Examen de programmation répartie
(25 mars 2019)

(durée 2 heures - documents autorisés : 10 pages imprimées ou bien 20 pages manuscrites maximum)
2 pages

Exercice 1.— On rappelle que pour le *consensus* chaque thread a une valeur initiale et doit décider une valeur qui

doit assurer (1) accord : il n'y a qu'une seule valeur de décision (2) intégrité : une valeur décidée doit être la valeur initiale d'un des threads et (3) wait-freeness : une thread décide toujours même si d'autres threads arrêtent d'exécuter leur code.

Un objet de la classe GAS, est un booléen initialisée à `true` et avec une seule méthode `getAndSet(boolean new)` qui positionne la valeur de l'objet à `new` et retourne l'ancienne valeur.

1. Proposer un algorithme d'exclusion mutuelle utilisant des objets de la classe GAS (et bien sûr des registres partagés)
2. Peut-on faire un consensus entre deux threads en utilisant des objets GAS (et bien sûr des registres partagés)? Si oui proposer un algorithme pour cela. (On pourra remarquer que si deux threads essaient de façon concurrente de faire un `getAndSet(false)`, une seule aura `true` comme valeur retournée par `getAndSet(false)`).
3. Peut-on implémenter un objet GAS avec uniquement des registres partagés?
4. (question plus difficile!) Peut-on réaliser un consensus entre 3 threads avec des objets GAS (et bien sûr des registres partagés)?

Exercice 2.— On considère (on suppose ici qu'un tel algorithme existe!) un algorithme quelconque de consensus binaire (les valeurs initiales sont 0 ou 1) utilisant des `read` et des `write` de registres atomiques, et deux threads A et B. On rappelle qu'un état est 0-valent (resp. 1-valent) si à partir de cet état seule la décision 0 (resp. 1) est possible. Un état est bivalent si à partir de cet état la décision 0 et la décision 1 sont possibles (il existe une exécution de l'algorithme qui partant de cet état amène à une décision 0 et il existe une exécution de l'algorithme qui partant de cet état amène à une décision 1).

1. Soit S un état initial avec valeur initiale 0 pour A, et valeur initiale 0 pour B, quelles sont les décisions possibles? S est-il un état bivalent?
2. Partant de S on considère l'exécution dans laquelle *seul* A fait des pas de calculs, cette exécution termine-t-elle et si oui avec quelle valeur?
3. Partant de S' avec valeur initiale 0 pour A et 1 pour B, on considère l'exécution dans laquelle *seul* A fait des pas de calculs, cette exécution termine-t-elle et si oui avec quelle valeur? (On pourra remarquer que cette exécution pour A est indistinguable de celle de la question précédente).
4. En appliquant le même raisonnement à B montrer que à partir de S' B ne peut décider que 1 s'il est le seul à faire des pas de calculs.
5. En déduire que pour tout algorithme de consensus il existe un état initial bivalent.
6. Si maintenant au lieu de n'utiliser que des `read` et des `write` sur des registres atomiques, l'algorithme de consensus peut aussi utiliser des `getAndSet()` d'objets de la classe GAS, les résultats des questions précédentes sont-ils encore valables? non
7. Si maintenant au lieu de n'utiliser que des `read` et des `write` sur des registres atomiques, l'algorithme de consensus peut aussi utiliser des opérations de n'importe quel objet linéarisable défini par une spécification séquentielle, les résultats des questions précédentes sont-ils encore valables? oui

Exercice 3.— On considère un objet INC dont la valeur est un entier dont la spécification séquentielle (*S*) est la suivante suivante :

- la valeur de l'objet est initialement 0
- on a une seule méthode `inc` qui retourne un entier telle que si avant l'appel de `inc` la valeur de l'objet est v alors la valeur de l'objet après `inc` est $v + 1$ et la valeur retournée est v .

On considère la classe Java suivante (I) :

```
class Inc1{
  private int val=0;
  public int inc(){
    return val++;
  }
}
```

1. S'il n'y a qu'une seule thread, un objet de la classe `Inc1` satisfait-il la spécification *S* ?
2. Ecrire un code java qui crée deux threads partageant un objet `Inc`, initialisé à '0'. Chacune des threads appellera 2 fois la méthode `Inc` et affichera la valeur retournée à chaque fois.
3. Quelles sont les valeurs possibles affichées dans chacune des threads ?
4. En déduire que `Inc1` ne satisfait pas la spécification séquentielle *S*.
5. On modifie le programme de la façon suivante (II) :

```
class Inc{
  private int val;
  public Inc(int v){val=v;}
  public synchronized int inc(){
    return val++;
  }
}
```

Quelles sont les valeurs possibles affichées dans chacune des threads pour le programme précédent ?
`Inc` est-il une implémentation linéarisable de INC ?

6. On modifie le programme de la façon suivante (III) :

```
class Inc{
  private volatile int val;
  public Inc(int v){val=v;}
  public int inc(){
    return val++;
  }
}
```