Automated Reasoning

Lecture 3: The NuSMV Model Checker

Jacques Fleuriot
jdf@inf.ed.ac.uk

Recap

- Previously:
  - Model Checking Introduction
  - Linear Temporal Logic
- This time: An implementation of LTL Model Checking
  - NuSMV

NuSMV

NuSMV is a symbolic model checker developed by ITC-IRST and UniTN with the collaboration of CMU and UniGE.

`http://nusmv.fbk.eu/`

The NuSMV project aims at the development of a state-of-the-art model checker that:

- is robust, open and customizable;
- can be applied in technology transfer projects;
- can be used as research tool in different domains.

NuSMV is OpenSource

- developed by a distributed community, “Free Software” license

NuSMV

NuSMV provides:

1. A language for describing finite state models of systems
   - Reasonably expressive
   - Allows for modular construction of models
2. Model checking algorithms for checking specifications written in LTL and CTL (and some other logics) against finite state machines.

A first SMV program

```
MODULE main
  VAR
    b0 : boolean
  ASSIGN
    init(b0) := FALSE;
    next(b0) := !b0;
```

An SMV program consists of:

- Declarations of state variables (`b0` in the example); these determine the state space of the model.
- Assignments that constrain the valid initial states (`init(b0) := FALSE`).
- Assignments that constrain the transition relation (`next(b0) := !b0`).

Declaring state variables

SMV data types include:

boolean:

```
x : boolean;
```

enumeration:

```
st : {ready, busy, waiting, stopped};
```

bounded integers (intervals):

```
n : 1..8;
```

arrays and bit-vectors

```
arr : array 0..3 of {red, green, blue};
bv  : signed word[8];
```

Assignments

initialisation:

```
ASSIGN
init(x) := expression ;
```

progression:

```
ASSIGN
next(x) := expression ;
```

immediate:

```
ASSIGN
y := expression ;
```

or

```
DEFINE
y := expression ;
```

Assignments

- If no **init()** assignment is specified for a variable, then it is initialised non-deterministically;
- If no **next()** assignment is specified, then it evolves nondeterministically. i.e. it is unconstrained.
  - Unconstrained variables can be used to model nondeterministic inputs to the system.
- Immediate assignments constrain the current value of a variable in terms of the current values of other variables.
  - Immediate assignments can be used to model outputs of the system.

Expressions

$$
\begin{array}{lcll}
\textit{expr} & ::= & \text{atom} & \text{symbolic constant} \\
 & \mid & \text{number} & \text{numeric constant} \\
 & \mid & \text{id} & \text{variable identifier} \\
 & \mid & \texttt{!}\,\textit{expr} & \text{logical not} \\
 & \mid & \textit{expr} \bowtie \textit{expr} & \text{binary operation} \\
 & \mid & \textit{expr}\texttt{[}\textit{expr}\texttt{]} & \text{array lookup} \\
 & \mid & \texttt{next}(\textit{expr}) & \text{next value} \\
 & \mid & \textit{case\_expr} & \\
 & \mid & \textit{set\_expr} &
\end{array}
$$

where $\bowtie \in \{\texttt{\&}, \texttt{|}, \texttt{+}, \texttt{-}, \texttt{*}, \texttt{/}, \texttt{=}, \texttt{!=}, \texttt{<}, \texttt{<=}, \ldots\}$

Case Expression

$$
\begin{array}{l}
\mathit{case\_expr} ::= \\
\quad \texttt{case} \\
\quad\quad \mathit{expr}_{a1} \;\; \texttt{:} \;\; \mathit{expr}_{b1}\texttt{;} \\
\quad\quad \ldots \\
\quad\quad \mathit{expr}_{an} \;\; \texttt{:} \;\; \mathit{expr}_{bn}\texttt{;} \\
\quad \texttt{esac}
\end{array}
$$

- Guards are evaluated sequentially.
- The first true guard determines the resulting value

Set expressions

Expressions in SMV do not necessarily evaluate to one value.

- In general, they can represent a set of possible values.
  `init(var) := {a,b,c} union {x,y,z} ;`
- destination (lhs) can take any value in the set represented by the set expression (rhs)
- constant `c` is a syntactic abbreviation for singleton `{c}`

LTL Specifications

- LTL properties are specified with the keyword `LTLSPEC`:
  `LTLSPEC <ltl_expression> ;`
- `<ltl_expression>` can contain the temporal operators:
  `X_ F_ G_ _U_`
- E.g. condition `out = 0` holds until `reset` becomes false:
  `LTLSPEC (out = 0) U (!reset)`

ATM Example

```
MODULE main
VAR
  state: {welcome, enterPin, tryAgain, askAmount,
          thanksGoodbye, sorry};
  action: {cardIn, correctPin, wrongPin, ack, cancel,
           fundsOK, problem, none};
ASSIGN
  init(state) := welcome;
  next(state) := case
    state = welcome & action = cardIn       : enterPin;
    state = enterPin & action = correctPin : askAmount ;
    state = enterPin & action = wrongPin   : tryAgain;
    state = tryAgain & action = ack        : enterPin;
    state = askAmount & action = fundsOK   : thanksGoodbye;
    state = askAmount & action = problem   : sorry;
    state = enterPin & action = cancel     : thanksGoodbye;
    TRUE                                   : state;
  esac;
LTLSPEC F( G state = thanksGoodbye
         | G state = sorry
        );
```

Running NuSMV

Batch

```
$ NuSMV atm.smv
```

Interactive

```
$ NuSMV -int atm.smv
NuSMV > go
NuSMV > check_ltlspec
NuSMV > quit
```

- `go` abbreviates the sequence of commands `read_model`, `flatten_hierarchy`, `encode_variables`, `build_model`.
- For command options, use `-h` or look in the NuSMV User Manual.

Expected Failure

```
NuSMV > check_ltlspec
-- specification  F ( G state = thanksGoodbye
                    |  G state = sorry)  is false
-- as demonstrated by the following execution sequence
Trace Description: LTL Counterexample
Trace Type: Counterexample
-> State: 1.1 <-
  state = welcome
  input = cardIn
-> State: 1.2 <-
  state = enterPin
  input = correctPin
-- Loop starts here
-> State: 1.3 <-
  state = askAmount
  input = ack
-> State: 1.4 <-
```

Unexpected Failure

```
-- specification
    ( F ( G !(state = askAmount)) ->
      F ( G state = thanksGoodbye |  G state = sorry))
        is false
-- as demonstrated by the following execution sequence
Trace Description: LTL Counterexample
Trace Type: Counterexample
-> State: 2.1 <-
  state = welcome
  input = cardIn
-- Loop starts here
-> State: 2.2 <-
  state = enterPin
  input = ack
-> State: 2.3 <-
```

Success

```
-- specification
     ( G (((state = welcome ->  F input = cardIn) &
           (state = enterPin ->
              F (state = enterPin &
             (input = correctPin | input = cancel)))) &
           (state = askAmount ->  F (input = fundsOK
                                   | input = problem))) ->
      F ( G state = thanksGoodbye |  G state = sorry))
    is true
```

Modules

```
MODULE counter
VAR digit : 0..9;
ASSIGN
  init(digit) := 0;
  next(digit) := (digit + 1) mod 10;

MODULE main
VAR c0 : counter;
    c1 : counter;
    sum : 0..99;
ASSIGN
    sum := c0.digit + 10 * c1.digit;
```

- Modules are instantiated in other modules. The instantiation is performed inside the VAR declaration of the parent module.
- In each SMV specification there must be a module main. It is the top-most module.
- All the variables declared in a module instance are visible in the module in which it has been instantiated via the dot notation (e.g., `c0.digit`, `c1.digit`).

Modules

```
MODULE counter
VAR digit : 0..9;
ASSIGN
  init(digit) := 0;
  next(digit) := (digit + 1) mod 10;

MODULE main
VAR c0 : counter;
    c1 : counter;
    sum : 0..99;
ASSIGN
    sum := c0.digit + 10 * c1.digit;

LTLSPEC
  F sum = 13;
```

- Is this specification satisfied by this model?

```
-- specification  F sum = 13  is false
-- as demonstrated by the following execution sequence
Trace Description: LTL Counterexample
Trace Type: Counterexample
-- Loop starts here
-> State: 1.1 <-
  c0.digit = 0
  c1.digit = 0
  sum = 0
-> State: 1.2 <-
  c0.digit = 1
  c1.digit = 1
  sum = 11
-> State: 1.3 <-
  c0.digit = 2
  c1.digit = 2
  sum = 22
...
```

Modules with parameters

```
MODULE counter(inc)
VAR digit : 0..9;
ASSIGN
  init(digit) := 0;
  next(digit) := inc ? (digit + 1) mod 10
                     : digit;
DEFINE top := digit = 9;

MODULE main
VAR c0 : counter(TRUE);
    c1 : counter(c0.top);
    sum : 0..99;
ASSIGN
  sum := c0.digit + 10 * c1.digit;
```

- Formal parameters (inc) are substituted with the actual parameters (`TRUE`, `c0.top`) when the module is instantiated.
- Actual parameters can be any legal expression.
- Actual parameters are passed by reference.

```
-- specification  F sum = 13  is true
```

Summary

- Introduction to NuSMV
  - H&R Section 3.3
  - NuSMV Tutorial:
    `http://nusmv.fbk.eu/NuSMV/tutorial/v26/tutorial.pdf`
  - NuSMV Start-up Guide on FV Web Page
- Next time:
  - Introduction to the practical exercise.