

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

**Exercice 1** *Nous vous donnons des annotations que vous devez analyser avec Frama-c.*

Listing 1 – annotation3.c

**Question 1.1** */\*@ requires a >= 0 && b >= 0 ;  
@ assigns \nothing;  
@ ensures \result == \old(a)+\old(b)-2;*

```
/*@/  
int annotation(int a,int b)  
{  
    int x,y,z;  
    x = a;  
    /*@ assert l1: x == a; */  
    y = b;  
    /*@ assert l2: x == a && y == b; */  
    z = a+b-2;  
    /*@ assert l3: x == a && y == b && z==a+b-1; */  
    return(z); // \result = z  
}
```

Listing 2 – annotation4.c

**Question 1.2** */\*@ requires a >= 0 ;  
@ assigns \nothing;  
@ ensures \result == 0;*

```
/*@/  
int annotation(int a)  
{  
    int x;  
    x = a;  
    return(x);  
}
```

**Exercice 2** *Soit le petit programme suivant*

Listing 3 – 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 la correction des annotations avec *Frama-c* et trouver  $a$  pour que cela soit correctement analysé.

**Exercice 3** Soit le petit programme suivant

Listing 4 – td62.c

```
void ex(void) {
    int x0,y0,z0;
    int x=x0,y=x0,z=x0*x0;
    //@ assert l1: x == y && z == x*y;
    x = x*x;
    //@ assert l2: x == y*y && z == x;
    y = x;
    //@ assert l3: x + y + 2*z == (x0+x0)*(x0+x0);
    z = x + y + 2*z;

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

Analyser la correction des annotations avec *Frama-c*.

## TD6

**Exercice 4** Soit le petit programme suivant

Listing 5 – 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 5** La définition structurelle 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 :  $\{P\} skip \{P\}$ .

- Règle de composition : Si  $\{P\}\mathbf{S}_1\{R\}$  et  $\{R\}\mathbf{S}_2\{Q\}$ , alors  $\{P\}\mathbf{S}_1;\mathbf{S}_2\{Q\}$ .
- Si  $\{P \wedge B\}\mathbf{S}_1\{Q\}$  et  $\{P \wedge \neg B\}\mathbf{S}_2\{Q\}$ , alors  $\{P\}\mathbf{if\,B\,then\,S_1\,then\,S_2\,fi}\{Q\}$ .
- Si  $\{P \wedge B\}\mathbf{S}\{P\}$ , alors  $\{P\}\mathbf{while\,B\,do\,S\,od}\{P \wedge \neg B\}$ .
- Règle de renforcement / affaiblissement : Si  $P' \Rightarrow P$ ,  $\{P\}\mathbf{S}\{Q\}$ ,  $Q \Rightarrow Q'$ , alors  $\{P'\}\mathbf{S}\{Q'\}$ .

**Question 5.1** Simplifier les expressions suivantes :

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

**Question 5.2** On rappelle que  $\{P\}\mathbf{S}\{Q\}$  est défini par l'implication  $O \Rightarrow WP(S)(Q)$ . Pour chaque point énuméré ci-dessous, monter que la propriété  $\{P\}\mathbf{S}\{Q\}$  est valide ou pas en utilisant la définition suivante :

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

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

**Question 5.3** Utiliser frama-c pour vérifier les éléments suivants :

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

**Exercice 6** td65.c

Soit le petit programme suivant dans un fichier :

Listing 6 – td65.c

```
/*@
  assigns  \nothing;
*/
void swap1(int a, int b) {
  int x = a;
  int y = b;
  //@ assert x == a && y == b;
  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 6.1** 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 découvrirez le texte du fichier.

**Question 6.2** Cliquer à droite sur le mot-clé `assert` et cliquer sur `Prove annotation by WP`. Les boutons deviennent vert.

### Question 6.3

```
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 6.4** Ajoutez une précondition pour que les preuves soient possibles.

**Question 6.5** 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.

## MOVEX2-1

## MALG2-1

**Exercice 7** Etudier la correction de l'algorithme suivant en complétant l'invariant de boucle :

Listing 7 – 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;
}
```

## Exercice 8

On rappelle que l'annotation suivante du listing 8 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 8 – 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 *int*.

### Question 8.1 anq81.c

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

$b$

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

### Question 8.2 anq82.c

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.

## Exercice 9 td68.c

### Listing 9 – qpowers2.c

```
#include <limits.h>
/*@ axiomatic auxmath {
  @ axiom rule1: \forallall 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;
   requires x <= INT_MAX;
   requires x*x <= INT_MAX;
   assigns \nothing;
   ensures \result == x*x;
*/
int npower2(int x)
{int r,k,cv,cw;
  r=0;k=0;cv=0;cw=0;
  /*@ 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;
     @ loop variant x-k;
  */
  while (k<x)
  {
      k=k+1;
      cv=cv+cw+1;
      cw=cw+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;
        requires n*n <= INT_MAX;
    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;
}

```

```

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

int check2(int n){
    int r1,r2,r;
    r1 = power2(n);
    r2 = npower2(n);
    if (r1 != r2)
    { r = 0;
    }
    else
    { r = 1;
    };
    return r;
}

```

}

Listing 10 – mainpower2.c

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

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;
    while (k<x)
    {
        ok=k; ocv=cv; ocw=cw;
        k=ok+1;
        cv=ocv+ocw+1;
        cw=ocw+2;
    }
    r=cv;
    return(r);
}

int p(int x)
{
    int r;
    if (x==0)
    {
        r=0;
    }
    else
    {
        r= p(x-1)+2*x-1;
    }
    return(r);
}

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

int main()
{
    int val1, val2, val3, num;
    printf("Enter_a_number:_");
    scanf("%d", &num);
    val1 = power2(num);
    val2 = p(num);
}
```



```

    val3 = check(num);
    printf("Et le résultat pour n=%d: %d %d %d\n", num, val1, val2, val3);
    return 0;
}

```

Soit le fichier `qpower2.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 `mainpower2.c` peut être compilé pour que vous puissiez faire des expérimentations sur les valeurs calculées.

**Question 9.1** Compléter le fichier `qpower2.c` et produire le fichier `power2.c` qui est vérifié avec `fraama-c`.

**Question 9.2** Simplifier la fonction itérative en supprimant les variables commençant par la lettre `o`. Puis vérifier les fonctions obtenues avec `frama-c`.

**Question 9.3** En fait, vous avez montré que les deux fonctions étaient équivalentes. Expliquez pourquoi en quelques lignes.

#### Exercice 10 `td71.c`

Soit le contrat suivant :

```

variables X, Y, Z
requires  $x_0 \geq 0 \wedge y_0 \geq 0 \wedge z_0 \geq 0$  Roots1st  $\wedge z_0 = 25 \wedge y_0 = x_0 + 1$ 
ensures  $z_f = 100$ ;
begin
  0 :  $x^2 + y^2 = z \wedge z = 25$ ;
  (X, Y, Z) := (X+3, Y+4, Z+75);
  1 :  $x^2 + y^2 = z$ ;
end

```

**Question 10.1** Traduire ce contrat avec le langage `PlusCal` et proposer une validation pour que ce contrat soit valide.

**Question 10.2** 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.

#### Exercice 11 `anq11.c`

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

```

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

```

#### Exercice 12 `anq12.c`

Définir une fonction `abs` (`anq12.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; }

```

**Exercice 13** *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);
}
```

**Exercice 14 Question 14.1** *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.*

**Question 14.2** *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;
}
```

**Question 14.3** *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.

## Reprise

**Exercice 15** 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 15.1**

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

**Question 15.2**

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

**Question 15.3**

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

**Exercice 16** Evaluer la validité de chaque annotation dans les questions suivantes.

**Question 16.1**

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

**Question 16.2**

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

**Question 16.3**

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

Soit l'annotation suivante :

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

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

**Question 16.4** Ecrire la condition de vérification correspondant à cette annotation en supposant que  $X$  et  $Y$  sont deux variables entières.

**Question 16.5** Etudier la validité de cette condition de vérification selon la valeur de  $m$ .

**Exercice 17** *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  $v(0) \dots v(n)$ . On suppose que l'opérateur  $\oplus$  est défini comme suit  $a \oplus b = \max(a, b)$ .

**Question 17.1** Ecrire une solution contractuelle de cet algorithme.

**Question 17.2** 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 17.3** 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 18 *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 18.1** 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 19** Soit le petit programme suivant :

Listing 11 – 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 20** Soit le petit programme suivant :

Listing 12 – 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  
Utilisation d'un environnement de vérification Frama-c (III)  
par Dominique Méry  
25 mars 2025

**Exercice 21** *Utiliser frama-c pour vérifier ou non les annotations suivantes :*

**Question 21.1**

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

**Question 21.2**

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

**Question 21.3**

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

**Question 21.4**

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

**Question 21.5**

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

**Question 21.6**

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

**Exercice 22** *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 23** 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 1:** Algorithme du maximum d'une liste non annotée

**Exercice 24**

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 25** Soit la fonction suivante utilisée dans un programme

Listing 13 – 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**  $\text{impair}(y2)$  **then**

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

$y2 := y2 - 1;$

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

$y3 := y3 \cdot y1;$

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

;

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

$y1 := y1 \cdot y1;$

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

$y2 := y2 \text{ div } 2;$

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

;

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

$z := y3;$

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

**Algorithme 2:** 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 25.1** *Compiler ce programme et tester son exécution afin d'en dégager ses fonctionnalités.*

**Question 25.2** *Annoter les fonctions principales.*

**Question 25.3** *Vérifiez sa correction partielle et totale.*