

Cours Modélisation et vérification des systèmes informatiques
 Exercices (avec les corrections)
 Modélisation d'algorithmes en PlusCal (I)
 par Dominique Méry
 9 octobre 2024

Exercice 1 

*Nous allons utiliser la fonctionnalité de traduction d'un algorithme PlusCal en un module TLA pour vérifier des algorithmes. Pour chaque question, on écrira un module TLA contenant une expression d'*e* l'algorithme puis on traduira par un module et ensuite on analysera le module obtenu par rapport à la correction partielle et l'absence d'erreurs à l'exécution.*

Question 1.1

```

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

```

Question 1.2 *On suppose que p est un nombre premier :*

```

 $\ell_1 : x = 2^p \wedge y = 2^{p+1} \wedge x \cdot y = 2^{2 \cdot p + 1}$ 
 $x := y + x + 2^x$ 
 $\ell_2 : x = 5 \cdot 2^p \wedge y = 2^{p+1}$ 

```

Question 1.3

```

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

```

Question 1.4

```

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

```

Question 1.5 *Dans cette question, on considère l'algorithme correspondant au calcul du maximum de deux nombres.*

Exercice 2 

On considère l'algorithme suivant :

Requires : $x_0, y_0 \in \mathbb{N}$
Ensures : $z_f = \max(x_0, y_0)$

$\ell_0 : \{\dots\}$

if $X < Y$ **then**

$\ell_1 : \{\dots\}$

$Z := Y;$

$\ell_2 : \{\dots\}$

else

$\ell_3 : \{\dots\}$

$Z := X;$

$\ell_4 : \{\dots\}$

;

$\ell_5 : \{\dots\}$

Algorithme 1: MAX annotée

?) **START**

$$\{x_1 \geq 0 \wedge x_2 > 0\}$$

$$(y_1, y_2, y_3) \leftarrow (x_1, 0, x_2);$$

while $y_3 \leq y_1$ **do** $y_3 \leftarrow 2y_3;$

while $y_3 \neq x_2$ **do**

begin $(y_2, y_3) \leftarrow (2y_2, y_3/2);$

end; **if** $y_3 \leq y_1$ **do** $(y_1, y_2) \leftarrow (y_1 - y_3, y_2 + 1)$

$$(z_1, z_2) \leftarrow (y_1, y_2)$$

$$\{0 \leqq z_1 < x_2 \wedge x_1 = z_2 x_2 + z_1\}$$

HALT

Question 2.1 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 lanage PlusCal.

Question 2.2 Montrer qu'il est sans erreur à l'exécution.

Exercice 3 \square

On considère l'algorithme suivant :

START

$\{x_1 > 0 \wedge x_2 > 0\}$

$(y_1, y_2, y_3, y_4) \leftarrow (x_1, x_2, x_2, 0);$

while $y_1 \neq y_2$ **do**

(1)

begin while $y_1 > y_2$ **do** $(y_1, y_4) \leftarrow (y_1 - y_2, y_4 + y_3);$

end; **while** $y_2 > y_1$ **do** $(y_2, y_3) \leftarrow (y_2 - y_1, y_3 + y_4)$

$(z_1, z_2) \leftarrow (y_1, y_3 + y_4)$

$\{z_1 = gcd(x_1, x_2) \wedge z_2 = scm(x_1, x_2)\}$

HALT

where $scm(x_1, x_2)$ is the smallest common multiple of x_1 and x_2 ; that is, it is their product divided by the greatest common divisor.

Question 3.1 Montrer que cet algorithme est apertiellement 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.

◊ Solution de l'exercice 3

MODULE pluscal4

```
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 ≤ y1){
    y3 := 2·y3;
};
l3 : while (y3 ≠ x2){
    y2 := 2·y2;
    y3 := y3 ÷ 2;
    l4 : if (y3 ≤ y1) {
        y1 := y1 - y3;
        y2 := y2 + 1;
    };
};
```

```

};

l5 : z1 := y1;
z2 := y2;
l6 : print ⟨x1, x2, z1, z2⟩;
}

}

.)

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

vars  $\triangleq$  ⟨y1, y2, y3, z1, z2, pc⟩

Init  $\triangleq$  Global variables
 $\wedge$  y1 = defaultInitValue
 $\wedge$  y2 = defaultInitValue
 $\wedge$  y3 = defaultInitValue
 $\wedge$  z1 = defaultInitValue
 $\wedge$  z2 = defaultInitValue
 $\wedge$  pc = "I1"

l1  $\triangleq$   $\wedge$  pc = "I1"
 $\wedge$  y'1 = x1
 $\wedge$  y'2 = 0
 $\wedge$  y'3 = x2
 $\wedge$  pc' = "I2"
 $\wedge$  UNCHANGED ⟨z1, z2⟩

l2  $\triangleq$   $\wedge$  pc = "I2"
 $\wedge$  IF y3  $\leq$  y1
    THEN  $\wedge$  y'3 = 2·y3
         $\wedge$  pc' = "I2"
    ELSE  $\wedge$  pc' = "I3"
         $\wedge$  y'3 = y3
     $\wedge$  UNCHANGED ⟨y1, y2, z1, z2⟩

l3  $\triangleq$   $\wedge$  pc = "I3"
 $\wedge$  IF y3  $\neq$  x2
    THEN  $\wedge$  y'2 = 2·y2
         $\wedge$  y'3 = (y3  $\div$  2)
         $\wedge$  pc' = "I4"
    ELSE  $\wedge$  pc' = "I5"
         $\wedge$  UNCHANGED ⟨y2, y3⟩
     $\wedge$  UNCHANGED ⟨y1, z1, z2⟩

l4  $\triangleq$   $\wedge$  pc = "I4"
 $\wedge$  IF y3  $\leq$  y1
    THEN  $\wedge$  y'1 = y1 - y3
         $\wedge$  y'2 = y2 + 1
    ELSE  $\wedge$  TRUE
         $\wedge$  UNCHANGED ⟨y1, y2⟩
     $\wedge$  pc' = "I3"
     $\wedge$  UNCHANGED ⟨y3, z1, z2⟩

l5  $\triangleq$   $\wedge$  pc = "I5"

```

$$\begin{aligned}
& \wedge z'_1 = y1 \\
& \wedge z'_2 = y2 \\
& \wedge pc' = "l6" \\
& \wedge \text{UNCHANGED } \langle y_1, y_2, y_3 \rangle \\
l6 & \triangleq \wedge pc = "l6" \\
& \wedge \text{PrintT}(\langle x_1, x_2, z_1, z_2 \rangle) \\
& \wedge pc' = "Done" \\
& \wedge \text{UNCHANGED } \langle y_1, y_2, y_3, z_1, z_2 \rangle
\end{aligned}$$

$$\begin{aligned}
Next & \triangleq l_1 \vee l_2 \vee l_3 \vee l_4 \vee l_5 \vee l6 \\
& \vee \quad \text{Disjunct to prevent deadlock on termination} \\
& \quad (pc = "Done" \wedge \text{UNCHANGED } vars)
\end{aligned}$$

$$Spec \triangleq Init \wedge \square[Next]_{vars}$$

$$Termination \triangleq \diamondsuit(pc = "Done")$$

END TRANSLATION

$$Iloop(u, v) \triangleq x_1 = v \cdot x_2 + u$$

$$Qpc \triangleq pc = "Done" \Rightarrow x_1 = z_2 \cdot x_2 + z_1 \wedge 0 \leq z_1 \wedge z_1 < x_2$$

$$COND(U) \triangleq min \leq U \wedge U \leq max$$

$$Qef \triangleq min \leq y_1 \wedge y_1 \leq max$$

$$i \triangleq Iloop(y_1, y_2)$$

Modification History

Last modified Tue Oct 04 08:15:08 CEST 2016 by mery

Created Wed Nov 18 16:33:27 CET 2015 by mery

Fin 3

Exercice 4

Nous allons utiliser la fonctionnalité de traduction d'un algorithme PlusCal en un module TLA pour vérifier des algorithmes. Pour chaque question, on écrira un module TLA contenant une expression de l'algorithme puis on traduira par un module et ensuite on analysera le module obtenu par rapport à la correction partielle et l'absence d'erreurs à l'exécution.

Question 4.1 Soit l'annotation suivante :

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

Traduire en PlusCal.

← Solution de la question 4.1

MODULE tp4

EXTENDS TLC, Integers, Naturals

(·

```

--algorithm ex1 {
variables x,y,z;
{
init : x := 10; z := 2·x; y := z+x;
l1 : y := z+x;
print ⟨x,y,z⟩;
}

}
·)
i  $\triangleq$ 
 $\wedge pc = "I1" \Rightarrow x = 10 \wedge y = z+x \wedge z = 2 \cdot x$ 
 $\wedge pc = "Done" \Rightarrow x = 10 \wedge y = x+2 \cdot 10$ 

```

Fin 4.1

Exercice 5 On considère l'algorithme suivant :

```

START
{x1 > 0  $\wedge$  x2 > 0}
(y1, y2, y3, y4)  $\leftarrow$  (x1, x2, x2, 0);
while y1  $\neq$  y2 do
    begin while y1 > y2 do (y1, y4)  $\leftarrow$  (y1 - y2, y4 + y3);
        while y2 > y1 do (y2, y3)  $\leftarrow$  (y2 - y1, y3 + y4)
    end;
    (z1, z2)  $\leftarrow$  (y1, y3 + y4)
    {z1 = gcd(x1, x2)  $\wedge$  z2 = scm(x1, x2)}
HALT

```

where scm(x₁, x₂) is the smallest common multiple of x₁ and x₂; that is, it is their product divided by the greatest common divisor.

Question 5.1 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 5.2 Montrer qu'il est sans erreur à l'exécution.

Exercice 6 Exponentiation en PlusCal

Ecrire l'algorithme de l'exponentiation avec PlusCal.

On suppose que x₁ et x₂ sont des constantes.

← Solution de l'exercice 6 →

MODULE tp3

EXTENDS Naturals, Integers, TLC

CONSTANT MAXINT, u, v

```

(·
-termination
-wfNext
--algorithm Power {
variables x1 = u;
    1st integer
}
```

```

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 :  $y_1, y_2, y_3 \in \mathbb{Z}$ 

 $\ell_0 : \{y_1, y_2, y_3, z \in \mathbb{Z}\}$ 
 $(y_1, y_2, y_3) := (x_1, x_2, 1);$ 
 $\ell_1 : \{y_3 \cdot y_1^{y_2} = x_1^{x_2} \wedge y_1, y_2, y_3 \in \mathbb{N} \wedge z \in \mathbb{Z}\}$ 
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
         $\ell_3 : \{impair(y_2) \wedge y_2 \neq 0 \wedge \boxed{\dots} \wedge y_1, y_2, y_3 \in \mathbb{N} \wedge z \in \mathbb{Z}\}$ 
         $y_2 := y_2 - 1;$ 
         $\ell_4 : \{y_2 \geq 0 \wedge pair(y_2) \wedge y_3 \cdot y_1 \cdot y_1^{y_2} = x_1^{x_2} \wedge y_1, y_2, y_3 \in \mathbb{N} \wedge z \in \mathbb{Z}\}$ 
         $y_3 := y_3 \cdot y_1;$ 
         $\ell_5 : \{y_2 \geq 0 \wedge pair(y_2) \wedge y_3 \cdot y_1^{y_2} = x_1^{x_2} \wedge y_1, y_2, y_3 \in \mathbb{N} \wedge z \in \mathbb{Z}\}$ 
    ;
     $\ell_6 : \{y_2 \geq 0 \wedge pair(y_2) \wedge \boxed{\dots} \wedge y_1, y_2, y_3 \in \mathbb{N} \wedge z \in \mathbb{Z}\}$ 
     $y_1 := y_1 \cdot y_1;$ 
     $\ell_7 : \{y_2 \geq 0 \wedge pair(y_2) \wedge y_3 \cdot y_1^{y_2 \text{ div } 2} = x_1^{x_2} \wedge y_1, y_2, y_3 \in \mathbb{N} \wedge z \in \mathbb{Z}\}$ 
     $y_2 := y_2 \text{ div } 2;$ 
     $\ell_8 : \{y_2 \geq 0 \wedge \boxed{\dots} \wedge y_1, y_2, y_3 \in \mathbb{N} \wedge z \in \mathbb{Z}\}$ 
;
 $\ell_9 : \{y_2 = 0 \wedge \boxed{\dots} \wedge y_1, y_2, y_3 \in \mathbb{N} \wedge z \in \mathbb{Z}\}$ 
 $z := y_3;$ 
 $\ell_{10} : \{y_2 = 0 \wedge y_3 \cdot y_1^{y_2} = x_1^{x_2} \wedge y_1, y_2, y_3 \in \mathbb{N} \wedge z \in \mathbb{Z} \wedge z = x_1^{x_2}\}$ 

```

Algorithm 2: Algorithme de l'exponentiation indienne annoté

```

x2 = v;    2nd integer
y1 = 0;
y2 = 0;
y3 = 0;
z = 0;

{
l1 : print ⟨x1, x21;
y2 := x2;
y3 := 1;
l2 : while (y2 /= 0) {

L3 : if ( y2 % 2 != 0) {
    l4 : y2 := y2-1;
    l5 : y3 := y3·y1;
};

l6 : y1 := y1·y1;
l7 : y2 := y2 ÷ 2;
};

l8 : z := y3;
l9 : print ⟨x1, x2, z⟩;
}

}
.)
```

Modification History

Last modified Fri Feb 26 06:26:27 CET 2016 by mery

Created Wed Sep 09 17:02:47 CEST 2015 by mery

La traduction par l'outil donne alors le module suivant :

MODULE tp3

```

EXTENDS Naturals, Integers, TLC

CONSTANT MAXINT, u, v

(·
-termination
-wfNext
--algorithm Power {
    variables x1 = u;    1st integer
        x2 = v;    2nd integer
        y1 = 0;
        y2 = 0;
        y3 = 0;
        z = 0;

    {
        l1 : print ⟨x1, x2⟩;
```

```

y1 := x1;
y2 := x2;
y3 := 1;
l2 : while (y2 / = 0) {
    L3 : if (y2 % 2 ≠ 0) {
        l4 : y2 := y2 - 1;
        l5 : y3 := y3 · y1;
    };
    l6 : y1 := y1 · y1;
    l7 : y2 := y2 ÷ 2;
};
l8 : z := y3;
l9 : print ⟨x1, x2, z⟩;
}
.
```

BEGIN TRANSLATION

VARIABLES $x_1, x_2, y_1, y_2, y_3, z, pc$

$vars \triangleq \langle x_1, x_2, y_1, y_2, y_3, z, pc \rangle$

$Init \triangleq$ Global variables

$$\begin{aligned} & \wedge x_1 = u \\ & \wedge x_2 = v \\ & \wedge y_1 = 0 \\ & \wedge y_2 = 0 \\ & \wedge y_3 = 0 \\ & \wedge z = 0 \\ & \wedge pc = "I1" \end{aligned}$$

$$\begin{aligned} l1 \triangleq & \wedge pc = "I1" \\ & \wedge PrintT(\langle x_1, x_2 \rangle) \\ & \wedge y'_1 = x_1 \\ & \wedge y'_2 = x_2 \\ & \wedge y'_3 = 1 \\ & \wedge pc' = "I2" \\ & \wedge \text{UNCHANGED } \langle x_1, x_2, z \rangle \end{aligned}$$

$$\begin{aligned} l2 \triangleq & \wedge pc = "I2" \\ & \wedge \text{IF } y_2 / = 0 \\ & \quad \text{THEN } \wedge pc' = "I3" \\ & \quad \text{ELSE } \wedge pc' = "I8" \\ & \wedge \text{UNCHANGED } \langle x_1, x_2, y_1, y_2, y_3, z \rangle \end{aligned}$$

$$\begin{aligned} L3 \triangleq & \wedge pc = "I3" \\ & \wedge \text{IF } y_2 \% 2 \neq 0 \\ & \quad \text{THEN } \wedge pc' = "I4" \\ & \quad \text{ELSE } \wedge pc' = "I6" \\ & \wedge \text{UNCHANGED } \langle x_1, x_2, y_1, y_2, y_3, z \rangle \end{aligned}$$

$$\begin{aligned} l4 \triangleq & \wedge pc = "I4" \\ & \wedge y'_2 = y_2 - 1 \\ & \wedge pc' = "I5" \\ & \wedge \text{UNCHANGED } \langle x_1, x_2, y_1, y_3, z \rangle \end{aligned}$$

```

l5  $\triangleq$   $\wedge pc = "l5"$ 
     $\wedge y'_3 = y_3 \cdot y_1$ 
     $\wedge pc' = "l6"$ 
     $\wedge \text{UNCHANGED } \langle x_1, x_2, y_1, y_2, z \rangle$ 

l6  $\triangleq$   $\wedge pc = "l6"$ 
     $\wedge y'_1 = y_1 \cdot y_1$ 
     $\wedge pc' = "l7"$ 
     $\wedge \text{UNCHANGED } \langle x_1, x_2, y_2, y_3, z \rangle$ 

l7  $\triangleq$   $\wedge pc = "l7"$ 
     $\wedge y'_2 = (y_2 \div 2)$ 
     $\wedge pc' = "l2"$ 
     $\wedge \text{UNCHANGED } \langle x_1, x_2, y_1, y_3, z \rangle$ 

l8  $\triangleq$   $\wedge pc = "l8"$ 
     $\wedge z' = y_3$ 
     $\wedge pc' = "l9"$ 
     $\wedge \text{UNCHANGED } \langle x_1, x_2, y_1, y_2, y_3 \rangle$ 

l9  $\triangleq$   $\wedge pc = "l9"$ 
     $\wedge \text{PrintT}(\langle x_1, x_2, z \rangle)$ 
     $\wedge pc' = \text{"Done"}$ 
     $\wedge \text{UNCHANGED } \langle x_1, x_2, y_1, y_2, y_3, z \rangle$ 

```

Next \triangleq $l_1 \vee l_2 \vee L_3 \vee l_4 \vee l_5 \vee l_6 \vee l_7 \vee l_8 \vee l_9$

\vee Disjunct to prevent deadlock on termination

$(pc = \text{"Done"} \wedge \text{UNCHANGED } vars)$

Spec \triangleq *Init* $\wedge \square[Next]_{vars}$

Termination \triangleq $\diamondsuit(pc = \text{"Done"})$

END TRANSLATION

Modification History

Last modified Fri Feb 26 06:26:27 CET 2016 by mery

Created Wed Sep 09 17:02:47 CEST 2015 by mery

Fin 6

Exercice 7 Vérifier le programme C suivant en la traduisant dans un module PlusCal. Pour cela, on conservera les éléments algorithmiques suffisants pour analyser le programme C. Par exemple, on ne traduira pas les instructions de lecture ou les inclusions de bibliothèques.

```

/* C Program to find roots of a quadratic equation when coefficients are entered by
/* Library function sqrt() computes the square root. */

#include <stdio.h>
#include <math.h> /* This is needed to use sqrt() function.*/
int main()

```

```

{
    float a, b, c, determinant, r1, r2, real, imag;
    printf("Enter coefficients a, b and c: ");
    scanf("%f%f%f", &a, &b, &c);
    determinant = b * b - 4 * a * c;
    if (determinant > 0)
    {
        r1 = (-b + sqrt(determinant)) / (2 * a);
        r2 = (-b - sqrt(determinant)) / (2 * a);
        printf("Roots are: %.2f and %.2f", r1, r2);
    }
    else if (determinant == 0)
    {
        r1 = r2 = -b / (2 * a);
        printf("Roots are: %.2f and %.2f", r1, r2);
    }
    else
    {
        real = -b / (2 * a);
        imag = sqrt(-determinant) / (2 * a);
        printf("Roots are: %.2f + %.2fi and %.2f - %.2fi", real, imag, real, imag);
    }
    return 0;
}

```

Output 1

```

Enter coefficients a, b and c: 2.3
4
5.6
Roots are: -0.87+1.30i and -0.87-1.30i

```

Output 2

```

Enter coefficients a, b and c: 4
1
0
Roots are: 0.00 and -0.25

```

To solve this program, library function `sqrt()` is used. This function calculates the `sqrt()` function.

Similar C Programming Examples

[C Program to Add Two Complex Numbers by Passing Structure to a Function](#)

[C Program to Find Quotient and Remainder of Two Integers Entered by User](#)

[C Program to Check Prime or Armstrong Number Using User-defined Function](#)

[C Program to Display Prime Numbers Between Intervals Using User-defined Function](#)

[C Program to Display its own Source Code as Output](#)

← **Solution de l'exercice 7** →

MODULE *tp7*

EXTENDS *TLC, Integers, Naturals, Reals*

CONSTANTS *a₀, b₀, c₀*

```

(.)  

--algorithm resolution {
variables a = a0, b = b0, c = c0, determinant;
{  

l0 : determinant := b·b-4·a·c;  

if (determinant > 0)  

{  

    l1 : print ⟨"Two Roots are : "⟩;  

}  

else if (determinant = 0)  

{  

    l2 : print ⟨"One Double roots are : "⟩;  

}  

else  

{  

    l3 : print ⟨"Two complex Roots are : "⟩;  

}  

}  

}  

}  

.  

)
BEGIN TRANSLATION
CONSTANT defaultInitValue
VARIABLES a, b, c, determinant, pc

```

vars \triangleq ⟨ *a*, *b*, *c*, *determinant*, *pc* ⟩

Init \triangleq Global variables
 $\wedge a = a_0$
 $\wedge b = b_0$
 $\wedge c = c_0$
 $\wedge \text{determinant} = \text{defaultInitValue}$
 $\wedge \text{pc} = \text{"IO"}$

l0 \triangleq $\wedge \text{pc} = \text{"IO"}$
 $\wedge \text{determinant}' = b \cdot b - 4 \cdot a \cdot c$
 $\wedge \text{IF } \text{determinant}' > 0$
 THEN $\wedge \text{pc}' = \text{"I1"}$
 ELSE $\wedge \text{IF } \text{determinant}' = 0$
 THEN $\wedge \text{pc}' = \text{"I2"}$
 ELSE $\wedge \text{pc}' = \text{"I3"}$
 $\wedge \text{UNCHANGED } \langle a, b, c \rangle$

l1 \triangleq $\wedge \text{pc} = \text{"I1"}$
 $\wedge \text{PrintT}(\langle \text{"Two Roots are : "} \rangle)$
 $\wedge \text{pc}' = \text{"Done"}$
 $\wedge \text{UNCHANGED } \langle a, b, c, \text{determinant} \rangle$

l2 \triangleq $\wedge \text{pc} = \text{"I2"}$
 $\wedge \text{PrintT}(\langle \text{"One Double roots are : "} \rangle)$
 $\wedge \text{pc}' = \text{"Done"}$
 $\wedge \text{UNCHANGED } \langle a, b, c, \text{determinant} \rangle$

```

l3  $\triangleq$   $\wedge pc = "l3"$ 
 $\wedge PrintT(\langle "Two complex Roots are : " \rangle)$ 
 $\wedge pc' = "Done"$ 
 $\wedge \text{UNCHANGED } \langle a, b, c, \text{ determinant} \rangle$ 

```

```

Next  $\triangleq$   $l_0 \vee l_1 \vee l_2 \vee l3$ 
 $\vee \quad \text{Disjunct to prevent deadlock on termination}$ 
 $(pc = "Done" \wedge \text{UNCHANGED } vars)$ 

```

Spec \triangleq *Init* $\wedge \square[Next]_{vars}$

Termination \triangleq $\diamond(pc = "Done")$

END TRANSLATION

Modification History

Last modified Mon Nov 23 11:14:11 CET 2015 by mery

Created Wed Nov 18 16:33:27 CET 2015 by mery

Fin 7

Exercice 8

Vérifier le programme C suivant en la traduisant dans un module PlusCal.

/* C program to check whether a number is palindrome or not */

```

#include <stdio.h>
int main()
{
    int n, reverse=0, rem,temp;
    printf("Enter an integer: ");
    scanf("%d", &n);
    temp=n;
    while(temp!=0)
    {
        rem=temp%10;
        reverse=reverse*10+rem;
        temp/=10;
    }
    /* Checking if number entered by user and it's reverse number is equal. */
    if(reverse==n)
        printf("%d is a palindrome. ",n);
    else
        printf("%d is not a palindrome. ",n);
    return 0;
}

```

← Solution de l'exercice 8

MODULE pluscal5

```

EXTENDS TLC,Integers,Naturals
CONSTANTS n0

```

(·

```

--algorithm compte{
variables n = n0, reverse = 0, rem, temp;
{
    temp := n;
    while (temp ≠ 0)
    {
        rem := temp % 10;
        reverse := reverse·10+rem;
        temp := temp ÷ 10;
    };
    if (reverse = n) {
        print ("palin",n);
    }
    else
    {
        print ("not palin",n);
    }
}
}
.
```

BEGIN TRANSLATION
CONSTANT *defaultInitValue*
VARIABLES *n*, *reverse*, *rem*, *temp*, *pc*

vars \triangleq $\langle n, \text{reverse}, \text{rem}, \text{temp}, \text{pc} \rangle$

Init \triangleq Global variables
 $\wedge n = n_0$
 $\wedge \text{reverse} = 0$
 $\wedge \text{rem} = \text{defaultInitValue}$
 $\wedge \text{temp} = \text{defaultInitValue}$
 $\wedge \text{pc} = "Lbl_1"$

Lbl₋₁ \triangleq $\wedge \text{pc} = "Lbl_1"$
 $\wedge \text{temp}' = n$
 $\wedge \text{pc}' = "Lbl_2"$
 $\wedge \text{UNCHANGED} \langle n, \text{reverse}, \text{rem} \rangle$

Lbl₋₂ \triangleq $\wedge \text{pc} = "Lbl_2"$
 $\wedge \text{IF } \text{temp} \neq 0$
THEN $\wedge \text{rem}' = \text{temp} \% 10$
 $\wedge \text{reverse}' = \text{reverse} \cdot 10 + \text{rem}'$
 $\wedge \text{temp}' = (\text{temp} \div 10)$
 $\wedge \text{pc}' = "Lbl_2"$
ELSE $\wedge \text{IF } \text{reverse} = n$
THEN $\wedge \text{PrintT}(\langle \text{"palin"}, n \rangle)$
ELSE $\wedge \text{PrintT}(\langle \text{"not palin"}, n \rangle)$
 $\wedge \text{pc}' = \text{"Done"}$
 $\wedge \text{UNCHANGED} \langle \text{reverse}, \text{rem}, \text{temp} \rangle$
 $\wedge n' = n$

Next \triangleq *Lbl₋₁* \vee *Lbl₋₂
 \vee Disjunct to prevent deadlock on termination
 $(\text{pc} = \text{"Done"} \wedge \text{UNCHANGED} \text{ vars})$*

$Spec \triangleq Init \wedge \square[Next]_{vars}$

$Termination \triangleq \diamond(pc = "Done")$

END TRANSLATION

Fin 8

Exercice 9 Vérifier le programme C suivant en la traduisant dans un module PlusCal.

```
#include <stdio.h>
int main()
{
    int n, count=0;
    printf("Enter_an_integer :_");
    scanf("%d", &n);
    while(n!=0)
    {
        n /=10;           /* n=n/10 */
        ++count;
    }
    printf("Number_of_digits :_%d", count);
}
```

← Solution de l'exercice 9

MODULE tp9

EXTENDS TLC, Integers, Naturals
CONSTANTS n0

```
(.
--algorithm compte{
variables n, count = 0;
{
l0 : n := n0;
l1 : while (n ≠ 0)
{
l2 : n := n ÷ 10;
l3 : count := count +1;
};
l4 : print ⟨"Number of digits : %d", count⟩;
}
}
```

.)

BEGIN TRANSLATION
CONSTANT defaultInitValue
VARIABLES n, count, pc

$vars \triangleq \langle n, count, pc \rangle$

$Init \triangleq$ Global variables
 $\wedge n = defaultInitValue$

$$\begin{aligned}
& \wedge \text{count} = 0 \\
& \wedge \text{pc} = "l0" \\
l0 \triangleq & \wedge \text{pc} = "l0" \\
& \wedge n' = n0 \\
& \wedge \text{pc}' = "l1" \\
& \wedge \text{count}' = \text{count} \\
l1 \triangleq & \wedge \text{pc} = "l1" \\
& \wedge \text{IF } n \neq 0 \\
& \quad \text{THEN } \wedge \text{pc}' = "l2" \\
& \quad \text{ELSE } \wedge \text{pc}' = "l4" \\
& \wedge \text{UNCHANGED } \langle n, \text{count} \rangle \\
l2 \triangleq & \wedge \text{pc} = "l2" \\
& \wedge n' = (n \div 10) \\
& \wedge \text{pc}' = "l3" \\
& \wedge \text{count}' = \text{count} \\
l3 \triangleq & \wedge \text{pc} = "l3" \\
& \wedge \text{count}' = \text{count} + 1 \\
& \wedge \text{pc}' = "l1" \\
& \wedge n' = n \\
l4 \triangleq & \wedge \text{pc} = "l4" \\
& \wedge \text{PrintT}(\langle \text{"Number of digits : \%d"}, \text{count} \rangle) \\
& \wedge \text{pc}' = "Done" \\
& \wedge \text{UNCHANGED } \langle n, \text{count} \rangle
\end{aligned}$$

$\text{Next} \triangleq l_0 \vee l_1 \vee l_2 \vee l_3 \vee l_4$

\vee Disjunct to prevent deadlock on termination

$(\text{pc} = \text{"Done"} \wedge \text{UNCHANGED vars})$

$\text{Spec} \triangleq \text{Init} \wedge \square[\text{Next}]_{\text{vars}}$

$\text{Termination} \triangleq \diamond(\text{pc} = \text{"Done"})$

END TRANSLATION

$\text{test}_1 \triangleq \text{pc} = \text{"Done"} \wedge n_0 = 345 \Rightarrow \text{count} = 3$

$\text{test}_2 \triangleq \text{pc} = \text{"Done"} \wedge n_0 = 0 \Rightarrow \text{count} = 0$

$\text{safe} \triangleq n \neq \text{defaultValue} \Rightarrow 0 \leq n \wedge n \leq n0$

Fin 9

Exercice 10 Nous avons récupéré un programme C qui réalise le tri à bulles. Traduire ce programme en PlusCal et analyser sa correction partielle et l'absence d'erreurs à l'exécution.

```

void tri_a_bulle(int* t, int const size)
{
    int en_desordre = 1;
    int i, j;

```

```

for (i = 0; (i < size) && en_desordre; ++i)
{
    en_desordre = 0;
    for (j = 1; j < (size - i); ++j)
    {
        if (t[j-1] > t[j])
        {
            int temp = t[j-1];
            t[j-1] = t[j];
            t[j] = temp;
            en_desordre = 1;
        }
    }
}

#include<algorithm> // pour la fonction swap

void tri_a_bulle(int *t, int const size)
{
    bool en_desordre = true;
    for (int i = 0; i < size && en_desordre; ++i)
    {
        en_desordre = false;
        for (int j = 1; j < size - i; ++j)
        {
            if (t[j-1] > t[j])
            {
                std::swap(t[j], t[j-1]);
                en_desordre = true;
            }
        }
    }
}

```

◊ Solution de l'exercice 10

MODULE tp10

EXTENDS *TLC, Integers, Naturals*

```

n ≡ 5
t0 ≡ [k ∈ 0..n-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]

```

```

(.
--algorithm tribulle {
variables t, i, j, d = TRUE, temp;
{
    t := t0;
    i := 0;
    while(i < n ∧ d)
    {
        d := FALSE;
        j := 1;

```

```

while ( j < (n - i))
{
    if (t[j-1] > t[j])
    {
        temp := t[j-1];
        t[j-1] := t[j];
        t[j] := temp;
        d := TRUE;
    };
    j := j+1;
};
i := i+1;
}
}

BEGIN TRANSLATION
CONSTANT defaultInitValue
VARIABLES t, i, j, d, temp, pc

vars  $\triangleq$   $\langle t, i, j, d, \text{temp}, \text{pc} \rangle$ 

Init  $\triangleq$  Global variables
 $\wedge t = \text{defaultInitValue}$ 
 $\wedge i = \text{defaultInitValue}$ 
 $\wedge j = \text{defaultInitValue}$ 
 $\wedge d = \text{TRUE}$ 
 $\wedge \text{temp} = \text{defaultInitValue}$ 
 $\wedge \text{pc} = "Lbl\_1"$ 

Lbl_1  $\triangleq$   $\wedge \text{pc} = "Lbl\_1"$ 
 $\wedge t' = t0$ 
 $\wedge i' = 0$ 
 $\wedge \text{pc}' = "Lbl\_2"$ 
 $\wedge \text{UNCHANGED} \langle j, d, \text{temp} \rangle$ 

Lbl_2  $\triangleq$   $\wedge \text{pc} = "Lbl\_2"$ 
 $\wedge \text{IF } i < n \wedge d$ 
    THEN  $\wedge d' = \text{FALSE}$ 
 $\wedge j' = 1$ 
 $\wedge \text{pc}' = "Lbl\_3"$ 
ELSE  $\wedge \text{pc}' = "Done"$ 
 $\wedge \text{UNCHANGED} \langle j, d \rangle$ 
 $\wedge \text{UNCHANGED} \langle t, i, \text{temp} \rangle$ 

Lbl_3  $\triangleq$   $\wedge \text{pc} = "Lbl\_3"$ 
 $\wedge \text{IF } j < (n - i)$ 
    THEN  $\wedge \text{IF } t[j-1] > t[j]$ 
        THEN  $\wedge \text{temp}' = t[j-1]$ 
 $\wedge t' = [t \text{ EXCEPT } ![j-1] = t[j]]$ 
 $\wedge \text{pc}' = "Lbl\_4"$ 
ELSE  $\wedge \text{pc}' = "Lbl\_5"$ 
 $\wedge \text{UNCHANGED} \langle t, \text{temp} \rangle$ 
 $\wedge i' = i$ 

```

```

ELSE  $\wedge i' = i+1$ 
       $\wedge pc' = "Lbl\_2"$ 
       $\wedge \text{UNCHANGED } \langle t, temp \rangle$ 
       $\wedge \text{UNCHANGED } \langle j, d \rangle$ 

 $Lbl\_5 \triangleq \wedge pc = "Lbl\_5"$ 
       $\wedge j' = j+1$ 
       $\wedge pc' = "Lbl\_3"$ 
       $\wedge \text{UNCHANGED } \langle t, i, d, temp \rangle$ 

 $Lbl\_4 \triangleq \wedge pc = "Lbl\_4"$ 
       $\wedge t' = [t \text{ EXCEPT } !j] = temp]$ 
       $\wedge d' = \text{TRUE}$ 
       $\wedge pc' = "Lbl\_5"$ 
       $\wedge \text{UNCHANGED } \langle i, j, temp \rangle$ 

 $Next \triangleq Lbl\_1 \vee Lbl\_2 \vee Lbl\_3 \vee Lbl\_5 \vee Lbl\_4$ 
       $\vee \quad \text{Disjunct to prevent deadlock on termination}$ 
       $(pc = \text{"Done"} \wedge \text{UNCHANGED } vars)$ 

```

$Spec \triangleq Init \wedge \square[Next]_{vars}$

$Termination \triangleq \diamondsuit(pc = \text{"Done"})$

END TRANSLATION

$Safe_1 \triangleq pc \neq \text{"Done"}$
 $Safe_2 \triangleq pc = \text{"Done"} \Rightarrow \forall e \in 0..n-1 : t[e] \leq t[e+1]$
 $Safe_3 \triangleq pc = \text{"Done"} \Rightarrow \{ e \in 0..n-1 : t[e] \leq t[e+1] \} = 0..n-1$
 $Safe_4 \triangleq pc = \text{"Done"} \Rightarrow (t[0] \leq t[1]) \wedge (t[1] \leq t[2])$

Fin 10

Exercice 11 Analyser le programme C suivant qui réalise le tri par sélection.

```

void selection(int *t, int taille)
{
    int i, mini, j, x;

    for (i = 0; i < taille - 1; i++)
    {
        mini = i;
        for (j = i + 1; j < taille; j++)
            if (t[j] < t[mini])
                mini = j;
        x = t[i];
        t[i] = t[mini];
        t[mini] = x;
    }
}

```

◊ Solution de l'exercice 11

MODULE tp11

EXTENDS TLC, Integers, Naturals

```

CONSTANTS n0
(·
--algorithm trisel {
variables
{
    void selection(int ·t, int taille)
    {
        int i, mini, j, x;

        for (i = 0; i < taille - 1; i++)
        {
            mini = i;
            for (j = i + 1; j < taille; j++)
                if (t[j] < t[mini])
                    mini = j;
            x = t[i];
            t[i] = t[mini];
            t[mini] = x;
        }
    }
    ·)
}

```

Fin 11

Exercice 12 GCD en Event-B

Soit l'algorithme GCD calculant le pgcd de deux nombres entiers positifs quelconques a et b .

```

begin
start;
while y1 ≠ y2 do
loop;
if y1 < y2 then
inloop1:
y2' := y2 - y1 ;
endif1;
else
inloop2:
y1' := y1 - y2 ;
endif2;
fi
endif;
endwhile;
endloop;
g := y1;
end.

```

Question 12.1 Traduire l'algorithme en un module TLA⁺ qui précise les annotations et qui précise les invariants pour la correction partielle et l'absence d'erreurs à m'l'exécution.

← Solution de la question 12.1

MODULE gcd

EXTENDS Integers, TLC

CONSTANTS a, b

VARIABLES y_1, y_2, z, l

$Init \triangleq y_1 = a \wedge y_2 = b \wedge z = 0 \wedge l = "start"$
 $L1 \triangleq l = "start" \wedge y_1 \neq y_2 \wedge l' = "loop" \wedge \text{UNCHANGED} \langle y_1, y_2, z \rangle$
 $L2 \triangleq l = "start" \wedge y_1 = y_2 \wedge l' = "outloop" \wedge \text{UNCHANGED} \langle y_1, y_2, z \rangle$
 $L3 \triangleq l = "loop" \wedge y_1 < y_2 \wedge l' = "inloop1" \wedge \text{UNCHANGED} \langle y_1, y_2, z \rangle$
 $L4 \triangleq l = "inloop1" \wedge y_2' = y_2 - y_1 \wedge l' = "endif1" \wedge \text{UNCHANGED} \langle y_1, z \rangle$
 $L5 \triangleq l = "endif1" \wedge l' = "endif" \wedge \text{UNCHANGED} \langle y_1, y_2, z \rangle$
 $L6 \triangleq l = "loop" \wedge y_1 > y_2 \wedge l' = "inloop2" \wedge \text{UNCHANGED} \langle y_1, y_2, z \rangle$
 $L7 \triangleq l = "inloop2" \wedge y_1' = y_1 - y_2 \wedge l' = "endif2" \wedge \text{UNCHANGED} \langle y_2, z \rangle$
 $L8 \triangleq l = "endif2" \wedge l' = "endif" \wedge \text{UNCHANGED} \langle y_1, y_2, z \rangle$
 $L9 \triangleq l = "endif" \wedge y_1 \neq y_2 \wedge l' = "loop" \wedge \text{UNCHANGED} \langle y_1, y_2, z \rangle$
 $L10 \triangleq l = "endif" \wedge y_1 = y_2 \wedge l' = "endloop" \wedge \text{UNCHANGED} \langle y_1, y_2, z \rangle$

$$L11 \triangleq l = \text{"endloop"} \wedge z' = y_1 \wedge l' = \text{"end"} \wedge \text{UNCHANGED} \langle y_1, y_2 \rangle$$
$$\begin{aligned} Next &\triangleq \vee L_1 \vee L_2 \vee L_3 \vee L_4 \vee L_5 \\ &\quad \vee L_6 \vee L_7 \vee L_8 \vee L_9 \vee L_{10} \vee L_{11} \\ safe_1 &\triangleq 0 \leq y_1 \wedge y_1 \leq a \wedge 0 \leq y_2 \wedge y_2 \leq b \\ safe_2 &\triangleq y_1 \neq y_2 \\ safe_3 &\triangleq l \neq \text{"end"} \\ safe_4 &\triangleq l \neq \text{"endif2"} \end{aligned}$$

Fin 12.1

Question 12.2 Construire une machine Event-B qui liste les actions de l'algorithme et qui donne l'invariant pour la correction partielle et l'absence d'erreurs à l'exécution.