DryVR Documentation

Release 2.0

Chuchu Fan, Bolun Qi

Jan 31, 2018

CONTENTS

Release 2.0

Date 01/30/2018

DryVR is a framework for verifying cyber-physical systems. It specifically handles systems that are described by a combination of a *Black-box Simulator* for trajectories and a white-box *Transition Graph* specifying mode switches. The framework uses a probabilistic algorithm for learning sensitivity of the continuous trajectories from simulation data and includes a bounded reachability analysis algorithm that uses the learned sensitivity.

CHAPTER

ONE

STATUS

Jan 24.2018. DryVR 2.0 is done. Adding state dependent transition and control synthesis.

April 18.2017. The installation is tested on Ubuntu 16.04 (64 bit version).

March 23.2017. The tool is tested on Ubuntu 16.04 (64 bit version).

CHAPTER

TWO

INSTALLATION

To install the required packages, please run:

```
sudo ./installRequirement.sh
```

The current version of installation file has been tested on a clean install of Ubuntu 16.04. If you wish to install DryVR on other versions of Linux operation system, please make sure the following packages are correctly installed.

To install packages indepently, the following will be required:

- python 2.7
- numpy
- scipy
- sympy
- matplotlib
- python igraph
- python Z3
- glpk(4.39 or ealier eversion)
- pyglpk
- python-cairo
- python tk
- gmpc
- graphviz
- pygraphviz

CHAPTER

THREE

USAGE

3.1 Run DryVR Verfication

To run DryVR verfication, please run:

```
python main.py input/*/[input_file]
```

for example:

```
python main.py input/daginput/input_thermo.json
```

3.2 Run DryVR Control Synthesis

To run DryVR graph search algorithm, please run:

```
python rrt.py input/*/[input_file]
```

for example:

```
python rrt.py input/rrtinput/mazefinder.json
```

3.3 Plotter

After you run the our tool, a reachtube.txt file will be generated in output folder unless the model is determined unsafe during simulation test.

To plot the reachtube, please run:

```
python plotter.py -x [x dimension number] -y [y dimension number list] -f [input file␣
↪name] -o [output file name]
```

-x is the dimension number for x-axis, the default value will be 0, which is the dimension of time.

-y is dimension number lists indicates the dimension you want to draw for y-axis. For example -y [1,2]. The default value will be [1].

-f is the file path for reach tube file that you want to plot, the default value will be output/reachtube.txt.

-o is output file option, the default value is plotResult.png.

To get help for plotter, please run:

```
python plotter.py -h
```

Note that the dimension 0 is local time and last dimension is global time. For example, input_AEB's inital set is [[0.0,-23.0,0.0,1.0,0.0,-15.0,0.0,1.0],[0.0,-22.8,0.0,1.0,0.0,-15.0,0.0,1.0]]. Therefore, it has 8 dimensions in total. You can choose to plot dimension from 0 to 9. Where dimension 0 is the local time and dimension 9 is global time. Dimension 1~8 is corresponding to the dimension you specify in initial set.

for example:

```
python plotter.py -y [1,2] -f output/reachtube.txt
```

More plot results can be found at the *Examples* page.

CHAPTER

FOUR

DRYVR'S LANGUAGE

In DryVR, a hybrid system is modeled as a combination of a white-box that specifies the mode switches (*Transition Graph*) and a black-box that can simulate the continuous evolution in each mode (*Black-box Simulator*).

4.1 Black-box Simulator

The black-box simulator for a (deterministic) takes as input a mode label, an initial state x_0, and a finite sequence of time points $t_1, \ldots, t_k$, and returns a sequence of states $sim(mode, x_0, t_1), \ldots, sim(mode, x_0, t_k)$ as the simulation trajectory of the system in the given mode starting from x_0 at the time points $t_1, \ldots, t_k$.

DryVR uses the black-box simulator by calling the simulation function:

```
TC_Simulate(Modes,initialCondition,time_bound)
```

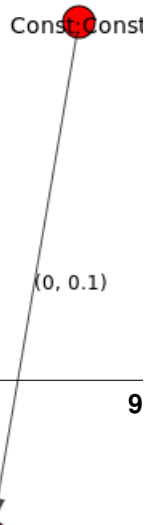
Given the mode name "Mode", initial state "initialCondition" and time horizon "time_bound", the function TC_Simulate should return an python array of the form:

```
[[t_0,variable_1(t_0),variable_2(t_0),...],[t_1,variable_1(t_1),variable_2(t_1),...],.
↪..]
```

We provide several example simulation functions and you have to write your own if you want to verify systems that use other black-boxes. Once you create the TC_Simulate function and corresponding input file, you can run DryVR to check the safety of your system. To connect DryVR with your own black-box simulator, please refer to section *Advanced Tricks: Verify your own black-box system* for more details.

4.2 Transition Graph

A transition graph is a labeled, directed acyclic graph as shown on the right. The vertex labels (red nodes in the graph) specify the modes of the system, and the edge labels specify the transition time from the predecessor node to the successor node.



The transition graph shown on the right defines an automatic emergency braking system. Car1 is driving ahead of Car2 on a straight lane. Initially, both car1 and car2 are in the constant speed mode (Const;Const). Within a short amount of time ([0,0.1]s) Car1 transits into brake mode while Car2 remains in the cruise mode (Brk;Const). After [0.8,0.9]s, Car2 will react by braking as well so both cars are in the brake mode (Brk;Brk).

The transition graph will be generated automatically by DryVR and stored in the tool's root directory as curgraph.png

4.3 Input Format

The input for DryVR is of the form

```
{
  "vertex":[transition graph vertex labels (modes)]
  "edge":[transition graph edges, (i,j) means there is a directed edge from vertex i␣
↪to vertex j]
  "variables":[the name of variables in the system]
  "guards":[transition graph edge labels (transition condition)]
  "resets":[reset condition after transition] # This is optional if you do not want␣
↪reset
  "initialMode":[label for initial mode] # This is optional for DAG graph
  "initialSet":[two arrays defining the lower and upper bound of each variable]
  "unsafeSet":@[mode name]:[unsafe region]
  "timeHorizon":[Time bound for the verification]
  "directory": directory of the folder which contains the simulator for black-box␣
↪system
  "bloatingMethod": specify the bloating method, which can be either "PW" or "GLOBAL"
↪# This is optional, if you don't have this field in input file, DryVR will use␣
↪GLOBAL as default bloating method.
  "kvalue": specify the k-value that used by piecewise bloating method # This field␣
↪must be specified if you choose the bloatingMethod to "PW"
}
```

Some fields are optional in DryVR's input langauge such as resets, initialMode, bloatingMethod and kvalue under some conditions. Please read the comment.

Example input for the Automatic Emergency Braking System

```
{
  "vertex":["Const;Const","Brk;Const","Brk;Brk"],
  "edge":[[0,1],[1,2]],
  "variables":["car1_x","car1_y","car1_vx","car1_vy","car2_x","car2_y","car2_vx",
↪"car2_vy"],
  "guards":[
    "And(t>0.0,t<=0.1)",
    "And(t>0.8,t<=0.9)"
  ],
  "initialSet":[[0.0,0.5,0.0,1.0,0.0,-17.0,0.0,1.0],[0.0,1.0,0.0,1.0,0.0,-15.0,0.0,1.
↪0]],
  "unsafeSet":"@Allmode:And(car1_y-car2_y<3, car2_y-car1_y<3)",
  "timeHorizon":5.0,
  "directory":"examples/cars"
}
```

4.4 Output Interpretation

The tool will print background information like the current mode, transition time, initial set on the run. The final result about goal reached/cannot find graph will be printed at the bottom.

When the system find transition graph, the final result will look like

```
System is Safe!
```

When the system is unsafe from simulation, the final result will look like

```
Current simulation is not safe. Program halt
```

When the system is unsafe from verification, the final result will look like

```
System is not safe in Mode [Mode name]
```

When the system is unknown from verification, the final result will look like

```
Hit refine threshold, system halt, result unknown
```

If the simulation result is not safe, the unsafe simulation trajectory will be stored in "output/Traj.txt". Otherwise the last simulation result will be stored in "Traj.txt".

If the verfication result is not safe, the counter example reachtube will be stored in "output/unsafeTube.txt".

4.5 Advanced Tricks: Verify your own black-box system

We use a very simple example of a thermostat as the starting point to show how to use DryVR to verify your own black-box system.

The thermostat is a one-dimensional linear hybrid system with two modes "On" and "Off". The only state variable is the temperature x. In the "On" mode, the system dynamic is

$$\dot{x} = 0.1x,$$

and in the "Off" mode, the system dynamic is

$$\dot{x} = -0.1x,$$

As for DryVR, of course, all the information about dynamics is hidden. Instead, you need to provide the simulator function TC_Simulate as discussed in *Black-box Simulator*.

Step 1: Create a folder in the DryVR root directory for your new model and enter it.

```
cd examples
mkdir Thermostats
cd Thermostats
```

Step 2: Inside your model folder, create a python script for your model.

```
touch Thermostats_ODE.py
```

Step 3: Write the TC_Simulate function in the python file Thermostats_ODE.py.

For the thermostat system, one simulator function could be:

```
def thermo_dynamic(y,t,rate):
    dydt = rate*y
    return dydt

def TC_Simulate(Mode,initialCondition,time_bound):
    time_step = 0.05;
    time_bound = float(time_bound)
    initial = [float(tmp)  for tmp in initialCondition]
    number_points = int(np.ceil(time_bound/time_step))
    t = [i*time_step for i in range(0,number_points)]
    if t[-1] != time_step:
        t.append(time_bound)

    y_initial = initial[0]

    if Mode == 'On':
        rate = 0.1
    elif Mode == 'Off':
        rate = -0.1
    else:
        print('Wrong Mode name!')
    sol = odeint(thermo_dynamic,y_initial,t,args=(rate,),hmax = time_step)

    # Construct the final output
    trace = []
    for j in range(len(t)):
        tmp = []
        tmp.append(t[j])
        tmp.append(sol[j,0])
        trace.append(tmp)
    return trace
```

In this example, we use odeint simulator from Scipy, but you use any programming language as long as the TC_Simulate function follows the input-output requirement:

```
TC_Simulate(Mode,initialCondition,time_bound)
Input:
    Mode (string) -- a string indicates the model you want to simulate. Ex. "On"
```

```
    initialCondition (list of float) -- a list contains the initial condition. Ex.
↪"[32.0]"
    time_bound (float) -- a float indicates the time horizon for simulation. EX. '10.0
↪'
Output:
    Trace (list of list of float) -- a list of lists contain the trace from a␣
↪simulation.
    Each index represents the simulation for certain time step.Represents as [time,␣
↪v1, v2, ........].
    Ex. "[[0.0,32.0],[0.1,32.1],[0.2,32.2].........[10.0,34.3]]"
```

Step 4: Inside your model folder, create a Python initiate script.

```
touch __init__.py
```

Inside your initiate script, import file with function TC_Simulate.

```
from Thermostats_ODE import *
```

Step 5: Go to inputFile folder and create an input file for your new model using the format discussed in *Input Format*.

Create a transition graph specifying the mode transitions. For example, we want the temperature to start within the range $[75, 76]$ in the "On" mode. After $[1, 1.1]$ second, it transits to the "Off" mode, and transits back to the "On" mode after another $[1, 1.1]$ seconds. For bounded time $3.5s$, we want to check whether the temperature is above 90.

The input file can be written as:

```
{
  "vertex":["On","Off","On"],
  "edge":[[0,1],[1,2]],
  "variables":["temp"],
  "guards":["And(t>1.0,t<=1.1)","And(t>1.0,t<=1.1)"],
  "initialSet":[[75.0],[76.0]],
  "unsafeSet":"@Allmode:temp>91",
  "timeHorizon":3.5,
  "directory":"examples/Thermostats"
}
```

Save the input file in the folder input/daginput and name it as input_thermo.json.

Step6: Run the verification algorithm using the command:

```
python main.py input/daginput/input_thermo.json
```

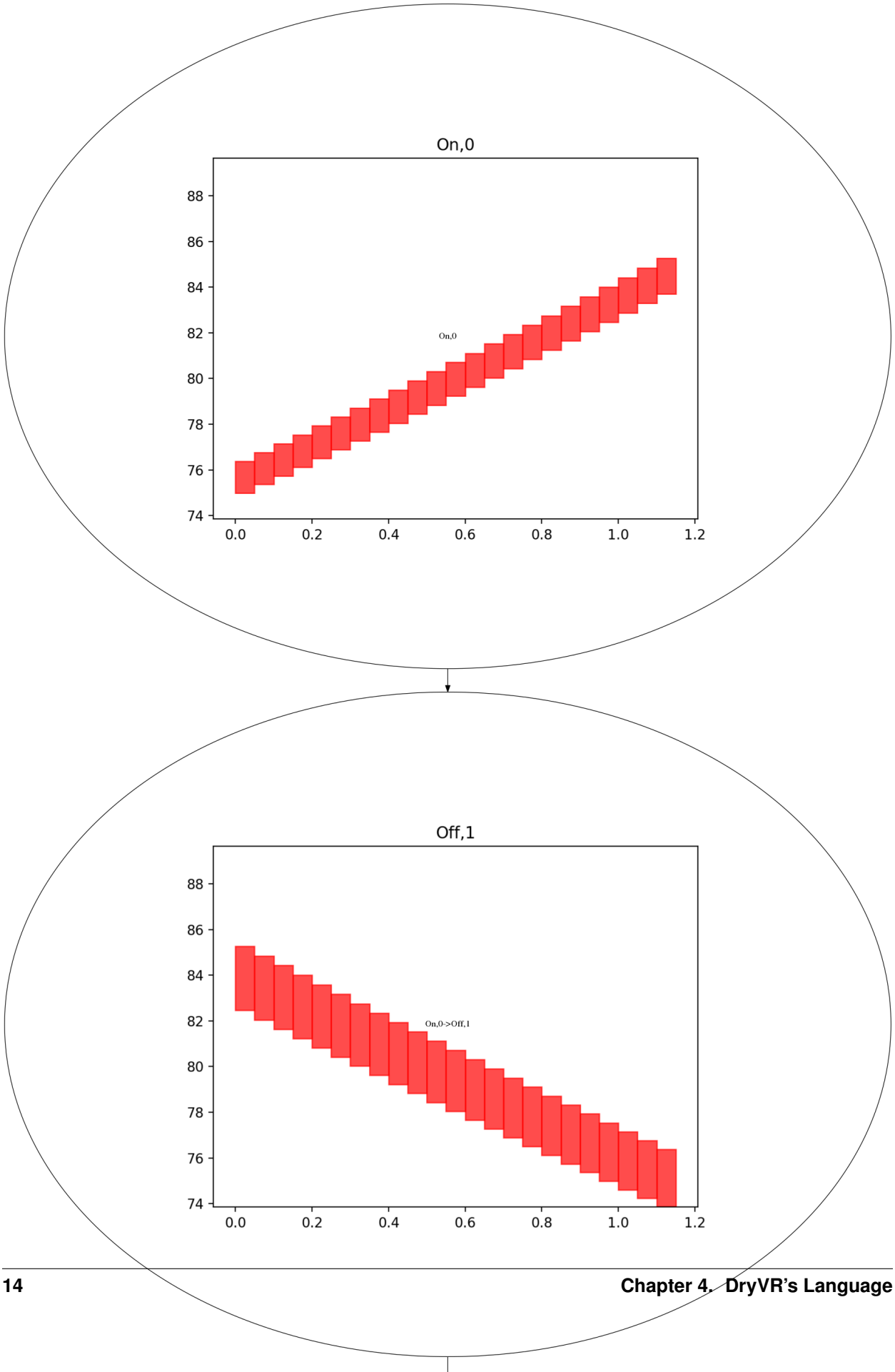
The system has been checked to be safe with the output:

```
System is Safe!
```

We can plot the reachtube using the command:

```
python plotter.py
```

And the reachtube for the temperature is shown as


On,0
88
86
84
82
80
78
76
74
On,0
0.0
0.2
0.4
0.6
0.8
1.0
1.2
Off,1
88
86
84
82
80
78
76
74
On,0->Off,1
0.0
0.2
0.4
0.6
0.8
1.0
1.2

CHAPTER

FIVE

DRYVR'S CONTROL SYNTHESIS

In DryVR, a hybrid system is modeled as a combination of a white-box that specifies the mode switches (*Transition Graph*) and a black-box that can simulate the continuous evolution in each mode (*Black-box Simulator*).

The control synthesis problem for DryVR is to find a white-box transition graph given the black-box simulator with addition inputs listed in (*Input Format*).

5.1 Input Format

The input for DryVR control synthesis is of the form

```
{
  "modes":[modes that black simulator takes]
  "initialMode":[initial mode that DryVR start to search]
  "variables":[the name of variables in the system]
  "initialSet":[two arrays defining the lower and upper bound of each variable]
  "unsafeSet":@[mode name]:[unsafe region]
  "goalSet":[two arrays defining the lower and upper bound of each variable for goal]
  "timeHorizon":[time bound for control synthesis, the graph should be bounded in␣
↪time horizon]
  "directory": directory of the folder which contains the simulator for black-box␣
↪system
  "minTimeThres": minimal staying time for each mode to limit number of trainsition.
  "goal":[[goal variables],[lower bound][upper bound]] # This is a rewrite for goal␣
↪set for dryvr to calculate distance.
}
```

Example input for the robot in maze example

```
{
  "modes":["0", "1", "2", "3", "4", "5", "6", "7"],
  "initialMode":"1",
  "variables":["x","y","vx","vy"],
  "initialSet":[[1.0,1.0,1.0,1.0],[1.1,1.0,1.0,1.0]],
  "unsafeSet":"@Allmode:Or(And(x>=2.0, x<3.0, y>=3.0, y<=4.0), And(x>=3.0, x<=4.0, y>
↪=2.0, y<3.0), x<0, x>5, y<0, y>5)",
  "goalSet":"And(x>=3.0, x<=4.0, y>=3.0, y<=4.0)",
  "timeHorizon":10.0,
  "minTimeThres":1.0,
  "directory":"examples/carinmaze",
  "goal":[["x","y"],[3.0,3.0],[4.0,4.0]]
}
```

5.2 Output Interpretation

The tool will print background information like the current mode, transition time, initial set on the run. The final result about goal reached or not reached will be printed at the bottom.

When the system find the transition graph that statisfy the requirement, the final result will look like

```
goal reached
```

When the system cannot find graph, the final result will look like

```
could not find graph
```

Note that DryVR's algorithm is searching the graph randomly, if the system cannot find the graph, it does not mean the graph is not exist with current input. You can try run the algorithm multiple times to get more accurate result. If the the system find the transition graph, the system will plot the transition graph and will be stored in "output/rrtGraph.png"

5.3 Advanced Tricks: Making control synthesis work on your own black-box system

Creating black box simulator is exactly same as we introduced in DryVR's language page (*Advanced Tricks: Verify your own black-box system*) up to Step 4.

For the Step 5, instead of creating a verification input file, you need to create control synthesis input file we have discussed in *Input Format*.

For example, Let's set the intial temperature within the range $[75, 76]$, and we want to reach the target temperature within the range $[68, 72]$, while avoiding temperature that is larger than 90. We want to start our search from "On" mode and reach our goal in bounded time $4s$, and set the minimal staying time to $1s$.

the input file can be written as:

```
{
  "modes":["On", "Off"],
  "initialMode":"On",
  "variables":["temp"],
  "initialSet":[[75.0],[76.0]],
  "unsafeSet":"@Allmode:temp>90",
  "goalSet":"And(temp>=68.0, temp<=72.0)",
  "timeHorizon":4.0,
  "minTimeThres":1.0,
  "directory":"examples/Thermostats",
  "goal":[["temp"],[68.0],[72.0]]
}
```

Save the input file in the folder input/rrtinput and name it as temp.json.

Run the graph search algorithm using the command:

```
python rrt.py input/rrtinput/temp.json
```

The graph has been found with the output:

```
goal reached!
```

If you check the the output/rrtGraph.png, you would get a transition graph for this problem. As you can see the system turn from On state to Off state to reach the goal.

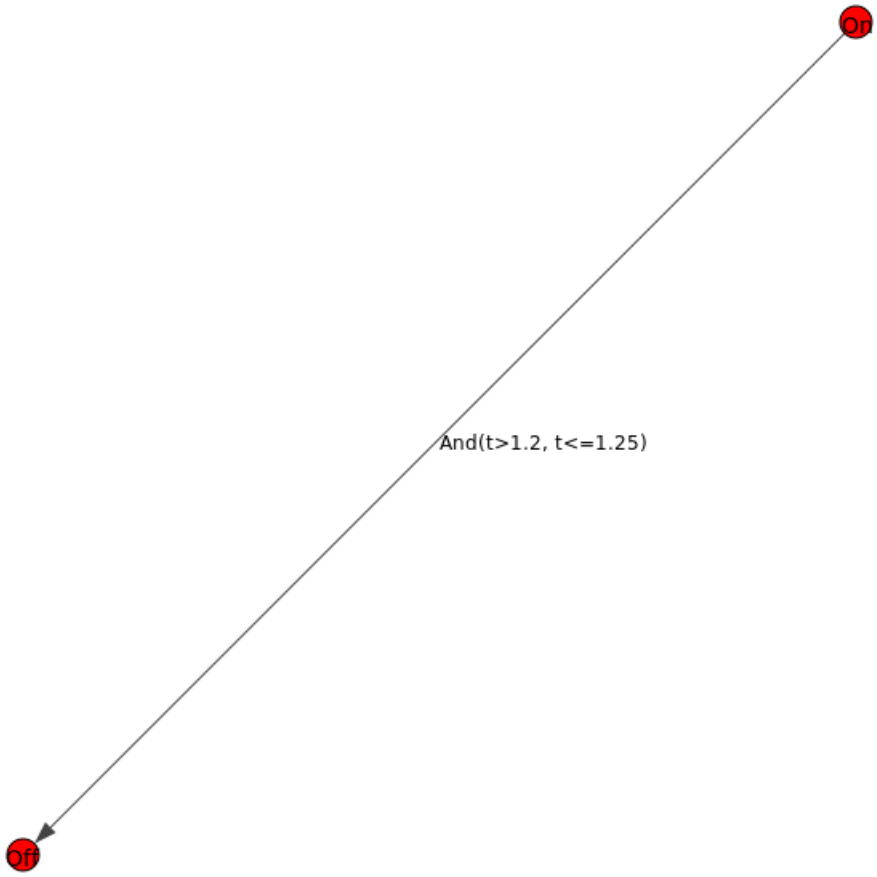


Fig. 5.1: The white box transition graph of the thermostat system

CHAPTER

SIX

EXAMPLES

6.1 Getting started: Simple Automatic Emergency Braking

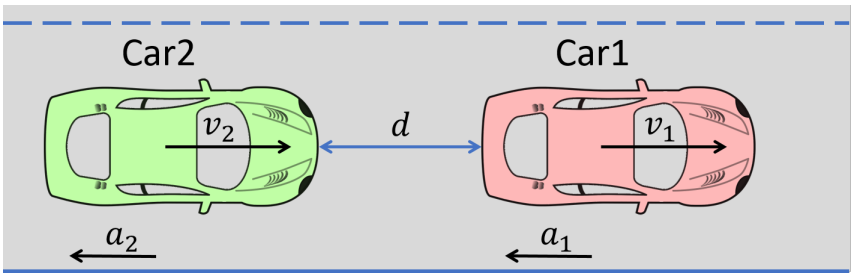


Fig. 6.1: An illustration of Automatic Emergency Braking System

Consider the example an AEB as shown above: Cars 1 and 2 are cruising down the highway with zero relative velocity and certain initial relative separation; Car 1 suddenly switches to a braking mode and starts slowing down according, certain amount of time elapses, before Car 2 switches to a braking mode. We are interested to analyze the severity (relative velocity) of any possible collisions.

6.1.1 Safety Verification of the AEB System

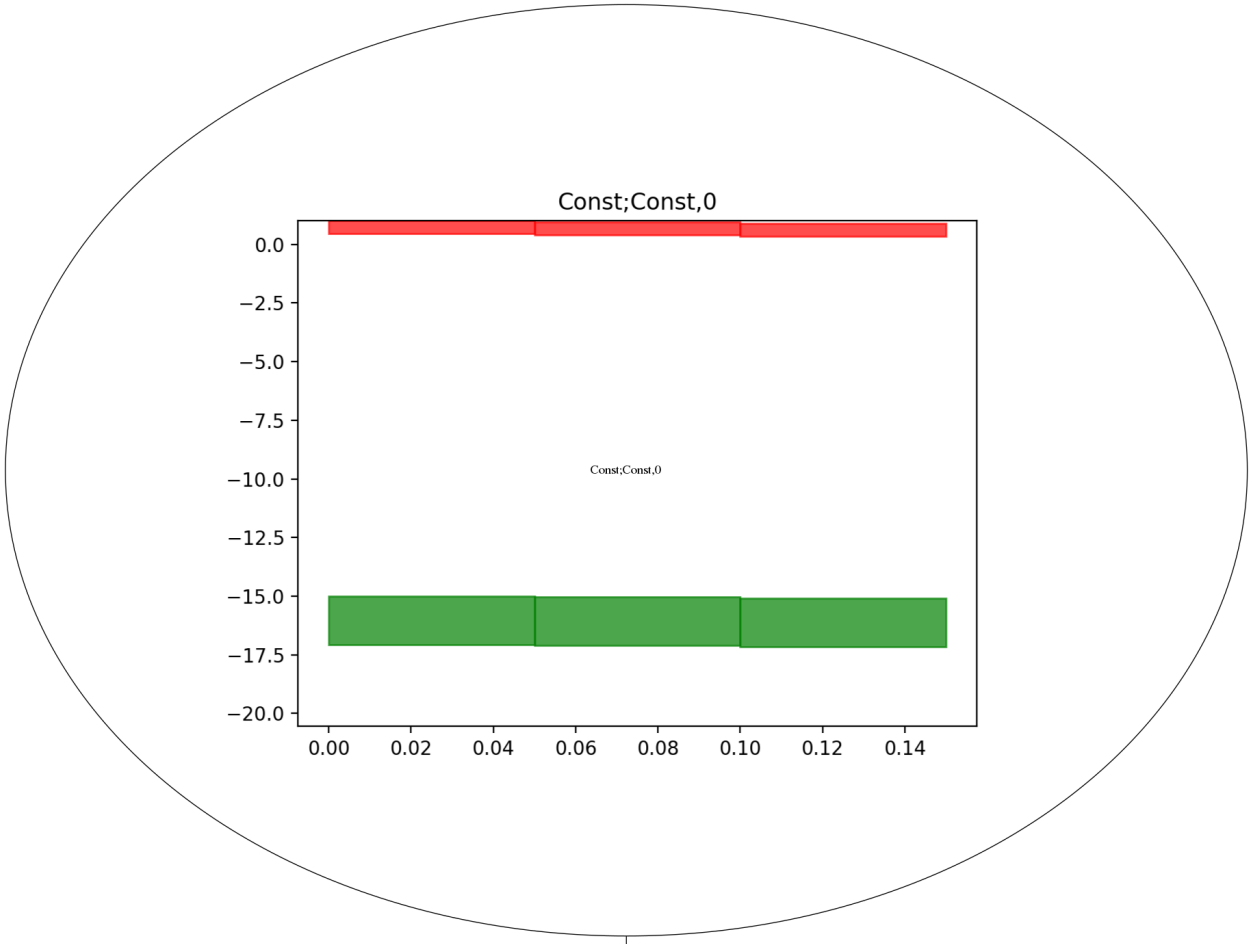
The black-box of the vehicle dynamics is described in ADAS-label, and the transition graph of the above AEB is shown in *Transition Graph*. The unsafe region is that the relative distance between the two cars are too close ($|sy_1 - sy_2| < 3$). The input files describing the hybrid system is shown in *Input Format*.

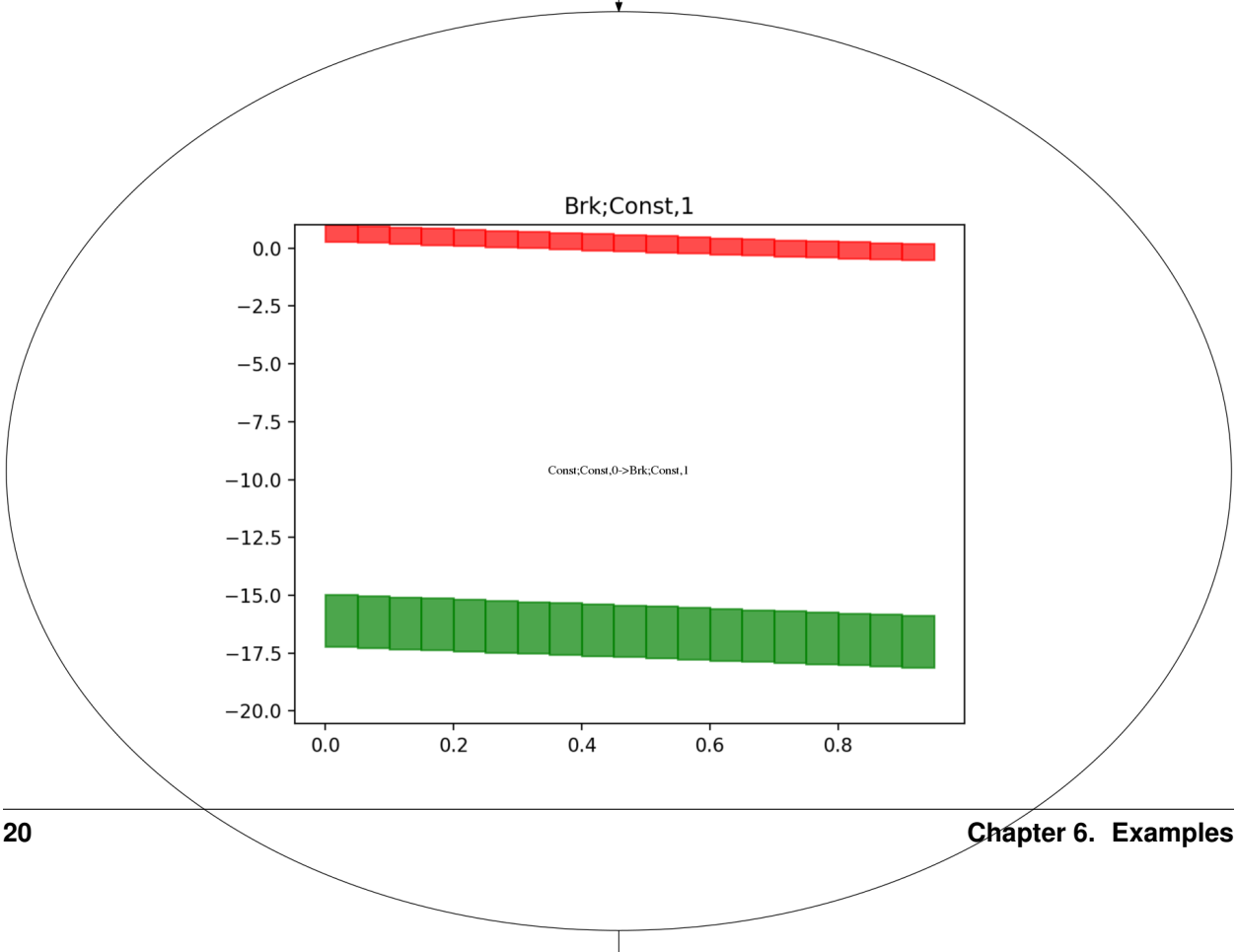
6.1.2 Verification Result of the AEB System

Run DryVR's verification algorithm for the AEB system:

```
python main.py input/daginput/input_brake.json
```

The system is checked to be safe. We can also plot the reachtubes for different variables. For example, the reachtubes for the position of Car1 and Car2 along the road the direction are shown below. From the reachtube we can also clearly see that the relative distance between the two cars are never too small.


Const;Const,0
0.0
−2.5
−5.0
−7.5
−10.0
−12.5
−15.0
−17.5
−20.0
0.00
0.02
0.04
0.06
0.08
0.10
0.12
0.14
Const;Const,0


Brk;Const,1
0.0
−2.5
−5.0
−7.5
−10.0
−12.5
−15.0
−17.5
−20.0
0.0
0.2
0.4
0.6
0.8
Const;Const,0->Brk;Const,1

6.2 Verification Peformance

We have measured performance for examples come with DryVR 2.0. Peformance is measured using computer with i7 6600u, 16gb ram, Ubuntu 16.04 OS.

Model	Dimension	Simulation time	Verfication Time	Total Time	Flow* time
Biological model I	7	0.01s	0.03s	0.04s	66.4s
Biological model II	7	0.01s	0.03s	0.04s	223.4s
Coupled Vanderpol	4	0.03s	0.11s	0.14s	1038.3s
Spring pendulum	4	0.05s	0.11s	0.16s	1377.5s
Roessler	3	0.02s	0.34s	0.36s	17.1s
Lorentz system	3	0.34s	0.73s	1.07s	316.7s
Lac operon	2	0.47s	170.88s	171.35s	44.2s
Lotka-Volterra	2	0.02s	0.08s	0.10s	3.9s
Buckling column	2	0.04s	0.39s	0.43s	26.4s
Jet engine	2	0.07s	12.03s	12.1s	6.8s
Brusselator	2	0.10s	2.92s	3.02s	5.2s
Vanderpol	2	0.05s	2.87s	2.92s	6.4s
Vehicle platoon 3	9	0.32s	3.96s	4.28s	21.08s
Uniform nor sigmoid	3	120.91s	1193.31s	1314.22s	Exception
Uniform inverter loop	2	10.94s	267.62	278.56s	Exception
Uniform inverter sigmoid	2	24.87s	221.94s	246.76s	Exception
Uniform nor ramp	3	173.77s	1591.78s	1765.55s	Exception
Uniform or ramp	4	176.70s	1602.17s	1778.87s	Exception
Uniform or sigmoid	4	168.75s	2017.25s	2186.00s	Exception
Clamped beam	348	540.80s	5176.83s	5717.63s	Time out
Building model	48	3.28s	16.96s	20.24s	Time out
Partial differential equation	20	12.05s	29.16s	41.21s	Time out
FOM	20	12.18s	28.79s	40.9s	Time out
Motor control system	8	5.22s	12.67s	17.89s	Time out
International space station	25	79.99s	193.61s	243.60s	Time out
Lane merge	8	0.29s	563.23s	563.52s	N/A

6.3 Graph Search Performance

Peformance is measured using computer with i7 6600u, 16gb ram, Ubuntu 16.04 OS. Note the running time for graph search can be very different since the alogirthm is randomly search for the graph. It may also return nothing as well. Try to run algorithm multiple times if it does not return the graph.

Example	Dimension	Time horizon	Min staying time	Running Time
vehicle collision avoidance	4	50.0s	2.0s	1896.26s
robot in maze	4	10.0s	1.0s	98.93s
motion plan	3	6.0s	1.0s	4.55s
DC motor	2	1.0s	0.1s	0.35s
room heating	3	25.0s	2.0s	2.66s
inverted pendulum	2	2.0s	0.2s	6.06s

CHAPTER

SEVEN

PUBLICATIONS

- Chuchu Fan, Bolun Qi, Sayan Mitra and Mahesh Viswanathan, DRYVR:Data-driven verification and compositional reasoning for automotive systems, CAV 2017. [Video]
- Chuchu Fan, Bolun Qi and Sayan Mitra, Road to safe autonomy with data and formal reasoning, (To appear in IEEE Design & Test).

CHAPTER

EIGHT

PEOPLE INVOLVED

If you have any problem using the DryVR, contact the authors of the accompanying paper(s)

Chuchu Fan PhD candidate, ECE, Email

Bolun Qi Graduate student, ECE, Email

Sayan Mitra Associate Professor, ECE, Email

Mahesh Viswanathan Professor, CS, Email