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 [2]. *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.ovf", 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:

```
$ java -jar antlr-4.7.1-complete.jar -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: False (size = 607, translation size = 5)
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	$\Box_{[1,2)}$	`[] [1,2)`
Disjunction	$\vee$	`\/`	Eventually	$\Diamond_{(1,2)}$	`<> (1,2)`
Conjunction	$\wedge$	`/\`	Until	$U_{(1,2]}$	`U (1,2]`
Implication	$\rightarrow$	`->`	Release	$R_{[1,2]}$	`R [1,2]`

Notice that intervals can be either open, closed, or half-closed. For example, the STL formula "$\Box_{[0.0,3.0]}(x \geq 0 \rightarrow \Diamond_{[0.5,2.0)} x < 0)$" 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.

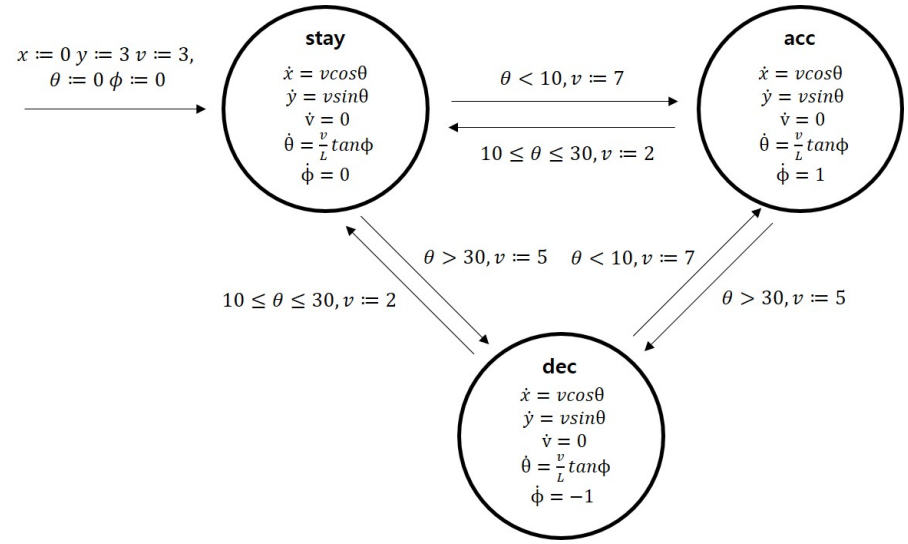

Figure 1 Hybrod Automaton Example

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 $acc = mf \wedge \sim ms$.

```
1.  mf = Bool('mf')
2.  ms = Bool('ms')
```

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

```
1.  x = Real('x')
2.  y = Real('y')
3.  v = Real('v')
4.  phi = Real('phi')
5.  theta = Real('theta')
6.  constv = Real('constv')
7.  constphi = Real('constphi')
8.  consttheta = Real('consttheta')
```

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

```
1.  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), xNext == x, yNext == y, phiNext == phi, thetaNext == theta, And(constv == v, constphi == phi, consttheta == theta)), \
        \
        And(Not(mf), Not(ms), theta > RealVal(30)): \
        And(And(mfNext, msNext), vNext == RealVal(-5), xNext == x, yNext == y, phiNext == phi, thetaNext == theta, And(constv == v, constphi == phi, consttheta == theta)), \
        \
        And(Not(mf), ms, theta >= RealVal(10), theta <= RealVal(30)): \
        And(Not(mf), Not(ms), vNext == RealVal(2), xNext == x, yNext == y, phiNext == phi, thetaNext == theta, And(constv == v, constphi == phi, consttheta == theta)), \
        \
        And(Not(mf), ms, theta < RealVal(10)): \
        And(And(mfNext, Not(msNext)), vNext == RealVal(7), xNext == x, yNext == y, phiNext == phi, thetaNext == theta, And(constv == v, constphi == phi, consttheta == theta)), \
        \
        And(mf, Not(ms), theta >= RealVal(10), theta <= RealVal(30)): \
        And(Not(mf), Not(ms), vNext == RealVal(2), xNext == x, yNext == y, phiNext == phi, thetaNext == theta, And(constv == v, constphi == phi, consttheta == theta)), \
        \
        And(mf, Not(ms), theta < RealVal(10)): \
        And(And(Not(mfNext), msNext), vNext == RealVal(7), xNext == x, yNext == y, phiNext == phi, thetaNext == theta, And(constv == v, constphi == phi, consttheta == theta))}
```

4 Benchmark Models

4.1 Networked thermostat controllers

There are two rooms, 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 of each room is higher than 19 degrees and less than 21 degrees, and the heater is off. Every thermostat controller synchronously performs its discrete transitions with common action a. The number of actions of each thermostat controller is two. One of them is "The heater is turned off". It is performed when the temperature of the room is higher than 20. The other is "The heater is turned on". It is performed when the temperature of the room is smaller than or equal to 20. The temperature of each room(x_i) is changed according to the following functions:

(i) Linear ODE

$$\dot{x_\iota} = -K_i\left((1-2c)x_i + c\sum_{j\in A_i} constx_j\right) \; if \; q_i = m_{off}$$

$$\dot{x_\iota} = -K_i\left((1-2c)x_i + c\sum_{j\in A_i} constx_j\right) \; if \; q_i = m_{off}$$

$$\dot{constx_\iota} = 0$$

(ii) Polynomial ODE

$$\dot{x_\iota} = K_i\left(h_i - \left((1-2c)x_i + c\sum_{j\in A_i} x_j\right)\right) \; if \; q_i = m_{on}$$

$$\dot{x_\iota} = -K_i\left((1-2c)x_i + c\sum_{j\in A_i} x_j\right) \; if \; q_i = m_{off}$$

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 of $room_i$, A_i is set of adjacent rooms of $room_i$. The safety property is that the temperature of each room is in the range (10, 30). We designed the following STL formulas for the model.

STL formula	ID	Explanation
$\Diamond_{[0,40]} x_1 > 21$	φ_1	The temperature of the $room_1$ is greater than 21 degrees within 40 time units.
$\Box_{[0,20]}((on_1 \wedge on_2) \rightarrow \Box_{[0,2]}(x_2 \leq 20))$	φ_2	For 20 time units, if the heater's mode of each room is both on, then the temperature of the

		$room_2$ is smaller than or equal to 20 within 2 time units.
$\Diamond_{[0,10]}((x_2 \leq 20) \; U_{[0,5]}(on_1 \wedge on_2))$	φ_3	The temperature of the $room_2$ should be less than or equal to 20 degrees until the heater's mode of each room is both on within 10 time units.
$\Box_{[0,20]}((off_1 \wedge off_2) \rightarrow \Diamond_{[0,5]}(on_1 \vee on_2))$	φ_4	For 20 time units, if the heater's mode of each room is both off, then either one of them eventually is turned on within 5 time units
$\Box_{[5,50]}(on_1 \rightarrow \Box_{[0,30]}(x_1 \geq 20 \; U_{(0,5]} x_1 < 20))$	φ_5	For 45 time units, if the heaters' mode of the $room_1$ is on, then the temperature of the $room_1$ is always smaller than 20 within 5 time units.
$\Box_{[0,20]}(x_1 > 10 \wedge x_1 < 30 \wedge x_2 > 10 \wedge x_2 < 30)$	φ_6	For 20 time units, the temperature of the each room is always smaller than 30 and bigger than 10.

4.2 Networked water tank controllers

There are two water tanks, they are connected by a pipe. The water level x_i of each tank is separately controlled by each pump, depending on both the pump's mode $m_i \in \{on, off\}$ 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 pump is off. Every water tank controller synchronously performs its discrete transitions with common action a. The number of actions of each water tank controller is two. One of them is "The pump is turned off". It is performed when the water level of the tank is greater than or equal to 5. The other is "The pump is turned on". It is performed when the water level of the tank is smaller than 5. The water level of each tank(x_i) is changed according to the ODEs.

(i) Linear ODE

$$A_i \dot{x_i} = \left(q_i + a\sqrt{2g}\sqrt{constx_{i-1}}\right) - a\sqrt{2g}\sqrt{constx_i} \quad if \;\; m_i = m_{on}$$

$$A_i \dot{x_i} = a\sqrt{2g}\sqrt{constx_{i-1}} - a\sqrt{2g}\sqrt{constx_i} \quad if \;\; m_i = m_{off}$$

$$\dot{constx_i} = 0$$

(ii) Polynomial ODE

$$A_i \dot{x_i} = \left(q_i + a\sqrt{2g}\sqrt{x_{i-1}}\right) - a\sqrt{2g}\sqrt{x_i} \quad if \;\; m_i = m_{on}$$

$$A_i \dot{x_i} = a\sqrt{2g}\sqrt{x_{i-1}} - a\sqrt{2g}\sqrt{x_i} \quad if \;\; m_i = m_{off}$$

Where $A_i, q_i, a \in \mathbb{R}$ are constants determined by the size of the tank, the power of the pump, the width of the I/O pipe, and g is the standard gravity constant. The safety property is that the temperature of each room is in the range $(0, 9]$. We designed the following STL formulas for the model.

STL formula	ID	Explanation

$\Diamond_{[0,40]} x_1 > 5.1$	φ_1	The water level of the $tank_1$ is greater than 5.1 within 40 time units.
$\Box_{[0,10]}(x_1 < 4.9 \rightarrow \Diamond_{[0,10]}(x_1 \geq 5.1))$	φ_2	For 10 time units, if the water level of the $tank_1$ is less than 4.9, then the level is greater than or equal to 5.1 within 10 time units eventually.
$\Box_{[0,10]}(x_1 < 5 \rightarrow \Box_{[0,5]}\Diamond_{[0,5]} on_1)$	φ_3	For 10 time units, if the water level of the $tank_1$ is less than 5, then pump of the $tank_1$ should be always turned on eventually within 5 time units.
$\Box_{[0,20]}((on_1 \wedge \mathrm{on}_2) \rightarrow \Diamond_{[0,5]}(\mathrm{off}_1 \vee off_2))$	φ_4	For 20 time units, if the pump's mode of each tank is both on, then either one of them is turned 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, the water level of each tank is bigger than 0 and smaller than or equal to 9.
$\Diamond_{[5,15)}(off_1 \rightarrow \Diamond_{[0,7]}(on_1 \wedge x_1 > 5.5))$	φ_6	For 10 time units, if the $tank_1$ pump is off, the $tank_1$ pump is on and the water level of the $tank_1$ is greater than 5.5 within 7 time units.
$\Diamond_{[50,100)}((off_1 \wedge x_2 \geq 5.5) \rightarrow \Box_{[3,4]}(on_2 \wedge x_2 < 5.2))$	φ_7	For 50 time units, if the $tank_1$ pump is off and the water level of the $tank_2$ is greater than or equal to 5.5, then the $tank_2$ pump is on and the water level of the $tank_2$ is smaller than 5.5 during the interval [3,4].

4.3 Driving Simple Cars

4.3.1 Linear ODE

Two cars are running in a straight road. The car_2 precedes the car_1. The velocity of each car depends on the distance between car_1 and car_2. There are three modes $q_i \in \{keep, acc, dec\}$ for each car. Initially, the position of car_1 is in range [0, 1], car_2 is [3, 10] and the mode of each car is 'acc'. If the distance is less than 1, the mode of $car_1(q_1)$ is changed to 'dec' and car_2 is changed to 'acc'. If the distance is bigger than or equal to 1 and smaller than 2, the mode of $car_1(q_1)$ is changed to 'stay' and car_2 is changed to 'acc'. If the distance is bigger than or equal to 2 and smaller than 3, the mode of $car_1(q_1)$ is changed to 'acc' and car_2 is changed to 'acc'. If the distance is bigger than or equal to 3 and smaller than 4, the mode of $car_1(q_1)$ is changed to 'acc' and car_2 is changed to 'stay'. If the distance is bigger than or equal to 4 and smaller than 5, the mode of $car_1(q_1)$ is changed to 'stay' and car_2 is changed to 'dec'. If the distance is bigger than or equal to 5, the mode of $car_1(q_1)$ is changed to 'acc' and car_2 is changed to 'dec'. The value of x_i are changed according to the ODEs:

$$\dot{x_i} = v_i$$

$$\dot{v_1} = 4 \;\; if \; q_1 = acc$$

$$\dot{v_1} = 0.3 \ \ if \ q_1 = dec$$

$$\dot{v_2} = 3 \ \ if \ q_2 = acc$$

$$\dot{v_2} = 0.5 \ \ if \ q_2 = dec$$

$$\dot{v_1} = \dot{v_2} = 1 \ \ if \ q_1 = stay$$

The safety property is that the car_2 always precede car_1. We designed the following STL formulas for the model.

STL formula	ID	Explanation
$\Diamond_{[0,40]}(x_2 - x_1) \leq 3$	φ_1	The distance between the car_1 and the car_2 is less than or equal to 3 within 40 time units.
$\Diamond_{[0,20]}(((x_2 - x_1) < 4) \rightarrow \Diamond_{[0,5]}(stay_1 \wedge acc_2))$	φ_2	At some time, if the distance between the car_1 and the car_2 is less than 4, then the car_1 keeps its velocity, and the car_2 raise its velocity within 5 time units.
$\Diamond_{[0,20]}(\Box_{[0,5]}((x_2 - x_1) < 4))$	φ_3	The distance between car_1 and car_2 is always less than 4 within 20 time units.
$\Box_{[0,10]}((stay_1 \wedge acc_2) \rightarrow \Diamond_{[0,5]}(\neg \mathrm{stay}_1 \vee \neg \mathrm{acc}_2))$	φ_4	For 10 time units, if the car_1 keeps its velocity, and the car_2 raise its velocity, then one of them changes its velocity within 5 time units.
$\Box_{[0,100]}(x_2 > x_1)$	φ_5	For 100 time units, the car_2 is always front of the car_1
$\Box_{(0,30]}(\Diamond_{[5,10]}(stay_1 \wedge dec_2))$	φ_6	For 30 time units, the car_1 keeps its velocity and car_2 decreases its velocity within 5 time units.
$\Diamond_{(10,40]}(((\mathrm{x}_2 - x_1) \geq 4.5) \rightarrow \Box_{[0,10]}\, dec_2)$	φ_7	For 30 time units, if the distance between the car_1 and the car_2 is greater than or equal to 4.5, then the car_2 decreases its velocity during 10 time units.

4.3.2 Polynomial ODE

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). There are three modes $q_i \in \{keep, acc, dec\}$ for each car. The position (x_i, y_i) and the direction θ_i of each car_i depends on its seed v_i, and steering angle ϕ_i. Initially, the x position of car_1 is in the interval (0,3) and y position is in the interval (3,10). The x position of car_2 is in the interval (5,10) and y position is in the interval (3,10). The velocity of car_1 and car_2 are in [3,4]. The direction of car_1 is in the interval (0,1) and car_2 is in the interval (-1,0). The steering angle of car_1 is in the interval (0,1) and car_2 is in the interval (-1,0). We used taylor series for linearization of trigonometric functions. For each step, if the distance between two cars are bigger than 6 smaller than 9, then the mode of car(q_i) is 'stay'. If the distance between two cars are smaller than 9, then the mode of car(q_i) is

'dec'. If the distance between two cars are bigger than 9, then the mode of car(q_i) is 'acc'. The values of variable $x_i, y_i, \theta_i, \phi_i$ are changed according to the ODEs:

$$\dot{x}_i = v_i \cos\theta_i, \quad \dot{y}_i = v_i \sin\theta_i, \quad \dot{\theta}_i = v_i \tan\phi_i$$

$$\dot{\phi}_1 = 0, \quad \dot{\phi}_2 = \phi_2 - \phi_1$$

$$\dot{v}_1 = 0$$

$$\dot{v}_2 = -(v_2 - v_1) \quad if\ q_i = stay$$

$$\dot{v}_2 = 5 \quad if\ q_i = acc$$

$$\dot{v}_2 = -5 \quad if\ q_i = dec$$

The goal is to find mode change schedule for driving cars safely. We designed following STL formulas for the model:

STL formula	ID	Explanation
$\Diamond_{[0,20]} dist(car_1, car_2) < 6$	φ_1	The distance between the car_1 and the car_2 is less than 6 within 40 time units.
$\Diamond_{[0,20]}((dist(car_1, car_2) \geq 10) \rightarrow \Box_{[0,10]}\ (dist(car_1, car_2) \geq 10))$	φ_2	At some time, if the distance between the car_1 and the car_2 is greater than 10, then the distance is greater than or equal to 10 for 10 time units.
$\Box_{[0,20]}((dist(car_1, car_2) < 6) \rightarrow \Diamond_{[0,15]}\ \mathrm{acc}_2)$	φ_3	For 20 time units, if the distance between the car_1 and the car_2 is smaller than 6, then the car_2 turns to the right
$\Box_{[0,20]}(\begin{pmatrix} abs(x_1 - x_2) \leq 0.5 \\ \wedge\ abs(y_1 - y_2) \leq 0.5 \end{pmatrix})$	φ_4	For 20 time units, the difference of each position of car_1 and car_2 should be less than or equal to 0.5.
$\Box_{[0,100)}(dist(car_1, car_2) > 0.1)$	φ_5	For 100 time units, the distance between the car_1 and the car_2 is always greater than 0.1.
$\Box_{[0,50)}(\Diamond_{[5,15]} acc_2)$	φ_6	For 50 time units, the car_2 increases its velocity during the interval [5,15].
$\Diamond_{(4,40)}(\Box_{[10,20]}(dist(car_1, car_2) > 3))$	φ_7	During the interval (4,40), the distance between the car_1 and the car_2 is greater than 3 for 100 time units.
$\Diamond_{[12,20]}(dec_2 \rightarrow \Diamond_{[3,18]} acc_2)$	φ_8	During the interval [12,20], if the car_2 decreases its velocity, then the car_2 increases its velocity within 15 time units.

4.4 Railroad

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 size of the track is 100. A train is going around and around the track. There are four modes $t \in \{Far, Approach, Near, Past\}$ for the train. The barrier opens when the train approaches to the barrier and closes when the train passes the barrier. The variable $theta$ is angular between ground and the barrier. If $theta$ is 0, then the barrier closes com-

pletely. If $theta$ is 90, then the barrier opens completely. The angular velocity of barrier is constant and the value of velocity is different according to train's mode. Initial relative distance of the train to the barrier is greater than or equal to 60 and less than or equal to 70, and the mode of train(t) is 'Far'. If the relative distance is greater than or equal to 40 and smaller than 50, then the mode is changed to 'Approach'. If the relative distance is greater than or equal to 20 and smaller than 30, then the mode is changed to 'Near'. If the relative distance is greater than or equal to -5 and smaller than or equal to 0, then the mode is changed to 'Past'. If the relative distance is greater than or equal to -10 and smaller than or equal to -5, then the mode is changed to 'Approach' and the relative distance is updated to '$100 + current\ relative\ distance$'. The values of variables are changed according to the ODEs

(i) Linear ODE

$$\dot{theta} = 0 \quad if\ t = Far$$
$$\dot{theta} = 5 \quad if\ t = Approach$$
$$\dot{theta} = 10 \quad if\ t = Near$$
$$\dot{theta} = -5 \quad if\ t = Past$$
$$\dot{bx} = \dot{theta}$$

(ii) Polynomial ODE

$$\dot{theta} = 0 \quad if\ t = Far$$
$$\dot{theta} = \dot{vbx}_{approach} \quad if\ t = Approach$$
$$\dot{vbx}_{approach} = 5$$
$$\dot{theta} = \dot{vbx}_{near} \quad if\ t = Near$$
$$\dot{vbx}_{near} = 10$$
$$\dot{theta} = \dot{vbx}_{past} \quad if\ t = Past$$
$$\dot{vbx}_{past} = -5$$
$$\dot{bx} = \dot{theta}$$

where bx is the current degree between the crossing bar and ground. We designed the following STL formulas for the model.

STL formula	ID	Explanation
$\Diamond_{(10,50)}(pos \leq 0)$	φ_1	The position of the train is less than or equal to zero within time units.
$\Box_{[20,40)}((angle \geq 80) \rightarrow \Diamond_{[1,20]}(pos \leq 0))$	φ_2	For 40 time units, if the bar is almost open, then train will cross the bar.
$\Box_{(10,40]}((angle \geq 80) \rightarrow \Diamond_{[1,20]}(angle \leq 40))$	φ_3	If the angle between bar and ground is bigger than 80 degrees, the degree will be smaller than 40 degrees in the future.
$\Box_{[0,40]}(passing \rightarrow \Box_{[0,7]}passing)$	φ_4	For 40 time units, if the train pass the crossing bar, then it stays its mode during 7 time units.

$\square_{[0,100]}((pos < 5 \wedge pos \geq -2) \rightarrow (angle \geq 80))$	φ_5	For 100 time units, if the position of the train is greater than or equal to -2 and less than 5, then the angle is greater than or equal to 80.
$\square_{[3,50]}(\Diamond_{[5,20]}(angle \geq 80))$	φ_6	For 47 time units, the angle is greater than or equal to 80 within 15 time units.
$\Diamond_{[10,60]}((angle \geq 80) \rightarrow \square_{[20,40]}(\text{angle} < 60))$	φ_7	For 50 time units, if the angle is greater than or equal to 80, then the angle is always smaller than 60 during the interval [20, 40] eventually.
$\Diamond_{[10,50]}(\square_{[20,30]}(angle \leq 10))$	φ_8	For 40 time units, the angle is always smaller than or equal to 10 during the interval [20,30] eventually.

4.5 Battery

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 $b_i \in \{on, off, dead\}$ for each battery. Initially, the total energy of $battery_1$ is 8.5 and $battery_2$ is 7.5. The mode of each battery is 'on'. If the total energy of battery is smaller than specific value, the battery's mode becomes dead. Otherwise, the mode can be either 'on' or 'off'. The values of variable are changed according to the ODEs:

(i) Linear ODE

$$\dot{d_1} = \dot{d_2} = \frac{1}{2C}, \dot{g_1} = \dot{g_2} = -\frac{1}{2} \quad if\ b_1 = on, b_2 = on$$

$$\dot{d_i} = \frac{0.7}{C}, \dot{d_j} = -C, \dot{g_i} = -1, \dot{g_j} = 0 \quad if\ b_i = on, b_j = off, i \neq j, i, j \in \{1,2\}$$

$$\dot{d_i} = \frac{0.7}{C}, \dot{d_j} = 0, \dot{g_i} = -1, \dot{g_j} = 0 \quad if\ b_i = on, b_j = dead, i \neq j, i, j \in \{1,2\}$$

$$\dot{d_1} = \dot{d_2} = 0, \dot{g_1} = \dot{g_2} = 0 \quad if\ b_1 = dead, b_2 = dead$$

(ii) Polynomial ODE

$$\dot{d_i} = \frac{1}{2C} - 2 * constd_i, \dot{g_1} = \dot{g_2} = -\frac{1}{2} \quad if\ b_1 = on, b_2 = on$$

$$\dot{d_i} = \frac{1}{C} - 2 * constd_i, \dot{d_j} = -2 * constd_j, \dot{g_i} = -1, \dot{g_j} = 0 \quad if\ b_i = on, b_j = off, i \neq j, i, j \in \{1,2\}$$

$$\dot{d_i} = \frac{1}{C} - 2 * constd_i, \dot{d_j} = 0, \dot{g_i} = -1, \dot{g_j} = 0 \quad if\ b_i = on, b_j = dead, i \neq j, i, j \in \{1,2\}$$

$$\dot{d_1} = \dot{d_2} = 0, \dot{g_1} = \dot{g_2} = 0 \quad if\ b_1 = dead, b_2 = dead$$

$$\dot{constd_i} = 0 \quad i \in \{1,2\}$$

with variable d_i is its kinetic energy, variable g_i is its total charge, and constant $C \in [0,1]$ is its threshold. We designed the following STL formulas for the model.

STL formula	ID	Explanation
$\Diamond_{[10,50]} g_1 \leq 1$	φ_1	The total charge of the $battery_1$ is less than or equal to 1 within 40 time units.
$\Diamond_{(5,30]}((\text{live}_1 \wedge live_2) \rightarrow \Box_{[7.5,25)}(live_1 \wedge live_2))$	φ_2	If two batteries are alive and used, then the system doesn't make mode change during 17.5 time units.
$\Box_{(10,50]}((g_1 \geq 0 \vee g_2 \geq 0) \; U_{(1,15)} \; (dead_1 \wedge dead_2))$	φ_3	For 40 time units, the total energy of both batteries are greater or equal to zero until they both dead.
$\Box_{(10,40.8)}(\text{dead}_1 \rightarrow \Box_{[0.5,30)}\text{live}_2)$	φ_4	If the $battery_1$ is dead during the time intervals (10,40.8), then the $battery_2$ is always alive during the time interval [0.5,30).
$\Box_{(0,20.5)}(\Diamond_{[3,14]}(d_1 \geq 1.4))$	φ_5	For 20.5 time units, the kinetic energy of the $battery_1$ is greater than or equal to 1.4 within 11 time units.
$\Box_{(10,60)}(live_1 \rightarrow \Diamond_{[0,5]}\text{dead}_1)$	φ_6	For 50 time units, if the $battery_1$ is alive, then $battery_1$ will be dead within 5 time units.
$\Diamond_{(10,50]}(\Box_{(0,20)}(g_1 < 0 \vee g_2 < 0))$	φ_7	For 40 time units, the total charge of the $battery_1$ or $battery_2$ is smaller than 0 during the interval (10,20) eventually.
$\Box_{(0,50)}(d_1 > 0.5 \wedge d_2 > 0.5)$	φ_8	For 50 time units, the kinetic energy of $battery_1$ and $battery_2$ is 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. Leonardo De Moura and Nikolaj Bjørner: Z3: An efficient SMT solver. In: TACAS, Springer (2008), pp. 337–340.