Spécification et preuve de programmes
Solution du devoir 2 : Preuves avec micro-C

Alain Giorgetti

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

Ce devoir doit être rendu sous la forme d'un unique fichier texte *NomPrenom*2.c, où ***NomPrenom*** sont les 8 premières lettres de votre nom de famille, suivies des 8 premières lettres de votre prénom, sans espaces ni accents. Ce fichier doit commencer par un commentaire entre `/*` et `*/` indiquant vos nom et prénom. Ensuite, des commentaires entre `/*` et `*/` doivent indiquer le numéro de chaque question traitée, par exemple

```
/* Exercice 1 question 2 */
```

juste avant la réponse à la question 2 de l'exercice 1.

La preuve et l'exécution du contenu complet de ce fichier doivent être possibles avec l'interface de micro-C en ligne. Dans ce but, les parties incorrectes ou incomplètes doivent être entourées par `/*` et `*/`.

Lire l'énoncé jusqu'à la fin avant de commencer à traiter les exercices.

1 Propriétés des tableaux d'entiers micro-C (5 points)

L'objectif de cet exercice est d'écrire des formules micro-C qui formalise des propriétés sur les tableaux d'entiers C.

1. Reproduire et compléter la définition

   ```
   predicate cte(int a[], int c) =
   ```

   pour définir en micro-C la propriété que tous les éléments du tableau `a` ont la même valeur `c`.

   (1 point) Prédicat `cte` dans le listing 1.

2. De même, définir en micro-C, par un prédicat `no_dup(int a[])`, la propriété que toutes les valeurs du tableau `a` sont deux à deux distinctes.

   (1 point) Prédicat `no_dup` dans le listing 1.

3. Utiliser ces deux prédicats pour formaliser en micro-C le lemme suivant : "Tout tableau constant qui a au moins deux éléments n'a pas tous ses éléments deux à deux distincts.".

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

4. Même si ce lemme est vrai et bien écrit en micro-C, l'interface de micro-C ne permet pas de le démontrer, car le prouveur ne sait pas bien choisir des indices particuliers dans le tableau comme témoins. Pour aider le prouveur, formalisez les lemmes suivants :

(a) "Dans tout tableau t constant qui a au moins deux éléments, les cases $t[0]$ et $t[1]$ sont égales", et

(b) "Tout tableau t qui a au moins deux éléments et dont les cases $t[0]$ et $t[1]$ sont égales a des éléments non deux à deux distincts."

(1 point) Lemmes `cte01` et `eq01_no_dup` dans le listing 1.

5. Où doivent être placés ces lemmes pour aider la démonstration du lemme de la question 3 ?

   (1 point) Pour permettre la preuve du lemme `cte_no_dup`, les lemmes `cte01` et `eq01_no_dup` doivent être placés avant.

```
/*@ predicate cte(int a[], int c) = forall i. 0 ≤ i < length(a) → a[i] == c;

    predicate no_dup(int a[]) =
      forall i,j. 0 ≤ i < j < length(a) → a[i] != a[j];

    lemma cte01: forall t[]. (exists c. cte(t,c)) && length(t) > 1 → t[0] == t[1];

    lemma eq01_no_dup: forall t[]. length(t) > 1 && t[0] == t[1] → !no_dup(t);

    lemma cte_no_dup:
      forall a[]. length(a) > 1 → forall c. cte(a,c) → !no_dup(a);
*/
```

Listing 1: Réponses de l'exercice 1.

2 Calcul du carré d'un nombre entier (4 points)

1. Définir une fonction C calculant le carré de son paramètre entier `x`, selon l'algorithme suivant :

$$i := 0;\ c := 0;\ \text{while } i < x \text{ do } c := c + 2i + 1;\ i := i + 1 \text{ od}$$

2. Spécifier le contrat de cette fonction en micro-C.
3. Spécifier un invariant de boucle assez précis pour permettre de démontrer la postcondition de ce contrat.
4. Spécifier un variant de boucle permettant de démontrer la terminaison de cette boucle.
5. Prouver le contrat avec l'interface de micro-C.

(code : 1 point, contrat : 1 point, invariant : 1 point, variant : 1 point)
Toutes les réponses sont dans le listing suivant :

```
int square (int x) {
/*@ requires x ≥ 0;
    ensures result == x * x; */
  int i = 0;
  int c = 0;
  while (i < x) {
  /*@ invariant c == i * i && i ≤ x;
      variant x - i; */
    c = c + 2 * i + 1;
    i = i+1;
  }
  /* Here i == x and c == i * i, so c == x*x */
  return(c);
}
```

3 Racine carrée entière par dichotomie (4 points)

Cet exercice porte sur le programme

$rmax := n + 1; rmin := 0;$
while $rmin + 1 \neq rmax$ do
  $m := (rmin + rmax + 1)$ **div** $2;$
  if $m^2 \leq n$ then $rmin := m$ else $rmax := m$ fi
od;
$r := rmin$

qui calcule la racine carrée entière r d'un nombre entier donné n par recherche dichotomique.

1. Traduire ce programme en une fonction micro-C de calcul de la racine carrée entière par défaut par dichotomie.
2. Spécifier le contrat de cette fonction.
3. Prouver que cette fonction termine et est conforme à son contrat, en annotant sa boucle.

(code : 1 point, contrat : 1 point, invariant : 1 point, variant : 1 point)
Toutes les réponses sont dans le listing suivant :

```
/*@ requires n ≥ 0;
    ensures result*result ≤ n < (result+1)*(result+1); */
  int rmin = 0;
  int rmax = n+1;
  int m = 0;

  while (rmin+1 != rmax) {
  /*@ invariant rmin * rmin ≤ n < rmax * rmax && rmin < rmax;
      variant rmax - rmin; */
    m = (rmin + rmax + 1) / 2;
    if (m * m ≤ n)
      rmin = m;
    else
      rmax = m;
  }
  return(rmin);
}
```

4 Initialisation d'un tableau (3 points)

Compléter le code suivant avec des spécifications permettant de prouver automatiquement avec l'interface de micro-C que la fonction initialise le tableau `tab` avec la valeur `v`.

```
void cteValue (int tab[], int n, int v) {
  int k = 0;
  while (k < n) {
    tab[k] = v;
    k = k + 1;
  }
}
```

(contrat : 1 point, invariants : 1 point, variant : 1 point)
Toutes les réponses sont dans le listing suivant :

```
void cteValue (int tab[], int n, int v) {
/*@ requires 0 ≤ n == length(tab);
    ensures forall j. 0 ≤ j < n → tab[j] == v; */
  int k = 0;
  while (k < n) {
  /*@ invariant 0 ≤ k ≤ n && forall j. 0 ≤ j < k → tab[j] == v;
      variant n - k; */
    tab[k] = v;
    k = k + 1;
  }
}
```

5 Calcul du maximum (4 points)

1. Définir en micro-C une fonction

   `int max (int t[], int n)`

   qui calcule l'indice d'un élément maximal dans tout tableau d'entiers `t` de longueur `n` non nulle.

2. Spécifier cette fonction et prouver ce contrat en ajoutant un invariant et un variant de boucle.

3. Vérifier que la preuve est automatique avec l'interface de micro-C.

(code : 1 point, contrat : 1 point, invariant : 1 point, variant : 1 point)
Toutes les réponses sont dans le listing suivant :

```
int max (int t[], int n) {
/*@ requires 0 < n == length(t);
    ensures 0 ≤ result < n;
    ensures forall j. 0 ≤ j < n → t[result] ≥ t[j]; */
  int imax = 0;
  int k = 0;
  while (k < n) {
  /*@ invariant 0 ≤ imax ≤ k;
      invariant imax < n;
      invariant 0 ≤ k ≤ n;
      invariant forall j. 0 ≤ j < k → t[imax] ≥ t[j];
      variant n - k; */
   if (t[imax] < t[k]) imax = k;
   k = k + 1;
  }
  return(imax);
}
```