

## Théorie et pratique de la concurrence – Master 1 II

### TP 4 : Sections critiques (preuves)

[www.liafa.jussieu.fr/~sighirea/cours/concur/](http://www.liafa.jussieu.fr/~sighirea/cours/concur/)

#### **Exercice 1 :**

*Variante de l'algorithme de Dekker*

Soit la variante suivante de l'algorithme de Dekker, qu'on appellera Flekker où les instructions des lignes p6 et q6 ont été changées :

```
boolean D1 := False // variables partagées
boolean D2 := False
int turn := 1

-- Processus P1
loop forever :
p1: section NC
p2: D1 := True
p3: while (D2 == True) :
p4:   if (turn == 2)
p5:     D1 := False
p6:     await (D2==False) /* changée */
p7:     D1 := True
p8: section critique
p9: turn := 2
p10: D1 := False

-- Processus P2
loop forever :
q1: section NC
q2: D2 := True
q3: while (D1 == True) :
q4:   if (turn == 1)
q5:     D2 := False
q6:     await (D1==False) /* changée */
q7:     D2 := True
q8: section critique
q9: turn := 1
q10: D2 := False
```

1. Modélisez l'algorithme ci-dessus en Promela et montrez avec Spin qu'il satisfait la propriété d'exclusion mutuelle.
2. Prouvez formellement que l'algorithme satisfait la propriété d'exclusion mutuelle.
3. L'algorithme de Dekker satisfait la propriété d'absence de famine et d'attente bornée sous hypothèse d'équité des processus et en supposant que toute section critique termine. Qu'en est-il de l'algorithme Flekker ? Justifiez votre réponse en utilisant Spin.

**Exercice 2 :**

*Algorithme de Doran et Thomas*

Prover la correction de la variante suivante de l'algorithme de Dekker :

```
boolean D1 := False // variables partagées
boolean D2 := False
int turn := 1

-- Processus P1
loop forever :
p1: section NC
p2: D1 := True
p3: while (D2 == True) :
p4:   if (turn == 2)
p5:     D1 := False
p6:     await (turn==1)
p7:     D1 := True
p8:   await (D2 == False)
p9: section critique
p10: D1 := False
p11: turn := 2

-- Processus P2
loop forever :
q1: section NC
q2: D2 := True
q3: while (D1 == True) :
q4:   if (turn == 1)
q5:     D2 := False
q6:     await (turn==2)
q7:     D2 := True
q8:   await (D1 == False)
q9: section critique
q10: D2 := False
q11: turn := 1
```

Correction 1 :  
Question 1 :

*Variante de l'algorithme de Dekker*

```
#define true 1
#define false 0
int turn = 1;
bool D1 = false;
bool D2 = false;

active proctype P1() {
do :: true ->
p1: printf("MSC: P1 SNC\n");
    D1 = true;
p3: do :: (D2 == false) -> break
    :: (D2 == true) ->
        if :: turn == 1
            :: turn == 2 ->
                D1 = false;
p6: /* condition originale (turn == 2); */
        /* changee en (D2 == false); */
        /* explicitee pour voir la famine: */
        do :: (D2==true) -> printf("MSC: P1 await\n")
        :: else ->
            break
        od;
        D1 = true
    fi
od;
p8: printf("MSC: P1 SC\n");
    turn = 2;
    D1 = false
od
}

active proctype P2() {
do :: true ->
q1: printf("MSC: P2 SNC\n");
    D2 = true;
q3: do :: (D1 == false) -> break
    :: (D1 == true) ->
        if :: turn == 2
            :: turn == 1 ->
                D2 = false;
q6: /* condition initiale (turn == 2); */
        /* changee en (D1 == false); */
        /* explicitee pour voir la famine: */
        do :: (D1==true) -> printf("MSC: P2 await\n")
        :: else -> break
        od;
        D2 = true
    fi
od;
q8: printf("MSC: P2 SC\n");
```

```

    turn = 1;
    D2 = false
od
}

/*
#define p1p3 P1@p3
#define p1p8 P1@p8
#define p2q3 P2@q3
#define p2q8 P2@q8
([] <> p1p3) -> [] (p1p3 -> <> p1p8)
*/

```

**Question 2 :** On fait le même raisonnement que pour l’algorithme de Dekker. Ainsi, on a les invariants :

$$\begin{aligned}
 & \text{turn}==1 \vee \text{turn}==2 \\
 & p3-5 \vee p8-10 \Leftrightarrow D1==\text{True} \\
 & pq-5 \vee q8-10 \Leftrightarrow D2==\text{True}
 \end{aligned}$$

Alors, pour montrer  $\Box \neg(p8 \wedge q8)$  on procède par induction en montrant que  $p8 \wedge q8$  est fausse dans tout état.

La propriété est fausse dans l’état initial.

Supposons que  $p8 \wedge q8$  est fausse dans un état  $s$  ; quelles sont les instructions atomiques amèneront l’algorithme dans un état  $s'$  dans lequel  $p8 \wedge q8$  est vraie ? Déjà, comme il s’agit d’une seule instruction (qui changera un seul état — celui de P1 ou celui de P2), il résulte que soit  $p8$  soit  $q8$  est vraie en  $s$ . Alors, il y a deux instructions : à  $p3$  sortir de la boucle et à  $q3$  sortir de la boucle. Comme l’algorithme est symétrique, considérons une des possibilités.

Donc soit  $s$  tel que  $q8$  est vraie en  $s$  et l’instruction à la ligne  $p3$  qui fait le test de la boucle `while` et le saut à la ligne  $p8$  (on suppose que cette instruction est atomique, mais un raisonnement similaire peut être fait sinon). Si à  $p3$  P1 sort de la boucle `while`, c’est qu’il a trouvé `D2==False`. Mais selon les invariants ci-dessus, quand P2 est à `q8,D2==True`. Donc il est impossible que  $p8 \wedge q8$  soit vraie.

**Question 3 :** la réponse est négative pour les deux propriétés si les hypothèses sur l’équité restent les mêmes que dans l’algorithme de Dekker, c’est-à-dire qu’on suppose une équité faible. En effet, en testant la propriété d’absence de famine ( $\Box (p2 \rightarrow \langle \rangle p8)$ ), Spin nous indique une exécution (contre-exemple) qui ne satisfait pas cette propriété même en présence d’une équité faible (option “*Weak fairness*” activée à la vérification) : Dans cette exécution, le processus P1 est à la ligne `p6` et le processus P2 accède à la section critique (en exécutant les lignes `q1-q10`) en boucle.

Un examen superficiel de ce contre-exemple nous amènerait à conclure que Spin implémente de façon erronée l’équité faible. En effet, l’hypothèse d’équité faible dit que,

si une instruction est continuellement activée pendant une exécution, alors elle sera forcément exécutée,

or, l’exécution indiquée par Spin contient une instruction (à la ligne `p6`) continuellement activée mais apparemment jamais exécutée.

Cette analyse est fautive. Dans la boucle infinie (*cycle*) indiquée par Spin, le processus P1 exécute l’instruction à la ligne `p6` mais uniquement au moment où le processus P2 a mis son

drapeau D2 à vrai (ligne q7). Dans ce cas, l'exécution de l'instruction `await (D2==False)` retourne P1 à la ligne p6. L'hypothèse d'équité faible est donc bien implémentée par Spin, mais la façon de modéliser l'algorithme n'est pas suffisamment explicite pour s'y rendre compte.

En effet, si au lieu de traduire l'instruction `await(D2==False)` en `(D2==false)` on utilise la forme explicite de l'attente active qui utilise une boucle `do :: (D2==false) -> skip :: else -> break od`, il est possible d'observer dans la boucle infinie donnée dans le contre-exemple que le processus P1 exécute une instruction dans cette boucle, tout en revenant à la ligne p6.

Il faut donc utiliser une hypothèse d'équité plus forte que l'équité faible afin d'assurer l'absence de famine dans cette version de l'algorithme.

### Correction 2 :

*Algorithme de Doran et Thomas*

**Exclusion mutuelle :** La preuve est similaire à celle donnée pour l'algorithme de Dekker car aucune affectation des variables est faite dans cette variante de l'algorithme. Donc les invariants auxiliaires restent les mêmes et on prouve par induction que  $\Box \neg (p9 \wedge q9)$ .

**Absence de famine :** La preuve est similaire à celle du cours sur un point : montrer qu'il est impossible de rester bloqué à la ligne p6 à cause de l'attente sur une condition.

Comme on introduit une autre instruction `await` à la ligne p8, il faut montrer aussi l'absence de blocage. Le raisonnement est similaire : pour rester bloqué, il faut qu'il existe une séquence d'exécution dans laquelle la condition `D2==False` est infiniment souvent fausse, mais pas toujours fausse. Autrement dit, il faut trouver une séquence infinie d'exécution dans laquelle D2 est `True` infiniment souvent mais pas toujours fausse. Or, on remarque que à la ligne p8, D2 a la valeur `True`. Donc P2 ne peut pas rentrer dans sa section critique à l'infini, mais il doit exécuter les instructions aux lignes q3--q6 et se bloquer sur l'instruction à la ligne q6 (car `turn` est 1!). Ceci étant la seule exécution possible, par équité faible, P1 doit avancer, donc on a l'absence de blocage à cette ligne aussi.