NuSMV: Planning as Model Checking *

Alessandra Giordani
agiordani@disi.unitn.it
http://disi.unitn.it/~agiordani

Formal Methods Lab Class, April 30, 2014

Università degli Studi di Trento

*These slides are derived from those by Stefano Tonetta, Alberto Griggio, Silvia Tomasi, Thi Thieu Hoa Le for FM lab 2011/13

Contents

Contents

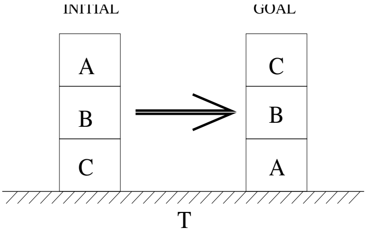
1. Planning problem

2. Examples
   - The Tower of Hanoi
   - The Ferryman
   - Tic-Tac-Toe

The problem

- Problem: Given a set of action operators OP, (a representation of) an initial state I and goal state G, find a sequence of operator applications $o_1, .., o_n$, leading from the initial state to the goal state.
- Idea: Encode it into a model checking problem.

Example



$$
\begin{array}{lll}
\mathit{Init}: & & On(A,B), On(B,C), On(C,T), Clear(A) \\
\mathit{Goal}: & & On(C,B), On(B,A), On(A,T) \\
Move(b,s,d) & & \\
 & \mathit{Precond}: & Block(b) \wedge Clear(b) \wedge On(b,s) \wedge \\
 & & (Clear(d) \vee Table(d)) \wedge \\
 & & b \neq s \wedge b \neq d \wedge s \neq d \\
 & \mathit{Effect}: & Clear(s) \wedge \neg On(b,s) \wedge \\
 & & On(b,d) \wedge \neg Clear(d)
\end{array}
$$

Encoding in SMV

- Initial states:

$$On(A, B) \wedge On(B, C) \wedge On(C, T) \wedge Clear(A).$$

- Goal states:

$$On(C, B) \wedge On(B, A) \wedge On(A, T).$$

- Action preconditions and effects:

$$\begin{aligned} & Move(A, B, C) \rightarrow \\ & \qquad Clear(A) \wedge On(A, B) \wedge Clear(C) \wedge \\ & \qquad Clear(B') \wedge \neg On(A', B') \wedge \\ & \qquad On(A', C') \wedge \neg Clear(C'). \end{aligned}$$

Planning strategy

- Specification: The goal is not reachable.
- Plan: If the property is false, NuSMV produces a counterexample. The counterexample is a plan to reach the goal.

Contents

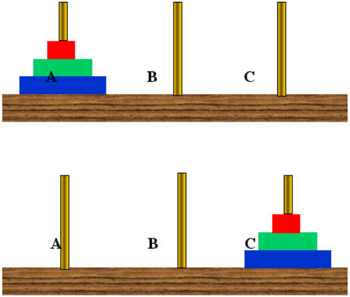
1. Planning problem

2. Examples
   - The Tower of Hanoi
   - The Ferryman
   - Tic-Tac-Toe

The Tower of Hanoi

Mathematical game constisting of three poles and `N` disks of different sizes:

- it starts with the disks in a stack in ascending order of size on the left pole (the smallest at the top → conical shape)
- the goal is to move the entire stack to the right pole:
    - only one disk may be moved at a time
    - each move consists of moving the upper disk from one pole to another one
    - no disk may be placed on top of a smaller disk



The Tower of Hanoi - Variables

```
MODULE main
-- Hanoi problem with three poles (left, middle, right)
-- and four ordered disks d1, d2, d3, d4,
-- disk d1 is the biggest one
VAR
  d1 : {left,middle,right};
  d2 : {left,middle,right};
  d3 : {left,middle,right};
  d4 : {left,middle,right};
  move : 1..4; -- possible moves
DEFINE
  move_d1 := move=1;
  move_d2 := move=2;
  move_d3 := move=3;
  move_d4 := move=4;
```

The Tower of Hanoi - Macros

```
-- A block is clear iff there is no disk on it
-- di is clear iff di!=dj for every j>i
DEFINE
  clear_d1 :=
        d1!=d2 &
        d1!=d3 &
        d1!=d4;
  clear_d2 :=
        d2!=d3 &
        d2!=d4;
  clear_d3 :=
        d3!=d4;
  clear_d4 := TRUE;
```

The Tower of Hanoi - Initial states

```
-- initially all items are on the left pole
INIT
  d1 = left &
  d2 = left &
  d3 = left &
  d4 = left;
```

The Tower of Hanoi - Transitions

```
TRANS
  move_d1 ->
-- only d1 changes
        next(d1) != d1 &
        next(d2) = d2 &
        next(d3) = d3 &
        next(d4) = d4 &
-- no other disks on d1
        clear_d1 &
-- no smaller disks on the next pole
        next(d1) != d2 &
        next(d1) != d3 &
        next(d1) != d4


...
```

The Tower of Hanoi - Specification

```
-- spec to find a solution to the problem
CTLSPEC
  ! EF (d1=right & d2=right & d3=right & d4=right)
```

```
> NuSMV hanoi4.smv
-- specification  !EF (((d1 = right & d2 = right) & d3 = right) & d4 = right)  is false
-- as demonstrated by the following execution sequence
Trace Description: CTL Counterexample
Trace Type: Counterexample
-> State: 1.1 <-
  d1 = left
  d2 = left
  d3 = left
  d4 = left
  move = 4
  clear_d4 = 1
  clear_d3 = 0
  clear_d2 = 0
  clear_d1 = 0
  move_d4 = 1
  move_d3 = 0
  move_d2 = 0
  move_d1 = 0
...
```

The Ferryman

A ferryman has to bring a goat, a cabbage, and a wolf safely across a river.

The ferryman can cross the river with at most one passenger on his boat. However he cannot leave unattended on the same side the cabbage and the goat or the goat and wolf (because the goat would eat the cabbage or the wolf would eat the goat).

Can the ferryman transport all the goods to the other side safely?

The Ferryman - Variables

```
MODULE main
VAR
-- the man and the three items
  cabbage : {right,left};
  goat    : {right,left};
  wolf    : {right,left};
  man     : {right,left};
-- possible moves
  move    : {c, g, w, e};

DEFINE
  carry_cabbage := move=c;
  carry_goat := move=g;
  carry_wolf := move=w;
  no_carry := move=e;
```

The Ferryman

```
-- initially everything is on the right bank
ASSIGN
  init(cabbage) := right;
  init(goat)    := right;
  init(wolf)    := right;
  init(man)     := right;

TRANS
  carry_cabbage ->
                cabbage=man &
                next(cabbage)!=cabbage &
                next(man)!=man &
                next(goat)=goat &
                next(wolf)=wolf
...
```

The Ferryman

```
-- goat and wolf    must not be left unattended !
-- goat and cabbage must not be left unattended !
DEFINE
  safe_state := (goat = wolf | goat = cabbage) -> goat = man;
  goal := cabbage = left & goat = left & wolf = left;

-- spec to find a solution to the problem
CTLSPEC
  ! E[safe_state U goal]
```

Tic-Tac-Toe

Tic-tac-toe is a game for two players (X and O) who take turns marking the squares of a board (→ a 3×3 grid). The player who succeeds in placing three respective marks in a horizontal, vertical or diagonal row wins the game.

The tic-tac-toe puzzle is modeled with an array of size nine.

```
  1 | 2 | 3
____|___|____
  4 | 5 | 6
____|___|____
  7 | 8 | 9
    |   |
```

Tic-Tac-Toe - The board

```
-- a square of the board can be empty or filled:
-- "0" means empty,
-- "1" filled by player 1, "2" filled by player 2
VAR
  B : array 1..9 of {0,1,2};
-- initially, all squares are empty
INIT
  B[1] = 0 &
  B[2] = 0 &
  B[3] = 0 &
  B[4] = 0 &
  B[5] = 0 &
  B[6] = 0 &
  B[7] = 0 &
  B[8] = 0 &
  B[9] = 0;
```

Tic-Tac-Toe - The players

```
-- let us assume that player 1 is the first player
-- players move alternatively
VAR
  player : 1..2;
ASSIGN
  init(player) := 1;
  next(player) := case
    player = 1 : 2;
    player = 2 : 1;
  esac;
```

Tic-Tac-Toe - The moves

```
-- move=0 means no move
-- move=i with i>0 means the current player fills B[i]
VAR  move : 0..9;
INIT  move=0
TRANS
  next(move=0) ->
      next(B[1])=B[1] &
      next(B[2])=B[2] &
      next(B[3])=B[3] &
      next(B[4])=B[4] &
      next(B[5])=B[5] &
      next(B[6])=B[6] &
      next(B[7])=B[7] &
      next(B[8])=B[8] &
      next(B[9])=B[9]
...
```

Tic-Tac-Toe - The moves

```
-- move=i with i>0 means the current player fills B[i]
TRANS
  next(move=1) ->
     B[1] = 0 & next(B[1]) = player &
     next(B[2])=B[2] &
     next(B[3])=B[3] &
     next(B[4])=B[4] &
     next(B[5])=B[5] &
     next(B[6])=B[6] &
     next(B[7])=B[7] &
     next(B[8])=B[8] &
     next(B[9])=B[9]
...
```

Tic-Tac-Toe - The end of the game

```
-- "win1" means player 1 wins
-- "win2" means player 2 wins
DEFINE
  win1 := (B[1]=1 & B[2]=1 & B[3]=1) |
          (B[4]=1 & B[5]=1 & B[6]=1) |
          (B[7]=1 & B[8]=1 & B[9]=1) |
          (B[1]=1 & B[4]=1 & B[7]=1) |
          (B[2]=1 & B[5]=1 & B[8]=1) |
          (B[3]=1 & B[6]=1 & B[9]=1) |
          (B[1]=1 & B[5]=1 & B[9]=1) |
          (B[3]=1 & B[5]=1 & B[7]=1);
  win2 := ...
```

Tic-Tac-Toe - The end of the game

```
-- "draw" means nobody wins
  draw := !win1 & !win2 &
          B[1]!=0 & B[2]!=0 & B[3]!=0 & B[4]!=0 &
          B[5]!=0 & B[6]!=0 & B[7]!=0 & B[8]!=0 & B[9]!=0;

TRANS
  (win1 | win2 | draw) <-> next(move)=0
```

Tic-Tac-Toe - Specification

A strategy is a plan that need to be accomplished for winning the game
“if the opponent has two in a row, play the third to block them”

```
-- SPECIFICATIONS

-- PLAYER 2

-- player 2 does not have a "winning" strategy
```

Tic-Tac-Toe - Specification

A strategy is a plan that need to be accomplished for winning the game
“if the opponent has two in a row, play the third to block them”

```
-- SPECIFICATIONS


-- PLAYER 2


-- player 2 does not have a "winning" strategy

CTLSPEC
  ! (AX (EX (AX (EX (AX (EX (AX (EX (AX win2)))))))))
```

Tic-Tac-Toe - Specification

A strategy is a plan that need to be accomplished for winning the game
“if the opponent has two in a row, play the third to block them”

```
-- SPECIFICATIONS


-- PLAYER 2


-- player 2 does not have a "winning" strategy

CTLSPEC
  ! (AX (EX (AX (EX (AX (EX (AX (EX (AX win2)))))))))

-- player 2 has a "non-losing" strategy
```

Tic-Tac-Toe - Specification

A strategy is a plan that need to be accomplished for winning the game "if the opponent has two in a row, play the third to block them"

```
-- SPECIFICATIONS


-- PLAYER 2


-- player 2 does not have a "winning" strategy

CTLSPEC
  ! (AX (EX (AX (EX (AX (EX (AX (EX (AX win2)))))))))

-- player 2 has a "non-losing" strategy

CTLSPEC
  AX (EX (AX (EX (AX (EX (AX (EX (AX !win1))))))))


...
```

Tic-Tac-Toe - Let’s play

Suppose player one fills 5:

```
NuSMV > check_ctlspec -p ’AG (B[1]=0 & B[2]=0 & B[3]=0 & B[4]=0 & B[5]=1 &
B[6]=0 & B[7]=0 & B[8]=0 & B[9]=0 & player=2 -> ! EX (AX (EX (AX (EX (AX
(EX (AX !win1))))))))’ ...  -> State: 2.2 <- B[5] = 1 player = 2 move = 5
-> State: 2.3 <- B[9] = 2 player = 1 move = 9 ...
```

Player two may fill 9.

Tic-Tac-Toe - Exercises

```
-- player 2 has also a "non-winning" strategy
-- player 2 does not have a "losing" strategy
-- player 2 does not have a "drawing" strategy
-- player 2 has a "non-drawing" strategy
-- player 1 does not have a "winning" strategy
-- player 1 has a "non-losing" strategy
-- player 1 has also a "non-winning" strategy
-- player 1 does not have a "losing" strategy
-- player 1 does not have a "drawing" strategy
-- player 1 has a "non-drawing" strategy
```