Temporal Logic and Model Checking

Sanjit Seshia

EECS

UC Berkeley

Recap: Design Verification Techniques

Software Simulation

- Application of simulation stimulus to model of circuit

Hardware Accelerated Simulation

- Use of special purpose hardware to accelerate simulation of circuit

Emulation

- Emulate actual circuit behavior - e.g. using FPGA's

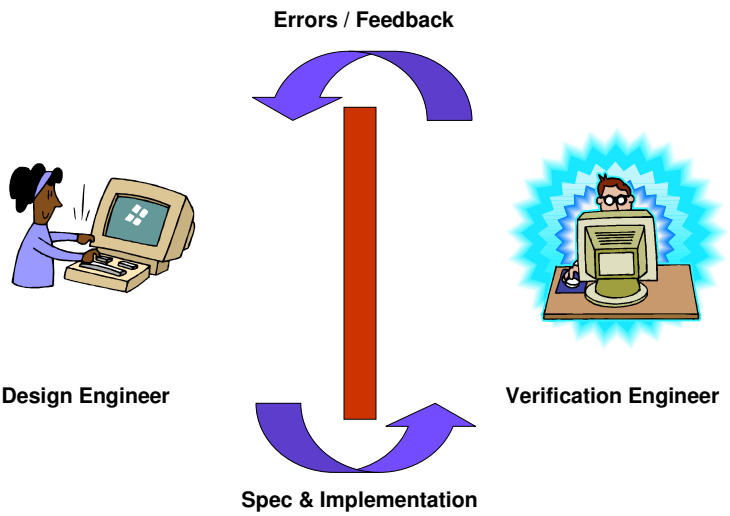
Rapid Prototyping

- Create a prototype of actual hardware

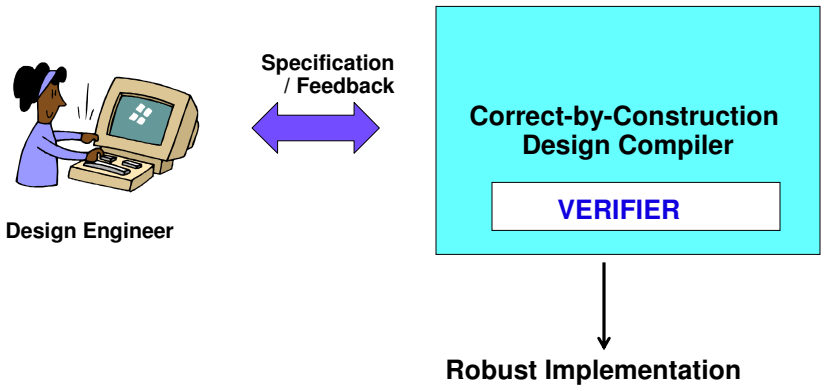
Formal Verification

- **Model checking** – exhaustively enumerate all possible states of (a model of) the circuit and check for errors
- Theorem proving – (automatically) prove theorems regarding properties of a circuit model

Current Practice



What we want



Formal Verification as practiced today

Verification Engineer

RTL

Properties

(Spec)

Correct /
Error trace

MODEL GENERATER

MODEL CHECKER

Today's Lecture

Formal (mathematical) Specification

How do you formally state
what your design should do?

Current status of property specification

Usage of formal property specification languages is increasing (gradually)

- 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)

- All of these are just ways of writing **Temporal Logic**

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."

"Started using Sugar PSL and OVA. Not clear yet as to the advantages. You have to debug the assertions, 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."

"Awesome Baby!!!!!!!!! Use PSL and they are very useful. "

Compiled by John Cooley, at DVCon'05

Lecture Outline

- Behavior/Trace/Execution
- Specifying Properties: Safety vs. Liveness
- Temporal Logic (its 2 main flavors)
- Model Checking (simplest case)
- Synthesizing Monitors from Properties

Behavior / Trace / Execution / Path

Suppose a finite-state system has

- n inputs, $x = (x_1, x_2, \ldots, x_n)$
- m outputs, $z = (z_1, z_2, \ldots, z_m)$
- k state variables, $s = (s_1, s_2, \ldots, s_k)$

Let t = 0, 1, 2, 3, … denote time (clock ticks)

What's a behavior of the system?

Folding Everything into the State

Let's assume that inputs and outputs are latched and are part of the state bits

Then, what is a behavior of the system?

Transformational vs. Reactive Systems

Transformational System

- Computes a function mapping inputs to outputs
- Terminates and generates output

Reactive System

- Non-terminating, runs forever reacting to inputs

- Assume we have a complete specification.
- How do we verify a transformational system? A reactive system?

Simulating Reactive Systems

What does simulation do?

- What's the specification?
- What behaviors are simulated?

Specifying Properties

- Ideally, want a complete specification
  - Implementation must be equivalent to the specification
- In practice, only have partial specifications
  - Specify some "good" behaviors and some "bad" behaviors
  - View this as sets

Safety vs. Liveness

Safety property

- "something bad must not happen"
- E.g.: system should not crash

Liveness property

- "something good must happen"
- E.g.: every packet sent must be received at its destination

Examples: Safety or Liveness?

1. "No more than one processor (in a multi-processor system) should have a cache line in write mode"

2. "The grant signal must be asserted at some time after the request signal is asserted"

3. "A request signal must receive an acknowledge and the request should stay asserted until the acknowledge signal is received"

Example: Safety or Liveness?

4. "From any state, it is possible to return to the reset state"

5. "The grant signal must be asserted 3 cycles after the request signal is asserted"

The term "Logic" (for this lecture)

- What it doesn't mean: a circuit
- It roughly means: A formal way of describing a design, along with algorithms to reason about it ("argue" about its correctness or lack thereof)
- E.g., Boolean (propositional) logic is a way to describe the functionality of combinational circuits and prove their correctness

Temporal Logic

- A logic for reasoning about properties over time
  - E.g., Behavior of a finite-state system
- We will study propositional temporal logic
  - Other temporal logics exist: e.g., real-time temporal logic

Property on a Single State

A Boolean formula over state variables

We will denote each unique Boolean formula by
- a distinct color
- a name such as p, q, …

req

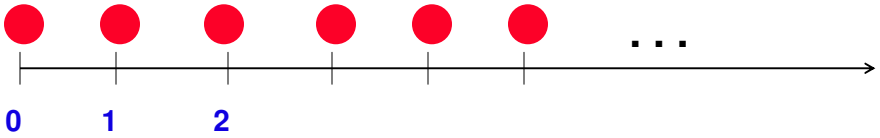
req & !ack

Globally p: G p

G p is true in a state if p holds at all points of time (along the path) starting from that state

p =

Suppose G p holds in the initial state

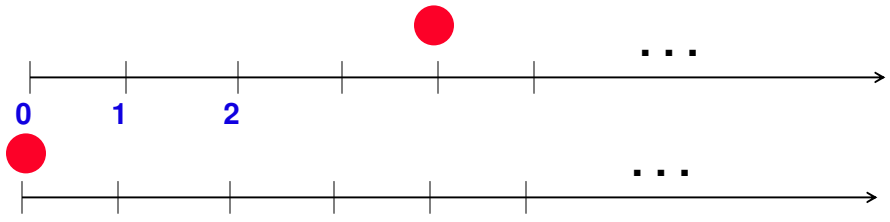


Eventually p: F p

F p is true in a state if p holds at some point in the future, starting from that state

p =

Suppose F p holds in the initial state

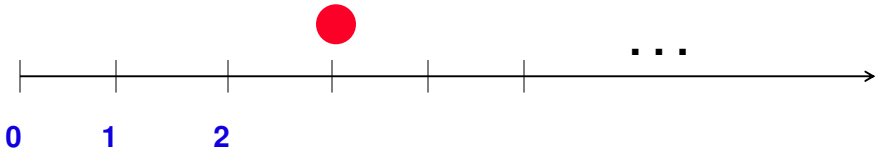


Next p: X p

X p is true along a path starting in a state if p holds in the next state

p = 

Suppose X p holds in state at t = 2



Nesting of Formulas

p need not be just a Boolean formula.

It can be a temporal logic formula itself!

p = 

"X p holds in all states, starting from initial state"

How can we write this in temporal logic?

How do we draw this?

Examples: What do they mean?

- G F p
- F G p
- G(p → F q)
- F(p → (X X q))

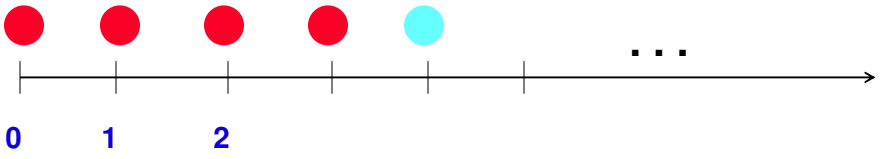
p Until q: p U q

p U q is true in a state if

- q is true in some state reachable from s
- p is true in all states from s until q holds

p = ● q = ●

Suppose p U q holds in initial state



Temporal Operators & Relationships

G, F, X, U: All express properties along traces/ paths

- Can you express G p purely in terms of F, p, and Boolean operators ?
- How about G and F in terms of U?
- What about X in terms of G, F, or U?

Examples in Temporal Logic

1. "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
2. "The grant signal must be asserted at some time after the request signal is asserted"
   - Signals: grant, req
3. "A request signal must receive an acknowledge and the request should stay asserted until the acknowledge signal is received"
   - Signals: req, ack

Examples in Temporal Logic

4. "From any state, it is possible to return to the reset state"
   - Signal: reset

5. "The grant signal must be asserted 3 cycles after the request signal is asserted"
   - Signals: grant, req

Asserts in PSL/Sugar (Verilog flavor)

G (req → X(X(X grant))))

assert always req → next[3] (grant);

G(req → X (ack U grant))

assert always req → next (ack until grant);

Temporal Logic Flavors

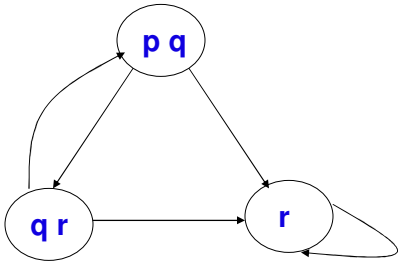
Linear Temporal Logic

- Properties expressed over a single time-line
- What we've seen so far

Computation Tree Logic

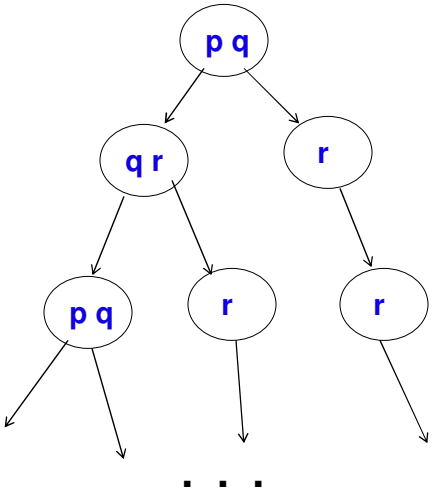
- Properties expressed over a tree of all possible executions
- Where does this "tree" come from?

Labelled State Transition Graph



"Kripke structure"

What's different about this picture and other FSM pictures we've seen so far?



Infinite Computation Tree

Non-determinism

- A technique used for modeling choice, where you don't know at design time how the choice will be resolved, or what the underlying probability distribution will be
  - E.g., a lossy network might either drop a packet or not
- Explore both choices "in parallel"

Computation Tree Logic (CTL*)

Introduce two new operators called "Path quantifiers"

- **A p** : Property p holds along all computation paths
- **E p** : Property p holds along at least one path
- How to express this:

  "The grant signal must be asserted at some time after the request signal is asserted"

  A G (req → A F grant)

More CTL

"From any state, it is possible to get to the reset state"

A G (E F reset)

CTL vs. LTL

- Have different expressive powers
- Seems like LTL is easier for designers to understand, hence more commonly used

Verification of G p

Suppose p is a Boolean formula (no temporal operators in p)

How can we verify G p? (based on what you've already learnt in this course)

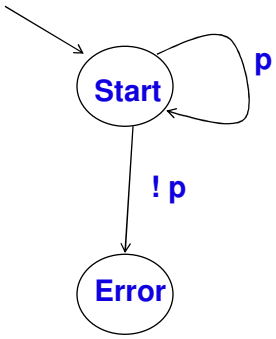
From Temporal Logic to Monitors

A monitor for a temporal logic formula

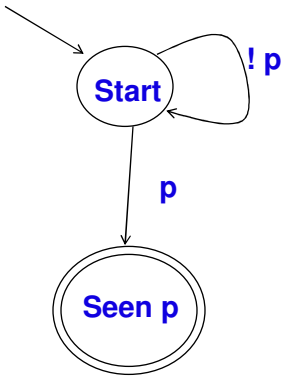
- is a state machine
- represents all the behaviors that satisfy the temporal logic formula

Why are monitors useful?

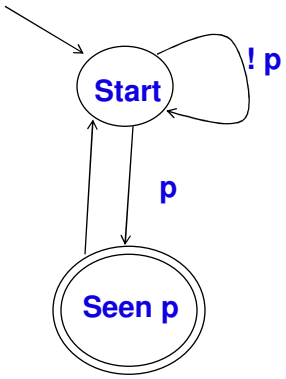
Monitor for G p, p a Boolean formula



Monitor for F p, p a Boolean formula ?



Monitor for GFp, 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

Next Lecture

How can we automatically verify arbitrary temporal logic properties?

- Using model checking
- Using BDDs and SAT

What's next in verification?