CPSGrader: A Tutorial

Version 0.1

University of California, Berkeley

November 24, 2014

Outline

“Grading” a Cyber-Physical System (CPS) Model Design

Purpose of grading

1. Does the design meet the assignement ?

   $\rightarrow$ Verification problem (model checking).
2. In case of imperfect design, provide a hint/explanation of what is wrong.

   $\rightarrow$ Fault identification and localization problem.

- Both are hard problems, especially for CPS.
- CPSGrader takes a simple/scalable route by reducing verification and fault identification to testing using a library of fault models
- This document describes the specification language used (Signal Temporal Logic) and how to write tests.

Background: Model Checking

Model — satisfies ? — Specifications

- Should be true for *all* behaviors of the model
  $\rightarrow$ doesn't scale for general CPS

Background: Model Checking

$\boxed{\text{Model}}$ satisfies ? $\boxed{\text{Specifications}}$

- Should be true for *all* behaviors of the model
  $\rightarrow$ doesn't scale for general CPS

- A more tractable approach: simulation + monitoring

$$\boxed{\text{Model}} \longrightarrow \text{trace:}\, aaaabbbaa \ldots \longrightarrow \boxed{\text{Property } \varphi} \longrightarrow \mathit{true}/\mathit{false}$$

Background: Linear Temporal Logic

Used to specify properties on discrete sequences of Boolean propositions.

Background: Linear Temporal Logic

Used to specify properties on discrete sequences of Boolean propositions.

Temporal operators:
"next" (X), "always" (G), "eventually" (F) and "until" ($\mathcal{U}$)

Background: Linear Temporal Logic

Used to specify properties on discrete sequences of Boolean propositions.

Temporal operators:
"next" (X), "always" (G), "eventually" (F) and "until" ($\mathcal{U}$)

A formula is true for a sequence if it is true for the first element.

Background: Linear Temporal Logic

Used to specify properties on discrete sequences of Boolean propositions.

Temporal operators:
"next" (X), "always" (G), "eventually" (F) and "until" ($\mathcal{U}$)

A formula is true for a sequence if it is true for the first element.

Example:

w	a	a	a	b	b	a	a	a	$\ldots$
a	$true$	$true$	$true$	$false$	$false$	$true$	$true$	$true$	$\ldots$
b	$false$	$false$	$false$	$true$	$true$	$false$	$false$	$false$	$\ldots$
$X\ b$	$false$	$false$	$false$	$true$	$true$	$false$	$false$	$?$	$\ldots$

Background: Linear Temporal Logic

Used to specify properties on discrete sequences of Boolean propositions.

Temporal operators:
"next" (X), "always" (G), "eventually" (F) and "until" ($\mathcal{U}$)

A formula is true for a sequence if it is true for the first element.

Example:

w	a	a	a	b	b	a	a	a	$\dots$
a	*true*	*true*	*true*	*false*	*false*	*true*	*true*	*true*	$\dots$
b	*false*	*false*	*false*	*true*	*true*	*false*	*false*	*false*	$\dots$
Xb	*false*	*false*	*true*	*true*	*false*	*false*	*false*	?	$\dots$
G a	*false*	*false*	*false*	*false*	*false*	*true?*	*true?*	*true?*	$\dots$

Background: Linear Temporal Logic

Used to specify properties on discrete sequences of Boolean propositions.

Temporal operators:
"next" (X), "always" (G), "eventually" (F) and "until" ($\mathcal{U}$)

A formula is true for a sequence if it is true for the first element.

Example:

w	a	a	a	b	b	a	a	a	$\dots$
a	$true$	$true$	$true$	$false$	$false$	$true$	$true$	$true$	$\dots$
b	$false$	$false$	$false$	$true$	$true$	$false$	$false$	$false$	$\dots$
Xb	$false$	$false$	$true$	$true$	$false$	$false$	$false$	$?$	$\dots$
G a	$false$	$false$	$false$	$false$	$false$	$true?$	$true?$	$true?$	$\dots$
F b	$true$	$true$	$true$	$true$	$true$	$false?$	$false?$	$false?$	$\dots$

Background: Linear Temporal Logic

Used to specify properties on discrete sequences of Boolean propositions.

Temporal operators:
"next" (X), "always" (G), "eventually" (F) and "until" ($\mathcal{U}$)

A formula is true for a sequence if it is true for the first element.

Example:

w	a	a	a	b	b	a	a	a	...
a	*true*	*true*	*true*	*false*	*false*	*true*	*true*	*true*	...
b	*false*	*false*	*false*	*true*	*true*	*false*	*false*	*false*	...
Xb	*false*	*false*	*true*	*true*	*false*	*false*	*false*	?	...
G a	*false*	*false*	*false*	*false*	*false*	*true?*	*true?*	*true?*	...
F b	*true*	*true*	*true*	*true*	*true*	*false?*	*false?*	*false?*	...
a $\mathbf{U}$ b	*true*	*true*	*true*	*false*	*false*	*false?*	*false?*	*false?*	...

From LTL to STL

Extension of LTL with real-time and real-valued constraints

From LTL to STL

Extension of LTL with real-time and real-valued constraints

Ex: request-grant property

LTL G(r => F g)

Boolean predicates, discrete-time

From LTL to STL

Extension of LTL with real-time and real-valued constraints

Ex: request-grant property

LTL G(r => F g)

Boolean predicates, discrete-time

MTL G(r => $F_{[0,.5s]}$ g)

Boolean predicates, real-time

From LTL to STL

Extension of LTL with real-time and real-valued constraints

Ex: request-grant property

LTL G(r $=>$ F g)

Boolean predicates, discrete-time

MTL G(r $=>$ F$_{[0,.5s]}$ g)

Boolean predicates, real-time

STL G($x[t] > 0$ $=>$ F$_{[0,.5s]}$ $y[t] > 0$)

Predicates over real values , real-time

STL: Syntax

Signals are functions from $\mathbb{R}^n$ to $\mathbb{R}$.

E.g.: positions (x,y,z), orientation θ, sensor values (acc. ax,ay,az), etc.

We denote by $x[\tau]$ the value of signal x at time τ.

STL: Syntax

Signals are functions from $\mathbb{R}^n$ to $\mathbb{R}$.

E.g.: positions (x,y,z), orientation θ, sensor values (acc. ax,ay,az), etc.

We denote by $x[\tau]$ the value of signal x at time τ.

Atomic predicates are inequalities over signal values at **symbolic** time t

E.g.: $x[t] > 0.5$, $z[t] < 4$, $|lws[t] + rws[t]| > 100$, etc.

STL: Syntax

Signals are functions from $\mathbb{R}^n$ to $\mathbb{R}$.

E.g.: positions (x,y,z), orientation θ, sensor values (acc. ax,ay,az), etc.

We denote by $x[\tau]$ the value of signal x at time τ.

Atomic predicates are inequalities over signal values at **symbolic** time t

E.g.: $x[t] > 0.5$, $z[t] < 4$, $|lws[t] + rws[t]| > 100$, etc.

Temporal operators are $\mathbf{F}$, $\mathbf{G}$, $\mathbf{U}$, equiped with a time interval

e.g. $\mathbf{F}_{[0,2]}(x[t] > 0.5)$, $\mathbf{G}_{[0,40]}(y[t] < 0.3)$, $\varphi\mathbf{U}_{[1,2.5]}\psi$, etc.

STL Semantics

A **formula** φ is true if it is true **at time 0**

A **subformula** ψ is evaluated on **future values** depending on temporal operators

Examples

- $\varphi = (x[t] > 0.5)$ is true iff $x[t] > 0.5$ is true when t is replaced by 0, i.e., at the first value of the signal.

- $\varphi = \mathbf{F}_{[0,1.3]}(x[t] > 0.5)$ is true iff $x[t] > 0.5$ is true when t is replaced by any value in [0,1.3].

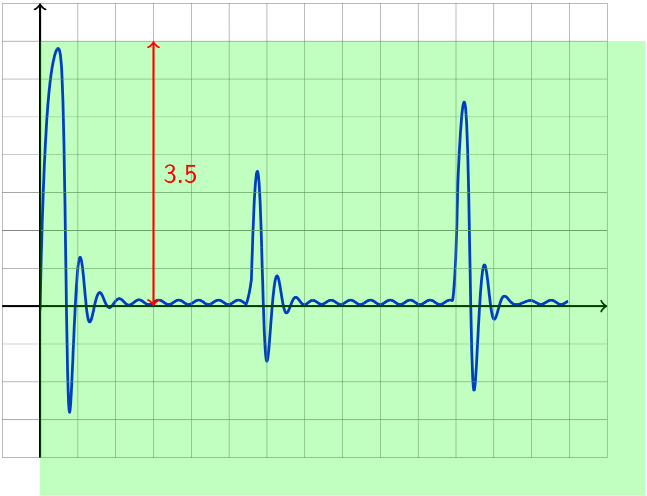
- $\varphi = \mathbf{G}_{[0,1.3]}(\psi)$ is true iff ψ is true at all time in $[0, 1.3]$, i.e., for all suffixes of signals starting at a time in $[0, 1.3]$

STL Examples

STL Examples

The signal is never above 3.5

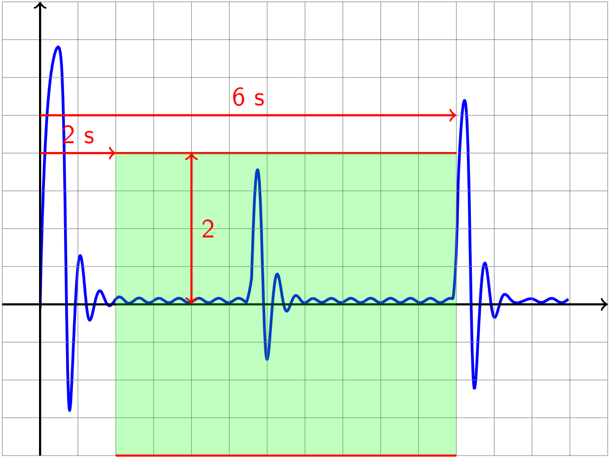
$$\varphi := \mathsf{G}\ (x[t] < 3.5)$$



STL Examples

Between 2s and 6s the signal is between -2 and 2

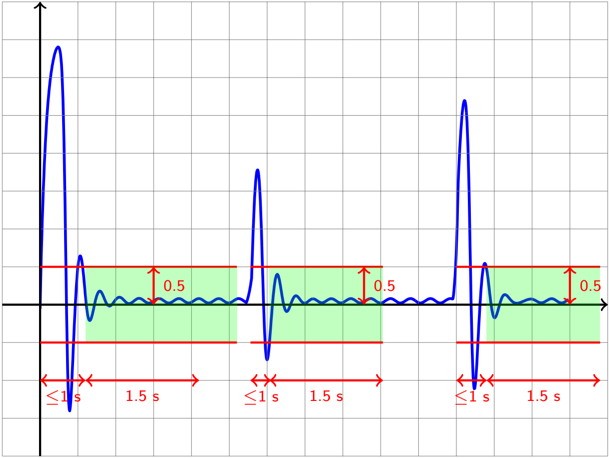
$$\varphi := \ \mathsf{G}_{[2,6]} \ \ (|x[t]| < 2)$$



STL Examples

Always $|x| > 0.5 \Rightarrow$ *after 1 s,* $|x|$ *settles under 0.5 for 1.5 s*

$$\varphi := \mathsf{G}(x[t] > .5 \rightarrow \ \mathsf{F}_{[0,.6]} \ \ (\ \mathsf{G}_{[0,1.5]}\ x[t] < 0.5))$$



Some Theoretical Background

Using CPSGrader

CPSGrader Test Plans

Grading is based on test plans comprizing:

- Test traces:
  System traces obtained in a specific environment setting.
  Should cover all situations relevant to the design requirement.

CPSGrader Test Plans

Grading is based on test plans comprizing:

- Test traces:
  System traces obtained in a specific environment setting.
  Should cover all situations relevant to the design requirement.
- Fault monitors
  STL properties characterizing faults in the design.
  They should detect any behavior of the design indicative of known faults.

CPSGrader Test Plans

Grading is based on test plans comprizing:

- Test traces:
  System traces obtained in a specific environment setting.
  Should cover all situations relevant to the design requirement.
- Fault monitors
  STL properties characterizing faults in the design.
  They should detect any behavior of the design indicative of known faults.

Note that “known faults” should include “not satisfying the design requirement”.

CPSGrader test plans

The general structure of a test plan is as follows:

```
# signal, parameters and formula declarations
...
# test declarations
test test1 {
   fault1 { ...
   fault2 { ...
   ...
}
test test2 {  ...
}
...
test testN { ...
}
```

CPSGrader: Executing Test Plans

CPSGrader will execute test plans as follows

For each test trace
  Get trace $\mathbf{x}$ from simulator
  For each fault with STL formula φ
    Check whether $\mathbf{x} \models \varphi$
    Print feedback
    If fault is critical **then** return
  end
end

CPSGrader: Writing a Test Plan

First, declare signals, parameters and STL formulas and subformulas:

```
# declare signals used in formulas
signal x,y

# Defines some parameters
param y_min= 3., x_max = 5.

# sub formula: defining an (x,y) region which goal is to leave
in_region_to_leave := (y[t]<y_min) or (x[t]>x_max)

# top formula
phi_goal_missed  := alw_[0, 20] (in_region_to_leave)
```

CPSGrader: A concrete example with CyberSim

- Second, define tests and faults. E.g., with CyberSim:

```
# Defining a test
test nav1:"Environment - obstacle south left.xml",20.1,true
{
  fault_goal_missed                     #name of fault
  {phi_goal_missed,                     #formula to monitor
   "PROBLEM:Couldn't avoid obstacle",   #feedback if true
   "",                                  #feedback if false
   true                                 #feedback is critical?
  }
}
```

- Here, `"Environment - obstacle south left.xml,20.1,true"` is a configuration file for CyberSim environment, simulation time and Boolean related to visualization in the CyberSim GUI.
- Using the program `CPSFileGrader` in the `Simulators` folder, a file name for a trace can be specified instead of an XML file.

CPSGrader: Trace Format

Traces in CPSGrader are time-data series in column format, e.g.:

```
0.0  -1.514648  2.5648     0  -3
0.05 -1.514648  3.514648   1  -2
0.15 -1.662522 -21.662522 2  -1
0.25 -1.746353 -3.746353  3   0
0.35 -1.600062 -55.600062 4   10
...
```

where the first column is time.

- The declaration `signal x,y,q,r` means that column 2 is `x[t]`, column 3 is `y[t]`, etc.
- Note: For CyberSim, signals are implicitely declared as `signal x,y,z,pitch,roll,yaw,dist,angle,ax,ay,az,bump_l,...` i.e., the signal declaration is optional.

CPSGrader: Testing with CyberSim

Instruction on how to program CyberSim using C or Statecharts are given in `http://leeseshia.org/lab/`

CyberSim executable is in `Simulators\EECS149lab\CyberSim`

Test plans for CyberSim are located in `Simulators\EECS149lab\CyberSim\data`.

They are executed when Check Grade is pressed.

To test a new test plan, simply (backup and) replace the file `feedback_nav.stl` by a the file with the new test plan.

To test new test plans based on a custom simulator, see instructions in `README.md` file at the root of CPSGrader distribution.

Web sites and contact information

Write to `cpsgrader-dev@lists.eecs.berkeley.edu` for any question.
CPSGrader: `cpsgrader.org`

Another toolbox (Matlab required) with STL support:
Breach: `www.eecs.berkeley.edu/~donze/breach_page.html`