Concurrence – Master 1
TD 3 : SPIN

Correction 1 :

1. Pour pouvoir écrire la propriété d'exclusion mutuelle, il faut savoir identifier les points de contrôle des processus qui correspondent aux sections critiques. Pour cela, on utilise les étiquettes, par exemple :

   ```
   active proctype P1 () {
   ...
   cs1:      /* section critique */
   ...
   }
   ```

   Après cela, la propriété à vérifier s'écrit `[] !(p1cs1 && p2cs2)` où :

   ```
   #define p1cs1 (P1@cs1)
   #define p2cs2 (P2@cs2)
   ```

2. L'absence de blocage dit qu'à tout moment il existe un processus qui peut faire une action, ou autrement dit qu'il est impossible d'attendre l'inaction, `[] X true`. Comme on ne peut pas utiliser l'opérateur X en Spin, on peut réécrire la propriété en disant qu'il est toujours possible d'entrer en section critique `[] <> (p1cs1 || p2cs2)`.

   Le résultat de la vérification doit être faux si aucune hypothèse sur l'équité est faite. En effet, le scénario montré par Spin indique que le processus P2 peut conserver à infini le processeur et ainsi bloquer la situation.

   Par contre, si l'équité faible est assurée (weak fairness dans les choix de la vérification), la propriété est vraie.

3. L'absence de délai dit que, par exemple si P1 reste toujours en section non-critique et P2 demande la section critique, P2 aura la section critique dès le pas suivant. Mais "dès le pas suivant" ne peut pas être exprimé dans la logique temporelle accepté par SPIN. On le remplace donc par "infiniment souvent". Pour écrire cette propriété, on doit introduire une étiquette pour la ligne section non-critique de P1 (et un `skip`) et la définition du symbole p1ncs1. (C'est pas la paine de faire une boucle infinie, on demandera dans la propriété que le processus P1 reste toujours à la ligne ncs1.) Alors la propriété à vérifier devient `[] (([]p1ncs1) -> []<> p2cs2)`. L'opérateur `[]` devant est nécessaire car `([]p1ncs1)` est parfois faux, ce qui rend la propriété trivialement vraie. Il faut exécuter pas à pas la trace donnée par Spin comme contre-exemple afin de comprendre qu'il a une boucle.

La propriété d'absence de famine pour P2 est P2 rentre en section critique infiniment souvent : `[]<> p2cs2`. Spin nous montre qu'elle est fausse.

Correction 2 :

1. La correction est similaire au point 1 ci-dessus. En regardant la trace qui permet aux processus d'entrer en section critique, on obtient : P1 teste want2 à -1 (true), P2 teste want1 à -1 (true), P1 affecte want1 à 1, P2 affecte want2 à 1, P1 passe le test `want1 !=-want2`, pareil P2, donc les deux sont en section critique.
2. Il faut écrire dans chaque processus à la place de :

```
do :: want1 == -1 -> want2 = 1
   :: else -> want2 = -1
od
```

```
do :: atomic { want1 == -1 -> want2 = 1 }
   :: atomic { else -> want2 = -1 }
od
```

Correction 3 : Il s'agit d'une propriété de vivacité. Pour la vérifier avec SPIN, il faut déclarer le symbole correspondant à la propriété y égal à 42 :

```
#define y42 (y==42)
```

La propriété devient `<> y42`. Ceci doit être vrai pour un ordonnanceur à équité inconditionnelle.

Correction 4 : La propriété s'écrit pareil que ci-dessus. Mais le texte du programme à changé car l'incrémentation de y est maintenant conditionnée par une variable qui change à chaque tour de P1. Donc la propriété est fausse pour un ordonnanceur inconditionnel ou à équité faible, mais c'est vrai pour l'équité forte.