

Votre nom: \_\_\_\_\_ (à rendre sur votre travail)

## Test de génie logiciel

20.12.2008

### Problème 1

On suppose que l'objet instancié à partir de la classe suivante (on a laissé de côté les détails non significatifs)

```
class Synchro {  
    public void synchronized go() {  
        notify() ;  
        wait() ;  
    }  
}
```

est appelé par trois threads qui appellent la méthode *go* en boucle. On considère les états significatifs (que vous déterminez vous-même) situés à l'intérieur de l'objet instancié à partir de cette classe, mais toutes les instructions des threads situées à l'extérieur de cet objet sont rassemblées dans un même état.

- Dessinez le graphe des états globaux du programme composé des trois threads et accessibles à partir de la position initiale définie par la présence des trois threads dans l'état extérieur.
- Indiquez s'il est possible de revenir à l'état initial.

### Problème 2

On donne les agents CCS décrits par les expressions suivantes :

$$\begin{aligned}A &= m_0 . A' + m_1 . A'' \\A' &= m_0 . A' + \text{ack}_0 . A \\A'' &= m_1 . A'' + \text{ack}_1 . A\end{aligned}$$
$$B = m_0 . B + m_1 . B$$
$$C = m_0 . \text{ack}_0 . C + m_1 . \text{ack}_1 . C$$

dessiner le graphe des états globaux atteignables à partir de la combinaison  
 $(A|B|C) \setminus m_0 \setminus \text{ack}_0 \setminus m_1 \setminus \text{ack}_1$ .

**tourner la page**

### **Problème 3**

Définir un verrou d'exclusion mutuelle (permettant d'assurer qu'un seul agent soit dans son état d'exclusion à un moment donné) au moyen du langage CCS. Démontrer, de la façon que vous jugez adéquate, que ce verrou fonctionne correctement avec trois automates (ou agents).

### **Problème 4** (optionnel, mais donne un bonus)

- a) Expliquer les raisons qui conduisent à utiliser le pattern « Value Object » (ou « Data Transfer Object »)
- b) Qu'est le pattern MVC (Model View Controller) ?

Votre nom: \_\_\_\_\_ (à rendre sur votre travail)

## Test de génie logiciel

20.12.2008

### Problème 1

Soit un programme composé de deux threads exécutant les méthodes *run()* décrites ci-dessous :

```

run () {
    while(true) {
        obj.inc() ;
    }
}

class OBJ {
    int x = 0 ;
    synchronized inc() {
        x++ ;
        notify() ;
    }
    synchronized delay() {
        if (x<3) {
            wait() ;
        }
    }
}

run() {
    while(true) {
        obj.delay() ;
    }
}

```

Dessiner le graphe des états globaux atteignables du point de départ défini par les débuts des méthodes *run()*  $x=0$ .

### Problème 2

On donne les agents CCS décrits par les expressions suivantes :

$$\begin{aligned}
 A &= m_0 . A' + m_1 . A'' \\
 A' &= m_0 . A' + ack_0 . A \\
 A'' &= m_1 . A'' + ack_1 . A
 \end{aligned}$$

$$B = m_0 . B + m_1 . B$$

$$C = m_0 . ack_0 . C + m_1 . ack_1 . C$$

dessiner le graphe des états globaux atteignables à partir de la combinaison  $(A|B|C) \setminus m_0 \setminus ack_0 \setminus m_1 \setminus ack_1$ .

**tourner la page**

### Problème 3

Décrire en CCS un automate dont l'état est défini par la valeur d'un compteur (autrement dit, l'automate *est* le compteur lui-même). Ce compteur peut varier de 0 à 3. Il peut être incrémenté ou décrémenté, pour autant qu'il ne soit pas en bout de course, auquel cas une des opérations n'est pas définie.

Décrire deux automates identiques qui peuvent incrémenter ou décrémenter le compteur, mais qui ne peuvent pas faire plus de deux incréments en avance sur les décrémentations (quand il a fait deux incréments, il doit faire une décrément avant de refaire une nouvelle incrémentation).

Définir les états accessibles à partir de la mise en parallèle du compteur et des deux agents qui l'appellent. L'automate est infini. A partir du point où vous voyez la répétition d'un pattern, indiquez-le au moyen de ... (ou etc.).

### Problème 4 (optionnel, mais donne un bonus)

- a) Pourquoi utiliser le pattern « Session Facade » ?
- b) Qu'est le pattern MVC (Model View Controller) ?