Formal Methods for Cyber-Physical Systems

Lab 1: Introduction to NuSMV

Davide Bresolin
Last updated on: October 26, 2021

Università degli Studi di Padova

Lesson schedule

14:30 Introduction to NuSMV

Exercises (in groups of 2/3 students)

15:15 First review

Exercises (in groups of 2/3 students)

16:00 Final review

NuSMV

NuSMV is a symbolic model checker developed by FBK and UniTN in collaboration with CMU and UniGE

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

NuSMV is a state-of-the art tool:

- open, robust and customizable
- of industry standard level
- used by many research groups and academic institutions

NuSMV is Open Source

- distributed under the LGPL licence

NuSMV

NuSMV provides:

1. A language to define finite state synchronous models
   - with good expressivity
   - that allow compositional definition
2. A simulator to generate executions of the model
3. A number of model checking algorithms to verify safety and liveness properties

Our first SMV program


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

An SMV program consists of:

- declaration of state variables (`b0` in the example), that defines the set of states of the model;
- initialisation assignments that defines initial states (`init(b0) := FALSE`)
- reaction assignments that defines the transition relation (`next(b0) := !b0`)

Variable declaration

SMV datatypes includes:

Booleans:

```
x : boolean;
```

Enumerated types:

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

Bounded integers:

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

Arrays and bitvectors:

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

Assignments

Initialisation:

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

Progress:

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

Immediate:

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

or

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

Assignments (1)

- `next(x)` is the value of variable `x` on the next round, and can be used in expressions
  - e.g. `next(y) := next(x);`
- order of assignments does not matter
  - `next(x) := y; next(y) := x;` is the same of `next(y) := x; next(x) := y;` (why?)

Assignments (2)

- Variables with no `init()` are nondeterministically initialised with all possible values;
- Variables with no `next()` evolve nondeterministically, that is, are unconstrained
  - unconstrained variables can model inputs
- Immediate assignments constrain the current valute of a variable w.r.t. the current value of other variables
  - can be used to model outputs

Expressions

Arithmetical operators:

```
+ - * / mod
```

Comparison operators:

```
= != > < <= >=
```

Logical operators:

```
& | xor ! (not) -> <->
```

Conditional expressions

```
case
    guard_1 : expression_1;
    guard_2 : expression_2;
    ...
    TRUE    : expression_n;
esac
```

- Guards are evaluated sequentially
- The first guard that is true gives the value of the expression

Set expressions

Expressions in SMV are not always evaluated as a single value

- In general, they represent a set of values
  `init(var) := {a, b, c} union {x, y, z}`
- destination (LHS) nondeterministically takes a value in the set expression (RHS)
- a constant `c` is a shortcut for the set `{c}`

Invariants

- The safety properties to be verified (invariants) are defined with the keyword `INVARSPEC`
  `INVARSPEC <boolean_expression> ;`
- `<boolean_expression>` is defined using logical operators

Examples:

- the value of `x` is between 0 and 3:
  `INVARSPEC x >= 0 & x <= 3 ;`
- when `mode` is `off` the value of `x` is 0:
  `INVARSPEC mode = off -> x = 0 ;`
- trains should not be on bridge simultaneously:

```
INVARSPEC !(trainW.mode = bridge &
            trainE.mode = bridge)
```

Example: the switch


```
MODULE main
    -- Model of the switch
    IVAR
        press   : boolean;
    VAR
        mode    : {on, off};
        x       : 0..15;
    ASSIGN
        init(mode) := off;
        next(mode) := case
            mode = off & press               : on;
            mode = on & (press | x >= 10)    : off;
            TRUE                             : mode;
        esac;
        init(x) := 0;
        next(x) := case
            mode = on & next(mode) = off     : 0;
            mode = on & x < 10               : x + 1;
            TRUE                             : x;
        esac;

INVARSPEC x <= 10
INVARSPEC mode = off -> x = 0
INVARSPEC x < 10
INVARSPEC mode = off
```

Running NuSMV

Batch mode:

```
$ NuSMV switch.smv
```

Interactive mode:

```
$ NuSMV -int switch.smv
NuSMV > go
NuSMV > check_invar
NuSMV > quit
```

- `go` is a shortcut for the sequence of commands `read_model`, `flatten_hierarchy`, `encode_variables`, `build_model`
- More information on the NUSMV Tutorial and Manual available on Moodle

Verified properties


```
NuSMV > check_invar
-- invariant x <= 10  is true
-- invariant (mode = off -> x = 0)  is true
```

Counterexamples


```
NuSMV > check_invar
-- invariant mode = off  is false
-- as demonstrated by the following execution sequence
Trace Description: AG alpha Counterexample
Trace Type: Counterexample
  -> State: 2.1 <-
    mode = off
    x = 0
  -> Input: 2.2 <-
    press = TRUE
  -> State: 2.2 <-
    mode = on
```

Simulation

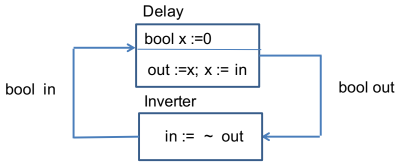
- Simulation in NuSMV proceeds as follows:
  1. an initial state is selected and become the current state
  2. the next current state is selected following the evolution of the system
  3. repeat step (2)

- NuSMV has three simulation strategies:
  - Deterministic: the first available state is choosen;
  - Random: states are selected randomly
  - Interactive: the user select states

Simulation commands

- `pick_state` selects an initial state (wiht deterministic strategy)
  - `-r` to use random strategy
  - `-i` to use interactive strategy
- `print_current_state` show the current state
  - `-v`: verbose, show values of state variables
- `simulate -k` *n* simulate *n* execution steps
  - `-r` to use random strategy
  - `-i` to use interactive strategy
- `show_traces` show the traces stored in memory
  - `-v`: verbose
  - `-t`: print total number of traces
  - *n*: print trace number *n*

Composition



- delay and inverter are components with a formal parameter
- the main module instantiate and compose delay and inverter
- the formal paramter input is used to define the connections

```
MODULE delay(input)
    -- Model of the delay component
    VAR
        x : boolean;
    ASSIGN
        init(x) := FALSE;
        next(x) := input;
    DEFINE
        out := x;

MODULE inverter(input)
    -- Model of the inverter
    DEFINE
        out := !input;

MODULE main
    -- Composition delay||inverter
    VAR
        del : delay(inv.out);
        inv : inverter(del.out);
```