Théorie et pratique de la concurrence – Master 1 Informatique
TD 1 : Concurrence et exclusion mutuelle

Exercice 1 : *Algorithmes concurrents et leur propriétés*

Soit une fonction f pour laquelle il existe une valeur i telle que $f(i) = 0$. Les algorithmes concurrents ci-dessous cherchent une telle valeur i (appelée aussi le point zéro) en divisant l'espace de recherche en deux parties : les points zéro positifs $i \geq 0$ et les points zéro négatifs $i < 0$.

Un algorithme est correct si, pour toutes les exécutions, les deux processus terminent après que l'un d'entre eux a trouvé un point zéro. Pour chaque algorithme, prouvez qu'il est correct en utilisant les diagrammes d'états ou trouvez une exécution incorrecte.

Algorithme Zéro A.

```
boolean found;
process P()
    integer i:=0;
p1: found:=false;
p2: while not found
p3:   i:=i+1
p4:   found:=(f(i)==0)
```

```
process Q()
    integer j:=1;
q1: found:=false;
q2: while not found
q3:   j:=j-1
q4:   found:=(f(j)==0)
```

Algorithme Zéro B.

```
boolean found:=false
process P()
    integer i:=0;
p1: while not found
p2:   i:=i+1
p3:   found:=(f(i)==0)
```

```
process Q()
    integer j:=1;
q1: while not found
q2:   j:=j-1
q3:   found:=(f(j)==0)
```

Algorithme Zéro C.

```
boolean found:=false
process P()
    integer i:=0;
p1: while not found
p2:   i:=i+1
p3:   if (f(i)==0)
p4:     found:=true
```

```
process Q()
    integer j:=1;
q1: while not found
q2:   j:=j-1
q3:   if (f(j)==0)
q4:     found:=true
```

Dans les algorithmes suivants, on utilise la construction `await B then R` qui bloque l'exécution de `R` jusqu'à ce que la condition booléenne `B` soit vraie. De plus, l'exécution de `R` est faite atomiquement après le test de `B`. Si la condition `B` est évaluée (atomiquement) à faux, alors le processus qui exécute `await` est suspendu et il peut réessayer plus tard d'exécuter `await`.

Algorithme Zéro D.

```
boolean found:=false
integer turn:=1
process P()                                  process Q()
    integer i:=0;                                integer j:=1;
p1: while not found                          q1: while not found
p2:   await turn==1 then turn:=2             q2:   await turn==2 then turn:=1
p3:   i:=i+1                                 q3:   j:=j-1
p4:   if (f(i)==0)                           q4:   if (f(j)==0)
p5:     found:=true                          q5:     found:=true
```

Algorithme Zéro E.

```
boolean found:=false
integer turn:=1
process P()                                  process Q()
    integer i:=0;                                integer j:=1;
p1: while not found                          q1: while not found
p2:   await turn==1 then turn:=2             q2:   await turn==2 then turn:=1
p3:   i:=i+1                                 q3:   j:=j-1
p4:   if (f(i)==0)                           q4:   if (f(j)==0)
p5:     found:=true                          q5:     found:=true
p6: turn:=2                                  q6: turn:=1
```

Exercice 2 : *Sûreté et vivacité*

Pour chacune des phrases suivantes, indiquez quel type de propriété (sûreté ou vivacité) elle exprime en identifiant bien la partie qui y est intéressante :

1. Au plus 5 personnes sont dans l'ascenseur à un moment donné.
2. Les patrons sont servis dans l'ordre de leur arrivée.
3. Le coût de la vie ne décroît jamais.
4. Toute bonne chose a une fin.
5. Un livre s'améliore à chaque lecture.
6. Ce qui croit doit décroître.
7. Si deux processus attendent pour leur section critique, exactement un va y entrer.
8. Au plus une personne doit parler à un moment, les autres doivent écouter.
9. Si une interruption arrive, alors un message sera affiché.
10. Si une interruption arrive, alors un message sera affiché dans une seconde.

Exercice 3 : *Algorithme de Manna et Pnueli*

Soit la solution suivante pour l'implémentation d'une section critique (Manna-Pnueli) :

```
int wantP = 0, wantQ = 0; // variable partagée

-- Processus P                          -- Processus Q
loop forever:                           loop forever:
    section NC                              section NC
    if (wantQ = -1)                         if (wantP = -1)
        wantP := -1                             wantQ := 1
    else                                    else
        wantP := 1                              wantQ := -1
    await wantP != wantQ                    await wantP != -wantQ
    section critique                        section critique
    wantP := 0                              wantQ := 0
```

1. Construisez le diagramme d'états de cet algorithme.
2. En utilisant ce diagramme, indiquez quelles sont les propriétés de correction (exclusion mutuelle, absence d'inter-blocage, absence de famine, attente bornée) satisfaites par cet algorithme.
3. Est-il possible de modifier cet algorithme afin de satisfaire les propriétés ci-dessus non satisfaites ?

Exercice 4 : *Mutex avec instructions atomiques*

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

```
flip(int lock) :
      atomic { lock = (lock+1) % 2; /* inverse le lock */
               return (lock) }      /* renvoie sa nouvelle valeur */
```

Un programmeur propose la solution suivante pour l'implementation d'une section critique pour deux processus :

```
int lock = 0; /* variable partagee */
-- Processus i: /* i est soit 1 soit 2 */
loop forever :
     section NC

     while (flip(lock) != 1) :
         await lock == 0
     section critique
     lock = 0
```

1. Expliquer-lui pourquoi sa solution ne marche pas en lui montrant une trace d'exécution dans laquelle les deux processus se retrouvent en même temps dans leur section critique.
2. Il vous propose de garder sa solution mais de changer dans la machine l'instruction atomique `flip` : au lieu d'une addition modulo 2, utiliser une addition modulo 3.

   Cette nouvelle solution est-elle correcte pour deux processus ?

   Justifier votre réponse (si la réponse est positive donner un invariant, sinon exhiber une trace d'exécution).
3. Pour cette dernière solution, commenter informellement les propriétés d'absence d'inter-blocage et de famine ainsi que celle d'attente bornée.