Automata

We measure the SMT encoding size and execution time for robust STL model checking. We consider three variants of continuous dynamics (linear, polynomial, and ODE dynamics) for the five models: **Bat**, **Wat**, **Car**, **Rail**, and **Thm**. We also consider the additional three models with ODE dynamics: **Oscil**, **Space**, and **Nav**. For each model, we use three STL formulas with nested temporal operators.

For discrete bounds, we use  $N = 20$  for linear models,  $N = 10$  for polynomial models, and  $N = 5$  for ODE models. We use different time bounds  $\tau$  and robustness thresholds  $\epsilon$  for different models, since  $\tau$  and  $\epsilon$  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-step SMT solving. We set a timeout of 60 minutes.

The experimental results for linear and polynomial models are summarized in Table 2 and the experimental results for ODE models are summarized in Table 3, where  $\tau$  denotes time bounds, and  $|\Psi|$  denotes the size of the SMT encoding  $\Psi$  (in thousands) as the number of connectives in  $\Psi$ . For the model checking results,  $\top$  indicates that the tool found no counterexample up to bound  $N$ , and  $\perp$  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 performance. For the **2-step** case, we also write the number of minimal scenarios generated ( $\#\pi$ ).

As shown in Table 2 and 3, 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 minutes. Therefore, the two algorithms implemented in our tool are complementary.

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

Dyn.	Model	STL formula	$\tau$	$\epsilon$	$ \Psi $	Time (s)	Result	k	Alg.
Linear ( $N = 20$ )	Bat	$\diamond_{[4,10]}(p_1 \rightarrow \square_{[4,10]} p_2)$		0.1	12.9	119.5	$\top$	-	1-step
		$(\diamond_{[1,5]} p_1) \mathbf{R}_{[5,20]} p_2$	30	3.5	2.8	6.0	$\perp$	5	1-step
		$\square_{[4,14]}(p_1 \rightarrow \diamond_{[0,10]} p_2)$		0.1	3.8	44.6	$\perp$	8	1-step
	Wat	$\square_{[1,3]}(p_1 \mathbf{R}_{[1,10]} p_2)$		2.5	18.8	25.1	$\top$	-	1-step
		$(\diamond_{[1,10]} p_1) \mathbf{U}_{[2,5]} p_2$	20	0.1	1.9	4.3	$\perp$	4	1-step
		$\diamond_{[4,10]}(p_1 \rightarrow \square_{[2,5]} p_2)$		0.01	11.2	16.3	$\top$	-	1-step
	Car	$(\diamond_{[3,5]} p_1) \mathbf{U}_{[2,10]} p_2$		0.1	2.5	7.6	$\perp$	5	1-step
		$\diamond_{[4,14]}(\square_{[5,15]} p_1)$	40	0.5	10.8	559.2	$\top$	-	1-step
		$(\square_{[2,5]} p_1) \mathbf{R}_{[0,10]} p_2$		1.0	2.5	7.8	$\perp$	5	1-step
	Rail	$\diamond_{[3,10]}(p_1 \mathbf{U}_{[0,10]} p_2)$		2.0	17.9	21.3	$\top$	-	1-step
		$(\square_{[2,10]} p_1) \mathbf{R}_{[0,15]} p_2$	30	1.0	1.2	3.8	$\perp$	3	1-step
		$\square_{[0,5]}(\diamond_{[0,10]} p_1)$		0.5	0.8	2.7	$\perp$	3	1-step
	Thm	$\diamond_{[2,5]}(p_1 \mathbf{R}_{[2,10]} p_2)$		3.0	16.8	20.2	$\top$	-	1-step
		$\square_{[1,4]}(p_1 \rightarrow \diamond_{[5,15]} p_2)$	30	1.0	11.1	14.6	$\top$	-	1-step
		$p_1 \mathbf{U}_{[10,20]}(\square_{[3,5]} p_2)$		2.5	1.0	3.7	$\perp$	2	1-step
Poly ( $N = 10$ )	Bat	$\square_{[3,5]}(p_1 \rightarrow \diamond_{[6,10]} p_2)$		2.5	1.7	4.7	$\perp$	4	1-step
		$(\square_{[4,14]} p_1) \mathbf{U}_{[4,10]} p_2$	25	1.0	2.4	5.0	$\perp$	4	1-step
		$\diamond_{[0,15]}(p_1 \mathbf{U}_{[0,6]} p_2)$		0.5	7.2	185.2	$\top$	-	1-step
	Wat	$\square_{[1,4]}(p_1 \mathbf{U}_{[2,5]} p_2)$		1.5	1.0	2.6	$\perp$	2	1-step
		$\diamond_{[0,3]}(p_1 \rightarrow \square_{[3,4]} p_2)$	10	2.0	4.1	5.4	$\top$	-	1-step
		$(\diamond_{[0,4]} p_1) \mathbf{R}_{[3,5]} p_2$		4.0	0.9	2.6	$\perp$	2	1-step
	Car	$\square_{[0,4]}(p_1 \rightarrow \diamond_{[2,5]} p_2)$		0.5	2.2	5.5	$\perp$	5	1-step
		$(\diamond_{[0,4]} p_1) \mathbf{U}_{[0,5]} p_2$	15	2.0	1.7	4.7	$\perp$	3	1-step
		$\diamond_{[0,3]}(p_1 \mathbf{U}_{[0,5]} p_2)$		0.1	7.3	7.7	$\top$	-	1-step
	Rail	$\diamond_{[0,5]}(p_1 \mathbf{U}_{[1,8]} p_2)$		1.0	2.3	3.0	$\perp$	5	1-step
		$\diamond_{[0,4]}(p_1 \rightarrow \square_{[2,10]} p_2)$	20	5.0	3.8	3.8	$\top$	-	1-step
		$(\square_{[0,5]} p_1) \mathbf{U}_{[2,10]} p_2$		4.0	1.9	2.7	$\perp$	4	1-step
	Thm	$(\square_{[2,10]} p_1) \mathbf{U}_{[1,4]} p_2$		0.5	2.0	4.4	$\perp$	4	1-step
		$\diamond_{[0,5]}(p_1 \rightarrow \square_{[2,5]} p_2)$	10	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

Table 3: Robust Bounded Model Checking of STL ( $N = 5$  and time in seconds)

Model	STL formula	$\tau$	$\epsilon$	$ \Psi $	Time (s)	Result	k	Alg.	$\#\pi$
Bat	$\diamond_{[0,4]}(p_1 \mathbf{R}_{[0,5]} p_2)$		1.0	1.3	19.2	$\perp$	3	2-step	78
	$\square_{[3,4]}(p_1 \rightarrow \diamond_{[0,6]} p_2)$	10	1.5	0.9	28.0	$\perp$	3	2-step	151
	$\square_{[1,3]}(\diamond_{[0,5]} p_1)$		1.0	0.8	12.7	$\perp$	3	2-step	80
Thm	$\diamond_{[0,15]}(p_1 \mathbf{U}_{[0,\infty]} p_2)$		0.5	1.2	818.7	$\top$	-	2-step	3,580
	$\square_{[2,4]}(p_1 \rightarrow \diamond_{[3,10]} p_2)$	25	2.0	0.7	14.7	$\perp$	2	2-step	91
	$\square_{[0,10]}(p_1 \mathbf{R}_{[0,\infty]} p_2)$		2.0	1.2	161.7	$\perp$	4	2-step	279
Oscil	$\diamond_{[0,3]}(p_1 \mathbf{R}_{[0,\infty]} p_2)$		0.1	1.5	108.9	$\top$	-	2-step	326
	$\diamond_{[2,5]}(\square_{[0,3]} p_1)$	8	1.0	1.2	192.8	$\perp$	3	2-step	601
	$(\square_{[1,3]} p_1) \mathbf{R}_{[2,5]} p_2$		0.1	1.8	112.1	$\perp$	3	2-step	258
Space	$\square_{[0,2]}(p_1 \rightarrow \diamond_{[0,3]} p_2)$		1.5	0.8	278.3	$\perp$	2	2-step	79
	$\diamond_{[2,3]}(\square_{[1,2]} p_1)$	5	0.1	1.1	37.0	$\perp$	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
Nav	$\diamond_{[2,4]}(p_1 \rightarrow \square_{[1,5]} p_2)$		3.0	1.2	399.3	$\perp$	3	2-step	1,388
	$\diamond_{[2,4]}(\square_{[3,6]} p_1)$	10	2.0	1.1	332.2	$\perp$	3	2-step	1,213
	$\diamond_{[1,5]}(p_1 \mathbf{R} p_2)$		1.0	1.4	749.6	$\top$	-	2-step	2,411
Wat	$p_1 \mathbf{R}_{[0,3]}(\diamond_{[2,4]} p_2)$		2.0	1.1	34.7	$\perp$	3	2-step	150
	$\square_{[1,3]}(p_1 \rightarrow \diamond_{[0,5]} p_2)$	8	0.1	0.5	3.3	$\perp$	2	2-step	27
	$(\diamond_{[0,4]} p_1) \mathbf{R}_{[0,\text{inf}]} p_2$		0.5	0.8	178.2	$\top$	-	2-step	947
Car	$\diamond_{[1,3]}(\square_{[0,2]} p_1)$		1.0	1.2	10.8	$\perp$	2	2-step	15
	$(\diamond_{[1,2]} p_1) \mathbf{R}_{[0,2]} p_2$	5	0.5	1.5	14.6	$\perp$	2	2-step	34
	$\square_{[0,2]}(p_1 \rightarrow \diamond_{[2,3]} p_2)$		2.0	1.2	45.6	$\perp$	2	2-step	162
Rail	$\square_{[1,3]}(p_1 \rightarrow \diamond_{[2,4]} p_2)$		2.0	0.8	51.2	$\perp$	3	2-step	240
	$(\diamond_{[1,3]} p_1) \mathbf{R}_{[0,\text{inf}]} p_2$	8	0.1	0.9	856.4	$\top$	-	2-step	3,391
	$\diamond_{[0,4]}(\square_{[2,4]} p_1)$		1.0	0.6	64.6	$\perp$	2	2-step	50

## 5.2 Comparison with Reachability Analysis Tools

We compare the performance of STLMC for invariant properties with four reachability analysis tools for hybrid automata: HyComp [10], SpaceEx [22], Flow\* [9], and dReach [30]. For each model, we consider a true invariant property so that reachability analysis explores all possible behaviors up to given bounds. We measure the execution times (in seconds) for analyzing the invariant properties with a timeout of 60 minutes.

Since each tool has a different notion of bounds, we use the number of discrete jumps  $N$  and the maximum time horizon  $\tau$  as the common bound parameters (which are "encoded" in the models) if needed. For STLMC, we use direct SMT solving (1-step) and Yices as an underlying SMT solver for linear and polynomial models. For ODE models, we use two-step SMT solving (2-step) and dReal. For SpaceEx, we use PHAVer for linear models, and STC for polynomial and ODE models. For Flow\*, we use adaptive steps and TM orders 1 (for linear),

Table 4: Execution time for analyzing invariant properties (in seconds)

Model	Linear ( $N = 20$ )					Poly ( $N = 10$ )			
	STLmc	HyComp	SpaceEx	Flow*	dReach	STLmc	SpaceEx	Flow*	dReach
Bat	0.25	7.54	0.07	44	3.54	0.44	0.01	19.4	0.58
Wat	0.18	1.56	0.49	128	T/O	2.51	0.001	0.08	2116
Car	0.21	0.74	1.0	26.5	T/O	0.32	2607	27	41.7
Rail	0.14	0.77	0.14	3.91	1158	0.07	0.25	2.73	0.09
Thm	0.21	1.25	0.59	402	T/O	0.34	3.25	5.21	T/O

Table 5: Execution time (in seconds) for analyzing invariant properties ( $N = 5$ )

Model	STLmc	SpaceEx	Flow*	dReach	Model	STLmc	SpaceEx	Flow*	dReach
Bat	0.1	0.03	4.53	0.1	Nav	0.1	0.02	1.0	0.07
Thm	1.3	0.24	16.4	2.3	Wat	0.06		167	0.04
Oscil	0.07	0.02	0.48	0.17	Car	39.2	N/A	T/O	47.8
Space	0.06	0.02	0.42	2.6	Rail	0.1		0.08	0.08

2 (for polynomial) and [2, 15] (for ODE). For dReach, we set precision to 0.1. We use BMC for HyComp.

The experimental results are summarized in Table 4, with execution times in seconds. T/O denotes time out. HyComp only supports linear models, and so the results for polynomial and ODE models do not include HyComp. SpaceEx only supports linear ODE, and so is not applicable (N/A) for models with nonlinear ODEs (Car, Rail, and Wat). As seen, STLmc has comparable performance to the other tools for the invariant properties.

## 6 Related Work.

There exist many tools for falsifying STL properties of hybrid systems, including Breach [15], S-talrio [2], and TLTK [12]. STL falsification techniques are based on STL monitoring [14, 34], and often use stochastic optimization techniques, such as Ant-Colony Optimization [2], Monte-Carlo tree search [44], deep reinforcement learning [43], 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 [17], Flow\* [9], Hylaa [5], and SpaceEx [22]. They can be used to guarantee the correctness of invariant properties of the form  $p \rightarrow \square_I q$ , 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 [30], HyComp [10], and HybridSAL [41]. 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  $\varphi$  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.

## References

1. Alur, R.: Principles of cyber-physical systems. The MIT Press (2015)
2. Annpureddy, Y., Liu, C., Fainekos, G., Sankaranarayanan, S.: S-taliro: A tool for temporal logic falsification for hybrid systems. In: Proc. TACAS. LNCS, vol. 6605, pp. 254–257. Springer (2011)
3. Bae, K., Gao, S.: Modular SMT-based analysis of nonlinear hybrid systems. In: Proc. FMCAD. pp. 180–187. IEEE (2017)
4. Bae, K., Lee, J.: Bounded model checking of signal temporal logic properties using syntactic separation. Proc. ACM Program. Lang. **3**, POPL(51), 1–30 (2019)
5. Bak, S., Duggirala, P.S.: Hylaa: A tool for computing simulation-equivalent reachability for linear systems. In: Proc. HSCC. pp. 173–178. ACM (2017)
6. Barrett, C., Fontaine, P., Tinelli, C.: The SMT-LIB Standard: Version 2.5. Tech. rep., Dept. of Computer Science, University of Iowa (2015), [www.SMT-LIB.org](http://www.SMT-LIB.org)
7. Chan, N., Mitra, S.: Verifying safety of an autonomous spacecraft rendezvous mission. In: Proc. ARCH. EPiC Series in Computing, vol. 48. EasyChair (2017)
8. Chen, G., Liu, M., Kong, Z.: Temporal-logic-based semantic fault diagnosis with time-series data from industrial internet of things. IEEE Transactions on Industrial Electronics **68**(5), 4393–4403 (2020)
9. Chen, X., Ábrahám, E., Sankaranarayanan, S.: Flow\*: An analyzer for non-linear hybrid systems. In: Proc. CAV. LNCS, vol. 8044, pp. 258–263. Springer (2013)
10. Cimatti, A., Griggio, A., Mover, S., Tonetta, S.: HyComp: an SMT-based model checker for hybrid systems. In: Proc. TACAS. LNCS, vol. 9035. Springer (2015)

11. Cimatti, A., Mover, S., Tonetta, S.: A quantifier-free SMT encoding of non-linear hybrid automata. In: Proc. FMCAD. pp. 187–195. IEEE (2012)
12. Cralley, J., Spantidi, O., Hoxha, B., Fainekos, G.: TLTK: A toolbox for parallel robustness computation of temporal logic specifications. In: Proc. RV. LNCS, vol. 12399, pp. 404–416. Springer (2020)
13. De Moura, L., Bjørner, N.: Z3: an efficient SMT solver. In: Proc. TACAS. LNCS, vol. 4963, pp. 337–340. Springer (2008)
14. Deshmukh, J.V., Donzé, A., Ghosh, S., Jin, X., Juniwal, G., Seshia, S.A.: Robust online monitoring of signal temporal logic. *Form. Methods Syst. Des.* **51**(1) (2017)
15. Donzé, A.: Breach, a toolbox for verification and parameter synthesis of hybrid systems. In: Proc. CAV. pp. 167–170. Springer (2010)
16. Donzé, A., Ferrère, T., Maler, O.: Efficient robust monitoring for STL. In: Proc. CAV. LNCS, vol. 8044. Springer (2013)
17. Duggirala, P.S., Mitra, S., Viswanathan, M., Potok, M.: C2E2: A verification tool for stateflow models. In: Proc. TACAS. pp. 68–82. Springer (2015)
18. Dutertre, B.: Yices 2.2. In: Biere, A., Bloem, R. (eds.) Proc. CAV. LNCS, vol. 8559, pp. 737–744. Springer (2014)
19. Eggers, A., Ramdani, N., Nediakov, N.S., Fränzle, M.: Improving the SAT modulo ODE approach to hybrid systems analysis by combining different enclosure methods. *Software & Systems Modeling* **14**(1), 121–148 (2015)
20. Fehnker, A., Ivančić, F.: Benchmarks for hybrid systems verification. In: Proc. HSCC. pp. 326–341. Springer (2004)
21. Fox, M., Long, D., Magazzeni, D.: Plan-based policies for efficient multiple battery load management. *JAIR* **44**, 335–382 (2012)
22. 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: Proc. CAV. LNCS, vol. 6806, pp. 379–395. Springer (2011)
23. 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)
24. Gao, S., Kong, S., Clarke, E.M.: dReal: An SMT solver for nonlinear theories over the reals. In: Proc. CADE. LNCS, vol. 7898, pp. 208–214. Springer (2013)
25. Gao, S., Kong, S., Clarke, E.M.: Satisfiability modulo ODEs. In: Proc. FMCAD. pp. 105–112. IEEE (2013)
26. Goldman, R.P., Bryce, D., Pelican, M.J., Musliner, D.J., Bae, K.: A hybrid architecture for correct-by-construction hybrid planning and control. In: Proc. NFM. LNCS, vol. 9690. Springer (2016)
27. Henzinger, T.: The theory of hybrid automata. In: *Verification of Digital and Hybrid Systems*, NATO ASI Series, vol. 170, pp. 265–292. Springer (2000)
28. Jin, X., Deshmukh, J.V., Kapinski, J., Ueda, K., Butts, K.: Powertrain control verification benchmark. In: Proc. HSCC. ACM (2014)
29. Jovanović, D., de Moura, L.: Solving non-linear arithmetic. In: Proc. IJCAR. pp. 339–354. Springer (2012)
30. Kong, S., Gao, S., Chen, W., Clarke, E.M.: dReach:  $\delta$ -reachability analysis for hybrid systems. In: Proc. TACAS. LNCS, vol. 7898, pp. 200–205. Springer (2015)
31. Lee, J., Yu, G., Bae, K.: Efficient SMT-based model checking for signal temporal logic. In: Proc. ASE. pp. 343–354. IEEE (2021)
32. Ma, M., Bartocci, E., Lifland, E., Stankovic, J., Feng, L.: SaSTL: spatial aggregation signal temporal logic for runtime monitoring in smart cities. In: Proc. ICCPS. pp. 51–62. IEEE (2020)
33. Maler, O., Nickovic, D.: Monitoring temporal properties of continuous signals. In: Proc. FORMATS. LNCS, vol. 3253, pp. 152–166. Springer (2004)

34. Ničković, D., Lebeltel, O., Maler, O., Ferrère, T., Ulus, D.: AMT 2.0: qualitative and quantitative trace analysis with extended signal temporal logic. In: Proc. TACAS. vol. 10806, pp. 303–319. Springer (2018)
35. Niemetz, A., Preiner, M., Biere, A.: Turbo-charging lemmas on demand with don't care reasoning. In: Proc. FMCAD. pp. 179–186. IEEE (2014)
36. 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: Hybrid systems V. LNCS, vol. 1567, pp. 279–303. Springer (1999)
37. Roehm, H., Oehlerking, J., Heinz, T., Althoff, M.: STL model checking of continuous and hybrid systems. In: Proc. ATVA. LNCS, vol. 9938. Springer (2016)
38. Roohi, N., Kaur, R., Weimer, J., Sokolsky, O., Lee, I.: Parameter invariant monitoring for signal temporal logic. In: Proc. HSCC. pp. 187–196. ACM (2018)
39. Sankaranarayanan, S., Fainekos, G.: Falsification of temporal properties of hybrid systems using the cross-entropy method. In: Proc. HSCC. pp. 125–134 (2012)
40. Sebastiani, R.: Lazy satisfiability modulo theories. *Journal on Satisfiability, Boolean Modeling and Computation* **3**(3-4), 141–224 (2007)
41. Tiwari, A.: HybridSAL relational abstracter. In: Proc. CAV. Lecture Notes in Computer Science, vol. 7358, pp. 725–731. Springer (2012)
42. 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)
43. Yamagata, Y., Liu, S., Akazaki, T., Duan, Y., Hao, J.: Falsification of cyber-physical systems using deep reinforcement learning. *IEEE Transactions on Software Engineering* **47**(12), 2823–2840 (2020)
44. 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: Proc. CAV. LNCS, vol. 12759, pp. 595–618. Springer (2021)

## A Benchmark Examples

### A.1 Load Management of Batteries

There are two fully charged batteries and a control system balancing the load between these batteries to achieve longer lifetime (adapted from [21]). The total charge  $g_i$  and kinetic energy  $d_i$  of the battery  $i = 1, 2$  change according to the controller's modes: *on* (On), *off* (Off), and *dead* (Dead). The dynamics for each battery is given as follows:

$$\dot{d}_i = \begin{cases} L/c - 0.5d_i & (\text{On}) \\ -0.5d_i & (\text{Off}) \\ 0 & (\text{Dead}) \end{cases} \quad \dot{g}_i = \begin{cases} -L & (\text{On}) \\ 0 & (\text{Off}) \\ 0 & (\text{Dead}) \end{cases}$$

where  $L$  is load of the battery and  $c$  is a threshold constant. If the charge  $g_i$  greater than  $(1 - c)d_i$ , the battery is dead. Otherwise, it can be either on or off. In this paper, we consider three battery models with different dynamics.

**Linear Dynamics.** In the case of linear dynamics, the amount of change in kinetic energy  $d_i$  and total charge  $g_i$  are constant as follows.

$$\begin{array}{lllll} \dot{d}_1 = 1, & \dot{g}_1 = -0.3, & \dot{d}_2 = 1, & \dot{g}_2 = -0.3 & (\text{On}_1\text{On}_2) \\ \dot{d}_1 = 0.8, & \dot{g}_1 = -0.5, & \dot{d}_2 = -0.166, & \dot{g}_2 = 0 & (\text{On}_1\text{Off}_2) \\ \dot{d}_1 = -0.166, & \dot{g}_1 = 0, & \dot{d}_2 = 0.8, & \dot{g}_2 = -0.5 & (\text{Off}_1\text{On}_2) \\ \dot{d}_1 = 0.7, & \dot{g}_1 = -7, & \dot{d}_2 = 0, & \dot{g}_2 = 0 & (\text{On}_1\text{Dead}_2) \\ \dot{d}_1 = 0, & \dot{g}_1 = 0, & \dot{d}_2 = 0.7, & \dot{g}_2 = -7 & (\text{Dead}_1\text{On}_2) \\ \dot{d}_1 = 0, & \dot{g}_1 = 0, & \dot{d}_2 = 0, & \dot{g}_2 = 0 & (\text{Dead}_1\text{Dead}_2) \end{array}$$

Figure 10 shows the hybrid automaton for the linear battery system. Table 6 shows three STL formulas for our battery system. Figure 11 is a model file of the load management of two batteries in STLMC.

Label	STL formula
f1:	$\diamond_{[4,10]}(d_2 \geq 4 \rightarrow \square_{[4,10]}(b_2 = 2))$
f2:	$(\diamond_{[1,5]} g_2 \leq 5) \mathbf{R}_{[5,20]}(d_2 \geq 4.5)$
f3:	$\square_{[4,14]}(g_2 > 4 \rightarrow \diamond_{[0,10]}(d_2 > 1))$

Table 6: STL properties for the linear battery

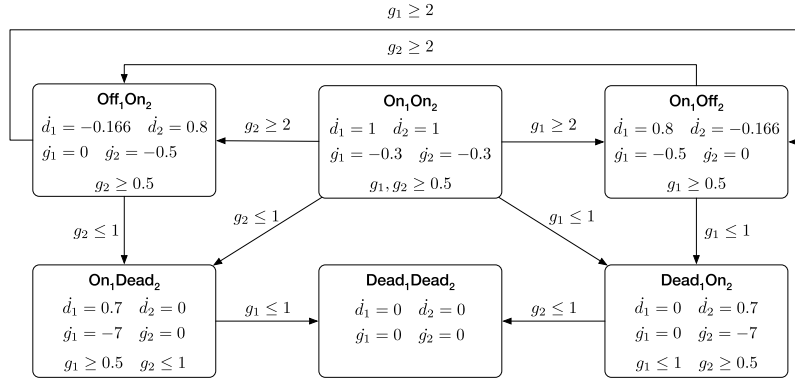


Fig. 10: A hybrid automaton of the battery system

**Polynomial Dynamics.** In the case of polynomial dynamics, the amount of change in kinetic energy  $d_i$  and total charge  $g_i$  are change as follows.

$$\begin{cases} d_i(t) = 0.1 * (6 - 2 * t + 2 * t^2) + d_i(0), g_i(t) = -0.5 * t + g_i(0) & (1) \text{ On} \\ d_i(t) = -0.1 * (1 - 2 * t + 2 * t^2) + d_i(0), g_i(t) = g_i(0) & (2) \text{ Off} \\ d_i(t) = 0, g_i(t) = g_i(0) & (3) \text{ Dead} \end{cases}$$

where  $t$  is time.

Table 7 shows three STL formulas for our battery system. Figure 12 is a model file of the load management of two batteries in STLMC.

Label	STL formula
f1:	$\square_{[3,5]}(g_2 \geq 3 \rightarrow \diamond_{[6,10]}(d_2 \geq 4))$
f2:	$(\diamond_{[4,14]} g_2 > 4) \mathbf{U}_{[4,10]}(d_2 > 1)$
f3:	$\diamond_{[0,15]}(d_1 \leq 2 \mathbf{U}_{[0,6]}(g_1 < 4))$

Table 7: STL properties for the polynomial battery

**ODE Dynamics.** We consider the load management with only one battery for ODE dynamic model with following dynamics.

$$\begin{aligned} \dot{d}_1 &= 2 + (0.122d_1), & \dot{g}_1 &= -0.5, & (\text{On}) \\ \dot{d}_1 &= -0.05d_1, & \dot{g}_1 &= 0, & (\text{Off}) \\ \dot{d}_1 &= 0, & \dot{g}_1 &= 0, & (\text{Dead}) \end{aligned}$$



```

real b1; real b2;
[0, 10] g1; [0, 10] g2;
[-30, 30] d1; [-30, 30] d2;
{ mode: b1 = 1;
  b2 = 1;
  inv: g1 > 2;
      g2 > 2;
  flow: d1(t) = 0.1 * (1 - 2 * t + 2 * t * t)
        + 0.5 + d1(0);
        d2(t) = 0.1 * (1 - 2 * t + 2 * t * t)
        + 0.5 + d2(0);
        g1(t) = -0.5 * t + g1(0);
        g2(t) = -0.5 * t + g2(0);
  jump: g1 > 1 => (and (b1' = 1) (b2' = 2)
                    (d2' = (d2 - 0.122 * g2))
                    (g1' = g1) (d1' = d1)
                    (g2' = g2));
        g2 > 1 => (and (b1' = 2) (b2' = 1)
                    (d2' = (d2 - 0.122 * g2))
                    (g1' = g1) (d1' = d1)
                    (g2' = g2));
}
{ mode: b1 = 1;
  b2 = 2;
  inv: g1 > 0.5;
      d2 > 1;
  flow: d1(t) = 0.1 * (1 - 2 * t + 2 * t * t)
        + 1 + d1(0);
        d2(t) = -0.1 * (1 - 2 * t + 2 * t * t)
        + d2(0);
        g1(t) = g1(0) - t;
        g2(t) = g2(0);
  jump: g2 > 1 => (and (b1' = 2) (b2' = 1)
                    (d2' = (d2 - 0.122 * g2))
                    (g1' = g1) (d1' = d1)
                    (g2' = g2));
        g1 <= 1 => (and (b1' = 3) (b2' = 1)
                    (d2' = (d2 - 0.122 * g2))
                    (g1' = g1) (d1' = d1)
                    (g2' = g2));
}
{ mode: b1 = 2;
  b2 = 1;
  inv: d1 > 1;
      g2 > 0.5;
  flow: d1(t) = -0.1 * (1 - 2 * t + 2 * t * t)
        + d1(0);
        d2(t) = 0.1 * (1 - 2 * t + 2 * t * t)
        + 1 + d2(0);
        g1(t) = g1(0);
        g2(t) = g2(0) - t;
  jump: g1 > 1 => (and (b1' = 1) (b2' = 2)
                    (d2' = (d2 - 0.122 * g2))
                    (g1' = g1) (d1' = d1)
                    (g2' = g2));
        g2 <= 1 => (and (b1' = 1) (b2' = 3)
                    (d2' = (d2 - 0.122 * g2))
                    (g1' = g1) (d1' = d1)
                    (g2' = g2));
}
}
}

(d2' = (d2 - 0.122 * g2))
(g1' = g1) (d1' = d1)
(g2' = g2));
}
{ mode: b1 = 3;
  b2 = 1;
  inv: g1 < 1;
      g2 > 0.5;
  flow: d1(t) = d1(0);
        d2(t) = 0.1 * (1 - 2 * t + 2 * t * t)
        + 1 + d2(0);
        g1(t) = g1(0);
        g2(t) = g2(0) - t;
  jump: g2 <= 1 => (and (b1' = 3) (b2' = 3)
                    (d2' = (d2 - 0.122 * g2))
                    (g1' = g1) (d1' = d1)
                    (g2' = g2));
}
{ mode: b1 = 1;
  b2 = 3;
  inv: g1 > 0.5;
      g2 < 1;
  flow: d1(t) = 0.1 * (1 - 2 * t + 2 * t * t)
        + 1 + d1(0);
        d2(t) = d2(0);
        g1(t) = g1(0) - t;
        g2(t) = g2(0);
  jump: g1 <= 1 => (and (b1' = 3) (b2' = 3)
                    (d2' = (d2 - 0.122 * g2))
                    (g1' = g1) (d1' = d1)
                    (g2' = g2));
}
}
{ mode: b1 = 3;
  b2 = 3;
  inv:
  flow: d/dt[d1] = 0;
        d/dt[d2] = 0;
        d/dt[g1] = 0;
        d/dt[g2] = 0;
  jump:
}
}

init:
b1 = 1; b2 = 1; 8.4 <= g1; g1 <= 8.5;
0 <= d1; d1 <= 0.1; 7.4 <= g2; g2 <= 7.5;
0 <= d2; d2 <= 0.1;

proposition:

# timebound 25
# threshold: f1=2.5, f2=1, f3=0.5
goal:
[f1]: [] [3, 5] (g2 >= 3 -> <> [6, 10] d2 >= 4);
[f2]: (<> [4, 14] g2 > 4) U [4, 10] d2 > 1;
[f3]: <> [0, 15] (d1 <= 2 U [0, 6] g1 < 4);
    
```

Fig. 12: The STLMC model for Bat with polynomial dynamics

Table 8 shows three STL formulas for our battery system. Figure 13 is a model file of the load management of two batteries in STLMC.

Label	STL formula
f1:	$\diamond_{[0,4]}(d_1 \geq 1 \mathbf{R}_{[0,6]}(d_1 \geq 3))$
f2:	$\square_{[3,4]}(g_1 < 6 \rightarrow \diamond_{[0,6]} d_1 \geq 3)$
f3:	$\square_{[1,3]}(\diamond_{[0,5]} d_1 > 1.5)$

Table 8: STL properties for the nonlinear battery

<pre> int b1; [-10, 10] g1; [-10, 10] d1; # state # 1 : ON # 2 : OFF # 3 : DEAD { mode: b1 = 1;   inv: g1 &gt;= 0;   flow: d/dt[d1] = 2 + (0.122 * d1) ;         d/dt[g1] = -0.5 ;   jump: g1 &gt; 1 =&gt; (and (b1' = 2)                       (d1' = d1) (g1' = g1));         g1 &lt;= 0.5 =&gt; (and (b1' = 3)                           (d1' = d1) (g1' = g1)); } { mode: b1 = 2;   inv: d1 &gt; 0;   flow: d/dt[d1] = -(0.05 * d1) ;         d/dt[g1] = 0 ;   jump: g1 &gt; 1 =&gt; (and (b1' = 1) (d1' = d1)                       (g1' = g1)); } </pre>	<pre> } { mode: b1 = 3;   inv:   flow: d/dt[d1] = 0 ;         d/dt[g1] = 0 ;   jump: }  init:   b1 = 1; 8 &lt;= g1; g1 &lt;= 8.5;   0 &lt;= d1; d1 &lt;= 0.1;  proposition: # timebound 10 # threshold: f1=1, f2=1.5, f3=1 # time-horizon: f1=4, f2=3, f3=3 goal: [f1]: &lt;&gt;[0, 4] (d1 &gt;= 1 R[0, 5] g1 &lt;= 7); [f2]: [][3,4] (g1 &lt; 6 -&gt; &lt;&gt;[0, 6] d1 &gt;= 3); [f3]: [][1, 3](&lt;&gt;[0, 5] d1 &gt; 1.5); </pre>
--	---

Fig. 13: The STLMC model for Bat with ODE dynamics



## A.2 Navigation

There is a vehicle that moves in the  $\mathbb{R}^2$  plane. The velocity of the object is determined by the position of the object in  $2 \times 2$  grid. (adapted from [20]). Figure 14 shows trajectory of the object. The height and the width of each cell is 50 . The vehicle is initially at the upper left corner of the grid, i.e, Zone 1, and it moves counter-clockwise. Figure 15 shows a hybrid automaton of the system. The velocities of the objects  $v_x$  and  $v_y$  are changes according to the ODEs in four modes  $Zone_1$ ,  $Zone_2$ ,  $Zone_3$  and  $Zone_4$ . Figure 15 shows a hybrid automaton  $H_1$  of one battery component. (the other component is similar). The object positions  $x$ ,  $y$ , velocities  $v_x$ ,  $v_y$ , change according to the ODEs in modes.

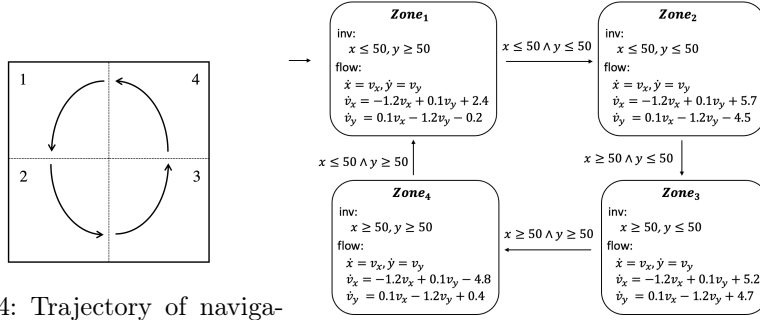


Fig. 14: Trajectory of navigation

Fig. 15: A hybrid automaton of navigation

For the experiments, we consider the following approximate dynamics to obtain linear flow conditions.

$$(\text{Zone}_1) \quad \dot{x} = v_x, \dot{y} = v_y, \dot{v}_x = 0.02v_x + 0.02v_y - 0.45, \dot{v}_y = 0.02v_x + 0.02v_y - 0.45$$

$$(\text{Zone}_2) \quad \dot{x} = v_x, \dot{y} = v_y, \dot{v}_x = 0.02v_x - 0.02v_y - 0.45, \dot{v}_y = -0.02v_x + 0.02v_y + 0.45$$

$$(\text{Zone}_3) \quad \dot{x} = v_x, \dot{y} = v_y, \dot{v}_x = 0.02v_x - 0.02v_y + 0.45, \dot{v}_y = -0.02v_x - 0.02v_y - 0.45$$

$$(\text{Zone}_4) \quad \dot{x} = v_x, \dot{y} = v_y, \dot{v}_x = 0.02v_x + 0.02v_y + 0.45, \dot{v}_y = 0.02v_x + 0.02v_y + 0.45$$

Table 9 shows the STL formulas for the navigation systems and Figure 16 is its model file.

Label	STL formula
f1:	$\Diamond_{[2,4]}(y \geq 30 \rightarrow \Box_{[1,5]} y \geq 60)$
f2:	$\Diamond_{[2,4]}(\Box_{[3,6]}(y \leq 40))$
f3:	$(\Box_{[0,4]}(\Diamond_{[0,4]} x_0 > 40))$

Table 9: STL properties for the navigation systems

```

bool dx; bool dy;
[-10, 10] vx; [-10, 10] vy;
[0, 100] x; [0, 100] y;
# zone 1
{ mode: dx = false;
  dy = false;
  inv: y >= 50;
  flow: d/dt[x] = vx;
        d/dt[y] = vy;
        d/dt[vx] = 0.02 * vx + 0.02 * vy - 0.45;
        d/dt[vy] = 0.02 * vx + 0.02 * vy - 0.45;
  jump: (and (x <= 50) (y <= 50)) =>
        (and (dx' = true) (dy' = false)
         (x' = x) (y' = y)
         (vx' = 3) (vy' = -3));
}
# zone 2
{ mode: dx = false;
  dy = true;
  inv: x >= 50;
  flow: d/dt[x] = vx;
        d/dt[y] = vy;
        d/dt[vx] = 0.02 * vx - 0.02 * vy - 0.45;
        d/dt[vy] = -0.02 * vx + 0.02 * vy + 0.45;
  jump: (and (x <= 50) (y >= 50)) =>
        (and (dx' = false) (dy' = false)
         (x' = x) (y' = y)
         (vx' = -3) (vy' = -3));
}
# zone 3
{ mode: dx = true;
  dy = false;
  inv: x < 50;
  flow: d/dt[x] = vx;
        d/dt[y] = vy;
        d/dt[vx] = 0.02 * vx - 0.02 * vy + 0.45;
        d/dt[vy] = -0.02 * vx - 0.02 * vy - 0.45;
  jump: (and (x >= 50) (y <= 50)) =>
        (and (dx' = true) (dy' = true)
         (x' = x) (y' = y)
         (vx' = 3) (vy' = 3));
}
# zone 4
{ mode: dx = true;
  dy = true;
  inv: y <= 50;
  flow: d/dt[x] = vx;
        d/dt[y] = vy;
        d/dt[vx] = 0.02 * vx + 0.02 * vy + 0.45;
        d/dt[vy] = 0.02 * vx + 0.02 * vy + 0.45;
  jump: (and (x >= 50) (y >= 50)) =>
        (and (dx' = false) (dy' = true)
         (x' = x) (y' = y)
         (vx' = -3) (vy' = 3));
}

init:
not(dx); not(dy); 29 <= x; x <= 31;
79 <= y; y <= 81; -3 <= vx; vx <= -2.5;
-3 <= vy; vy <= -2.5;

proposition:

# timebound 10
# threshold: f1=3, f2=2, f3=2
# time-horizon: f1=3, f2=2.5, f3=3
goal:
[f1]: <>[2, 4] ((y >= 30) -> [][1, 5] y >= 60);
[f2]: <> [2, 4] ([][3, 6] (y <= 40));
[f3]: [][1, 3](<>[0, 4](x > 40));

```

Fig. 16: An input model of the navigation systems

### A.3 Two Networked Water Tank Systems

Two water tanks are connected by pipes, where water flows from one tank to another (adapted from [36]). The water level  $x_i$  of tank  $i = 0, 1$  is controlled by its pump. There are two pump modes: **On** and **Off**. Each pump changes its mode according to the water level of its tank and the other water tank.

The continuous dynamics of  $x_i$  is defined as follows:

$$\dot{x}_i = \begin{cases} q_i + a\sqrt{2g}(\sqrt{x_{i-1}} - \sqrt{x_i}) & (\text{On}) \\ a\sqrt{2g}(\sqrt{x_{i-1}} - \sqrt{x_i}) & (\text{Off}) \end{cases}$$

where  $a$  and  $q_i$  depend on the width of the pipe and the pump's power, and  $g$  is the standard gravity constant. Figure 17 shows a hybrid automaton  $H_1$  of one water tank (the other component is similar). In this paper, we consider three battery models with different dynamics.

**Linear Dynamics.** In the case of linear dynamics, the amount of change in water level  $x_i$  is constant as follows.

$$\dot{x}_1 = \begin{cases} 0.9 & (\text{On}) \\ -0.8 & (\text{Off}) \end{cases} \quad \dot{x}_2 = \begin{cases} 1 & (\text{On}) \\ -0.6 & (\text{Off}) \end{cases}$$

Table 10 and Figure 18 shows three STL formulas and a model file in STL<sub>MC</sub> for the water tank systems with linear dynamics.

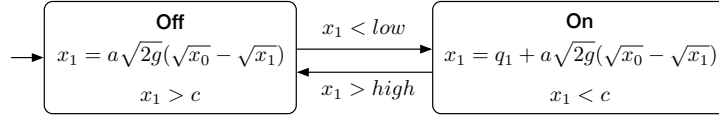


Fig. 17: A hybrid automaton of one water tank controller  $H_1$

Label	STL formula
f1:	$\square_{[1,3]}((x_0 \leq 7) \mathbf{R}_{[1,10]}(x_1 \leq 3))$
f2:	$(\diamond_{[1,10]}(x_1 < 5.5)) \mathbf{U}_{[2,5]}(x_0 \geq 5)$
f3:	$(\diamond_{[4,10]}(x_0 \geq 4 \rightarrow \square_{[2,5]}x_1 \leq 2))$

Table 10: STL properties for Wat systems with linear dynamics

```

int on0;    int on1;
[0, 10] x0; [0, 10] x1;
{ mode: on0 = 0;    on1 = 0;
  inv:  x0 >= 1; x1 >= 1;
  flow: d/dt[x0] = -0.8;
        d/dt[x1] = -0.6;
  jump: x0 <= 2 => (and (on0' = 1) (on1' = on1)
                    (x0' = x0) (x1' = x1));
        x1 <= 3 => (and (on0' = on0) (on1' = 1)
                    (x0' = x0) (x1' = x1));
        x0 <= 2 => (and (on0' = 1) (on1' = 1)
                    (x0' = x0) (x1' = x1));
        x1 <= 3 => (and (on0' = 1) (on1' = 1)
                    (x0' = x0) (x1' = x1));
}
{ mode: on0 = 0;    on1 = 1;
  inv:  x0 >= 1; x1 <= 9;
  flow: d/dt[x0] = -0.8;
        d/dt[x1] = 1;
  jump: x1 >= 6 => (and (on0' = on0) (on1' = 0)
                    (x0' = x0) (x1' = x1));
        x0 <= 2 => (and (on0' = 1) (on1' = on1)
                    (x0' = x0) (x1' = x1));
        x1 >= 6 => (and (on0' = 1) (on1' = 0)
                    (x0' = x0) (x1' = x1));
        x0 <= 2 => (and (on0' = 1) (on1' = 0)
                    (x0' = x0) (x1' = x1));
}
{ mode: on0 = 1;    on1 = 0;
  inv:  x0 <= 8; x1 >= 1;
  flow: d/dt[x0] = 0.9;
        d/dt[x1] = -0.6;
}

jump: x0 >= 5 => (and (on0' = 0) (on1' = on1)
                  (x0' = x0) (x1' = x1));
        x1 <= 3 => (and (on0' = on1) (on1' = 1)
                  (x0' = x0) (x1' = x1));
        x0 >= 5 => (and (on0' = 0) (on1' = 1)
                  (x0' = x0) (x1' = x1));
        x1 <= 3 => (and (on0' = 0) (on1' = 1)
                  (x0' = x0) (x1' = x1));
}
{ mode: on0 = 1;    on1 = 1;
  inv:  x0 <= 9; x1 <= 9;
  flow: d/dt[x0] = 0.9;
        d/dt[x1] = 1;
  jump: x0 >= 5 => (and (on0' = 0) (on1' = on1)
                    (x0' = x0) (x1' = x1));
        x1 >= 6 => (and (on0' = on0) (on1' = 0)
                    (x0' = x0) (x1' = x1));
        x0 >= 5 => (and (on0' = 0) (on1' = 0)
                    (x0' = x0) (x1' = x1));
        x1 >= 6 => (and (on0' = 0) (on1' = 0)
                    (x0' = x0) (x1' = x1));
}
init: on0 = 0; 4.4 <= x0; x0 <= 4.5;
      on1 = 0; 5.9 <= x1; x1 <= 6;

proposition:

# timebound : 20
goal:
[f1]: [][1, 3]((x0 <= 7) R[1, 10] (x1 <= 3));
[f2]: (<>[1, 10] x1 < 5.5) U[2, 5] (x0 >= 5);
[f3]: <>[4, 10] (x0 >= 4 -> [][2, 5] x1 <= 2);

```

Fig. 18: The STLMC model for Wat with linear dynamics

**Polynomial Dynamics.** In the case of polynomial dynamics, the water level  $x_i$  change as follows.

$$x_1(t) = \begin{cases} 0.02 * t^2 + 0.4 * t + x_1(0) & (\text{On}) \\ 0.005 * t^2 - 0.4 * t + x_1(0) & (\text{Off}) \end{cases}$$

$$x_2(t) = \begin{cases} 0.02 * t^2 + 0.5 * t + x_2(0) & (\text{On}) \\ 0.005 * t^2 - 0.6 * t + x_2(0) & (\text{Off}) \end{cases}$$

where  $t$  is time and  $x_i(0)$  indicates the initial value after a transition.

Table 11 shows three STL formulas for the water tank systems with polynomial and ODE dynamics, Figure 20 is a model file of the networked watertank system with polynomial dynamics.

**ODE Dynamics.** For ODE dynamics, we consider the watertank model with only one watertank for simplification. We consider the watertank model with

Label	STL formula
f1:	$\Box_{[1,4]}(x_1 \leq 6 \mathbf{U}_{[2,5]} x_2 \geq 7)$
f2:	$\Diamond_{[0,3]}(x_1 \geq 7 \rightarrow \Box_{[3,4]} x_2 \geq 4)$
f3:	$(\Diamond_{[0,4]} x_2 \geq 3) \mathbf{R}_{[3,5]} x_1 \geq 2)$

Table 11: STL properties for Wat systems with polynomial dynamics

only one tank for nonlinear dynamic model with following dynamics.

$$\dot{x} = \begin{cases} 0.8 - 0.01 * \sqrt{2} * g * \sqrt{x} & \text{(On)} \\ -0.01 * \sqrt{2} * g * \sqrt{x} & \text{(Off)} \end{cases}$$

where  $g$  is standard gravity constant.

Table 12 shows three STL formulas for the water tank systems with polynomial and ODE dynamics, Figure 19 is a model file of the networked watertank system with ODE dynamics.

Label	STL formula
f1:	$x \geq 2 \mathbf{R}_{[0,3]}(\Diamond_{[2,4]} x > 5)$
f2:	$\Diamond_{[0,4]}(x > 3 \rightarrow \Box_{[2,3]} x \leq 2.5)$
f3:	$(\Diamond_{[0,4]} \mathbf{R}_{[0,\infty]} x \geq 2)$

Table 12: STL properties for Wat systems with ODE dynamics

<pre> bool a; [0, 10] x; {   mode: a = false;   inv: x &gt; 1;   flow: d/dt[x] = (-0.01 * 4.43 * sqrt(x));   jump: (x &lt; 5) =&gt; (and a' (x' = x)); } {   mode: a = true;   inv: x &lt; 8; </pre>	<pre> flow: d/dt[x] = 0.8 - 0.01 * 4.43 * sqrt(x); jump: (x &gt;= 2) =&gt; (and (not a') (x' = x)); } init: (and (a = false) (4 &lt;= x) (x &lt;= 4.5));  proposition:  #timebound 8 goal: [f1]: (x &gt;= 2) R[0, 3] (&lt;&gt; [2, 4] x &gt; 5); [f2]: &lt;&gt;[0, 4](x &gt; 3 -&gt; [] [2, 3] x &lt;= 2.5); [f3]: (&lt;&gt;[0, 4] x &gt;= 5) R[0, inf] x &gt;= 2; </pre>
--	---

Fig. 19: The STLMC model for Wat with ODE dynamics

```

bool a; bool b;
[0, 10] x1; [0, 10] x2;
{
  mode: a = false; b = false;
  inv: (x1 > 1); (x2 > 1);
  flow:
    x1(t) = 0.005 * t * t - 0.4 * t + x1(0);
    x2(t) = 0.005 * t * t - 0.6 * t + x2(0);
  jump:
    x2 < 6 =>
      (and (not a') b' (x1' = x1) (x2' = x2));
    (x1 < 5) =>
      (and a' (not b') (x1' = x1) (x2' = x2));
}
{
  mode: a = false; b = true;
  inv: (x1 > 1); (x2 < 9);
  flow:
    x1(t) = 0.005 * t * t - 0.4 * t + x1(0);
    x2(t) = 0.5 * t + 0.02 * t * t + x2(0);
  jump:
    x2 > 2 =>
      (and (not a') (not b') (x1' = x1)
        (x2' = x2));
    x1 < 5 =>
      (and a' b' (x1' = x1) (x2' = x2));
}
{
  mode: a = true; b = false;
  inv: (x1 < 8); (x2 > 1);
  flow:
    x1(t) = 0.4 * t + 0.02 * t * t + x1(0);
    x2(t) = 0.005 * t * t - 0.6 * t + x2(0);
  jump:
    x1 > 2 =>
      (and (not a') (not b') (x1' = x1)
        (x2' = x2));
    x2 < 6 =>
      (and a' b' (x1' = x1) (x2' = x2));
}
}

init:
(and (not(a)) not(b) (4 <= x1) (x1 <= 5)
  (5 <= x2) (x2 <= 6));

proposition:
# timebound : 10
# threshold: f1=1.5, f2=2, f3=4
goal:
[f1]: [] [1,4] (x1 <= 6 U [2, 5] x2 >= 7);
[f2]: <> [0, 3] (x1 >= 7 -> [] [3, 4] (x2 >= 4));
[f3]: (<> [0,4] x2 >= 3) R[3, 5] x1 >=2;

```

Fig. 20: The STLMC model for Wat with polynomial dynamics

#### A.4 Autonomous Driving of Two Cars

**Linear Dynamics.** There are two cars that moves in the  $\mathbb{R}^2$  plane. One of the car (*leader*) drives according to its own scenario and the other (*follower*) follows the leader. Autonomous car controller controls the behavior of the followers, adapted from [1]. Figure 21 shows a hybrid automaton of one of the following car, where  $(x_i, y_i)$  is the position,  $v_i$  is velocity,  $\theta_i$  is direction, and  $\phi_i$  is steering angle. If the follower is to the left of the leader, the follower rotate counter-clockwise. If the follower is to the right of the leader, the follower rotate clockwise. Otherwise, the follower drives in a straight way.

We instantiate the autonomous car model with two cars. For linear dynamics, the dynamic of positions is constant as follows.

$$\begin{cases} \dot{x}_1 = 3, \dot{y}_1 = 0 & (1) \text{ straight} \\ \dot{x}_1 = 1.5, \dot{y}_1 = 3 & (2) \text{ counter-clockwise} \\ \dot{x}_1 = 1.5, \dot{y}_1 = -3 & (3) \text{ clockwise} \end{cases}$$

Table 13 shows three STL formulas for the autonomous cars. Figure 22 is a model file of the autonomous car in STLMC.

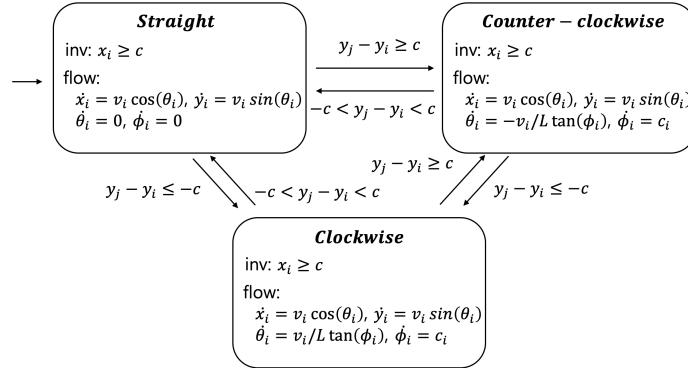


Fig. 21: Hybrid automata of the autonomous car

Table 13: STL properties for the autonomous car with linear dynamics

Label	STL formula
f1:	$(\diamond_{[3,5]} y_2 - y_1 \leq 5) \mathbf{U}_{[2,10]} y_2 - y_1 \geq 8$
f2:	$\diamond_{[3,10]} (\square_{[5,15]} x_2 - x_1 \geq -2)$
f3:	$(\square_{[2,5]} y_2 - y_1 \leq 4) \mathbf{R}_{[0,10]} x_2 - x_1 \geq 4$

**Polynomial Dynamics.** We consider two autonomous cars, a *leader* and a *follower*. The leader drives according to its own scenario, and the follower follows the leader (adapted from [1]). The velocities  $v_x$  and  $v_y$  change of the follower change according to the controller's modes: CxCy, CxFy, FxCy, and FxFy. Cx (or Cy) means the relative distance of the  $x$ (or  $y$ )-coordinate is closed. Fx (or Fy) means the relative distance of the  $x$ (or  $y$ )-coordinate is far. When the relative distance of the car is too far (or close), the car accelerates (or decelerates).

The relative positions,  $r_x$  and  $r_y$ , and the velocities of the flower,  $v_x$  and  $v_y$ , change as follows:

$$\begin{array}{llll}
 \dot{r}_x = v_x, & \dot{r}_y = v_y, & \dot{v}_x = 1.2, & \dot{v}_y = 1.4 & (\text{CxCy}) \\
 \dot{r}_x = v_x, & \dot{r}_y = v_y, & \dot{v}_x = 1.2, & \dot{v}_y = -1.4 & (\text{CxFy}) \\
 \dot{r}_x = v_x, & \dot{r}_y = v_y, & \dot{v}_x = -1.2, & \dot{v}_y = 1.4 & (\text{FxCy}) \\
 \dot{r}_x = v_x, & \dot{r}_y = v_y, & \dot{v}_x = -1.2, & \dot{v}_y = -1.4 & (\text{FxFy})
 \end{array}$$

Figure 23 shows a hybrid automaton of the autonomous car. Initially, the relative distance is (CxCy). The following car tries to maintain the distance within certain range. Table 14 shows three STL formulas for the autonomous cars. Figure 24 is a model file of the autonomous car in STLMC.

```

bool r;
bool d;
[-100, 100] x1;
[-100, 100] y1;
[-100, 100] x2;
[-100, 100] y2;

# car1 : straight
{
  mode:
    r = false;
    d = false;
  inv:
    x2 - x1 <= 5;
    y2 - y1 <= 5;
  flow:
    d/dt[x1] = 3;
    d/dt[y1] = 0;
    d/dt[x2] = 2;
    d/dt[y2] = 2;
  jump:
    (y2 - y1) >= 3 =>
      (and (r' = true) (d' = false)
        (x1' = x1) (y1' = y1)
        (x2' = x2) (y2' = y2));
    (y2 - y1) <= -3 =>
      (and (r' = true) (d' = true)
        (x1' = x1) (y1' = y1)
        (x2' = x2) (y2' = y2));
}
# car1 : up
{
  mode:
    r = true;
    d = false;
  inv:
    x2 - x1 >= -5;
    y2 - y1 >= -5;
  flow:
    d/dt[x1] = 1.5;
    d/dt[y1] = 3;
    d/dt[x2] = 2;
    d/dt[y2] = 2;
  jump:
    (and ((y2 - y1) <= 1)
      ((y2 - y1) >= -1)) =>
      (and (r' = false) (d' = false)
        (x1' = x1) (y1' = y1)
        (x2' = x2) (y2' = y2));
}

(x1' = x1) (y1' = y1)
(x2' = x2) (y2' = y2));
(y2 - y1) <= -3 =>
  (and (r' = true) (d' = true)
    (x1' = x1) (y1' = y1)
    (x2' = x2) (y2' = y2));
}
# car1 : down
{
  mode:
    r = true;
    d = true;
  inv:
    x2 - x1 >= -5;
    x2 - x1 <= 5;
  flow:
    d/dt[x1] = 1.5;
    d/dt[y1] = -3;
    d/dt[x2] = 2;
    d/dt[y2] = 2;
  jump:
    (and ((y2 - y1) <= 1)
      ((y2 - y1) >= -1)) =>
      (and (r' = false) (d' = false)
        (x1' = x1) (y1' = y1)
        (x2' = x2) (y2' = y2));
    (y2 - y1) >= 3 =>
      (and (r' = true) (d' = false)
        (x1' = x1) (y1' = y1)
        (x2' = x2) (y2' = y2));
}

init:
  (and not(r) not(d) (0 <= x1) (x1 <= 1)
    (0 <= y1) (y1 <= 1) (2 <= x2)
    (x2 <= 3) (2 <= y2) (y2 <= 3));

proposition:
  [yd]: ((y2 - y1) >= 8);
  [xd]: ((x2 - x1) >= 4);

# timebound 40
goal:
  [f1]: (< [3, 5] (y2 - y1) <= 5) U[2, 10] yd;
  [f2]: [] [3, 10] (< [5, 15] ((x2 - x1) >= -2));
  [f3]: ([] [2, 5] (y2 - y1 <= 4)) R [0, 10] xd;

```

Fig. 22: An input model of the autonomous car with linear dynamics



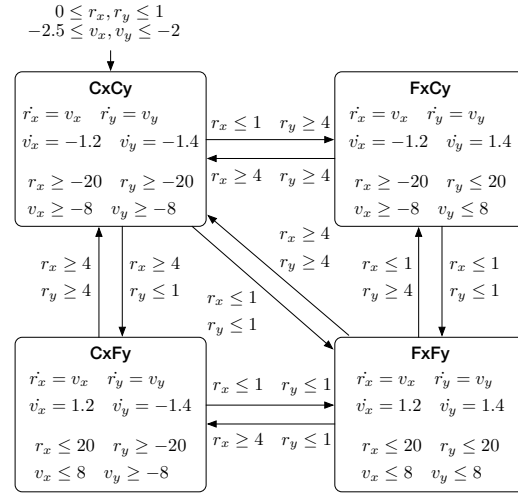


Fig. 23: A hybrid automaton of the autonomous car (F: Far, C: Close)

Table 14: STL properties for the autonomous car with polynomial dynamics

Label	STL formula
f1:	$\square_{[0,4]}(v_x < -2 \rightarrow \diamond_{[2,5]} r_x \leq -2)$
f2:	$(\diamond_{[0,4]} r_y > 3) \mathbf{U}_{[0,5]}(v_y > 1.5)$
f3:	$\diamond_{[0,3]}(r_x \leq 5 \mathbf{U}_{[0,5]} ix = false)$

**ODE Dynamics** For ODE case, we assume that we can obtain relative positions between follower and leader and the follower changes its velocity based on the relative distance. If the x-axis relative distance is less than some constant value, the dynamic of  $v_x$  is positive. If the x-axis relative distance is greater than some constant value, the dynamic of  $v_x$  is negative. The velocity of y changes in a similar way.

(iii) ODE dynamics

$$\begin{cases}
 \dot{r}_x = 1, \dot{r}_y = 2 * \sin(\theta), \dot{\theta} = 0.05 & (1) \text{ (x-close, y-close)} \\
 \dot{r}_x = 1, \dot{r}_y = -2 * \sin(\theta), \dot{\theta} = 0.05 & (2) \text{ (x-close, y-far)} \\
 \dot{r}_x = -1, \dot{r}_y = 2 * \sin(\theta), \dot{\theta} = 0.05 & (3) \text{ (x-far, y-close)} \\
 \dot{r}_x = -1, \dot{r}_y = -2 * \sin(\theta), \dot{\theta} = 0.05 & (4) \text{ (x-far, y-far)}
 \end{cases}$$

where  $r_x, r_y$  represent relative distance and  $\theta$  is direction.

Table 15 shows three STL formulas for the autonomous cars. Figure 25 is a model file of the autonomous car in STLMC.

```

bool ix;      bool iy;
[-40, 40] rx; [-40, 40] ry;
[-30, 30] vx; [-30, 30] vy;
# acc, acc
{ mode: ix = true; iy = true;
  inv: rx < 20; ry < 20;
      vx < 8; vy < 8;
  flow: d/dt[rx] = vx;
        d/dt[ry] = vy;
        d/dt[vx] = 1.2;
        d/dt[vy] = 1.4;
  jump: (and (rx >= 3) (ry < 3)) =>
        (and (ix' = false) (iy' = true)
             (rx' = rx) (ry' = ry)
             (vx' = 0) (vy' = 0));
        (and (rx < 3) (ry >= 3)) =>
        (and (ix' = true) (iy' = false)
             (rx' = rx) (ry' = ry)
             (vx' = 0) (vy' = 0));
}
# car1 : acc, dec
{ mode: ix = true; iy = false;
  inv: rx < 20; ry > -20;
      vx < 8; vy > -8;
  flow: d/dt[rx] = vx;
        d/dt[ry] = vy;
        d/dt[vx] = 1.2;
        d/dt[vy] = -1.4;
  jump: (and (rx >= 3) (ry >= 3)) =>
        (and (ix' = false) (iy' = false)
             (rx' = rx) (ry' = ry)
             (vx' = 0) (vy' = 0));
        (and (rx >= 3) (ry < 3)) =>
        (and (ix' = false) (iy' = true)
             (rx' = rx) (ry' = ry)
             (vx' = 0) (vy' = 0));
}
# car1 : dec, acc
{ mode: ix = false; iy = true;
  inv: rx > -20; ry < 20;
      vx > -8; vy < 8;
  flow: d/dt[rx] = vx;
        d/dt[ry] = vy;
        d/dt[vx] = -1.2;
        d/dt[vy] = 1.4;
  jump: (and (rx < 3) (ry < 3)) =>
}

      (and (ix' = true) (iy' = true)
           (rx' = rx) (ry' = ry)
           (vx' = 0) (vy' = 0));
      (and (rx >= 3) (ry >= 3)) =>
      (and (ix' = false) (iy' = false)
           (rx' = rx) (ry' = ry)
           (vx' = 0) (vy' = 0));
      (and (rx < 3) (ry >= 3)) =>
      (and (ix' = true) (iy' = false)
           (rx' = rx) (ry' = ry)
           (vx' = 0) (vy' = 0));
      (and (rx < 3) (ry < 3)) =>
      (and (ix' = true) (iy' = true)
           (rx' = rx) (ry' = ry)
           (vx' = 0) (vy' = 0));
      (and (ix' = false) (iy' = true)
           (rx' = rx) (ry' = ry)
           (vx' = 0) (vy' = 0));
      (and (rx >= 3) (ry < 3)) =>
      (and (ix' = false) (iy' = false)
           (rx' = rx) (ry' = ry)
           (vx' = 0) (vy' = 0));
      (and (rx < 3) (ry >= 3)) =>
      (and (ix' = true) (iy' = false)
           (rx' = rx) (ry' = ry)
           (vx' = 0) (vy' = 0));
}
init: not(ix); 0 <= rx; rx <= 1;
      not(iy); 0 <= ry; ry <= 1;
      -2.5 <= vx; vx <= -2;
      -2.5 <= vy; vy <= -2;

proposition:
[pix]: ix;

# bound: 10, timebound: 10, solver: yices
goal:
[f1]: [] [0,4] (vx < -2 -> <> [2,5] rx <= -2);
[f2]: (<> [0, 4] ry > 3) U [0, 5] (vy > 1.5);
[f3]: <> [0,3] (rx <= 5 U [0, 5] pix = false);

```

Fig. 24: An input model of the autonomous car with polynomial dynamics

```

bool ix;
bool iy;
[-40, 40] rx;
[-40, 40] ry;
[0, 1.5] theta1;

# car1 : inc, inc
{
  mode:
  ix = true;
  iy = true;
  inv:
  rx < 20;
  ry < 20;
  theta1 < 1.4;
  flow:
  d/dt[rx] = 1;
  d/dt[ry] = 2 * sin(theta1);
  d/dt[theta1] = 0.05;
  jump:
  (and (rx < 2) (ry < 2)) =>
    (and (ix' = true) (iy' = true)
      (rx' = rx) (ry' = ry)
      (theta1' = 0));
  (and (rx >= 2.4) (ry >= 2.4)) =>
    (and (ix' = false) (iy' = false)
      (rx' = rx) (ry' = ry)
      (theta1' = 0));
  (and (rx >= 2.4) (ry < 2)) =>
    (and (ix' = false) (iy' = true)
      (rx' = rx) (ry' = ry)
      (theta1' = 0));
  (and (rx < 2) (ry >= 2.4)) =>
    (and (ix' = true) (iy' = false)
      (rx' = rx) (ry' = ry)
      (theta1' = 0));
}
# car1 : inc, dec
{
  mode:
  ix = true;
  iy = false;
  inv:
  rx < 20;
  ry > -20;
  theta1 < 1.4;
  flow:
  d/dt[rx] = 1;
  d/dt[ry] = -2 * sin(theta1);
  d/dt[theta1] = 0.05;
  jump:
  (and (rx < 2) (ry < 2)) =>
    (and (ix' = true) (iy' = true)
      (rx' = rx) (ry' = ry)
      (theta1' = 0));
  (and (rx >= 2.4) (ry >= 2.4)) =>
    (and (ix' = false) (iy' = false)
      (rx' = rx) (ry' = ry)
      (theta1' = 0));
  (and (rx >= 2.4) (ry < 2)) =>
    (and (ix' = false) (iy' = true)
      (rx' = rx) (ry' = ry)
      (theta1' = 0));
  (and (rx < 2) (ry >= 2.4)) =>
    (and (ix' = true) (iy' = false)
      (rx' = rx) (ry' = ry)
      (theta1' = 0));
}
# car1 : dec, inc
{
  mode:
  ix = false;
  iy = true;
  inv:
  rx > -20;
  ry < 20;
  theta1 < 1.4;
  flow:
  d/dt[rx] = -1;
  d/dt[ry] = 2 * sin(theta1);
  d/dt[theta1] = 0.05;
  jump:
  (and (rx < 2) (ry < 2)) =>
    (and (ix' = true) (iy' = true)
      (rx' = rx) (ry' = ry)
      (theta1' = 0));
  (and (rx >= 2.4) (ry >= 2.4)) =>
    (and (ix' = false) (iy' = false)
      (rx' = rx) (ry' = ry)
      (theta1' = 0));
  (and (rx >= 2.4) (ry < 2)) =>
    (and (ix' = false) (iy' = true)
      (rx' = rx) (ry' = ry)
      (theta1' = 0));
  (and (rx < 2) (ry >= 2.4)) =>
    (and (ix' = true) (iy' = false)
      (rx' = rx) (ry' = ry)
      (theta1' = 0));
}

init:
(and not(ix) not(iy) (0 <= rx) (rx <= 1)
  (0 <= ry) (ry <= 1)
  (0 <= theta1) (theta1 <= 1));

proposition:

# timebound 5
goal:
[f1]: <>[1, 3]([0, 2] (theta > 1));
[f2]: (<>[1, 2] ry >= 1) R[0, 2] theta > 2;
[f3]: [][0, 2] (rx > 2 -> <> [2, 3] (ry < -1));

```

Fig. 25: An input model of the autonomous car with polynomial dynamics

Table 15: STL properties for the autonomous car with ODE dynamics

Label	STL formula
f1:	$\Diamond_{[1,3]}(\Box_{[0,2]}\theta > 1)$
f2:	$(\Diamond_{[2,5]}r_y \geq 1)\mathbf{R}_{[0,2]}\theta > 2$
f3:	$\Box_{[0,2]}(r_x > 2 \rightarrow \Diamond_{[2,3]}r_y < -1)$

### A.5 A Railroad Gate Controller

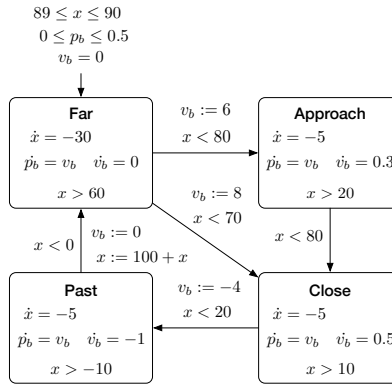


Fig. 26: A hybrid automaton of the railroad gate controller

There are a gate and a gate controller with a crossing bar on a circular railroad track and a train moves on the track. The gate controller opens (or closes) the crossing bar when the train approaches to (or leaves from) the gate (adapted from [27]). Figure 26 shows a hybrid automaton for this system. The position of the crossing bar  $p_b$  from the ground changes according to the velocity of bar  $v_b$ , depending on the modes (Far, Approach, Close, and Past) of the gate controller and the distance  $x$ , between the gate and the train.

We consider three variants of dynamics. In linear and polynomial dynamics, we assume the behavior of the crossing bar depends on the approaching train to increase complexity. The velocity of the crossing bar depends on the train position  $t$ . There are 4 cases, (i) when the train is *far* away enough from the crossing bar, (ii) when the train *approaches* to the crossing bar, (iii) when the train *closes* to the crossing bar, (iv) when the train *passes* the crossing bar. For linear dynamics, the dynamics of all variables are constant as follows. For polynomial dynamics, the velocity of the bar is additionally considered.

$$\begin{array}{ll}
 \text{(i) Linear dynamics} & \text{(ii) Polynomial dynamics} \\
 \left\{ \begin{array}{l} \dot{t} = -30, \dot{b} = 0 \quad (1) \\ \dot{t} = -5, \dot{b} = 5 \quad (2) \\ \dot{t} = -5, \dot{b} = 10 \quad (3) \\ \dot{t} = -5, \dot{b} = -5 \quad (4) \end{array} \right. & \left\{ \begin{array}{l} \dot{t} = -30, \dot{b} = v_b, \dot{v}_b = 0 \quad (1) \\ \dot{t} = -5, \dot{b} = v_b, \dot{v}_b = 0.3 \quad (2) \\ \dot{t} = -5, \dot{b} = v_b, \dot{v}_b = 0.5 \quad (3) \\ \dot{t} = -5, \dot{b} = v_b, \dot{v}_b = -1 \quad (4) \end{array} \right.
 \end{array}$$

For nonlinear dynamics, the crossing bar opens and closes repeatedly according to the following dynamics,

$$\left\{ \begin{array}{ll} \dot{b} = v_b, \dot{v}_b = 1.2 & (1) \text{ Open} \\ \dot{b} = v_b, \dot{v}_b = -1 + 0.2 * v_b^2 & (1) \text{ Close} \end{array} \right.$$

Table 16 shows three STL formulas for the railroad gate controller with linear dynamics. Figure 27 is a model file of the railroad gate controller with linear dynamics in STLMC.

Label	STL formula
f1:	$\Diamond_{[0,5]}((p_b \geq 40) \mathbf{U}_{[1,8]}(x < 40))$
f2:	$\Diamond_{[0,4]}(x < 50 \rightarrow \Box_{[2,10]}(p_b > 40))$
f3:	$\Box_{[0,5]}((x < 50) \mathbf{U}_{[2,10]}(p_b > 5))$

Table 16: STL properties for the railroad gate controller

### A.6 Networked thermostat controller

Two rooms are interconnected by an open doors (28). Networked thermostat controller controls the heaters in each room, adapted form [3]. Figure 29 shows a hybrid automaton of the thermostat model that controls one of the rooms. The temperature  $x_i$  of room <sub>$i$</sub>  is controlled by the heater and the temperatures of both rooms, where  $A_i$  is the set of the adjacent rooms, and  $K_i, h_i, d_i$  depend on the size of the room, the heater’s power, and the size of the door. If the room temperature is low, the heater turns on, and if the room temperature is high, the heater turns off. We consider following various dynamics for the thermostat model.

In linear and polynomial dynamics models, we consider a model with two rooms. For ode dynamics, we consider the thermostat model with only one room for simplification.

(i) Linear dynamic

$$\dot{x}_1 = \begin{cases} -0.4 & (\text{On}) \\ 0.7 & (\text{Off}) \end{cases} \quad \dot{x}_2 = \begin{cases} -0.6 & (\text{On}) \\ 1 & (\text{Off}) \end{cases}$$

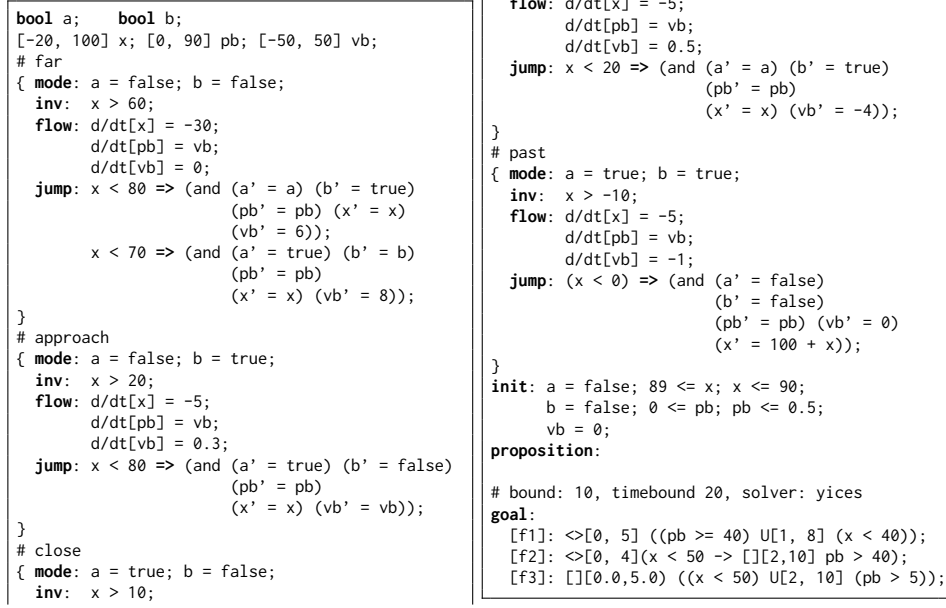


Fig. 27: An input model of the railroad gate controller

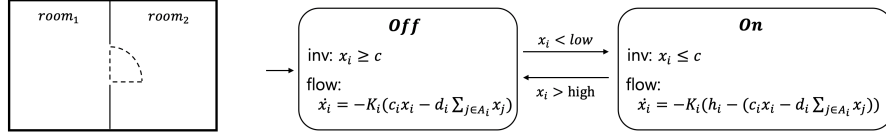


Fig. 28: Connected rooms

Fig. 29: Hybrid automata of the thermostat

(ii) Polynomial dynamic

$$\dot{x}_1 = \begin{cases} 0.05 * t^2 + 0.5 * t + x_1(0) & (\text{On}) \\ -0.0175 * t^2 - 0.35 * t + x_1(0) & (\text{Off}) \end{cases}$$

$$\dot{x}_2 = \begin{cases} 0.06 * t^2 + 0.6 * t + x_2(0) & (\text{On}) \\ -0.275 * t^2 - 0.55 * t + x_2(0) & (\text{Off}) \end{cases}$$

where  $t$  is time and  $x_i(0)$  indicates the initial value after a transition.

Table 17 shows three STL formulas for the filtered oscillator. Figure 30 is a model file of the filtered oscillator in STLMC.

We consider the thermostat model with ODE dynamics in Example 1.

```

# model description:
#
# Two rooms are connected by an open door.
# The room' temperature is controlled by
# its own heater and changes over time.
# It is also affected by each other's room
# temperature.

bool a;
[0, 30] x1;
bool b;
[0, 30] x2;
{
  mode:
  a = false;
  b = false ;
  inv:
  x1 >= 10;
  (x2 >= 10);
  flow:
  d/dt[x1] = -0.4;
  d/dt[x2] = -0.6;
  jump:
  x2 <= 19 =>
    (and (a' = false) (b' = true)
      (x1' = x1) (x2' = x2));
  x2 <= 19 =>
    (and (a' = true) (b' = true)
      (x1' = x1) (x2' = x2));
  x1 <= 17 =>
    (and (a' = true) (b' = false)
      (x1' = x1) (x2' = x2));
  x1 <= 17 =>
    (and (a' = true) (b' = true)
      (x1' = x1) (x2' = x2));
}
{
  mode:
  a = false;
  b = true;
  inv:
  x1 >= 10;
  (x2 <= 30);
  flow:
  d/dt[x1] = -0.4;
  d/dt[x2] = 1;
  jump:
  x1 <= 17 =>
    (and (a' = true) (b' = true)
      (x1' = x1) (x2' = x2));
  x1 <= 17 =>
    (and (a' = true) (b' = false)
      (x1' = x1) (x2' = x2));
  x2 >= 23 =>
    (and (a' = false) (b' = false)
      (x1' = x1) (x2' = x2));
  x2 >= 23 =>
    (and (a' = true) (b' = false)
      (x1' = x1) (x2' = x2));
}
}

mode:
a = true;
b = false;
inv:
x1 <= 30;
(x2 >= 10);
flow:
d/dt[x1] = 0.7;
d/dt[x2] = -0.6;
jump:
x2 <= 19 =>
  (and (a' = true) (b' = true)
    (x1' = x1) (x2' = x2));
x2 <= 19 =>
  (and (a' = false) (b' = true)
    (x1' = x1) (x2' = x2));
x1 >= 22 =>
  (and (a' = false) (b' = false)
    (x1' = x1) (x2' = x2));
x1 >= 22 =>
  (and (a' = false) (b' = true)
    (x1' = x1) (x2' = x2));
}
{
  mode:
  a = true;
  b = true;
  inv:
  x1 <= 30;
  (x2 <= 30);
  flow:
  d/dt[x1] = 0.7;
  d/dt[x2] = 1;
  jump:
  x1 >= 22 =>
    (and (a' = false) (b' = true)
      (x1' = x1) (x2' = x2));
  x1 >= 22 =>
    (and (a' = false) (b' = false)
      (x1' = x1) (x2' = x2));
  x2 >= 23 =>
    (and (a' = true) (b' = false)
      (x1' = x1) (x2' = x2));
  x2 >= 23 =>
    (and (a' = false) (b' = false)
      (x1' = x1) (x2' = x2));
}
}

init:
(and not(a) not(b) (18 <= x1) (x1 <= 20)
  (22.5 <= x2) (x2 <= 23));

proposition:

#time bound 30
goal:
[f1]: (<> [2, 5] (x1 < 23 R[2, 10] x2 > 24));
[f2]: [] [1, 4] (x2 < 19 -> <> [5, 15] x2 > 23);
[f3]: (x2 > 18) U [10, 20] ( [] [3, 5] x1 < 19 ) ;

```

Fig. 30: An input model of the thermostat controller with linear dynamics

Table 17: STL properties for thermostat controller with linear dynamics

Label	STL formula
f1:	$\Diamond_{[2,5]}(x_1 < 23 \mathbf{R}_{[2,10]} x_2 > 24)$
f2:	$\Box_{[1,4]}(x_2 < 19 \rightarrow \Diamond_{[5,16]} x_2 > 23)$
f3:	$x_2 > 18 \mathbf{U}_{[10,20]}(\Box_{[3,5]} x_1 < 19)$

### A.7 A Filtered Oscillator

We consider a filtered oscillator model adapted from [22]. The filtered oscillator consists of a 2-dimensional switched oscillator of signals  $x$  and  $y$  with a  $k$ -series of first-order filters. The filters smoothens an input signal  $x$ , producing an output signal  $z$ . When the signal  $x$  goes through each filter, the amplitude of the signal diminishes as it passing by. Each filter  $i \in \{0, 1, 2, 3\}$  takes an input signal  $x_i$  and outputs a diminished signal  $x_{i+1}$ , where  $x_0 = x$  and  $x_4 = z$ .

Figure 31 shows a hybrid automaton of the filtered oscillator. Initially, the  $x$  and  $y$  are in  $[0.2, 0.3]$  and  $[-0.1, 0.1]$ , respectively and all filters output zero signals. The jumps between modes define the behavior of the filtered oscillator to maintain a stable oscillation.

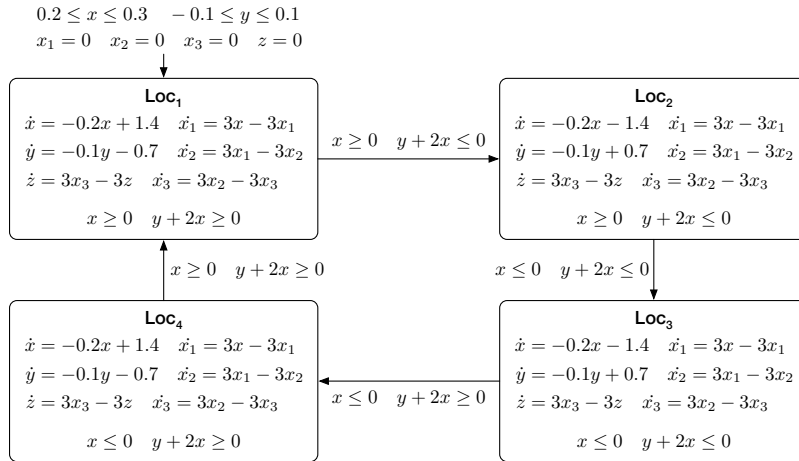


Fig. 31: A hybrid automaton of the filtered oscillator

Table 18 shows three STL formulas for the filtered oscillator. Figure 32 is a model file of the filtered oscillator in STLMC.



```

real loc;
[-100, 100] x; [-100, 100] y; [-100, 100] z;
[-100, 100] x1; [-100, 100] x2; [-100, 100] x3;
{ mode: loc = 1;
  inv: x >= 0;
      y + 2 * x >= 0;
  flow: d/dt[x] = -0.2 * x + 1.4;
         d/dt[y] = -0.1 * y - 0.7;
         d/dt[x1] = 3 * x - 3 * x1;
         d/dt[x2] = 3 * x1 - 3 * x2;
         d/dt[x3] = 3 * x2 - 3 * x3;
         d/dt[z] = 3 * x3 - 3 * z;
  jump: (and (x >= 0) (y + 2 * x <= 0)) =>
         (and (loc' = 2) (x' = x) (y' = y)
              (x1' = x1) (x2' = x2) (x3' = x3)
              (z' = z));
}
{ mode: loc = 2;
  inv: x >= 0;
      y + 2 * x <= 0;
  flow: d/dt[x] = -0.2 * x - 1.4;
         d/dt[y] = -0.1 * y + 0.7;
         d/dt[x1] = 3 * x - 3 * x1;
         d/dt[x2] = 3 * x1 - 3 * x2;
         d/dt[x3] = 3 * x2 - 3 * x3;
         d/dt[z] = 3 * x3 - 3 * z;
  jump: (and (x <= 0) (y + 2 * x <= 0)) =>
         (and (loc' = 3) (x' = x) (y' = y)
              (x1' = x1) (x2' = x2) (x3' = x3)
              (z' = z));
}
{ mode: loc = 3;
  inv: x <= 0;
      y + 2 * x <= 0;
  flow: d/dt[x] = -0.2 * x - 1.4;
         d/dt[y] = -0.1 * y + 0.7;
         d/dt[x1] = 3 * x - 3 * x1;
         d/dt[x2] = 3 * x1 - 3 * x2;
         d/dt[x3] = 3 * x2 - 3 * x3;
         d/dt[z] = 3 * x3 - 3 * z;
  jump: (and (x <= 0) (y + 2 * x >= 0)) =>
         (and (loc' = 4) (x' = x) (y' = y)
              (x1' = x1) (x2' = x2) (x3' = x3)
              (z' = z));
}
{ mode: loc = 4;
  inv: x <= 0;
      y + 2 * x >= 0;
  flow: d/dt[x] = -0.2 * x + 1.4;
         d/dt[y] = -0.1 * y - 0.7;
         d/dt[x1] = 3 * x - 3 * x1;
         d/dt[x2] = 3 * x1 - 3 * x2;
         d/dt[x3] = 3 * x2 - 3 * x3;
         d/dt[z] = 3 * x3 - 3 * z;
  jump: (and (x >= 0) (y + 2 * x >= 0)) =>
         (and (loc' = 1) (x' = x) (y' = y)
              (x1' = x1) (x2' = x2) (x3' = x3)
              (z' = z));
}
init: loc = 1; 0.2 <= x; x <= 0.3;
        -0.1 <= y; y <= 0.1;
        x1 = 0; x2 = 0; x3 = 0; z = 0;

proposition:

# bound: 5, timebound: 8, solver: dreal
goal:
[f1]: <[0,3]((x3 >= 1) R[0, inf) (y <= 10));
[f2]: <[2, 5] ([][0, 3] (x2 < 4));
[f3]: ([][1, 3] (x <= 2)) R[2, 5] (x3 > 2);

```

Fig. 32: An input model of a filtered oscillator

Table 18: STL properties for the oscillator

Label	STL formula
f1:	$\diamond_{[0,3]}((x_3 \geq 1) \mathbf{R}_{[0,\infty)}(y \leq 10))$
f2:	$\diamond_{[2,5]}(\square_{[0,3]}(x_2 < 4))$
f3:	$(\square_{[1,3]}(x \leq 2)) \mathbf{R}_{[2,5]}(x_3 > 2)$

### A.8 Spacecraft

There are two spaceships, chaser and target, in  $\mathbb{R}^2$ . The chaser tries to approach to the target [7]. The *target* spaceship rotates Earth at constant angular velocity  $\omega$ , and the *chaser* spaceship follows the target. The horizontal and vertical distance, between the target and the chaser are  $x$  and  $y$ , respectively, which change according to the relative velocities  $v_x$  and  $v_y$  with thrusts  $F_x$  and  $F_y$ . The dynamics is as follows:

$$\dot{x} = v_x, \quad \dot{y} = v_y, \quad \dot{v}_x = 3\omega^2 x + 2\omega v_y + \frac{F_x}{m_c}, \quad \dot{v}_y = -2\omega v_x + \frac{F_y}{m_c},$$

where  $m_c$  is the mass of the *chaser* spacecraft. The chaser tries to keep the distance between the two spaceship as small as possible.

Figure 33 shows a hybrid automaton of the rendezvous mission. There are three modes (Far, Close, and Recovery) of assigning different values to  $F_x$  and  $F_y$ . Initially, the (horizontal and vertical) distances  $x$  and  $y$  are between 60 and 70, and the *chaser* spacecraft starts its mission in the Far mode, moving toward the target. When each distance between the chaser and target is less than 50, the chaser decreases its thrust to keep the velocities  $v_x$  and  $v_y$  less than  $-4$  to avoid collision, resulting in the Close mode. When the velocity is still too fast, the chaser falls into the Recovery mode to change its thrust to the opposite direction.

Label	STL formula
f1:	$\square_{[0,2]}((v_x + v_y \leq -2) \rightarrow \diamond_{[0,3]}(v_x + v_y > 10))$
f2:	$\diamond_{[2,3]}(\square_{[1,2]}(x \leq 40))$
f3:	$\diamond_{[0,4]}(x \geq 30 \mathbf{U}_{[0,\infty)} v_x \leq -2)$

Table 19: STL properties for the spacecraft rendezvous

Table 19 shows the STL formulas for the spacecraft. Figure 34 is a model file of the spacecraft in STLMC.

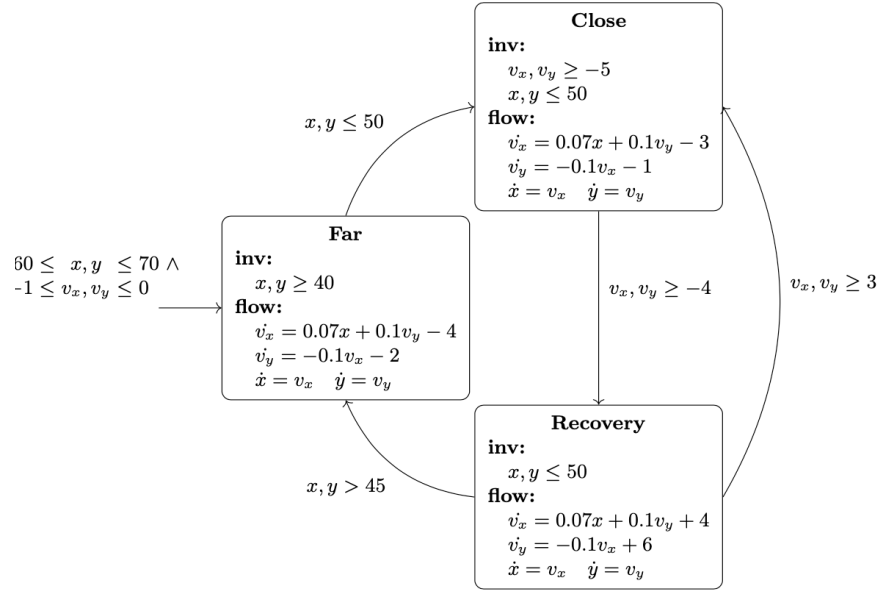


Fig. 33: A hybrid automaton of the spacecraft rendezvous

```

real m;
[-100, 100] x; [-100, 100] y;
[-30, 30] vx; [-30, 30] vy;
{ mode: m = 1;
  inv: x >= 40; y >= 40;
  flow: d/dt[x] = vx;
        d/dt[y] = vy;
        d/dt[vx] = 0.07 * x + 0.1 * vy - 4;
        d/dt[vy] = - 0.1 * vx - 2;
  jump: (or (x <= 50) (y <= 50)) =>
        (and (m' = 2) (x' = x) (y' = y)
             (vx' = vx) (vy' = vy));
}
{ mode: m = 2;
  inv: vx >= -5; vy >= -5;
      x <= 50; y <= 50;
  flow: d/dt[x] = vx;
        d/dt[y] = vy;
        d/dt[vx] = 0.07 * x + 0.1 * vy - 3;
        d/dt[vy] = - 0.1 * vx - 1;
  jump: (or (vx <= -4) (vy <= -4)) =>
        (and (m' = 3) (x' = x) (y' = y)
             (vx' = vx) (vy' = vy));
}
{ mode: m = 3;
  inv: x <= 50; y <= 50;
  flow: d/dt[x] = vx;
        d/dt[y] = vy;
        d/dt[vx] = 0.07 * x + 0.1 * vy + 4;
        d/dt[vy] = - 0.1 * vx + 6;
  jump: (and (x > 45) (y > 45)) =>
        (and (m' = 1) (x' = x) (y' = y)
             (vx' = vx) (vy' = vy));
        (or (vx >= 3) (vy >= 3)) =>
        (and (m' = 2) (x' = x) (y' = y)
             (vx' = vx) (vy' = vy));
}
init:
m = 1; 60 <= x; x <= 70; 60 <= y; y <= 70;
-1 <= vx; vx <= 0; -1 <= vy; vy <= 0;

proposition:
[xyvneg]: vx + vy <= -2;
[xyvpos]: vx + vy > 10;
[xyppos]: x + y >= 100;
[xypneg]: x + y <= 55;

# timebound : 5
# threshold: f1=1.5, f2=0.1, f3=1
# time-horizon: f1=2, f2=1.5, f3=2

goal:
[f1]: [] [0, 2] (xyvneg -> <> [0, 3] xyvpos);
[f2]: <> [2, 3] ( [] [1, 2] (x <= 40));
[f3]: [] [0, 2] (xyppos -> <> [1, 3] xypneg);

```

Fig. 34: An input model of the spacecraft rendezvous