Using Temporal Logics in CPS Autograders

Alexandre Donzé

University of California, Berkeley

October 15, 2015
Grading a CPS Model Design

Purpose of grading

1. Does the design solve the assignment?

2. In case of imperfect design, provide a hint/explanation of what is wrong.
Grading a CPS Model Design

Purpose of grading

1. Does the design solve the assignment?

   Model Checking

2. In case of imperfect design, provide a hint/explanation of what is wrong.
Grading a CPS Model Design

Purpose of grading

1. Does the design solve the assignment?
   Model Checking

2. In case of imperfect design, provide a hint/explanation of what is wrong.
   Counter-examples
Model Checking

- **Classical Setting**
  - Model is a Finite State Machine (FSM)
  - Specifications are LTL formulas

- **CPS Setting**
  - Model is a hybrid system
  - Specifications are Signal Temporal Logic (STL) formulas

\[
\text{Model} \quad \text{satisfies} \quad ? \quad \text{Specifications}
\]
Model Checking

Classical Setting

- Model is a Finite State Machine (FSM)
- Specifications are LTL formulas
Model Checking

Classical Setting
- Model is a Finite State Machine (FSM)
- Specifications are LTL formulas

CPS Setting
- Model is a hybrid system
- Specifications are Signal Temporal Logic (STL) formulas
From Model Checking to Runtime Monitoring

**Problem:** Model checking STL is intractable for complex CPS
From Model Checking to Runtime Monitoring

**Problem:** Model checking STL is intractable for complex CPS

**Runtime Monitoring**

- A more tractable approach: simulation + monitoring

\[
\text{Model} \rightarrow \text{aaaabbbbaa} \ldots \rightarrow \text{Property } \varphi \rightarrow \text{true/false}
\]
From Model Checking to Runtime Monitoring

**Problem:** Model checking STL is intractable for complex CPS

**Runtime Monitoring**

- A more tractable approach: simulation + monitoring
  
  Model → $aaaabbbbaa\ldots$ → Property $\varphi$ → true/false

- Think of (incomplete) depth-first search
From Model Checking to Runtime Monitoring

Problem: Model checking STL is intractable for complex CPS

Runtime Monitoring

- A more tractable approach: simulation + monitoring

\[ \text{Model} \rightarrow aaaaabbbbaa \ldots \rightarrow \text{Property } \varphi \rightarrow \text{true/false} \]

- Think of (incomplete) depth-first search

\[ \Rightarrow \text{can find counter-examples but usually not prove the system correct} \]
Let's evaluate some LTL formula on the trace $w = aaabbaaa$.

<table>
<thead>
<tr>
<th>$w$</th>
<th>$a$</th>
<th>$a$</th>
<th>$a$</th>
<th>$b$</th>
<th>$b$</th>
<th>$a$</th>
<th>$a$</th>
<th>$a$</th>
<th>$a$</th>
<th>$\ldots$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a$</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>false</td>
<td>false</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>$\ldots$</td>
</tr>
<tr>
<td>$b$</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>true</td>
<td>true</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>$\ldots$</td>
</tr>
</tbody>
</table>
Monitoring LTL

Let’s evaluate some LTL formula on the trace $w = aaabbaaa$.

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

<table>
<thead>
<tr>
<th>$w$</th>
<th>$a$</th>
<th>$a$</th>
<th>$a$</th>
<th>$b$</th>
<th>$b$</th>
<th>$a$</th>
<th>$a$</th>
<th>$a$</th>
<th>$a$</th>
<th>$\ldots$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a$</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>false</td>
<td>false</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>$\ldots$</td>
</tr>
<tr>
<td>$b$</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>true</td>
<td>true</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>$\ldots$</td>
</tr>
<tr>
<td>$X b$</td>
<td>?</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>$\ldots$</td>
</tr>
</tbody>
</table>
Monitoring LTL

Let’s evaluate some LTL formula on the trace \( w = aaabbaaa \).

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

\[
\begin{array}{ccccccccccc}
  w & | & a & a & a & b & b & a & a & a & \ldots \\
  a & | & \text{true} & \text{true} & \text{true} & \text{false} & \text{false} & \text{true} & \text{true} & \text{true} & \ldots \\
  b & | & \text{false} & \text{false} & \text{false} & \text{true} & \text{true} & \text{false} & \text{false} & \text{false} & \ldots \\
  X \; b & | & \text{false} & \ldots \\
\end{array}
\]
Let’s evaluate some LTL formula on the trace $w = aaabbaaa$.

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

<table>
<thead>
<tr>
<th>$w$</th>
<th>$a$</th>
<th>$a$</th>
<th>$a$</th>
<th>$b$</th>
<th>$b$</th>
<th>$a$</th>
<th>$a$</th>
<th>$a$</th>
<th>$a$</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a$</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>false</td>
<td>false</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>...</td>
</tr>
<tr>
<td>$b$</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>true</td>
<td>true</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>...</td>
</tr>
<tr>
<td>$Xb$</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>true</td>
<td>true</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>?</td>
</tr>
</tbody>
</table>
Monitoring LTL

Let’s evaluate some LTL formula on the trace \( w = aaabbaaa \).

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

<table>
<thead>
<tr>
<th>( w )</th>
<th>( a )</th>
<th>( a )</th>
<th>( a )</th>
<th>( b )</th>
<th>( b )</th>
<th>( a )</th>
<th>( a )</th>
<th>( a )</th>
<th>( a )</th>
<th>( \ldots )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( a )</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>false</td>
<td>false</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>( \ldots )</td>
</tr>
<tr>
<td>( b )</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>true</td>
<td>true</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>( \ldots )</td>
</tr>
<tr>
<td>( Xb )</td>
<td>false</td>
<td>false</td>
<td>true</td>
<td>true</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>?</td>
<td>( \ldots )</td>
</tr>
<tr>
<td>( alw a )</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>true?</td>
<td>true?</td>
<td>true?</td>
<td>( \ldots )</td>
</tr>
</tbody>
</table>
Let’s evaluate some LTL formula on the trace $w = aaabbaaa$.

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

<table>
<thead>
<tr>
<th>$w$</th>
<th>$a$</th>
<th>$a$</th>
<th>$a$</th>
<th>$b$</th>
<th>$b$</th>
<th>$a$</th>
<th>$a$</th>
<th>$a$</th>
<th>$a$</th>
<th>$\ldots$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a$</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>false</td>
<td>false</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>$\ldots$</td>
</tr>
<tr>
<td>$b$</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>$\text{true}$</td>
<td>$\text{true}$</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>$\ldots$</td>
</tr>
<tr>
<td>$Xb$</td>
<td>false</td>
<td>false</td>
<td>true</td>
<td>true</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>?</td>
<td>$\ldots$</td>
</tr>
<tr>
<td>alw $a$</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>true?</td>
<td>true?</td>
<td>true?</td>
<td>$\ldots$</td>
</tr>
<tr>
<td>ev $b$</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>false?</td>
<td>false?</td>
<td>false?</td>
<td>$\ldots$</td>
</tr>
</tbody>
</table>
Let’s evaluate some LTL formula on the trace \( w = aaabbaaa \).

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

<table>
<thead>
<tr>
<th>( w )</th>
<th>( a )</th>
<th>( a )</th>
<th>( a )</th>
<th>( b )</th>
<th>( b )</th>
<th>( a )</th>
<th>( a )</th>
<th>( a )</th>
<th>( a )</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>( a )</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>false</td>
<td>false</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>...</td>
</tr>
<tr>
<td>( b )</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>true</td>
<td>true</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>...</td>
</tr>
<tr>
<td>( Xb )</td>
<td>false</td>
<td>false</td>
<td>true</td>
<td>true</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>?</td>
<td>...</td>
</tr>
<tr>
<td>( alw \ a )</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>true?</td>
<td>true?</td>
<td>true?</td>
<td>true?</td>
<td>...</td>
</tr>
<tr>
<td>( ev \ b )</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>false?</td>
<td>false?</td>
<td>false?</td>
<td>false?</td>
<td>...</td>
</tr>
<tr>
<td>( a \ U \ b )</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>false</td>
<td>false</td>
<td>false?</td>
<td>false?</td>
<td>false?</td>
<td>false?</td>
<td>...</td>
</tr>
</tbody>
</table>
Outline

1. Signal Temporal Logic

2. CPSGrader: Writing Temporal Testers
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**

<table>
<thead>
<tr>
<th>LTL</th>
<th>G( r =&gt; F g)</th>
</tr>
</thead>
</table>

Boolean predicates, discrete-time
From LTL to STL

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

<table>
<thead>
<tr>
<th>Ex: request-grant property</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>LTL</strong> $\mathbf{G}( r \implies F g)$</td>
</tr>
<tr>
<td><strong>MTL</strong> $\mathbf{G}( r \implies F_{[0,.5s]} g)$</td>
</tr>
</tbody>
</table>
From LTL to STL

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

Ex: request-grant property

LTL \( G( r \Rightarrow F g) \)
Boolean predicates, discrete-time

MTL \( G( r \Rightarrow F_{[0,5s]} g) \)
Boolean predicates, real-time

STL \( G( x[t] > 0 \Rightarrow F_{[0,5s]} y[t] > 0) \)
Predicates over real values, real-time
STL: Syntax

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

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

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

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

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

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

**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 $\mathbb{R}$ to $\mathbb{R}$.

- E.g.: positions ($x,y,z$), orientation $\theta$, sensor values (acc. $ax,ay,az$), etc.

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

**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 $F$, $G$, $U$, equiped with a time interval

- e.g. $F_{[0,2]}(x[t] > 0.5)$, $G_{[0,40]}(y[t] < 0.3)$, $\varphi U_{[1,2.5]} \psi$, etc.

**Remark**: no “next” $X$?
STL Semantics

A **formula** $\varphi$ is true if it is true **at time 0**

A **subformula** $\psi$ 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.

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

- $\varphi = G_{[0,1.3]}(\psi)$ is true iff $\psi$ 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

\[ \varphi := \text{alw } (x[t] < 3.5) \]
Between 2s and 6s the signal is between -2 and 2

\[ \varphi := \text{alw}_{[2,6]} (|x[t]| < 2) \]
Always $|x| > 0.5 \Rightarrow$ after 1 s, $|x|$ settles under 0.5 for 1.5 s

$\varphi := \text{alw}(x[t] > 0.5 \rightarrow \text{ev}_{[0, 0.6]} (\text{alw}_{[0, 1.5]} x[t] < 0.5))$
Signal Temporal Logic

CPSGrader: Writing Temporal Testers
CPSGrader test plans

Grading is based on test plans comprizing:

**Test traces**

System traces obtained in a specific environment setting.
They 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.
They 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.

Known faults include not satisfying the design requirement.
CPSGrader test plans

The general structure of a test plan is as follows:

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

test testN { ...
}
```
CPSGrader will execute test plans as follows

**For each** test trace
   Get trace \( x \) from simulator
   **For each** fault with STL formula \( \varphi \)
      Check whether \( x \models \varphi \)
      Print feedback
      **If** fault is critical **then** return
   **end**
**end**
First, declare signals, parameters and STL formulas and subformulas:

```plaintext
# 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)
```
Second, define tests and faults.

```plaintext
# Defining the 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?
    }
}
```
Demo

- CyberSim/CPSGrader autograder and test plans
  - modify `data\feedback_nav.stl` or `data\feedback_nav_hill.stl`

- Exploring properties using Breach
  - [www.eecs.berkeley.edu/~donze/breach_page.html](http://www.eecs.berkeley.edu/~donze/breach_page.html)
  - At the Matlab prompt:
    ```
    >> Breach('feedback_nav.stl')
    ```
Demo

- CyberSim/CPSGrader autograder and test plans
  ⇒ modify data/feedback_nav.stl or data/feedback_nav_hill.stl

- Exploring properties using Breach
  (www.eecs.berkeley.edu/~donze/breach_page.html)
  ⇒ at the Matlab prompt >> Breach(‘feedback_nav.stl’) Then menu Files->Import Trajectory from File
Conclusion

- Tools:
  - **CPSGrader**: cpsgrader.org
    - CyberSim
    - Other simulator or data: monitoring trace files
  - **Breach**: www.eecs.berkeley.edu/~donze/breach_page.html
    - Simulink models + other simulators via simple wrapper
    - Richer GUI to work with STL formulas

- Suggestions of faults/feedback expressible in STL that would have helped you debug your controller?
- Project idea where CyberSim/CPSGrader or Breach could help with better testing your design?
- Send e-mail to (donze@berkeley.edu) for any related question