# DryVR Documentation

*Release 0.1*

**Chuchu Fan, Bolun Qi**

**Sep 27, 2021**

# Contents

**Release** 0.1

**Date** 04/18/2017

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 1

# Status

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 2

# 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

CHAPTER 3

# Usage

## 3.1 Run DryVR

To run DryVR, please run:

```
python main.py inputFile/[input_file]
```

for example:

```
python main.py inputFile/input_AEB
```

## 3.2 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 in the DryVR root directory:

```
python tubePlotter.py [dimension Number]
```

Where [dimension number] is the dimension you want to plot reachtube. Note that the dimension 0 is the local time for each mode and last dimension is the global time.

For example, input_AEB's has 8 dimentions: $sx_1, sy_1, vx_1, vy_1, sx_2, sy_2, vx_2, vy_2$ (refer to *The Autonomous Vehicle Benchmark* for more details). You can choose to plot dimension from 0 to 9. Dimension 0 is the local time for each mode and dimension 9 is the global time. Dimension 1~8 corresponds to the state variables as above.

For example, to plot the reachtube for $sx_1$, please run

```
python tubePlotter.py 1
```

**Note** please do not use the tubePlotter function when system is checked to be Unsafe or Unknown (when no more refinement can be used).

When system is checked to be **Unsafe**, the counter-example simulation trace is stored in the output folder as **unsafeSim**.

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

# CHAPTER 4

# 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)
```

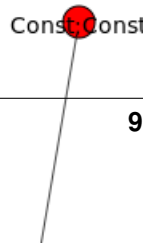
Given the mode name "Mode", initial state "initialCondition" and time horizon "time_bound", the function TC_Simulate should return an python array of the form:

```
[[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]
transtime:[transition graph edge labels (transition times)]
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
```

Example input for the Automatic Emergency Braking System

```
vertex:["Const;Const","Brk;Const","Brk;Brk"]
edge:[(0,1),(1,2)]
transtime:[(0,0.1),(0.8,0.9)]
initialSet:[[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]]
```

(continues on next page)

(continued from previous page)

```
unsafeSet:@Allmode:And(v2-v6<3,v6-v2<3)
timeHorizon:5
directory:ExamplesPython/
```

## 4.4 Output Interpretation

The tool will print background information like the current mode, transition time, initial set and discrepancy function information on the run. The final result about safe/unsafe will be printed at the bottom.

When the system is safe, the final result will look like

```
System is Safe!
System has been refined for * Times
Simulation safety check is * (seconds)
Verification safety check is * (seconds)
```

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

```
Simulation safety check is * (seconds)
System Unsafe from simulation, halt verification
```

The unsafe simulation trajectory will be stored as "unsafeSim" in the output folder.

## 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.

```
mkdir Thermostats
cd Thermostats
```

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

```
vim 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.

```
vim __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)]
transtime:[(1,1.1),(1,1.1)]
initialSet:[[75.0],[76.0]]
unsafeSet:@Allmode:And(v1>90)
timeHorizon:3.5
directory:Thermostats/
```

Save the input file in the folder inputFile and name it as input_thermo.

**Step6**: Run the verification algorithm using the command:

```
python main.py inputFile/input_thermo
```

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

```
System is Safe!
System has been refined for 0 Times
Simulation safety check is 0.150208
Verification safety check is 0.116688
```

We can plot the reachtube using the command:

```
python tubePlotter.py 1
```

And the reachtube for the temperature is shown as

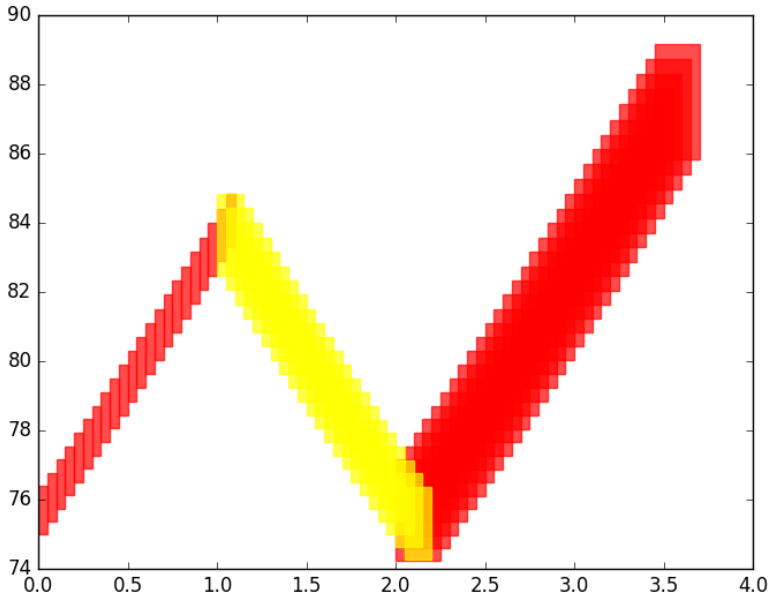



Fig. 2: The reachtube for the temperature of the thermostat system example

CHAPTER 5

# Examples

## 5.1 Getting started: Simple Automatic Emergency Braking

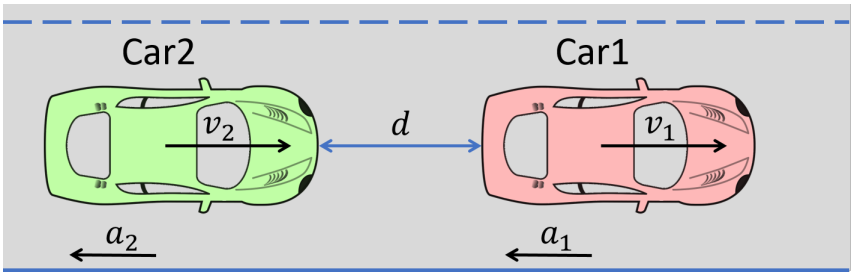



Fig. 1: An illustration of Automatic Emergency Braking System

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.

### 5.1.1 Safety Verification of the AEB System

The black-box of the vehicle dynamics is described in *The Autonomous Vehicle Benchmark*, 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*.

### 5.1.2 Verification Result of the AEB System

Run DryVR's verification algorithm for the AEB system:

```
python main.py inputFile/input_AEB
```

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.

Fig. 2: Reachtube of the position sy of Car1

## 5.2 The Autonomous Vehicle Benchmark

The hybrid system for a scenario is constructed by putting together several individual vehicles. The higher-level decisions (paths) followed by the vehicles are captured by the transition graphs discussed in *Transition Graph*.

Each vehicle has the following modes

- Const: move forward at constant speed,
- Acc1: constant acceleration,
- Brk or Dec: constant (slow) deceleration,
- TurnLeft and TurnRight: the acceleration and steering are controlled in such a manner that the vehicle switches to its left (resp. right) lane in a certain amount of time.

The mode for the entire system consists of n vehicles are the mode of each vehicle separated by semicolon. For example, Const;Brk means the first car is in the const speed mode, while the second car is in the brake mode. For each vehicle, we mainly analyze four variables: absolute position ($sx$) and velocity ($vx$) orthogonal to the road direction ($x$-axis), and absolute position ($sy$) and velocity ($vy$) along the road direction ($y$-axis). The throttle and steering is captured using the four variables.

Due to the MATLAB license issue, we are not able to release the Simulink benchmarks we have used in the publications. We have since reproduced the ADAS and autonomous vehicle benchmark in Python and connect it with DryVR as a simulator. We are hoping to move more examples to Python in the near future.

For more details, please refer to Section 2.5 of the CAV2017 paper.



Fig. 3: Reachtube of the position sy of Car2

## 5.3 Other examples

Next, we briefly introduce other examples included in the inputFile folder and their verification results. Note that as the algorithm uses nondeterministic method to generate traces, the verification result like refine times, running time may vary between different runs.

**AutoPassing**

Initial condition: Car1 is behind Car2 in the same lane, with Car1 in Acc1 and Car2 in Const.

Transition graph: Car1 goes through the mode sequence TurnLeft, Acc1, Brk, and TurnRight, Const with specified time intervals in each mode to complete the overtake maneuver. If Car2 switches to Acc1 before Car1 enters Acc1 then Car1 aborts and changes back to right lane. If Car2 switches to Dec before Car1 enters TurnLeft, then Car1 should adjust the time to switch to TurnLeft to avoid collision.

Requirement: Car1 overtakes Car2 or abort the overtaking while maintaining minimal safe separation.

Inputfiles:

- input_AutoPassingSafe: safe
- input_AutoPassingUnsafe: unsafe
- input_AutoPassingSimpleSafe: safe

- input_AutoPassingSimpleUnsafe: unsafe

**Merge**

Initial condition: Car1 is in left and Car2 is in the right lane; initial positions and speeds are in some range; Car1 is in Const mode, and Car2 is in Const mode.

Transition graph: Car1 goes through the mode Acc1, TurnRight, Const with specified intervals of time to transit from mode to another mode. Car2 goes through the mode Acc1 or Const, TurnRight, Const with specified intervals of time to transit from mode to another mode. Car1 will merge ahead of Car2 of behind of Car2 based on cars's mode transition.

Requirement: Car1 merges ahead or behind of Car2 and maintains at least a given safe separation.

InputFiles:

- input_MergeSafe: safe
- input_MergeUnsafe: unsafe

**MergeBetween**

Initial condition: Car1, Car2, Car3 are all in the same lane, with Car1 behind Car2, Car2 behind Car3, and in the Const mode, initial positions and speeds are in some range.

Transition graph: Car1 goes through the mode sequence TurnLeft, Acc1, Dec, and TurnRight, Const with specified time intervals in each mode to overtake Car2. Car3 transits from Const to Acc1 then transits back to Const, so Car3 is always ahead of Car1.

Requirement: Car1 merges between Car2 and Car3 and any two vehicles maintain at least a given safe separation.

InputFiles:

- input_MergeBetweenSafe: safe
- input_MergeBetweenUnsafe: unsafe

# CHAPTER 6

# Publications

- Chuchu Fan, Bolun Qi, Sayan Mitra and Mahesh Viswanathan, *DRYVR:Data-driven verification and compositional reasoning for automotive systems <https://arxiv.org/abs/1702.06902>*.
- 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 7

# 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