Cours Modélisation et vérification des systèmes informatiques
Exercices (avec les corrections)
Utilisation d'un environnement de vérification Frama-c (II)
par Dominique Méry
21 novembre 2024

**Exercice 1** *Définir une fonction* maxpointer *(gex1.c)calculant la valeur du maximum du ciontenu de deux adresses avec son contrat.*

```
int max_ptr ( int *p, int *q ) {
if ( *p >= *q ) return *p ;
return *q ; }
```

◇— **Solution de l'exercice 1**

Listing 1 – schema de contrat

```
// frama-c-gui -wp -wp-rte  -report -wp-print  maxpointer.c

/*@ requires \valid(p)  && \valid(q);
    ensures \result >= *p && \result >= *q;
    ensures \result == *p || \result == *q;
*/
int max_ptr ( int *p, int *q ) {
if ( *p >= *q ) return *p ;
return *q ; }
```

**Fin 1**

**Exercice 2** *Définir une fonction* abs *(gex2.c) calculant la valeur absolue d'un nombre entier avec son contrat.*

```
#include <limits.h>
int abs (int x) {
  if (x >= 0) return x ;
  return -x;}
```

◇— **Solution de l'exercice 2**

Listing 2 – schema de contrat

```
#include <limits.h>

/*@ requires x > INT_MIN;
    assigns \nothing;
    behavior pos:
      assumes x >= 0;
      ensures \result == x;
    behavior neg:
      assumes x < 0;
      ensures \result == â x ;
     complete behaviors;
     disjoint behaviors;
*/
int abs (int x) {
  if (x >= 0) return x ;
  return -x;}
```

**Fin 2**

**Exercice 3** *Etudier les fonctions pour la vérification de l'appel de abs et max (max-abs.c,max-abs1.c,max-abs2.c)*

```
int abs ( int x );
int max ( int x, int y );
// returns maximum of absolute values of x and y
int max_abs( int x, int y ) {
x=abs(x); y=abs(y);
return max(x,y);
}
```

◇– **Solution de l'exercice 3**

```
nt abs ( int x );
int max ( int x, int y );
// returns maximum of absolute values of x and y
int max_abs( int x, int y ) {
x=abs(x); y=abs(y);
return max(x,y);
}

#include <limits.h>
/*@ requires x > INT_MIN;
    ensures (x >= 0 ==> \result == x) && (x < 0 ==> \result ==âx);
    assigns \nothing ;
*/
int abs ( int x );
/*@ ensures \result >= x && \result >= y;
    ensures \result == x || \result == y;
    assigns \nothing ;
*/
int max ( int x, int y );
/*@ ensures \result >= x && \result >= âx && \result >= y && \result >= ây;
    ensures \result == x || \result == âx || \result==y || \result == ây;
    assigns \nothing ;
*/
// returns maximum of absolute values of x and y
int max_abs( int x, int y ) {
x=abs(x); y=abs(y);
return max(x,y);
}

#include <limits.h>
/*@ requires x > INT_MIN;
    ensures (x >= 0 ==> \result == x) && (x < 0 ==> \result ==âx);
    assigns \nothing ;
*/

int abs ( int x );
/*@ ensures \result >= x  && \result >= y;
    ensures \result == x || \result == y;
    assigns \nothing ;
```

```
*/
int max ( int x, int y );
/*@ requires x > INT_MIN;requires y > INT_MIN;
    ensures \result >= x && \result >= âx && \result >= y && \result >= ây;
    ensures \result == x || \result == âx || \result==y || \result==ây;
    assigns \nothing ;
*/
// returns maximum of absolute values of x and y
int max_abs( int x, int y ) {
x=abs(x); y=abs(y);
return max(x,y);
}
```

**Fin 3**

---

**Exercice 4** **Question 4.1** *Soit la fonction suivante calculant le reste de la division de a par b. Vérifier la correction de cet algorithme.*

```
int rem(int a, int b) {
  int r = a;
  while (r >= b) {
    r = r - b;
  };
  return r;
}
```

*Il faut utiliser une variable ghost.*

◇— **Solution de la question 4.1**

```
/*@ requires a >= 0 && b >= 0;
  ensures 0 <= \result;
  ensures \result < b;
  ensures \exists integer k; a == k * b + \result;
*/
int rem(int a, int b) {
  int r = a;
  /*@
    loop invariant
    (\exists integer i; a == i * b + r) &&
    r >= 0
    ;
    loop assigns r;
   */
  while (r >= b) {
    r = r - b;
  };
  return r;
}
```

**Fin 4.1**

---

**Question 4.2** *Soit la fonction suivante calculant la fonction fact. Vérifier la correction de cet algorithme. Pour vérifier cette fonction, il est important de définir la fonction mathématique Fact avec ses propriétés.*

```
/*@ axiomatic Fact {
  @ logic integer Fact(integer n);
  @ axiom Fact_1: Fact(1) == 1;
```

```
  @ axiom Fact_rec: \forall integer n; n > 1 ==> Fact(n) == n * Fact(n-1);
  @ } */

int fact(int n) {
  int y = 1;
  int x = n;
  while (x != 1) {
    y = y * x;
    x = x - 1;
  };
  return y;
```

◇— **Solution de la question 4.2**

```
/*@ axiomatic Fact {
  @ logic integer Fact(integer n);
  @ axiom Fact_1: Fact(1) == 1;
  @ axiom Fact_rec: \forall integer n; n > 1 ==> Fact(n) == n * Fact(n-1);
  @ } */

/*@ requires n > 0;
  ensures \result == Fact(n);
*/
int fact(int n) {
  int y = 1;
  int x = n;
  /*@ loop invariant x >= 1 &&
                     Fact(n) == y * Fact(x);
    loop assigns x, y;
   */
  while (x != 1) {
    y = y * x;
    x = x - 1;
  };
  return y;
```

**Fin 4.2**

**Question 4.3** *Annoter les fonctions suivantes en vue de montrer leur correction.*

```
 int max (int a, int b) {
  if (a >= b) return a;
  else return b;
}

int indice_max (int t[], int n) {
  int r = 0;
  for (int i = 1; i < n; i++)
    if (t[i] > t[r]) r = i;
  return r;
}



int valeur_max (int t[], int n) {
  int r = t[0];
```

```
  for (int i = 1; i < n; i++)
    if (t[i] > r) r = t[i];
  return r;
}
```

'La solution est donnée dans le fichier gex4-3.c.

◇— **Solution de la question 4.3**

```
/*@  ensures \result >= a;
  ensures \result >= b;
  ensures \result == a || \result == b;
*/
int max (int a, int b) {
  if (a >= b) return a;
  else return b;
}

/*@
  requires n > 0;
  requires \valid(t+(0..n-1));
  ensures 0 <= \result < n;
  ensures \forall int k; 0 <= k < n ==>
    t[k] <= t[\result];
*/
int indice_max (int t[], int n) {
  int r = 0;
  /*@ loop invariant 0 <= r < i <= n
    && (\forall int k; 0 <= k < i ==> t[k] <= t[r])
    ;
    loop assigns i, r;
  */
  for (int i = 1; i < n; i++)
    if (t[i] > t[r]) r = i;
  return r;
}


/*@
  requires n > 0;
  requires \valid(t+(0..n-1));
  ensures \forall int k; 0 <= k < n ==>
    t[k] <= \result;
  ensures \exists int k; 0 <= k < n && t[k] == \result;
*/
int valeur_max (int t[], int n) {
  int r = t[0];
  /*@ loop invariant 0 <= i <= n
    && (\forall int k; 0 <= k < i ==> t[k] <= r)
    && (\exists int k; 0 <= k < i && t[k] == r)
    ;
    loop assigns i, r;
  */
  for (int i = 1; i < n; i++)
    if (t[i] > r) r = t[i];
```

```
  return r;
}
```

**Fin 4.3**

*La solution sous la forme d'un fichier c est la suyivante :*

Listing 3 – schema de contrat

```
/*@  assigns \nothing;
     ensures \result >= a;
  ensures \result >= b;
  ensures \result == a || \result == b;
*/
int max (int a, int b) {
  if (a >= b) return a;
  else return b;
}



/*@  assigns \nothing;
     ensures \result >= a;
  ensures \result >= b;
  ensures \result == a || \result == b;
*/
int max2 (int a, int b) {
  int r;
  if (a >= b)
    { r=a;}
  else
    {r=b;};
  return r;
}

/*@
  requires n > 0;
  requires \valid(t+(0..n-1));
  assigns \nothing;

  ensures 0 <= \result < n;
  ensures \forall int k; 0 <= k < n ==>     t[k] <= t[\result];
*/
int indice_max (int t[], int n) {
  int r = 0;
  /*@ loop invariant 0 <= r < i <= n
    && (\forall int k; 0 <= k < i ==> t[k] <= t[r])
    ;
    loop assigns i, r;
  */
  for (int i = 1; i < n; i++)
    if (t[i] > t[r]) r = i;
  return r;
}


/*@
  requires n > 0;
```

```
  requires \valid(t+(0..n-1));
  assigns \nothing;

  ensures \forall int k; 0 <= k < n ==>
    t[k] <= \result;
  ensures \exists int k; 0 <= k < n && t[k] == \result;
*/
int valeur_max (int t[], int n) {
  int r = t[0];
  /*@ loop invariant 0 <= i <= n
    && (\forall int k; 0 <= k < i ==> t[k] <= r)
    && (\exists int k; 0 <= k < i && t[k] == r)
    ;
    loop assigns i, r;
  */
  for (int i = 1; i < n; i++)
    if (t[i] > r) r = t[i];
  return r;
}
```

**Exercice 5** *Pour chaque question, montrer que l' annotation est correcte ou incorrecte selon les conditions de vérifications énoncées comme suit*
$\forall x, y, , x', y'.P_\ell(x, y) \ \wedge \ cond_{\ell,\ell'}(x, y) \ \wedge \ (x', y') = f_{\ell,\ell'}(x, y) \Rightarrow P_{\ell'}(x', y')$
*Pour cela, on utilisera l'environnement Frama-c.*

**Question 5.1**

| |
|---|
| $\ell_1 : x = 10 \ \wedge \ y = z{+}x \ \wedge z = 2{\cdot}x$<br>$y := z{+}x$<br>$\ell_2 : x = 10 \ \wedge \ y = x{+}2{\cdot}10$ |

$\diamond$— **Solution de la question 5.1**

```
frama-c  -wp -rte -print  hoare1.c
[kernel] Parsing hoare1.c (with preprocessing)
[rte] annotating function q1
[wp] 4 goals scheduled
[wp] Proved goals:    4 / 4
    Qed:             4
/* Generated by Frama-C */
int q1(void)
{
  int __retres;
  int x = 10;
  int y = 30;
  int z = 20;
  /*@ assert x â¡ 10 â§ y â¡ z + x â§ z â¡ 2 * x; */ ;
  /*@ assert rte: signed_overflow: -2147483648 â¤ z + x; */
  /*@ assert rte: signed_overflow: z + x â¤ 2147483647; */
  y = z + x;
  /*@ assert x â¡ 10 â§ y â¡ x + 2 * 10; */ ;
  __retres = 0;
  return __retres;
}
```

**Fin 5.1**

---

**Question 5.2**

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

**Question 5.3**

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

**Exercice 6** *(6 points)*
*Evaluer la validité de chaque annotation dans les questions suivent.*

**Question 6.1**

$$\ell_1 : x = 64 \ \wedge \ y = x{\cdot}z \ \wedge z = 2{\cdot}x$$
$$Y := X{\cdot}Z$$
$$\ell_2 : \ y{\cdot}z = 2{\cdot}x{\cdot}x{\cdot}z$$

**Question 6.2**

$$\ell_1 : x = 2 \ \wedge \ y = 4$$
$$Z := X{\cdot}Y{+}3{\cdot}Y{\cdot}Y{+}3{\cdot}X{\cdot}Y{\cdot}Y{+}X^6$$
$$\ell_2 : \ z = 6{\cdot}(x{+}y)^2$$

**Question 6.3**

$$\ell_1 : x = z \ \wedge \ y = x{\cdot}z$$
$$Z := X{\cdot}Y{+}3{\cdot}Y{\cdot}Y{+}3{\cdot}X{\cdot}Y{\cdot}Y{+}Y{\cdot}X{\cdot}Z{\cdot}Z{\cdot}X;$$
$$\ell_2 : \ z = (x{+}y)^3$$

*Soit l'annotation suivante :*

$$\ell_1 : x = 1 \wedge y = 2$$
$$X := Y{+}2$$
$$\ell_2 : x{+}y \geq m$$

*où $m$ est un entier ($m \in \mathbb{Z}$).*

**Question 6.4** *Ecrire la condition de vérification correspondant à cette annotation en upposant que $X$ et $Y$ sont deux variables entières.*

**Question 6.5** *Etudier la validité de cette condition de vérification selon la valeur de $m$.*

**Exercice 7** *gex7.c*

*VARIABLES* $N, V, S, I$

$$pre(n_0, v_0, s_0, i_0) \stackrel{def}{=} \begin{cases} n_0 \in \mathbb{N} \wedge n_0 \neq 0 \\ v_0 \in 0..n_0{-}1 \longrightarrow \mathbb{Z} \\ s_0 \in \mathbb{Z} \wedge i_0 \in \mathbb{Z} \end{cases}$$

*REQUIRES* $\left( \begin{array}{l} n_0 \in \mathbb{N} \wedge n_0 \neq 0 \\ v_0 \in 0..n_0{-}1 \longrightarrow \mathbb{Z} \end{array} \right)$

*ENSURES* $\left( \begin{array}{l} s_f = \bigcup_{k=0}^{n_0-1} v_0(k) \\ n_f = n_0 \\ v_f = v_0 \end{array} \right)$

$\ell_0 : \left( \begin{array}{l} pre(n_0, v_0, s_0, i_0) \\ (n, v, s, i) = (n_0, v_0, s_0, i_0) \end{array} \right)$

$S := V(0)$

$\ell_1 : \left( \begin{array}{l} pre(n_0, v_0, s_0, i_0) \\ s = \bigcup_{k=0}^{0} v(k) \\ (n, v, i) = (n_0, v_0, i_0) \end{array} \right)$

$I := 1$

$\ell_2 : \left( \begin{array}{l} pre(n_0, v_0, s_0, i_0) \\ s = \bigcup_{k=0}^{i-1} v(k) \wedge i = 1 \\ (n, v) = (n_0, v_0) \end{array} \right)$

*WHILE* $I < N$ *DO*

$\ell_3 : \left( \begin{array}{l} pre(n_0, v_0, s_0, i_0) \\ s = \bigcup_{k=0}^{i-1} v(k) \wedge i \in 1..n{-}1 \\ (n, v) = (n_0, v_0) \end{array} \right)$

$S := S \oplus V(I)$

$\ell_4 : \left( \begin{array}{l} pre(n_0, v_0, s_0, i_0) \\ s = \bigcup_{k=0}^{i} v(k) \wedge i \in 1..n{-}1 \\ (n, v) = (n_0, v_0) \end{array} \right)$

$I := I{+}1$

$\ell_5 : \left( \begin{array}{l} pre(n_0, v_0, s_0, i_0) \\ s = \bigcup_{k=0}^{i-1} v(k) \wedge i \in 2..n \\ (n, v) = (n_0, v_0) \end{array} \right)$

*OD*;

$\ell_6 : \left( \begin{array}{l} pre(n_0, v_0, s_0, i_0) \\ s = \bigcup_{k=0}^{n-1} v(k) \wedge i = n \\ (n, v) = (n_0, v_0) \end{array} \right)$

*La notation* $\bigcup_{k=0}^{n} v(k)$ *désigne la valeur maximale de la suite* $v(0) \ldots v(n)$*. On suppose que l'opérateur* $\oplus$ *est défini comme suit* $a \oplus b = max(a, b)$*.*

**Question 7.1** *Ecrire une solution contractuelle de cet algorithme.*

**Question 7.2** *Que faut-il faire pour vérifier que cet algorithme est bien annoté et qu'il est partiellement correct en utilisant TLA*$^+$*? Expliquer simplement les éléments à mettre en œuvre et les propriétés de sûreté à vérifier.*

**Question 7.3** *Ecrire un module TLA*$^+$ *permettant de vérifier l'algorithme annoté à la fois pour la correction partielle et l'absence d'erreurs à l'exécution.*

**Exercice 8** *gex8.c*

*On considère le petit programme se trouvant à droite de cette colonne. Nous allons poser quelques questions visant à compléter les parties marquées en gras et visant à définir la relation de calcul.*
*On notera* $pre(n_0, x_0, b_0)$ *l'expression suivante* $n_0, x_0, b_0 \in \mathbb{Z}$ *et* $in(n, b, n_0, x_0, b_0)$ *l'expression* $n = n_0 \wedge b = b_0 \wedge pre(n_0, x_0, b_0)$.

**Question 8.1** *Ecrire un algorithme avec le contrat et vÃ©rifier le .*

***VARIABLES*** $N, X, B$
***REQUIRES*** $n_0, x_0, b_0 \in \mathbb{Z}$
***ENSURES*** $\left(\begin{array}{l} n_0 < b_0 \Rightarrow x_f = (n_0+b_0)^2 \\ n_0 \geq b_0 \Rightarrow x_f = b_0 \\ n_f = n_0 \\ b_f = b_0 \end{array}\right.$
***BEGIN***
$\ell_0 : n = n_0 \wedge b = b_0 \wedge x = x_0 \wedge pre(n_0, x_0, b_0)$
$X := N;$
$\ell_1 : x = n \wedge in(n, b, n_0, x_0, b_0)$
***IF*** $X < B$ ***THEN***
$\ell_2$ :
$X := X \cdot X + 2 \cdot B \cdot X + B \cdot B;$
$\ell_3$ :
***ELSE***
$\ell_4$ :
$X := B;$
$\ell_5$ :
***FI***
$\ell_6$ :
***END***

**Exercice 9** *Soit le petit programme suivant :*

Listing 4 – contrat91

```
#include <stdio.h>
#include <math.h>

int f1(int x)
{ if (x > 100)
    { return(x-10);
    }
  else
    { return(f1(f1(x+11)));
    }
}

/*@ requires  INT_MIN <= x-10;
    requires x-10 <= INT_MAX;
    assigns  \nothing;
  ensures 100 < x ==> \result  == x -10;
  ensures x <= 100 ==> \result == 91;
*/


int f2(int x)
{ if (x > 100)
    { return(x-10);
    }
  else
    { return(91);
    }
}
```

```
  int val1,val2,val3,num;
   printf("Enter␣a␣number:␣");
   scanf("%d", &num);
   // Computes the square root of num and stores in root.
   val1 = f1(num);
     val2 = f2(num);
     val3 = mc91(num);
     printf("Et␣le␣rÃ©sultat␣␣f1(%d)=%d␣et␣la␣vÃ©rification:␣%d␣et␣.....%d\n", num,
   return 0;
}
```

*On veut montrer que les deux fonctions f1 et f2 sont Ã©quivalentes avec frama-c en montrant qu'elles vÃ©rifient le mÃªme contrat;*