Examen Theorie et pratique de la concurrence (S2)

Lundi 27 juin 2011, 3h, aucun document autorisé

Ce sujet comporte deux exercices et un problème. Il est recommandé de commencer par les deux exercices. Le barème est donné à titre indicatif.

Exercice 1 : CCS { 6 points (3/2/2)

1. Construire les systèmes de transitions étiquetés associés aux termes CCS suivants :
   - $P_0 \stackrel{\text{def}}{=} a \cdot a \cdot \emptyset + b \cdot \emptyset + c \cdot P_0$
   - $P_1 \stackrel{\text{def}}{=} a \cdot (b \cdot \emptyset + \bar{c} \cdot P_1)$
   - $P_2 \stackrel{\text{def}}{=} a \cdot (b \cdot \emptyset + P_2)$
   - $P_3 \stackrel{\text{def}}{=} \bar{a} \cdot P_3$
   - $P_4 \stackrel{\text{def}}{=} (P_0 \mid P_3)$
   - $P_5 \stackrel{\text{def}}{=} (P_0 \mid P_3) \backslash \{a\}$
2. Est-ce que les états q_1 et r_1 des systèmes de transitions de la figure 1 sont bisimilaires ? Et s_1 et t_1 ? Et u_1 et v_1 ? A chaque fois, justifier votre réponse.

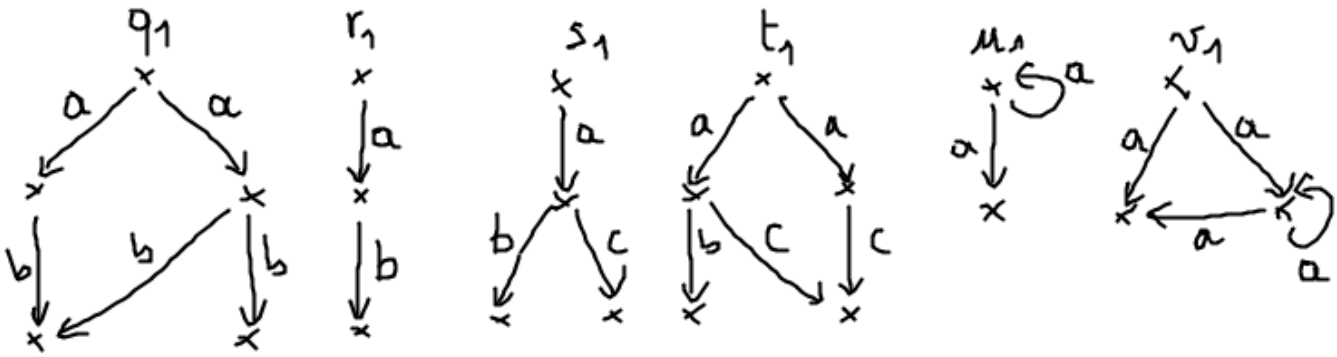

FIGURE 1 – Exemples de STE (exercice 1)

3. (a) Pour chaque formule de HML ci-dessous, dire si elle est vraie pour les états q_1, r_1, s_1, t_1, u_1 ou v_1 de la figure 1 ? (justifier vos réponses)
   - $\langle a \rangle (\langle b \rangle \top \wedge \langle c \rangle \top)$
   - $\langle a \rangle \langle a \rangle \langle a \rangle \top$
   - $\langle a \rangle ([b] \bot)$

   (b) Donner, si c'est possible, une formule de HML qui est vraie en r_1 mais fausse en s_1 ?

Probleme : Exclusion mutuelle { 10 points

Nous allons étudier deux algorithmes d'exclusion mutuelle pour n processus $P_0;:::;P_{n-1}$ utilisant une variable 'test-and-set".

Rappel : une variable "test-and-set" est une variable booléenne partagée disposant de deux opérations **atomiques** :

1. "test-and-set" : met la variable à VRAI et renvoie l'ancienne valeur,
   (si une variable "test-and-set" `lock` vaut FAUX, `test-and-set(lock)` met `lock` à VRAI et renvoie FAUX ; et si `lock` vaut VRAI, `test-and-set(lock)` met `lock` à VRAI et renvoie VRAI)
2. "reset" : met la variable à FAUX.

Algorithme 1. Le premier algorithme utilise un seul objet partagé : une variable "test-and-set" `lock` initialisée à FAUX. De plus, chaque processus utilise une variable locale booléenne `key`. Le processus P_i (avec $i = 0;:::;n-1$) est décrit par le code suivant :

```
Processus Pi:
variables locales:
         boolean : key
loop forever :
p1: section non critique (SNC)
p2: key := VRAI
p3: while (key) :
p4:           key := test-and-set(lock)
p5: section critique (SC)
p6: reset(lock)
```

1. Comment s'énoncent l'exclusion mutuelle pour cet algorithme ?
2. Expliquer cet algorithme.
3. Montrer que l'algorithme vérifie l'exclusion mutuelle.
   Pour cette question, on pourra utiliser (après l'avoir montré !) l'invariant $\mathcal{I}_1$ défini ci-dessous basé sur les deux ensembles de processus $\mathcal{P}_1$ et $\mathcal{P}_2$ suivants (NB : comme en cours, on notera $\texttt{key}^i$ la variable `key` du processus P_i) :
   - $\mathcal{P}_1$: contient les processus P_i en `p3` avec $\texttt{key}^i$ = FAUX (c'est-à-dire les processus sur le point d'entrer en section critique),
   - $\mathcal{P}_2$: contient les processus exécutant la section critique ou le post-protocole (c'est-à-dire en `p5` ou `p6`)

   L'invariant $\mathcal{I}_1$ est défini comme la conjonction des 4 propriétés suivantes :
   (a) $\mathcal{P}_1$ contient au plus un processus,
   (b) $\mathcal{P}_2$ contient au plus un processus,
   (c) ($\mathcal{P}_1$ est vide) $\vee$ ($\mathcal{P}_2$ est vide), et
   (d) $\texttt{lock} = VRAI \Leftrightarrow \Big((\mathcal{P}_1 \text{ est non vide}) \vee (\mathcal{P}_2 \text{ est non vide}) \Big)$.
4. Est-ce que cet algorithme garantit l'absence de famine ? Expliquer.

Algorithme 2. A présent, on considère un algorithme plus élaboré qui utilise les objets partagés suivants :

- `turn` : une variable partagée à valeur dans $\{0;:::;n-1\}$, elle est initialisée avec une de ces valeurs ;
- `lock` : une variable "test-and-set" initialisée à FAUX ;
- `Waiting[0...n-1]` : un tableau de booléens partagé, initialisé avec des FAUX.

De plus, chaque processus utilise deux variables locales : `key` et `aux`.
Pour ce second algorithme, le processus P_i (avec $i = 0;:::;n-1$) est décrit par le code suivant :

```
Processus Pi:
variables locales:
         int: aux
         boolean: key
loop forever :
p1: section non critique (SNC)
p2: Waiting[i] := VRAI
p3: key := VRAI
p4: while (Waiting[i] and key) :
p5:           key := test-and-set(lock)
p6: section critique (SC)
p7: Waiting[i] := FAUX
p8: if turn == i
p9:                 then: aux := (turn+1) % n
p10:                else: aux := turn
p11: if Waiting[aux]
p12:                then: turn := aux
p13:                      Waiting[aux] := FAUX
p14:                else:  turn := (aux+1) % n
p15:                       reset(lock)
```

5. Comment s'énoncent l'exclusion mutuelle et l'absence de famine en LTL pour cet algorithme ?
6. Expliquer comment fonctionne cet algorithme.
7. Montrer que l'algorithme vérifie l'exclusion mutuelle.

   Pour cette question, on pourra utiliser (après l'avoir montré !) l'invariant $\mathcal{I}_2$ défini ci-dessous basé sur les deux ensembles de processus $\mathcal{P}'_1$ et $\mathcal{P}'_2$ suivants :
   - $\mathcal{P}'_1$: contient les processus P_i en (1) `p3`, `p4` ou `p5` avec `Waiting[i]` = FAUX, ou en (2) `p4` ou `p5` avec $\texttt{key}^i$ = FAUX.
   - $\mathcal{P}'_2$: contient les processus exécutant la section critique ou leur post-protocole (c'est-à-dire en `p6`, `p7`, ... ou `p15`)

   L'invariant $\mathcal{I}_2$ est défini comme la conjonction des 4 propriétés suivantes :

   (a) $\mathcal{P}'_1$ contient au plus un processus,

   (b) $\mathcal{P}'_2$ contient au plus un processus,

   (c) ($\mathcal{P}'_1$ est vide) $\vee$ ($\mathcal{P}'_2$ est vide), et

   (d) $\texttt{lock} = VRAI \Leftrightarrow \Big((\mathcal{P}'_1 \text{ est non vide}) \vee (\mathcal{P}'_2 \text{ est non vide}) \Big)$.

8. Montrer qu'il n'y a pas d'interblocage.
9. Sous hypothèse d'équité entre processus et en supposant que toute section critique termine et conduit à l'exécution du post-protocole, montrer que l'algorithme vérifie l'absence de famine. Et, supposons que le processus P_j exécute son pré-protocole :
   (a) Combien de fois un certain processus P_i peut accéder à la section critique avant P_j ?
   (b) Combien de fois des processus peuvent accéder à la section critique avant P_j ?

Exercice 2 : Algorithme concurrent et diagramme d'etats - 4 points

On considère les deux processus `P1` et `P2` décrits à la figure 2.

1. Dessiner le diagramme d'états de cet algorithme où `A` et `B` sont des variables partagées initialisées avec : `A:=1` et `B:=10`.
2. Est-ce que toutes les exécutions finissent dans un état de la forme `(p5,q5,-,-)` ?
3. Est-ce que les formules LTL suivantes sont vérifiées par les exécutions (équitables) partant de l'état initial `(p1,q1,A=1,B=10)` : (justifier vos réponses)
   - $\Diamond\,(\texttt{p5})$
   - $\Big(\Box\,\neg\texttt{p2}\Big) \Rightarrow \Big(\Diamond\,(\texttt{B} = 14)\Big)$

```
-- Processus P1                          -- Processus P2
p1: if (A >= 2) :                        q1: A := A + 1
p2:      B := B * 2                      q2: B := B + 1
P3: await (B > 10 and B<=20)             q3: await (A <= 1)
P4: A := A - 1                           q4: B := B +3
p5: end                                  q5: end
```

FIGURE 2 – Algorithme de l'exercice 2