Modeling systems and specifying temporal properties in NuSMV 2.5

Metodi Formali per il Software
Laurea Magistrale in Informatica, AA 2013-2014
Prof.ssa Anna Labella

Esercitatore: Federico Mari
mari@di.uniroma1.it

Model Checking Laboratory Group
http://mclab.di.uniroma1.it/

Dipartimento di Informatica
Sapienza Università di Roma

14 aprile 2014
Models in NuSMV for:

- Mutual exclusion
- The ferryman
- The alternating bit protocol
Mutual exclusion
The ferryman
The alternating bit protocol
Shared resources
Mutual exclusion

Description of real system

- $N > 1$ processes
- 1 shared resource
- Each process wants to access the shared resource: critical session
- Each critical session must be as small as possible
- Only one process can be in its critical session at a time

Problem
Find a protocol for determining which process is allowed to enter its critical section at which time
Mutual exclusion

Description of properties

Safety
Only one process is in its critical section at any time

Liveness
Whenever any process requests to enter its critical section, it will eventually be permitted to do so

Non-blocking (not expressible in LTL)
A process can always request to enter its critical section

No strict sequencing
Processes need not enter their critical section in strict sequence
Mutual exclusion

Modelling and Specification

Safety
\[ G \neg(c_1 \land c_2) \]

Liveness
\[ G \left( t_1 \rightarrow F \ c_1 \right) \]

No strict sequencing (negation)
\[ G \left( c_1 \rightarrow c_1 W (\neg c_1 \land \neg c_1 W \ c_2) \right) \]

Liveness false

⇒ refine the model by splitting \( s_3 \) to remember who asked to enter its critical region first
Mutual exclusion

NuSMV model

Single process

```
MODULE prc(other_st, turn, myturn)
  VAR
    st : {n, t, c};
  ASSIGN
    init(st) := n;
    next(st) :=
      case
        (st = n) : {t, n};
        (st = t) & (other_st = n) : c;
        (st = t) & (other_st = t) &
          (turn = myturn) : c;
        (st = c) : {c, n};
        TRUE : st;
      esac;
    next(turn) :=
      case
        turn = myturn & st = c : !turn;
        TRUE : turn;
      esac;
    FAIRNESS running;  -- run infinitely often
    FAIRNESS !(st = c);  -- not always critical
```
NuSMV model

Main module

1 MODULE main
2 VAR
3 pr1 : process prc(pr2.st, turn, FALSE);
4 pr2 : process prc(pr1.st, turn, TRUE);
5 turn : boolean;
6 ASSIGN
7 init(turn) := FALSE;
8 -- Safety
9 LTLSPEC
10 G !(pr1.st = c & pr2.st = c);
11 -- Liveness
12 LTLSPEC
13 G (pr1.st = t -> F pr1.st = c);
14 -- No strict sequencing (negation)
15 LTLSPEC
16 G (pr1.st = c ->
17   (G pr1.st = c |
18    (pr1.st = c U (!pr1.st = c) &
19     G !(pr1.st = c) |
20     ((!pr1.st = c)) U pr2.st = c)
21   ) )
22 )
Running NuSMV

Output

-- specification G !(pr1.st = c & pr2.st = c) is true
-- specification G (pr1.st = t -> F pr1.st = c) is true
-- specification G (pr1.st = c -> ( G pr1.st = c | (pr1.st = c U (((!(pr1.st = c) & G !(pr1.st = c)) | (!(pr1.st = c) U pr2.st = c)))))
is false
-- as demonstrated by the following execution sequence
Trace Description: LTL Counterexample
Trace Type: Counterexample
-> State: 1.1 <-
  pr1.st = n
  pr2.st = n
  turn = FALSE
...

Try to run the model with NuSMV
Mutual exclusion

The ferryman

The alternating bit protocol
The ferryman
Modelling of real system

Automata composition

Variable f

```
Variable g
f = 1 \land carry' = "G" \land f' = 0
```

```
Variable c
f = 1 \land carry' = "C" \land f' = 0
```

```
Variable w
f = 1 \land carry' = "W" \land f' = 0
```

Variable carry

```
Variable g
f = 0 \land carry' = "G" \land f' = 1
```

```
Variable c
f = 0 \land carry' = "C" \land f' = 1
```

```
Variable w
f = 0 \land carry' = "W" \land f' = 1
```
The ferryman

Specification of properties

Strategy to avoid unsafe situation
If the goat and the cabbage, or the wolf and the goat, are on the same river bank then the goat is with the ferryman

Planning problem goal
The goat, the cabbage, and the wolf are on the other river bank (value 1)

Specification (LTL)
$(((g = c \lor w = g) \rightarrow g = f) \text{ U } (f \land g \land c \land w))$

Specification for planning problem (LTL)
$\neg(((g = c \lor w = g) \rightarrow g = f) \text{ U } (f \land g \land c \land w))$

$\Rightarrow$ Finding a counterexample yields to a plan
The ferryman

NuSMV model
Synchronous model

```plaintext
MODULE main

VAR
  ferryman : boolean;
  goat : boolean;
  cabbage : boolean;
  wolf : boolean;
  carry : {g, c, w, 0};

ASSIGN
  init(ferryman) := FALSE;
  init(goat) := FALSE;
  init(cabbage) := FALSE;
  init(wolf) := FALSE;
  init(carry) := 0;

  next(ferryman) := {FALSE, TRUE};

-- to be continued
```
NuSMV model

Synchronous model

```plaintext
-- continuing
next(carry) :=
case
  ferryman = goat : g;
  TRUE : 0;
esac
union
case
  ferryman = cabbage : c;
  TRUE : 0;
esac
union
case
  ferryman = wolf : w;
  TRUE : 0;
esac
union 0;

next(goat) := case
  ferryman = goat & next(carry) = g : next(ferryman);
  TRUE : goat;
esac;

-- to be continued
```
The ferryman

NuSMV model
Synchronous model

--- continuing

next(cabbage) := case
  ferryman = cabbage & next(carry) = c : next(ferryman);
  TRUE : cabbage;
esac;

next(wolf) := case
  ferryman = wolf & next(carry) = w : next(ferryman);
  TRUE : wolf;
esac;

LTLSPEC !(( (goat = cabbage | goat = wolf)
  -> goat=ferryman)
  U (cabbage & goat & wolf & ferryman))
The ferryman

Running NuSMV

Output

```
-- specification !(((goat = cabbage | goat = wolf) -> goat = ferryman) 
    U (((cabbage & goat) & wolf) & ferryman)) is false
-- as demonstrated by the following execution sequence
Trace Description: LTL Counterexample
Trace Type: Counterexample
-- Loop starts here
--> State: 1.1 <-
    ferryman = FALSE
    goat = FALSE
    cabbage = FALSE
    wolf = FALSE
    carry = 0
```

Try to run the model with NuSMV
Mutual exclusion

The alternating bit protocol
The alternating bit protocol: motivations

Alice

Bob

Insecure channel

I love you

So do I

What?

WOW!!

Yeah!!

Charlie
The alternating bit protocol

Description of real system

Alice ▶ Sender
Bob ▶ Receiver
Insecure channel ▶ Message channel
Ack channel ▶ Ack channel

Figure shows two instances of Alternating Bit Protocol (ABP)
1. Sender: Alice, Receiver: Bob
2. Sender: Bob, Receiver: Alice

Sender sends again the same message until he receives the corresponding ack
Receiver sends again the same ack until he receives a message with expected bit
Modelling of real system

The whole protocol

Note
Variable `forget` models insecure channel (non-deterministic)
The alternating bit protocol

Specification of properties

Safety (LTL)
\[ G \ (\text{Sender} \cdot \text{st} = \text{sent} \land \text{Sender} \cdot \text{message}1 = 1 \rightarrow \text{MsgCh} \cdot \text{output}1 = 1) \]

Liveness (LTL)
\[ G \ F \ (\text{Sender} \cdot \text{st} = \text{sent}) \]
\[ G \ F \ (\text{Receiver} \cdot \text{st} = \text{received}) \]
NuSMV model

The sender

```
MODULE sender(ack)

VAR
  st : {sending, sent};
  message1 : boolean;
  message2 : boolean;

ASSIGN
  init(st) := sending;
  next(st) :=
    case
      ack = message2 & !(st = sent) : sent;
      TRUE : sending;
    esac;

  next(message1) :=
    case
      st = sent : {FALSE, TRUE};
      TRUE : message1;
    esac;

  next(message2) :=
    case
      st = sent : !message2;
      TRUE : message2;
    esac;

FAIRNESS running

LTLSPEC G F st = sent
```
The alternating bit protocol

NuSMV model

The receiver

```
MODULE receiver (message1, message2)
VAR
    st : {receiving, received};
    ack : boolean;
    expected : boolean;
ASSIGN
    init (st) := receiving;
    next (st) :=
        case
            message2 = expected & !(st = received) : received;
            TRUE : receiving;
        esac;
    next (ack) :=
        case
            st = received : message2;
            TRUE : ack;
        esac;
    next (expected) :=
        case
            st = received : !expected;
            TRUE : expected;
        esac;
    FAIRNESS running;
    LTLSPEC G F st = received;
```
The alternating bit protocol

NuSMV model
The one bit channel (ack)

```c
MODULE one_bit_channel(input)
VAR
  output : boolean;
  forget : boolean;
ASSIGN
  next(output) := case
    forget : output;
    TRUE   : input;
  esac;
FAIRNESS running
FAIRNESS input & !forget
FAIRNESS !input & !forget
-- FAIRNESS !forget is not enough!
```
The alternating bit protocol

NuSMV model
The two bits channel (message and ABP control bit)

```
MODULE two_bit_channel (input1, input2)

VAR
  output1  : boolean;
  output2  : boolean;
  forget   : boolean;

ASSIGN
  next (output1) := case
    forget : output1;
    TRUE   : input1;
  esac;
  next (output2) := case
    forget : output2;
    TRUE   : input2;
  esac;

FAIRNESS running
FAIRNESS input1 & !forget
FAIRNESS !input1 & !forget
FAIRNESS input2 & !forget
FAIRNESS !input2 & !forget
```
The alternating bit protocol

NuSMV model
Main module

```
MODULE main

VAR
    s : process sender(ack_chan.output);
    msg_chan : process
two_bit_channel(s.message1, s.message2);
    r : process
        receiver(msg_chan.output1, msg_chan.output2);
    ack_chan : process one_bit_channel(r.ack);

ASSIGN
    init(s.message2) := FALSE;
    init(r.expected) := FALSE;
    init(r.ack) := TRUE;
    init(msg_chan.output2) := TRUE;
    init(ack_chan.output) := TRUE;

LTLSPEC
    G (s.st = sent & s.message1 -> msg_chan.output1);
```
Running NuSMV

Output

-- specification G ( F st = sent) IN s is true
-- specification G ( F st = received) IN r is true
-- specification G ((s.st = sent & s.message1) -> msg_chan.output1) is true

Try to run the model with NuSMV
Are there any questions?

Do you ever feel alone when you're with people?

I try to.