Concurrence – Master 1
TD 2 : Concurrence et synchronisation

Exercice 1 : Soit le programme concurrent suivant :

```
int x=0, y=0;
proctype P1() {
   x = x+1; y = y+x
}
proctype P2() {
   x = x+2; y = y-x
}
init { run P1(); run P2(); }
```

1. Si les affectations sont implémentées comme des actions atomiques, combien d'exécutions sont possibles ? Quels sont les valeurs finales possibles de x et y ?
2. Mêmes questions si les affectations sont implémentées par trois actions atomiques (charger un registre, ajouter ou soustraire une valeur de ce registre, écrire le registre dans la mémoire).

Exercice 2 : Pour chacune des **preuves** suivantes, étudier leur interférence avec la precondition et l'instruction suivantes :

$$\underbrace{\{x \geq 4\}}_{pre(a)} \quad \underbrace{\textbf{atomic}\{x = x - 4\}}_{a}$$

1. $\{x \geq 0\}$ $\quad$ **atomic**$\{x = x + 5\}$ $\quad$ $\{x \geq 5\}$
2. $\{x \geq 0\}$ $\quad$ **atomic**$\{x = x + 5\}$ $\quad$ $\{x \geq 0\}$
3. $\{x \geq 10\}$ $\quad$ **atomic**$\{x = x + 5\}$ $\quad$ $\{x \geq 11\}$
4. $\{x \geq 10\}$ $\quad$ **atomic**$\{x = x + 5\}$ $\quad$ $\{x \geq 12\}$
5. $\{x \ mod \ 2 = 1\}$ $\quad$ **atomic**$\{x = x + 5\}$ $\quad$ $\{x \ mod \ 2 = 0\}$
6. $\{x \ mod \ 2 = 1\}$ $\quad$ **atomic**$\{y = x + 1\}$ $\quad$ $\{y \ mod \ 2 = 0\}$
7. $\{y \ mod \ 2 = 1\}$ $\quad$ **atomic**$\{y = y + 1\}$ $\quad$ $\{y \ mod \ 2 = 0\}$
8. $\{x \ mod \ 3 = 0\}$ $\quad$ **atomic**$\{y = x\}$ $\quad$ $\{y \ mod \ 3 = 0\}$

Exercice 3 : En utilisant la méthode de l'exclusion des configurations, prouvez que dans l'exemple du producteur/consommateur, la variable `buf` n'est pas accédée (lue ou écrite) en même temps.

Exercice 4 : Soit le programme suivant :

```
int s = 1
proctype Proc(int i) {
```

```
    do
    :: atomic { (s > 0) -> s = s-1 }
       S_i
       atomic { s = s+1 }
    od
}
init { run Proc(1); run Proc(2) }
```

Les instructions `S_i` ne modifient pas la variable globale `s`.

1. Écrire une preuve pour chaque processus.
2. Montrez que les preuves écrites sont sans interférence.
3. Utilisez les prouves écrites et la technique de l'*exclusion des configurations* pour montrer les propriétés de sûreté suivantes :
   (a) Le programme ne bloque pas (absence de *deadlock*).
   (b) Les instructions `S_1` et `S_2` ne peuvent pas être exécutées en même temps.
4. Quel type d'équité doit implémenter le *scheduler* pour qu'un processus qui est en attante pour la première action atomique puisse effectuer `S_i` ?

Exercice 5 : Écrire un programme concurrent qui trouve la valeur maximum d'un tableau d'entiers $a[N]$ en regardant en parallèle les indices paires et impaires. Prouvez que les processus sont sans interférence et que votre programme est partiellement correct.

Correction 1 : Pour un programme concurrent à N processus, chaque processus ayant M actions atomique, le nombre d'exécutions est $(N * M)!/(M!)^N$ (prouvez-le !).

1. Ici, N = 2, M = 2, donc 6 exécutions. Si on note la j-ème instruction du processus Pi, aij, on obtient un arbre avec 6 chemins dont les feuilles donnent les valeurs finales de x et y :

```
00 --a11-> 10 --a12-> 11 --a21-> 31 --a22-> 3-2
 |          |
 |          +---a21-> 20 --a12-> 22 --a22-> 20
 |                     |
 |                     +---a22-> 2-2--a12-> 20
 |
 +---a21-> 20 --a22-> 2-2--a11-> 3-2--a12-> 31
            |
            +---a11-> 10 --a12-> 11 --a22-> 10
                       |
                       + --a22-> 1-1--a12-> 10
```

2. Maintenant, N = 2, M=6, donc le nombre d'exécutions est 924. Par exemple, l'instruction `y=y+x` sera implémentée par :

```
load  r, y
add   r, x
store r, y
```

Exemple de paire de valeurs différente de ceux ci-dessus (1,1) :

```
                      x   y
a21(load,add,store)   2   0
a11(load,add,store)   1   0
a22(load)
a12(load)
a22(sub,store)        1   -1
a12(add,store)        1   1
```

(a22,a11(load),a22(load),

Correction 2 : Comme on nous dit que les 8 triplettes (notation générale $\{P_i\}\ S\ \{Q_i\}, 1 \leq i \leq 8$ sont des preuves[1], pour chaque i, l'ensemble d'assertion critiques $\mathcal{C}_i = \{P_i, Q_i\}$. Il faut donc étudier la vérité de la formule suivante :

$$NI(a, P_i) \wedge NI(a, Q_i)$$

c'est-à-dire :

$$(\{P_i \wedge x \geq 4\}\ a\ \{P_i\}) \quad \wedge \quad (\{Q_i \wedge x \geq 4\}\ a\ \{Q_i\})$$

[1] Attention, dans la version du 7 février, en TD, j'avais dit qu'il s'agit des triplettes, pas de preuves, donc l'assertion en fin de triplettes n'était pas forcement critique et on testait que NI sur la pré-condition des triplettes.

Alors on obtient les réponses :

1. interférence (P_1 conservé mais pas Q_1)
2. non-interférence (P_2, Q_2 conservés)
3. interférence (P_3, Q_3 non-conservés)
4. interférence (P_3, Q_3 non-conservés)
5. non-interférence
6. non-interférence
7. non-interférence (variables disjointes)
8. interférence (Q_8 conservé mais pas P_8)

Correction 3 : La méthode d'exclusion des configurations est utilisée pour prouver les propriétés de sûreté et dit : si on a une preuve de programme et P et Q sont des conséquences de cette preuve qui décrivent respectivement l'état s_i d'un processus et l'état s_j d'un autre processus, alors ces processus ne peuvent pas se trouver en même temps dans les états s_i et s_j si $P \wedge Q = F$.

Dans l'exemple du producteur/consommateur (cours 2, pages 13-14), la variable `buf` est une variable globale

- écrite par le producteur dans l'état où $p == c$, avant que p soit incrémenté
- lue par le consommateur dans l'état où $p > c$, avant que c soit incrémenté

Comme $(p == c) \wedge (p > c) = F$, on a prouvé la propriété demandée.

Correction 4 : Comme on peut remarquer, il s'agit d'une implémentation du sémaphore binaire en utilisant de l'atomicité logicielle (instruction **atomic**).

1. Un schéma de preuve pour le processus i est :

```
proctype Proc(int i) {
    { I : s >= 0 }
    do
    :: { I }
       atomic { (s > 0) -> { s>= 1 } s = s-1 }
       { I }
       S_i
       { I }
       atomic { s = s+1 }
       { I }
    od
   { I & F }
}
```

2. On a une seule assertion critique I. ($s \geq 1$ n'est pas critique car dans une action atomique.) Les actions d'affectation sont les deux instructions **atomic**, notons les a_P resp. a_V. Il faut donc faire $2*1*2*1$ preuves de non-interférence dont 2 sont différentes (car les deux processus ont le même code) :
   (a) $NI(a_P, I) : \{I \wedge pre(a_P)\}\ a_P\ \{I\}$ est équivalent à $\{I \wedge s > 0\}\ a_P\ \{I\}$ qui est vrai.

(b) $NI(a_V, I) : \{I \wedge pre(a_V)\}\ a_V\ \{I\}$ est équivalent à $\{I \wedge T\}\ a_V\ \{I\}$ qui est vraie car $(s \geq 1) \Rightarrow I$ (donc on applique la règle de la Conséquence).

3. Pour l'absence de blocage, la preuve est simple car, si `S_i` ne bloque pas, le seul endroit où ça peut bloquer est le premier **atomic**. Mais ceci n'est pas possible car s ne peut rester à 0 indéfiniment vu que le deuxième **atomic** sera toujours atteint.

   Pour l'exclusion mutuelle des exécutions de `S_1` et `S_2`, il faut travailler plus, car on a besoin d'identifier les états dans lesquels chaque processus se trouve. On refait donc la preuve en ajoutant les variables booléennes globales `in_i` qui sont vraies quand le processus i a passé le premier **atomic** et deviennent fausses après le deuxième **atomic**. Pour simplifier la présentation, on refait la preuve pour i=1.

```
int s=1, in_1=0, in_2=0; { I : in_1 + in_2 <= 1 }
proctype Proc(1) {
    { I & s >= 0 & in_1 = 0 & in_2 = 0 }
    do
    :: { I & s >= 0 & in_1 = 0 }
       atomic { (s > 0) -> s = s-1; in_1 = 1 }
       { I & s >= 0 & in_1=1 }
       S_i
       { I & s >= 0 & in_1=1 }
       atomic { s = s+1; in_1=0 }
       { I & s >= 0 & in_1=0}
    od
}
```

   Les preuves d'interférence restent vraie car les affectations des variables `in_i` sont locales à chaque processus et ne dependent pas de l'autre variable.

   Si les deux processus exécuteraient en même temps leurs `S_i`, alors les deux assertions qui restent vraies pendant les `S_i` doivent être vraie en même temps, c'est à dire :

```
in_1+in_2 <= 1 & s >= 0 & in_1=1 & in_2=1
```

   c'est qui n'est pas possible, q.e.d.

4. Il faut une équité forte car on a une action conditionnelle et que l'équite faible n'est pas suffisante (le processus peut être empeché indéfiniment de rentrer dans `S_i`).

Correction 5 : Première solution : le maximum pour chaque parité d'incide est calcule dans 2 variables globales et puis on calcule le max global. Pour éviter les sélections selon le type d'indice, on peut faire deux proctype : un pour indices paires et l'autre pour les indices impaires. La solution ci-dessous permet d'introduire la deuxième solution.

```
#define N 10

int a[N]; /* on suppose initalise */
int MaxP, MaxI, Max;
int fin;
```

```
proctype MaxDemi(bool b) {
  int i;
  /* initialise le parcours du tableau */
  i = b;
  /* parcours du tableau */
  do
  :: i < N & b==0 -> (MaxP < a[i]) -> MaxP = a[i] ;
                     i = i + 2
  :: i < N & b==1 -> (MaxI < a[i]) -> MaxI = a[i] ;
                     i = i + 2
  od;
  /* signale la fin */
  atomic { fin = fin + 1 }
}

proctype MaxFin() {
  (fin == 2);
  if :: MaxP > MaxI -> Max = MaxP
     :: MaxP <= MaxI -> Max = MaxI
  fi
}

init {
  run MaxDemi(0); run MaxDemi(1); run MaxFin()
}
```

On peut montrer que chaque processus MaxDemi calcule le max de son type d'indices et qu'ils n'interfèrent pas car ils lisent et écrivent des ensembles disjoint de variables dans la boucle principale ; comme l'affectation de fin est atomique, toute interférence est évitée.

Deuxième solution, proposée en TD, est d'utiliser une seule variable globale Max et de rendre atomique son accès. Ce qui change :

```
int Max;
int fin;

proctype MaxDemi(bool b) {
  int i;
  /* initialise le parcours du tableau */
  i = b;
  /* parcours du tableau */
  do
  :: i < N & b==0 -> atomic { (Max < a[i]) -> Max = a[i] } ;
                     i = i + 2
  :: i < N & b==1 -> atomic { (Max < a[i]) -> Max = a[i] } ;
                     i = i + 2
  od;
  /* signale la fin */
```

```
  atomic { fin = fin + 1 }
}
...
```

Mais cette solution est faussement parallèle car l'accès à la variable Max en exclusion mutuelle transforme le calcule parallèle en un calcul séquentiel (on s'arrête pour chaque indice pour obtenir l'accès a Max). Elle est toutefois correcte, car l'interférence est exclue grâce aux sections atomiques.