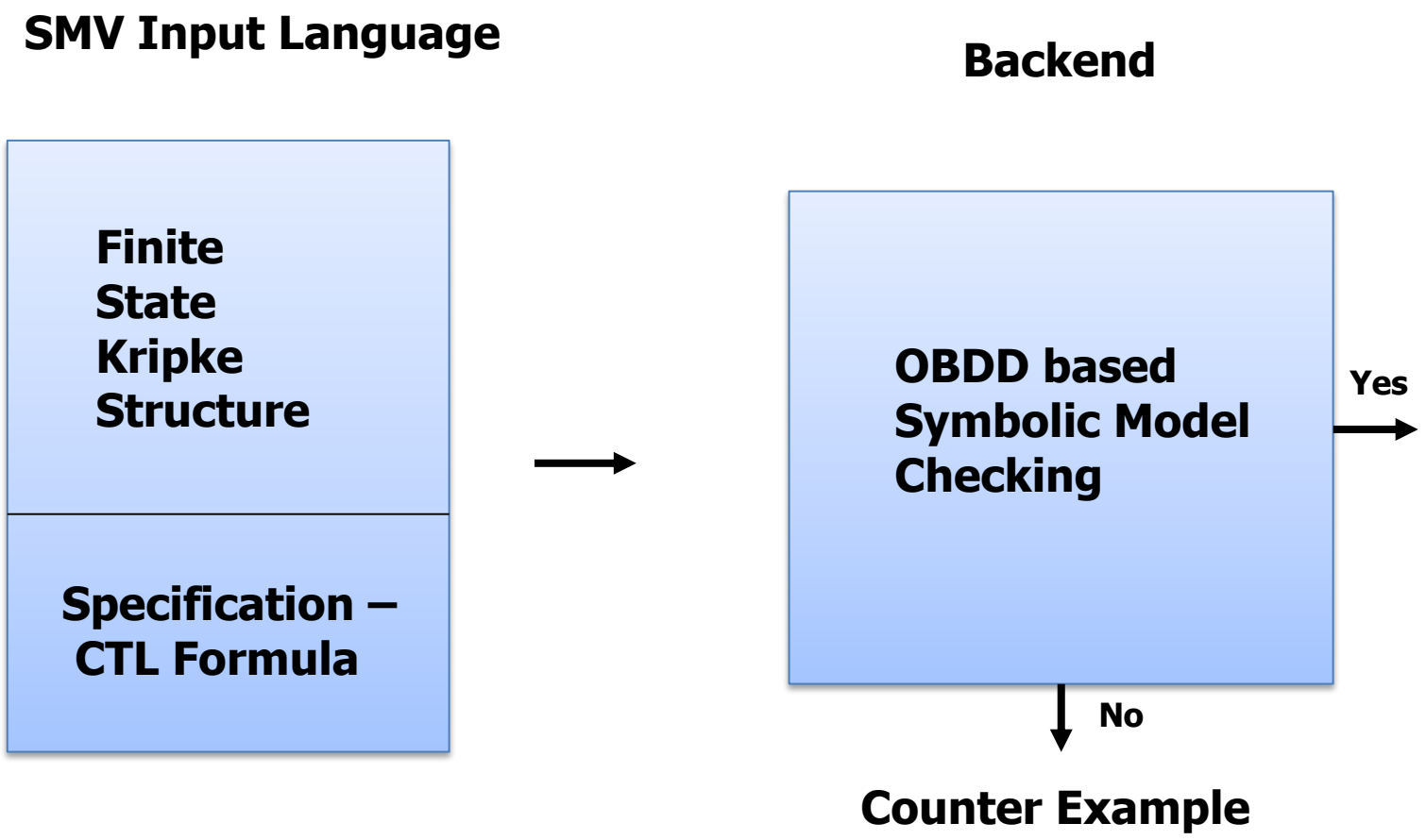
Cours 3 : Introduction à SMV (Symbolic Model Verifier)

Nga Nguyen

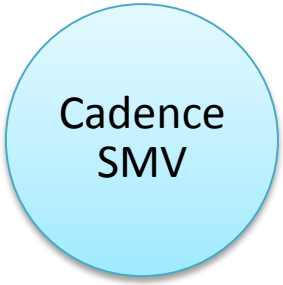
Overview SMV



Outils SMV

- Version plus ancienne

- 2.x: open source, basé sur BDD et SAT
- http://nusmv.fbk.eu/



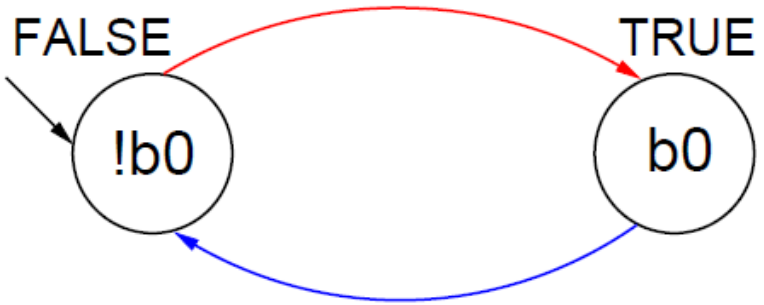
- Fonctions avancées
- GUI

NuSMV

- Description des systèmes synchrones et asynchrones
- Modularité
- Types de données finies :
  - booléen
  - énumération
  - intervalle entière
  - tableau
- Non- déterminisme

Hello world

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



Variables et Affectations

```
VAR
    atom1 : type1 ;
    atom2 : type2 ;


ASSIGN
    dest1 := Expr1 ;
    dest2 := Expr2 ;
          …
   dest ::     atom                -- current
             | init(atom)          -- initial
             | next(atom)          -- next state
```

Types de données

- Booléen :

  x : boolean;

- Énumération :

  s : {ready, busy, waiting, stopped};

- Intervalle :

  n : 1..8;

- Tableau des bits :

  unsigned word[3]; -- [0 .. 7]

  signed word[8]; -- [-128 .. 127]

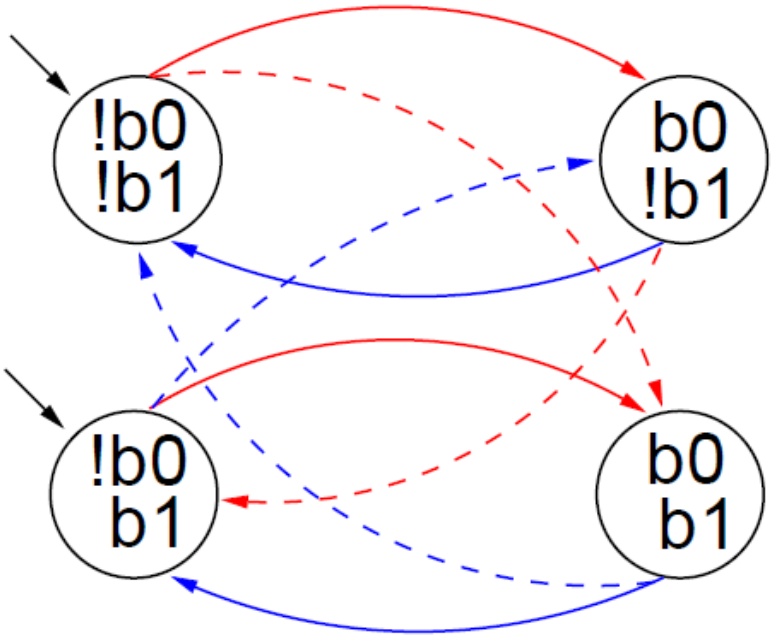
Tableau

```
VAR
   x : array 0..10 of boolean;
   y : array 2..4 of 0..10;
ASSIGN
   init(x[5]) := 1;
   init(y[2]) := {0,2,4,6,8,10};
```

- Des indices de tableau doivent être des constantes.

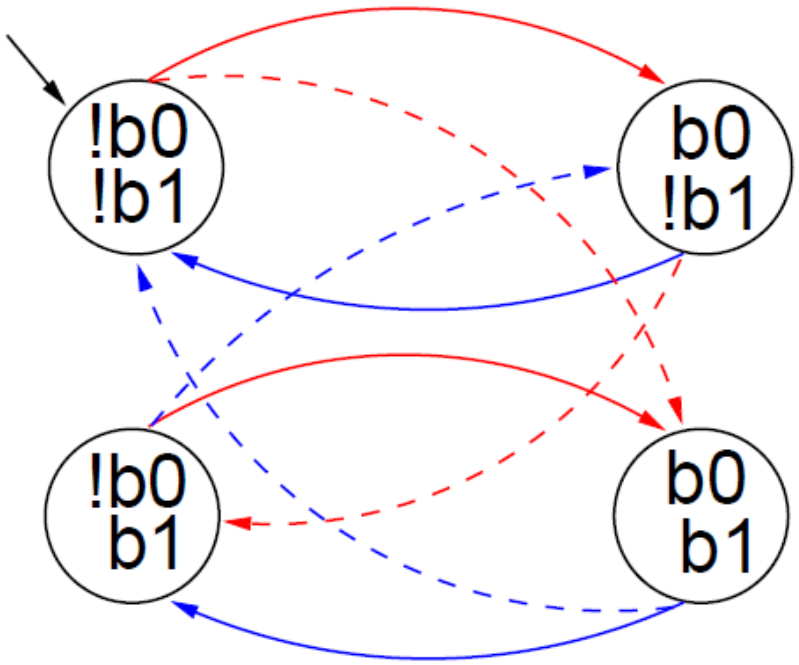
Ajouter une variable (1)

```
MODULE main
VAR
    b0 : boolean;
    b1 : boolean;
ASSIGN
    init(b0) := FALSE;
    next(b0) := !b0;
```



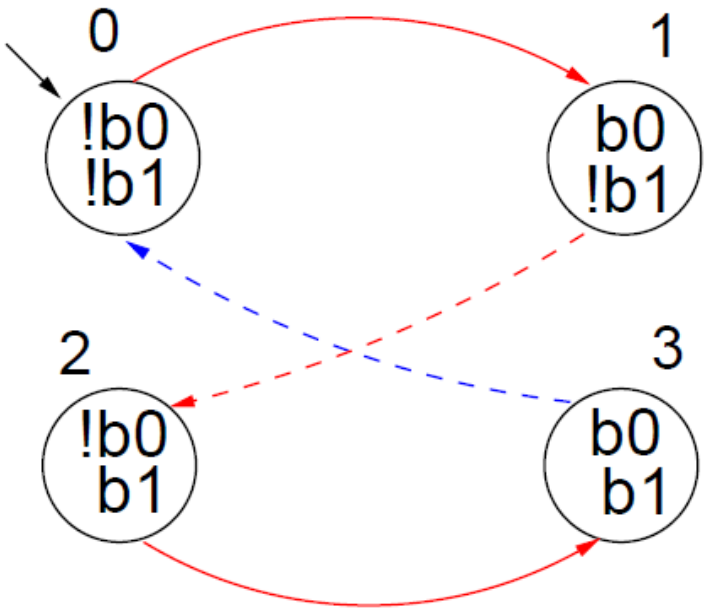
Ajouter une variable (2)

```
MODULE main
VAR
    b0 : boolean;
    b1 : boolean;
ASSIGN
    init(b0) := FALSE;
    next(b0) := !b0;

    init(b1) := FALSE;
```



Ajouter une variable (3) : compteur modulo 4

```
MODULE main
VAR
    b0 : boolean;
    b1 : boolean;
ASSIGN
    init(b0) := FALSE;
    next(b0) := !b0;

    init(b1) := FALSE;
    next(b1) := ((!b0 & b1) | (b0 & !b1));
```



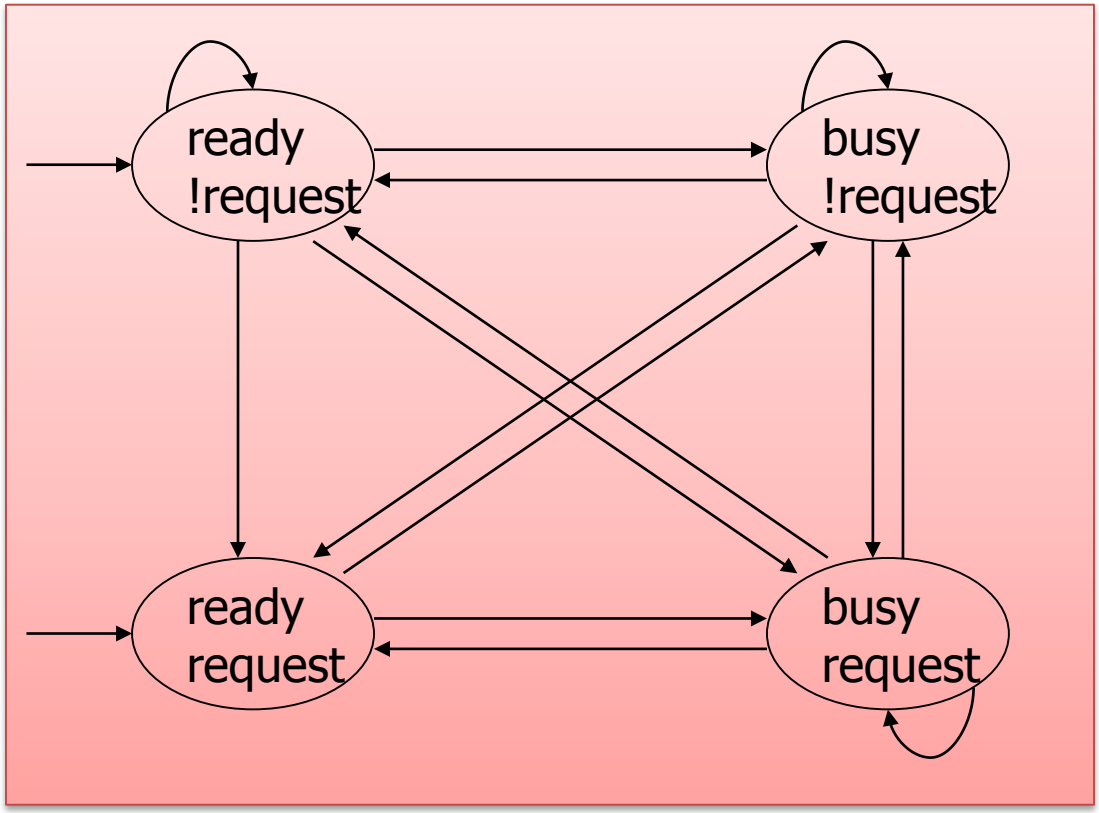
NuSMV Syntaxe: Expressions

```
Expr ::
          atom                          -- symbolic constant
        | number                        -- numeric constant
        | id                            -- variable identifier
        | Arith_expr                    -- +, -, *, /, mod, -(unary)
        | Comp_expr                     -- =, !=, >, <, <=, >=
        | ! Expr                        -- logical not
        | Expr & Expr                   -- logical and
        | Expr | Expr                   -- logical or
        | Expr xor Expr                 -- logical xor
        | Expr -> Expr                  -- logical implication
        | Expr <-> Expr                 -- logical equivalence
        | next(id)                      -- next value
        | Case_expr
        | Set_expr
```

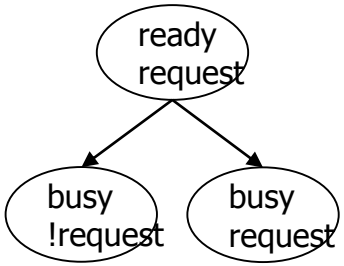
Exemple : Request

```
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))
```

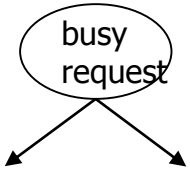
Structure de Kripke



Arbre de calcul



Vraie après un étape



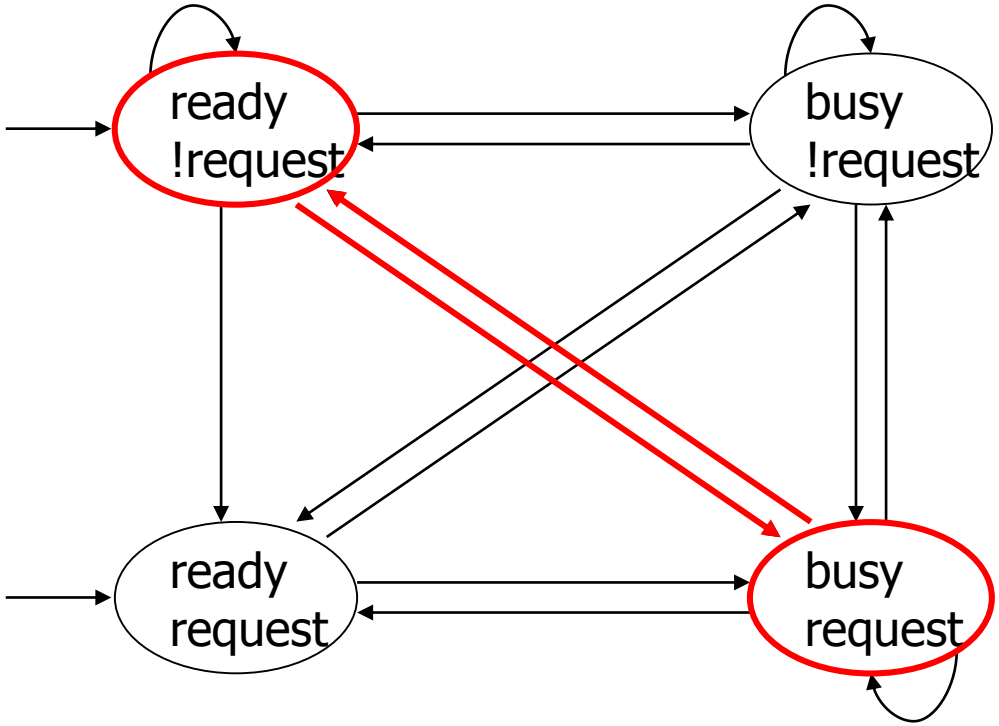
Vraie à l'état initial

```
AG(request -> AF (state = busy))
```

Exemple

```
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 -> AX (state = busy))
```

`AG(request -> AX (state = busy))` is false



Expression **Case**

```
Case_expr :: case
                  expr_a1 : expr_b2 ;
                        …
                  expr_an : expr_bn ;
             esac
```

- Les gardes sont évaluées séquentiellement
- La première qui est vraie détermine le résultat
- Les cas doivent être exhaustifs
- C'est une erreur si **toutes** les expressions à gauche sont évaluées FALSE

Expression avec des ensembles

`in` **`union`**

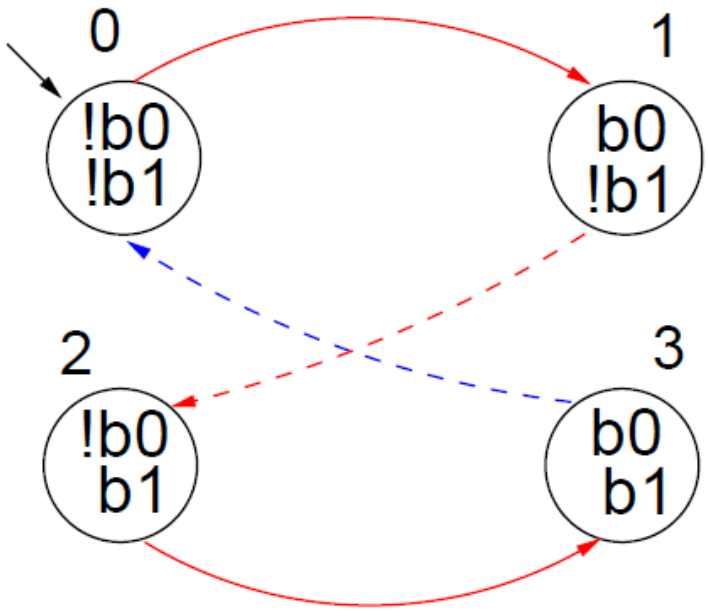
- a := state in {s0, s1, s2};
- init(var) := {a, b, c} union {x, y, z};

Variables et Affectations

- Affectation pour l'état initial:
  - `init(value) := FALSE;`
- Affectation pour l'état suivant (relation de transition)
  - `next(value) := value xor carry_in;`
- Affectation pour l'état actuel (invariant)
  - `carry_out := value & carry_in;`
- *Ou init-next ou invariant, pas les deux en même temps !*

Exemple : compteur modulo 4

```
MODULE main
VAR
    b0 : boolean;
    b1 : boolean;
    out : 0..3
ASSIGN
    init(b0) := FALSE;
    next(b0) := !b0;

    init(b1) := FALSE;
    next(b1) := ((!b0 & b1) | (b0 & !b1));

    out := toint(b0) + 2*toint(b1);
```



Restrictions

- La relation de transition doit être totale (chaque état doit avoir au moins un successeur) :
  - Règle d'affection double :
    - Chaque variable peut être affectée 1 seule fois!
  - Règle de dépendance circulaire :
    - Une variable ne peut pas avoir de cycles dans son graphe de dépendance qui ne sont pas cassés par des délais

Quels exemples sont illégaux ?

```
init(status) := ready;
init(status) := busy;

init(status) := {ready, busy};

x := (y + 1) mod 2;
y := (x + 1) mod 2;

next(x) := x & next(y);
next(y) := y & next(x);

next(x) := x & next(y);
next(y) := y & x;
```

Quels exemples sont illégaux ?

```
init(status) := ready;                -- illégal, affectation double
init(status) := busy;

init(status) := {ready, busy};        -- ok

x := (y + 1) mod 2;                   -- illégal, dépendance circulaire
y := (x + 1) mod 2;

next(x) := x & next(y);               -- illégal, dépendance circulaire
next(y) := y & next(x);

next(x) := x & next(y);               -- ok, pas de cycle
next(y) := y & x;
```

Non-déterministe

- Variables non affectées désignent des entrées sans contrainte
- `{val_1, …, val_n} :` expression qui peut prendre n'importe quelle valeur dans la liste de manière non-déterministe
  - `next(b) := {TRUE, FALSE};`
- Un choix non-déterministe peut être utilisé pour modéliser :
  - un environnement en dehors de contrôle du système
  - une implémentation pas encore raffinée
  - un comportement abstrait

DEFINE : compteur modulo 4

```
MODULE main
VAR
    b0 : boolean;
    b1 : boolean;

ASSIGN
    init(b0) := FALSE;
    next(b0) := !b0;
    init(b1) := FALSE;
    next(b1) := ((!b0 & b1) | (b0 & !b1));

DEFINE
    out := toint(b0) + 2*toint(b1);
```

- Il n'y a pas de définition avec VAR, pas de variable supplémentaire

Spécifications

- Propriétés :
  - CTL : SPEC
  - LTL : LTLSPEC
  - Sur les états atteignables (invariants) : INVARSPEC
    - INVARSPEC (reset -> pressed)

Déclaration de spécification CTL

```
Decl     :: SPEC ctlform

Ctlform ::   expr          -- bool expression
           | ! ctlform
           | ctlform <op> ctlform
           | E pathform
           | A pathform

Pathform ::  X ctlform
           | F ctlform
           | G ctlform
           | [ctlform U ctlform]
```

NuSMV : mode interactif

- nusmv –int
  - Usage de base :
    - `NuSMV > read_model –i input_file`
    - `NuSMV > go`

      préparer le modèle à vérifier
    - `NuSMV > check_ctlspec`
    - `NuSMV > check_invar`

      vérifier les propriétés
- NuSMV home page, tutorial, user manual, …
  - http://nusmv.fbk.eu/