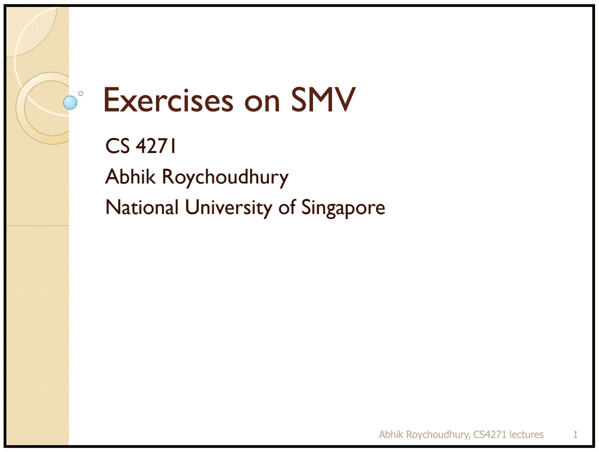
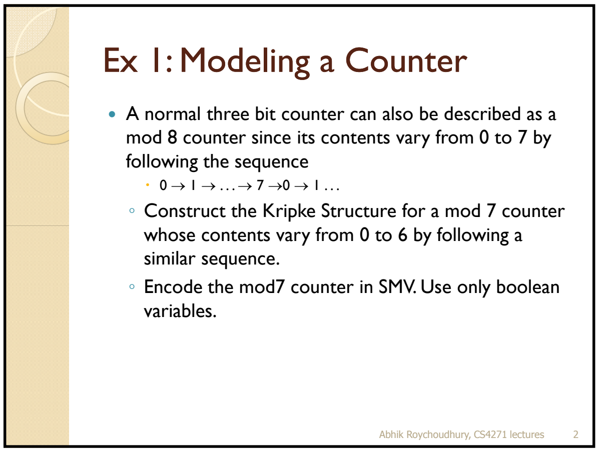


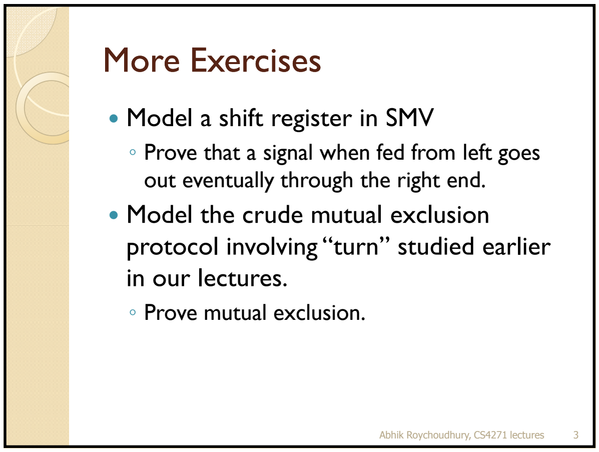
Exercises on SMV

CS 4271
Abhik Roychoudhury
National University of Singapore



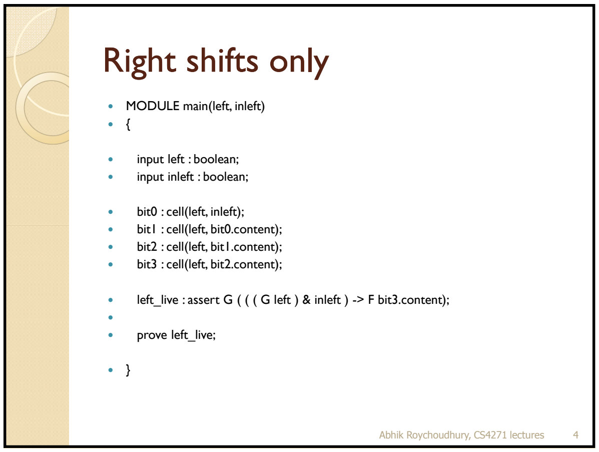
Ex 1: Modeling a Counter

- A normal three bit counter can also be described as a mod 8 counter since its contents vary from 0 to 7 by following the sequence
    - 0 → 1 → . . . → 7 →0 → 1 . . .
  - Construct the Kripke Structure for a mod 7 counter whose contents vary from 0 to 6 by following a similar sequence.
  - Encode the mod7 counter in SMV. Use only boolean variables.



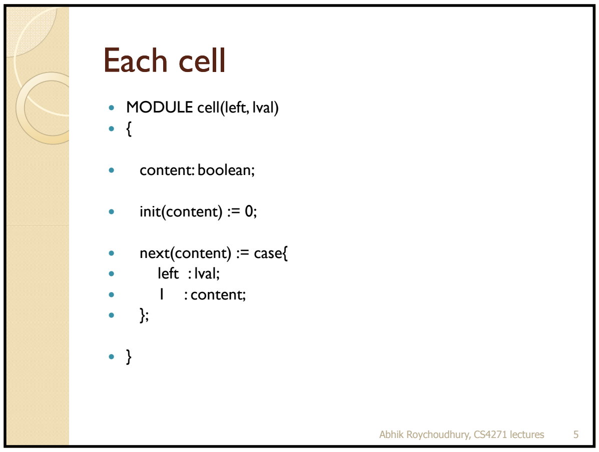
More Exercises

- Model a shift register in SMV
  - Prove that a signal when fed from left goes out eventually through the right end.
- Model the crude mutual exclusion protocol involving "turn" studied earlier in our lectures.
  - Prove mutual exclusion.



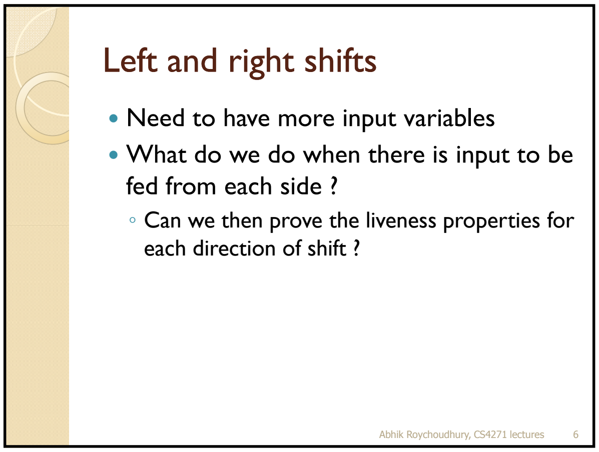
Right shifts only

```
•  MODULE main(left, inleft)
•  {

•     input left : boolean;
•     input inleft : boolean;

•     bit0 : cell(left, inleft);
•     bit1  : cell(left, bit0.content);
•     bit2 : cell(left, bit1.content);
•     bit3 : cell(left, bit2.content);

•     left_live : assert G ( ( ( G left ) & inleft ) -> F bit3.content);
•
•     prove left_live;

•  }
```



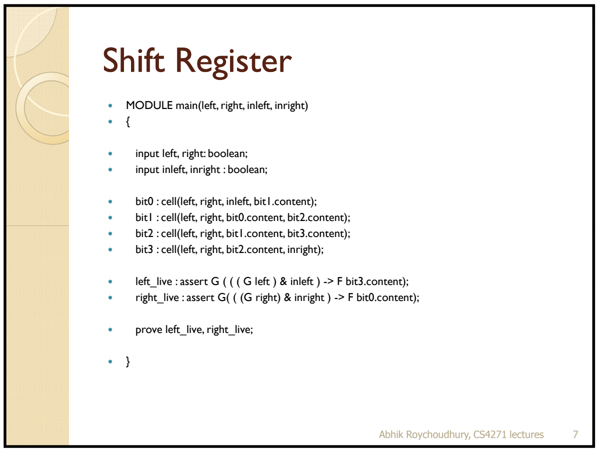
Each cell

```
•  MODULE cell(left, lval)
•  {

•     content: boolean;

•     init(content) := 0;

•     next(content) := case{
•         left  : lval;
•         1     : content;
•     };

•  }
```



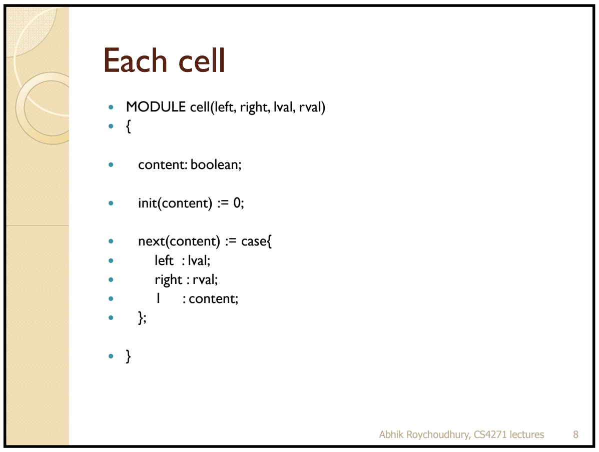
Left and right shifts

- Need to have more input variables
- What do we do when there is input to be fed from each side ?
  - Can we then prove the liveness properties for each direction of shift ?



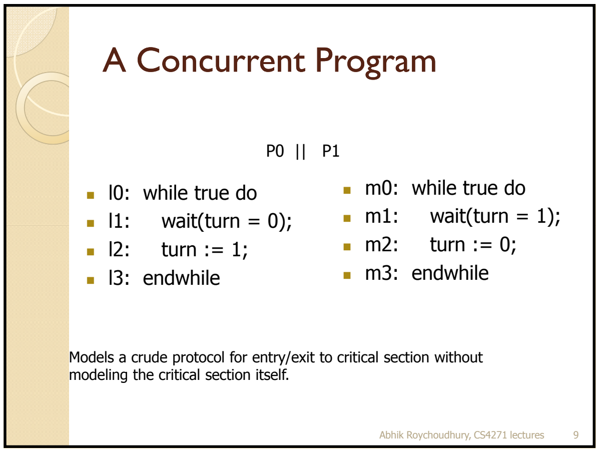
Shift Register

```
MODULE main(left, right, inleft, inright)
{

   input left, right: boolean;
   input inleft, inright : boolean;

   bit0 : cell(left, right, inleft, bit1.content);
   bit1 : cell(left, right, bit0.content, bit2.content);
   bit2 : cell(left, right, bit1.content, bit3.content);
   bit3 : cell(left, right, bit2.content, inright);

   left_live : assert G ( ( ( G left ) & inleft ) -> F bit3.content);
   right_live : assert G( ( (G right) & inright ) -> F bit0.content);

   prove left_live, right_live;

}
```



Each cell

```
MODULE cell(left, right, lval, rval)
{

   content: boolean;

   init(content) := 0;

   next(content) := case{
       left  : lval;
       right : rval;
       1     : content;
   };

}
```



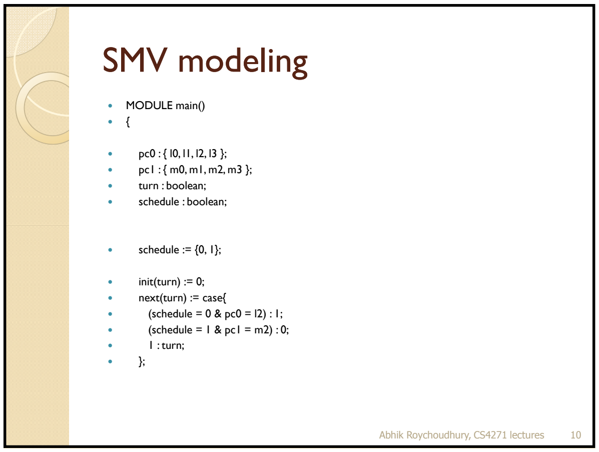
A Concurrent Program

P0 || P1

- l0: while true do
- l1: wait(turn = 0);
- l2: turn := 1;
- l3: endwhile

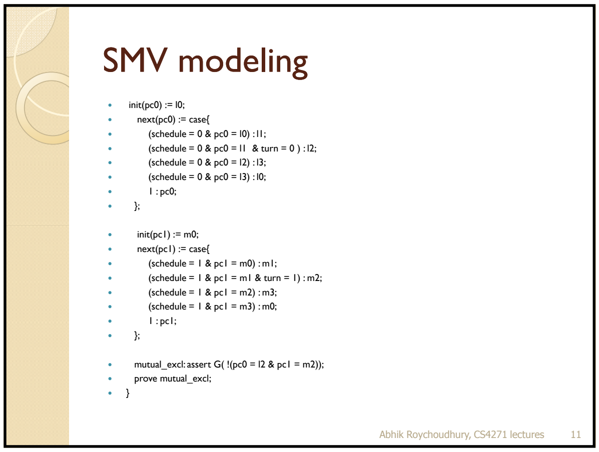
- m0: while true do
- m1: wait(turn = 1);
- m2: turn := 0;
- m3: endwhile

Models a crude protocol for entry/exit to critical section without modeling the critical section itself.



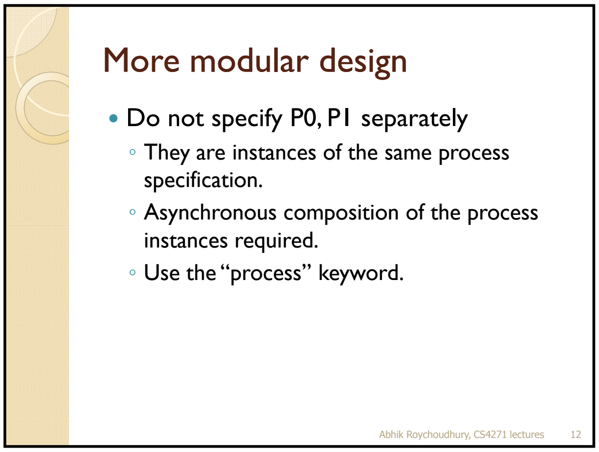
SMV modeling

```
MODULE main()
{

   pc0 : { l0, l1, l2, l3 };
   pc1 : { m0, m1, m2, m3 };
   turn : boolean;
   schedule : boolean;


   schedule := {0, 1};

   init(turn) := 0;
   next(turn) := case{
      (schedule = 0 & pc0 = l2) : 1;
      (schedule = 1 & pc1 = m2) : 0;
      1 : turn;
   };
```



SMV modeling

```
   init(pc0) := l0;
     next(pc0) := case{
        (schedule = 0 & pc0 = l0) : l1;
        (schedule = 0 & pc0 = l1  & turn = 0 ) : l2;
        (schedule = 0 & pc0 = l2) : l3;
        (schedule = 0 & pc0 = l3) : l0;
        1 : pc0;
    };

     init(pc1) := m0;
     next(pc1) := case{
        (schedule = 1 & pc1 = m0) : m1;
        (schedule = 1 & pc1 = m1 & turn = 1) : m2;
        (schedule = 1 & pc1 = m2) : m3;
        (schedule = 1 & pc1 = m3) : m0;
        1 : pc1;
    };

    mutual_excl: assert G( !(pc0 = l2 & pc1 = m2));
    prove mutual_excl;
 }
```



More modular design

- Do not specify P0, P1 separately
  - They are instances of the same process specification.
  - Asynchronous composition of the process instances required.
  - Use the "process" keyword.