Introduction a la vérification formelle 2017-2018
TP de Model-checking : premiers éléments de correction

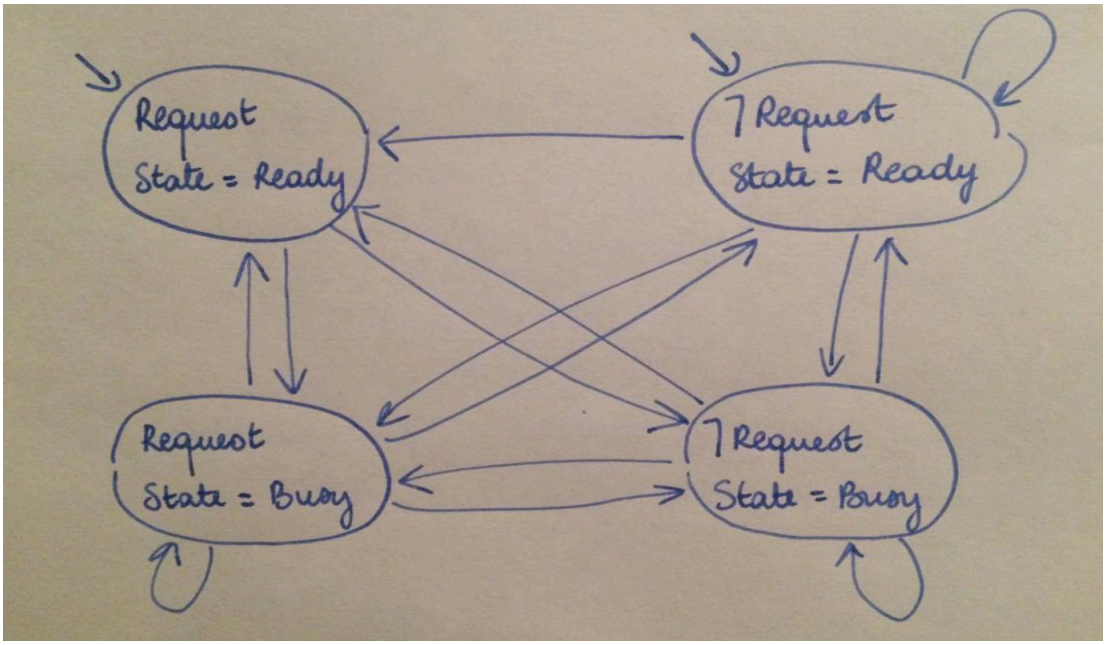
Exercice 1 : Prise en main

Considérons l'exemple SMV suivant (disponible dans le fichier exemple.smv) :

```
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)
```

1. Dessiner l'automate décrit par le modèle SMV et traduire en langage naturel la propriété de logique temporelle donnée dans le modèle SMV.



Remarquons que :

(a) le modèle a deux états initiaux, les deux états vérifiant $state = ready$.

(b) Seul l'état vérifiant *request* à vrai et $state = ready$ a deux successeurs seulement, les deux états avec $state = busy$. Tous les autres états ont tous les états comme successeurs (en particulier, comme rien n'est décrit en ce qui concerne la variable *request*, toutes les configurations d'évolution de la variable sont possibles).

2. Ouvrir le fichier `exemple.smv` en mode interactif (commande `NuSMV -int exemple.smv` en ligne de commande dans le répertoire contenant le fichier) et se référer au chapitre 3 du tutoriel Nusmv pour une première prise en main en mode interactif.

```
tintin-4:CodeSMV pascalelegall$ nusmv -int exemple.smv
*** This is NuSMV 2.6.0 (compiled on Wed Oct 14 15:31:33 2015)
*** Enabled addons
...
WARNING *** Please contact Sharad Malik (malik@ee.princeton.edu)      ***
WARNING *** for details.                                              ***

NuSMV >
```

Je vous laisse découvrir, mais objectivement, le mode interactif ne sera pas vraiment utile pour la suite.

3. Utiliser NuSMV pour vérifier la formule CTL présente dans le fichier (commande `NuSMV exemple.smv` en ligne de commande dans le répertoire contenant le fichier)

```
tintin-4:CodeSMV pascalelegall$ nusmv exemple.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>
...
WARNING *** Please contact Sharad Malik (malik@ee.princeton.edu)      ***
WARNING *** for details.                                              ***

-- specification AG (request -> AF state = busy)  is true
tintin-4:CodeSMV pascalelegall$
```

La dernière ligne "– ... is true" indique que la formule (CTL) est vérifiée pour le modèle. Cette formule signifie que "pour tout état futur vérifiant *request* à vrai, alors nécessairement après cet état, il existera un moment vérifiant $state = busy$". Cette propriété est trivialement vrai, car sous l'hypothèse *request* à vrai, alors soit l'état courant vérifie $state = busy$ et la propriété est vraie, soit l'état courant vérifie $state = ready$ et le prochain état vérifiera $state = busy$ et la propriété est vraie.

Proposer d'autres formules CTL et LTL vraies et fausses pour le modèle exemple.smv

```
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)

SPEC AG request

LTLSPEC X(request -> X !(state = busy))

LTLSPEC G F state = busy
```

J'ai formulé les formules au hasard, et donc,il est fort probable qu'elles soient vérifiées par le modèle.

Les formules CTL sont introduites par le mot clé SPEC tandis que les formules LTL sont introduites par le mot clé LTLSPEC.

Les opérateurs temporels sont dénotés par les lettres X (neXt), F (Future), G (Globally) et U (Until). Dans le cas de la logique CTL, ils sont associés à l'un des deux quantificateurs A (Always) et E (Exists).

Etonnamment, NuSMV ne traite pas les formules dans l'ordre donné dans le fichier. Seule la formule

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

est vraie. Les autres formules sont fausses. Un contre-exemple est alors fourni (sous forme d'un état pour les formules CTL, sous forme d'une séquence d'états pour les formules LTL). Pour les chemins contre-exemples, seules les variables qui changent d'un état courant à l'état suivant sont montrées, les autres étant implicites. Enfin, les chemins sont décrits à l'aide d'une boucle (ou lasso) qui se referme sur elle-même.

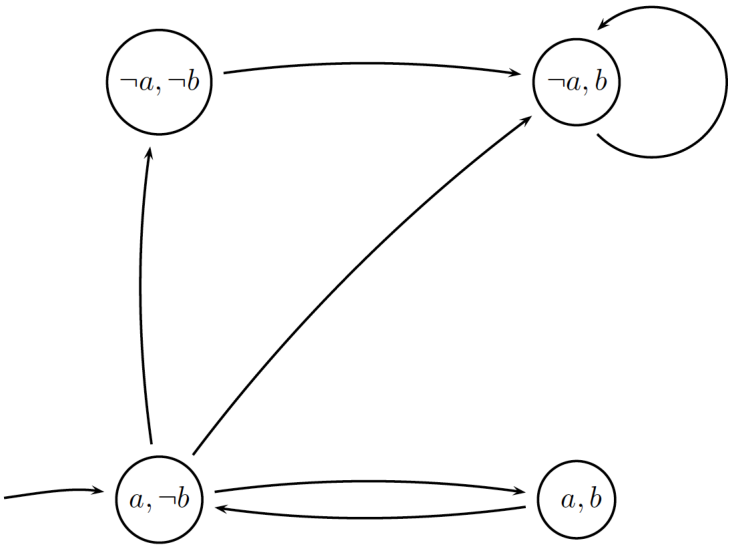
```
-- specification AG request  is false
-- as demonstrated by the following execution sequence
Trace Description: CTL Counterexample
Trace Type: Counterexample
  -> State: 1.1 <-
    request = FALSE
    state = ready
-- specification AG (request -> AF state = busy)  is true
-- specification  X (request ->  X !(state = busy))  is false
-- as demonstrated by the following execution sequence
Trace Description: LTL Counterexample
Trace Type: Counterexample
  -- Loop starts here
  -> State: 2.1 <-
    request = TRUE
    state = ready
  -> State: 2.2 <-
    state = busy
  -> State: 2.3 <-
    request = FALSE
  -> State: 2.4 <-
    request = TRUE
    state = ready
-- specification  G ( F state = busy)  is false
-- as demonstrated by the following execution sequence
Trace Description: LTL Counterexample
Trace Type: Counterexample
  -> State: 3.1 <-
    request = TRUE
```

```
  state = ready
-> State: 3.2 <-
  request = FALSE
  state = busy
-- Loop starts here
-> State: 3.3 <-
  state = ready
-- Loop starts here
-> State: 3.4 <-
-> State: 3.5 <-
```

Nu

Exercice 2 : Un premier modèle simple

Pour le modèle ci-dessous avec a et b variables booléennes (une flèche entrante signalant un état initial) :



1. Ecrire trois modèles NuSMV le modélisant
   (a) le premier à l'aide de deux variables booléennes, via la construction `ASSIGN` ;

```
MODULE main
VAR
 a : boolean ;
 b : boolean ;

ASSIGN
 init(a) := TRUE;
 init(b) := FALSE;

 next(a) := case
  a & b : TRUE;
  !a : FALSE;
  TRUE : {TRUE, FALSE};
 esac;
 next(b) := case
  a & b : FALSE;
  !a : TRUE;
  !b & next(a) : TRUE;
  TRUE : {TRUE, FALSE};
 esac;

LTLSPEC G a
LTLSPEC a U b
LTLSPEC a U X(a & !b)
LTLSPEC X !b & G(!a | !b)
LTLSPEC X (a & b) & F (!a & !b)
SPEC EF EG a
SPEC EF EG b
LTLSPEC F((G a) | (G b))
```

Comme discuté en cours, next(a) dépend des valeurs courantes de a et de b, tandis que next(b) dépend des valeurs courantes de a et de b ainsi que de next(a). Cette dernière dépendance est nécessaire pour définir correctement le modèle. Attention aux définitions circulaires qui sont proscrites : il n'est pas possible de définir next(b) en fonction de next(a) et next(a) en fonction de next(b).

(b) le second à l'aide d'une variable définie à l'aide d'un type énuméré de 4 valeurs et d'un champ DEFINE pour la définition des variables booléennes a et b, via la construction `ASSIGN` ;

```
MODULE main

VAR
 state : {a_b, nota_b, a_notb, nota_notb};

ASSIGN
 init(state) := a_notb;
 next(state) := case
  state = a_b : a_notb;
```

```
  state = nota_b : nota_b;
  state = a_notb : {a_b, nota_b, nota_notb};
  state = nota_notb : nota_b;
 esac;

DEFINE
 a := state = a_b | state = a_notb;
 b := state = a_b | state = nota_b;

LTLSPEC G a
LTLSPEC a U b
LTLSPEC a U X(a & !b)
LTLSPEC X !b & G(!a | !b)
LTLSPEC X (a & b) & F (!a & !b)
SPEC EF EG a
SPEC EF EG b
LTLSPEC F((G a) | (G b))
```

Il suffit d'introduire un type énuméré en charge de représenter l'ensemble des quatre états du modèles (ici `a_b, nota_b, a_notb, nota_notb`) et de définir la relation de transitions directement sur ces états. Dans un second temps des variables a et b sont introduites via le champ DEFINE, et définies en relation avec les états (`a_b, nota_b, a_notb, nota_notb`). Attention à ne pas ajouter les variables a et b comme de nouvelles variables d'états, sauf à vouloir augmenter inutilement le nombre d'états (qui serait alors de $4 \times 2 \times 2$ au lieu de 4).

(c) le troisième en définissant la relation de transitions entre états à l'aide de formules logiques, via les constructions `INIT` et `TRANS`.

```
MODULE main

VAR
 a : boolean;
 b : boolean;

INIT
 a & !b

TRANS
 (a & b -> next(a) & !next(b))
 & (a & !b -> next(a) & next(b) | !next(a) & next(b) | !next(a) & !next(b))
 & (!a & !b -> !next(a) & next(b)) & (!a & b -> !next(a) & next(b))

LTLSPEC G a
LTLSPEC a U b
LTLSPEC a U X(a & !b)
LTLSPEC X !b & G(!a | !b)
LTLSPEC X (a & b) & F (!a & !b)
SPEC EF EG a
```

```
SPEC EF EG b
LTLSPEC F((G a) | (G b))
```

Les états initiaux et la relation de transitions sont chacun à l'aide d'une formule logique qui les caractérise.

2. Pour chacun des modèles proposés et pour chacune des formules ci-dessous :

   (a) $\mathbf{G}\, a$

   (b) $a\, \mathbf{U}\, b$

   (c) $a\, \mathbf{U}\ \mathbf{X}\, (a \wedge \neg b)$

   (d) $\mathbf{X}\, \neg b \wedge\ \mathbf{G}\, (\neg a \vee \neg b)$

   (e) $\mathbf{X}\, (a \wedge b) \wedge\ \mathbf{F}\, (\neg a \wedge \neg b)$

   (f) $\mathbf{EF}\ \mathbf{EG}\, a$

   (g) $\mathbf{EF}\ \mathbf{EG}\, b$

   (h) $\mathbf{F}\, ((\ \mathbf{G}\, a) \vee (\ \mathbf{G}\, b))$

   Utiliser le model checker NuSMV pour indiquer si la formule est valide pour le modèle, en donnant pour chaque formules non valide, une trace qui l'invalide.

   Les trois modèles fournissent les mêmes résultats (voir ci-dessous, les "...." indiquent la suppression des traces contre-exemples fournies par NuSMV.

   Avec le premier modèle

```
-- specification EF (EG a)  is true
-- specification EF (EG b)  is true
-- specification  G a  is false
-- as demonstrated by the ....
-- specification (a U b)  is false
-- as demonstrated by the ....
-- specification (a U ( X (a & !b)))  is false
-- as demonstrated by the ....
-- specification ( X !b &  G (!a | !b))  is false
-- as demonstrated by the ....
-- specification ( X (a & b) &  F (!a & !b))  is false
-- as demonstrated by the .....
-- specification  F ( G a |  G b)  is true
```

   Avec le second modèle

```
-- specification EF (EG a)  is true
-- specification EF (EG b)  is true
-- specification  G a  is false
-- as demonstrated by the ....
-- specification (a U b)  is false
-- as demonstrated by the .....
-- specification (a U ( X (a & !b)))  is false
```

```
-- as demonstrated by the ....
-- specification ( X !b &  G (!a | !b))  is false
-- as demonstrated by the .....
-- specification ( X (a & b) &  F (!a & !b))  is false
-- as demonstrated by the .....
-- specification  F ( G a |  G b)  is true
```

Avec le troisième modèle

```
-- specification EF (EG a)  is true
-- specification EF (EG b)  is true
-- specification  G a  is false
-- as demonstrated by the ....
-- specification (a U b)  is false
-- as demonstrated by the .....
-- specification (a U ( X (a & !b)))  is false
-- as demonstrated by the .....
-- specification ( X !b &  G (!a | !b))  is false
-- as demonstrated by the ....
-- specification ( X (a & b) &  F (!a & !b))  is false
-- as demonstrated by the .....
-- specification  F ( G a |  G b)  is true
```

Pour finir, remarquez que comme souvent en informatique dans les activités de conception ou de programmation, il n'y a pas unicité d'écriture. Que cela concerne les modèles ou les propriétés, il existe différentes façons équivalentes d'écrire ces modèles ou formules.