Formal Methods in System Design, Lec 4
Recap, and look forward

- Safety and Liveness recap
- Need for Automata on infinite words to express Liveness (and violations thereof)
- Next week, the TAs will introduce to you Promela (modeling language) and SPIN (verifier)
- Readings: Chapter 21 of my book, and also many-promela-exs. {tex, pdf} which will be kept online.
- I’ll resume lecturing on 1/27/11 and show you many of the protocols you’ve seen, now in Promela
- We will check additional properties also
- Asg2 : Inductive invariants, “man-wolf..”, Bakery protocol, Reading assignment: Chapter 21 plus “many-promela-exs..”
Recap

• Transition systems
• Assertions: “weakest” and “strongest”
• Invariants
• Inductive invariants
• Non-inductive invariants
• Strongest invariant
“Geometric” view of invariants

• Consider the transition system
• Var : x : {0,..,8}
• Init: x := 0;
• Rules:
  – R1: true ➔ if (0<= x <= 2) then x := (x+1)%3
  – elseif (3<= x <= 5) then x := (x+1)%3 + 3
  – else // if (6<= x <= 8) then
  – x := (x+1)%3 + 6
• What is the strongest invariant?
• Find two inductive and two non-inductive invariants
• Have a Geometric view of these
Invariant of BinSrch

• Draw a flow-chart of BinSrch
• Show how to infer the WEAKEST precondition
• Show how to infer the STRONGEST postcondition
• Drill : proof that the XOR based swap swaps!
• This shows the power of “weak” versus “strong” assertions
Invariant of BinSrch

• How do we compute the wp (weakest precondition) of “if-then”?
• How do we find an invariant for BinSrch’s loop?
• How do we confirm that this invariant is inductive around the Binsrch loop?
• Inductive lets you “forget” how many times you iterated
• How a good loop invariant allows you to confirm the desired postcondition of BinSrch
Asg2, Problem 1

• Design your own example similar to the one shown in the “Geometric” interpretation of invariants
Question on Murphi that you asked

• Symmetry and Scalar Sets

• Any other questions on Murphi?
  – Helpful hints:
    • Use “put” sparingly, but it can help
    • The “-s” random-simulation flag
    • The “-p” flag
    • The “-ndl” flag
    • Just do a “mu file.m” ; then “Make”
    • Then “file –ndl [if needed] –p” -- that will pretty much do
Asg2, Problem 2

• Simple practice of Floyd-Hoare Logic on a small example (more coming later; but this much helps you lock in your knowledge of invariants)
  • 2a: Urn with colored balls
  • 2b: gcd iteration scheme
Asg2, Problem 3

• Boat, wolf, goat, cabbage
• Man with boat always, so model man as boat
• Only one object/being can be moved at most
• Zero objects/beings can be moved (i.e. boat can travel w/o anything... of course man always on boat)
• All being start on left bank
• They must end up on right bank
• Never leave W and G unattended (safe if man around)
• Never leave G and C unattended (safe, if man around)
• Formulate the invariant to be preserved at any time
• Write a transition system in Murphi that maintains the invariant
• Use the power of counter-examples to compute a sequence of moves that transports the objects
Summary of our discussion of locking protocols (from Herlihy and Shavit’s book, “The Art of Multiprocessor Programming”)

• First study a mutex protocol that will deadlock upon interleaving
• Then study a mutex protocol that will deadlock when there is NO interleaving!
• Peterson’s brilliant solution: combine the ideas in these two!
• Peterson’s N process solution
• Now study paper/pencil proofs vs. Model Checking proofs
Lamport’s Bakery Protocol

• Only known (to me) mutual exclusion protocol which has two neat properties
  – Crash of a node does not hang the protocol
  – If a read/write overlap returns garbage for Read, then too no sweat!
  – Read Lamport’s CACM paper + follow-ups from Leslie Lamport’s webpage
  – You will model these protocols plus also follow Lamport’s paper/pencil proofs
Now for the details

• Present the “WP” rule for assignment
  – Reference: Gordon’s book online
• Show on XOR-based swap
• Present WP rule for if-then
• Present proof for binsrch
• Reinforce loop invariant
• Connect back to exit condition
Now for the details

• Then present locking protocols from Herlihy/Shavit
• Present paper/pencil proofs
• Go thru Locking1 class, Locking2 class, Peterson, N-Peterson, r-bounded overtaking, and Lamport’s Bakery which is FIFO
• Model Check using Murphi (asg2)
WP for assignment

• \( x := E \)

• Suppose \( P \) is true after this

• What is the weakest (most general) statement we can make before this?

• That is the wp!

• \( \text{Wp( "x:=E", "Post:P" )} \)

• Denoted \( \{ \ ? \ \} \ x := E \ \{ \ P \} \)
WP for assignment

• Derive for $x := 5$

• Derive for $x := x + 1$

• Derive for $x := y$

• Why wp? Why not SP? Later!
XOR swap proof

- Will show XOR as $\neq$ because that is what it is
- $a := a \neq b$
- $b := a \neq b$
- $a := a \neq b$
- Let $a$ and $b$ be either single bits or bit-vectors
WP rule for if-then

- \{ ? \} if (C) then S \{ P \}
- C => wp(S, P)
- \{ ? \} if (C) then S1 else S2 \{ P \}
- C => wp(S1, P) \(\land\) !C => wp(S2, P)
WP based proof for BinSrch

• Draw flow-chart
• Write Precondition
• Write Postcondition
• Write Loop Invariant
  – Hint : one can often make it from Postcondition
• Go thru proof
• This is what’s due next week by the way of Asg2
Locking Protocols

• Read Lamport’s paper (available from his website)
• Just browse his collected works! What a voluminous and impactful contribution!!
• The paper to read is “A New Solution of Dijkstra’s Concurrent Programming Problem”
• Read all the follow-up papers – also on his website
• Read Herlihy / Shavit’s notes that I pointed to in Lecture 3 (it is from his “Art of Multiprocessor Programming”) – a link will be kept on the class site
• We will go thru all the proofs
• Meanwhile you can pretty much “code up” Lamport’s protocol from his CACM 1974 paper
• Later you can understand how the model checker “proves” the invariant
• While a model checker does make things far easier than paper/pencil proofs
  – One HAS to have paper/pencil proofs
  – Especially sketches of correctness arguments for new locking protocols being designed
  – A model checker is then one’s assistant that checks one’s proofs and helps patch up mistakes
  – Then one of course has to attempt a general proof (for all N) using a Theorem prover
  – One can also seek Cutoff Bounds : “Proved for k => Proved for all n > k ? “