Master FIIL Paris-Sud
Master 2

Software Model Checking
Année 2015-2016

Examen

20 novembre 2015
Durée : 2h

Exercice 1. — *Composition d'automates*

Le programme ci-dessous est un protocole d'exclusion mutuelle pour deux processus. Il est composé d'une variable partagée **turn** qui permet à chaque processus d'entrer en section critique.

```
integer turn <- 1
P
loop forever
p1:  non-critical section
p2: await turn = 1
p3: critical section
p4: turn <- 2

Q
loop forever
q1: non-critical section
q2: await turn = 2
q3: critical section
q4: turn <- 1
```

L'instruction **await p** permet d'attendre jusqu'à ce que la condition p devienne vraie.

Questions.

1. Faites l'automate produit de la composition parallèle de P et Q sachant que l'état initial est le vecteur d'état (p1,q1,1) (location de p, location de q, valeur de turn).
2. Est-ce que cet algorithme garantit l'exclusion mutuelle ? Justifier votre réponse.
3. Est-ce que cet algorithme garantit l'absence de famine ?

Exercice 2.

Soit la spécification suivante pour les opérations `alloc` et `free` de gestion dynamique de mémoire en C :

« (1) une zone mémoire est allouée par un `alloc` et désallouée par un `free`, (2) toute zone mémoire accédée en lecture / écriture doit être allouée au moment de l'accès, (3) désallouer une zone non allouée (au moment de la désallocation) est une erreur, (4) allouer une zone allouée (au moment de l'allocation) est une erreur, (5) toute zone allouée doit finir par être désallouée. »

On se concentre sur une zone mémoire particulière, et les opérations de `alloc`, `free` et `read/write` sur cette zone.

Questions :

1. Quelle partie de cette spécification est de la sûreté, quelle partie est de la vivacité ?
2. Dessinez un automate de Buchi pour la spécification complète.

Exercice 3.

On appelle ECTL l'extension de CTL avec les connecteurs $\mathbf{EF}^\infty$ et $\mathbf{AG}^\infty$, définis par $\mathbf{F}^\infty p$ est vraie ssi p est infiniment souvent vrai le long de l'exécution, et $\mathbf{G}^\infty p$ est vraie ssi p est toujours vrai à partir d'un moment de l'exécution. Remarque : on ne peut pas exprimer $\mathbf{EF}^\infty$ et $\mathbf{AG}^\infty$ dans CTL.

Questions.

1. Définissez $\models$ pour $\mathbf{F}^\infty$ et $\mathbf{G}^\infty$.
2. Définissez $\mathbf{F}^\infty$ et $\mathbf{G}^\infty$ en fonction de $\mathbf{X}, \mathbf{F}, \mathbf{G}, \mathbf{U}$.
3. Donnez les automates de Büchi pour $\mathbf{AF}^\infty$ et $\mathbf{AG}^\infty$.
4. Montrez qu'on peut exprimer $\mathbf{EG}^\infty$ et $\mathbf{AF}^\infty$ dans CTL usuel (donner une formule CTL équivalente).

Question Bonus. Montrer comment adapter l'algorithme de marquage de CTL pour ECTL (c'est à dire ajouter les cas pour $\mathbf{EF}^\infty$ et $\mathbf{AG}^\infty$).

Exercice 4. — Vérification de l'algorithme de Szymanski en Cubicle

Le but de cet exercice est de modéliser l'algorithme d'exclusion mutuelle de Boleslaw K. Szymanski [1988] donné ci-dessous. Pour rentrer en section critique, chaque processus `x` exécute le code suivant :

```
L0 : Bx := true
L1 : await (forall y. x <> y => not Sy) then Bx := false
L2 : Wx := true ; Sx := true ;
L3 : if (exists y. x <> y /\ not By /\ not Wy)
       then Sx := false ; goto 4
       else Wx := false ; goto 5
L4 : await (exists y. x <> y /\ Sy /\ not Wy) then Wx := false ; Sx := true;
L5 : await (forall y. x <> y => not Wy)
L6 : await (forall y. y < x => not Sy)
L7 : {Critical section}
     Sx := false ; goto 0
```

Explications : chaque processus `x` dispose de trois variables booléennes locales `Bx` (lire « la variable `B` de `x` »), `Sx` et `Wx`. Les lignes $\mathtt{L}_i$ correspondent à des points de programmes *atomiques*. Comme pour l'exercice 1, l'instruction `await p` signifie « attendre que `p` soit vraie pour continuer ». La variante `await p then i` attend que `p` soit vraie et, quand c'est le cas, exécute `i` de manière atomique.

Voici le début du code Cubicle pour modéliser un tel algorithme.

```
type loc = L0 | L1 | L2 | L3 | L4 | L5 | L6 | L7
array PC[proc] : loc
array B[proc] : bool
array S[proc] : bool
array W[proc] : bool

init (x) { PC[x] = L0 && S[x] = False && W[x] = False && B[x] = False }

transition t0 (x)
 requires { PC[x] = L0 }
 { PC[x] := L1; B[x] := True }
```

Questions.

1. Compléter le fichier Cubicle afin de modéliser complètement l'algorithme de Szymanski.
2. Écrire une propriété pour vérifier que cet algorithme est sûr, c'est-à-dire qu'il n'y a jamais deux processus en section critique en même temps.
3. Modifier le fichier Cubicle afin de vérifier que l'arrêt d'un processus, en dehors de la section critique, n'empêche pas un autre processus d'y entrer.