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

15-817: Intro to Model Checking
April 6, 2011

Will Klieber
Review of Temporal Logic

\( a \) \hspace{1cm} “a is true now”

\( X a \) \hspace{1cm} “a is true in the next state”

\( F a \) \hspace{1cm} “a will be true sometime in the Future”

\( G a \) \hspace{1cm} “a will be Globally true in the future”

\( a U b \) \hspace{1cm} “a will hold true Until b becomes true”

Remember \((\pi \models f)\) implies \((\pi \models F f)\).

Path Quantifiers:

• “E f”: There exists a path (from the current state) on which \( f \) is true.
• “A f”: Along all paths (from the current state), \( f \) 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”
SMV

- Input:
  - Model (specified in the SMV system-description language)
  - Properties to verify (in CTL or LTL)

- Output:
  - "All properties are true"
  - Counterexample trace

- SMV was originally developed by Ken McMillan at CMU (1992)

- Two versions:
  - Cadence SMV
  - NuSMV

- Both support an older-style syntax that is used in this presentation.
- Cadence SMV also supports a new-style syntax that is more expressive.
- Cadence SMV can run on both Windows and Linux.
**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.
Microwave Oven Example

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:
foo :=
case
    cond1 : expr1
    cond2 : expr2
    cond3 : expr3
esac
means
if (cond1) {foo = expr1}
else if (cond2) {foo = expr2}
else if (cond3) {foo = expr3}

Integer 1 is interpreted as boolean true.
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: SMV code is not normally created from a state diagram. (Indeed, for many systems, the state space is too large to explicitly create the state diagram.) Rather, the SMV code is created from the underlying logic of the system (e.g., a digital circuit in hardware verification).
Microwave Example in Cadence SMV

http://www.kenmcmil.com/smv.html
MODULE main
VAR
request : boolean;
state   : {ready,busy};
ASSIGN
init(state) := ready;
next(state) :=
    case
        state = ready & request : busy;
        1 : {ready,busy}; // non-det choice
    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
if (cond1) {foo = expr1}
else if (cond2) {foo = expr2}
else if (cond3) {foo = expr3}

Integer 1 is interpreted as boolean true.

Exercise: Draw the corresponding Kripke structure.
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)

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;
      l : req_self;
    esac;
DEFINE
    request:= pc = L1;
    critical:= pc = L4;
FAIRNESS
    running

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

agent1:
L0: Idle;
L1: req1 = 1; //request to enter critical section
L3: while (req0 == 1) {};
/* critical region */
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 );
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

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 );