

Cours Modélisation et vérification des systèmes informatiques  
 Exercices (avec les corrections)  
 Utilisation d'un environnement de vérification Frama-c (I)  
 par Dominique Méry  
 4 décembre 2025

**Exercice 1** Soit le petit programme suivant

Listing 1 – 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 2** Soit le petit programme suivant

Listing 2 – 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.

**Exercice 3** Soit le petit programme suivant

Listing 3 – 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 4** 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	$P$
$S_1; S_2$	$wp(S_1)(wp(S_2)(P))$
<b>IF</b> $B$ $S_1$ <b>ELSE</b> $S_2$ <b>FI</b>	$(B \Rightarrow wp(S_1)(P)) \wedge (\neg B \Rightarrow wp(S_2)(P))$

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

**Question 4.1** Simplifier les expressions suivantes :

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

**Question 4.2** On rappelle que  $\{P\}S\{Q\}$  est défini par l'implication  $O \Rightarrow WP(S)(Q)$ . Pour chaque point énuméré ci-dessous, montrer 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.  $\{x < y\}IF x \neq y THEN x := 5 ELSE x := 8 FI\{x \in \{5, 8\}\}$

**Question 4.3** Utiliser frama-c pour vérifier les éléments suivants :

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

**Exercice 5** *td65.c*

Soit le petit programme suivant dans un fichier :

Listing 4 – *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 5.1** 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 5.2** Cliquer à droite sur le mot-clé `assert` et cliquer sur *Prove annotation by WP*. Les boutons deviennent vert.

**Question 5.3**

```

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 5.4** Ajoutez une précondition pour que les preuves soient possibles.

◊ **Solution de la question 5.4** \_\_\_\_\_

```

/*@ 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 5.4**

**Question 5.5** 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 6** *Etudier la correction de l'algorithme suivant en complétant l'invariant de boucle :*

Listing 5 – 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 6 – td66c.c

```

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

    /*@ loop invariant i * i == s && 0 <= i && 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 7**

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

- $\text{pre}(v_0) \wedge v = v_0 \Rightarrow A(v_0, v)$
- $\text{pre}(v_0) \wedge B(v_0, v) \Rightarrow \text{post}(v_0, v)$
- $A(v_0, v) \Rightarrow \text{wp}(v = f(v))(B(v_0, v))$  où  $\text{wp}(v = f(v))(B(v_0, v))$  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  $\text{at}(v, \text{Pre})$  et aussi  $\text{old}(v)$ . Nous utiliserons la notation  $v_0$  dans cet exercice.

#### Listing 7 – 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 7.1

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

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

#### Listing 8 – 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 7.2 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 ce soit correcte.

#### Listing 9 – td68.c

**Exercice 8** // #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 `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 8.1** Compléter le fichier `qpower2.c` et produire le fichier `power2.c` qui est vérifié avec `fraama-c`.

**Question 8.2** a Simplifier la fonction itérative en supprimant les variables commençant par la lettre `o`. Puis vérifier les fonctions obtenues avec `frama-c`.

**Question 8.3** En fait, vous avez montré que les deux fonctions étaient équivalentes. Expliquez pourquoi en quelques lignes.