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
Parallel assignment language

In each step all variables are reassigned – possible issues of circular dependencies, ...

Allows for easy OBDD modelling

Initial 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)
EXAMPLE II.

**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 of specification
- FAIRNESS – a fairness constraint, mostly used with running
- INIT – initial value of local variables
- TRANS – definition of transition relation, e.g.:
  - \( \text{next}(\text{output}) = !\text{input} \mid \text{next}(\text{output}) = \text{output} \)
- INVAR – conditions restricting valid states

Use of INIT, TRANS and INVAR NOT recommended as “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

- Does not introduce new variable, just new symbol

```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 of models can be verified via `nuxmv model.smv`  
All CTL properties specified inside the models are checked  
Model can be simulated via `nuxmv -int model.smv`  
– interactively, randomly, or deterministically
nuXmv -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
The original tool and manual downloadable at
http://www.cs.cmu.edu/~modelcheck/smv.html

Implementations to use: NuSMV and NuXMV:
- http://nusmv.fbk.eu/
- http://nuxmv.fbk.eu/

NuXMV is newer and recommended, documentation: