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

Exercice 1 *Soit le contrat suivant :*

variables X, Y, Z
requires $x_0 >= 0 \wedge y_0 >= 0 \wedge z_0 >= 0 \wedge z_0 = 25 \wedge y_0 = x_0+1$
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 1.1 *Traduire ce contrat avec le langage PlusCal et proposer une validation pour que ce contrat soit valide.*

Listing 1 – 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 2 – 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 1.2 *Traduire ce contrat en ACSL et vÃ©rifier qu'il est valide ou non. S'il est non valide, proposer une correction de la prÃ©-condition et/ou de la postcondition.*

Listing 3 – 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 2 *Définir une fonction* maxpointer *(gex1.c) calculant la valeur du maximum du ciontenu de deux adresses avec son contrat.*

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

◇– **Solution de l'exercice 2**

Listing 4 – 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 2

Exercice 3 *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 3**

Fin 3

Exercice 4 *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 4**

Listing 5 – 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 6 – 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 7 – 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 8 – 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 4

◇— **Solution de l'exercice 4**

Fin 4

Exercice 5 **Question 5.1** *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 5.1**

```
/*@ 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 5.1

Question 5.2 *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 5.2**

Listing 9 – 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 5.2

Question 5.3 *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 5.3**

Listing 10 – 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 5.3

Reprise

Exercice 6 *Pour chaque question, montrer que l' annotation est correcte ou incorrecte selon les conditions de vérifications énoncées comme suit*
$\forall x, y, , x', y'.P_\ell(x,y) \ \wedge \ cond_{\ell,\ell'}(x,y) \ \wedge \ (x',y') = f_{\ell,\ell'}(x,y) \Rightarrow P_{\ell'}(x',y')$
Pour cela, on utilisera l'environnement Frama-c.

Question 6.1

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

◇– **Solution de la question 6.1**

Listing 11 – 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 6.1

Question 6.2

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

Question 6.3

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

Exercice 7 *(6 points)*
Evaluer la validité de chaque annotation dans les questions suivent.

Question 7.1

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

Question 7.2

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

Question 7.3

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

Soit l'annotation suivante :

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

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

Question 7.4 *Ecrire la condition de vérification correspondant à cette annotation en upposant que X et Y sont deux variables entières.*

Question 7.5 *Etudier la validité de cette condition de vérification selon la valeur de m.*

Exercice 8 *gex7.c*

VARIABLES N, V, S, I

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

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

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

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

$S := V(0)$

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

$I := 1$

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

WHILE $I < N$ *DO*

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

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

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

$I := I{+}1$

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

OD;

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

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

Question 8.1 *Ecrire une solution contractuelle de cet algorithme.*

Question 8.2 *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 8.3 *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 9 *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 9.1 *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***
ℓ_2 :
$X := X \cdot X + 2 \cdot B \cdot X + B \cdot B;$
ℓ_3 :
ELSE
ℓ_4 :
$X := B;$
ℓ_5 :
FI
ℓ_6 :
END

Exercice 10 *Soit le petit programme suivant :*

Listing 12 – f91

```
#include <limits.h>

/*@ requires  INT_MIN <= x-10;
    requires x-10 <= INT_MAX;
    assigns  \nothing;
  ensures 100 < x ==> \result  == x -10;
  ensures x <= 100 ==> \result == 91;
*/
int f1(int x)
{ if (x > 100)
    { return(x-10);
    }
  else
    { return(f1(f1(x+11)));
    }
}


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


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

```
    }
  else
    { return(91);
    }
}


/*@    requires  INT_MIN <= n-10;
     requires n-10 <= INT_MAX;
  assigns  \nothing;
  ensures  \result  == 0;
*/

int f(int n){
  int r1,r2,r;
  r1 = f1(n);
  r2 = f2(n);
  if (r1 == r2)
    { r = 1;
    }
  else
    { r = 0;
    };
  return r;
 }
```

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

Exercice 11 *Soit le petit programme suivant :*

Listing 13 – 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;
     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;
 }
```

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