designing a traffic signal

Daniel Jackson
February 20, 2008
announcements

project clarifications
- problem refinement
- abstract design
- code design

Java question up
- do before Monday
- due in next Thursday’s LNB review

reminders
- project 1A due tomorrow
- second Java lab on Friday
plan for today

topics

• response to feedback
• invariants and interlocks
• designing a traffic signal
response to feedback

simultaneous events?

- can’t happen, by definition!
- interleaving model of concurrency
- if not instantaneous, model with start/end events

parallel machines

- more on this today

absorbing key repeats

- discussed in first lecture, but omitted in code
- show how to incorporate in code (see course website, with these slides)
invariants & interlocks
specification

so far

- state machines as a design notation
- but what if you don’t implement directly?
- how much leeway?

ways to specify

- machine: allow these transitions
  \[
  \text{op } \text{insertCoin}(c) \{ \text{when } \text{bal} < \$1 \text{ do } \text{bal} += c \}\]

- traces: allow these event sequences
  \[
  \text{<insertCoin}($1), \text{deliverChocolate}>\]

- invariant: allow these states
  \[
  \text{balance} - (\#\text{chocolateBars} \times \$1) \leq \$2\]
invariants are your friend

often they give you
  · the simplest way to express important properties

can often be checked in code
  · with runtime assertions

can be reasoned about
  · inductive reasoning especially powerful
interlocks

simplest way to maintain an invariant

- before each transition, check it won’t break invariant
- also called a “gatekeeper”

advantages

- simple to understand and often easy to implement
- because it’s local, can be very reliable (“small trusted base”)

© Daniel Jackson 2008
Therac-25 accident

Therac-25 radiotherapy machine

- two modes: electron beam and xray
- rotating turntable with 3 positions
- two power levels, lo and hi
- invariant: in xray mode, use flattener
  \[ \text{power} = \text{HI} \Rightarrow \text{turntable} = \text{FLATTENER} \]

what happened

- earlier version had hardware interlocks
- software had concurrency bug
- invariant violated and 6 patients overdosed, of whom at least 3 died

is your PC secure?

typical patch size
  - 100MB

typical time to download
  - 10 minutes

average time to infection*
  - 4 minutes

* [Windows XP, default firewall settings] Unprotected PCs Fall To Hacker Bots In Just Four Minutes
Gregg Keizer; Nov 30, 2004; http://www.techweb.com/wire/security/54201306
From: Security Absurdity: The Complete, Unquestionable, And Total Failure of Information Security
Noam Eppel; http://securityabsurdity.com

© Daniel Jackson 2008
buffer overflows

problem of buffer overflows

- a major source of security vulnerabilities
- huge cost to industry and individuals

how they work

- program reads messages into buffer of fixed capacity
- buffer is stack allocated; below it is return address for call
- rogue agent passes big message
- buggy code writes message over return address
- return address is replaced by address of code inside message
how to avoid overflows

interlocks

• invariant: buffer size < buffer capacity
• check before writing message into buffer
• eg, for each array update, check bounds
  
    a[i] = e  // only if 0 ≤ i < MAX

so why don’t people do this?

• most programs in “unsafe” languages like C that don’t check bounds
• programmers say too costly to check (but cost of not checking?)

lesson

• add an interlock (with a safe language, or a data abstraction)
• (or prove invariant preservation -- more on this soon)
why not interlocks?

when possible, interlocks are great

but they don’t always work

- rejecting events makes things complicated (and users unhappy)
- doing the check might damage performance
  - eg: database index is properly ordered
- may not be able to see the state
  - state is distributed, so nobody has global view
    - eg. node can’t see state of whole network
  - state cannot be read at all
    - eg. radiotherapy machine can’t read dose received by patient
simple traffic lights
simple traffic lights

parallel machine semantics

- in each step, one event occurs
- if event belongs to both machines, both must do the transition
- if event belongs to just one, only that machine moves
product machine

can form a single product machine
  · states are tuples, one state from each machine

“state explosion”
  · \( k \) machines of \( N \) states
  · product machine has \( N^k \) states
  · this is why concurrency is hard!
traces

what's a trace?
- a trace is an event history
- machine has set of traces
- includes empty trace

example
- traces of traffic light include
  
  
  <>,
  <r2>,
  <r2, g1>,
  <r2, g1, r1>,
  <r2, g1, r1, g2>, ...

© Daniel Jackson 2008
traffic light invariant

what's the traffic light invariant?

• crucial property: never green both ways
• so invariant is not GG

how to check?

• just look at product machine
• colour satisfying states yellow
• check all reachable states are yellow

but doesn’t scale

• how would you do this for 3 parallel machines of 10 states each?
induction

strategy

• check invariant holds initial state (1)
  \[ I(S_0) \]
• check that transitions preserve invariant (2)
  \[ (s, e, s') \in R \land I(s) \Rightarrow I(s') \]
• then invariant holds in all reachable states

why?

• consider any trace
• holds at start by (1)
• can repeatedly add events using (2), and holds after each
• (in general, this unfolding gives a tree: can you see why?)
a classic problem

problem statement

† 8 x 8 chessboard can be filled with 32 dominos
† suppose we remove top-left and bottom-right squares
† can remaining 62 squares be tiled with 31 dominos?

reasoning with an invariant

† consider number of black, white squares covered
† invariant: \texttt{black} = \texttt{white}
† initially, \texttt{black} = \texttt{white} = 0
† only operation is \texttt{placeDomino (loc)}
  always adds 1 to \texttt{black} and to \texttt{white}
  so it preserves the invariant
† board with corners removed has 32 black, 30 white
  this state does not satisfy the invariant, so it’s not reachable
not all invariants are inductive

consider our invariant not GG
- unfortunately, it’s not inductive
- consider this transition: (GW, g2, GG)
- invariant holds before but not after

what’s wrong?
- need to strengthen the invariant
- standard trick in induction: “strengthening the inductive hypothesis”

what’s the strengthened invariant here?
- R1 or R2 // one machine is in red state
- note that this implies not GG
asynchronous traffic lights
problem

road works

• road narrows to one lane
• workers have flags but can’t see each other

invariant desired

• not both flags green

a strategy

• start out one green, one waiting
• worker gives last car to get through a message and shows red flag
• worker at other end gets message and shows green flag
state machine model

events

r0: worker 0 raises red flag
send0: worker 0 sends message to worker 1
recv0: worker 0 receives message from worker 1
product machine

showing reachable states only

• still deterministic
• two more states than before, for messages in transit
• easy to see that invariant holds
summary
what did we do?

designed a traffic signalling scheme

- using invariants -- very powerful technique

invariants

- an invariant describes a state set
- all reachable states should be in set
- can reason inductively, one operation at a time
- may need to strengthen invariant

fault tolerance

- noted that our scheme is not fault-tolerant: more on this next time
lecture exercise

by next lecture (Monday) in your LNBs

chocolate machine

› [easy] model a machine with two events (insertCoin and deliverChoc) and prove that it preserves an invariant about the balance

clayton railway accident

› [hard, optional] model the protocol and explain what went wrong