

Introduction à la vérification formelle

NuSMV Model-checker

Pascale Le Gall

CentraleSupélec

Janvier 2016

NuSMV

NuSMV est un model checker développé par les universités de Trento (Italie), Genève (Suisse) et Carnegie Mellon University (Etats-Unis)

<http://nusmv.fbk.eu/>

<https://en.wikipedia.org/wiki/NuSMV>

Open Source, Free Software license

Un premier programme

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

- ▶ Déclaration de variables
b0
- ▶ Désignation des états initiaux
init(b0) := TRUE ;
- ▶ Construction des transitions
next(b0) := !b0 ;

Déclaration des variables des systèmes de transition (MODULE)

- ▶ variables booléennes
x : boolean ;
avec pour valeurs, les éléments de TRUE, FALSE
- ▶ variables définies par un type énuméré
st : {ready, busy, waiting} ;
- ▶ variables définies par un intervalle borné d'entiers
n : 1..8 ;
- ▶ variables définies par un tableau
a : array 1..8 of {rouge, vert, orange} ;
- ▶ variables définies par vecteur de bits
v : unsigned word[3] ;

Déclaration des variables des systèmes de transition (MODULE)

- ▶ variables définies comme des tableaux

VAR

```
x : array 0..10 of boolean ;  
y : array 1..3 of {red, green, orange};
```

ASSIGN

```
init(x[0]) := TRUE;  
init(y[2]) := green ;
```

- ▶ variables définies comme des tableaux de tableaux

VAR

```
z : array 0..10 of array 0..1 of boolean ;
```

ASSIGN

```
init(z[8][1]) := TRUE;
```

Affectations (Assignments)

- ▶ initialisation

ASSIGN

`init(x) := expression ;`

- ▶ état suivant

ASSIGN

`next(x) := expression ;`

- ▶ état courant

ASSIGN

`x := expression ;`

(utile pour modéliser les valeurs de sortie des modules)

- ▶ définition auxiliaire

DEFINE

`x := expression ;`

(x n'est pas une variable du système de transition, mais une variable auxiliaire définie à partir des variables du système)

Un second exemple

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

- ▶ L'ensemble des états est le produit cartésien des domaines de valeurs des variables du module
- ▶ La variable booléenne b1 est sous-spécifiée : elle peut prendre les valeurs TRUE ou FALSE à l'état initial et toutes les transitions sont autorisées.
(en l'absence d'initialisation ou de définition de la fonction next, alors toutes les valeurs du domaine de définition sont possibles)
- ▶ Les évolutions de b0 et b1 sont synchronisées (i.e. les fonctions next de b0 et b1 sont appliquées en même temps)

Un compteur modulo 4

```
MODULE main
```

```
VAR
```

```
    b0 : boolean;
```

```
    b1 : boolean;
```

```
ASSIGN
```

```
    init(b0) := FALSE ;
```

```
    init(b1) := FALSE ;
```

```
    next(b0) := !b0 ;
```

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


Un compteur modulo 4

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

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

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

toint permet de convertir les booléens en entiers

Un compteur modulo 4

```
MODULE main
```

```
VAR
```

```
  b0 : boolean;
```

```
  b1 : boolean;
```

```
ASSIGN
```

```
  init(b0) := FALSE ;
```

```
  init(b1) := FALSE ;
```

```
  next(b0) := !b0 ;
```

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

```
DEFINE
```

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

DEFINE définit des abréviations (sorte de macro définitions) : out n'est pas une *nouvelle* variable du modèle, et la taille des modèles n'est pas impactée par la définition de out.

Syntaxe des expressions

- ▶ Expressions arithmétiques
+, -, *, /, mod
=, !=, <, <=, >=
- ▶ Expressions booléennes
&, |, xor, !, ->, <->
- ▶ next(x) pour désigner la prochaine valeur de la variable x
- ▶ Expressions conditionnelles

case

 c1 : e1 ;

 c2 : e2 ;

 ...

 TRUE : en ;

esac

(se lit comme if c1 then e1 else if c2 then e2 ...
else en)

Expressions ensemblistes

- ▶ définition en extension
 $\{a, b, c\}$
- ▶ union de deux ensembles
 $E1 \text{ union } E2$
- ▶ une constante c est une abréviation pour l'ensemble singleton
 $\{c\}$
- ▶ Le résultat de `init()` ou de `next()` peut être un ensemble afin de capturer les situations de non-déterminisme

Les expressions utilisées dans les champs ASSIGN et DEFINE

- ▶ par défaut les expressions incluent les valeurs de l'état courant et de l'état suivant
- ▶ une seule règle assignation (définition) par variable

```
init(x) := TRUE ;
```

```
init(x) := FALSE ;
```

est donc interdit

- ▶ les définitions circulaires sont proscrites

```
x := !x ;
```

et

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

sont donc interdits

Une syntaxe alternative

ASSIGN

`init(st) := {ready,busy} ;`

`next(st) := ready ;`

`out := ready & busy ;`

est équivalent à

INIT

`state in {ready,busy}`

TRANS

`next(state)=ready`

INV

`out = ready & busy`

Risque : les propriétés exprimées dans INIT, TRANS et INV peuvent être inconsistantes

Plusieurs définitions de modules conjointes (utilisation de la notation dot)

```
MODULE modulo
VAR
  out : 0..9 ;
ASSIGN
next(out) := (out + 1) mod 10 ;
```

```
MODULE main
VAR
  m1 : modulo ;
  m2 : modulo ;
  sum : 0..18 ;
```

```
ASSIGN
  sum := m1.out + m2.out ;
```

La présence d'un module main est nécessaire.

Modules paramétrés

```
MODULE modulo(x)
VAR
  out : 0..9 ;
ASSIGN
next(out) := (out + x) mod 10 ;
```

```
MODULE main
VAR
  m1 : modulo(m2.out) ;
  m2 : modulo(m1.out) ;
  sum : 0..18 ;

ASSIGN
  sum := m1.out + m2.out ;
```

les paramètres formels peuvent être utilisés comme les autres variables dans les expressions


```
MODULE counter(reset)
VAR
    output : 0..7 ;
ASSIGN
    init(output) := 0;
    next(output) :=
        case
            reset = TRUE : 0 ;
            output < 7 : output +1 ;
            output = 7 : 0 ;
        esac ;
MODULE main
VAR
    reset : boolean ;
    dut : counter(reset) ;
ASSIGN
    init(reset) := TRUE ;
DEFINE
    cnt_out := dut.output ;
```

Composition synchrone des modules

```
MODULE cell(input)
VAR
  val : {rouge, orange, vert} ;
ASSIGN
  next(val) := input ;
```

```
MODULE main
VAR
  c1 : cell(c3.val) ;
  c2 : cell(c1.val) ;
  c3 : cell(c2.val) ;
```

A chaque étape, tous les modules avancent d'une transition

Composition asynchrone des modules

```
MODULE cell(input)
VAR val : {rouge, orange, vert} ;
ASSIGN
next(val) := input ;
FAIRNESS running
MODULE main
VAR
  c1 : process cell(c3.val) ;
  c2 : process cell(c1.val) ;
  c3 : process cell(c2.val) ;
```

Avec le mot clé `process` la composition est alors asynchrone (un seul processus avance à chaque étape).

Une variable `running` est associée implicitement à chaque module, et est à `TRUE` lorsque le processus est sélectionné/actif.

La contrainte `FAIRNESS running` est utilisée pour garantir l'équité entre les processus (i.e. seules les traces qui activent infiniment souvent chacun des processus seront considérées)

LTL ET CTL Spécifications

- ▶ Mot-clé LTLSPEC
- ▶ Opérateurs
X, F, G, U
- ▶ Exemple de déclaration de formule LTL
LTLSPEC b0 U (!b1)
- ▶ Mot-clé SPEC (pour les formules CTL)
- ▶ Quantificateurs A et E
- ▶ Exemple de déclaration de formule CTL
SPEC E(b0 U (!b1))

ATM : un exemple d'utilisation de propriétés LTL

```
MODULE main
VAR state: {welcome, enterPin, tryAgain, askAmount, thanksGoodbye}
    input: {cardIn, correctPin, wrongPin, ack, cancel, fundsOK}
ASSIGN
    init(state) := welcome;
    next(state) := case
        state = welcome & input = cardIn      : enterPin;
        state = enterPin & input = correctPin   : askAmount ;
        state = enterPin & input = wrongPin     : tryAgain;
        state = tryAgain & input = ack          : enterPin;
        state = askAmount & input = fundsOK     : thanksGoodbye;
        state = askAmount & input = problem    : sorry;
        state = enterPin & input = cancel       : thanksGoodbye;
        TRUE                                     : state;
    esac;
LTLSPEC F( G state = thanksGoodbye | G state = sorry ) ;
LTLSPEC (F G !(state = askAmount))
    -> F( G state = thanksGoodbye | G state = sorry ) ;
...
```

Exécuter NuSMV sous forme de commande

```
...$ NuSMV atm.smv
*** This is NuSMV 2.6.0 (compiled on Wed Oct 14 15:31:33 2015)
*** Enabled addons are: compass
*** For more information on NuSMV see <http://nusmv.fbk.eu>
*** or email to <nusmv-users@list.fbk.eu>.
*** Please report bugs to <Please report bugs to <nusmv-users@fbk
...
-- specification  F ( G state = thanksGoodbye |  G state = sorry
-- as demonstrated by the following execution sequence
Trace Description: LTL Counterexample
Trace Type: Counterexample
-> State: 1.1 <-
    state = welcome
    input = cardIn
-> State: 1.2 <-
    state = enterPin
    input = correctPin
-- Loop starts here
-> State: 1.3 <-
    state = askAmount
```

Exécuter NuSMV en mode interactif

Analyser le modèle et les traces pas à pas

```
...$ NuSMV -int atm.smv
```

```
...
```

```
***
```

```
NuSMV > go
```

```
NuSMV > pick_state -r
```

```
NuSMV > print_current_state -v
```

```
NuSMV > simulate -k 3
```

```
NuSMV > reset
```

```
NuSMV > help
```

```
add_property
```

```
alias
```

```
bmc_inc_simulate
```

```
bmc_pick_state
```

```
...
```

```
NuSMV > goto_state
```

```
NuSMV > show_trace ...
```

```
NuSMV > quit
```

Pour conclure

Sur le site web de NuSMV (<http://nusmv.fbk.eu/>), vous trouverez

- ▶ la version 2.6 à installer sur votre portable pour la séance de mardi 24 janvier
- ▶ NuSMV User Manual
- ▶ NuSMV Tutorial

et sur le site claroline, vous trouverez une copie des slides.