Centre de Tele-enseignement Universitaire

UNIVERSITÉ DE FRANCHE-COMTÉ

SPP

Durée : 2 heures – **Session** 1 du 17 mai 2021, solution

Ce sujet comporte : 6 page(s)

Matériel autorisé : Aucun

Les questions qui commencent par le symbole ✎ doivent être rédigées **à la main**, sur des feuilles au format A4 portant le nom et le prénom de l'étudiant.e. Ces feuilles doivent être numérisées au format PDF ou JPEG, par scan ou photographie, et déposées sur Moodle dans le devoir prévu à cet effet, avant la fin de l'épreuve.

Les questions qui commencent par le symbole ⌨ doivent être traitées dans des fichiers C (fichiers texte avec le suffixe `.c`) commençant par un commentaire (entre `/*` et `*/`) donnant le nom et le prénom de l'étudiant.e. Ces fichiers doivent aussi être déposés sur Moodle dans le devoir prévu à cet effet, avant la fin de l'épreuve.

Exercice 1 : Racine carrée entière par excès (*9 points*)

Dans cet exercice, on considère une logique de Hoare dont toutes les constantes et toutes les variables sont des entiers relatifs (nombres positifs ou négatifs), et qui admet une addition, une multiplication et des comparaisons ($\leq$, $<$, $\geq$, $>$ et $=$) entre deux entiers relatifs.

Soit n un entier naturel ($n \geq 0$ en tant qu'entier relatif). Par définition, la **racine carrée entière par excès** de n est le plus petit entier naturel dont le carré est supérieur ou égal à n.

Question 1 : ✎ Dans le langage de la logique de Hoare, donner un programme qui calcule dans une variable s la racine carrée entière par excès d'un entier n **strictement positif** quelconque, par décrémentation à partir de la valeur initiale n. Ce programme doit être composé d'une initialisation, d'une boucle while et d'une instruction ($s := s - 1$) de décrémentation.

(1 point) $s := n$; while $(s-1)^2 \geq n$ do $s := s - 1$ od

Question 2 : ✎ Formaliser par un triplet de Hoare la précondition la plus générale possible et la postcondition la plus précise possible pour ce programme. La postcondition doit caractériser une valeur unique pour s, quel que soit l'entier strictement positif n.

(1 point) $\{n > 0\}$ $s := n$; while $(s-1)^2 \geq n$ do $s := s - 1$ od $\{(s-1)^2 < n \leq s^2\}$

Question 3 : ✎ Proposer un invariant pour la boucle while de ce programme, assez précis pour permettre de démontrer la validité de ce triplet de Hoare.

(1 point) La formule $n \leq s^2$ est un invariant de la boucle du programme précédent, suffisant pour démontrer la validité de ce triplet de Hoare.

Question 4 : ✎ En appliquant une par une les règles de la logique de Hoare, donner dans un tableau une démonstration formelle de la validité de ce triplet de Hoare.

(4 points : 1 point pour les lignes 1 à 4 + 0,5 point par ligne pour les lignes 5 à 10) Dans la logique de Hoare, une démonstration de validité de ce triplet de Hoare est :

N°	Triplet ou formule	Justification
1	$(s-1)^2 \geq n \wedge n \leq s^2 \Rightarrow (s-1)^2 \geq n$	évident
2	$\{(s-1)^2 \geq n\}\ s := s-1\ \{s^2 \geq n\}$	(*assignment*)
3	$s^2 \geq n \Rightarrow n \leq s^2$	évident
4	$\{(s-1)^2 \geq n \wedge n \leq s^2\}\ s := s-1\ \{n \leq s^2\}$	(*conseq.*) 1 2 3
5	$n > 0 \wedge s = n \Rightarrow n \leq s^2$	OK
6	$\{n \leq s^2\}$ while $(s-1)^2 \geq n$ do $s := s-1$ od $\{(s-1)^2 \ngeq n \wedge n \leq s^2\}$	(*while*) 4
7	$(s-1)^2 \ngeq n \wedge n \leq s^2 \Rightarrow (s-1)^2 < n \leq s^2$	OK
8	$\{n > 0\}\ s := n\ \{n > 0 \wedge s = n\}$	(*assignment*)
9	$\{n > 0 \wedge s = n\}$ while $(s-1)^2 \geq n$ do $s := s-1$ od $\{(s-1)^2 < n \leq s^2\}$	(*conseq.*) 5 6 7
10	$\{n > 0\}\ s := n$; while $(s-1)^2 \geq n$ do $s := s-1$ od $\{(s-1)^2 < n \leq s^2\}$	(*sequence*) 8 9

Les lignes marquées OK méritent une justification plus détaillée, à rédiger en dehors du tableau.

Question 5 : ⌨ Dans un fichier C, traduire ce triplet de Hoare en une fonction C

```
int sqrt(int n)
```

et sa spécification formelle (précondition, postcondition, invariant et variant de boucle). La fonction `sqrt` doit retourner la valeur que le programme de ce triplet calcule dans sa variable s. Un variant doit permettre de plus de démontrer la terminaison du programme.

Essayez de vérifier que ce programme satisfait sa spécification, avec l'interface en ligne `http://why3.lri.fr/micro-C/`.

Attention : il est possible que des limitations du prouveur automatique de cette interface ne permettent pas de tout démontrer automatiquement, même avec des spécifications correctes et assez précises.

(2 points) Voir le listing 1 : 1 point pour le code, 1 point pour la spécification.

```
int sqrt(int n) {
  /*@ requires 0 < n;
      ensures (result-1)*(result-1) < n <= result*result; */
  int s = n;
  while ((s-1)*(s-1) >= n) {
      /*@ invariant n <= s*s && 0 <= s;
          variant s; */
      s = s-1;
  }
  return(s);
}
```

Listing 1 – Solution de l'exercice sur la racine carrée en micro-C.

Au 8 juin 2021, l'interface en ligne ne permettait pas de démontrer l'inégalité non linéaire `n <= s*s`, néanmoins correcte.

Exercice 2 : Égalité de tableaux en micro-C (*5 points*)

Pour deux tableaux d'entiers `a` et `b`, la syntaxe du langage micro-C accepte l'expression `a == b` dans une spécification. L'objectif de cet exercice est d'utiliser la preuve pour comprendre la sémantique de cette égalité.

Question 1 : ⌨ On peut considérer que deux tableaux sont égaux s'ils ont la même longueur et s'ils contiennent les mêmes éléments aux mêmes indices. Dans un commentaire de bloc d'un fichier C, reproduire la définition

```
predicate eq(int a[], int b[]) = ..;
```

et remplacer les pointillés pour que ce prédicat soit une définition en micro-C de cette notion d'égalité entre les tableaux d'entiers `a` et `b`.

(1 point)

```
/*@ predicate eq(int a[], int b[]) = length(a) == length(b) &&
      forall i. 0 <= i < length(a) -> a[i] == b[i]; */
```

Question 2 : ✎ Lorsque les deux lemmes du listing 2 sont soumis à l'interface en ligne de micro-C, on obtient l'affichage reproduit dans la figure 1.

```
/*@ lemma eq_ext: forall a[]. forall b[]. a == b -> eq(a,b);
    lemma ext_eq: forall a[]. forall b[]. eq(a,b) -> a == b; */
```

Listing 2 – Lemmes sur l'égalité de deux tableaux en micro-C.

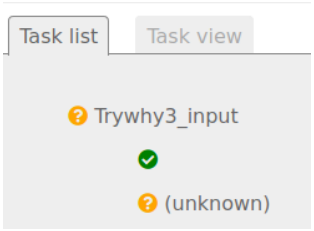

FIGURE 1 – Affichage de l'interface en ligne micro-C

Expliquer clairement et simplement la signification de cet affichage.

(1 point) Cet affichage signifie que le premier lemme a été démontré et que le second lemme n'a pas été démontré. Le prouveur étant supposé correct, ceci signifie que le premier lemme est vrai. Le prouveur n'étant pas complet, on ne peut rien déduire de l'absence de preuve pour le second lemme.

Question 3 : ✎ D'après ces résultats, vos connaissances et votre bon sens, proposer une signification pour l'expression `a == b` lorsqu'elle figure dans une spécification micro-C.

(1 point) D'après ces résultats, l'expression `a == b` dans une spécification micro-C ne signifie probablement pas que les deux tableaux ont la même longueur et le même contenu. Elle pourrait signifier, comme en C, que ces deux tableaux (qui sont des pointeurs en C) sont stockés à la même adresse, ce qui est une condition plus forte que l'égalité définie dans la première question.
On ne peut pas le vérifier en micro-C car la syntaxe `a == b` n'est pas acceptée dans un programme.

Question 4 : ⌨ Reproduire le listing 3 dans le même fichier C et compléter les pointillés pour que cette fonction vérifie ses postconditions.

```
int b_eq(int a[], int b[], int n) {
  /*@ requires n == length(a) == length(b);
      ensures result == 0 || result == 1;
      ensures result == 1 <-> eq(a,b); */
  int j = 0;

  while (..) {
    /*@ invariant ..;
        variant ..; */
    ..
    ..
  }
  ..
```

Listing 3 – Code à recopier et à compléter.

(2 points) 1 point pour le code, 1 point pour la spécification.

```
int b_eq(int a[], int b[], int n) {
  /*@ requires n == length(a) == length(b);
      ensures result == 0 || result == 1;
      ensures result == 1 <-> eq(a,b); */
  int j = 0;

  while (j < n) {
    /*@ invariant 0 <= j <= n;
        invariant forall i. 0 <= i < j -> a[i] == b[i];
        variant n-j; */
    if (a[j] != b[j]) return 0;
    j++;
  }
```

Exercice 3 : Transposition dans un tableau d'entiers (*6 points*)

L'exercice porte sur la fonction C suivante :

```
void arith_swap (int a[], int i, int j) {
  a[i] = a[i] + a[j];
  a[j] = a[i] - a[j];
  a[i] = a[i] - a[j];
}
```

Question 1 : ✎ Exprimer en français des conditions nécessaires et suffisantes pour que cette fonction `arith_swap` échange les valeurs stockées dans les cases d'indices `i` et `j` du tableau d'entiers `a`.

(1 point) La fonction `arith_swap` échange les valeurs stockées dans les cases d'indices `i` et `j` du tableau d'entiers `a` si et seulement si les entiers `i` et `j` sont compris entre les bornes `0` et `length(a)-1` du tableau et soit ils sont distincts, soit ils sont égaux et `a[i] == a[j] == 0` avant d'exécuter la fonction.
En effet, si les entiers `i` et `j` sont égaux, on obtient `a[i] == 2 * old(a[i])` après la première affectation et après l'exécution de la fonction, car `a[j] == 0` après la deuxième affectation. Dans ce cas, les valeurs ont été échangées si et seulement si `old(a[i]) == 0`.

Question 2 : ⌨ Dans un fichier C, formaliser ces conditions sous la forme d'une précondition micro-C pour la fonction `arith_swap`.

(1 point) Voir une solution dans le listing 4.

Question 3 : ⌨ Exprimer par des postconditions micro-C que la fonction `arith_swap` échange les valeurs stockées dans les cases d'indices `i` et `j` du tableau d'entiers `a`, **et ne modifie aucune autre valeur dans ce tableau**.

(2 points) Voir une solution dans le listing 4.

Question 4 : ⌨ Définir une fonction C

```
void swap (int a[], int i, int j)
```

qui a la même postcondition que la fonction `arith_swap`, mais qui utilise une variable locale pour échanger les valeurs stockées dans les cases d'indices `i` et `j` du tableau d'entiers `a`.

(1 point) Voir une solution dans le listing 4.

Question 5 : ⌨ Formaliser en micro-C une précondition correcte la plus générale possible pour la fonction `swap`.

Vérifiez toutes vos réponses à l'aide de l'interface en ligne `http://why3.lri.fr/micro-C/`.

(1 point) Voir réponse dans le listing 4.

```
void arith_swap_false (int a[], int i, int j) {
  /*@ requires 0 <= i < length(a) && 0 <= j < length(a);
      ensures a[i] == old(a[j]);
      ensures a[j] == old(a[i]);
      ensures forall k. 0 <= k < length(a) -> k != i -> k != j -> a[k] == old(a[k]); */
  a[i] = a[i] + a[j];
  a[j] = a[i] - a[j];
  a[i] = a[i] - a[j];
}

void arith_swap (int a[], int i, int j) {
  /*@ requires 0 <= i < length(a) && 0 <= j < length(a);
      requires i != j || (i == j && a[i] == 0);
      ensures a[i] == old(a[j]);
      ensures a[j] == old(a[i]);
      ensures forall k. 0 <= k < length(a) -> k != i -> k != j -> a[k] == old(a[k]); */
  a[i] = a[i] + a[j];
  a[j] = a[i] - a[j];
  a[i] = a[i] - a[j];
}

void swap (int a[], int i, int j) {
  /*@ requires 0 <= i < length(a) && 0 <= j < length(a);
      ensures a[i] == old(a[j]);
      ensures a[j] == old(a[i]);
      ensures forall k. 0 <= k < length(a) -> k != i -> k != j -> a[k] == old(a[k]); */
  int tmp;
  tmp = a[i];
  a[i] = a[j];
  a[j] = tmp;
}
```

Listing 4 – Solutions de l'exercice sur le swap arithmétique en micro-C.