

Cours Modélisation et vérification des systèmes informatiques

Exercices

Modélisation d'algorithmes en PlusCal (I)

par Dominique Méry

8 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'un 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**

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

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

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

**Question 1.3**

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

**Question 1.4**

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

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

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.

**Exercice 5** On considère l'algorithme suivant :

```

START
{x1 > 0 ∧ x2 > 0}
(y1, y2, y3, y4) ← (x1, x2, x2, 0);
while y1 ≠ y2 do
    begin while y1 > y2 do (y1, y4) ← (y1 - y2, y4 + y3);
        while y2 > y1 do (y2, y3) ← (y2 - y1, y3 + y4)
    end;
    (z1, z2) ← (y1, y3 + y4)
    {z1 = gcd(x1, x2) ∧ z2 = scm(x1, x2)}
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 5.1** 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 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  $x_1$  et  $x_2$  sont des constantes.

```

precondition : x1 ∈ N ∧ x2 ∈ N ∧ x1 ≠ 0
postcondition : z = x1x2
local variables : y1, y2, y3 ∈ Z

ℓ0 : {y1, y2, y3, z ∈ Z}
(y1, y2, y3) := (x1, x2, 1);
ℓ1 : {y3 · y1y2 = x1x2 ∧ y1, y2, y3 ∈ N ∧ z ∈ Z}
while y2 ≠ 0 do
    ℓ2 : {y2 ≠ 0 ∧ y3 · y1y2 = x1x2 ∧ y1, y2, y3 ∈ N ∧ z ∈ Z}
    if impair(y2) then
        ℓ3 : {impair(y2) ∧ y2 ≠ 0 ∧ [•••] ∧ y1, y2, y3 ∈ N ∧ z ∈ Z}
        y2 := y2 - 1;
        ℓ4 : {y2 ≥ 0 ∧ pair(y2) ∧ y3 · y1y2 = x1x2 ∧ y1, y2, y3 ∈ N ∧ z ∈ Z}
        y3 := y3 · y1;
        ℓ5 : {y2 ≥ 0 ∧ pair(y2) ∧ y3 · y1y2 = x1x2 ∧ y1, y2, y3 ∈ N ∧ z ∈ Z}
    ;
    ℓ6 : {y2 ≥ 0 ∧ pair(y2) ∧ [•••] ∧ y1, y2, y3 ∈ N ∧ z ∈ Z}
    y1 := y1 · y1;
    ℓ7 : {y2 ≥ 0 ∧ pair(y2) ∧ y3 · y1y2 div 2 = x1x2 ∧ y1, y2, y3 ∈ N ∧ z ∈ Z}
    y2 := y2 div 2;
    ℓ8 : {y2 ≥ 0 ∧ [•••] ∧ y1, y2, y3 ∈ N ∧ z ∈ Z}
;
ℓ9 : {y2 = 0 ∧ [•••] ∧ y1, y2, y3 ∈ N ∧ z ∈ Z}
z := y3;
ℓ10 : {y2 = 0 ∧ y3 · y1y2 = x1x2 ∧ y1, y2, y3 ∈ N ∧ z ∈ Z ∧ z = x1x2}
```

**Algorithme 2:** Algorithme de l'exponentiation indienne annoté

**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*

### 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

---

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);
}

```

**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;
            }
        }
    }
}

```

**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;
    }
}

```

**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<sup>+</sup> qui précise les annotations et qui précise les invariants pour la correction partielle et l'absence d'erreurs à m'l'exécution.

**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.