Centre de Tele-enseignement Universitaire

UNIVERSITÉ DE FRANCHE-COMTÉ

SPP (Spécification et preuve de programmes)

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

Ce sujet comporte : 5 page(s)

Matériel autorisé : Aucun

Toutes les consultations de documents sont interdites. Toutes les utilisations de dispositifs électroniques sont interdites, sauf pour télécharger et imprimer ce sujet, utiliser l'interface en ligne de micro-C, numériser son travail, et le déposer sur Moodle comme détaillé ci-dessous. Le barème est donné à titre indicatif.

Les règles de la logique de Hoare et la syntaxe de la logique du langage micro-C sont rappelées dans l'annexe à la fin de ce document.

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.

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.

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.

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.

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.

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`.

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

```
/*@ 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 1 – Lemmes sur l'égalité de deux tableaux en micro-C.

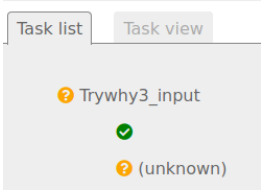

FIGURE 1 – Affichage de l'interface en ligne micro-C

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.

Question 4 : ⌨ Reproduire le listing 2 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 2 – Code à recopier et à compléter.

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`.

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

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**.

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`.

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/`.

Annexe : rappels de cours

Logique de Hoare

La logique de Hoare est le système formel <L, R> où

— L est l'ensemble des formules du premier ordre et des formules de la forme $\{P\}\ c\ \{Q\}$, appelées triplets de Hoare, où P et Q sont des formules de la logique du premier ordre et c est un programme du mini-langage défini par la grammaire

$$
\begin{array}{rll}
c, c' ::= & \mid x := e & \text{affectation} \\
& \mid c;\ c' & \text{séquence} \\
& \mid \mathsf{if}\ b\ \mathsf{then}\ c\ \mathsf{fi} & \text{conditionnelle simple} \\
& \mid \mathsf{if}\ b\ \mathsf{then}\ c\ \mathsf{else}\ c'\ \mathsf{fi} & \text{conditionnelle double} \\
& \mid \mathsf{while}\ b\ \mathsf{do}\ c\ \mathsf{od} & \text{répétition}
\end{array}
$$

— R est l'ensemble de règles de déduction de la figure 2.

$$\frac{}{\{P\ (e/x)\}\ x := e\ \{P\}}\ \textit{(assignment)} \qquad \frac{\{P\}\ c\ \{Q\} \qquad \{Q\}\ c'\ \{R\}}{\{P\}\ c; c'\ \{R\}}\ \textit{(sequence)}$$

$$\frac{\{b \wedge P\}\ c\ \{Q\} \qquad (\neg b \wedge P) \Rightarrow Q}{\{P\}\ \mathsf{if}\ b\ \mathsf{then}\ c\ \mathsf{fi}\ \{Q\}}\ \textit{(if-then)} \qquad \frac{\{b \wedge P\}\ c_1\ \{Q\} \qquad \{\neg b \wedge P\}\ c_2\ \{Q\}}{\{P\}\ \mathsf{if}\ b\ \mathsf{then}\ c_1\ \mathsf{else}\ c_2\ \mathsf{fi}\ \{Q\}}\ \textit{(if-then-else)}$$

$$\frac{\{b \wedge P\}\ c\ \{P\}}{\{P\}\ \mathsf{while}\ b\ \mathsf{do}\ c\ \mathsf{od}\ \{\neg b \wedge P\}}\ \textit{(while)} \qquad \frac{P \Rightarrow P' \qquad \{P'\}\ c\ \{Q'\} \qquad Q' \Rightarrow Q}{\{P\}\ c\ \{Q\}}\ \textit{(consequence)}$$

FIGURE 2 – Règles de la logique de Hoare.

Syntaxe de la logique du langage micro-C

La logique du langage micro-C doit être écrite dans un fichier C, soit sur une ligne qui commence par `//@`, soit dans un bloc de commentaires qui commence par `/*@` et se termine par `*/`.

Les types de micro-C sont les types C `void`, `int` (entiers relatifs des mathématiques) et `int ..[]` (tableaux d'entiers). Sur le type `int` des entiers sont définis le symbole fonctionnel unaire `-` (préfixé) et les symboles fonctionnels binaires infixés `+`, `-`, `*`, `/` et `%`, respectivement pour l'opposé, l'addition, la soustraction, la multiplication, le quotient et le reste de la division euclidienne.

La logique propositionnelle de micro-C contient les formules `true` et `false` pour "vrai" et "faux". Les connecteurs propositionnels de la négation, du "et", du "ou", de l'implication et de l'équivalence sont respectivement `!`, `&&`, `||` (comme en C), `->` et `<->`. Dans nos listings, ces symboles apparaissent sous la forme plus lisible !, && , || , -> et <-> .

La quantification universelle $\forall x$... s'écrit `forall x. ..` si `x` est un entier et `forall x[]. ..` si `x` est un tableau d'entiers. De même pour la quantification existentielle. Le quantificateur existentiel $\exists$ s'écrit `exists`. Dans nos listings, ces symboles apparaissent sous la forme plus lisible $\forall$ et $\exists$.

Le prédicat d'égalité infixé (`==`) est prédéfini sur les types des entiers et des tableaux d'entiers.

Sur le type `int`, les symboles relationnels binaires (infixés) `!=`, `<`, `>`, `<=` et `>=` sont aussi définis. Ils correspondent respectivement à la négation de l'égalité et aux relations d'ordre strictes et larges sur les entiers. Ces symboles sont notés $\neq$, $<$, $>$, $\leq$ et $\geq$ dans nos listings.

Dans la partie logique du langage micro-C le terme `length(a)` désigne la longueur du tableau d'entiers `a`. Comme en C, le terme `a[i]` désigne l'élément du tableau `a` à l'indice `i`, si cet indice est entre `0` et `length(a)-1` inclus.

En micro-C, la ligne

```
predicate p();
```

déclare une proposition atomique p, aussi appelée "variable propositionnelle". La ligne

```
predicate p(int i, int t[]);
```

déclare un prédicat binaire p dont le premier argument est un entier et le second argument est un tableau d'entiers. La ligne

```
function T f(int i, int t[]);
```

déclare une fonction binaire f dont le premier argument est un entier, le second argument est un tableau d'entiers, et qui calcule une donnée de type `T`, qui peut être `void` ou `int`. Les lignes

```
predicate p(int i, int t[]) = ..;
```

et

```
function T f(int i, int t[]) = ..;
```

déclarent un prédicat p et une fonction similaires, tout en donnant une définition à ce prédicat ou à cette fonction. Pour le prédicat, il faut remplacer les pointillés par une formule micro-C. Pour la fonction, il faut remplacer les pointillés par une expression de type `T`. La ligne

```
lemma l: ..
```

déclare un lemme nommé `l`. Les pointillés doivent être remplacés par une formule micro-C.

Pour décrire en micro-C la valeur d'une expression **avant** exécution d'une fonction, il faut entourer cette expression par `old(` et `)`, dans toute postcondition ou invariant d'une boucle de cette fonction.