Examen de CPS – 4I506

Frédéric Peschanski & Antoine Miné

Mai 2016

Durée : 2 heures **Barème** donné à titre indicatif.

Documents autorisés : Notes de cours

ATTENTION : les 3 exercices de l'énoncé devront être rendus sur des feuilles séparées.

Exercice 1 : Spécifications et tests ($\approx$ 7 points)

L'objectif de cet exercice est de spécifier un principe (simplifié) de gestion de géométrie similaire à ceux proposés dans les bibliothèques Swing, QT, etc.

Question 1.1. L'entité graphique de base est le *widget* (pour *window gadget*) dont une spécification (partielle) est donnée ci-dessous :

service : Widget
observators :
  x : [Widget] $\rightarrow$ Integer
  y : [Widget] $\rightarrow$ Integer
  width : [Widget] $\rightarrow$ Integer
  height : [Widget] $\rightarrow$ Integer
  const bestWidth : [Widget] $\rightarrow$ Integer
  const bestHeight : [Widget] $\rightarrow$ Integer
  visible : [Widget] $\rightarrow$ Boolean
constructors :
  init : Integer $\times$ Integer $\rightarrow$ [Widget]
    pre : init(bw, bh) **require** (bw > 0) $\wedge$ (bh > 0)
operators :
  // A COMPLETER
observations :
[invariants]
  (x $\geq$ 0) $\wedge$ (y $\geq$ 0) $\wedge$ (width $>$ 0) $\wedge$ (height $>$ 0)
[init]
  (x(init(bw, bh)) = 0) $\wedge$ (y(init(bw, bh)) = 0) $\wedge$ (width(init(bw, bh)) = 0) $\wedge$ (height(init(bw, bh)) = h)
  (bestWidth(init(bw, bh)) = bw) $\wedge$ (bestHeight(init(bw, bh)) = bh)
  visible(init(bw, bh)) = false
// A COMPLETER
...

Compléter la spécification ci-dessus en ajoutant :
- — un opérateur move permettant de déplacer un widget selon les axes x, y
- — un opérateur resize permettant de le redimensionner (largeur et hauteur)
- — un opérateur show prenant en argument un booléen indiquant le statut visible ou non du widget.
- — toutes les observations nécessaires pour assurer la cohérence et la complétude de la spécification.

Question 1.2. Un conteneur (*container*) permet de regrouper plusieurs widgets en gérant leurs positions relatives selon un principe de gestion de géométrie spécifique.

Proposer un service Container permettant de spécifier un conteneur «minimal» dont les fonctionnalités sont les suivantes :
- — ajout et retrait d'un widget du conteneur
- — mise en page layout du conteneur

La propriété fondamentale que l'on souhaite associer au service est la suivante :

> Deux widgets visibles ne peuvent se chevaucher

Question 1.3. Dans la plupart des bibliothèques d'interfaces graphiques, on peut instancier un conteneur de type HorizontalBox dont le principe de gestion est le suivant :

- Les widgets sont placés les uns derrière les autres à l'horizontale et de gauche à droite
- ils sont centrés verticalement
- leur largeur et hauteur préférées (bestWidth, bestHeight) sont prises en compte, dans la mesure du possible

Proposer une spécification du service HorizontalBox par raffinement de Container.

Attention : Votre spécification doit être un raffinement correct, et vous devez le justifier.

Exercice 2 : Promela/Spin ($\approx$ 6 points)

Dans cet exercice nous revisitons le célèbre exemple du dîner des philosophes. Nous en rappelons les principes : cinq philophes affamés sont attablés à des places numérotées de 0 à 4. Pour se nourrir, le philosophe à la place n doit récupérer deux fourchettes devant lui : par convention de numéros n et $n + 1$ *modulo* 5.

L'objectif (1) est de permettre à chaque philosophe de manger.

L'objectif (2) est de garantir que tous les philosophes puissent se nourrir.

Notre modèle de départ est décrit en Promela ci-dessous :

```
bool forks[5] = { true, true, true, true, true };
bool eats[5] = { false, false, false, false, false };
bool eaten[5] = { false, false, false, false, false };

proctype Philo(byte num) {

loop:

  forks[num] -> forks[num] = false;
  forks[(num+1)%5] -> forks[(num+1)%5] = false;

  eats[num] = true;
  eaten[num] = true;
  eats[num] = false;

  goto loop;
}

init {
  atomic {
    run Philo(0); run Philo(1); run Philo(2);
    run Philo(3); run Philo(4)
  }
}
```

Question 2.1. L'outil Spin détecte un *deadlock* sur ce modèle. Décrire précisément un chemin conduisant à cette situation.

Question 2.2. Proposer une solution la plus simple et concise possible pour éliminer le deadlock et couvrir l'objectif (1) mais *pas* l'objectif (2). Justifier votre réponse.

Question 2.3. Proposer une modification du modèle permettant de couvrir les objectifs (1) et (2) (et bien sûr sans problème de deadlock).

Exercice 3. : Logique de Hoare ($\approx$ 7 points)

Question 3.1. Soit le programme P suivant, qui prend en argument un entier n et calcule dans x sa racine carrée (arrondie à l'entier inférieur) :

```
{
  var x = 0;
  var y = 0;
  while (y < n)
  {
    y = y + 2*x + 1;
    x = x + 1;
  }
}
```

1. Simulez l'exécution du programme pour une valeur de $n > 0$.
2. Déduisez-en un invariant de boucle.
3. Prouvez la correction de la spécification suivante en logique de Hoare :

$$\{n \geq 0\}\ P\ \{x^2 \leq n < (x+1)^2\}$$

Question 3.2. Considérons le programme suivant qui multiplie une matrice `m` par un vecteur `in` et stocke le résultat dans le vecteur `out` :

```
{
  var i = 0;
  while (i < n)
  {
    var j = 0;
    out[i] = 0;
    while (j < m)
    {
      out[i] = out[i] + m[i][j] * in[j];
      j = j + 1;
    }
    i = i + 1;
  }
}
```

1. Donnez une spécification pour ce programme
2. Donnez un invariant pour chacune des deux boucles (on ne cherchera pas à prouver ces invariants en logique de Hoare).