Spécification et preuve de programmes
Solution du devoir 1 : Vérification déductive, logique de Hoare et logique de micro-C

Alain Giorgetti

Licence d'Informatique de l'Université de Franche-Comté 2019-20

1 Preuve propositionnelle avec micro-C (2 points)

Donner le contenu d'un fichier micro-C qui permet de déterminer si l'ensemble de formules propositionnelles

$$\{p \vee q, \neg q \vee r, p \vee \neg r, \neg p \vee q, \neg p \vee \neg q\}$$

est contradictoire ou non.

(1 point pour la syntaxe, 1 point si la formule correspond à l'énoncé)
Pour montrer qu'un ensemble de formules est contradictoire, on peut par exemple démontrer que la négation d'une d'entre elles est impliquée par la conjonction des autres formules. Une solution est dans le listing suivant :

```
/*@
predicate p();
predicate q();
predicate r();

lemma f1: (p || q) && (! q || r) && (p || !r) && (! p || q) → !(! p || !q); */
```

2 Logique des prédicats avec micro-C (3 points)

Donner le contenu d'un fichier micro-C qui permet de déterminer si les formules suivantes sont valides, pour des prédicats sur les entiers.

1. $(\exists X.\, (\forall Y.\, S(X) \Leftrightarrow S(Y))) \wedge (\forall Z.\, S(Z))$

   (1 point) Lemme `f1` dans le listing 1.

2. $\forall X.\, \exists Y, Z.\, (R(X,X) \Rightarrow R(Z,Y)) \wedge (\neg R(X,X) \vee R(Y,Z))$

   (1 point) Lemme `f2` dans le listing 1.

3. $\forall X, Y.\, \exists Z.\, (R(X,X) \wedge R(X,Y)) \Rightarrow R(X,Z)$

   (1 point) Lemme `f3` dans le listing 1.

```
/*@
predicate S(int i);
predicate R(int i, int j);

lemma f1: exists X. (forall Y. S(X) ↔ S(Y))) && (forall Z. S(Z));

lemma f2: forall X. exists Y,Z. (R(X,X) → R(Z,Y)) && (not R(X,X) || R(Y,Z));

lemma f3: forall X,Y. exists Z. (R(X,X) && R(X,Y)) → R(X,Z);
*/
```

Listing 1 – Solution de l'exercice 2.

3 Recherche dichotomique (8 points)

Cet exercice porte sur l'exemple de recherche dichotomique de la partie 2.4.2 du cours.

1. Adapter la spécification de ce programme de recherche dichotomique à un tableau $t[0..n-1]$ au lieu de $t[1..n]$, en remplaçant MAX par la longueur du tableau, ici notée $|t|$.

(3 points, détaillés ci-dessous)
(0,5 points) **Données** :
t : tableau de n entiers classés en ordre croissant,
x : entier dont on recherche la position d'insertion dans $t[0..n-1]$.
(0,5 points) **Résultat** : $posx$: entier compris entre 0 et n, position d'insertion de x dans t.
(1 point) **Précondition** : t est un tableau éventuellement vide trié en ordre croissant. Ceci est exprimé formellement ainsi :

$$n \geq 0 \wedge n \leq |t| \wedge \forall i.\forall j.((i \in [0..n-1] \wedge j \in [0..n-1] \wedge i < j) \Rightarrow t[i] \leq t[j])$$

Remarque : Une meilleure réponse contiendrait $n < |t|$, afin que la case $t[n]$ existe pour stocker x, mais ce n'est pas exigé ici car ce n'était pas envisagé dans le cours.
(0,5 points) **Postcondition** : tous les éléments de t de position comprise entre 0 et $posx - 1$ sont inférieurs ou égaux à x, ceux de position comprise entre $posx$ et $n - 1$ sont supérieurs à x. Ceci s'exprime formellement ainsi :
(0,5 points)

$$\forall k.(k \in [0..posx-1] \Rightarrow t[k] \leq x) \wedge \forall k.(k \in [posx..n-1] \Rightarrow t[k] > x)$$

2. Même question pour le programme de recherche dichotomique, donné dans la figure 2.4 du cours.

(2 points)
$min := -1;\ max := n - 1;$
while $min \neq max$ do
  $m := (min + max + 1)$ **div** 2;
  if $t[m] \leq x$ then $min := m$ else $max := m - 1$ fi
od ;
$posx := min + 1$

3. Formaliser dans la logique de micro-C la postcondition sur les bornes de *posx* après exécution du programme.

(1 point)

```
predicate posx_post(int posx, int n) = 0 ≤ posx ≤ n;
```

4. Formaliser dans la logique de micro-C la précondition de votre réponse à la question 1.

(1 point)

```
predicate pre(int t[], int n) =
  0 ≤ n ≤ length(t) && forall i, j. 0 ≤ i < j < n → t[i] ≤ t[j];
```

5. Même question pour la postcondition.

(1 point)

```
predicate post(int t[], int n, int x, int posx) =
  forall k. 0 ≤ k < posx → t[k] ≤ x &&
  forall k. posx ≤ k < n → t[k] > x;
```

4 Multiplication par additions, en logique de Hoare (7 points)

1. Exprimer par un triplet de Hoare que, si x est positif ou nul et si le programme

$$i := 0;\ p := 0;\ \text{while } i < x \text{ do } i := i + 1;\ p := p + y \text{ od}$$

termine, alors il calcule dans p le produit des entiers x et y.

(0,5 points)

$$\{x \geq 0\}\ i := 0;\ p := 0;\ \text{while } i < x \text{ do } i := i + 1;\ p := p + y \text{ od } \{p = x * y\}$$

2. En appliquant une par une les règles de la logique de Hoare, démontrer formellement **dans un tableau** que ce triplet de Hoare est valide.

(5,5 points : -0,5 par déduction manquante)
Dans la logique de Hoare, une preuve de validité de ce triplet de Hoare est :

No	Triplet ou formule	Justification
1	$\{x \geq 0\}\ i := 0;\ p := 0;\ \text{while } i < x \text{ do } i := i + 1;\ p := p + y \text{ od } \{p = x * y\}$	(*seq.*) 2 3
2	$\{x \geq 0\}\ i := 0\ \{x \geq 0 \wedge i = 0\}$	(*assign.*)
3	$\{x \geq 0 \wedge i = 0\}\ p := 0;\ \text{while } i < x \text{ do } i := i + 1;\ p := p + y \text{ od } \{p = x * y\}$	(*seq.*) 4 5
4	$\{x \geq 0 \wedge i = 0\}\ p := 0\ \{x \geq 0 \wedge i = 0 \wedge p = 0\}$	(*assign.*)
5	$\{x \geq 0 \wedge i = 0 \wedge p = 0\}\ \text{while } i < x \text{ do } i := i + 1;\ p := p + y \text{ od } \{p = x * y\}$	(*conseq.*) 6 7 8
6	$x \geq 0 \wedge i = 0 \wedge p = 0 \Rightarrow i \leq x \wedge p = i * y$	(voir (1))
7	$\{i \leq x \wedge p = i * y\}\ \text{while } i < x \text{ do } i := i + 1;\ p := p + y \text{ od } \{i \geq x \wedge i \leq x \wedge p = i * y\}$	(*while*) 9
8	$i \geq n \wedge i \leq x \wedge p = i * y \Rightarrow p = x * y$	évident
9	$\{i < x \wedge i \leq x \wedge p = i * y\}\ i := i + 1;\ p := p + y\ \{i \leq x \wedge p = i * y\}$	(*seq.*) 10 11
10	$\{i < x \wedge p = i * y\}\ i := i + 1\ \{i \leq x \wedge p = (i - 1) * y\}$	(voir (2))
11	$\{i \leq x \wedge p = (i - 1) * y\}\ p := p + y\ \{i \leq x \wedge p = i * y\}$	(voir (3))

(1 point pour les justifications)
(1) L'implication

$$x \geq 0 \wedge i = 0 \wedge p = 0 \Rightarrow i \leq x \wedge p = i * y$$

est évidente : il suffit de remplacer i et p par 0 à droite de l'implication.
(2) Le triplet

$$\{i < x \wedge p = i * y\}\ i := i + 1\ \{i \leq x \wedge p = (i-1) * y\}$$

se justifie comme suit :

$$\frac{}{\{(i \leq x \wedge p = (i-1) * y)\ (i+1/i)\}\ i := i + 1\ \{i \leq x \wedge p = (i-1) * y\}}$$

selon la règle (*assignment*). Or $i + 1 \leq x$ est équivalent à $i < x$ et $p = (i + 1 - 1) * y$ se simplifie en $p = i * y$.
(3) Le triplet

$$\{i \leq x \wedge p = (i-1) * y\}\ p := p + y\ \{i \leq x \wedge p = i * y\}$$

se justifie comme suit :

$$\frac{}{\{(i \leq x \wedge p = i * y)\ (p + y/p)\}\ p := p + y\ \{i \leq x \wedge p = i * y\}}$$

selon la règle (*assignment*). Or $p + y = i * y$ est équivalent à $p = (i-1) * y$.