Cours Modélisation et vérification des systèmes informatiques
Exercices (avec les corrections)
Modélisation d'algorithmes en PlusCal (II)
par Dominique Méry
17 novembre 2024

Exercice 1 *(Vérification de l'annotation de l'algorithme du calcul du maximum d'une liste) appex5_1.tla*

Question 1.1 *Ecrire un module TLA$^+$ contenant une définition PlusCal de cet algorithme.*

Question 1.2 *Ecrire la propriété à vérifier pour la correction partielle.*

Question 1.3 *Ecrire la propriété à vérifier pour l'absence d'erreurs à l'exécution.*

Vérification **precondition** : $\left(\begin{array}{l} n \in \mathbb{N} \wedge \\ n \neq 0 \wedge \\ f \in 0\,..\,n-1 \rightarrow \mathbb{N} \end{array}\right)$

postcondition : $\left(\begin{array}{l} m \in \mathbb{N} \wedge \\ m \in ran(f) \wedge \\ (\forall j \cdot j \in 0\,..\,n-1 \Rightarrow f(j) \leq m) \end{array}\right)$

local variables : $i \in \mathbb{Z}$

$m := f(0)$;
$i := 1$;
while $i < n$ **do**
  if *f(i)>m* **then**
    $m := f(i)$;
  ;
  i++;
;

Algorithme 1: Algorithme du maximum d'une liste non annotée

Listing 1 – appex5-1.tla

```
---------------------- MODULE appex5_1 --------------------------------
EXTENDS Naturals, Integers, TLC

CONSTANT n0

f0 == [k \in 0..n0-1 |->
                    IF k=0 THEN 3
                    ELSE IF k=1 THEN 6
                    ELSE IF k=2 THEN 2*k
                    ELSE IF k=3 THEN 9
                    ELSE 5]


(*
-termination
-wfNext
--algorithm Maximum {
  variables i = 0;
```

/* algorithme de calcul du maximum avec une boucle while de l'exercice **??** */

precondition : $\begin{pmatrix} n \in \mathbb{N} \wedge \\ n \neq 0 \wedge \\ f \in 0 \,..\, n{-}1 \rightarrow \mathbb{N} \end{pmatrix}$

postcondition : $\begin{pmatrix} m \in \mathbb{N} \wedge \\ m \in ran(f) \wedge \\ (\forall j \cdot j \in 0 \,..\, n{-}1 \Rightarrow f(j) \leq m) \end{pmatrix}$

local variables : $i \in \mathbb{Z}$

$\ell_0 : \{ \begin{pmatrix} n \in \mathbb{N} \wedge \\ n \neq 0 \wedge \\ f \in 0 \,..\, n{-}1 \rightarrow \mathbb{N} \end{pmatrix} \wedge i \in \mathbb{Z} \wedge i \in \mathbb{Z} \wedge ...\}$

$m := f(0);$

$\ell_1 : \{ \begin{pmatrix} n \in \mathbb{N} \wedge \\ n \neq 0 \wedge \\ f \in 0 \,..\, n{-}1 \rightarrow \mathbb{N} \end{pmatrix} \wedge i \in \mathbb{Z} \wedge m = f(0)\}$

$i := 1;$

$\ell_2 : \{ \begin{pmatrix} n \in \mathbb{N} \wedge \\ n \neq 0 \wedge \\ f \in 0 \,..\, n{-}1 \rightarrow \mathbb{N} \end{pmatrix} \wedge i = 1 \wedge \begin{pmatrix} m \in \mathbb{N} \wedge \\ m \in ran(f[0..i{-}1]) \wedge \\ (\forall j \cdot j \in 0 \,..\, i{-}1 \Rightarrow f(j) \leq m) \end{pmatrix}\}$

while $i < n$ **do**

$\ell_3 : \{ \begin{pmatrix} n \in \mathbb{N} \wedge \\ n \neq 0 \wedge \\ f \in 0 \,..\, n{-}1 \rightarrow \mathbb{N} \end{pmatrix} \wedge i \in 1..n{-}1 \wedge \begin{pmatrix} m \in \mathbb{N} \wedge \\ m \in ran(f[0..i{-}1]) \wedge \\ (\forall j \cdot j \in 0 \,..\, i{-}1 \Rightarrow f(j) \leq m) \end{pmatrix}\}$

if *f(i)>m* **then**

$\ell_4 : \{ \begin{pmatrix} n \in \mathbb{N} \wedge \\ n \neq 0 \wedge \\ f \in 0 \,..\, n{-}1 \rightarrow \mathbb{N} \end{pmatrix} \wedge i \in 1..n{-}1 \wedge \begin{pmatrix} m \in \mathbb{N} \wedge \\ m \in ran(f[0..i{-}1]) \wedge \\ (\forall j \cdot j \in 0 \,..\, i{-}1 \Rightarrow f(j) \leq m) \end{pmatrix} \wedge$
$f(i) > m\}$

$m := f(i);$

$\ell_5 : \{ \begin{pmatrix} n \in \mathbb{N} \wedge \\ n \neq 0 \wedge \\ f \in 0 \,..\, n{-}1 \rightarrow \mathbb{N} \end{pmatrix} \wedge i \in 1..n{-}1 \wedge \begin{pmatrix} m \in \mathbb{N} \wedge \\ m \in ran(f[0..i]) \wedge \\ (\forall j \cdot j \in 0 \,..\, i \Rightarrow f(j) \leq m) \end{pmatrix}\}$

;

$\ell_6 : \{ \begin{pmatrix} n \in \mathbb{N} \wedge \\ n \neq 0 \wedge \\ f \in 0 \,..\, n{-}1 \rightarrow \mathbb{N} \end{pmatrix} \wedge i \in \mathbb{Z} \wedge \wedge i \in 1..n{-}1 \wedge \begin{pmatrix} m \in \mathbb{N} \wedge \\ m \in ran(f[0..i]) \wedge \\ (\forall j \cdot j \in 0 \,..\, i \Rightarrow f(j) \leq m) \end{pmatrix}\}$

$i{+}{+};$

$\ell_7 : \{ \begin{pmatrix} n \in \mathbb{N} \wedge \\ n \neq 0 \wedge \\ f \in 0 \,..\, n{-}1 \rightarrow \mathbb{N} \end{pmatrix} \wedge i \in 1..n{-}1 \wedge \begin{pmatrix} m \in \mathbb{N} \wedge \\ m \in ran(f[0..i{-}1]) \wedge \\ (\forall j \cdot j \in 0 \,..\, i{-}1 \Rightarrow f(j) \leq m) \end{pmatrix}\}$

;

$\ell_8 : \{ \begin{pmatrix} n \in \mathbb{N} \wedge \\ n \neq 0 \wedge \\ f \in 0 \,..\, n{-}1 \rightarrow \mathbb{N} \end{pmatrix} \wedge i = n \wedge \begin{pmatrix} m \in \mathbb{N} \wedge \\ m \in ran(f) \wedge \\ (\forall j \cdot j \in 0 \,..\, n{-}1 \Rightarrow f(j) \leq m) \end{pmatrix}\}$

Algorithme 2: Algorithme du maximum d'une liste annoté

```
            m=0;
            f=f0;
            n=n0;
            r;
{
            l0:m:=f[0];
            l1:i:=1;
            l2: while (i<n) {
            l3: if (f[i]>m){
            l4: m:=f[i];
            } ;
            l5:i:=i+1;
            };
            r := m;
}
}
*)
```

===

Exercice 2 *Exponentiation appex5_2.tla*
Soit l'algorithme annoté calculant la puissance $z = {x_1}^{x_2}$.

- *Precondition :* $x_1 \in \mathbb{N} \wedge x_2 \wedge \mathbb{N}$
- *Postcondition :* $z = {x_1}^{x_2}$

On suppose que x_1 *et* x_2 *sont des constantes.*

Question 2.1 *Ecrire un module TLA/TLA*$^+$ *permettant de valider les conditions de vérification et, en particulier, de montrer la correction partielle.*

Question 2.2 *Modifier la machine pour prendre en compte l'absence d'erreurs à l'exécution.*

Listing 2 – appex5-2.tla

```
---------------------- MODULE appex5_2 --------------------------------
EXTENDS Naturals, Integers, TLC
---------------------------------------------
CONSTANT MAXINT, x10, x20, MININT
----------------------
typeInt(u) == u \in Int
pre == x10 \in Nat /\ x20 \in Nat /\ x10 # 0
-----------------------------------------------------------------
(* precondition *)
ASSUME pre
----------------------------------------------------
(*
--algorithm Exponentiation {
  variables
            x1=x10;
            x2=x20;
            y1;
            y2;
            y3;
            z;
{
     l0:
     y1:=x1; y2:=x2; y3:=1;
```

precondition : $x_1 \in \mathbb{N} \wedge x_2 \in \mathbb{N} \wedge x_1 \neq 0$
postcondition : $z = {x_1}^{x_2}$
local variables : $y_1, y_2, y_3 \in \mathbb{Z}$

$\ell_0 : \{y_1, y_2, y_3, z \in \mathbb{Z}\}$
$y_1 := x_1; y_2 := x_2 : y_3 := 1;$
$\ell_1 : \{y_1 = x_1 \wedge y_2 = x_2 \wedge y_3 = 1 \wedge y_1, y_2, y_3 \in \mathbb{N} \wedge z \in \mathbb{Z}\}$
$\ell_{11} : \{y_3 \cdot {y_1}^{y_2} = {x_1}^{x_2} \wedge y_1, y_2, y_3 \in \mathbb{N} \wedge z \in \mathbb{Z}\}$
while $y_2 \neq 0$ **do**
  $\ell_2 : \{y_2 \neq 0 \wedge y_3 \cdot {y_1}^{y_2} = {x_1}^{x_2} \wedge y_1, y_2, y_3 \in \mathbb{N} \wedge z \in \mathbb{Z}\}$
  if $impair(y_2)$ **then**
    $\ell_3 : \{impair(y_2) \wedge y_2 \neq 0 \wedge y_3 \cdot {y_1}^{y_2} = {x_1}^{x_2} \wedge y_1, y_2, y_3 \in \mathbb{N} \wedge z \in \mathbb{Z}\}$
    $y_2 := y_2 - 1;$
    $\ell_4 : \{y_2 \geq 0 \wedge pair(y_2) \wedge y_3 \cdot y_1 \cdot {y_1}^{y_2} = {x_1}^{x_2} \wedge y_1, y_2, y_3 \in \mathbb{N} \wedge z \in \mathbb{Z}\}$
    $y_3 := y_3 \cdot y_1;$
    $\ell_5 : \{y_2 \geq 0 \wedge pair(y_2) \wedge y_3 \cdot {y_1}^{y_2} = {x_1}^{x_2} \wedge y_1, y_2, y_3 \in \mathbb{N} \wedge z \in \mathbb{Z}\}$
  ;
  $\ell_6 : \{y_2 \geq 0 \wedge pair(y_2) \wedge y_3 \cdot {y_1}^{y_2} = {x_1}^{x_2} \wedge y_1, y_2, y_3 \in \mathbb{N} \wedge z \in \mathbb{Z}\}$
  $y_1 := y_1 \cdot y_1;$
  $\ell_7 : \{y_2 \geq 0 \wedge pair(y_2) \wedge y_3 \cdot y_1 .^{y_2 \ div 2} = {x_1}^{x_2} \wedge y_1, y_2, y_3 \in \mathbb{N} \wedge z \in \mathbb{Z}\}$
  $y_2 := y_2 \ div \ 2;$
  $\ell_8 : \{y_2 \geq 0 \wedge y_3 \cdot {y_1}^{y_2} = {x_1}^{x_2} \wedge y_1, y_2, y_3 \in \mathbb{N} \wedge z \in \mathbb{Z}\}$
;
$\ell_9 : \{y_2 = 0 \wedge y_3 \cdot {y_1}^{y_2} = {x_1}^{x_2} \wedge y_1, y_2, y_3 \in \mathbb{N} \wedge z \in \mathbb{Z}\}$
$z := y_3;$
$\ell_{10} : \{y_2 = 0 \wedge y_3 \cdot {y_1}^{y_2} = {x_1}^{x_2} \wedge y_1, y_2, y_3 \in \mathbb{N} \wedge z \in \mathbb{Z} \wedge z = {x_1}^{x_2}\}$

Algorithme 3: Version solution annotée

```
    w: while (y2 /= 0) {

      l2:

if ( y2 % 2 # 0) {
        l3:y2:=y2-1;
        l4:y3:=y3*y1;
        l5:skip;
      };
      l6:y1 := y1*y1;     l7:y2:= y2 \div  2;
      l8:skip;
    };
    l9: z := y3;
    l10: print <<x1, x2,z>>;
}

}
*)
\* BEGIN TRANSLATION (chksum(pcal) = "14eb71f" /\ chksum(tla) = "f9286308")
CONSTANT defaultInitValue
VARIABLES x1, x2, y1, y2, y3, z, pc

vars == << x1, x2, y1, y2, y3, z, pc >>

Init == (* Global variables *)
        /\ x1 = x10
        /\ x2 = x20
        /\ y1 = defaultInitValue
        /\ y2 = defaultInitValue
        /\ y3 = defaultInitValue
        /\ z = defaultInitValue
        /\ pc = "l0"

l0 == /\ pc = "l0"
      /\ y1'␣=␣x1
␣␣␣␣␣␣/\␣y2' = x2
      /\ y3'␣=␣1
␣␣␣␣␣␣/\␣pc' = "w"
      /\ UNCHANGED << x1, x2, z >>

w == /\ pc = "w"
     /\ IF y2 /= 0
           THEN /\ pc'␣=␣"l2"
␣␣␣␣␣␣␣␣␣␣␣ELSE␣/\␣pc' = "l9"
     /\ UNCHANGED << x1, x2, y1, y2, y3, z >>

l2 == /\ pc = "l2"
      /\ IF y2 % 2 # 0
            THEN /\ pc'␣=␣"l3"
␣␣␣␣␣␣␣␣␣␣␣␣ELSE␣/\␣pc' = "l6"
      /\ UNCHANGED << x1, x2, y1, y2, y3, z >>

l3 == /\ pc = "l3"
      /\ y2'␣=␣y2-1
␣␣␣␣␣␣/\␣pc' = "l4"
      /\ UNCHANGED << x1, x2, y1, y3, z >>
```

```
l4 == /\ pc = "l4"
       /\ y3'␣=␣y3*y1
␣␣␣␣␣␣/\␣pc' = "l5"
       /\ UNCHANGED << x1, x2, y1, y2, z >>

l5 == /\ pc = "l5"
       /\ TRUE
       /\ pc'␣=␣"l6"
␣␣␣␣␣␣/\␣UNCHANGED␣<<␣x1,␣x2,␣y1,␣y2,␣y3,␣z␣>>

l6␣==␣/\␣pc␣=␣"l6"
␣␣␣␣␣␣/\␣y1' = y1*y1
       /\ pc'␣=␣"l7"
␣␣␣␣␣␣/\␣UNCHANGED␣<<␣x1,␣x2,␣y2,␣y3,␣z␣>>

l7␣==␣/\␣pc␣=␣"l7"
␣␣␣␣␣␣/\␣y2' = (y2 \div  2)
       /\ pc'␣=␣"l8"
␣␣␣␣␣␣/\␣UNCHANGED␣<<␣x1,␣x2,␣y1,␣y3,␣z␣>>

l8␣==␣/\␣pc␣=␣"l8"
␣␣␣␣␣␣/\␣TRUE
␣␣␣␣␣␣/\␣pc' = "w"
       /\ UNCHANGED << x1, x2, y1, y2, y3, z >>

l9 == /\ pc = "l9"
       /\ z'␣=␣y3
␣␣␣␣␣␣/\␣pc' = "l10"
       /\ UNCHANGED << x1, x2, y1, y2, y3 >>

l10 == /\ pc = "l10"
        /\ PrintT(<<x1, x2,z>>)
        /\ pc'␣=␣"Done"
␣␣␣␣␣␣␣/\␣UNCHANGED␣<<␣x1,␣x2,␣y1,␣y2,␣y3,␣z␣>>

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

Next␣==␣l0␣\/␣w␣\/␣l2␣\/␣l3␣\/␣l4␣\/␣l5␣\/␣l6␣\/␣l7␣\/␣l8␣\/␣l9␣\/␣l10
␣␣␣␣␣␣␣␣␣␣␣\/␣Terminating

Spec␣==␣Init␣/\␣[][Next]_vars

Termination␣==␣<>(pc␣=␣"Done")

\*␣END␣TRANSLATION



L␣==␣{"l0","l1"}
D␣==␣MININT..MAXINT

DD(X)␣==␣X=defaultInitValue␣=>␣X␣\in␣D

i␣==
```

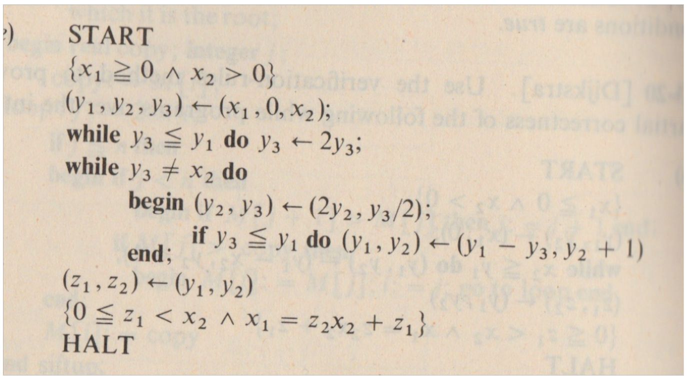
```
␣␣␣␣/\␣pc␣\in␣L
␣␣␣␣/\␣DD(y1)␣/\␣DD(y2)␣/\␣DD(y3)␣/\␣DD(z)
␣␣␣␣/\␣typeInt(x1)/\␣typeInt(x2)␣/\␣typeInt(y1)␣/\␣typeInt(y2)␣/\␣typeInt(y3)␣/\␣ty
␣␣␣␣/\␣pc="y0"␣=>␣x1=x10␣/\␣x2=x20
␣␣␣␣/\␣pc="l1"␣=>␣x1=x10␣/\␣x2␣=␣x20␣/\␣␣y2␣\geq␣0␣␣/\␣y3*␣y1^y2␣=␣x1^x2
␣␣␣␣/\␣␣pc␣=␣"w"␣=>␣x1=x10␣/\␣x2␣=␣x20␣/\␣␣y2␣\geq␣0␣␣/\␣y3*␣y1^y2␣=␣x1^x2
␣␣␣␣/\␣pc="l2"␣=>␣␣y2␣#␣0␣/\␣y3*␣y1^y2␣=␣x1^x2



Q1␣==␣pc␣#␣"Done"
Qpc␣==␣pc␣="Done"␣=>␣z=x1^x2

=============================================================================
\*␣Modification␣History
\*␣Last␣modified␣Sun␣Nov␣17␣20:05:05␣CET␣2024␣by␣mery
\*␣Created␣Wed␣Sep␣09␣17:02:47␣CEST␣2015␣by␣mery
```

Exercice 3 *(appex5_3.tla)*
On considère l'algorithme suivant :



START
$\{x_1 \geqq 0 \wedge x_2 > 0\}$
$(y_1, y_2, y_3) \leftarrow (x_1, 0, x_2)$;
while $y_3 \leqq y_1$ **do** $y_3 \leftarrow 2y_3$;
while $y_3 \neq x_2$ **do**
begin $(y_2, y_3) \leftarrow (2y_2, y_3/2)$;
if $y_3 \leqq y_1$ **do** $(y_1, y_2) \leftarrow (y_1 - y_3, y_2 + 1)$
end;
$(z_1, z_2) \leftarrow (y_1, y_2)$
$\{0 \leqq z_1 < x_2 \wedge x_1 = z_2 x_2 + z_1\}$
HALT

Question 3.1 *Montrer que cet algorithme est aprtiellement correct par rapport à sa précondition et à sa postcondition qu'il faudra énoncer. Pour cela, on traduira cet algorithme sous forme d'un module à partir du langage PlusCal.*

Question 3.2 *Montrer qu'il est sans erreur à l'exécution.*

Listing 3 – appex5-3.tla

```
---------------------------------- MODULE appex5_3 ----------------------------------
EXTENDS TLC, Integers , Naturals
CONSTANTS x1 , x2 , min , max

(*
-wfNext
--algorithm division {
variables y1 , y2 , y3 , z1 , z2;
{
l1 : y1:=x1 ; y2:=0; y3:=x2 ;
l2 : while (y3 \leq y1){
     y3:=2*y3 ;
```

```
    };
 l3:while (y3#x2){
    assert x1=y2*x2+y1;
    y2:=2*y2;
    y3:=y3 \div 2;
    l4:if (y3\leq y1) {
    y1:=y1-y3;
    y2:=y2+1;
    };
    assert x1=y2*x2+y1;
    };
   l5: z1:=y1;
    z2:=y2;
    assert x1=y2*x2+y1;
    print <<x1,x2,z1,z2>>;
    }
 }

*)
\* BEGIN TRANSLATION
CONSTANT defaultInitValue
VARIABLES y1, y2, y3, z1, z2, pc

vars == << y1, y2, y3, z1, z2, pc >>

Init == (* Global variables *)
        /\ y1 = defaultInitValue
        /\ y2 = defaultInitValue
        /\ y3 = defaultInitValue
        /\ z1 = defaultInitValue
        /\ z2 = defaultInitValue
        /\ pc = "l1"

l1 == /\ pc = "l1"
      /\ y1'␣=␣x1
␣␣␣␣␣␣/\␣y2' = 0
      /\ y3'␣=␣x2
␣␣␣␣␣␣/\␣pc' = "l2"
      /\ UNCHANGED << z1, z2 >>

l2 == /\ pc = "l2"
      /\ IF y3 \leq y1
            THEN /\ y3'␣=␣2*y3
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣/\␣pc' = "l2"
            ELSE /\ pc'␣=␣"l3"
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣/\␣y3' = y3
      /\ UNCHANGED << y1, y2, z1, z2 >>

l3 == /\ pc = "l3"
      /\ IF y3#x2
            THEN /\ Assert(x1=y2*x2+y1,
                           "Failure␣of␣assertion␣at␣line␣15,␣column␣5.")
                 /\ y2'␣=␣2*y2
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣/\␣y3' = (y3 \div 2)
                 /\ pc'␣=␣"l4"
␣␣␣␣␣␣␣␣␣␣␣ELSE␣/\␣pc' = "l5"
```

```
                    /\ UNCHANGED << y2, y3 >>
       /\ UNCHANGED << y1, z1, z2 >>

l4 == /\ pc = "l4"
       /\ IF y3\leq y1
             THEN /\ y1'␣=␣y1-y3
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣/\␣y2' = y2+1
             ELSE /\ TRUE
                    /\ UNCHANGED << y1, y2 >>
       /\ Assert(x1=y2'*x2+y1', "Failure␣of␣assertion␣at␣line␣22,␣column␣5.")
       /\ pc'␣=␣"l3"
␣␣␣␣␣␣␣/\␣UNCHANGED␣<<␣y3,␣z1,␣z2␣>>

l5␣==␣/\␣pc␣=␣"l5"
␣␣␣␣␣␣␣/\␣z1' = y1
       /\ z2'␣=␣y2
␣␣␣␣␣␣␣/\␣Assert(x1=y2*x2+y1,␣"Failure␣of␣assertion␣at␣line␣26,␣column␣5.")
␣␣␣␣␣␣␣/\␣PrintT(<<x1,x2,z1',z2'>>)
␣␣␣␣␣␣␣/\␣pc' = "Done"
       /\ UNCHANGED << y1, y2, y3 >>

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

Next == l1 \/ l2 \/ l3 \/ l4 \/ l5
           \/ Terminating

Spec == Init /\ [][Next]_vars

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

\* END TRANSLATION


Iloop(u,v)  == x1=v*x2+u
Qpc == pc="Done" => x1=z2*x2+z1 /\ 0 \leq z1 /\ z1 \leq x2
COND(U) ==    min \leq U /\ U \leq max

Qof ==  COND(y1) /\ COND(y2) /\ COND (y3) /\ COND(z1) /\ COND (z2)

i == Iloop(y1,y2)
================================================================================
\* Modification History
\* Last modified Tue Nov 24 21:30:57 CET 2020 by mery
\* Created Wed Nov 18 16:33:27 CET 2015 by mery
```

Exercice 4 *annotation*
Montrer que chaque 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 utlisera une machine et un ciontexte Event-B.

—

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

—

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

— *On suppose que p est un nombre premier :*

$$\begin{array}{|l|}\hline \ell_1 : x = 2^p \;\wedge\; y = 2^{p+1} \;\wedge\; x{\cdot}y = 2^{2\cdot p+1} \\ x := y{+}x{+}2^x \\ \ell_2 : x = 5{\cdot}2^p \;\wedge\; y = 2^{p+1} \\ \hline\end{array}$$

—

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

Exercice 5 *(Vérification de l'annotation de l'algorithme du calcul du maximum d'une liste) Vérifier l'annotation de l'algorithme de calcul du maximum d'une liste* **??**. *On se donne l'annotation et on demande de construire une machine permettant de vérifier cette annotation.*

Vérification **precondition** : $\begin{pmatrix} n \in \mathbb{N} \wedge \\ n \neq 0 \wedge \\ f \in 0\,..\,n{-}1 \rightarrow \mathbb{N} \end{pmatrix}$

postcondition : $\begin{pmatrix} m \in \mathbb{N} \wedge \\ m \in ran(f) \wedge \\ (\forall j \cdot j \in 0\,..\,n{-}1 \Rightarrow f(j) \leq m) \end{pmatrix}$

local variables : $i \in \mathbb{Z}$

$m := f(0)$;
$i := 1$;
while $i < n$ **do**
  if *f(i)>m* **then**
    $m := f(i)$;
  ;
  $i{+}{+}$;
;

Algorithme 4: Algorithme du maximum d'une liste non annotée

CONTEXT context0 SETS
C
CONSTANTS
f n l0 l1 l2 l3 l4 l5 l6 l7 l8 l9

axm1 : $n \in \mathbb{N}_1$
axm2 : $f \in 0\,..\,n{-}1 \rightarrow \mathbb{N}$
axm3 : $partition(C, \{l0\}, \{l1\}, \{l2\}, \{l3\}, \{l4\}, \{l5\}, \{l6\}, \{l7\}, \{l8\}, \{l9\})$
axm4 : $\forall P \cdot P \subseteq \mathbb{N} \wedge finite(P) \Rightarrow (\exists am \cdot am \in P \wedge (\forall k \cdot k \in P \Rightarrow k \leq am))$

algorithm
context0 **VARIABLES**
l m i
INVARIANTS

inv1 : $l \in C$
inv2 : $m \in \mathbb{N}$
inv3 : $i \in \mathbb{N}$
inv4 : $i \in 0\,..\,n$
inv5 : $l = l0 \Rightarrow m \in \mathbb{N} \wedge i \in \mathbb{N}$

/* algorithme de calcul du maximum avec une boucle while de l'exercice **??** */

precondition : $\begin{pmatrix} n \in \mathbb{N} \wedge \\ n \neq 0 \wedge \\ f \in 0 \mathinner{..} n{-}1 \rightarrow \mathbb{N} \end{pmatrix}$

postcondition : $\begin{pmatrix} m \in \mathbb{N} \wedge \\ m \in ran(f) \wedge \\ (\forall j \cdot j \in 0 \mathinner{..} n{-}1 \Rightarrow f(j) \leq m) \end{pmatrix}$

local variables : $i \in \mathbb{Z}$

$$\ell_0 : \{ \begin{pmatrix} n \in \mathbb{N} \wedge \\ n \neq 0 \wedge \\ f \in 0 \mathinner{..} n{-}1 \rightarrow \mathbb{N} \end{pmatrix} \wedge i \in \mathbb{Z} \wedge i \in \mathbb{Z} \wedge ... \}$$

$m := f(0);$

$$\ell_1 : \{ \begin{pmatrix} n \in \mathbb{N} \wedge \\ n \neq 0 \wedge \\ f \in 0 \mathinner{..} n{-}1 \rightarrow \mathbb{N} \end{pmatrix} \wedge i \in \mathbb{Z} \wedge m = f(0) \}$$

$i := 1;$

$$\ell_2 : \{ \begin{pmatrix} n \in \mathbb{N} \wedge \\ n \neq 0 \wedge \\ f \in 0 \mathinner{..} n{-}1 \rightarrow \mathbb{N} \end{pmatrix} \wedge i = 1 \wedge \begin{pmatrix} m \in \mathbb{N} \wedge \\ m \in ran(f[0..i{-}1]) \wedge \\ (\forall j \cdot j \in 0 \mathinner{..} i{-}1 \Rightarrow f(j) \leq m) \end{pmatrix} \}$$

while $i < n$ **do**

$$\ell_3 : \{ \begin{pmatrix} n \in \mathbb{N} \wedge \\ n \neq 0 \wedge \\ f \in 0 \mathinner{..} n{-}1 \rightarrow \mathbb{N} \end{pmatrix} \wedge i \in 1..n{-}1 \wedge \begin{pmatrix} m \in \mathbb{N} \wedge \\ m \in ran(f[0..i{-}1]) \wedge \\ (\forall j \cdot j \in 0 \mathinner{..} i{-}1 \Rightarrow f(j) \leq m) \end{pmatrix} \}$$

if *f(i)>m* **then**

$$\ell_4 : \{ \begin{pmatrix} n \in \mathbb{N} \wedge \\ n \neq 0 \wedge \\ f \in 0 \mathinner{..} n{-}1 \rightarrow \mathbb{N} \end{pmatrix} \wedge i \in 1..n{-}1 \wedge \begin{pmatrix} m \in \mathbb{N} \wedge \\ m \in ran(f[0..i{-}1]) \wedge \\ (\forall j \cdot j \in 0 \mathinner{..} i{-}1 \Rightarrow f(j) \leq m) \end{pmatrix} \wedge$$
$$f(i) > m \}$$

$m := f(i);$

$$\ell_5 : \{ \begin{pmatrix} n \in \mathbb{N} \wedge \\ n \neq 0 \wedge \\ f \in 0 \mathinner{..} n{-}1 \rightarrow \mathbb{N} \end{pmatrix} \wedge i \in 1..n{-}1 \wedge \begin{pmatrix} m \in \mathbb{N} \wedge \\ m \in ran(f[0..i]) \wedge \\ (\forall j \cdot j \in 0 \mathinner{..} i \Rightarrow f(j) \leq m) \end{pmatrix} \}$$

;

$$\ell_6 : \{ \begin{pmatrix} n \in \mathbb{N} \wedge \\ n \neq 0 \wedge \\ f \in 0 \mathinner{..} n{-}1 \rightarrow \mathbb{N} \end{pmatrix} \wedge i \in \mathbb{Z} \wedge \wedge i \in 1..n{-}1 \wedge \begin{pmatrix} m \in \mathbb{N} \wedge \\ m \in ran(f[0..i]) \wedge \\ (\forall j \cdot j \in 0 \mathinner{..} i \Rightarrow f(j) \leq m) \end{pmatrix} \}$$

$i{+}{+};$

$$\ell_7 : \{ \begin{pmatrix} n \in \mathbb{N} \wedge \\ n \neq 0 \wedge \\ f \in 0 \mathinner{..} n{-}1 \rightarrow \mathbb{N} \end{pmatrix} \wedge i \in 1..n{-}1 \wedge \begin{pmatrix} m \in \mathbb{N} \wedge \\ m \in ran(f[0..i{-}1]) \wedge \\ (\forall j \cdot j \in 0 \mathinner{..} i{-}1 \Rightarrow f(j) \leq m) \end{pmatrix} \}$$

;

$$\ell_8 : \{ \begin{pmatrix} n \in \mathbb{N} \wedge \\ n \neq 0 \wedge \\ f \in 0 \mathinner{..} n{-}1 \rightarrow \mathbb{N} \end{pmatrix} \wedge i = n \wedge \begin{pmatrix} m \in \mathbb{N} \wedge \\ m \in ran(f) \wedge \\ (\forall j \cdot j \in 0 \mathinner{..} n{-}1 \Rightarrow f(j) \leq m) \end{pmatrix} \}$$

Algorithme 5: Algorithme du maximum d'une liste annoté

inv6 : $l = l1 \Rightarrow m = f(0)$
inv7 : $l = l2 \Rightarrow i = 1 \wedge m = f(0) \wedge i \leq n \wedge 0 \,..\, i-1 \subseteq dom(f) \wedge (\forall j \cdot j \in 0 \,..\, i-1 \Rightarrow f(j) \leq m) \wedge m \in ran(f)$
inv8 : $l = l3 \Rightarrow i < n \wedge 0 \,..\, i \subseteq dom(f) \wedge (\forall j \cdot j \in 0 \,..\, i-1 \Rightarrow f(j) \leq m) \wedge m \in ran(f)$
inv9 : $l = l4 \Rightarrow i < n \wedge 0 \,..\, i \subseteq dom(f) \wedge (\forall j \cdot j \in 0 \,..\, i-1 \Rightarrow f(j) \leq m) \wedge f(i) > m \wedge m \in ran(f)$
inv10 : $l = l5 \Rightarrow i < n \wedge 0 \,..\, i \subseteq dom(f) \wedge (\forall j \cdot j \in 0 \,..\, i-1 \Rightarrow f(j) \leq m) \wedge (\forall j \cdot j \in 0 \,..\, i \Rightarrow f(j) \leq m) \wedge m \in ran(f)$
inv11 : $l = l6 \Rightarrow i < n \wedge 0 \,..\, i \subseteq dom(f) \wedge (\forall j \cdot j \in 0 \,..\, i \Rightarrow f(j) \leq m) \wedge m \in ran(f)$
inv12 : $l = l7 \Rightarrow i \leq n \wedge 0 \,..\, i-1 \subseteq dom(f) \wedge (\forall j \cdot j \in 0 \,..\, i-1 \Rightarrow f(j) \leq m) \wedge m \in ran(f)$
inv13 : $l = l8 \Rightarrow i = n \wedge dom(f) \subseteq 0 \,..\, i-1 \wedge (\forall j \cdot j \in 0 \,..\, i-1 \Rightarrow f(j) \leq m) \wedge m \in ran(f)$
post : $l = l8 \Rightarrow (\forall j \cdot j \in 0 \,..\, n-1 \Rightarrow f(j) \leq m) \wedge m \in ran(f)$
pre : $f \in 0 \,..\, n-1 \rightarrow \mathbb{N} \wedge i \in 0 \,..\, n \wedge m \in \mathbb{N} \Rightarrow m \in \mathbb{N} \wedge i \in \mathbb{N}$

BEGIN

act5 : $l := l0$
act6 : $m :\in \mathbb{N}$
act7 : $i :\in 0 \,..\, n$

END

EVENT al0l1
WHEN

grd1 : $l = l0$

THEN

act4 : $l := l1$
act5 : $m :|(m' = f(0))$

END

EVENT al1l2
WHEN

grd1 : $l = l1$

THEN

act1 : $l := l2$
act2 : $i := 1$

END

EVENT al2l3
WHEN

grd1 : $l = l2$
grd2 : $i < n$

THEN

act1 : $l := l3$

END

EVENT al2l8
WHEN

grd1 : $l = l2$
grd2 : $i \geq n$

THEN

act1 : $l := l8$

END

EVENT am3l4
WHEN

grd1 : $l = l3$
grd2 : $f(i) > m$

THEN

act1 : $l := l4$

END

EVENT el3l6
WHEN

grd1 : $l = l3$
grd2 : $f(i) \leq m$

THEN

act1 : $l := l6$

END

EVENT al4l5
WHEN

grd1 : $l = l4$

THEN

act1 : $l := l5$
act2 : $m := f(i)$

END

EVENT el5l6
WHEN

grd1 : $l = l5$

THEN

act1 : $l := l6$

END

EVENT al6l7
WHEN

grd1 : $l = l6$

THEN

act1 : $l := l7$
act2 : $i := i{+}1$

END

EVENT el7l3
WHEN

grd1 : $l = l7$
grd2 : $i < n$

THEN

act1 : $l := l3$

END

EVENT el3l8
WHEN

grd1 : $l = l3$
grd2 : $i \geq n$

THEN

act1 : $l := l8$

END

Exercice 6 *(Annotation du calcul de la racine carrée entière appex5_6.tla)*
L'algorithme annoté **??** *calcule la racine carrée entière d'un nombre entrier. Vérifier les annotations par un mdodèle Event-B.*

variables X, Y_1, Y_2, Y_3, Z
requires
$x0 \in \mathbb{N}$
$y10 \in Int$
$y20 \in Int$
$y30 \in Int$
$z0 \in Int$
ensures
$zf \cdot zf \leq x < (zf+1) \cdot (zf+1)$
$xf = x0$
$zf = y1f$
$y2f = y1f+1$
$y3f = 2 \cdot y1f+1$
begin
$\ell_0 : \{x \in \mathbb{N} \wedge z \in \mathbb{Z} \wedge y1 \in \mathbb{Z} \wedge y2 \in \mathbb{Z} \wedge y3 \in \mathbb{Z}\}$
$(Y_1, Y_2, Y_3) := (0, 1, 1);$
$\ell_1 : \{y2 = (y1+1) \cdot (y1+1) \wedge y3 = 2 \cdot y1+1 \wedge y1 \cdot y1 \leq x\}$
While $(Y_2 \leq X)$
$\ell_2 : \{y2 = (y1+1) \cdot (y1+1) \wedge y3 = 2 \cdot y1+1 \wedge y2 \leq x\}$
$(Y_1, Y_2, Y_3) := (Y_1+1, Y_2+Y_3+2, Y_3+2);$
$\ell_3 : \{y2 = (y1+1) \cdot (y1+1) \wedge y3 = 2 \cdot y1+1 \wedge y1 \cdot y1 \leq x\}$
od;
$\ell_4 : \{y2 = (y1+1) \cdot (y1+1) \wedge y3 = 2 \cdot y1+1 \wedge y1 \cdot y1 \leq x \wedge x < y2\}$
$Z := Y_1;$
$\ell_5 : \{y2 = (y1+1) \cdot (y1+1) \wedge y3 = 2 \cdot y1+1 \wedge y1 \cdot y1 \leq x \wedge x < y2 \wedge z = y1 \wedge z \cdot z \leq x \wedge x < (z+1) \cdot (z+1)\}$
end

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

\https://filesender.renater.fr/?s=download&token=ecd2b8be-6e28-4d35-979e-7380ee97a1

Exercice 7 *Soit le petit programme suivant*

Listing 4 – exercice fex2.c

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

  //@ 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 8 *Soit le petit programme suivant*

Listing 5 – exercice fex1.c

```
vvoid 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 9 *Soit le petit programme suivant*

Listing 6 – exercice fex1.c

```
#include <limits.h>
// returns the maximum of x and y
/*@
    ensures \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 10 *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
$\mathrm{S}_1;\mathrm{S}_2$	$wp(\mathrm{S}_1)(wp(\mathrm{S}_2)(P))$
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* $\{P\}\boldsymbol{S}_1\{R\}$ *et* $\{R\}\boldsymbol{S}_2\{Q\}$*, alors* $\{P\}\boldsymbol{S}_1;\boldsymbol{S}_2\{Q\}$.
— *Si* $\{P \wedge B\}\boldsymbol{S}_1\{Q\}$ *et* $\{P \wedge \neg B\}\boldsymbol{S}_2\{Q\}$*, alors* $\{P\}$***if B then*** $\boldsymbol{S}_1$ ***then*** $\boldsymbol{S}_2$ ***fi***$\{Q\}$.
— *Si* $\{P \wedge B\}\boldsymbol{S}\{P\}$*, alors* $\{P\}$***while B do S od***$\{P \wedge \neg B\}$.
— *Règle de renforcement/affaiblissement : Si* $P' \Rightarrow P$, $\{P\}\boldsymbol{S}\{Q\}$, $Q \Rightarrow Q'$*, alors* $\{P'\}\boldsymbol{S}\{Q'\}$.

Question 10.1 *Simplifier les expressions suivantes :*

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

Question 10.2 *On rappelle que* $\{P\}\boldsymbol{S}\{Q\}$ *est défini par l'implication* $O \Rightarrow WP(S)(Q)$*. Pour chaque point énuméré ci-dessous, monter que la propriété* $\{P\}\boldsymbol{S}\{Q\}$ *est valide ou pas en utilisant la définition suivante :*

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

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 10.3 *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 11 *fex5.c*
Soit le petit programme suivant dans un fichier fex5..c

```c
void swap1(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 == b && y == a;
}
```

Question 11.1 *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 11.2 *Cliquer à droite sur le mot-clé assert et cliqur sur Prove annotation by WP. Les boutons deviennent vert.*

Question 11.3

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

◇– **Solution de la question 11.4** ______________________________

```
/*@ 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 11.4

Question 11.5 *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 12 *Etudier la correction de l'algorithme suivant en complétant l'invariant de boucle :*

```
    /*@
  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;
}
```

◇— **Solution de l'exercice 12**

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

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

Fin 12

Exercice 13

*On rappelle que l'annotation suivante du listing **??** 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 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 13.1

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

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

Question 13.2 *Soient trois constantes n,m,p*

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

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 14

```
#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 pariellement 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 experimentations sur les valeurs calculées.*

Question 14.1 *Compléter le fichier* `qpower2.c` *et produire le fichier* `power2.c` *qui est vérifié avec fraama-c.*

Question 14.2 *Simplifier la fonction itérative en supprimant les variables commençant par la lettre* `o`*. Puis vérifier les fonctions obtenues avec frama-c.*

Question 14.3 *En fait, vous avez montré que les deux fonctions étaient équivalentes. Expliquez pourquoi en quelques lignes.*

Cours Modélisation et vérification des systèmes informatiques Exercices (avec les corrections) Utilisation d'un environnement de vérification Frama-c (II) par Dominique Méry 17 novembre 2024

Exercice 15

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

Question 15.2 *Traduire ce contrat en ACSL et vï¾ï½©rifier qu'il est valide ou non. S'il est non valide, proposer une correction de la prï¾ï½©-condition et/ou de la postcondition.*

Exercice 16 *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 16** ________________________________

Listing 8 – schema de contrat

```
// frama-c-gui -wp -wp-rte  -report -wp-print  maxpointer.c

/*@ requires \valid(p)  && \valid(q);
    ensures \result >= *p && \result >= *q;
    ensures \result == *p || \result == *q;
*/
int max_ptr ( int *p, int *q ) {
if ( *p >= *q ) return *p ;
return *q ; }
```

Fin 16

Exercice 17 *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 17**

Listing 9 – schema de contrat

```
#include <limits.h>

/*@ requires x > INT_MIN;
    assigns \nothing;
    behavior pos:
      assumes x >= 0;
      ensures \result == x;
    behavior neg:
      assumes x < 0;
      ensures \result == â x;
     complete behaviors;
     disjoint behaviors;
*/
int abs (int x) {
  if (x >= 0) return x ;
  return -x;}
```

Fin 17

Exercice 18 *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 18**

```
nt 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);
}
```

```
#include <limits.h>
/*@ requires x > INT_MIN;
    ensures (x >= 0 ==> \result == x) && (x < 0 ==> \result ==âx);
    assigns \nothing ;
*/
int abs ( int x );
/*@ ensures \result >= x && \result >= y;
    ensures \result == x || \result == y;
    assigns \nothing ;
*/
int max ( int x, int y );
/*@ ensures \result >= x && \result >= âx && \result >= y && \result >= ây;
    ensures \result == x || \result == âx || \result==y || \result == ây;
    assigns \nothing ;
*/
// 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);
}
```

```
#include <limits.h>
/*@ requires x > INT_MIN;
    ensures (x >= 0 ==> \result == x) && (x < 0 ==> \result ==âx);
    assigns \nothing ;
*/

int abs ( int x );
/*@ ensures \result >= x  && \result >= y;
    ensures \result == x || \result == y;
    assigns \nothing ;
*/
int max ( int x, int y );
/*@ requires x > INT_MIN;requires y > INT_MIN;
    ensures \result >= x && \result >= âx && \result >= y && \result >= ây;
    ensures \result == x || \result == âx || \result==y || \result==ây;
    assigns \nothing ;
*/
// 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);
}
```

Fin 18

Exercice 19 Question 19.1 *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 19.1**

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

Fin 19.1

Question 19.2 *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 19.2**

```
/*@ 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);
  @ } */

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

Fin 19.2

Question 19.3 *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 19.3**

```
/*@  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;
```

```
}

/*@
  requires n > 0;
  requires \valid(t+(0..n-1));
  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;
  */
  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));
  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;
  */
  for (int i = 1; i < n; i++)
    if (t[i] > r) r = t[i];
  return r;
}
```

Fin 19.3

La solution sous la forme d'un fichier c est la suyivante :

Listing 10 – schema de contrat

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

return *r ;*
}

Exercice 20 *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 20.1

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

◇– **Solution de la question 20.1**

```
frama-c  -wp -rte -print  hoare1.c
[kernel] Parsing hoare1.c (with preprocessing)
[rte] annotating function q1
[wp] 4 goals scheduled
[wp] Proved goals:    4 / 4
    Qed:             4
/* Generated by Frama-C */
int q1(void)
{
  int __retres;
  int x = 10;
  int y = 30;
  int z = 20;
  /*@ assert x ç«ï½¡ 10 ç«ï½§ y ç«ï½¡ z + x ç«ï½§ z ç«ï½¡ 2 * x; */ ;
  /*@ assert rte: signed_overflow: -2147483648 ç«ï½¤ z + x; */
  /*@ assert rte: signed_overflow: z + x ç«ï½¤ 2147483647; */
  y = z + x;
  /*@ assert x ç«ï½¡ 10 ç«ï½§ y ç«ï½¡ x + 2 * 10; */ ;
  __retres = 0;
  return __retres;
}
```

Fin 20.1

Question 20.2

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

Question 20.3

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

Exercice 21 *(6 points)*
Evaluer la validité de chaque annotation dans les questions suivent.

Question 21.1

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

Question 21.2

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

Question 21.3

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

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

Question 21.5 *Etudier la validité de cette condition de vérification selon la valeur de m.*

Exercice 22 *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 $\begin{pmatrix} n_0 \in \mathbb{N} \wedge n_0 \neq 0 \\ v_0 \in 0..n_0{-}1 \longrightarrow \mathbb{Z} \end{pmatrix}$

ENSURES $\begin{pmatrix} s_f = \bigcup_{k=0}^{n_0-1} v_0(k) \\ n_f = n_0 \\ v_f = v_0 \end{pmatrix}$

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

$S := V(0)$

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

$I := 1$

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

WHILE $I < N$ *DO*

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

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

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

$I := I{+}1$

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

OD;

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

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 22.1 *Ecrire une solution contractuelle de cet algorithme.*

Question 22.2 *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 22.3 *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 23 *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 23.1 *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 24 *Soit le petit programme suivant :*

Listing 11 – contrat91

```
#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 25 *Soit le petit programme suivant :*

Listing 12 – contrat91

```
#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 ;