Falsification of Synchronous and Hybrid Systems using Automatic Differentiation

SYNCHRON'19

Ismail Bennani

PhD Student, PARKAS, INRIA

supervised by

Marc Pouzet	Goran Frehse	Timothy Bourke
PARKAS, INRIA	U2IS, ENSTA Paris	PARKAS, INRIA

November 28, 2019

Introduction

The bouncing ball

Discrete

```
let node ball y0 = y where
  rec y = y0 → pre (y +. dt *. y_v)

  and y_v = 0. →
    if bounce then
        (-. 0.8 *. pre y_v)
    else
        pre (y_v -. dt *. g)

  and bounce = y < 0.
```

Hybrid

```
let hybrid ball y0 = y where
  rec der y = y_v init y0

  and der y_v = -. g init 0.0
      reset bounce →
        (-. 0.8 *. last y_v)

  and bounce = up(-. y)
```

In this talk: a (very) quick overview of FADBADml[1], a library for Automatic Differentiation in OCaml implemented by François Bidet and myself.

[1] https://fadbadml-dev.github.io/FADBADml/

Summary

- Falsification problem
  - Example
  - State of the art
- Automatic Differentiation : FADBADml
- Falsification with FADBADml
  - A word about specification
  - Example: a simple ODE
  - Synchronous system: the automatic transmission
  - WIP: the heater

Summary

Falsification problem
- Example
- State of the art

Automatic Differentiation : FADBADml

Falsification with FADBADml
- A word about specification
- Example: a simple ODE
- Synchronous system: the automatic transmission
- WIP: the heater

The Falsification problem

Falsification problem Given a System Under Test *SUT* and a specification *spec*, find an input I such that the output O doesn't satisfy $spec(I, O)$

Example: SUT = model of an F-16 aircraft with a Ground Collision Avoidance System (GCAS)

$\hookrightarrow$ **inputs**: initial position and rotation

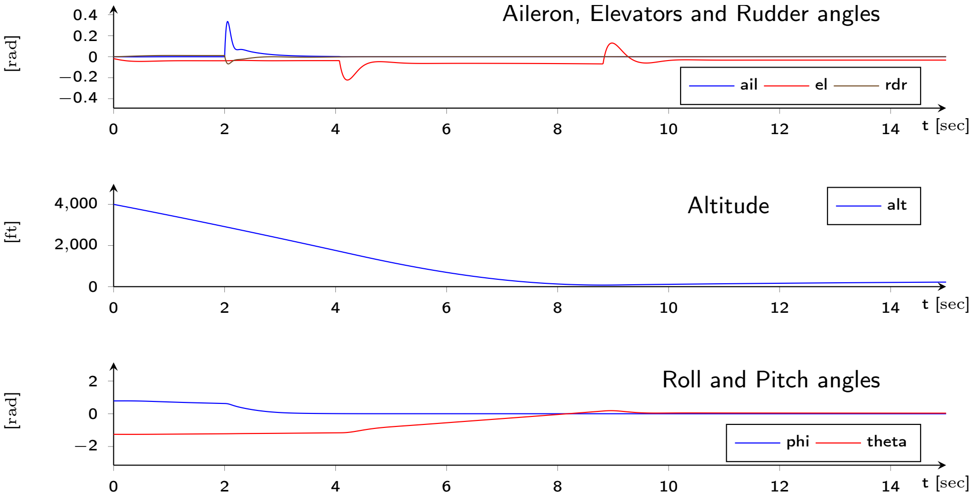
$\hookrightarrow$ **outputs** at time t: current position, rotation, velocity, angular velocity, ...

$spec$ = the aircraft doesn't crash

Summary

- Falsification problem
  - Example
  - State of the art
- Automatic Differentiation : FADBADml
- Falsification with FADBADml
  - A word about specification
  - Example: a simple ODE
  - Synchronous system: the automatic transmission
  - WIP: the heater

Falsification problem: F-16 aircraft



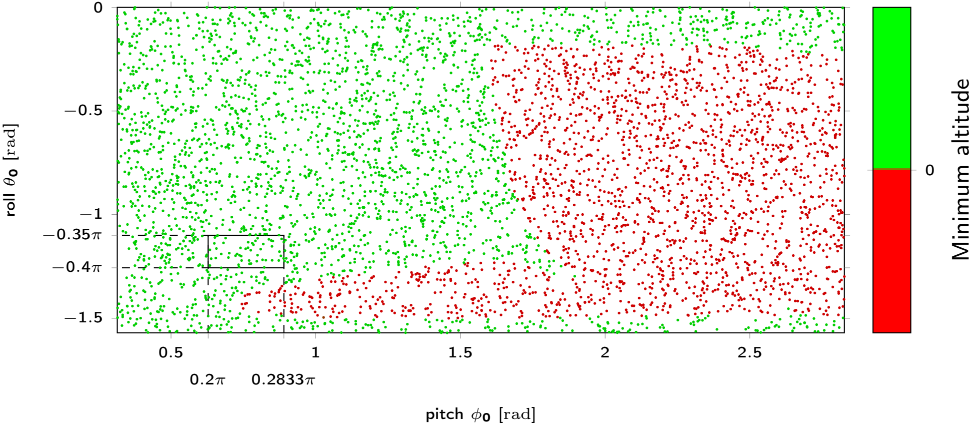
Falsification problem: Uniform Random sampling

Specification : The F16 does not crash

Plot space $\phi_0 \in [0.1\pi, 0.9\pi]$, $\theta_0 \in [-0.5\pi, 0]$

Search space $\phi_0 \in [0.2\pi, 0.2833\pi]$, $\theta_0 \in [-0.4\pi, -0.35\pi]$

Number of points : 5000



Summary

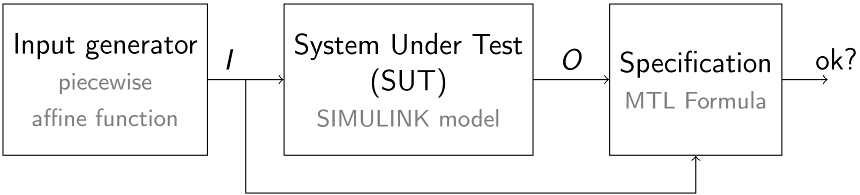
Falsification problem
- Example
- State of the art

Automatic Differentiation : FADBADml

Falsification with FADBADml
- A word about specification
- Example: a simple ODE
- Synchronous system: the automatic transmission
- WIP: the heater

Falsification problem : State of the art

Breach Donzé (2010), S-TaLiro Annpureddy et al. (2011)



Main Idea

instead of asking "Is spec satisfied by SUT ?",
ask "How much is spec satisfied by SUT ?"

$\rightarrow$ compute a robustness (float) instead of a boolean.

The **falsification** problem becomes a **minimization** problem :

$$spec(I, O) \text{ is not satisfied} \Leftrightarrow spec(I, O) < 0$$

Falsification problem : State of the art

other tools: FalStar, falsify

Method	**FalStar**: Two layered falsification with tree search (MCTS Zhang et al. (2018), aLVTS Ernst et al. (2018))
	falsify: Deep Reinforcement Learning Akazaki et al. (2018)

Both methods are **online**: at each step, the tools produce a new input based on the last outputs

Summary

Falsification problem
- Example
- State of the art

Automatic Differentiation : FADBADml

Falsification with FADBADml
- A word about specification
- Example: a simple ODE
- Synchronous system: the automatic transmission
- WIP: the heater

Automatic Differentiation : FADBADml

Porting of FADBAD++ in OCaml by François Bidet and I

FADBAD++ Stauning (1997):

- C++ library for Automatic Differentiation
- written by Ole Stauning as part of his PhD thesis.

FADBADml is available at github and can be installed with opam

Automatic Differentiation : FADBADml

Porting of FADBAD++ in OCaml by François Bidet and I

FADBAD++ Stauning (1997):

- C++ library for Automatic Differentiation
- written by Ole Stauning as part of his PhD thesis.

FADBADml is available at github and can be installed with opam

Example: let `z = x + y`, FADBADml defines a type `F<float>` and overloads the operator + to compute $\frac{dz}{dx}$ and $\frac{dz}{dy}$ at runtime (knowing that $\frac{dx}{dx} = 1$ and $\frac{dy}{dy} = 1$)

Note: FADBAD++ (and FADBADml) implements forward and backward automatic differentiation as well as a library to compute Taylor coefficients of an equation with respect to one variable. However I will only present FAD, the library for forward differentiation.

Automatic Differentiation : FADBADml

How does it work ?

```
(* a,b floats *)
...
let x = make a in
let y = make b in
```

$$x \begin{cases} v = \mathtt{a} \\ \mathit{diff} = [] \end{cases} \qquad y \begin{cases} v = \mathtt{b} \\ \mathit{diff} = [] \end{cases}$$

Automatic Differentiation : FADBADml

How does it work ?

```
(* a,b floats *)
...
let x = make a in
let y = make b in
```

$$x \begin{cases} v = \texttt{a} \\ \mathit{diff} = \texttt{[]} \end{cases} \qquad y \begin{cases} v = \texttt{b} \\ \mathit{diff} = \texttt{[]} \end{cases}$$

```
diff x 0 2;
diff y 1 2;
```

$$x \begin{cases} v = \texttt{a} \\ \mathit{diff} = \texttt{[1,0]} \end{cases} \qquad y \begin{cases} v = \texttt{b} \\ \mathit{diff} = \texttt{[0,1]} \end{cases}$$

Automatic Differentiation : FADBADml

How does it work ?

```
(* a,b floats *)
...
let x = make a in
let y = make b in
```

$$x \begin{cases} v = \texttt{a} \\ diff = \texttt{[]} \end{cases} \qquad y \begin{cases} v = \texttt{b} \\ diff = \texttt{[]} \end{cases}$$

```
diff x 0 2;
diff y 1 2;
```

$$x \begin{cases} v = \texttt{a} \\ diff = \texttt{[1,0]} \end{cases} \qquad y \begin{cases} v = \texttt{b} \\ diff = \texttt{[0,1]} \end{cases}$$

```
let z = op(x,y) in
...
(* (d z 0) contains dz/dx
and (d z 1) contains dz/dy *)
```

$$z \begin{cases} v = op_{float}(v_x, v_y) \\ diff = op_{diff}(v_x, v_y, diff_x, diff_y) \end{cases}$$

Example if $op(x, y) = x *_{fad} y$ then
$op_{float}(x, y) = x *_{float} y$ and
$op_{diff}(v_x, v_y, diff_x, diff_y) = v_y * diff_x +_{vec} v_x * diff_y$

Summary

Falsification problem
- Example
- State of the art

Automatic Differentiation : FADBADml

Falsification with FADBADml
- A word about specification
- Example: a simple ODE
- Synchronous system: the automatic transmission
- WIP: the heater

Summary

Falsification problem
- Example
- State of the art

Automatic Differentiation : FADBADml

Falsification with FADBADml
- A word about specification
- Example: a simple ODE
- Synchronous system: the automatic transmission
- WIP: the heater

A word about specification

A quantitative semantic of basic constructions

Most simple formulas: usual boolean operators

$$\rho(x > f) = x - f$$

$$\rho(x < f) = f - x$$

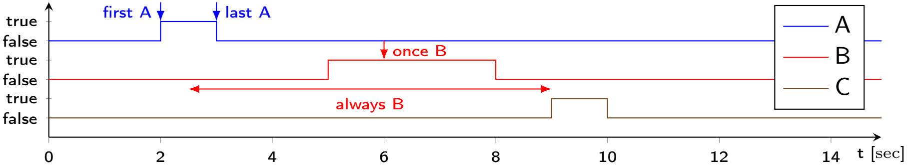
$$\rho(x \vee y) = max(x, y)$$

$$\rho(x \wedge y) = min(x, y)$$

- **Sign**: encodes the boolean value
- **Absolute value**: encodes some kind of score

A word about specification

Temporal constructions



after	first last	A holds,	once always	B holds until C holds

↪ these 4 constructions are enough to express all the properties that I found in several benchmarks Ernst et al. (2019) Dokhanchi et al. (2018) Hoxha et al. (2014)

These macros are synchronous observers Halbwachs et al. (1994)
They express a subset of MITL (Metric Interval Temporal Logic, the logic used by S-TaLiro, Breach, ...)

My contribution on this: quantitative semantics + continuous version

Summary

Falsification problem
- Example
- State of the art

Automatic Differentiation : FADBADml

Falsification with FADBADml
- A word about specification
- Example: a simple ODE
- Synchronous system: the automatic transmission
- WIP: the heater

Falsification with FADBADml

Example: a simple ODE

$$\begin{cases} \dot{x}(t) = x(t) + y(t) + 0.1t \\ \dot{y}(t) = y(t) * cos(2\pi y(t)) + x(t) * sin(2\pi x(t)) + 0.1t \\ x(0) = x_0 \in [-1, 1] \\ y(0) = y_0 \in [-1, 1] \end{cases}$$

Specification: x is always in $[-1.6, -1.4]$

Falsification with FADBADml

Example: a simple ODE

$$
\begin{cases}
\dot{x}(t) = x(t) + y(t) + 0.1t \\
\dot{y}(t) = y(t) * cos(2\pi y(t)) + x(t) * sin(2\pi x(t)) + 0.1t \\
x(0) = x_0 \in [-1, 1] \\
y(0) = y_0 \in [-1, 1]
\end{cases}
$$

Specification: x is always in $[-1.6, -1.4]$

Falsification problem: find $x_0 \in [-1, 1]$ and $y_0 \in [-1, 1]$ such that $\exists t \in [0, t_{max}] \; / \; x(t) \notin [-1.6, -1.4]$

Falsification with FADBADml

Example: a simple ODE

$$\begin{cases} \dot{x}(t) = x(t) + y(t) + 0.1t \\ \dot{y}(t) = y(t) * cos(2\pi y(t)) + x(t) * sin(2\pi x(t)) + 0.1t \\ x(0) = x_0 \in [-1, 1] \\ y(0) = y_0 \in [-1, 1] \end{cases}$$

Specification: x is always in $[-1.6, -1.4]$

Falsification problem: find $x_0 \in [-1, 1]$ and $y_0 \in [-1, 1]$ such that $\exists t \in [0, t_{max}] \ / \ x(t) \notin [-1.6, -1.4]$

(* $(x \geq -1.6) \Leftrightarrow (-1.6 - x \geq 0)$ | $(x \leq -1.4) \Leftrightarrow (x + 1.4 \geq 0)$ *)

Robustness: $\rho(x) = min(-1.6 - x, x + 1.4)$

Falsification with FADBADml

Example: a simple ODE solved with Euler's integration scheme (fixed step)

System

$$\begin{cases} \dot{x}(t) = x(t) + y(t) + 0.1t \\ \dot{y}(t) = y(t) * cos(2\pi y(t)) + x(t) * sin(2\pi x(t)) + 0.1t \\ x(0) = x_0 \\ y(0) = y_0 \end{cases}$$

Execution (`dt` is a fixed parameter)

```
rec t = 0 → (pre t) + dt
and x = (pre x) + dt * ((pre x) + (pre y) + 0.1 * t)
and y = (pre y) + dt * ((pre y) * cos(2*pi*(pre y)) +
                        (pre x) * sin(2*pi*(pre x)) + 0.1 * t)
```

With FADBADml, we can compute dx/dx_0 and dx/dy_0 after any number of steps

Falsification with FADBADml

Example: a simple ODE with Euler and Gradient Descent Reddi et al. (2019)

Specification

$$x \in [-1.6, -1.4] \wedge$$
$$y \in [-1.1, -0.9]$$

Parameters

$$\text{input space} = [-1, 1] \; x \; [-1, 1]$$
$$\texttt{dt} = 0.1$$
$$\texttt{n\_steps} = 100$$

Falsification with FADBADml

Example: a simple ODE with Euler and Gradient Descent Reddi et al. (2019)

Specification

$$x \in [-1.6, -1.4] \wedge$$
$$y \in [-1.1, -0.9]$$

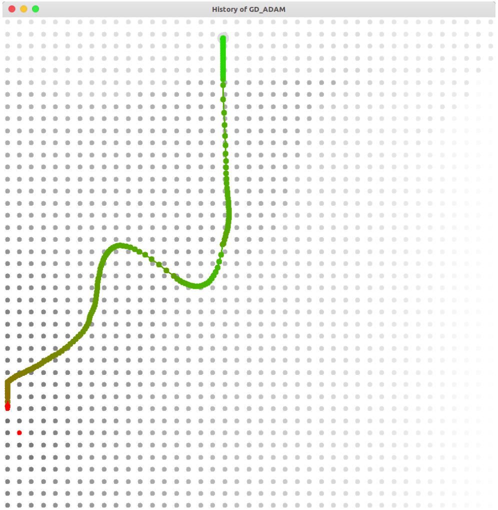
Parameters

$$\text{input space} = [-1, 1] \ x \ [-1, 1]$$
$$\texttt{dt} = 0.1$$
$$\texttt{n\_steps} = 100$$

Falsified after 172 tries:

- best rob = -0.0115205
- sample =

$$\begin{cases} x_0 = -1. \\ y_0 = -0.589587520984 \end{cases}$$



Summary

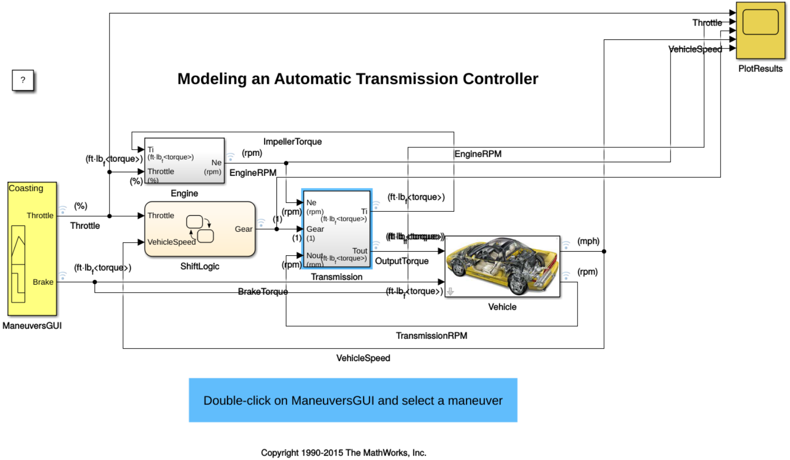
Falsification problem
- Example
- State of the art

Automatic Differentiation : FADBADml

Falsification with FADBADml
- A word about specification
- Example: a simple ODE
- Synchronous system: the automatic transmission
- WIP: the heater

Falsification with FADBADml

Synchronous system: automatic transmission (ARCH Comp. 2014) Hoxha et al. (2014)Hoxha et al. (2015)

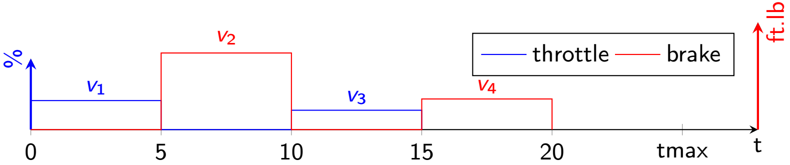


Input: `throttle`, `brake` **Output**: `gear`, `speed`, `rpm`

Falsification with FADBADml

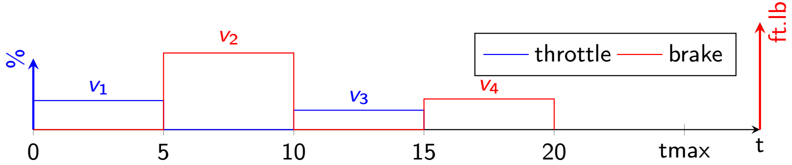
Synchronous system: the automatic transmission, offline falsification

Input shape (dim. 4)



Falsification with FADBADml

Synchronous system: the automatic transmission, offline falsification

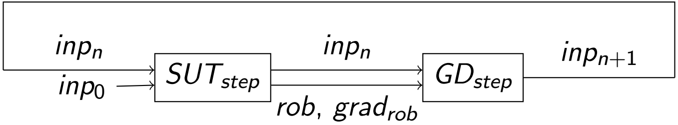
Input shape (dim. 4)	
Specification	The engine and the vehicle speed never reach $\bar{w}$ and $\bar{v}$ resp.
Robustness	$\rho(w, v) = min(\bar{w} - w, \bar{v} - v)$

This is the same as before: instead of picking 2 values at the beginning, we pick 4 of them and construct the input with them

Falsification with FADBADml

Synchronous system: the automatic transmission, online falsification

Online falsification :



DEMO

Summary

- Falsification problem
  - Example
  - State of the art
- Automatic Differentiation : FADBADml
- Falsification with FADBADml
  - A word about specification
  - Example: a simple ODE
  - Synchronous system: the automatic transmission
  - WIP: the heater

Falsification with FADBADml

WIP: the heater (Nicolas Halbwachs, Collège de France, 2010)

```
let node euler(h)(x0, xprime) = x
  where rec x = x0 → pre (x +. h *. xprime)

let node heater(c, α, β, temp_ext, temp0, u) = temp where
  rec der_temp =
    if u then (c - temp) *. α
         else (temp_ext - temp) *. β
  and temp = euler(0.2)(temp0, der_temp)

 let node relay(low, high, v) = u where
   rec u = if v < low then true
          else if v > high then false
          else false → pre u

let node system(reference) = (u, temp) where
  rec u = relay(reference -. 1., reference +. 1., temp)
  and temp = heater(50.0, 0.1, 0.05, 0.0, 15.0, u)
```

Falsification with FADBADml

WIP: the heater

$$\dot{temp} = \begin{cases} \alpha * (c - temp) & \text{if } u \\ \beta * (temp_{ext} - temp) & \text{if } \neg u \end{cases}$$

integrated using euler

$$temp_{n+1} = temp_n + \delta_n * \begin{cases} \alpha * (c - temp_n) & \text{if } u_n \\ \beta * (temp_{ext} - temp_n) & \text{if } \neg u_n \end{cases}$$

$$u_{n+1} = \begin{cases} true & \text{if } temp_n < ref - low \\ false & \text{if } temp_n > ref - high \\ u_n & \text{else} \end{cases}$$

Falsification with FADBADml

WIP: the heater

$$temp_{n+1} = temp_n + \delta_n * \begin{cases} \alpha * (c - temp_n) & \text{if } u_n \\ \beta * (temp_{ext} - temp_n) & \text{if } \neg u_n \end{cases}$$

and $temp_0$ is a constant

$$u_{n+1} = \begin{cases} true & \text{if } temp_n < ref - low \\ false & \text{if } temp_n > ref - high \\ u_n & \text{else} \end{cases}$$

and $u_0 = false$

Falsification with FADBADml

WIP: the heater

$$temp_{n+1} = temp_n + \delta_n * \begin{cases} \alpha * (c - temp_n) & \text{if } u_n \\ \beta * (temp_{ext} - temp_n) & \text{if } \neg u_n \end{cases}$$

and $temp_0$ is a constant

$$u_{n+1} = \begin{cases} true & \text{if } temp_n < ref - low \\ false & \text{if } temp_n > ref - high \\ u_n & \text{else} \end{cases}$$

and $u_0 = false$

We take the partial derivative of *temp* w.r.t. *ref*

$$\frac{d}{dref} temp_{n+1} = \frac{d}{dref} temp_n + \delta_n * \begin{cases} -\alpha * \frac{d}{dref} temp_n & \text{if } u_n \\ -\beta * \frac{d}{dref} temp_n & \text{if } \neg u_n \end{cases}$$

Falsification with FADBADml

WIP: the heater

$$temp_{n+1} = temp_n + \delta_n * \begin{cases} \alpha * (c - temp_n) & \text{if } u_n \\ \beta * (temp_{ext} - temp_n) & \text{if } \neg u_n \end{cases}$$

and $temp_0$ is a constant

$$u_{n+1} = \begin{cases} true & \text{if } temp_n < ref - low \\ false & \text{if } temp_n > ref - high \\ u_n & \text{else} \end{cases}$$

and $u_0 = false$

We take the partial derivative of $temp$ w.r.t. ref

$$\frac{d}{dref} temp_{n+1} = \frac{d}{dref} temp_n + \delta_n * \begin{cases} -\alpha * \frac{d}{dref} temp_n & \text{if } u_n \\ -\beta * \frac{d}{dref} temp_n & \text{if } \neg u_n \end{cases}$$

Let's apply with $n = 0$

$$\frac{d}{dref} temp_1 = 0 + \delta_0 * \begin{cases} -\alpha * 0 & \text{if } u_0 \\ -\beta * 0 & \text{if } \neg u_0 \end{cases}$$

Falsification with FADBADml

WIP: the heater

What happens if we trigger mode transitions ?

First: quantitative semantics over boolean formulas :

$$u_{n+1} = \begin{cases} (ref - low) - temp_n & \text{if } temp_n < ref - low \\ temp_n - (ref - high) & \text{if } temp_n > ref - high \\ u_n & \text{else} \end{cases}$$

Falsification with FADBADml

WIP: the heater

What happens if we trigger mode transitions ?

First: quantitative semantics over boolean formulas :

$$u_{n+1} = \begin{cases} (ref - low) - temp_n & \text{if } temp_n < ref - low \\ temp_n - (ref - high) & \text{if } temp_n > ref - high \\ u_n & \text{else} \end{cases}$$

Then: derivatives w.r.t. the input

$$\frac{d}{dref} u_{n+1} = \begin{cases} 1 - \frac{d}{dref} temp_n & \text{if } temp_n < ref - low \\ \frac{d}{dref} temp_n - 1 & \text{if } temp_n > ref - high \\ \frac{d}{dref} u_n & \text{else} \end{cases} = \begin{cases} 1 & \text{if } temp_n < ref - low \\ -1 & \text{if } temp_n > ref - high \\ \frac{d}{dref} u_n & \text{else} \end{cases}$$

Falsification with FADBADml

WIP: the heater

```
let node relay(low, high, v) = u where
  automaton
  | HIGH →
    do u = false
    until (v < low) then LOW             (* zc: "trigger_low", rob: low - v *)
    else (v < high) then MID(false)      (* zc: "trigger_mid", rob: high - v *)
  | MID(out) →
    do u = out
    until (v > high) then HIGH           (* zc: "trigger_high", rob: v - high *)
    else (v < low) then LOW              (* zc: "trigger_low", rob: low - v *)
  | LOW →
    do u = true
    until (v > high) then HIGH           (* zc: "trigger_high", rob: v - high *)
    else (v > low) then MID(true)        (* zc: "trigger_mid", rob: v - low *)
```

Falsification with FADBADml

WIP: the heater

Specification: once the temperature is in $[ref - low, ref + high]$, it always stays in $[ref - 2 * low, ref + 2 * high]$

DEMO

References I

Alexandre Donzé. Breach, a toolbox for verification and parameter synthesis of hybrid systems. In Tayssir Touili, Byron Cook, and Paul Jackson, editors, *Computer Aided Verification*, pages 167–170, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg. ISBN 978-3-642-14295-6.

Yashwanth Annpureddy, Che Liu, Georgios Fainekos, and Sriram Sankaranarayanan. S-taliro: A tool for temporal logic falsification for hybrid systems. In Parosh Aziz Abdulla and K. Rustan M. Leino, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, pages 254–257, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg. ISBN 978-3-642-19835-9.

Zhenya Zhang, Ichiro Hasuo, Gidon Ernst, and Sean Sedwards. Two-layered falsification of hybrid systems guided by monte carlo tree search. *CoRR*, abs/1803.06276, 2018. URL http://arxiv.org/abs/1803.06276.

References II

Gidon Ernst, Sean Sedwards, Zhenya Zhang, and Ichiro Hasuo. Fast falsification of hybrid systems using probabilistically adaptive input. *CoRR*, abs/1812.04159, 2018. URL http://arxiv.org/abs/1812.04159.

Takumi Akazaki, Shuang Liu, Yoriyuki Yamagata, Yihai Duan, and Jianye Hao. Falsification of cyber-physical systems using deep reinforcement learning. *CoRR*, abs/1805.00200, 2018. URL http://arxiv.org/abs/1805.00200.

Ole Stauning. *Automatic validation of numerical solutions*. PhD thesis, 11 1997.

Gidon Ernst, Paolo Arcaini, Alexandre Donz\'e, Georgios Fainekos, Logan Mathesen, Giulia Pedrielli, Shakiba Yaghoubi, Yoriyuki Yamagata, and Zhenya Zhang. Arch-comp 2019 category report: Falsification. In Goran Frehse and Matthias Althoff, editors, *ARCH19. 6th International Workshop on Applied Verification of Continuous and Hybrid Systems*, volume 61 of *EPiC Series in Computing*, pages 129–140. EasyChair, 2019. doi: 10.29007/68dk. URL https://easychair.org/publications/paper/5VWq.

References III

Adel Dokhanchi, Shakiba Yaghoubi, Bardh Hoxha, Georgios Fainekos, Gidon Ernst, Zhenya Zhang, Paolo Arcaini, Ichiro Hasuo, and Sean Sedwards. Arch-comp18 category report: Results on the falsification benchmarks. In Goran Frehse, editor, *ARCH18. 5th International Workshop on Applied Verification of Continuous and Hybrid Systems*, volume 54 of *EPiC Series in Computing*, pages 104–109. EasyChair, 2018. doi: 10.29007/t85q. URL https://easychair.org/publications/paper/HjJ8.

Bardh Hoxha, Houssam Abbas, and Georgios E. Fainekos. Benchmarks for temporal logic requirements for automotive systems. In *ARCH@CPSWeek*, 2014.

References IV

Nicolas Halbwachs, Fabienne Lagnier, and Pascal Raymond. Synchronous observers and the verification of reactive systems. In *Proceedings of the Third International Conference on Methodology and Software Technology: Algebraic Methodology and Software Technology*, AMAST '93, pages 83–96, Berlin, Heidelberg, 1994. Springer-Verlag. ISBN 3-540-19852-0. URL http://dl.acm.org/citation.cfm?id=646055.677894.

Sashank J. Reddi, Satyen Kale, and Sanjiv Kumar. On the convergence of adam and beyond. *CoRR*, abs/1904.09237, 2019. URL http://arxiv.org/abs/1904.09237.

Bardh Hoxha, Houssam Abbas, and Georgios Fainekos. Benchmarks for temporal logic requirements for automotive systems. In Goran Frehse and Matthias Althoff, editors, *ARCH14-15. 1st and 2nd International Workshop on Applied veRification for Continuous and Hybrid Systems*, volume 34 of *EPiC Series in Computing*, pages 25–30. EasyChair, 2015. doi: 10.29007/xwrs. URL https://easychair.org/publications/paper/4bfq.