EE219C HW3: Model Checking

Vighnesh Iyer

1 Interrupt Driven Program

(a) Describe the properties in the `Sys` module in English. Note the composition of `main` and `ISR` within the module `Sys` is incorrectly done (you will need to fix it later).

- `invariant main_ISR_mutex: (M_enable != I_enable);`
  Only the main module or the ISR module can be active at a given timestep.
- `property[LTL] one_step_ISR_return: G(return_ISR ==> X(!return_ISR));`
  If `return_ISR` is true in a given timestep, it should be false in the next timestep for any trace of the system. This should ensure that the ISR module can't be advanced through in less than 1 timestep.
- `property[LTL] main_after_ISR: G((I_enable && X(M_enable)) ==> return_ISR);`
  If on a particular timestep `I_enable` is true and on the next time step `M_enable` will be true, then `return_ISR` should also be true on this timestep.
- `property[LTL] ISR_after_main: G((M_enable && X(I_enable)) ==> (assert_intr));`
  The 'dual' of the previous property: if we are in the main module execution, and we are going to move into the ISR module next, the interrupt from the environment must also have been asserted right now.

(b) Run the file and interpret results.

The invariant passes a 20 cycle unrolling because `mode` isn't being updated, and `M_enable` and `I_enable` are mutually exclusive conditions.

The latter 2 LTL properties fail to check because `mode` is being set arbitrarily and the counter-example traces contain transitions between the main and ISR modules that don't match the havoc behavior of `mode` being arbitrarily set by the solver.

(c) Fix to correctly compose main and ISR in `Sys`, and eliminate the above CEXs. Change `update_mode`.

```
procedure update_mode() modifies mode; {
  case
    mode == main_t: {
      if (assert_intr) {
        mode = ISR_t;
      } else {
        mode = main_t;
      }
    }
    mode == ISR_t: {
      if (return_ISR) {
        mode = main_t;
      } else {
        mode = ISR_t;
      }
    }
  esac
}
```

Now all the assertions pass.

(d) Correctly compose `Sys` and `Env` (should it be async composition with interleaving semantics?). Explain why the `consec_main_pc_values` property may fail when the composition is corrected.

I modified the `init` and `next` blocks as such:

```
init { havoc turn; }

next {
  if (turn) {
    next (Sys_i);
  } else {
    next (Env_i);
  }
  if (assert_intr) {
    turn' = true;
  } else {
    havoc turn;
  }
}
```

I don't think it's acceptable for `Sys` and `Env` to be composed with async composition since once `Env` raises the interrupt line, it is required that `Sys` executes next to properly receive and interrupt. I think the decision as to which module to execute should be arbitrary, except with this one constraint.

With this modification the `consec_main_pc_values` property fails because the solver makes `turn` false all the time and prevents forward movement in `Sys` which causes the property to fail in limited time.