Using Cadence SMV
to verify temporal properties
of finite-state machines

15-817: Intro to Model Checking
October 9, 2008
Review of Temporal Logic

\( a \) “\( a \) is true now”
\( X a \) “\( a \) is true in the neXt state”
\( F a \) “\( a \) will be true sometime in the Future”
\( G a \) “\( a \) will be Globally true in the future”
\( a \lor b \) “\( a \) will hold true Until \( b \) becomes true”

- Potential source of confusion: “in the future” ≠ “in a future state”. I.e., the remaining time of the current state is part of the future.
- So, “\( F a \)” is satisfied if \( a \) is true in the current state.
- And, “\( G a \)” is falsified if \( a \) is false in the current state.

Path Quantifiers:
- “\( E a \)” : There exists a path (from the current state) on which \( a \) is true.
- “\( A a \)” : Along all paths (from the current state), \( a \) is true.
EF $g$ : "it is possible that $g$ will be true"
AF $g$ : “$g$ will necessarily be true sometime in the future”
$AG \ g : \ "g \ is \ an \ invariant"$
EG g : “on some path, g is always true”
Microwave Oven Example

• Suppose we are designing a controller for a microwave oven.
  • The heating element should be off if the door is open. (Else the user gets cooked!)
  • If the user presses the Start button when the door is open, then enter an error mode. Leave the error mode when the user presses the Reset button.
MODULE main
VAR
    start, reset, closed : boolean;
    error, heat : boolean;

ASSIGN
    init(error) := 0;
    init(heat) := 0;

    next(error) :=
        case
            start & ~closed : 1;
            closed & reset : 0;
            1 : error;
        esac;

    next(heat) :=
        case
            start & closed : 1;
            ~closed : 0;
            1 : heat;
        esac;

SPEC AG (!closed -> AX (!heat))
SPEC EF (heat)

Note: The SMV code is not created from the state diagram. (Indeed, for non-trivial systems, one cannot even explicitly create the state diagram, because it is exponential in the number of bits of state.) Rather, the SMV code and the state diagram are both created from the underlying logic of the system (e.g., a digital circuit in hardware verification).
Microwave Example in Cadence SMV


- Cadence SMV and NuSMV and two model checkers.
- Both support an older-style syntax that is used in this presentation.
- Cadence also supports a new-style syntax that is more expressive.
- Cadence can run on both Windows and Linux.
simple01.smv

MODULE main
VAR
  request : boolean;
  state : \{ready, busy\};
ASSIGN
  init(state) := ready;
  next(state) :=
    case
      state = ready & request : busy;
      1 : \{ready, busy\};
    esac;

SPEC AG(request -> AF state = busy)

The state variable is declared to take one of two enumerated values: ready and busy.

Note:
foo :=
case
  cond1 : expr1
  cond2 : expr2
  cond3 : expr3
esac
means the same as
if (cond1) \{foo = expr1\}
else if (cond2) \{foo = expr2\}
else if (cond3) \{foo = expr3\}
Since 1 is true, it is used to specify the default case.
Three-Bit Counter; Modules

MODULE counter_cell(carry_in)
VAR
  value : boolean;
ASSIGN
  init(value) := 0;
  next(value) := value + carry_in mod 2;
DEFINE
  carry_out := value & carry_in;

MODULE main
VAR
  bit0 : counter_cell(1);
  bit1 : counter_cell(bit0.carry_out);
  bit2 : counter_cell(bit1.carry_out);

SPEC AG AF bit2.carry_out
Peterson’s Mutual-Exclusion Algorithm (broken version)

```plaintext
MODULE agent(req_self, req_other, turn, num)
VAR
  pc : {L0, L1, L3, L4 };

ASSIGN
  init(req_self):= 0;
  init(pc):= L0;

next(pc):=
  case
    pc = L0 : { L0, L1 };
    pc = L1 : L3;
    pc = L3 & ( req_other ) : L3;
    pc = L3 : L4;
    pc = L4 : L0;
  esac;

next(req_self):=
  case
    pc = L1 : 1;
    pc = L4 : 0;
    1 : req_self;
  esac;

DEFINE
  request := pc = L1;
  critical := pc = L4;

FAIRNESS
  running
```

```plaintext
module main
VAR
  turn : boolean;
  req0 : boolean;
  req1 : boolean;

agent0 : process agent(req0, req1, turn, 0);
agent1 : process agent(req1, req0, turn, 1);

safe: SPEC AG( !agent0.critical | !agent1.critical );
fairness0: SPEC AG( agent0.request -> EF agent0.critical );
fairness1: SPEC AG( agent1.request -> EF agent1.critical );
```

agent0:
L0: Idle;
L1: req0 = 1; //request to enter critical section
L3: while (req1 == 1) {};
L4: req0 = 0; //clear the request

agent1:
L0: Idle;
L1: req1 = 1; //request to enter critical section
L3: while (req0 == 1) {};
L4: req1 = 0; //clear the request

```

```
MODULE agent(req_self, req_other, turn, num)
VAR
  pc : {L0, L1, L2, L3, L4 };
ASSIGN
  init(req_self):= 0;
  init(pc):= L0;

  next(pc):=
    case
      pc = L0
        : { L0, L1 };
      pc = L1
        : L2;
      pc = L2
        : L3;
      pc = L3 & (turn = !num) & req_other: L3;
      pc = L3
        : L4;
      pc = L4
        : L0;
    esac;

  next(req_self):=
    case
      pc = L1 : 1;
      pc = L4 : 0;
      1   : req_self;
    esac;

  next(turn):=
    case
      pc = L2 : !num;
      1   : turn;
    esac;

DEFINE
  request:= pc = L1;
  critical:= pc = L4;
FAIRNESS
  running

peterson's Mutex (Fixed)

agent0:
L0: Idle;
L1: req0 = 1; //request to enter critical section
L2: turn = 1; //give priority to the other process
L3: while (turn == 1 && req1 == 1) {};
L4: req0 = 0; //clear the request

agent1:
L0: Idle;
L1: req1 = 1; //request to enter critical section
L2: turn = 0; //give priority to the other process
L3: while (turn == 0 && req0 == 1) {};
L4: req1 = 0; //clear the request

MODULE main
VAR
  turn : boolean;
  req0 : boolean;
  req1 : boolean;

  agent0 : process agent(req0, req1, turn, 0);
  agent1 : process agent(req1, req0, turn, 1);

  safe: SPEC AG( !agent0.critical | !agent1.critical );
  fairness0: SPEC AG( agent0.request -> EF agent0.critical );
  fairness1: SPEC AG( agent1.request -> EF agent1.critical );