Cours MOdélisation, Vérification et EXpérimentations
Exercices
Sémantique des langages de programmation
par Dominique Méry
23 mai 2025

Exercices sur Frama-c et wp (I)

Exercice 1 **Question 1.1** *Soit le petit programme suivant annoté mais incomplet.*

```
/*@ requires  A(x,y,z) ;
  ensures \result == 49 ;
*/

int q6(int x,int y, int z){

   z = y*(x+y);
   y = x*y;
   x=x*x;
   z = z+x+y;
/*@  assert  z == 49; */

  return z;
  }
```

En utilisant l'opérateur wp, proposer des assertions pour $A(x,y,z)$, afin que le contrat soit correct.

Question 1.2 *Soit le petit programme suivant annoté mais incomplet.*

```
/*@ requires  A(x,y,z);
  ensures \result == 144 ;
*/

int q7(int x,int y, int z){
  int  u;
  u = x+y+z;
  x=x*x;
   /*@ assert  x == 9;*/
  y=y*y;
 /*@ assert  y == 16;*/
  z=z*z;
  u = u*u;
    return u;
  }
```

En utilisant l'opérateur wp, proposer une assertion pour A, afin que le contrat soit correct. Les annotations indiquées sont correctes et font partie des données du problème.

Exercice 2 *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,
```

```
#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

#include <limits.h>
#include <qmathiinv1.h>

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

Si on utilise la fonction power2, on obtient la suite suivante :

```
0 --> 1,1 --> 2,2 --> 4,3 --> 8,4 --> 16,5 --> 32,6 --> 64,7 --> 128,8 --> 256,9 --
```

Question 2.1 *On comprend que l'algorithme calcule la suite u_n d'entiers telle que $\forall n \in \mathbb{N}$: $u_n = 2^n - 1$. En particulier, $u_0 = 0$.*
Donner une définition de u_{n+1} en fonction de u_n en calculant le rapport $\frac{u_{n+1}+1}{u_n+1}$

Question 2.2 *Ecrire un contrat pour cette algorithme en précisant la clause* requires *et la clause* ensures*.*

Question 2.3 *Proposer un invariant de boucle en vous aidant de la suite u_n et monter qu'il est correct pur cette preuve de correction.*

Question 2.4 *Exprimer la terminaison de cet algorithme et justifier qu'il termine pour la précondition choisie.*

Exercice 3 *On dit que S1 est équivalent à S2 et on note $S1 \equiv S2$, si pour touts les états s et s', $(S1, s) \underset{nat}{\longrightarrow} s'$ si, et seulement si, $(S2, s) \underset{nat}{\longrightarrow} s'$.*

Question 3.1 *Montrer que* $\textbf{while}\ b\ \textbf{do}\ S\ \textbf{od} \equiv \textbf{if}\ b\ \textbf{then}\ S; \textbf{while}\ b\ \textbf{do}\ S\ \textbf{od}\ \textbf{else}\ skip\ \textbf{fi}$

Question 3.2 *Etendre la fonction sémantique pour l'instruction* $\textbf{repeat}\ S\ \textbf{until}\ b$.

Question 3.3 *Montrer que* $\textbf{repeat}\ S\ \textbf{until}\ b\ \equiv\ S; \textbf{if}\ b\ \textbf{then}\ skip\ \textbf{else}\ \textbf{repeat}\ S\ \textbf{until}\ b\ \ \textbf{fi}$

Exercice 4 *On rappelle que $wp(X := E)(P(x)) = P[e(x)/x]$ et que $\{A(x)\}X := E\{B(x)\}$ est définie par $A \Rightarrow wp(X := E)(B)$. On peut assez naturellement appliquer cette définition pour*

$$\boxed{\begin{array}{l} \ell_1 : A(x) \\ X := E(X) \\ \ell_2 : B(x) \end{array}}$$

Montrer la correction des triplets suivants et vérifier avec Frama-C en examinant les conditions de vérification engendrées :

— $\ell_1 : x = 10 \ \wedge \ y = z{+}x \ \wedge z = 2{\cdot}x$
$y := z{+}x$
$\ell_2 : x = 10 \ \wedge \ y = x{+}2{\cdot}10$

— $\ell_1 : x = 1 \ \wedge \ y = 12$
$x := 2{\cdot}y$
$\ell_2 : x = 1 \ \wedge \ y = 24$

— $\ell_1 : x = 11 \ \wedge \ y = 13$
$z := x; x := y; y := z;$
$\ell_2 : x = 26/2 \ \wedge \ y = 33/3$

— $\ell_1 : x = 9 \ \wedge \ y = z{+}x$
$y := x{+}9$
$\ell_2 : x = 9 \ \wedge \ y = x{+}9$

— $\ell_1 : x = 1 \ \wedge \ y = 3 \ \wedge \ x{+}y = 12$
$x := y{+}x$
$\ell_2 : x = 567 \ \wedge \ y = 34$

Exercice 5 *Calculer wp(S)(P) dans les cas suivants :*

1. $wp(X := E(X); Y := F(X))(P(x,y))$
2. $wp(X := Y, Y := X)(P(x,y))$
3. $wp(while\ TRUE\ do\ X := E(X)\ od)(P(x,y))$
4. $wp(while\ FALSE\ do\ X := E(X)\ od)(P(x,y))$
5. $wp(while \times < 20\ do\ X := X{+}1\ od)(TRUE)$

Sémantique naturelle et sémantique SOS

Exercice 6

$$
\begin{array}{lcl}
n & ::= & 0 \mid 1 \mid n0 \mid n1 \\
e & ::= & n \mid x \mid e1{+}e2 \mid e1{-}e2 \mid e1{\cdot}e2 \\
b & ::= & tt \mid ff \mid e1 = e2 \mid e1 \neq e2 \mid e1 \leq e2 \mid e1 \geq e2 \mid e1 < e2 \mid e1 > e2 \mid \neg\, b \mid b1 \ \&\& \ b2 \\
S & ::= & x := e \mid skip \mid S1; S2 \mid (\textbf{if}\ b\ \textbf{then}\ S_1\ \textbf{else}\ S_2\ \textbf{fi} \mid \textbf{while}\ b\ \textbf{do}\ S\ \textbf{od}
\end{array}
$$

Question 6.1 *Définir une fonction sémantique pour la catégorie syntaxique des chaines numériques* NUM *à valeurs dans* $\mathbb{Z}$ *:* $\mathcal{N} \in NUM \longrightarrow \mathbb{Z}$.

Question 6.2 *Evaluer les valeurs suivantes :*
— $\mathcal{N}(11)$
— $\mathcal{N}(101)$
— $\mathcal{N}(0100)$

Question 6.3 *Montrer que* $\mathcal{N}$ *est bien définie pour toutes les expressions.*

Exercice 7 *On définit lénsemble des états* $States = Var \longrightarrow \mathbb{Z}$ *où* Var *est lénsemble des variables.*

Question 7.1 *Une expression arithmétique $e \in Exp$ est évaluée dans un état ar la fonction sémantique $\mathcal{E} \in Exp \longrightarrow (States \longrightarrow \mathbb{Z})$.*
Définir $\mathcal{E}$ par induction sur la syntaxe.

Question 7.2 *Soit $s \in States$ tel que $s(x) = 2$ et $s(y) = 3$ où $x, y \in Var$ et $s \in States$.*
Evaluer les expressions suivantes en s : $x+y+101$, $x \cdot y$.

Question 7.3 *Une expression logique $b \in Bexp$ est évaluée dans un état ar la fonction sémantique $\mathcal{B} \in Bexp \longrightarrow (States \longrightarrow \mathbb{B})$.*
Définir $\mathcal{B}$ par induction sur la syntaxe.

Question 7.4 *Soit $s \in States$ tel que $s(x) = 2$ et $s(y) = 3$ où $x, y \in Var$ et $s \in States$.*
Evaluer les expressions suivantes en s : $x = y$, $x \neq y$, $x \leq y$, $x < y \;\&\&\; x + -6 \leq y$.

Question 7.5 *On étend le langage des expressions logiques par les deux constructions $b1 \Rightarrow b2$ et $b1 \Leftrightarrow b2$. Ce langage est noté $Bexp1$.*
Montrer que pour tout expression $b \in Bexp1$, il existe une expression $b' \in Bexpt$ telle que $\mathcal{B}(b) = \mathcal{B}(b')$.

Exercice 8 *Nous définissons deux opérations substitution et mise à jour. Ces deux opérations seront utilisées plus tard dans léxpression de la sémantique des instructions :*

— *la notation de substitution $e[x \mapsto e1]$ qui est la substitution de x par $e1$ dans e.*
— *la mise à jour pour un état s et on la note $s[x \mapsto v]$ qui est le nouvel état obtenu par mise à jour de la valeur de x pour s.*

Question 8.1 *Ecrire une définition inductive de $e[x \mapsto f]$.*

Question 8.2 *Définir la mise à jour pour un état s et on la note $s[x \mapsto v]$ qui est le nouvel état obtenu par mise à jour de la valeur de x pour s.*

Question 8.3 *Montrer que $s[x \mapsto v][y \mapsto w] = s[y \mapsto w][x \mapsto v]$ et que $s[x \mapsto v][\mapsto w] = s[x \mapsto v]$.*

Question 8.4 *Montrer que $\mathcal{E}(e[x \mapsto f])(s) = \mathcal{E}(e)(s[x \mapsto \mathcal{E}(f)(s)$.*

Question 8.5 *Définir la substitution pour les expressions booléennes $b[x \mapsto e]$ où b est une expression booléenne de BExp et e est une expression arithmétqiue de Exp.*

Question 8.6
Montrer que $\mathcal{E}(b[x \mapsto e])(s) = \mathcal{E}(b)(s[x \mapsto \mathcal{E}(e)(s)$.

Exercice 9
On rappelle les règles définissant la sémantique naturelle du langage de programmation $\mathcal{PL}$

Règles de définition selon la syntaxe

Axiome Ass $(x := e, s) \underset{nat}{\longrightarrow} s[x \mapsto \mathcal{E}(e)(s)]$
Axiome Skip $(skip, s) \underset{nat}{\longrightarrow} s$
Règle Comp *Si* $(S_1, s) \underset{nat}{\longrightarrow} s'$ *et* $(S_2, s') \underset{nat}{\longrightarrow} s"$, *alors* $(S_1; S_2, s) \underset{nat}{\longrightarrow} s"$.
Règle Iftt *Si* $(S_1, s) \underset{nat}{\longrightarrow} s'$ *et* $\mathcal{B}(b)(s) = TRUE$, *alors* $(\textbf{if } b \textbf{ then } S_1 \textbf{ else } S_2 \textbf{ fi}, s) \underset{nat}{\longrightarrow} s'$.
Règle Ifff *Si* $(S_2, s) \underset{nat}{\longrightarrow} s'$ *et* $\mathcal{B}(b)(s) = FALSE$, *alors* $(\textbf{if } b \textbf{ then } S_1 \textbf{ else } S_2 \textbf{ fi}, s) \underset{nat}{\longrightarrow} s'$.
Règle Whilett] Si $(S, s) \underset{nat}{\longrightarrow} s'$ *et* $(\textbf{while } b \textbf{ do } S \textbf{ od}, s') \underset{nat}{\longrightarrow} s"$ *et* $\mathcal{B}(b)(s) = TRUE$, *alors* $(\textbf{while } b \textbf{ do } S \textbf{ od}, s) \underset{nat}{\longrightarrow} s"$.
Règle Whieff] Si $\mathcal{B}(b)(s) = FALSE$, *alors* $(\textbf{while } b \textbf{ do } S \textbf{ od}, s) \underset{nat}{\longrightarrow} s$.

Question 9.1 *Soit s tel que $s(u) = 0$ et $s(v) = 1$.*

— *Evaluer (u :=11 ;v :=u+100 ;u :=u+v,s) en sémantique naturelle.*
— *Evaluer (w :=u ;u :=v ;v :=w,s) en sémantique naturelle.*

Exercices sur Frama-c et wp (II)

Exercice 10 *Soit le petit programme suivant annoté mais incomplet.*

Listing 1 – qassert6.c

```
/*@ requires  A(x,y,z) ;
  ensures \result == 49 ;
*/

int q6(int x,int y, int z){

   z = y*(x+y);
   y = x*y;
   x=x*x;
   z = z+x+y;
/*@  assert  z == 49; */

  return z;
  }
```

Proposer une assertion pour $A(x, y, z)$, afin que le contrat soit correct. L'assertion $A(x, y, z)$ doit Ãªtre 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 11 *Soit le petit programme suivant annoté mais incomplet.*

Listing 2 – qassert7.c

```
/*@ requires  A(x,y,z);
  ensures \result == 144 ;
*/

int q7(int x,int y, int z){
  int  u;
  u = x+y+z;
  x=x*x;
  /*@ assert  x == 9;*/
  y=y*y;
 /*@ assert  y == 16;*/
  z=z*z;
  u = u*u;
    return u;
  }
```

Proposer une assertion pour $A(x, y, z)$, afin que le contrat soit correct. L'assertion $A(x, y, z)$ doit Ãªtre 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 12 *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,
```

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 $\forall n \in \mathbb{N} : u_{n/+1} = 2 \cdot u_n + 1$.
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 13 *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 Ãªtre utilisée pour cette fonction f mais il faut ajouter des éléments pour ocz.

Listing 5 – qarootcubique.c

```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

```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;
*/
  while (k<x)
        {
          ocz=cz;ok=k;ocv=cv;ocw=cw;oct=ct;ocu=cu;
          cz=ocz+ocv+ocw;
          cv=ocv+oct;
          ct=oct+6;
          cw=ocw+3;
          cu=ocu+1;
          k=ok+1;}
  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.