Université de Lorraine
DIPLOME: Telecom Nancy 2A SLE et 2A ENSEM
Épreuve: MOVEX TP Noté
Durée du sujet : 1 h 30

Date : Lundi 19 mai 2025 de 8 h 00 à 9 h 30
Nom du rédacteur : Dominique Méry
Documents personnels autorisés

Il est recommandé de bien lire les questions. Les explications et les justifications doivent être aussi simples et claires que possible. Les documents sont autorisés à l'exclusion des documents qui vous seraient transmis durant l'épreuve. Le sujet comprend quatre (4) exercices. Vous devez rendre une archive zip et exclusivement zip contenant les fichiers demandés et le nom de cette archive sera votrenom-votreprenom.zip et elle produire un dossier de nom votrenom-votreprenom. Le sujet de votre message sera tpmalg. De plus, vous déposerez votre archive via Arche et selon le devoir qui vous concerne.
Vous pouvez ajouter un fichier ex.txt qui explique des éléments de vos solutions ou qui fait un commentaire.

TP Noté

Exercice 1 (4 points)

Soit le petit programme suivant annoté mais incomplet.

Listing 1: qassert6.c

```
1  /*@ requires  A(x,y,z) ;
2    ensures \result == 49 ;
3  */
4
5  int q6(int x,int y, int z){
6
7     z = y*(x+y);
8     y = x*y;
9     x=x*x;
10    z = z+x+y;
11 /*@  assert  z == 49; */
12
13   return z;
14   }
```

Proposer une assertion pour $A(x, y, z)$, afin que le contrat soit correct. L'assertion $A(x, y, z)$ doit être satisfaisable c'est-à-dire qu'il existe des valeurs entières pour x,y et z validant A(x,y,z). Vous ne pouvez pas utiliser l'assrtion `\FALSE`.

Exercice 2 (4 points)

Soit le petit programme suivant annoté mais incomplet.

Listing 2: qassert7.c

```
1  /*@ requires  A(x,y,z);
2    ensures \result == 144 ;
3  */
4
5  int q6(int x,int y, int z){
6    int  u;
7    u = x+y+z;
8    x=x*x;
9    /*@ assert  x == 9; */
10   y=y*y;
11  /*@ assert  y == 16; */
12   z=z*z;
13   u = u*u;
14     return u;
15   }
```

Proposer une assertion pour $A(x, y, z)$, afin que le contrat soit correct. L'assertion $A(x, y, z)$ doit être satisfaisable c'est-à-dire qu'il existe des valeurs entières pour x,y et z validant A(x,y,z). Vous ne pouvez pas utiliser l'assrtion `\FALSE`.

Exercice 3 6 points

Nous étudions ce petit algorithme qui calcule quelque chose et nous avons exécuté cet algorithme de 0 et 10 pour obtenir la suite suivante:

```
0 --> 0,1 --> 1,2 --> 3,3 --> 7,4 --> 15,5 --> 31,6 --> 63,7 --> 127,8 --> 255,9 --> 5
```

On comprend que l'algorithme calcule la suite u_n d'entiers telle que $\forall n \in \mathbb{N} : u_n = 2^n - 1$. De plus, une observation nous conduit à $u_0 = 0$ et $\forall n \in \mathbb{N} : u_{n/+1} = 2 * u_n + 1$.

Les deux fichiers qmathinv1.h et qinv1.c définissent les éléments C nécessaires, pour écrire la correction partielle et la terminaison de la fonction inv1.

Listing 3: qmathinv1.h

```
#ifndef _A_H
#define _A_H
// Definition of the  mathematical function mathpower2
/*@ axiomatic mathpower {
  @ logic integer mathpower(integer n,integer m);
  @ axiom mathpower_0: \forall integer n; n >= 0 ==> mathpower(n,0) == 1;
  @ axiom mathpower_in: \forall integer n,m; n >= 0 && m >= 0
  ==> mathpower(n,m+1) == mathpower(n,m)*n;
  @ } */

int inv1(int x);
#endif
```

Listing 4: qinv1.c

```
#include <limits.h>
#include <qmathinv1.h>

int inv1(int x)
{ int u=0;
  int k=0;
   while (k  < x)
    { u=2*u+1;
      k=k+1;
    };
  return(u);
    }
```

Compléter les fichiers, afin de montrer que la fonction inv1 calcule correctement la valeur de la suite u_x pour $x \geq 0$. Il est important de choisir un invariant de boucle et un variant, afin que frama-c permette de prouver automatiquement toutes les conditions de vérification engendrées.

Exercice 4 6 points

La fonction `power3` calcule la puissance 3 de x et satisfait l'invariant de boucle indiqué. De plus, le contrat est donné pour exprimer la correction partielle et la terminaison de power3. La racine cubique rc entière de x est le nombre entier rc dont le cube est le plus proche inférieurement de x: $rc^3 \leq x < (rc+1)^3$. On note $rootcubique$ la fonction qui calcule cet entier:

- $rootcubique(0) = 0$
- $rootcubique(1) = 1$
- $rootcubique(27) = 3$
- $rootcubique(30) = 3 \ldots$

La fonction f donnée ci-dessous permet de calcul une paire constituée de deux champs d'une part r qui contient la valeur de la racine cubique calculée et d'autre part la valeur du cube de r. Pour reprendre nos exemples, on pourrait avoir:

- $f(0) = (0, 0)$
- $f(1) = (1, 1)$
- $f(27) = (3, 27)$
- $f(30) = (3, 27) \ldots$

Cette fonction est construite à partir de la fonction power3 et modifie le test $k < x$ sous la forme $cz \leq x$ et on récupère les valeurs de ocz et de k calculées. Une partie de l'invariant de power3 peut être utilisée pour cette fonction f mais il faut ajouter des éléments pour ocz.

Listing 5: qarootcubique.c

```
struct paire {
    unsigned r;
    unsigned p;
};



struct paire f(int x)
{int  cz,cv,cu,cw,ct, k,ocz;
  struct paire r;
  cz=0;cv=0;cw=1;ct=3;cu=0;k=0;ocz=-1;
  while (cz<=x)
        {
          ocz = cz;
          cz=cz+cv+cw;
          cv=cv+ct;
          ct=ct+6;
          cw=cw+3;
          cu=cu+1;
          k=k+1;}
  r.r=k-1;r.p=ocz;
 return(r);}
```

Listing 6: qapower3.c

```
#include <limits.h>
/*@  requires  0 <= x;
     ensures \result ==x*x*x;
*/
int power3(int x)
{int  r,ocz,cz,cv,cu,ocv,cw,ocw,ct,oct,ocu,k,ok;
  cz=0;cv=0;cw=1;ct=3;cu=0; ocw=cw;ocz=cz;
  oct=ct;ocv=cv;ocu=cu;k=0;ok=k;
       /*@
         @ loop invariant cu  == k;
         @ loop invariant ct == 6*cu +3;
         @ loop invariant cv== 3*cu*cu;
         @ loop invariant cw == 3*cu+1;
         @ loop invariant cz == k*k*k;
         @ loop invariant k  <= x;
         @ loop invariant 6*cw == 3*ct-3;
         @ loop assigns ct,oct,cu,ocu,cz,ocz,k,cv,ocv,cw,ocw,r,ok;
         @ loop assigns ocv,ocw;
          @ loop variant x-k;
*/
```

```
21    while (k<x)
22          {
23             ocz=cz; ok=k; ocv=cv; ocw=cw; oct=ct; ocu=cu;
24             cz=ocz+ocv+ocw;
25             cv=ocv+oct;
26             ct=oct+6;
27             cw=ocw+3;
28             cu=ocu+1;
29             k=ok+1;}
30    r=cz; return(r);}
```

Compléter le fonction f en ajoutant un contrat assurant la correction partielle par rapport à ce qui est décrit ci-dessus c'est-à-dire que la paire renvoyée contient pour sa composante r la valeur de la racine cubique de x et pour la seconde composante une valeur à définir.

Ajouter un invariant de boucle permattant de valider automatiquement le contrat et un variant pour la terminaison.