Autograding a CPS Lab using Signal Temporal Logic

Alexandre Donzé

University of California, Berkeley

October 16, 2014
Grading a CPS Model Design

Purpose of grading

1. Does the design meet 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 meet 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 meet the assignment?
   
   Model Checking!

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

   Counter-examples!
Model Checking

- Model
- satisfies ?
- Specifications

Last time:
- Model is an FSM
- Specifications are LTL formulas

Today:
- Model is a hybrid system
- Specifications are Signal Temporal Logic (STL) formulas
Model Checking

Model \quad \text{satisfies} \quad ? \quad \text{Specifications}

Last time

- Model is an FSM
- Specifications are LTL formulas
Model Checking

Model satisfies Specifications

Last time
- Model is an FSM
- Specifications are LTL formulas

Today
- Model is a hybrid system
- Specifications are Signal Temporal Logic (STL) formulas
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

<table>
<thead>
<tr>
<th>Model</th>
<th>aaaabbbbaa...</th>
<th>Property $\varphi$</th>
<th>true/false</th>
</tr>
</thead>
</table>
From Model Checking to Runtime Monitoring

Problem: Model checking STL is intractable for complex CPS

Runtime Monitoring

- A more tractable approach: simulation + monitoring
  
  Model → aaaabbbaa... → 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
  
  \[ \text{Model} \rightarrow \text{aaaabbbbaa...} \rightarrow \text{Property } \varphi \rightarrow \text{true/false} \]

- Think of (incomplete) depth-first search

  \[ \Rightarrow \text{can find counter-examples but usually not prove the system correct} \]
Monitoring LTL

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>...</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>...</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>...</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>$X \ b$</td>
<td>?</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></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>...</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>
</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>
</tr>
<tr>
<td>$X$ $b$</td>
<td>false</td>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></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>( X \ 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>
</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>...</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>...</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>...</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>?</td>
<td>...</td>
</tr>
<tr>
<td>$G\ a$</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>
</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>...</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>...</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>...</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>?</td>
<td>...</td>
</tr>
<tr>
<td>$G$ $a$</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>...</td>
</tr>
<tr>
<td>$F$ $b$</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>...</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.

\[
\begin{array}{cccccccccc}
  w & a & a & a & b & b & a & a & a & \ldots \\
  a & \text{true} & \text{true} & \text{true} & \text{false} & \text{false} & \text{true} & \text{true} & \text{true} & \ldots \\
  b & \text{false} & \text{false} & \text{false} & \text{true} & \text{true} & \text{false} & \text{false} & \text{false} & \ldots \\
  Xb & \text{false} & \text{false} & \text{true} & \text{true} & \text{false} & \text{false} & \text{false} & \text{false} & \text{?} & \ldots \\
  G a & \text{false} & \text{false} & \text{false} & \text{false} & \text{false} & \text{false} & \text{false} & \text{false} & \text{false} & \text{false} & \text{false} & \text{false} & \text{false} & \text{false} & \text{false} & \text{false} \ldots \\
  F b & \text{true} & \text{true} & \text{true} & \text{true} & \text{true} & \text{true} & \text{false} & \text{false} & \text{false} & \text{false} & \text{false} & \text{false} & \text{false} & \text{false} & \text{false} & \text{false} \ldots \\
  a \ U \ b & \text{true} & \text{true} & \text{true} & \text{false} & \text{false} & \text{false} & \text{false} & \text{false} & \text{false} & \text{false} & \text{false} & \text{false} & \text{false} & \text{false} & \text{false} & \text{false} \ldots \\
\end{array}
\]
Outline

1. Signal Temporal Logic
2. CPSGrader: Writing Temporal Testers
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

LTL \( G( r \Rightarrow F g) \)

Boolean predicates, discrete-time
From LTL to STL

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

Ex: request-grant property

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

**MTL** \( G( r \Rightarrow F_{[0,.5s]} g) \)
Boolean predicates, real-time
From LTL to STL

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

**Ex: request-grant property**

**LTL** $G( r \implies F g)$  
Boolean predicates, discrete-time

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

**STL** $G( x[t] > 0 \implies F_{[0,.5s]} y[t] > 0 )$  
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 $F$, $G$, $U$, equipped 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
The signal is never above 3.5

\[ \varphi := G \left( x[t] < 3.5 \right) \]
STL Examples

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

\[ \phi := G_{[2,6]} (|x[t]| < 2) \]
STL Examples

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

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

```
# 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 $x$ 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:

```csharp
# 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.

```csharp
# Defining the test
test nav1: "Environment - obstacle south left.xml", 20.1, true {
    fault_goal_missed
    {phi_goal_missed,
        "PROBLEM: Couldn’t avoid obstacle",
        "",
        true
    }
}
```
Demo
Demo

- CyberSim/CPSGrader autograder and test plans
- Exploring properties using Breach
- Writing new properties and test plans
Conclusion

- You are strongly encouraged to use STL to test your design for your project when applicable
- We will provide support on request on how to use
  
  **CPSGrader**: cpsgrader.org and/or
  **Breach**: www.eecs.berkeley.edu/~donze/breach_page.html
- Contact me (donze@berkeley.edu) for any related question