NuXMV System

Behavior models and verification
• State-of-the-art symbolic model checker
  ▪ Recall: Spin is an explicit model checker
• Based on NuSMV, extending it
• Originally developed at Carnegie Mellon University (SMV)
• Special input language, CTL properties
  ▪ Uses OBDDs for states' manipulation
  ▪ Both synchronous and asynchronous systems may be described
The SMV Input Language

- Parallel assignment syntax
  - In each step all variables are reassigned
  - Issues of circular dependencies, ...
  - Allows for easy OBDD modeling
- Initial values and “next” values are specified
Example I.

MODULE main

VAR
  request : boolean;
  state : {ready,busy};

ASSIGN
  init(state) := ready;
  next(state) := case
    state = ready & request : busy;
    TRUE : {ready,busy};
  esac;

SPEC
  AG(request -> AF state = busy)
MODULE counter_cell(carry_in)

VAR
  value : boolean;

ASSIGN
  init(value) := FALSE;
  next(value) := value xor carry_in;

DEFINE
  carry_out := value & carry_in;

MODULE main

VAR
  bit0 : counter_cell(TRUE);
  bit1 : counter_cell(bit0.carry_out);
  bit2 : counter_cell(bit1.carry_out);

SPEC AF(bit2.carry_out)
**Types**

- boolean
- enum
- word – specifying bit width
  - e.g. word[3] → three-bit range, i.e., 0-7
  - Cannot create unions thereof → use integer sets instead
- integer
- real – rational numbers 😊
- arrays, wordarrays, intarrays (unbounded arrays)
  - nesting of types
- sets – limited, just sets of boolean, integer, symbolic and mixed enums
FAIRNESS, INIT, TRANS and INVAR

- Propositional way
- **FAIRNESS**
  - A fairness constraint – mostly used with *running*
- **INIT**
  - Initial value of local variables
- **TRANS**
  - Definition of transition relation
    - next(output) = !input | next(output) = output
- **INVAR**
  - Conditions restricting valid states
The use of **INIT**, **TRANS** and **INVAR** NOT recommended

- “Logical absurdities (...) can lead to unimplementable descriptions”
- Resulting in systems with no transitions, etc.

**However**

- It may be flexible when translating from other languages to SMV


- **DEFINE** – symbols definition

```plaintext
DEFINE

start := state = 0 & timeout;
finish := state = 3;
request := case
    state = 0: 0;
    TRUE : 1;
esac;
```
Modules

- Encapsulation of a group of declarations
  - Can be parametrized when reusing
  - Can contain instances of other modules
  - Example: see above

- A parameterless `main` module has to be in each program
Properties inside models can be verified

- nuxmv model.smv
- All CTL properties specified inside the models are checked

Or the model can be simulated

- nuxmv -int model.smv
- Interactively, randomly, or deterministically
Simulation

- `nusmv -int model.smv`
  - Starts the nuxmv in interactive model
- `go`
  - Prepares the model
- `pick_state -r`
  - Picks up initial state (-r randomly, -i interactively)
- `print_current_state -v`
  - Prints the current state (-v verbosely)
- `simulate -r -k 3`
  - Simulates randomly three steps
SMV Information

- The original tool and manual downloadable at
  [http://www.cs.cmu.edu/~modelcheck/smv.html](http://www.cs.cmu.edu/~modelcheck/smv.html)

- Implementations to use: NuSMV and NuXMV
  [http://nuxmv.fbk.eu/](http://nuxmv.fbk.eu/)

- NuXMV is newer and recommended