stlMC: Python API for STL Bounded Model Checking

Jia Lee and Kyungmin Bae

Pohang University of Science and Technology (POSTECH), Pohang, South Korea
{cee5539,kmbae}@postech.ac.kr

Abstract. This document shows a prototype tool, called stlMC, that implements our STL model checking algorithm. We have defined a simple Python API to specify hybrid automata and STL formulas, and implemented functionality to perform STL bounded model checking and reachability analysis. In our tool, we use the Z3 solver to check the satisfiability of the generated formulas. We have conducted experiments on STL model checking of various polynomial hybrid automata, adapted from existing benchmarks on nonlinear hybrid systems.

1 Introduction

stlMC is a prototype Python implementation of our bounded model checking algorithm for signal temporal logic (STL) [1] that is refutationally complete for STL properties of bounded signals. Our algorithm is based on the reduction of a STL bounded model checking problem into the satisfiability of a first-order logic formula. The satisfiability of the resulting formula can be determined using the Z3 SMT solver [4]. *stlMC* includes a Python API to specify hybrid automata and STL formulas. It also provides functions to perform STL bounded model checking and reachability analysis of those models.

We have conducted several experiments on STL model checking of various systems, adapted from benchmarks on polynomial hybrid automata. We consider two variants for each of five different models (i.e., 10 in total): a polynomial model with nonlinear functions, and a simplified model with only linear functions.

The rest of this document is organized as follows. Section 2 describes how to install and run the tool. Section 3 explains a Python API in the tool to specify hybrid automata and STL formulas. Finally, Section 4 explains the details of the benchmark models and their STL requirements used in the experiments.

2 Using *stlMC*

2.1 Installation

Virtual Box Image. You can download the OVA file named “STLModelChecing.ova” from our webpage (http://sevlab.postech.ac.kr/popl/). To import the virtual machine, you need to start Oracle VM VritualBox Manager and select “Import Appliance” in the File menu. Browse to the location containing the “STLModelChecking.ova”, and click “Open”. After logging into the VM, you can run stlMC in Python3.

Source Code. *stlMC* is implemented in Python 3, and makes use of two libraries, namely, *Z3* and *antlr4*. The Z3 API can be installed using pip3 as follows:

```
$ pip3 install z3-solver
```

To install *antlr4*, see the instructions on the webpage (http://www.antlr.org). Our STL syntax is given by the ANTLR syntax *core/syntax/stl.g4*. You can generate the Python3 lexer and parser using the following command:

```
$ antlr4 -Dlanguage=Python3 stl.g4 -no-listener -visitor
```

2.2 Executing the Benchmarks

In the tool directory, the file *example.py* contains a simple example to show how to use and run our tool. For example, this file includes the following code:

```
1.  model = SingleCar()
2.
3.  # STL model checking
4.  formula = "[][0.0,60.0] (propleft /\ propright)"
5.  result  = model.modelCheck(formula, 2, 60)
6.  print("result: %s (size = %d, translation size = %d)" % result)
7.
8.  # reachability analysis
9.  goal = And(Real('x') == RealVal(50), Real('y') == RealVal(100))
10. result = model.reach(2, goal)
11. print("result: %s (size = %d)" % result)
```

Line 1 creates a hybrid automaton model by our Python API, and Line 4 defines an STL formula as a string (see Section 3 for more details). Line 5 performs bounded model checking of the formula up to step bound k = 2 and time bound $\tau_{max} = 60$. The result of *modelCheck* is a triple *(True/False, constraint size, FOL translation size)*. Similarly, Line 9 defines a reachability goal as a constraint, Line 5 performs bounded reachability analysis for the goal up to step bound k = 2, and the result is a pair of sat/unsat and the constraint size. The execution result of example.py is as follows:

```
$ python3 example.py
model checking: [][0.0,60.0] (propleft /\ propright)
    result: True (size = 9426, translation size = 32)
reachability checking: (and (= x 50) (= y 100))
    result: unsat (constration size = 2112)
```

We also provide the script *experimentRun.py* that executes all the benchmark models (in Section 5 of the paper [1]) at the same time. The execution results will be generated in the data directory as comma-separated values (CSV) files.

3 Python API for STL Specification

3.1 STL Syntax

STL formulas specify linear-time properties of real-valued signals (see [1] for details). In *stlMC*, an STL formula can be given as a string, which is then be parsed using the ANTLR parser. As mentioned, the syntax is defined in the file *core/syntax/stl.g4*.

An atomic formula can be any quantifier-free expressions. For simplicity reasons, *stlMC* requires that each atomic formula is defined as a Boolean signal. E.g., $x \geq 0$ is an atomic formula, and declared in *stlMC* as a Boolean signal $p \equiv x \geq 0$.

In STL, there are two kinds of logical connectives: Boolean operators and temporal operators, where temporal operators are further constrained by time intervals. The table below shows textual representations of those operators in *stlMC*.

Name	**Operator**	***stlMC* Notation**	**Name**	**Operator**	***stlMC* Notation**
Negation	$\sim$	`~`	Always	$\square_{[1,2)}$	`[] [1,2)`
Disjunction	$\vee$	`\/`	Eventually	$\Diamond_{(1,2)}$	`<> (1,2)`
Conjunction	$\wedge$	`/\`	Until	$\mathrm{U}_{(1,2]}$	`U (1,2]`
Implication	$\rightarrow$	`->`	Release	$\mathrm{R}_{[1,2]}$	`R [1,2]`

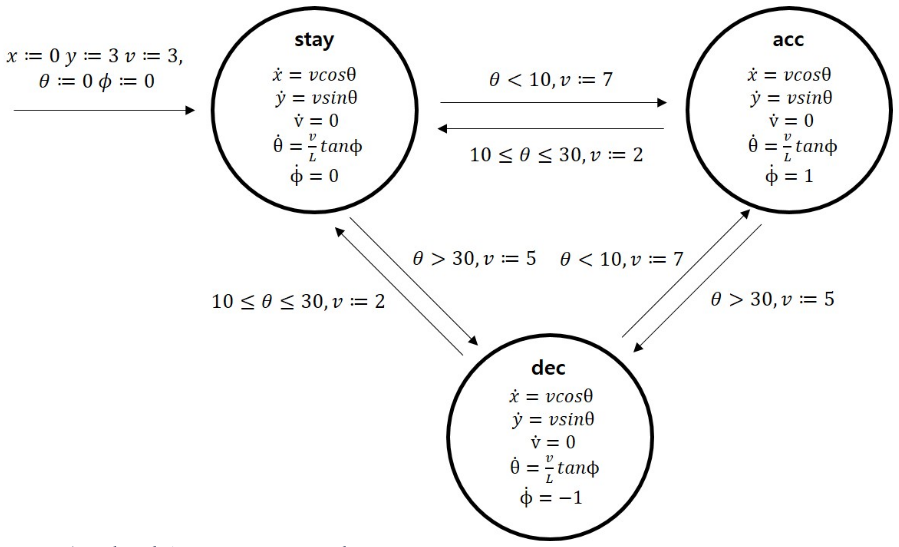
Notice that intervals can be either open, closed, or half-closed. For example, the STL formula "$\square_{[0.0,3.0]}(x \geq 0 \rightarrow \Diamond_{[0.5,2.0)} x < 0)$" can be expressed in *stlMC* as the string "`[] [0.0,3.0] (p -> <> [0.5,2.0) q)`", where $p \equiv x \geq 0$ and $q \equiv x < 0$.

3.2 API for Hybrid Automata Specification

Hybrid automata are basically state machines with continuous variables, and widely used for formalizing CPS programs. Formally, a hybrid automaton is defined as a tuple $H = (Q, X, init, inv, flow, jump)$, where

- Q is a finite set of modes;
- X is a finite set of real-valued variables;
- $init$ is an initial condition;
- inv is an invariant condition for each mode;
- $flow$ is a flow condition of X for each mode; and
- $jump$ is a jump condition between two modes in Q;

For example, Figure 1 describes a hybrid automaton for a simple autonomous car model, adapted from Example 4.3 in the paper [1]. The position (x, y) and the direction θ of the car changes according to the speed v and the steering angle ϕ.



$x \coloneqq 0\ y \coloneqq 3\ v \coloneqq 3,$
$\theta \coloneqq 0\ \phi \coloneqq 0$
stay
$\dot{x} = v\cos\theta$
$\dot{y} = v\sin\theta$
$\dot{v} = 0$
$\dot{\theta} = \frac{v}{L}\tan\phi$
$\dot{\phi} = 0$
$\theta < 10, v \coloneqq 7$
$10 \le \theta \le 30, v \coloneqq 2$
acc
$\dot{x} = v\cos\theta$
$\dot{y} = v\sin\theta$
$\dot{v} = 0$
$\dot{\theta} = \frac{v}{L}\tan\phi$
$\dot{\phi} = 1$
$\theta > 30, v \coloneqq 5$
$\theta < 10, v \coloneqq 7$
$10 \le \theta \le 30, v \coloneqq 2$
$\theta > 30, v \coloneqq 5$
dec
$\dot{x} = v\cos\theta$
$\dot{y} = v\sin\theta$
$\dot{v} = 0$
$\dot{\theta} = \frac{v}{L}\tan\phi$
$\dot{\phi} = -1$

Figure 1 Hybrod Automaton Example

Mode Conditions. Modes are defined using Boolean type variables. For example, three modes {stay, dec, acc} can be encoded using two Boolean variables mf and ms as follows, where $stay = {\sim}mf \wedge {\sim}ms$, $dec = {\sim}mf \wedge ms$, and $acc = mf \wedge {\sim}ms$.

```
mf = Bool('mf')
ms = Bool('ms')
```

Variables. Continuous variables are declared in *stlMC* as Real-typed variables, e.g.,

```
x = Real('x')
y = Real('y')
v = Real('v')
phi = Real('phi')
theta = Real('theta')
constv = Real('constv')
constphi = Real('constphi')
consttheta = Real('consttheta')
```

Initial Conditions. An initial condition is just declared in *stlMC* as a constraint over modes and variables. For example:

```
init = And(And(Not(mf), Not(ms)), x > RealVal(0), x < RealVal(3),
y < RealVal(10), y > RealVal(3), v >= RealVal(1), v <= RealVal(3),
 theta > RealVal(0), theta < RealVal(1), phi <= RealVal(1), phi >=
 RealVal(0), And(constv == v, constphi == phi, con-
sttheta == theta))
```

Flow Conditions. Flow conditions are declared as Python dictionaries with mode keys and ODE values. Ordinary differential equations (ODEs) are again declared as Python dictionaries with variable keys and equation values. For example:

```
flowx = constv * cos(consttheta)
flowy = constv * sin(consttheta)
flowtheta = constv / L * tan(constphi)
flowphi = ZERO

flow = {And(Not(mf), Not(ms)): \
        {x: flowx, y: flowy, theta: flowtheta, phi: flowphi,v: ZERO, constv: ZERO, constphi: ZERO, consttheta: ZERO}, \
        \
        And(Not(mf), ms): \
        {x: flowx, y: flowy, theta: flowtheta, phi: flowphi,v: ZERO, constv: ZERO, constphi: ZERO, consttheta: ZERO}, \
        \
        And(mf, Not(ms)): \
        {x: flowx, y: flowy, theta: flowtheta, phi: flowphi,v: ZERO, constv: ZERO, constphi: ZERO, consttheta: ZERO}}
```

As usual for hybrid automata, parameters of ODEs are declared using extra variables whose derivatives are 0. In the above example, *constv*, *constphi*, and *consttheta* are parameters of the ODEs, which are determined by jump conditions. Notice that the derivatives of these variables are 0 (i.e., *RealVal(0)*), and thus the values of the variables will only change when discrete jumps happen in the model.

Invariant Conditions. Invariant conditions are declared as Python dictionaries with mode keys and invariant constraint values. In addition, we can also declare the domain of each variable using Python dictionaries. For example:

```
vars = {x: (0, 100), y: (0, 100), v: (0, 10), theta: (-1.5, 1.5), phi: (-1, 1)}
inv = {And(Not(mf), Not(ms)): BoolVal(True), \
       And(Not(mf), ms): BoolVal(True), \
       And(mf, Not(ms)): BoolVal(True)}
```

STL Propositions. As mentioned, atomic formulas in STL are declared as propositions in *stlMC*. Those propositions are declared as Python dictionaries with Boolean variable keys and the corresponding constraint values. For example:

```
propLeft = Bool('propleft')
propRight = Bool('propright')
prop = {propLeft: v < RealVal(180), propRight: phi < RealVal(60)}
```

Jump Conditions. Jump conditions are declared as Python dictionaries with mode keys and jump constraint values. A jump constraint is a Boolean constraint over modes and variables, where the “next” modes/variables are declared using the *NextVar* function.

```
mfNext = NextVar(mf)
msNext = NextVar(ms)
xNext = NextVar(x)
yNext = NextVar(y)
vNext = NextVar(v)
phiNext = NextVar(phi)
thetaNext = NextVar(theta)
constvNext = NextVar(constv)
constphiNext = NextVar(constphi)
constthetaNext = NextVar(consttheta)

jump = {And(Not(mf), Not(ms), theta < RealVal(10)): \
        And(And(mfNext, Not(msNext)), vNext == RealVal(7), xNe
xt == x, yNext == y, phiNext == phi, the-
taNext == theta, And(constv == v, constphi == phi, con-
sttheta == theta)), \
        \
        And(Not(mf), Not(ms), theta > RealVal(30)): \
        And(And(mfNext, msNext), vNext == RealVal(-
5), xNext == x, yNext == y, phiNext == phi, the-
taNext == theta, And(constv == v, constphi == phi, con-
sttheta == theta)), \
        \
        And(Not(mf), ms, theta >= RealVal(10), theta <= RealVa
l(30)): \
        And(Not(mf), Not(ms), vNext == RealVal(2), xNext == x,
 yNext == y, phiNext == phi, thetaNext == theta, And(con-
stv == v, constphi == phi, consttheta == theta)), \
        \
        And(Not(mf), ms, theta < RealVal(10)): \
        And(And(mfNext, Not(msNext)), vNext == RealVal(7), xNe
xt == x, yNext == y, phiNext == phi, the-
taNext == theta, And(constv == v, constphi == phi, con-
sttheta == theta)), \
        \
        And(mf, Not(ms), theta >= RealVal(10), theta <= RealVa
l(30)): \
        And(Not(mf), Not(ms), vNext == RealVal(2), xNext == x,
 yNext == y, phiNext == phi, thetaNext == theta, And(con-
stv == v, constphi == phi, consttheta == theta)), \
        \
        And(mf, Not(ms), theta < RealVal(10)): \
        And(And(Not(mfNext), msNext), vNext == RealVal(7), xNe
xt == x, yNext == y, phiNext == phi, the-
taNext == theta, And(constv == v, constphi == phi, con-
sttheta == theta))}
```

4 Benchmark Models

4.1 Networked thermostat controllers (adapted from [2])

There are two rooms and they are interconnected by an open door. The temperature x_i of each room$_i$ is separately controlled by each thermostat, depending on both the heater's mode $m_i \in \{on, off\}$ and the temperature of the adjacent room. Initially, the temperature x_1 of the room$_1$ is $19.9 \leq x_1 \leq 20.1$, and the temperature x_2 of the room$_2$ is $23 \leq x_2 \leq 23.5$. For both rooms, the heaters are initially off.

The behavior of each heater is as follows. If the temperature is higher than the mean value of both temperatures, the heater will be turned off. Otherwise, the heater will be turned on. The temperatures change according to the following ODEs.

1. Linear dynamics (the solutions of the ODEs are linear functions)

$$\dot{x_1} = \begin{cases} -0.4 & if\ m_1 = off \\ 0.7 & if\ m_1 = on \end{cases} \qquad \dot{x_2} = \begin{cases} -0.6 & if\ m_2 = off \\ 1 & if\ m_2 = on \end{cases}$$

2. Polynomial dynamics (the solutions of the ODEs are polynomials)

$$\dot{x_1} = \begin{cases} -K_1\big((1-2c)cx_1 + c * cx_2\big) & if\ m_1 = off \\ K_1\Big(h_1 - \big((1-2c)cx_1 + c * cx_2\big)\Big) & if\ m_1 = on \end{cases} \qquad c\dot{x}_1 = c\dot{x}_2 = 0$$

where $c, K_i, h_i \in \mathbb{R}$ are constants depending on the size of the door, the size and the heater's power. The variable cx_i is a constant (with derivative 0) that captures the value of x_1 at a certain moment, so that the dynamics approximately becomes a polynomial. The dynamics of x_2 is similar to one of x_1. The following STL formulas are considered.

STL formula	ID	Explanation
$\Diamond_{[0,40]} x_1 > 21$	φ_1	Within 40 time units, x_1 become greater than 21.
$\square_{[0,20]}((on_1 \land on_2) \rightarrow \square_{[0,2]}(x_2 \leq 20))$	φ_2	For 20 time units, if both heaters are on, x_2 is less than or equal to 20 within 2 time units.
$\Diamond_{[0,10]}((x_2 \leq 20) U_{[0,5]} (on_1 \land on_2))$	φ_3	Within 20 time units, x_2 is less than or equal to 20, until both heaters are on within 10 time units.
$\square_{[0,20]}((off_1 \land off_2) \rightarrow \Diamond_{[0,5]}(on_1 \lor on_2))$	φ_4	For 20 time units, if both heaters are off, one of them is turned on within 5 time units.
$\square_{[5,50]}(on_1 \rightarrow \square_{[0,30]}\big(x_1 \geq 20\ U_{(0,5]} x_1 < 20\big))$	φ_5	For time interval [5,50], if the room$_1$'s heater is on, x_1 is always less than 20 within 5 time units.
$\square_{[0,20]}(x_1 > 10 \land x_1 < 30 \land x_2 > 10 \land x_2 < 30)$	φ_6	For 20 time units, both temperatures are always less than 30 and greater than 10.

4.2 Networked water tank controllers (adapted from [2])

There are two water tanks and they are connected by a pipe. The water level $\mathrm{x_i}$ of each tank is controlled by each pump, depending on the pump's mode $\mathrm{m_i} \in \{on, off\}$ and the water level of the adjacent water tank. Initially, the water level of each water tank is higher than 4.9 and less than 5.1, and the both pumps are on.

The behavior of each pump is as follows. If the water level $\mathrm{x_1}$ is less than 1, the pump$_1$ is turned on. If the water level $\mathrm{x_1}$ is greater than the differences of both water levels, the pump$_1$ is turned off. For pump$_2$, it is on only if $\mathrm{x_2}$ is less than 1. The water level of each tank changes according to the following dynamics.

1. Linear dynamics

$$\dot{x_1} = \begin{cases} -0.2 & if\ m_1 = off \\ 0.5 & if\ m_1 = on \end{cases} \qquad \dot{x_2} = \begin{cases} -0.3 & if\ m_2 = off \\ 0.6 & if\ m_2 = on \end{cases}$$

2. Polynomial dynamics (using Tylor approximation of square root at t = 1)

$$\dot{x_1} = \begin{cases} -(ag * cx_1)/2A_1 & if\ \ m_1 = off \\ (q_1 - ag * cx_1)/2A_1 & if\ \ m_1 = on \end{cases}$$

$$\dot{x_2} = \begin{cases} (ag * (cx_1 - cx_2))/2A_2 & if\ \ m_1 = off \\ (q_2 + ag * (cx_1 - cx_2))/2A_2 & if\ \ m_1 = on \end{cases}$$

where $A_i, q_i, a \in \mathbb{R}$ are constants determined by the size of the tank, the power of the pump, and the width of the pipe, and g is the standard gravity constant. . The variable cx_i is a constant (with derivative 0) that captures the value of x_1 at a certain moment, so that the dynamics is a polynomial. The following STL formulas are considered.

STL formula	**ID**	**Explanation**
$\Diamond_{[0,40]} x_1 > 5.1$	φ_1	Within 40 time units, the water level x_1 becomes greater than 5.1.
$\Box_{[0,10]}(x_1 < 4.9 \to \Diamond_{[0,10]}(x_1 \geq 5.1))$	φ_2	For 10 time units, whenever x_1 is less than 4.9, x_1 becomes greater than or equal to 5.1 within 10 time units.
$\Box_{[0,10]}(x_1 < 5 \to \Box_{[0,5]}\Diamond_{[0,5]} on_1)$	φ_3	For 10 time units, if x_1 is less than 5, for 5 time units, the pump$_1$ can be on within 5 time units.
$\Box_{[0,20]}((on_1 \wedge \mathrm{on}_2) \to \Diamond_{[0,5]}(\mathrm{off}_1 \vee off_2))$	φ_4	For 20 time units, whenever both pumps are on, one of them is off within 5 time units.
$\Box_{[0,50]}(x_1 > 0 \wedge x_1 \leq 9 \wedge x_2 > 0 \wedge x_2 \leq 9)$	φ_5	For 50 time units, both water levels are greater than 0 and less than or equal to 9.
$\Diamond_{[5,15)}(off_1 \to \Diamond_{[0,7]}(on_1 \wedge x_1 > 5.5))$	φ_6	Within time interval [5,15), if pump$_1$ is off, pump$_1$ is on and x_1 is greater than 5.5 within 7 time units.
$\Diamond_{[50,100)}((off_1 \wedge x_2 \geq 5.5) \to \Box_{[3,4]}(on_2 \wedge x_2 < 5.2))$	φ_7	Within time interval [50,100), if pump$_1$ is off and x_2 is greater than or equal to 5.5, then pump$_2$ is on and x_2 is less than 5.5 for time interval [3,4].

4.3 Driving Simple Cars

4.3.1 Linear dynamics (adapted from [5])

Two cars car_1 and car_2 are running in a straight road, while car_1 follows car_2. The velocity of each car depends on the distance between two cars. Each car can move at different velocities, depending on its mode $m_i \in \{fast, mid, slow\}$. Initially, both cars are in the $fast$ mode, where the position x_1 of car₁ is in [0, 1], and the position x_2 of car₂ is in [3, 10]. The dynamics of two cars are as follows.

$$\dot{x_1} = \begin{cases} 70 & if\ m_1 = fast \\ 40 & if\ m_1 = mid \\ 33 & if\ m_1 = slow \end{cases} \qquad \dot{x_2} = \begin{cases} 60 & if\ m_2 = fast \\ 40 & if\ m_2 = mid \\ 35 & if\ m_2 = slow \end{cases}$$

The mode of each car is determined by the distance $x_2 - x_1$. For example, if the distance is less than 1, car_1 moves slow and car_2 moves fast. If the distance is greater than or equal to 3 and less than 4, car_1 moves fast and car_2 moves at normal speed. If the distance is greater than or equal to 5, car_1 accelerates and car_2 decelerates. We consider the following STL formulas for this model.

STL formula	**ID**	**Explanation**
$\Diamond_{[0,40]}(x_2 - x_1) \leq 3$	φ_1	Within 40 time units, the distance between car_1 and car_2 becomes less than or equal to 3.
$\Diamond_{[0,20]}((x_2 - x_1 < 4) \rightarrow \Diamond_{[0,5]}(mid_1 \wedge fast_2))$	φ_2	Within 40 time units, if the distance becomes less than 4, within 5 time units, car_1 moves at normal speed and car_2 becomes fast.
$\Diamond_{[0,20]}(\Box_{[0,5]}\ x_2 - x_1 < 4)$	φ_3	At some point within 20 time units, the distance is always less than 4 for 5 time units.
$\Box_{[0,10]}((mid_1 \wedge fast_2) \rightarrow \Diamond_{[0,5]}(\neg mid_1 \vee \neg \mathrm{fast}_2))$	φ_4	For 10 time units, if car_1 moves at normal speed and car_2 moves fast, one of them changes its mode within 5 time units.
$\Box_{[0,100]}\ (x_2 > x_1)$	φ_5	For 100 time units, car_2 is always ahead of car_1.
$\Box_{(0,30]}(\Diamond_{[5,10]}(mid_1 \wedge fast_2))$	φ_6	For 30 time units, within time interval [5,10], car_1 moves at normal speed and car_2 moves fast.
$\Diamond_{(10,40]}((\mathrm{x}_2 - x_1 \geq 4.5) \rightarrow \Box_{[0,10]}\ dec_2)$	φ_7	Within time interval (10,40], if the distance is greater than or equal to 4.5, car_2 moves slowly during 10 time units.

4.3.2 Polynomial dynamics (adapted from [6])

Two cars are running in sequence, while each car follows the behavior of the car in front (the first car moves according to its own scenario). Each car can rotate and it can move at different velocities, depending on its mode $m_i \in \{stay, acc, dec\}$. Initially, the value of each variables is as follows.

Variables	**Initialized value**
the position x_1 of car_1	$0 < x_1 < 3$
the position y_1 of car_1	$3 < y_1 < 10$
the position x_2 of car_2	$5 < x_2 < 10$
the position y_2 of car_2	$3 < y_2 < 10$
the velocity v_1 of the car_1	$1 \leq v_1 \leq 3$
the velocity v_2 of the car_2	$3 \leq v_2 \leq 4$
the direction θ_1 of car_1	$0 < \theta_1 < 1$
the direction θ_2 of car_2	$-1 < \theta_2 < 0$
the steering angle $\phi_1$of car_1	$0 < \phi_1 < 1$
the steering angle $\phi_2$of car_2	$-1 < \phi_2 < 0$

The behavior of each car is as follows. If the $distance(car_1, car_2)$ is $distance(car_1, car_2) < 6$, then the car_2 changes its velocity to -5. If the distance is $6 \leq distance(car_1, car_2) \leq 9$, then the car_2 changes its velocity to $-(v_2 - v_1)$. If the distance is $distance(car_1, car_2) > 9$, then the car_2 changes its velocity to 5. The dynamics of two cars are as follows.

We used Tylor approximation of linearization of trigonometric functions at t = 2.)

$$\dot{x_i} = v_i \cos\theta_i, \ \ \dot{y_i} = v_i \sin\theta_i, \ \ \dot{\theta_i} = v_i \tan\phi_i$$

$$\dot{\phi_2} = \phi_2 - \phi_1$$

$$\dot{v_2} = \begin{cases} -(cv_2 - cv_1) & if\ m_2 = stay \\ 5 & if\ m_2 = acc \\ -5 & if\ m_2 = dec \end{cases}$$

The variable cx_i is a constant (with derivative 0) that captures the value of x_i at a certain moment, so that the dynamics is a polynomial. The values of ϕ_1 and v_1 change depending on the scheduled scenario of the car_1. The following STL formulas are considered.

STL formula	**ID**	**Explanation**
$\lozenge_{[0,20]} dist(car_1, car_2) < 6$	φ_1	Within 20 time units, the distance between car_1 and car_2 is less than 6.

$\Diamond_{[0,20]}((dist(car_1,car_2) \geq 10) \rightarrow \square_{[0,10]}\ (dist(car_1,car_2) \geq 10))$	φ2	Within 20 time units, if the distance is greater than or equal to 10, the distance is always greater than or equal to 10 during 10 time units.
$\square_{[0,20]}((dist(car_1,car_2) < 6) \rightarrow \Diamond_{[0,15]}\ \mathrm{acc}_2)$	φ3	For 20 time units, whenever the distance is smaller than 6, the car_2 increases its acceleration within 15 time units.
$\square_{[0,20]}\left(\begin{pmatrix} abs(x_1 - x_2) \leq 0.5 \\ \wedge\, abs(y_1 - y_2) \leq 0.5 \end{pmatrix}\right)$	φ4	For 20 time units, the difference of each position should be less than or equal to 0.5.
$\square_{[0,100)}(dist(car_1,car_2) > 0.1)$	φ5	For 100 time units, the distance is always greater than 0.1.
$\square_{[0,50)}(\Diamond_{[5,15]}acc_2)$	φ6	For 50 time units, within time interval [5,15], the car_2 increases its acceleration.
$\Diamond_{(4,40)}(\square_{[10,20]}(dist(car_1,car_2) > 3))$	φ7	At some point within time interval (4,40), the distance is greater than 3 for time interval [10,20].
$\Diamond_{[12,20]}(dec_2 \rightarrow \Diamond_{[3,18]}acc_2)$	φ8	Within time interval [12,20], if the car_2 decreases its acceleration, within time interval [3,18], the car_2 increases its acceleration.

4.4 Railroad (adapted from [3])

This is a system of modeling a crossing barrier controller and a train on a track. There is a circular railroad track and there is a crossing barrier on the track. The tracks are 100 meters long. A train is going around and around the track. The angular between ground and the crossing barrier $theta$ changes depending on its mode $t \in \{Far, Approach, Near, Past\}$. Initially, the relative distance of the train to the barrier $distance(t,b)$ is $60 \leq distance(t,b) \leq 70$, the train far away from the crossing barrier.

The mode of the train is determined by the distance between the train and the crossing barrier. If the distance is $distance(t,b) \geq 50$, then the train's mode is Far. If the $distance(t,b)$ is $40 \leq distance(t,b) < 50$, then the train's mode is $Approach$. If the $distance(t,b)$ is $20 \leq distance(t,b) < 30$, then the train's mode is $Near$. If the $distance(t,b)$ is $-5 \leq distance(t,b) < 0$, then the train's mode is $Past$. If the $distance(t,b)$ is $-10 \leq distance(t,b) < -5$, then the train's mode is Far and the relative distance is updated to '$100 + current\ relative\ distance'$. The dynamics of the train and the crossing barrier are as follows.

1. Linear dynamics

$$\dot{theta} = \begin{cases} 0 & if\ t = Far \\ 5 & if\ t = Approach \\ 10 & if\ t = Near \\ -5 & if\ t = Past \end{cases} \qquad \dot{trainPosition} = -5$$

2. Polynomial dynamics

$$\dot{theta} = \begin{cases} 0 & if\ t = Far \\ cv_{approach} & if\ t = Approach \\ cv_{near} & if\ t = Near \\ cv_{past} & if\ t = Past \end{cases} \qquad \dot{v}_t = \begin{cases} 5 & if\ t = Approach \\ 10 & if\ t = Near \\ -5 & if\ t = Past \end{cases}$$

$$\dot{trainPosition} = -5$$

where $trainPosition$ is the current degree between the crossing bar and ground. The variable cv_i is a constant (with derivative 0) that captures the value of v_i at a certain moment, so that the dynamics is a polynomial. The following STL formulas are considered.

STL formula	**ID**	**Explanation**
$\Diamond_{(10,50)}(pos \leq 0)$	φ_1	Within time interval (10,50), the position of the train is less than or equal to zero.
$\Box_{[20,40)}((angle \geq 80) \rightarrow \Diamond_{[1,20]}\ (pos \leq 0))$	φ_2	For time interval [20,40), whenever the angle is greater than or equal to 80 degrees, train passes the crossing bar within time interval [1,20].
$\Box_{(10,40]}((angle \geq 80) \rightarrow \Diamond_{[1,20]}(angle \leq 40))$	φ_3	For time interval(10,40], whenever the angle is bigger than 80 degrees, the degree will be less than 40 degrees within time interval [1,20].
$\Box_{[0,40]}(passing \rightarrow \Box_{[0,7]}passing)$	φ_4	For 40 time units, whenever the train pass the crossing bar, it keeps its mode within 7 time units.
$\Box_{[0,100]}((pos < 5 \wedge pos \geq -2) \rightarrow (angle \geq 80)$	φ_5	For 100 time units, if the position of the train is greater than or equal to -2 and less than 5, the angle becomes greater than or equal to 80.
$\Box_{[3,50]}(\Diamond_{[5,20]}(angle \geq 80))$	φ_6	For time interval [3,50], within time interval [5,20], the angle is greater than or equal to 80.
$\Diamond_{[10,60]}((angle \geq 80) \rightarrow \Box_{[20,40]}(\mathrm{angle} < 60))$	φ_7	Within time interval [10,60], whenever the angle is greater than or equal to 80, the angle is always smaller than 60 during time interval [20, 40].
$\Diamond_{[10,50]}(\Box_{[20,30]}(angle \leq 10))$	φ_8	At some point within time interval [10,50], the angle is always less than or equal to 10 for time interval [20,30].

4.5 Battery(adapted from [2])

There are two of fully charged batteries, and a control system switches load between these batteries to achieve longer lifetime out of the batteries. There are three modes $\mathrm{m}_i \in \{on, off, dead\}$ for each battery. Initially, the total energy of $battery_1$ is 8.5 and $battery_2$ is 7.5. The both batteries are switched on.

The behavior of each battery is as follows. If the total energy of the battery is smaller than $(1-c) * its\ kinetic\ energy$, $c \in [0,1]$ is threshold, then the battery is dead. Otherwise, the battery can be either turned on or turned off. The dynamics of two batteries are as follows.

1. Linear dynamics

$$\dot{d}_1 = \dot{d}_2 = \frac{1}{2C}, \qquad \dot{g}_1 = \dot{g}_2 = -\frac{1}{2} \qquad if\ m_1 = on, m_2 = on$$

$$\dot{d}_i = \frac{0.7}{C}, \dot{d}_j = -C, \quad \dot{g}_i = -1, \dot{g}_j = 0 \qquad if\ m_i = on, m_j = off, i \neq j, i, j \in \{1,2\}$$

$$\dot{d}_i = \frac{0.7}{C}, \dot{d}_j = 0, \quad \dot{g}_i = -1, \dot{g}_j = 0 \qquad if\ m_i = on, m_j = dead, i \neq j, i, j \in \{1,2\}$$

$$\dot{d}_1 = \dot{d}_2 = 0, \quad \dot{g}_1 = \dot{g}_2 = 0 \qquad if\ m_1 = on, m_2 = on$$

2. Polynomial dynamics

$$\dot{d}_i = \frac{1}{2C} - 2 * cd_i, \quad \dot{g}_1 = \dot{g}_2 = -\frac{1}{2} \qquad if\ m_1 = on, m_2 = on$$

$$\dot{d}_i = \frac{1}{C} - 2 * cd_i, \dot{d}_j = -2 * cd_j, \ \dot{g}_i = -1, \dot{g}_j = 0, \qquad if\ m_i = on, m_j = off, i \neq j, i, j \in \{1,2\}$$

$$\dot{d}_i = \frac{1}{C} - 2 * cd_i, \dot{d}_j = 0, \quad \dot{g}_i = -1, \dot{g}_j = 0 \qquad if\ m_i = on, m_j = dead, i \neq j, i, j \in \{1,2\}$$

$$\dot{d}_1 = \dot{d}_2 = 0, \quad \dot{g}_1 = \dot{g}_2 = 0 \qquad if\ m_1 = on, m_2 = on$$

where variable d_i is its kinetic energy, variable g_i is its total charge, and constant $C \in [0,1]$ is its threshold. The variable cd_i is a constant (with derivative 0) that captures the value of d_i at a certain moment, so that the dynamics is a polynomial. The following STL formulas are considered.

STL formula	**ID**	**Explanation**
$\Diamond_{[10,50]} g_1 \leq 1$	φ₁	Within 40 time units, the total charge of the $battery_1$ becomes less than or equal to 1.
$\Diamond_{(5,30]}((\mathrm{live}_1 \wedge live_2) \rightarrow \Box_{[7.5,25)}(live_1 \wedge live_2))$	φ₂	Within time interval (5,30], if both are switched on, they keep their mode for time interval [7.5,25).
$\Box_{(10,50]}((g_1 \geq 0 \vee g_2 \geq 0)\ U_{(1,15)}\ (dead_1 \wedge dead_2))$	φ₃	For time interval (10,50], the total energy of both are greater than or equal to zero, until both are dead within time interval (1,15).
$\Box_{(10,40.8)}(\mathrm{dead}_1 \rightarrow \Box_{[0.5,30)}\mathrm{live}_2)$	φ₄	For time interval (10,40.8), if the $battery_1$ is dead, the $battery_2$ is always switched on for time interval [0.5,30). during the time interval.
$\Box_{(0,20.5)}(\Diamond_{[3,14]}(d_1 \geq 1.4))$	φ₅	For 20.5 time units, the kinetic energy of the $battery_1$ is greater than or equal to 1.4 for time interval [3,14].
$\Box_{(10,60)}(live_1 \rightarrow \Diamond_{[0,5]}\mathrm{dead}_1)$	φ₆	For time interval (10,60), whenever the $battery_1$ is switched on, then the $battery_1$ is dead within 5 time units.

$\Diamond_{(10,50]}(\Box_{(0,20)}(g_1 < 0 \lor g_2 < 0))$	φ_7	At some point within time interval (10,50], the total charge of both are smaller than 0 for 40 time units.
$\Box_{(0,50)}(d_1 > 0.5 \land d_2 > 0.5)$	φ_8	For 50 time units, the kinetic energy of both are always greater than 0.5.

References

1. Kyungmin Bae and Jia Lee: Symbolic Model Checking of Signal Temporal Logic Properties Using Syntactic Separation, the attached manuscript, 2018.
2. Kyungmin Bae and Sicun Gao: Modular SMT-Based Analysis of Nonlinear Hybrid Systems. In Formal Methods in Computer-Aided Design (2017), pp 180-187.
3. T. A. Henzinger: The theory of hybrid automata. In Logic in Computer Science (1996).
4. Leonardo De Moura and Nikolaj Bjørner: Z3: An efficient SMT solver. In: TACAS, Springer (2008), pp. 337–340.
5. A. Platzer. Logical analysis of hybrid systems: proving theorems for complex dynamics. Springer Science & Business Media, 2010.
6. Steven M La Valle: Planning algorithms. Cambridge university press (2006).