Université Paris Diderot – Master 1 II

Théorie et pratique de la concurrence

Corrigé du partiel du 30 mars 2010

Durée : 1h30. Tous les documents sont autorisés.

Question 1 :

1. Si `K` est positif, quelles sont les valeurs possibles de `n` ?
   Remarquez que le programme ne termine pas. Les valeurs de n *durant l'exécution du programme changent tant que les processus exécutent la première branche de la boucle* `do`*. A cause de l'entrelacement des instructions, les valeurs de* n *pendant* ***une*** *exécution varient en fonction de l'exécution. Par exemple, pour* `K=1`*, il y a une exécution tel que* n *prend que les valeurs* 0 *et* -1 *et une autre exécution avec des valeurs* 0 *et* 1*. Si on raisonne sur l'ensemble des exécutions du programme, on peut construire deux exécutions telles que* n *atteint les valeurs extrêmes* `K` *(le processus* P *prend la main et exécute entièrement la première partie de sa boucle* `do`*) et* $-$`K` *(pareil mais pour* Q*). Pour toute valeur entre ces valeurs extrêmes on peut construire une exécution du programme (même pour* 0 *!). Donc les valeurs possibles se trouvent dans l'ensemble* $\{-K, \ldots, K\}$*.*
2. Ecrire en LTL les propriétés suivantes en explicitant les propositions atomiques utilisées :
3. Pour chaque propriété ci-dessous, indiquer sa classe (vivacité, sureté ou les deux) et sa valeur de vérité pour le programme considéré.
   (a) "`n` ne prend jamais la valeur `2K`"
   $\Box\neg(n = 2K)$ *(le prédicat atomique sur les états est ici* $n = 2K$*). Il s'agit d'une propriété de sureté qui, d'après le premier point est fausse.*
   (b) "`n` prend la valeur 0 un nombre fini de fois"
   $\Diamond\Box\neg(n = 0)$ *(le prédicat atomique sur les états est ici* $n = 0$*). Il s'agit d'une propriété de sureté qui, d'après le premier point est fausse car il y a une exécution dans laquelle* n *aura la valeur* 0 *à la fin de toutes les affectations des deux boucles, donc une infinité de* 0 *car le programme ne termine pas.*
   (c) "si `n` vaut 1, il prendra la valeur 2 un jour"
   $\Box((n = 1) \Rightarrow \Diamond(n = 2))$*. Il s'agit d'une propriété de vivacité qui est fausse car on peut trouver une exécution du programme pour laquelle la propriété n'est pas vraie (exécuter d'abord toutes les affectations de* P *puis celles de* Q*).*
   (d) "`n` prend la valeur 1 après avoir eu la valeur 0"
   La propriété est ambiguë car il n'est pas clair s'il s'agit de "juste après" ou "que après", "toujours" ou "une fois". Par exemple, pour "toujours n *prend la valeur* 1 *juste après avoir eu la valeur* 0*" on a la formule* $\Box((n = 0) \Rightarrow \bigcirc(n = 0))$ *qui est une propriété de sureté (pas autre chose que 0) et de vivacité (prend une valeur) qui est fausse.*
   (e) "`n` prend toutes les valeurs entre 0 et 2"
   Pareil, il faut préciser ce qu'on comprend : que les valeurs entre 0 *et* 2 *(*$\Box((n = 0) \vee (n = 1) \vee (n = 2))$*) ou ces valeurs sont parmi celles prises (*$\Diamond(n =$

$0) \wedge \Diamond(\boldsymbol{n} = 1) \wedge \Diamond(\boldsymbol{n} = 2)$*). Les deux interprétations sont fausses pour toute valeur de* K*.*

Question 2 : (7 points)

Supposons qu'une machine fournit l'instruction atomique suivante :

```
TS(int common, int local) :
      atomic { local = common;
               common = 1 }
```

Cette instruction est utilise pour implementer l'exclusion mutuelle entre deux processus comme suit :

```
int common = 0; /* variable partagee */

-- Processus 1:                    -- Processus 2:
int local1;                        int local2;
loop forever :                     loop forever :
l1:  section NC                    l1:  section NC
     repeat                             repeat
l2:    TS(common, local1);         l2:    TS(common, local2);
l3:  until local1 = 0;             l3:  until local2 = 0;
l4:  section critique              l4:  section critique
l5:  common = 0;                   l5:  common = 0;
```

1. Expliquer (10 lignes maximum) comment fonctionne cet algorithme.
   Il s'agit d'un protocole d'exclusion mutuelle avec mémoire partagée. Le pré-protocole est constitué des lignes l2 et l4 et le post-protocole de la ligne l5. Le pré-protocole consiste à appliquer l'opération atomique TS qui affecte la variable partagée `common` *à 1 juste avant d'avoir mémorisé sa valeur dans la variable locale du processus. Ainsi, le processus qui exécute cette opération exprime son désir d'entrer dans la section critique en affectant* `common` *à 1. La valeur 1 de* `common` *indique donc la présence d'un processus dans son pré-protocole ou dans la section critique. L'algorithme permet uniquement au processus ayant affecté en premier* `common` *à 1 d'entrer dans la section critique, car c'est celui qui voit la valeur 0 de* `common` *dans sa variable locale. Le post-protocole permet de changer la valeur de* `common` *pour signaler que la section critique est libre.*
2. Montrer, en raisonnant sur l'algorithme, que cet algorithme satisfait les propriétés d'exclusion mutuelle et d'absence d'inter-blocage.
   Deux solutions :
   - *En utilisant le diagramme d'états ci-dessous, on remarque q'il est impossible d'être à la fois en 1@l4 et 2@l4 :*

	1@l1	*1@l2*	*1@l3*	*1@l4*	*1@l5*
2@l1	*(?, ?,0)*→	*(?, ?,0)*→	*(0, ?,1)*→	*(0, ?,1)*→	*(0, ?,1)*→ *debut ligne*
	↓	↓	↓	↓	↓
2@l2	*(?, ?,0)*→	*(?, ?,0)*→	*(0, ?,1)*→	*(0, ?,1)*→	*(0, ?,1)*→ *debut ligne*
	↓	↓	↕	↓	↓
2@l3	*(?,0,1)*→	*(?,0,1)*↔	*(1,0,1)/(0,1,1)*	*(0,1,1)*	*(0,1,1)*→*(?, ?,0) line above*
	↓	↓			
2@l4	*(?,0,1)*→	*(?,0,1)*→	*(1,0,1)*		
	↓	↓			
2@l5	*(?,0,1)*→	*(?,0,1)*→	*(1,0,1)*		
	↓	↓	↓		
	debut col.	*debut col.*	*(?, ?,0) debut col.*		

– *En raisonnant sur l'algorithme, on peut montrer les invariants suivants :*

$$I_1 : \quad (common = 1) \Leftrightarrow (\texttt{1@l3-5} \vee \texttt{2@l3-5})$$
$$I_2 : \quad (common = 0) \Leftrightarrow (\texttt{1@l1-2} \wedge \texttt{2@l1-2})$$
$$I_3 : \quad (\texttt{1@l3} \wedge \texttt{2@l3}) \Rightarrow (common = 1 \wedge (\texttt{local1} = 1 \vee \texttt{local2} = 1))$$

Les premiers deux invariants sont faciles à montrer en examinant les différentes façons d'atteindre les lignes indiquées. Le troisième est une propriété de TS. La preuve de l'exclusion mutuelle est dérivée de ces invariants par absurde : si les deux processus sont à `l4` *c'est que leurs variables locales dont à 0, or* I_3 *l'interdit.*

La propriété d'interblocage résulte de même soit du diagramme, soit en remarquant que vu que `common` *a la valeur 0 avant d'exécuter TS (par* I_2*), l'opération TS affecte inévitablement l'un de local à 0.*

3. Qu'en est-il des propriétés d'absence de famine et d'attente borne ?

   Il peut y avoir de la famine même si le scheduleur est équitable (faiblement ou fortement) car l'un de processus peut toujours prendre le processeur avant l'autre pour exécuter TS. Un exemple d'exécution : $(1@l1, 2@l1) \rightarrow (1@l2, 2@l1) \rightarrow (1@l2, 2@l2) \rightarrow (1@l3, 2@l2) \rightarrow (1@l3, 2@l3) \rightarrow (1@l4, 2@l3) \rightarrow (1@l4, 2@l2) \rightarrow (1@l5, 2@l2) \rightarrow (1@l1, 2@l2) \rightarrow (1@l2, 2@l2)$ *(cycle).*

4. Y-a-t-il un processus privilégié ?

   Le protocole est complètement symétrique : l'opération TS qui donne l'accès à la SC ne dépend pas des identificateurs des processus. Donc pas de processus privilégié.

Question 3 : (7 points)

Soit l'algorithme suivant :

```
semaphore S=1, T=0;

active proctype P() {
p1: wait(S);
p2: printf("p");
p3: signal(T)
}

active proctype Q() {
q1: wait(T);
q2: printf("q");
q3: signal(S);
}
```

1. Quels sont les affichages possibles de ce programme ?

   La seule exécution possible est $(p1, q1, S = 1, T = 0) \rightarrow (p2, q1, S = 0, T = 0) \rightarrow (p3, q1, S = 0, T = 0) \rightarrow (\_, q1, S = 0, T = 1) \rightarrow (\_, q2, S = 0, T = 0) \rightarrow (\_, q3, S = 0, T = 0) \rightarrow (\_, \_, S = 1, T = 0)$ *donc affichage pq.*

2. Quels sont ces affichages si on commente l'instruction `wait(S)` ?

   La seule exécution possible est $(p2, q1, S = 1, T = 0) \rightarrow (p3, q1, S = 1, T = 0) \rightarrow (\_, q1, S = 1, T = 1) \rightarrow (\_, q2, S = 1, T = 0) \rightarrow (\_, q3, S = 1, T = 0) \rightarrow (\_, \_, S = 2, T = 0)$ *donc affichage pq.*

3. Quels sont ces affichages si on commente l'instruction `wait(T)` ?

   Deux affichages sont possibles pq (similaire ci-dessus) et qp par $(p2, q2, S = 1, T = 0) \rightarrow (p2, q3, S = 1, T = 0) \rightarrow ....$

4. Y-a-t-il un changement de comportement si les deux sémaphores sont des sémaphores binaires ?

   Pas de changement dans les affichages, mais pour les deux derniers cas, le `signal(S)` *de Q se bloquera car S est déjà à 1.*

5. Proposer une implementation en Promela pour les sémaphores (non binaires) généralisés.

   Exercice donné mais pas corrigé en TP.