my_example1.smv

```
-------------------- Example 1 ----------------
MODULE main
VAR
    button : boolean;
    machine : {ready, busy, serving};

ASSIGN
    button := {0,1};

    init(machine) := ready;
    next(machine) :=
        case
            machine=ready & button : busy;
            machine=ready : machine;
            machine=serving : ready;
            1 : {busy, serving};
        esac;
```

my_example2.smv

```
MODULE main
VAR
  bit0 : counter_cell(1);
  bit1 : counter_cell(bit0.carry_out);
  bit2 : counter_cell(bit1.carry_out);

MODULE counter_cell(carry_in)
VAR
  value : boolean;
ASSIGN
  init(value) := 0;
  next(value) := value + carry_in mod 2;
DEFINE
  carry_out := value & carry_in;
```

```
expr ::
        atom                          ;; a symbolic constant
        | number                      ;; a numeric constant
        | id                          ;; a variable identifier
        | "!" expr                    ;; logical not
        | expr1 "&" expr2             ;; logical and
        | expr1 "|" expr2             ;; logical or
        | expr1 "->" expr2            ;; logical implication
        | expr1 "<->" expr2           ;; logical equivalence
        | expr1 "=" expr2             ;; equality
        | expr1 "!=" expr2            ;; disequality
        | expr1 "<" expr2             ;; less than
        | expr1 ">" expr2             ;; greater than
        | expr1 "<=" expr2            ;; less that or equal
        | expr1 ">=" expr2            ;; greater than or equal
        | expr1 "+" expr2             ;; integer addition
        | expr1 "-" expr2             ;; integer subtraction
        | expr1 "*" expr2             ;; integer multiplication
        | expr1 "/" expr2             ;; integer division
        | expr1 "mod" expr2           ;; integer remainder
        | "next" "(" id ")"           ;; next value
        | set_expr                    ;; a set expression
        | case_expr                   ;; a case expression
```

```
*,/
+,-
mod
=,!=,<,>,<=,>=
!
&
|
->,<->
```

```
case_expr ::
        "case"
          expr_a1 ":" expr_b1 ";"
          expr_a2 ":" expr_b2 ";"
          ...
          expr_an ":" expr_bn ";"
        "esac"


set_expr ::
        "{" val1 "," ... "," valn "}"
        | expr1 "in" expr2          ;; set inclusion predicate
        | expr1 "union" expr2       ;; set union


decl :: "VAR"
          atom1 ":" type1 ";"
          atom2 ":" type2 ";"
          ...
```

```
type :: boolean
        | "{" val1 "," val2 "," ... valn "}"
        | "array" expr1 ".." expr2 "of" type
        | atom [ "(" expr1 "," expr2 "," ... exprn ")" ]
        | "process" atom [ "(" expr1 "," expr2 "," ... exprn ")" ]

val  :: atom | number
```

```
decl :: "ASSIGN"
          dest1 ":=" expr1 ";"
          dest2 ":=" expr2 ";"
          ...

dest :: atom
        | "init" "(" atom ")"
        | "next" "(" atom ")"



decl :: "DEFINE"
          atom1 ":=" expr1 ";"
          atom2 ":=" expr2 ";"
          ...
          atomn ":=" expr3 ";"
```

```
MODULE main
VAR
   gate1 : inverter(gate3.output);
   gate2 : inverter(gate1.output);
   gate3 : inverter(gate2.output);


MODULE inverter(input)
VAR
   output : boolean;
INIT
   output = 0
TRANS
   next(output) = !input | next(output) = output
```

```
MODULE main
VAR
  semaphore : boolean;
  proc1 : process user(semaphore);
  proc2 : process user(semaphore);
ASSIGN
  init(semaphore) := 0;



MODULE user (semaphore)
VAR
  state : {idle,entering ,critical,exiting};
ASSIGN
  init(state) := idle;
  next(state) :=
    case
      state = idle : {idle,entering};
      state = entering & !semaphore : critical;
      state = critical : {critical,exiting};
      state = exiting : idle;
      1 : state;
    esac;
  next(semaphore) :=
    case
      state = entering : 1;
      state = exiting : 0;
      1 : semaphore;
    esac;
```

```
MODULE main
VAR
  y : boolean;  -- the semaphore variable. It is assigned by both processes.
  proc[1] : process user(y);  -- The two processes have interleaved executior
  proc[2] : process user(y);

ASSIGN
  init(y) := 1;
```

```
MODULE user(y)
VAR
  loc : {0,1,2,3,4};
ASSIGN
  init(loc) := 0;

  next(loc) :=
    case
      loc in {0,3}     : loc+1;
      loc = 1          : {1,2};
      loc = 2 & y = 1  : 3;
      loc = 4          : 0;
      1 : loc;
    esac;

  next(y) :=                          -- changes to the semaphore variable.
    case
      loc = 2 & next(loc) = 3 : 0; -- turned off when moving from 1_2 to 1_3
      loc = 4 & next(loc) = 0 : 1; -- turned on when moving from 1_4 to 1_0
      1 : y;
    esac;
```

```
MODULE main
VAR
  y : boolean;  -- the semaphore variable. It is assigned by both processes.
  proc[1] : process user(y);  -- The two processes have interleaved executior
  proc[2] : process user(y);

ASSIGN
  init(y) := 1;

JUSTICE
  !proc[1].loc=3, !proc[2].loc=3

COMPASSION
  (proc[1].loc = 2 & y > 0,proc[1].loc = 3),
  (proc[2].loc = 2 & y > 0,proc[2].loc = 3)


MODULE user(y)
VAR
  loc : {0,1,2,3,4};
ASSIGN
  init(loc) := 0;

  next(loc) :=
    case
      loc in {0,3}    : loc+1;
      loc = 1         : {1,2};
      loc = 2 & y = 1 : 3;
      loc = 4         : 0;
      1 : loc;
    esac;

  next(y) :=                          -- changes to the semaphore variable.
    case
      loc = 2 & next(loc) = 3 : 0; -- turned off when moving from l_2 to l_3
      loc = 4 & next(loc) = 0 : 1; -- turned on when moving from l_4 to l_0
      1 : y;
    esac;
```

```
# Each step should either increase or decrease (mod 5) bc, and either
# subtract 1 (mod 5) from f or add one to d. f,d are oreserved when
# they are not assigned to).

MODULE main

VAR
   bc  : 0..4;
   d   : 0..4;
   f   : 0..4;

   Z : add_one(bc) ;
   B : add_one(bc) ;
   C : subtract_one(bc) ;
   D : add_one(d) ;
   F : subtract_one(f) ;

 COMPOSED
   (B || C) ||| ( D || F )

 MODULE add_one(x)
 ASSIGN
   init(x) := 0;
   next(x) := (x + 1) mod 5;

 MODULE subtract_one(x)
 ASSIGN
   init(x) := 0;
   next(x) := case
                x = 0 : 4;
                    1 : x - 1;
              esac;
```

Note that since Z does not appear in the COMPOSED section, its declaration is meaningless