Formal Specification: Temporal Logic

Design Verification

Is the design consistent with the original specification?

Is what I think I want what I really want?
Today’s Lecture

Formal (mathematical) Specification

How do you formally state what your design should do?

Temporal Logic

- A mathematical way to express properties of a system over time
  - E.g., Behavior of an FSM or Hybrid System

- Many flavors of temporal logic
  - Propositional temporal logic (we will study this)
  - Real-time temporal logic

- Amir Pnueli won ACM Turing Award, in part, for the idea of using temporal logic for specification
Example: Specification of the *SpaceWire* Protocol
(European Space Agency standard)

8.5.2.2 **ErrorReset**

a. The *ErrorReset* state shall be entered after a system reset, after link operation is terminated for any reason or if there is an error during link initialization.

b. In the *ErrorReset* state the Transmitter and Receiver shall all be reset.

c. When the reset signal is de-asserted the *ErrorReset* state shall be left unconditionally after a delay of 6.4 μs (nominal) and the state machine shall move to the *ErrorWait* state.

d. Whenever the reset signal is asserted the state machine shall move immediately to the *ErrorReset* state and remain there until the reset signal is de-asserted.

---

**Current status of property specification in HW**

Usage of formal property specification languages is becoming widespread

- 68% in 2007 (John Cooley, DVCon’07)
- Properties often called “assertions”

Properties are used not just in formal verification, but also in simulation

- “Assertion-Based Verification” (ABV)

Some property specification languages: PSL/Sugar, System Verilog Assertions (SVA), OVA, OVL, etc.

All of these are just ways of writing variants of Temporal Logic
Execution Trace of a State Machine

An execution trace is a sequence of the form

$q_0, q_1, q_2, q_3, \ldots,$

where $q_j = (x_j, s_j, y_j)$ where $s_j$ is the state at step $j$, $x_j$ is the input valuation at step $j$, and $y_j$ is the output valuation at step $j$. Can also write as

$$s_0 \xrightarrow{x_0/y_0} s_1 \xrightarrow{x_1/y_1} s_2 \xrightarrow{x_2/y_2} \cdots$$
Propositional Logic on Traces

A propositional logic formula $p$ holds for a trace $q_0, q_1, q_2, q_3, \ldots$, if and only if it holds for $q_0$.

This may seem odd, but we will provide temporal logic operators to reason about the trace.

Linear Temporal Logic (LTL)

LTL formulas: Statements about an execution trace $q_0, q_1, q_2, q_3, \ldots$.

<table>
<thead>
<tr>
<th>formula</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>$p$</td>
<td>$p$ holds in $q_0$</td>
</tr>
<tr>
<td>$G\phi$</td>
<td>$\phi$ holds for every suffix of the trace</td>
</tr>
<tr>
<td>$F\phi$</td>
<td>$\phi$ holds for some suffix of the trace</td>
</tr>
<tr>
<td>$X\phi$</td>
<td>$\phi$ holds for the trace $q_1, q_2, \cdots$</td>
</tr>
<tr>
<td>$\phi_1 U \phi_2$</td>
<td>$\phi_1$ holds for all suffixes of the trace until a suffix for which $\phi_2$ holds.</td>
</tr>
</tbody>
</table>

Here, $p$ is propositional logic formula and $\phi$ is either a propositional logic or an LTL formula.
Linear Temporal Logic (LTL)

**LTL formulas**: Statements about an execution trace $q_0, q_1, q_2, q_3, \dots$

<table>
<thead>
<tr>
<th>formula</th>
<th>mnemonic</th>
</tr>
</thead>
<tbody>
<tr>
<td>$p$</td>
<td>proposition</td>
</tr>
<tr>
<td>$G\phi$</td>
<td>globally</td>
</tr>
<tr>
<td>$F\phi$</td>
<td>finally, future, eventually</td>
</tr>
<tr>
<td>$X\phi$</td>
<td>next state</td>
</tr>
<tr>
<td>$\phi_1 U \phi_2$</td>
<td>until</td>
</tr>
</tbody>
</table>

Here, $p$ is propositional logic formula and $\phi$ is either a propositional logic or an LTL formula.

First LTL Operator: $G$ (Globally)

The LTL formula $Gp$ holds for a trace $q_0, q_1, q_2, q_3, \dots$ if and only if it holds for every suffix of the trace:

$\bar{q}_0, \bar{q}_1, \bar{q}_2, \bar{q}_3, \dots$

$\bar{q}_1, \bar{q}_2, \bar{q}_3, \dots$

$\bar{q}_2, \bar{q}_3, \dots$

$\bar{q}_3, \dots$

If $p$ is a propositional logic formula, this means it holds for each $q_i$. 
Second LTL Operator: F (Eventually, Finally)

The LTL formula $\mathcal{F}p$ holds for a trace

$q_0, q_1, q_2, q_3, \ldots$

if and only if it holds for some suffix of the trace:

$q_0, q_1, q_2, q_3, \ldots$
$q_1, q_2, q_3, \ldots$
$q_2, q_3, \ldots$
$q_3, \ldots$

If $p$ is a propositional logic formula, this means it holds for some $q_i$.

---

Propositional Linear Temporal Logic

LTL operators can apply to LTL formulas as well as to propositional logic formulas.

E.g. Every input $x$ is eventually followed by an output $y$

$\mathcal{G}(x \implies \mathcal{F}y)$

Globally (at any point in time)
If $x$ occurs
It is eventually followed by $y$
Every input $x$ is eventually followed by an output $y$

The LTL formula $G(x \implies Fy)$ holds for a trace

$q_0, q_1, q_2, q_3, \ldots,$

if and only if it holds for any suffix of the trace where $x$ holds, there is a suffix of that suffix where $y$ holds:

$q_0, q_1, q_2, q_3, \ldots$

$q_1, q_2, q_3, \ldots \quad y \text{ holds}$

$x \text{ holds} \quad q_2, q_3, \ldots$

$q_3, \ldots$

Third LTL Operator: $X$ (Next)

The LTL formula $Xp$ holds for a trace

$q_0, q_1, q_2, q_3, \ldots,$

if and only if it holds for the suffix $q_1, q_2, q_3, \ldots$

$q_0, q_1, q_2, q_3, \ldots$

$q_1, q_2, q_3, \ldots$

$q_2, q_3, \ldots$

$q_3, \ldots$
Fourth LTL Operator: U (Until)

The LTL formula \( p_1 U p_2 \) holds for a trace

\[ q_0, q_1, q_2, q_3, \ldots, \]

if and only if \( p_2 \) holds for some suffix of the trace, and \( p_1 \) holds for all previous suffixes:

\[ q_0, q_1, q_2, q_3, \ldots \]
\[ q_1, q_2, q_3, \ldots \]
\[ q_2, q_3, \ldots \]
\[ q_3, \ldots \]

- \( p_1 \) holds
- \( p_2 \) holds (and maybe \( p_1 \) also)

Examples: What do they mean?

- **G F p**  
  *p holds infinitely often*
- **F G p**  
  *Eventually, \( p \) holds henceforth*
- **G( p => F q )**  
  *Every \( p \) is eventually followed by a \( q \)*
- **F( p => (X X q) )**  
  *Every \( p \) is followed by a \( q \) two steps later*

Remember:

- **Gp** \( p \) holds in all states
- **Fp** \( p \) holds eventually
- **Xp** \( p \) holds in the next state
Temporal Operators & Relationships

G, F, X, U: All express properties along system traces

- Can you express \( G \phi \) purely in terms of F, p, and Boolean operators?
  \[ G\phi = \neg F\neg \phi \]

- How about \( F \phi \) in terms of U?
  \[ F\phi = true \ U \ \phi \]

- What about \( X \phi \) in terms of G, F, or U?
  Cannot be done

Examples in Temporal Logic

“No more than one processor (in a 2-processor system) should have a cache line in write mode”
  - \( wr_1 \) / \( wr_2 \) are respectively true if processor 1 / 2 has the line in write mode

“The grant signal must be asserted at some time after the request signal is asserted”
  - Signals: grant, req

“A request signal must receive an acknowledge and the request should stay asserted until the acknowledge signal is received”
  - Signals: req, ack
Safety vs. Liveness

Safety property
“something bad must not happen”
E.g.: system should not crash
Finite length error trace

Liveness property
“something good must happen”
E.g.: every packet sent must be received at its destination
Infinite length error trace

Asserts in PSL/Sugar (Verilog flavor)

G (req \rightarrow X(X(X grant))))
assert always req \rightarrow next[3] (grant);

G(req \rightarrow X ( ack U grant))
assert always req \rightarrow next (ack until grant);
From Temporal Logic to Monitors

A monitor for a temporal logic formula is a state machine that represents all the behaviors that satisfy the temporal logic formula.

Why are monitors useful?

Monitor for $G \ p$, $p$ a Boolean formula

- Start
- Error
- $\neg p$
- $p$
Monitor for \( F \, p \), \( p \) a Boolean formula

Monitor for \( GF \, p \), \( p \) a Boolean formula
Some User Reactions

“We’re using SVA. I expect new RTL to be as littered with assertions as the Wisconsin countryside is littered with cheese shops & taverns.”

Make it easy to write and embed in RTL

“Started using Sugar PSL and OVA. Not clear yet as to the advantages. You have to debug the assertions, too!”

Specifications must be debugged too!

“We use 0-In assertions. I would say that our current maturity with the 0-In tools puts us at 50% efficiency. In some instances the assertions have little to no value. In some instances they are essential.” Training and improved scalability needed

“Awesome Baby!!!!!!!!!! Use PSL and they are very useful. ”

What more can one say!

Compiled by John Cooley, at DVCon’05