

Cours MOdélisation, Vérification et EXpérimentations
Exercices (avec les corrections)
Utilisation d'un environnement de vérification Frama-c (I)
par Dominique Méry
5 mars 2025

TD5

Annotations en Frama-c

Exercice 1 *framac/ex1.c*

Vérifier l'annotation suivante :

```
l1: x== 10 && y == z+x && z==2*x;  
    y= z+x;  
l2: x== 10 && y == x+2*10;  
    x = x+1;  
l3: x-1== 10 && y == x-1+2*10;
```

◇ Solution de l'exercice 1

Listing 1 – annotation 1 (wp2.c)

```
int q1() {  
    int x=10,y=30,z=20;  
    //@ assert x== 10 && y == z+x && z==2*x;  
    y= z+x;  
    //@ assert x== 10 && y == x+2*10;  
    x = x+1;  
    //@ assert x-1== 10 && y == x-1+2*10;  
    return(0);  
}
```

Listing 2 – annotation 1 avec wp (wp2bis.c)

```
int q1() {  
    //@ assert 10== 10 && 30 == 20+10 && 20==2*10;  
    //@ assert 10== 10 && 20+10 == 10+2*10;  
    //@ assert 10+1-1== 10 && 20+10 == 10+1-1+2*10;  
    int x=10,y=30,z=20;  
    //@ assert x== 10 && y == z+x && z==2*x;  
    //@ assert x== 10 && z+x == x+2*10;  
    //@ assert x+1-1== 10 && z+x == x+1-1+2*10;  
    y= z+x;  
    //@ assert x== 10 && y == x+2*10;  
    //@ assert x+1-1== 10 && y == x+1-1+2*10;  
    x = x+1;  
    //@ assert x-1== 10 && y == x-1+2*10;  
    return(0);  
}
```

Fin 1

Exercice 2 *framac/ex2.c*

On suppose que *val* est une valeur entière. Vérifier l'annotation suivante :

```
int q1() {  
    int c = val ;  
l1: x == 2;  
    int x;  
l2: c == 2;  
    x = 3 * c ;  
l3: x == 6;  
    return(0);  
}
```

◇ **Solution de l'exercice 2**

Listing 3 – annotation 2 (wp3.c)

```
int q1() {  
    int c = 2 ;  
    /*@ assert c == 2; */  
    int x;  
    /*@ assert c == 2; */  
    x = 3 * c ;  
    /*@ assert x == 6; */  
    return(0);  
}
```

Fin 2

Exercice 3 Vérifier l'annotation suivante :

```
int main()  
{  
    int a = 42; int b = 37;  
    int c = a+b;  
l1: b == 37 ;  
    a -= c;  
    b += a;  
l2: b == 0 && c == 79;  
    return(0);  
}
```

◇ **Solution de l'exercice 3**

Listing 4 – annotation 2 (wp4.c)

```
int main()  
{  
    int a = 42; int b = 37;  
    int c = a+b; // i:1  
    //@assert b == 37 ;  
    a -= c; // i:2  
    b += a; // i:3  
    //@assert b == 0 && c == 79;  
    return(0);  
}
```

Fin 3

Exercice 4 Vérifier l'annotation suivante :

```

int main()
{
    int z;
    int a = 4;
l1:  a == 4 ;
    int b = 3;
l2:  b == 3 && a == 4;
    int c = a+b;
l3:  b == 3 && c == 7 && a == 4 ; */
    a += c;
    b += a;
l4:  a == 11 && b == 14 && c == 7 ;
l5:  a +b == 25 ;
    z = a*b;
l6:  a == 11 && b == 14 && c == 7 && z == 154;
    return(0);
}

```

◊ **Solution de l'exercice 4**

Listing 5 – annotation 4 (wp5.c)

```

int main()
{
    int z; // instruction 8
    int a = 4; // instruction 7
    //@assert a == 4 ;
    int b = 3; // instyruction 6
    //@assert b == 3 && a == 4;
    int c = a+b; // instruction 4
    /*@ assert b == 3 && c == 7 && a == 4 ; */
    a += c; // instruction 3
    b += a; // instruction 2
    //@ assert a == 11 && b == 14 && c == 7 ;
    //@ assert a +b == 25 ;
    z = a*b; // instruction 1
    //@assert a == 11 && b == 14 && c == 7 && z == 154;
    return(0);
}

```

Fin 4

Exercice 5 Vérifier l'annotation suivante :

```

int main()
{
    int a = 4;
    int b = 3;
    int c = a+b;
    a += c;
    b += a;
l:  a == 11 && b == 14 && c == 7 ;
    return(0);
}

```

◊ **Solution de l'exercice 5**

Listing 6 – annotation 5 (wp6.c)

```
int main()
{
    int a = 4;
    int b = 3;
    int c = a+b; // i:1
    a += c; // i:2
    b += a; // i:3
    //@assert a == 11 && b == 14 && c == 7 ;
    return(0);
}
```

Fin 5

Contrats en Frama-c

Exercice 6 Vérifier l'annotation suivante :

```
/*@ requires x0 >= 0;
    assigns \nothing;
    ensures \result == x0+1;
    @*/

int exemple(int x0) {
    int x=x0;
    //@ assert ...;
    x = x + 2;
    //@ assert x== ...;
    return x;
}
```

◇— Solution de l'exercice 6

Listing 7 – contrat (wp10.c)

```
/*@ requires x0 >= 0;
    assigns \nothing;
    ensures twp10 == x0+2;
    @*/

int exemple(int x0) {
    int x=x0;
    //@ assert x == x0;
    x = x + 2;
    //@ assert x== x0+2;
    return x;
}
```

Listing 8 – contrat avec wp (wp10bis.c)

```
/*@ requires x0 >= 0;
    assigns \nothing;
    ensures \result == x0+2;
    @*/

int exemple(int x0) {
    //@ assert x0 == x0;
    //@ assert x0 + 2== x0+2;
```

```
    int x=x0;
    //@ assert x == x0;
    //@ assert x0 + 2== x0+2;
    x = x + 2;
    //@ assert x== x0+2;
    return x;
}
```

Listing 9 – contrat avec wp (wp10.c)

```
/*@ requires x0 >= 0;
    assigns \nothing;
    ensures \result == x0+2;
    @*/

int exemple(int x0) {
    //@ assert x0+2 == x0+2;
    int x=x0;
    //@ assert x+2 == x0+2;
    x = x + 2;
    //@ assert x == x0+2;
    return x;
}
```

Fin 6

Exercice 7 Vérifier l'annotation suivante :

```
/*@
    requires x < 3 && x > 8;
    ensures \false;
    */
void fonc(int x){

}
```

◊ **Solution de l'exercice 7**

Listing 10 – annotation (wp9.c)

```
/*@
    requires x < 3 && x > 8;
    ensures \false;
    */
void fonc(int x){

}
```

Listing 11 – annotation (wp9bis.c)

```
/*@
    requires x < 3 && x > 8;
    ensures \true;
    */
void fonc(int x){

}
```

TD8-IL

Exercice 8 Analyser et compléter l'annotation suivante pour qu'elle soit valide :

```
int annotation(int a,int b)
{
    int x,y,z;
    x = a;
    l1: x == a; */
    y = b;
    l2: x == a && y == b; */
    z = a+b-2;
    l3: x == a && y == b && z==4; */
    return(z);
}
```

◇— Solution de l'exercice 8

Listing 12 – annotation wp11bis.c

```
/*@ requires a >= 0 && b>= 0 && a +b == 6;
   @ assigns \nothing;
   @ ensures \result == 4;
   @*/
int annotation(int a,int b)
{
    /*@ a == a; */
    /*@ a == a && b == b; */
    /*@ a == a && b == b && a+b-2wp1.c
       ==4; */
    int x,y,z;

    x = a;
    /*@ x == a; */
    /*@ x == a && b == b; */
    /*@ x == a && b == b && a+b-2==4; */
    y = b;
    /*@ x == a && y == b; */
    /*@ x == a && y == b && a+b-2==4; */
    z = a+b-2;
    /*@ x == a && y == b && z==4; */
    return(z);
}
```

Listing 13 – annotation (wp11.c)

```
/*@ requires a+b-2==4;
   assigns \nothing;
   ensures \result == 4;
   @*/

int annotation(int a,int b)
{

    int x,y,z;
    /*@ assert a == a;
```

```
//@ assert    a == a && b == b;
//@ assert a == a && b == b && a+b-2==4;
x = a;
//@ assert    x == a;
//@ assert    x == a && b == b;
//@ assert x == a && b == b && a+b-2==4;
y = b;
//@ assert    x == a && y == b;
//@ assert x == a && y == b && a+b-2==4;
z = a+b-2;
//@ assert x == a && y == b && z==4;
return(z);
}
```

Fin 8

Exercice 9 Définir une fonction *abs* avec son contrat.

```
int abs ( int x ) {
    if ( x >=0 ) return x ;
    return -x ; }
```

◊— **Solution de l'exercice 9**

Listing 14 – abs.c (wp1.c)

```
// returns the absolute value of x
#include <limits.h>
/*@
    requires x > INT_MIN;
    ensures (x >= 0 ==> \result == x);
    ensures (x < 0 ==> \result == -x);
*/

int abs ( int x ) {
    if ( x >=0 ) return x ;
    return -x ; }
```

Fin 9

Exercice 10 Définir une fonction *max* avec son contrat.

```
int max ( int x, int y ) {
    if ( x >=y ) return x ;
    return y ; }
```

◊— **Solution de l'exercice 10**

Listing 15 – max (mlax.c)

```
// returns the maximum of x and y
#include <limits.h>
/*@ requires x <= INT_MAX && x >= INT_MIN;
    requires y <= INT_MAX && y >= INT_MIN;
    assigns \nothing;
    ensures \result >= x && \result >= y && (x == \result || y == \result) ; */
int max ( int x, int y ) {
    if ( x >=y ) return x ;
    return y ; }
```

Exercice 11 Soit l'algorithme annoté comme suit :

Variables : X, Y, Z
Requires : $x_0, y_0 \in \mathbb{N} \wedge z_0 \in \mathbb{Z}$
Ensures : $z_f = \max(x_0, y_0)$

$\ell_0 : \{x = x_0 \wedge y = y_0 \wedge z = z_0 \wedge x_0, y_0 \in \mathbb{N} \wedge z_0 \in \mathbb{Z}\}$
if $X < Y$ **then**
 $\ell_1 : \{x < y \wedge x = x_0 \wedge y = y_0 \wedge z = z_0 \wedge x_0, y_0 \in \mathbb{N} \wedge z_0 \in \mathbb{Z}\}$
 $Z := Y;$
 $\ell_2 : \{x < y \wedge x = x_0 \wedge y = y_0 \wedge x_0, y_0 \in \mathbb{N} \wedge z_0 \in \mathbb{Z} \wedge z = y_0\}$
else
 $\ell_3 : \{x \geq y \wedge x = x_0 \wedge y = y_0 \wedge z = z_0 \wedge x_0, y_0 \in \mathbb{N} \wedge z_0 \in \mathbb{Z}\}$
 $Z := X;$
 $\ell_4 : \{x \geq y \wedge x = x_0 \wedge y = y_0 \wedge x_0, y_0 \in \mathbb{N} \wedge z_0 \in \mathbb{Z} \wedge z = x_0\}$
;
 $\ell_5 : \{z = \max(x_0, y_0) \wedge x = x_0 \wedge y = y_0 \wedge x_0, y_0 \in \mathbb{N} \wedge z_0 \in \mathbb{Z}\}$

Algorithme 1: maximum de deux nombres non annotée

Question 11.1 Ecrire un programme ACSL qui traduit ce contrat et qui le vérifie.

Question 11.2 Ecrire un programme ACSL qui traduit ce contrat et qui le vérifie mais qui enlève les assertions.

◊— **Solution de l'exercice 11**

Listing 16 – maximum.c (maximum.c)

```
/*@ axiomatic max{
  @ logic integer max(integer n, integer m);
  @ axiom max1: \forall integer n, m; n <= m
    ==> max(n, m) == m;
  @ axiom max2: \forall integer n, m; n > m
    ==> max(n, m) == n;
@} */

/*@ requires x0 >= 0 && y0 >= 0;
    assigns \nothing;
    ensures \result == max(x0, y0);
@*/

int maximum(int x0, int y0, int z0) {
  int x=x0;
  int y=y0;
  int z=z0;
  //@ assert max(x0, x0) == x0;
  //@ assert max(x0, y0) == max(y0, x0);
  //@ assert x== x0 && y == y0 && z==z0;
```



```

if (x < y) {
  //@ assert x < y && x== x0 && y == y0 && z==z0;
  //@ assert x < y && x== x0 && y == y0 && y==y;
  z = y;
  //@ assert x < y && x== x0 && y == y0 && z==y;
}
else
{
  //@ assert x >= y && x== x0 && y == y0 && z==z0;
  //@ assert x >= y && x== x0 && y == y0 && x==x;

  z = x;
  //@ assert x >= y && x== x0 && y == y0 && z==x;
};
//@ assert x== x0 && y == y0 && z==max(x,y);
return z;
}

```

Listing 17 – maximumlazy.c

```

/*@ axiomatic max{
  @ logic integer max(integer n, integer m);
  @ axiom max1: \forall integer n,m; n <= m
    ==> max(n,m) == m;
  @ axiom max2: \forall integer n,m; n > m
    ==> max(n,m) == n;
@} */

/*@ requires x0 >= 0 && y0 >=0;
   assigns \nothing;
   ensures \result == max(x0,y0);
   @*/

int maximum(int x0,int y0,int z0) {
  int x=x0;
  int y=y0;
  int z=z0;
  //@ assert (x<y && y == max(x0,y0)) || (x >= y && x== max(x0,y0));
  if (x < y) {
    //@ assert x<y && y == max(x0,y0);
    z = y;
    //@ assert z == max(x0,y0);
  }
  else
  {
    //@ assert x >= y && x== max(x0,y0);
    z = x;
    //@ assert z == max(x0,y0);
  };
  //@ assert z == max(x0,y0);
  return z;
}

```

Contrats avec invariants de boucle et ghosts en Frama-c

Exercice 12 *Ecrire un contrat pour la fonction factorielle*

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

◇— **Solution de l'exercice 12**

Listing 18 – factorial.c

```
/*@ axiomatic mathfact {
    @ logic integer mathfact(integer n);
    @ axiom mathfact_1: mathfact(1) == 1;
    @ axiom mathfact_rec: \forall integer n; n > 1
    ==> mathfact(n) == n * mathfact(n-1);
    @ } */

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

Fin 12

Exercice 13 *Ecrire un contrat pour la fonction calculant le reste de la division de a par b.*

```
int reste(int a, int b) {
    int r = a;
    int q = 0;
    while (r >= b) {
        r = r - b;
        q = q + 1;
    };
    return r;
}
```

◇— **Solution de l'exercice 13**

Listing 19 – remainder.c

```
#include <limits.h>

#include <limits.h>

/*@ requires a >= 0 && b > 0;
    requires a <= INT_MAX;
    requires b <= INT_MAX;
    assigns \nothing;
    ensures 0 <= \result;
    ensures \result < b;
    ensures \exists integer k; a == k * b + \result;
    ensures \result <= INT_MAX;
*/
int reste (int a, int b) {
    int r = a;
    int q = 0;
    /*@
        loop invariant
        ( a == q * b + r ) &&
        r >= 0 && r <= a;
        loop assigns r;
        loop assigns q;
    */
    while (r >= b) {
        r = r - b;
        q = q + 1;
    };
    return r;
}
```

Listing 20 – remainder.c

```
/*@ requires a >= 0 && b >= 0;
    assigns \nothing;
    ensures 0 <= \result;
    ensures \result < b;
    ensures \exists integer k; a == k * b + \result;
*/
int rem(int a, int b) {
    int r = a;
    /*@ ghost int q=0;
    */
    /*@
        loop invariant
        a == q * b + r &&
        r >= 0
    ;
        loop assigns r;
        loop assigns q;
    */
    while (r >= b) {
        r = r - b;
    /*@ ghost
        q = q+1;
    */
    }
}
```

```
    */  
};  
return r;  
}
```

Fin 13