

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

Exercice 2 *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 2

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 2

Exercice 3 *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 3**

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 3

Exercice 4 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 4**

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 4

Exercice 5 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 5** _____

Listing 5 – annotation 4 (wp5.c)

```

int main()
{
    int z; // instruction 8
    int a = 4; // instruction 7
    //@assert a == 4 ;
    int b = 3; // instyruccion 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 5

Exercice 6 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 6** _____

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 6

Contrats en Frama-c

Exercice 7 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 7

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 7

Exercice 8 Vérifier l'annotation suivante :

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

}
```

◊ **Solution de l'exercice 8**

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 9 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 9**

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 9

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

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

◇— **Solution de l'exercice 10**

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 10

Exercice 11 Définir une fonction *max* avec son contrat.

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

◇— **Solution de l'exercice 11**

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 12 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 12.1 Ecrire un programme ACSL qui traduit ce contrat et qui le vérifie.

Question 12.2 Ecrire un programme ACSL qui traduit ce contrat et qui le vérifie mais qui enlève les assertions.

◊— **Solution de l'exercice 12**

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 13 *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 13**

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 13

Exercice 14 *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 14**

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 14

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

Exercice 15 *Soit le petit programme suivant*

Listing 21 – td61.c

```

void ex(void) {
    int x=2,y=4,z,a=1;

    //@ assert x <= y;
    x = x*x;
    //@ assert x == a*y;
    y = 2*x;

    z = x + y;

    //@ assert z == x+y && x* y >= 8;
}

```

Analyser le correction des annotations avec Frama-c et trouver a pour que cela soit correctement analysé.

Exercice 16 *Soit le petit programme suivant*

Listing 22 – td62.c

```

void ex(void) {
    int x0,y0,z0;
    int x=x0,y=x0,z=x0*x0;

    //@ assert x == y && z == x*y;
    x = x*x;
    //@ assert x == y*y && z == x;
    y = x;
    z = x + y + 2*z;

    //@ assert z == (x0+x0)*(x0+x0);
}

```

Analyser la correction des annotations avec Frama-c.

Exercice 17 *Soit le petit programme suivant*

Listing 23 – td63.c

```

#include <limits.h>
// returns the maximum of x and y
/*@

```

```

    ensures \result >= x && \result >= y && (\result == x || \result == y);
*/
int max ( int x, int y ) {

    if ( x >= y )
    {
        //@ assert x >= y;
        return x ;
        //@ assert x >= y;
    }
    //@ assert x < y;
    return y ;
    //@ assert x < y;
}

```

Analyser la correction des annotations avec Frama-c.

Exercice 18 La définition structurale des transformateurs de prédicats est rappelée dans le tableau ci-dessous :

S	$wp(S)(P)$
$X := E(X, D)$	$P[e(x, d)/x]$
SKIP	P
$S_1; S_2$	$wp(S_1)(wp(S_2)(P))$
IF B S_1 ELSE S_2 FI	$(B \Rightarrow wp(S_1)(P)) \wedge (\neg B \Rightarrow wp(S_2)(P))$

- Axiome d'affectation : $\{P(e/x)\} X := E(X) \{P\}$.
- Axiome du saut : $\{P\} \text{skip} \{P\}$.
- Règle de composition : Si $\{P\} S_1 \{R\}$ et $\{R\} S_2 \{Q\}$, alors $\{P\} S_1; S_2 \{Q\}$.
- Si $\{P \wedge B\} S_1 \{Q\}$ et $\{P \wedge \neg B\} S_2 \{Q\}$, alors $\{P\} \text{if } B \text{ then } S_1 \text{ then } S_2 \text{ fi} \{Q\}$.
- Si $\{P \wedge B\} S \{P\}$, alors $\{P\} \text{while } B \text{ do } S \text{ od} \{P \wedge \neg B\}$.
- Règle de renforcement/affaiblissement : Si $P' \Rightarrow P$, $\{P\} S \{Q\}$, $Q \Rightarrow Q'$, alors $\{P'\} S \{Q'\}$.

Question 18.1 Simplifier les expressions suivantes :

1. $WP(X := X + Y + 7)(x + y = 6)$
2. $WP(X := X + Y)(x < y)$

Question 18.2 On rappelle que $\{P\} S \{Q\}$ est défini par l'implication $O \Rightarrow WP(S)(Q)$. Pour chaque point énuméré ci-dessous, monter que la propriété $\{P\} S \{Q\}$ est valide ou pas en utilisant la définition suivante :

$$\{P\} S \{Q\} = P \Rightarrow WP(S)(Q)$$

1. $\{x + y = 7\} X := Y + X \{2 \cdot x + y = 6\}$
2. $\{x < y\} \text{IF } x \neq y \text{ THEN } x := 5 \text{ ELSE } x := 8 \text{ FI} \{x \in \{5, 8\}\}$

Question 18.3 Utiliser frama-c pour vérifier les éléments suivants :

1. $\{x + y = 7\} X := Y + X \{2 \cdot x + y = 6\}$
2. $\{x < y\} \text{IF } x \neq y \text{ THEN } x := 5 \text{ ELSE } x := 8 \text{ FI} \{x \in \{5, 8\}\}$

Exercice 19 td65.c

Soit le petit programme suivant dans un fichier :

Listing 24 – td65.c

```

/*@
    assigns    \nothing;
*/

```

```
void swap1(int a, int b) {
    int x = a;
    int y = b;
    //@ assert x == a && y == b;
    // ==> ?
    //@ assert y == b && x == a;
    int tmp;
    //@ assert y == b && x == a;
    tmp = x;
    //@ assert y == b && tmp == a;
    x = y;
    //@ assert x == b && tmp == a;
    y = tmp;
    //@ assert x == b && y == a;
}
```

Question 19.1 Utiliser l'outil *frama-c-gui* avec la commande `$frama-c-gui ex1.c` et cliquer sur le lien *ex1.c* apparaissant sur la gauche. A partir du fichier source, une fenêtre est créée et vous découvrez le texte du fichier.

Question 19.2 Cliquer à droite sur le mot-clé *assert* et cliquer sur *Prove annotation by WP*. Les boutons deviennent vert.

Question 19.3

```
void swap2(int a, int b) {
    int x = a;
    int y = b;
    //@ assert x == a && y == b;
    int tmp;
    tmp = x;
    x = y;
    y = tmp;
    //@ assert x == a && y == a;
}
```

Répétez les mêmes suites d'opérations mais avec le programme suivant dans *ex2.c*.

Question 19.4 Ajoutez une précondition pour que les preuves soient possibles.

◇— **Solution de la question 19.4** _____

```
/*@ requires a==b;
*/
void swap2(int a, int b) {
    int x = a;
    int y = b;
    //@ assert x == a && y == b;
    int tmp;
    tmp = x;
    x = y;
    y = tmp;
    //@ assert x == a && y == a;
}
```

Fin 19.4

Question 19.5 Soit le nouvel algorithme avec un contrat qui établit ce que l'on attend de cet algorithme

```
/*@
requires \valid(a);
requires \valid(b);
ensures P: *a == \old(*b);
ensures Q: *b == \old(*a);
*/
void swap3(int
           *a, int *b) {
    int tmp;
    tmp = *a;
    *a = *b;
    *b = tmp;
}
```

Recommencer les opérations précédentes et observer ce qui a été utilisé comme outils de preuve.

Exercice 20 Etudier la correction de l'algorithme suivant en complétant l'invariant de boucle :

Listing 25 – td66.c

```
/*@
requires 0 <= n;
ensures \result == n * n;
*/
int f(int n) {
    int i = 0;
    /*@ assert i=0
    int s = 0;
    /*@ loop invariant ...;
    @ loop assigns ...; */
    while (i < n) {
        i++;
        s += 2 * i - 1;
    };
    return s;
}
```

Listing 26 – td66c.c

```
/*@
requires 0 <= n;
ensures \result == n * n;
*/
int f(int n) {
    int i = 0;
    //@ assert i==0;
    int s = 0;
    //@ assert 0*i == s && 0 <= n;

    /*@ loop invariant i * i == s && i <= n;
    @ loop assigns i, s;
    @ loop variant n-i; */
    while (i < n) {
        i++;
        s += 2 * i - 1;
    };
}
```

```
//@ assert i==n && s == n*n;
return s;
}
```

Exercice 21

On rappelle que l'annotation suivante du listing 27 est correcte, si les conditions suivantes sont vérifiées :

- $pre(v_0) \wedge v = v_0 \Rightarrow A(v_0, v)$
- $pre(v_0) \wedge B(v_0, v) \Rightarrow post(v_0, v)$
- $A(v_0, v) \Rightarrow wp(v = f(v))(B(v_0, v))$ où $wp(v = f(v))(B(v_0, v))$ est définie par $B(v_0, v)[f(v)/v]$.

Dans le cas de frama-c, la valeur initiale d'une variable v est notée $\backslash at(v, Pre)$ et aussi $\backslash old(v)$. Nous utiliserons la notation v_0 dans cet exercice.

Listing 27 – contrat

```
requires pre(v)
ensures post(\old(v), v)
type1 truc(type2 v)
/*@ assert A(v0, v); */
v = f(v);
/*@ assert B(v0, v); */
return val;
```

Soient les annotations suivantes. Les variables sont supposées de type integer.

Question 21.1

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

Montrer que l'annotation est correcte ou incorrecte en utilisant Frama-c

Listing 28 – td71.c

```
/*@
requires x0>=0 && y0 >= 0 && z0 >= 0 && z0 == 25 && y0==x0+1 && x0*x0 + y0*y0
ensures \result == 100;
*/
int f(int x0, int y0, int z0) {
    int x = x0;
    int y = y0;
    int z = z0;
    /*@ assert x*x + y*y == z && z == 25 ;*/
    x = x +3;
    y = y +4;
    z = z + 75;
    /*@ assert x*x + y*y == z ; */
    return z;
}
```

Question 21.2 Soient trois constantes n, m, p

$$\begin{aligned} \ell_1 : x = 3^n \wedge y = 3^p \wedge z = 3^m; \\ T := 8 \cdot X \cdot Y \cdot Z; \\ \ell_2 : t = (y+z)^3 \wedge y = x; \end{aligned}$$

Montrer que l'annotation est correcte ou incorrecte en utilisant Frama-c. On prendra soin de discuter sur les valeurs de m, n, p et notamment de donner une condition sur ces valeurs pour que cel soit correcte.

Listing 29 – td68.c

```

Exercice 22 // #include <limits.h>
/*@ axiomatic auxmath {
  @ axiom rule1: \forall int n; n > 0 ==> n*n == (n-1)*(n-1)+2*n+1;
  @ } */

/*@ requires 0 <= x;
    ensures \result == x*x;
*/
int power2(int x)
{int r, k, cv, cw, or, ok, ocv, ocw;
  r=0; k=0; cv=0; cw=0; or=0; ok=k; ocv=cv; ocw=cw;
  /*@ loop invariant cv == k*k;
      @ loop invariant k <= x;
      @ loop invariant cw == 2*k;
      @ loop invariant 4*cv == cw*cw;
      @ loop assigns k, cv, cw, or, ok, ocv, ocw; */
  while (k<x)
  {
    ok=k; ocv=cv; ocw=cw;
    k=ok+1;
    cv=ocv+ocw+1;
    cw=ocw+2;

  }
  r=cv;
  return(r);
}

/*@ requires 0 <= x;
    ensures \result == x*x;
*/
int p(int x)
{
  int r;
  if (x==0)
  {
    r=0;

  }
  else
  {
    r= p(x-1)+2*x+1;

  }
  return(r);
}

/*@ requires 0 <= n;
```

```

    ensures \result == 1;
*/

int check(int n){
    int r1,r2,r;
    r1 = power2(n);
    r2 = p(n);
    if (r1 != r2)
    { r = 0;
    }
    else
    { r = 1;
    };
    return r;
}

```

Soit le fichier `qpowers.c` qui est partiellement complété et qui permet de calculer le carré d'un nombre naturel. L'exercice vise à compléter les points d'interrogation puis de simplifier le résultat et de montrer l'équivalence de deux fonctions. Le fichier `mainpowers.c` peut être compilé pour que vous puissiez faire des expérimentations sur les valeurs calculées.

Question 22.1 Compléter le fichier `qpowers.c` et produire le fichier `power2.c` qui est vérifié avec `frama-c`.

Question 22.2 a Simplifier la fonction itérative en supprimant les variables commençant par la lettre `o`. Puis vérifier les fonctions obtenues avec `frama-c`.

Question 22.3 En fait, vous avez montré que les deux fonctions étaient équivalentes. Expliquez pourquoi en quelques lignes.

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

Exercice 23 Soit le contrat suivant :

```

variables X, Y, Z
requires x0 >= 0 ∧ y0 >= 0 ∧ z0 >= 0 Roots1st ∧ z0 = 25 ∧ y0 = x0+1
ensures zf = 100;
    begin
    0 : x2+y2 = z ∧ z = 25;
    (X, Y, Z) := (X+3, Y+4, Z+75);
    1 : x2+y2 = z;
    end

```

Question 23.1 Traduire ce contrat avec le langage `PlusCal` et proposer une validation pour que ce contrat soit valide.

Listing 30 – `td71.c`

```

/*@
requires x0>=0 && y0 >= 0 && z0 >= 0 && z0 == 25 && y0==x0+1 && x0*x0 + y0*y0
ensures \result == 100;
*/
int f(int x0,int y0,int z0) {

```

```

int x = x0;
int y = y0;
int z = z0;
  /*@ assert  x*x + y*y == z &&    z == 25 ;*/
  x = x +3;
y = y +4;
z = z + 75;
/*@ assert  x*x + y*y == z ; */
  return z;
}

```

Listing 31 – td71bis.c

```

  /*@
    requires x0>=0 && y0 >= 0 && z0 >= 0 && z0 == 25 && y0==x0+1 &&
    x0*x0 + y0*y0 == z0 &&    z0 == 25 && (x0+3)*(x0+3) + (y0+4)*(y0+4) == z0+75 ;
    ensures \result == 100;
  */
int f(int x0,int y0,int z0) {
  int x = x0;
  int y = y0;
  int z = z0;
  /*@ assert  x*x + y*y == z &&    z == 25 ;*/
  /*@ assert  (x+3)*(x+3) + (y+4)*(y+4) == z+75 ; */
  x = x +3;
  /*@ assert  x*x + (y+4)*(y+4) == z+75 ; */
  y = y +4;
  /*@ assert  x*x + y*y == z+75 ; */
  z = z + 75;
  /*@ assert  x*x + y*y == z ; */
  return z;
}

```

Question 23.2 Traduire ce contrat en ACSL et vérifier qu'il est valide ou non. S'il est non valide, proposer une correction de la pré-condition et /ou de la postcondition.

Listing 32 – td71.tla

```

----- MODULE  td71 -----
EXTENDS TLC, Integers, Naturals

CONSTANTS x0,y0,z0

ASSUME x0 \geq 0 /\ y0 \geq 0 /\ z0 \geq 0 /\ z0 = 25 /\ y0 = x0 +1
(*)
d71--fair algorithm q2 {
  variables x=x0,y=y0, z=z0;
  {
    l1: assert x*x + y*y = z /\ z=25;
    x:= x+3;y:=y+4;z:=z+75;
    l2: assert  x*x + y*y = z;
  }
}
*)
\* BEGIN TRANSLATION (chksum(pcal) = "82854ce8" /\ chksum(tla) = "4e9145d3")
VARIABLES x, y, z, pc

```

```

vars == << x, y, z, pc >>

Init == (* Global variables *)
        /\ x = x0
        /\ y = y0
        /\ z = z0
        /\ pc = "l1"

l1 == /\ pc = "l1"
      /\ Assert(x*x + y*y = z /\ z=25,
                "Failure_of_assertion_at_line_11,column_5.")
      /\ x' = x+3
      /\ y' = y+4
      /\ z' = z+75
      /\ pc' = "l2"

l2 == /\ pc = "l2"
      /\ Assert(x*x + y*y = z, "Failure_of_assertion_at_line_13,column_6.")
      /\ pc' = "Done"
      /\ UNCHANGED<<x, y, z>>

(* Allow infinite stuttering to prevent deadlock on termination. *)
Terminating == pc = "Done" /\ UNCHANGED vars

Next == l1 /\ l2
       /\ Terminating

Spec == /\ Init /\ [] [Next] vars
       /\ WF_vars(Next)

Termination == <>(pc = "Done")

\*_END_TRANSLATION

check == pc = "Done" => z = 100
=====

```

Exercice 24 Définir une fonction `maxpointer` (`gex1.c`) calculant la valeur du maximum du contenu de deux adresses avec son contrat.

```

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

```

◊ Solution de l'exercice 24

Listing 33 – `gex1.c`

```

// frama-c -wp -wp-rte -report -wp-print gex1.c

/*@ requires \valid(p) && \valid(q);

```

```

    ensures \result >= *p && \result >= *q &&
    \result == *p || \result == *q;
*/
int max_ptr ( int *p, int *q ) {

    //@ assert *p < *q ==> *q >= *p && *q >= *q &&      *q == *p || *q == *q ;
    //@ assert *p >= *q ==> *p >= *p && *p >= *q &&      *p == *p || *p == *q;
    if ( *p >= *q ) {
        //@ assert *p >= *p && *p >= *q &&      *p == *p || *p == *q;
        return *p; // \result = *p;
        //@ assert \result >= *p && \result >= *q &&      \result == *p || \result == *q;
    };

    //@ assert *q >= *p && *q >= *q &&      *q == *p || *q == *q ;
    return *q ; // \result = *q;
    //@ assert \result >= *p && \result >= *q &&      \result == *p || \result == *q;
}

```

Fin 24

Exercice 25 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 25**

Fin 25

Exercice 26 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 26**

Listing 34 – gex4-1.c

```

/*@ 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;
}

```

Listing 35 – gex4-1bis.c

```

/*@ 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;

  &&
  ( a == i * b + r) &&
  r >= 0 && r <= a
  ;
  loop assigns r,i;
  */
  while (r >= b) {
    r = r - b;
    ++i;
  };
  return r;
}

```

Listing 36 – gex4-2.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;  
}
```

Listing 37 – gex4-3.c

```
/*@ 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;  
        loop variant n-i;  
    */  
    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;
        loop variant n-i;
    */
    for (int i = 1; i < n; i++)
        if (t[i] > r) r = t[i];
    return r;
}

```

Fin 26

◊ **Solution de l'exercice 26**

Fin 26

Exercice 27 Question 27.1 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 27.1**

```

/*@ 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;
        loop variant r-b;
    */
    while (r >= b) {
        r = r - b;
    };
    return r;
}

```



```

}

/*@ 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;
    /*@ ghost    int q=0;
       */
    /*@
       loop invariant
       a == q * b + r &&
       r >= 0 && r <= a
       ;
       loop assigns r;
       loop assigns q;
       loop variant r-b;
    */
    while (r >= b) {
        r = r - b;
    }
    /*@ ghost
       q = q+1;
       */
    };
    return r;
}

```

Fin 27.1

Question 27.2 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 27.2**

Listing 38 – factoriel.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 27.2

Question 27.3 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 27.3**

Listing 39 – gex4-3.c

```

/*@ 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;
        loop variant n-i;
    */
    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;
    loop variant n-i;
*/
for (int i = 1; i < n; i++)
    if (t[i] > r) r = t[i];
return r;
}

```

Fin 27.3

Reprise

Exercice 28 Pour chaque question, montrer que l'annotation est correcte ou incorrecte selon les conditions de vérifications énoncées comme suit

$\forall x, y, x', y'. P_\ell(x, y) \wedge \text{cond}_{\ell, \ell'}(x, y) \wedge (x', y') = f_{\ell, \ell'}(x, y) \Rightarrow P_{\ell'}(x', y')$

Pour cela, on utilisera l'environnement *Frama-c*.

Question 28.1

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

◇— **Solution de la question 28.1**

Listing 40 – hoare1.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;
    return (0);
}

```

Fin 28.1

Question 28.2

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

Question 28.3

$$\begin{aligned} \ell_1 : x = 11 \wedge y = 13 \\ z := x; x := y; y := z; \\ \ell_2 : x = 26/2 \wedge y = 33/3 \end{aligned}$$

Exercice 29 Evaluer la validité de chaque annotation dans les questions suivantes.

Question 29.1

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

Question 29.2

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

Question 29.3

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

Soit l'annotation suivante :

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

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

Question 29.4 Ecrire la condition de vérification correspondant à cette annotation en supposant que X et Y sont deux variables entières.

Question 29.5 Etudier la validité de cette condition de vérification selon la valeur de m .

Exercice 30 *gex7.c*

VARIABLES N, V, S, I

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

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

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

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

$$S := V(0)$$

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

$$I := 1$$

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

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 $v(0) \dots v(n)$. On suppose que l'opérateur \oplus est défini comme suit $a \oplus b = \max(a, b)$.

Question 30.1 Ecrire une solution contractuelle de cet algorithme.

Question 30.2 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 30.3 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 31 *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 31.1 Ecrire un algorithme avec le contrat et vérifier 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 32 Soit le petit programme suivant :

Listing 41 – f91

```

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

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

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

int mc91tail(int n, int c)
{ if (c != 0) {
  if (n > 100) {
    return mc91tail(n-10, c-1);
  }
  else
  {

```

```

        return mc91tail(n+11,c+1);
    }
}
else
    { return n;}
}

int mc91(int n)
{
    return mc91tail(n,1);
}

int main()
{
    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_résultat_de_f1(%d)=%d_et_la_vérification:_%d_et_....%d\n", num,
    return 0;
}

```

On veut montrer que les deux fonctions *f1* et *f2* sont équivalentes avec *frama-c* en montrant qu'elles vérifient le même contrat;

Exercice 33 Soit le petit programme suivant :

Listing 42 – qpower2.c

```

#include <limits.h>
/*@ axiomatic auxmath {
    @ axiom rule1: \forall int n; n > 0 ==> n*n == (n-1)*(n-1)+2*n+1;
    @ } */

/*@ requires 0 <= x;
    requires x <= INT_MAX;
    requires x*x <= INT_MAX;
    assigns \nothing;
    ensures \result == x*x;
*/
int power2(int x)
{int r,k,cv,cw,or,ok,ocv,ocw;
  r=0;k=0;cv=0;cw=0;or=0;ok=k;ocv=cv;ocw=cw;
  /*@ loop invariant cv == k*k;
    @ loop invariant k <= x;
    @ loop invariant cw == 2*k;
    @ loop invariant 4*cv == cw*cw;
    @ loop assigns k,cv,cw,or,ok,ocv,ocw;
    @ loop variant x-k;
  */
  while (k<x)
  {
    ok=k;ocv=cv;ocw=cw;
    k=ok+1;
  }
}

```



```

        cv=ocv+ocw+1;
        cw=ocw+2;

    }
    r=cv;
    return(r);
}

/*@ requires 0 <= x;
    decreases x;
    assigns \nothing;

    ensures \result == x*x
;
*/
int p(int x)
{
    int r;
    if (x==0)
    {
        r=0;

    }
    else
    {
        r = p(x-1)+2*x+1;

    }
    return(r);
}

```

```

/*@ requires 0 <= n;
    assigns \nothing;
    ensures \result == 1;
*/

int check(int n){
    int r1,r2,r;
    r1 = power2(n);
    r2 = p(n);
    if (r1 != r2)
    { r = 0;
    }
    else
    { r = 1;
    };
    return r;
}

```

On veut montrer que les deux fonctions *p* et *power2* sont équivalentes avec *frama-c* en montrant qu'elles vérifient le même contrat ;

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

Exercice 34 *Utiliser frama-c pour vérifier ou non les annotations suivantes :*

Question 34.1

$$\begin{array}{l} \ell_1 : x = 10 \wedge y = z+x \wedge z = 2 \cdot x \\ y := z+x \\ \ell_2 : x = 10 \wedge y = x+2 \cdot 10 \end{array}$$

Question 34.2

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

Question 34.3

$$\begin{array}{l} \ell_1 : x = 11 \wedge y = 13 \\ z := x; x := y; y := z; \\ \ell_2 : x = 26/2 \wedge y = 33/3 \end{array}$$

Question 34.4

$$\begin{array}{l} \ell_1 : x = 3 \wedge y = z+x \wedge z = 2 \cdot x \\ y := z+x \\ \ell_2 : x = 3 \wedge y = x+6 \end{array}$$

Question 34.5

$$\begin{array}{l} \ell_1 : x = 2^4 \wedge y = 2^{345} \wedge x \cdot y = 2^{350} \\ x := y+x+2^x \\ \ell_2 : x = 2^{56} \wedge y = 2^{345} \end{array}$$

Question 34.6

$$\begin{array}{l} \ell_1 : x = 1 \wedge y = 12 \\ x := 2 \cdot y+x \\ \ell_2 : x = 1 \wedge y = 25 \end{array}$$

Exercice 35 *Traduire ce contrat dans le langage ACSL et vérifier le contrat.*

```

variables  $x$ 
requires
   $x_0 \in \mathbb{N}$ 
ensures
   $x_f \in \mathbb{N}$ 
begin
 $\ell_0 : \{x = x_0 \wedge x_0 \in \mathbb{N}\}$ 
While ( $0 < x$ )
 $\ell_1 : \{0 < x \leq x_0 \wedge x_0 \in \mathbb{N}\}$ 
 $x := x - 1$ ;
 $\ell_2 : \{0 \leq x \leq x_0 \wedge x_0 \in \mathbb{N}\}$ 
od;
 $\ell_4 : \{x = 0\}$ 
end

```

Exercice 36 Utiliser frama-c pour vérifier le contrat suivant :

```

Variables : F,N,M,I

Requires :  $\left( \begin{array}{l} n_0 \in \mathbb{N} \wedge \\ n_0 \neq 0 \wedge \\ f_0 \in 0 .. n_0 - 1 \rightarrow \mathbb{N} \end{array} \right)$ 

Ensures :  $\left( \begin{array}{l} m_f \in \mathbb{N} \wedge \\ m_f \in \text{ran}(f_0) \wedge \\ (\forall j. j \in 0 .. n_0 - 1 \Rightarrow f_0(j) \leq m_f) \end{array} \right)$ 

 $M := F(0)$ ;
 $I := 1$ ;
while  $I < N$  do
  if  $F(i) > M$  then
     $M := F(I)$ ;
  ;
   $I++$ ;
;
b

```

Algorithme 2: Algorithme du maximum d'une liste non annotée

Exercice 37

Utiliser frama-c pour vérifier le contrat suivant :

Soit l'algorithme annoté suivant se trouvant à la page suivante et les pré et postconditions définies pour cet algorithme comme suit : On suppose que x_1 et x_2 sont des constantes.

Exercice 38 Soit la fonction suivante utilisée dans un programme

Listing 43 – mainpower.c

```

#include <stdio.h>
#include <limits.h>
int power(int x)
{
  int r, cz, cv, cu, cw, ct, k;
  cz=0;cv=0;cw=1;ct=3;cu=0;k=0;
  while (k<x)
  {

```

Variables : X1,X2,Y1,Y2,Y3,Z

Requires : $x1_0 \in \mathbb{N} \wedge x2_0 \in \mathbb{N} \wedge x1_0 \neq 0$

Ensures : $z_f = x1_0^{x2_0}$

$\ell_0 : \{x1_0 \in \mathbb{N} \wedge x2_0 \in \mathbb{N} \wedge x1_0 \neq 0 \wedge y1_0, y2_0, y3_0, z_0 \in \mathbb{Z} \wedge (x1, x2, y1, y2, y3, z) = (x1_0, x2_0, y1_0, y2_0, y3_0, z_0)\}$

$(y1, y2, y3) := (x1, x2, 1);$

$\ell_1 : \{x1_0 \in \mathbb{N} \wedge x2_0 \in \mathbb{N} \wedge x1_0 \neq 0 \wedge y1_0, y2_0, y3_0, z_0 \in \mathbb{Z} \wedge (x1, x2, z) = (x1_0, x2_0, z_0) \wedge y3 \cdot y1^{y2} = x1^{x2}\}$

while $y2 \neq 0$ **do**

$\ell_2 : \{x1_0 \in \mathbb{N} \wedge x2_0 \in \mathbb{N} \wedge x1_0 \neq 0 \wedge y1_0, y2_0, y3_0, z_0 \in \mathbb{Z} \wedge (x1, x2, z) = (x1_0, x2_0, z_0) \wedge y3 \cdot y1^{y2} = x1^{x2} \wedge 0 < y2 \leq x2\}$

if $impair(y2)$ **then**

$\ell_3 : \{x1_0 \in \mathbb{N} \wedge x2_0 \in \mathbb{N} \wedge x1_0 \neq 0 \wedge y1_0, y2_0, y3_0, z_0 \in \mathbb{Z} \wedge (x1, x2, z) = (x1_0, x2_0, z_0) \wedge y3 \cdot y1^{y2} = x1^{x2} \wedge 0 < y2 \leq x2 \wedge impair(y2)\}$

$y2 := y2 - 1;$

$\ell_4 : \{x1_0 \in \mathbb{N} \wedge x2_0 \in \mathbb{N} \wedge x1_0 \neq 0 \wedge y1_0, y2_0, y3_0, z_0 \in \mathbb{Z} \wedge (x1, x2, z) = (x1_0, x2_0, z_0) \wedge y3 \cdot y1 \cdot y1^{y2} = x1^{x2} \wedge 0 \leq y2 \leq x2 \wedge pair(y2)\}$

$y3 := y3 \cdot y1;$

$\ell_5 : \{x1_0 \in \mathbb{N} \wedge x2_0 \in \mathbb{N} \wedge x1_0 \neq 0 \wedge y1_0, y2_0, y3_0, z_0 \in \mathbb{Z} \wedge (x1, x2, z) = (x1_0, x2_0, z_0) \wedge y3 \cdot y1^{y2} = x1^{x2} \wedge 0 \leq y2 \leq x2 \wedge pair(y2)\}$

;

$\ell_6 : \{x1_0 \in \mathbb{N} \wedge x2_0 \in \mathbb{N} \wedge x1_0 \neq 0 \wedge y1_0, y2_0, y3_0, z_0 \in \mathbb{Z} \wedge (x1, x2, z) = (x1_0, x2_0, z_0) \wedge y3 \cdot y1^{y2} = x1^{x2} \wedge 0 \leq y2 \leq x2 \wedge pair(y2)\}$

$y1 := y1 \cdot y1;$

$\ell_7 : \{x1_0 \in \mathbb{N} \wedge x2_0 \in \mathbb{N} \wedge x1_0 \neq 0 \wedge y1_0, y2_0, y3_0, z_0 \in \mathbb{Z} \wedge (x1, x2, z) = (x1_0, x2_0, z_0) \wedge y3 \cdot y1^{y2 \div 2} = x1^{x2} \wedge 0 \leq y2 \leq x2 \wedge pair(y2)\}$

$y2 := y2 \div 2;$

$\ell_8 : \{x1_0 \in \mathbb{N} \wedge x2_0 \in \mathbb{N} \wedge x1_0 \neq 0 \wedge y1_0, y2_0, y3_0, z_0 \in \mathbb{Z} \wedge (x1, x2, z) = (x1_0, x2_0, z_0) \wedge y3 \cdot y1^{y2} = x1^{x2} \wedge 0 \leq y2 \leq x2\}$

;

$\ell_9 : \{x1_0 \in \mathbb{N} \wedge x2_0 \in \mathbb{N} \wedge x1_0 \neq 0 \wedge y1_0, y2_0, y3_0, z_0 \in \mathbb{Z} \wedge (x1, x2, z) = (x1_0, x2_0, z_0) \wedge y3 \cdot y1^{y2} = x1^{x2} \wedge y2 = 0\}$

$z := y3;$

$\ell_{10} : \{x1_0 \in \mathbb{N} \wedge x2_0 \in \mathbb{N} \wedge x1_0 \neq 0 \wedge y1_0, y2_0, y3_0, z_0 \in \mathbb{Z} \wedge (x1, x2) = (x1_0, x2_0) \wedge y3 \cdot y1^{y2} = x1^{x2} \wedge y2 = 0 \wedge z = x1^{x2}\}$

Algorithme 3: Algorithme de l'exponentiation indienne annoté

```

        printf("%d_%d_%d_cz=%d_%d\n",cu,cv,cw,cz,ct);
        cz=cz+cv+cw;
        cv=cv+ct;
        ct=ct+6;
        cw=cw+3;
        cu=cu+1;
        k=k+1;}

    r=cz;
    return(r);
}

int p(int x)
{
    int r;
    if (x==0)
    {
        r=0;

    }
    else
    {
        r= p(x-1)+3*(x-1)*(x-1) + 3*(x-1)+1;

    }
    return(r);
}

int check(int n){
    int r1,r2,r;
    r1 = power(n);
    r2 = p(n);
    if (r1 != r2)
    { r = 0;
    }
    else
    { r = 1;
    };
    return r;
}

int main () {

    int counter;
    for( counter=0; counter<5; counter++ ) {
        int v,r;
        printf("Enter_a_natural_number:");
        scanf("%d", &v);
        r = power(v);
        printf ("Power_:_%d_---->_%d\n", v,r);

    };
}

```

Question 38.1 *Compiler ce programme et tester son exécution afin d'en dégager ses fonctionnalités.*

Question 38.2 *Annoter les fonctions principales.*

Question 38.3 *Vérifiez sa correction partielle et totale.*