

# CS 6110 – Formal Methods – Spring 2011

February 12, 2011

**Assignment 2, Handed out: January 20 – NEW due dates noted below**

## New Due Dates

- Question 1 : Due Jan 24th monday midnight
- Questions 2 and 5: Jan 26th wednesday midnight
- Question 6: Due Feb 2nd (Wed) midnight.
- Remaining : January 31st midnight

1. **(5 points)** Read Chapter 21 from my book. Also read the DDHY92 paper linked at <http://www.eng.utah.edu/~cs6110/week2/DDHY92.pdf>.

Finally, Skim through `many-promela-exs` that will be the subject of the lecture on Jan 25th. These are linked at

<http://www.eng.utah.edu/~cs6110/week3/many-promela-exs.pdf>.

Write a one-page summary covering these readings.

Answer: Their writing.

Points:

You can pretty much give all the points if a decent-looking summary exists. Spend no more than 30 second looking over each student's work. I expect the value to derive from their own diligence. If the summaries look hastily put together or are inadequate, of course knock off a few points.

2. **(5 points)** Develop your own example to understand invariants and inductive invariants geometrically, as was presented in class. Draw these invariants out as sets of points as I showed in class.

Answer:

I gave the transition systems in my slides. They were as follows.

Initial state: 0

Rules: allow these moves.

0 -> 1 -> 2 -> 0

3 → 4 → 5 → 3

6 → 7 → 8 → 6

An invariant: Anything that's true over executions. So {0,1,2} is an invariant, and so is {0,1,2,3}

{0,1,2,3} is not inductive, as it is not closed under the transition system.

{0,1,2,3,4,5} is inductive, as it is closed under the transition system.

Points: Any example of the above nature would constitute an acceptable answer.

3. (10 points) There is a large urn with a fixed but arbitrary amount of black and white balls (“initial state”). There is a supply of extra balls whenever you want. Someone executes this loop:

- If there are fewer than two balls in the urn, terminate.
- Takes two balls out of the urn.
- Now, if both of the same color, throw back a black (could be the one you took out; could be from the extra supply).
- Else throw back a white.

Answer these questions briefly (a sentence in most cases):

- Will this procedure terminate? Answer: Yes, since the number of items decrease. In general, one looks for a well-founded relation.  
See [http://en.wikipedia.org/wiki/Loop\\_variant](http://en.wikipedia.org/wiki/Loop_variant) for details on how termination is argued for while loops.
- For all initial states with at least two balls, what is the final state of the urn?  
Answer: Treating white as 1 and black as 0, the answer is the result of applying the “odd parity” function to the urn (are there an odd number of bits set?).  
[http://en.wikipedia.org/wiki/Parity\\_bit](http://en.wikipedia.org/wiki/Parity_bit).  
Recall that parity is what XOR calculates.
- Write this as a loop. Write down an inductive invariant for the loop. Show that it is inductive.  
Answer: The inductive invariant is that the parity is preserved. Thus XOR applied to the urn before and after yields the same answer.
- What post-condition would you assign for this loop?  
Answer: Post condition is = XOR(initial state). This means: XOR of all the balls in the initial urn. Thus, if the XOR of all the balls in the initial urn is 1, there will be a single white ball left. Else there will be a single black ball left.
- Show that the post-condition falls out (follows from) of the loop invariant when the loop terminates.  
Answer: It does!

The loop is described as follows:

```
// Initial Urn is given in urn0
// randball(urn) denotes a random member of an urn
// xor(urn) is the XOR of the contents of the urn
//
urn := urn0;
while (|urn| > 1) do {
  << LI : XOR(urn) = XOR(urn0) >>
  L1: b1 := randball(urn);
  L2: urn := urn \ { b1 };
  L3: b2 := randball(urn);
  L4: urn := urn \ { b2 };
  L5: urn := urn Union XOR(b1,b2);
}
// If the initial urn had one ball at least, then
// the final urn is a singleton set of color denoted by
// XOR(urn0)
//
<< Post: if |urn0| > 1 then urn = { XOR(urn0) } >>
```

Why is this an LI:

-----  
Inferring the WP of LI wrt the loop body, we get

LI:  $X(u) = X(u_0)$

Before L5:  $X(u \text{ Union } X(b_1, b_2)) = X(u_0)$

Before L4:  $X(u \setminus \{b_2\} \text{ Union } X(b_1, b_2)) = X(u_0)$

Before L3:  $X(u \setminus \{\text{rand}(u)\} \text{ Union } X(b_1, \text{rand}(u))) = X(u_0)$

...

Before L1:  $X(u \setminus \{\text{rand}_2(u)\} \text{ Union } X(\text{rand}_2(u))) = X(u_0)$

This means : remove 2 rand balls from u; then add back XOR of these two.  
This is the same as the XOR of the whole.

Why it implies Postcondition:

-----  
Upon exit, urn has 0 or 1 ball, and  $x(u)=x(u_0)$  captures this,  
because if  $|u| = 1$ , then that ball's color is equal to  $x(u_0)$ .

4. (10 points) An iterative GCD is roughly as follows:

```
// x0 and y0 are the initial values of x, y
<< To guarantee termination, PRE: x0 <> 0 and y0 <> 0 >>
//
function gcd(x : int, y : int): int;
begin
  while (x!=y) do
  << LI : gcd(x,y) = gcd(x0,y0) >>
  L1: if (x>y) then x := x-y;
  L2: else // (x<y)
  L3:   y := y-x;
  endwhile;
  return x;
<< Post: x = gcd(x0,y0) >>
end;
```

This LI works because of this:

LI:  $\text{gcd}(x,y) = \text{gcd}(x_0,y_0)$

Path back thru L3, L1: the verification condition is

$$\text{gcd}(x,y)=\text{gcd}(x_0,y_0) \text{ IMPLIES } [ (x \leq y) \text{ IMPLIES } (\text{gcd}(x,y-x) = \text{gcd}(x_0,y_0)) ]$$

i.e.

$$\text{gcd}(x,y)=\text{gcd}(x_0,y_0) \wedge (x \leq y) \text{ IMPLIES } (\text{gcd}(x,y-x) = \text{gcd}(x_0,y_0))$$

follows from the defn. of GCD.

Path back thru L2, L1:

Similar.

Annotate this pseudocode with suitable pre- and post conditions. Then, write a loop invariant that is inductive for the loop. Show that your loop invariant is “powerful enough” to infer the answer assertion (postcondition). Show your work.

Answer: The annotated program is above. The loop invariant (LI) can be established by computing the WP of the LI wrt the loop body, and showing that  $LI =_i wp(LI, \text{loop-body})$ .

5. (25 points) The rules for the man, wolf, goat, cabbage problem were described in the slides. Formulate this problem as a Murphi transition system and solve it (again as described in the slides).

Answer: See my solution posted.

6. (**45 points**) Model Lamport's Bakery Protocol in Murphi for a fixed number (say, 4) of processes. See if you can employ scalar sets for some of the indices. Come up with the necessary invariants to prove mutual exclusion. Show that these invariants hold. Develop a readable model that generates a low number of states. Present a `typescript` session of your work and a brief (one page) writeup of your observations.
7. Extra (say **20 points**): See if you can establish the "extra" features of this protocol using Murphi. Lamport mentions two such properties: one being that a node failure still preserves the overall workings; the second being that a concurrent read and write of a location may return "garbage" values and yet the protocol will work. I don't know how long these extra credit parts will take, hence the points are open-ended for now. Bound your work.