

Cours MOdélisation, Vérification et EXpérimentations  
Exercices (avec les corrections)  
Série A Annotation, modélisation, vérification - Validation en TLA<sup>+</sup>  
par Dominique Méry  
5 mars 2025

## TD1

### Exercice 1 (*malgtdlex1*)

Le PGCD de deux nombres vérifie les propriétés suivantes :

- $\forall a, b \in \mathbb{N}. \text{pgcd}(a, b) = \text{pgcd}(b, a)$
- $\forall a, b \in \mathbb{N}. \text{pgcd}(a, a+b) = \text{pgcd}(a, b)$

*mov*

- Ecrire une spécification TLA<sup>+</sup> calculant le PGCD de deux nombres donnés.
- Donner une explication ou une justification de la correction de cette solution

### ◇ Solution de l'exercice 1

Modèle solution TLA<sup>+</sup> :

../tlamodels/malgtdlex1.tla

```
----- MODULE malgtdlex1 -----
EXTENDS Naturals
-----
CONSTANTS a, b
-----
VARIABLES  x, y
-----
ASSUME a \in Nat /\ b \in Nat
toto == x=a /\ y=b
-----
(* actions *)
a1 ==
    /\ x > y
    /\ x' = x - y
    /\ y' = y
a2 == x < y /\ y' = y - x /\ x' = x
over == x = y /\ x' = x /\ y' = y
-----
go == a1 \/ a2 \/ over

-----
(* Propriétés de sûreté à vérifier *)
test == x # y
prop1 == x \geq 0 /\* ok
prop2 == x + y \leq a + b /\* ok
tocheck == prop1 /\ prop2
=====
```

**Fin 1**

### Exercice 2 (*malgtdlex2*)

L'accès à une salle est contrôlé par un système permettant d'observer les personnes qui entrent ou qui sortent de cette salle. Ce système est un ensemble de capteurs permettant d'identifier

le passage d'une personne de l'extérieur vers l'intérieur et de l'intérieur à l'extérieur. Le système doit garantir qu'au plus  $max$  personnes soient dans la salle. Ecrire un module  $TLA^+$  permettant de modéliser un tel système respectant la propriété attendue.

◊— **Solution de l'exercice 2** \_\_\_\_\_

Modèle solution  $TLA^+$  :

../tlamodels/malgtddlex2.tla

----- MODULE malgtddlex2 -----

(\* modules de base importables \*)

EXTENDS Naturals, TLC

CONSTANTS max

VARIABLES np

(\* tentative 1 \*)

entrer == np' = np + 1

sortir == np' = np - 1

Next ==

  \/ entrer

  \/ sortir

Init == np = 0

(\* tentative 2 \*)

sortir2 == np > 0 /\ np' = np - 1

next2 == entrer \/ sortir2

entrer2 == np < max /\ np' = np + 1

next3 == entrer2 \/ sortir2

-----

safety1 == np \leq max \\* ok

safety2 == 0 \leq np \\* ok

question1 == np # 6

tocheck == question1

=====

Nous donnons trois solutions possibles selon notre analyse :

- la première solution propose deux actions *entrer* et *sortir* et on définit une relation de transition *next*. On définit un modèle en instanciant *max* et en testant les deux propriétés de sûreté *question1* et *safety1*. Les deux questions produisent un échec et donc la propriété de sûreté n'est pas vérifiée.
- la deuxième solution propose deux actions *entrer2* et *sortir* et on définit une relation de transition *next2*. On définit un modèle en instanciant *max* et en testant les deux propriétés de sûreté *question1* et *safety1*. Il n'y a pas de retour sur la question *safety1*.
- la troisième solution propose deux actions *entrer2* et *sortir2* et on définit une relation de transition *next3*. On définit un modèle en instanciant *max* et en testant les deux propriétés de sûreté *question1* et *safety1*. L'utilisation de l'outil conduit à la vérification de la propriété de *safety1*.

**Fin 2**

**Exercice 3** (*malgtddlex3*)

On considère l'algorithme suivant décrit par un organigramme ou flowchart de la figure 1.

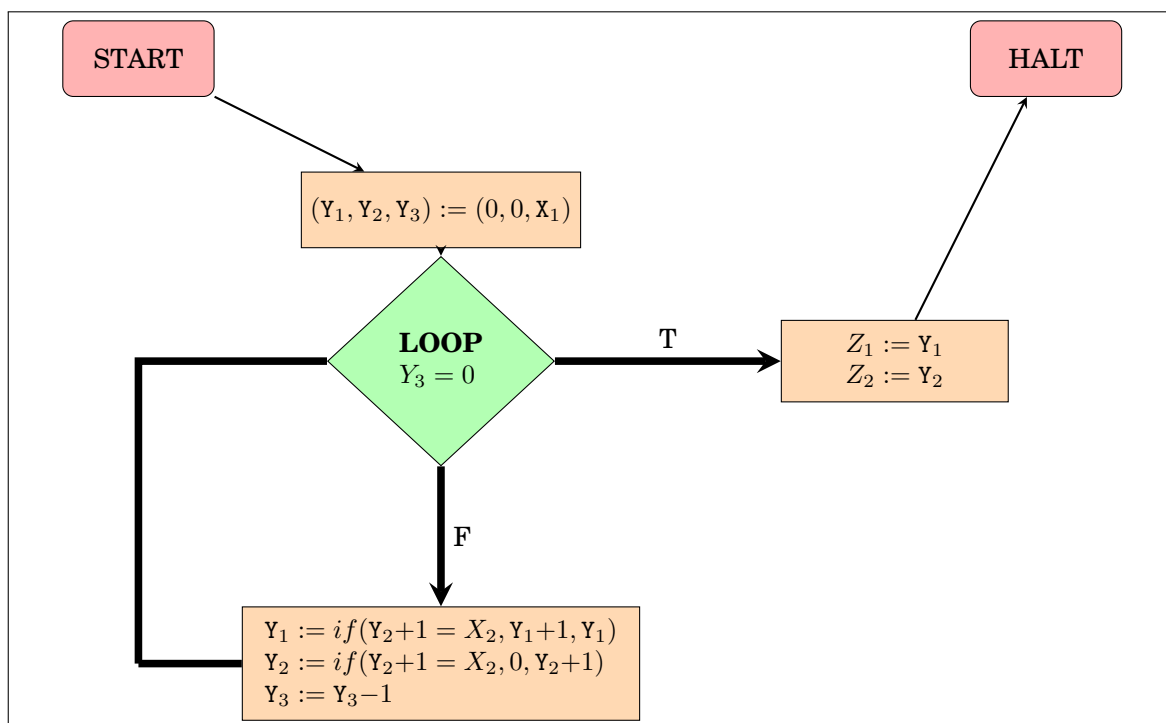


FIGURE 1 – Organigramme de calcul de la division entière

Cet algorithme calcule le reste et le quotient de la division de  $x_1$  par  $x_2$  :  $0 \leq z_2 \leq x_2 \wedge x_1 = z_1 \cdot x_2 + z_2$ . On suppose que  $x_1$  et  $x_2$  sont positifs et non nuls.

**Question 3.1** Donner la précondition et la postcondition associées à cet algorithme.

**Question 3.2** Traduire cet algorithme sous forme d'un module  $TLA^+$ .

**Question 3.3** Tester les valeurs des variables à l'exécution.

**Question 3.4** Montrer que cet algorithme est partiellement correct par rapport à sa précondition et à sa postcondition qu'il faudra énoncer.

### ◇ Solution de l'exercice 3

Modèle solution  $TLA^+$  :

../tlamodels/malgtd1ex3.tla

```

----- MODULE malgtd1ex3 -----
EXTENDS Integers, TLC, Naturals
CONSTANTS UND, x10, x20, maxi, mini
VARIABLES x1, x2, y1, y2, y3, z1, z2, pc
-----

ASSUME x10 \in Nat /\ x20 \in Nat /\ x20 # 0
labels == {"START", "LOOP", "HALT"}

Init ==
    /\ pc="START"
    /\ x1 = x10 /\ x2 = x20
    /\ y1=UND /\ y2=UND /\ y3 = UND
    /\ z1=UND /\ z2 =UND
    
```

```
(* y1 \in min..max /\ y2 \in min..max /\ y3 \in min..max /\ z1 \in min..max /\ z2

start_loop ==
  /\ pc = "START"
  /\ pc' = "LOOP"
  /\ y1'=0 /\ y2'=0 /\ y3'=x1
  /\ UNCHANGED <<z1,z2,x1,x2>>

loop_loop ==
  /\ pc = "LOOP" /\ y3 # 0
  /\ y1' = IF y2+1=x2 THEN y1+1 ELSE y1
  /\ y2' = IF y2+1=x2 THEN 0 ELSE y2+1
  /\ y3' = y3 -1
  /\ UNCHANGED <<pc,x1,x2,z1,z2>>

loop_halt ==
  /\ pc = "LOOP" /\ pc'="HALT" /\ y3 = 0
  /\ z1' = y1 /\ z2'=y2
  /\ UNCHANGED <<x1,x2,y1,y2,y3>>

Over ==
  /\ pc="HALT" /\ PrintT(z1) /\ PrintT(z2)
  /\ UNCHANGED<<pc,x1,x2,y1,y2,y3,z1,z2>>

next == start_loop \/ loop_loop \/ loop_halt\/ Over

-----
safety1 == pc="HALT" => 0 \leq z2 /\ z2 < x2 /\ x1=z1*x2+z2 /\ x1=x10 /\ x2=x20

D == mini..maxi

DD(X) == (X # UND => X \in D)

safety2 == DD(y1) /\ DD(y2) /\ DD(y3) /\ DD(z1) /\ DD(z2)

test == safety1
=====
```

---

**Fin 3**

## TD2

### Exercice 4 (malgtd1ex4)

La fonction de McCarthy  $f_{91}$  est définie pour tout entier  $x$   $f_{91}(x) = \text{if } x > 100 \text{ then } x-10 \text{ else } 91$  fi.

**Question 4.1** Définir le contrat établissant la correction partielle de l'algorithme ALG91 de la figure 2 qui est réputé calculer la fonction  $f_{91}$

**Question 4.2** Construire un module  $TLA^+$  modélisant les différents pas de calcul.

**Question 4.3** Evaluer l'algorithme en posant des questions de sûreté suivantes :

1. l'algorithme est partiellement correct.

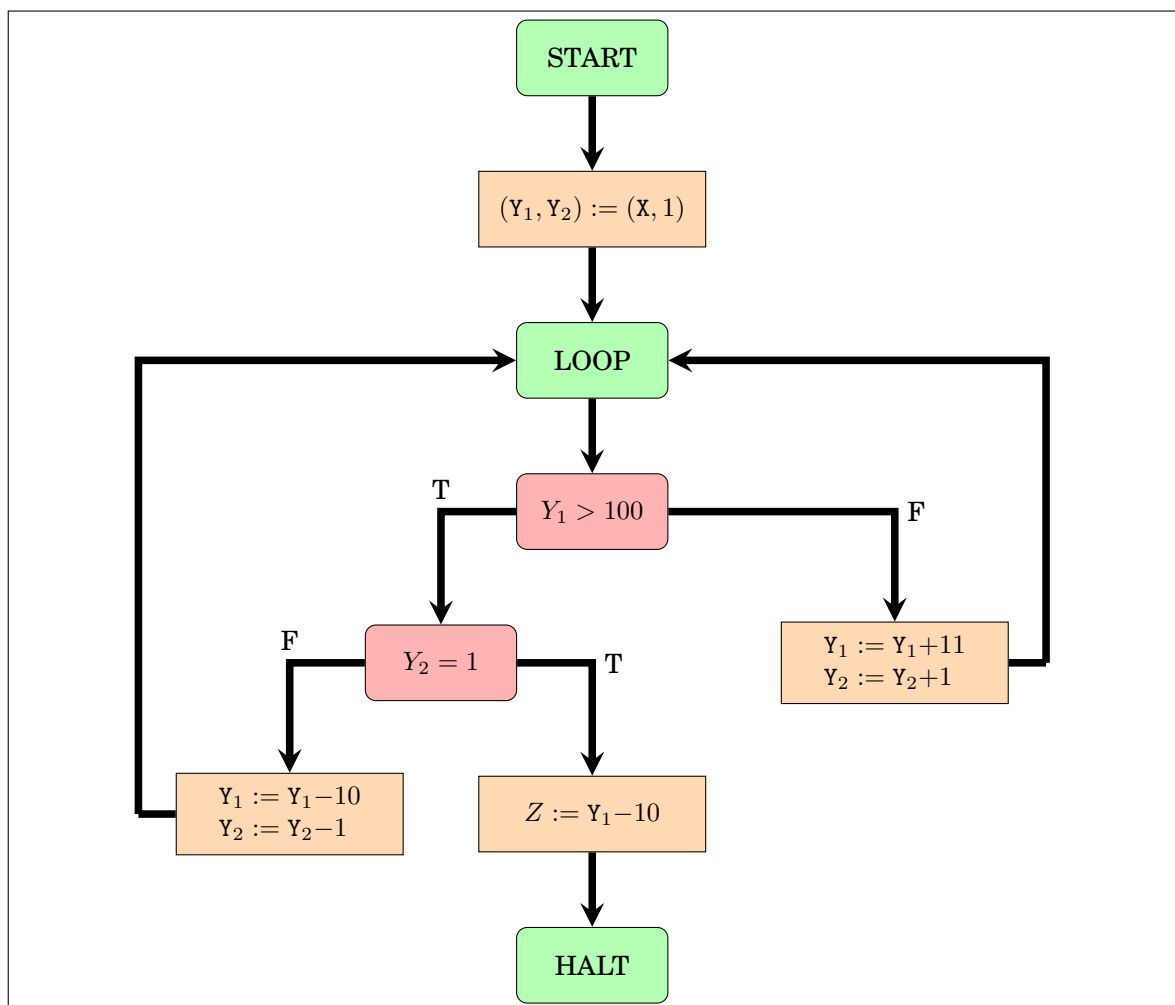


FIGURE 2 – Flowchart du calcul de la fonction de McCarthy

*2. l'algorithme n'a pas d'erreurs à l'exécution.*

◇ **Solution de l'exercice 4**

---

Modèle solution TLA<sup>+</sup> :

../tlamodels/malgtdlex4.tla

```

----- MODULE malgtdlex4 -----
EXTENDS Naturals, TLC, Integers

(* contract *)
(* variables x, y1, y2, z *)
(* requires x0 \in Nat /\ y10, y20, z0 \in Nat /\ pc="l0" *)
(* ensures zf=f91(x0) *)

-----

CONSTANTS x0

(* auxiliary definitions *)
mini == -2^15
maxi == 2^15-1
D == mini..maxi
UND == -650000
f91 == [i \in Int |-> IF i > 100 THEN i-10 ELSE 91]

-----

VARIABLES x, y1, y2, z, pc

(* preconditions *)
ASSUME x0 \geq 0

-----

(* actions *)
a ==
  /\ pc="START"
  /\ y1'=x /\ y2'=1
  /\ pc'="LOOP"
  /\ UNCHANGED <<x, z>>

b ==
  /\ pc="LOOP" /\ y1 \leq 100
  /\ y1'=y1+1 /\ y2'=y2+1
  /\ UNCHANGED <<x, z, pc>>

cc ==
  /\ pc="LOOP" /\ y1 > 100 /\ y2#1
  /\ y1'=y1-10 /\ y2'=y2-1
  /\ UNCHANGED <<x, z, pc>>
  /\ PrintT(y1) /\ PrintT(y2)

d ==
  /\ pc="LOOP" /\ y1 > 100 /\ y2=1
  /\ z'=y1-10 /\ pc'="HALT"
  /\ UNCHANGED <<x, y1, y2>>

-----

(* specification *)
Next == a \/ b \/ cc \/ d /\ UNCHANGED <<y1, y2, z, x, pc>>
init1 == x=x0 /\ y1 \in Int /\ y2 \in Int /\ z \in Int /\ pc = "START"
Init == y1 = UND /\ y2 = UND /\ z = UND /\ x = x0 /\ pc = "START"
-----

```



FIGURE 3 – Flowchart pour le test de primalité

```

(* analyse *)
Q1 == pc#"HALT" (* pc prned la valeur HALT *) (* fausse *)
Qpc == pc#"HALT" => z=IF x>100 THEN x-10 ELSE 91
Q(y) == y # UND => mini \leq y /\ y \leq maxi
Qover == Q(y1) /\ Q(y2) /\ Q(z) /\ Q(x)
Q2== Qpc /\ Qover
tocheck == Qover
=====
  
```

**Fin 4**

**Exercice 5** (*malgtd1ex5 et inmalgtd1ex5*)

Soit le schéma de la figure 3 définissant un calcul déterminant, si un nombre entier naturel est premier ou non.

**Question 5.1** Ecrire un module TLA/TLA<sup>+</sup> modélisant ce schéma de calcul et montrer que le modèle est sans blocage.

**Question 5.2** Définir la propriété *prime(x)* qui est vraie si *x* est premier et faux sinon.

**Question 5.3** Ecrire le contrat présumé du calcul du flowchart de la figure 3

requires  $x0 > 0 \wedge x0, y0 \in \mathbb{Z} \wedge z0 \in \mathbb{B}$   
 ensures  $zf = \text{prime}(x0) \wedge xf = x0$

**Question 5.4** Vérifier la correction partielle

**Question 5.5** Vérifier l'absence d'erreurs à l'exécution.

◊— **Solution de l'exercice 5**

Modèle solution TLA<sup>+</sup> :

../tlamodels/malgtdlex5.tla

```
-----MODULE malgtdlex5 -----

EXTENDS Integers, TLC

-----

(* contract *)
(* variables x,y,z *)
(* requires x0 \in Nat /\ y0 \in Nat /\ Z \IN BOOL *)
(* ensures zf= prime(x0) *)
CONSTANTS mini,maxi,und, bund (* constants for undefinedness, bounds of domain *)

-----

(* requires *)
CONSTANTS x0 (* x0 is the input *)

-----

(* precondition *)
ASSUME x0 \in Nat

-----

VARIABLES x,y,z,pc

-----

Init == x = x0 /\ y=und /\ z=bund /\ pc="start"

-----

L1 == pc = "start" /\ y'=2 /\ pc'="loop" /\ UNCHANGED <<x,z>>
L2 == pc = "loop" /\ y \geq x /\ z'=TRUE /\ pc'="halt" /\ UNCHANGED <<x,y>>
L3 == pc="loop" /\ y<x /\ x % y =0 /\ z'= FALSE /\ pc'="halt" /\ UNCHANGED <<x,y>>
L4 == pc="loop" /\ y<x /\ x % y # 0 /\ y'=y+1 /\ UNCHANGED << pc,x,z>>
skip == UNCHANGED << pc,x,z,y >>

-----

Next == L1 \/ L2 \/ L3 \/ L4 \/ skip

-----

(* auxiliary definitions *)
prime(u) == \A v \in 2..u-1: u % v # 0 (* define that u is a prime number *)
Dbool == {FALSE,TRUE}
Dint == mini..maxi (* domain for integer variables *)
DDint(v) == v # und => v \in Dint
DDbool(v) == v # bund => v \in Dbool

-----

(* properties to check *)
SafePC == pc="halt" => z=prime(x0) /\ PrintT(z) (* the algorithm is partially corre
SafeRTE == DDint(y) /\ DDbool(z) (* the algorithm is runtime errors free. *)
Safe == SafePC /\ SafeRTE

=====
```



Modèle solution TLA<sup>+</sup> :  
 ../tlamodels/inmalgtd1ex5.tla

```

-----MODULE inmalgtd1ex5 -----

EXTENDS Integers, TLC
-----

CONSTANTS mini,maxi,und, bund (* constants for undefinedness, bounds of domain *)
-----

prime(u) == \A v \in 2..u-1: u % v # 0 (* define that x is a prime number *)
(* requires *)
CONSTANTS x0,y0,z0 (* x is the input *)
ASSUME x0 \in Nat /\ y0 \in Int /\ z0 \in {FALSE,TRUE}
(* ensures *)
post(u0,v0,w0,u,v,w) == (w = prime(u0))
-----

VARIABLES x,y,z,pc
-----

Init == x=x0 /\ y=y0 /\ z=z0 /\ pc="start"
-----

L1 == pc = "start" /\ y'=2 /\ pc'="loop" /\ UNCHANGED <<x, z>> /\ PrintT(y)
L2 == pc = "loop" /\ y \geq x /\ z'=TRUE /\ pc'="halt" /\ UNCHANGED <<x,y>>
L3 == pc="loop" /\ y<x /\ x % y =0 /\ z'= FALSE /\ pc'="halt" /\ UNCHANGED <<x,y
L4 == pc="loop" /\ y<x /\ x % y # 0 /\ y'=y+1 /\ UNCHANGED <<x,pc,z>>
skip == UNCHANGED <<pc,z,y,x>>
-----

Next == L1 \/ L2 \/ L3 \/ L4 \/ skip
-----

(* auxiliary definitions *)
Dint == mini..maxi (* domain for integer variables *)
Dbool == {FALSE,TRUE}
DDint(v) == v # und => v \in Dint
DDbool(v) == v # bund => v \in Dbool
-----

Q1 == pc="halt" => post(x0,y0,z0,x,y,z) (* is the algorithm partially correct? *)
SafePC == pc="halt" => post(x0,y0,z0,x,y,z) (* the algorithm is partially correct
Q2 == pc # "halt"
Q3 == DDint(x) /\ DDint(y) /\ DDbool(z) (* is the algorithm runtime errors free? *)
SafeRTE == DDint(x) /\ DDint(y) /\ DDbool(z) (* the algorithm is runtime errors
Safe == SafePC /\ SafeRTE
=====
  
```

**Fin 5**

**Exercice 6** Dans cet exercice, il est question de découvrir les modules de base de TLA Toolbox comme TLC, Integers, Naturals ... afin de découvrir les fonctions qui sont prédéfinies.

## TD3

**Exercice 7** (Utilisation de ToolBox et TLA pour un labyrinthe, malgtd1ex7)

Le module truc permet de résoudre un problème très classique en informatique : trouver un chemin entre un sommet input et des sommets output supposés être des sommets de sortie.

**Question 7.1** Pour trouver un chemin de input à l'un des sommets de output, il faut poser une question de sûreté à notre système de vérification. Donner une question de sûreté à poser permettant de trouver un chemin de input vers un sommet de output.

MODULE *Naturals*

A dummy module that defines the operators that are defined by the real *Naturals* module.

$Nat \triangleq \{ \}$   
 $a+b \triangleq \{a, b\}$   
 $a-b \triangleq \{a, b\}$   
 $a \cdot b \triangleq \{a, b\}$   
 $a^b \triangleq \{a, b\}$   
 $a < b \triangleq a = b$   
 $a > b \triangleq a = b$   
 $a \leq b \triangleq a = b$   
 $a \geq b \triangleq a = b$   
 $a \% b \triangleq \{a, b\}$   
 $a \div b \triangleq \{a, b\}$   
 $a .. b \triangleq \{a, b\}$

MODULE *TLC*

LOCAL INSTANCE *Naturals*  
 LOCAL INSTANCE *Sequences*

$Print(out, val) \triangleq val$   
 $PrintT(out) \triangleq TRUE$   
 $Assert(val, out) \triangleq \text{IF } val = TRUE \text{ THEN } TRUE$   
 $\hspace{15em} \text{ELSE CHOOSE } v : TRUE$   
 $JavaTime \triangleq \text{CHOOSE } n : n \in Nat$   
 $TLCGet(i) \triangleq \text{CHOOSE } n : TRUE$   
 $TLCSet(i, v) \triangleq TRUE$

$d \rightarrow e \triangleq [x \in \{d\} \mapsto e]$   
 $f @@ g \triangleq [x \in (\text{DOMAIN } f) \cup (\text{DOMAIN } g) \mapsto$   
 $\hspace{10em} \text{IF } x \in \text{DOMAIN } f \text{ THEN } f[x] \text{ ELSE } g[x]]$   
 $Permutations(S) \triangleq$   
 $\{f \in [S \rightarrow S] : \forall w \in S : \exists v \in S : f[v] = w\}$

In the following definition, we use *Op* as the formal parameter rather than *\prec* because TLC Version 1 can't handle infix formal parameters.

$SortSeq(s, Op(\_, \_)) \triangleq$   
 $\text{LET } Perm \triangleq \text{CHOOSE } p \in Permutations(1 .. Len(s)) :$   
 $\hspace{10em} \forall i, j \in 1..Len(s) :$   
 $\hspace{15em} (i < j) \Rightarrow Op(s[p[i]], s[p[j]]) \vee (s[p[i]] = s[p[j]])$   
 $\text{IN } [i \in 1..Len(s) \mapsto s[Perm[i]]]$

$RandomElement(s) \triangleq \text{CHOOSE } x \in s : TRUE$

$Any \triangleq \text{CHOOSE } x : TRUE$

$ToString(v) \triangleq (\text{CHOOSE } x \in [a : v, b : STRING] : TRUE).b$

$TLCEval(v) \triangleq v$

FIGURE 4 – Modules *Naturals.tla* et *TLC.tla*

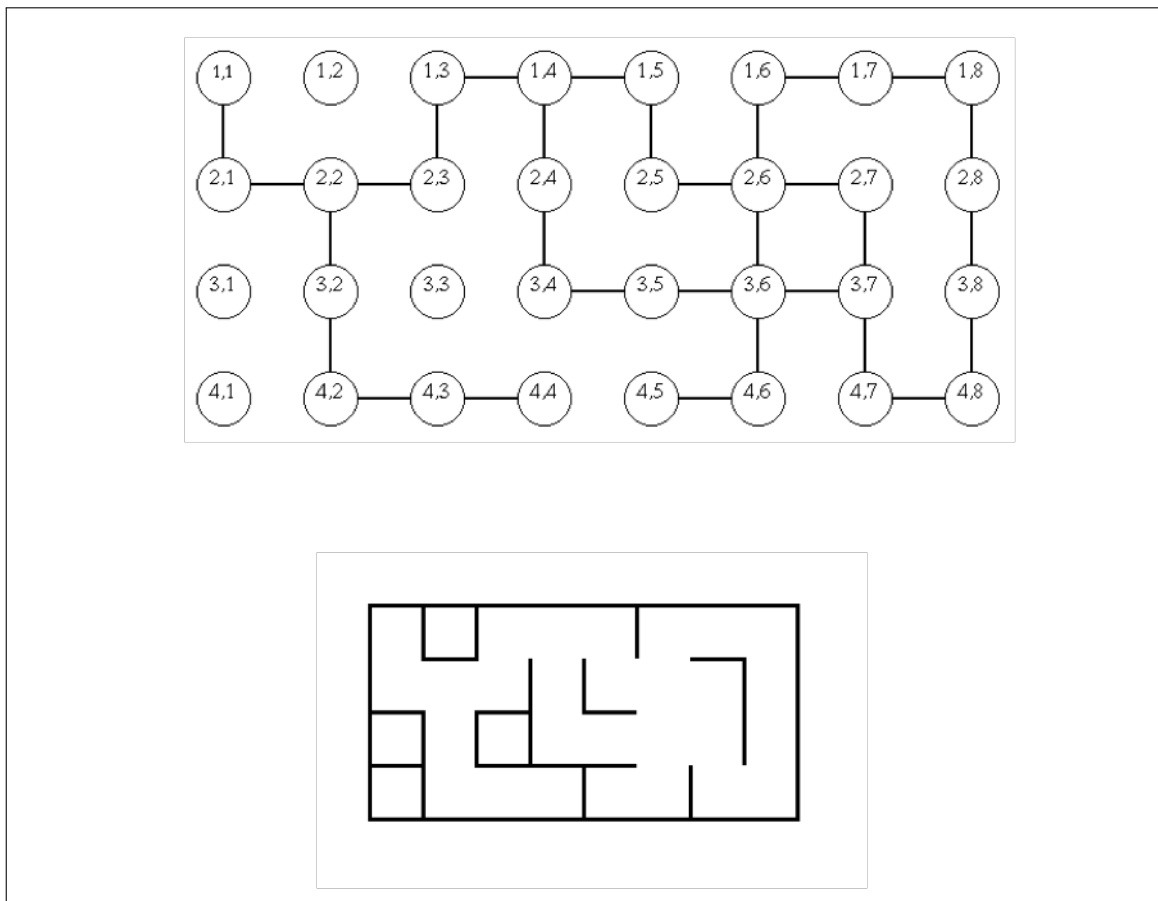


FIGURE 5 – Labyrinthe

**Question 7.2** On désire utiliser cette technique pour trouver un chemin dans un labyrinthe. Un labyrinthe est représenté par une matrice carrée de taille  $n$ . On définit ensuite pour chaque élément  $\langle\langle i, j \rangle\rangle$  de la matrice les voisins communiquant à l'aide de la fonction `lab` qui associe à  $\langle\langle i, j \rangle\rangle$  les éléments qui peuvent être atteints en un coup. Par exemple, le mouvement possible à partir de  $\langle\langle 1, 1 \rangle\rangle$  est  $\langle\langle 2, 1 \rangle\rangle$ , ou le mouvement possible à partir de  $\langle\langle 2, 1 \rangle\rangle$  est  $\langle\langle 2, 2 \rangle\rangle$  ou  $\langle\langle 1, 1 \rangle\rangle$ , ou le mouvement possible à partir de  $\langle\langle 2, 2 \rangle\rangle$  est  $\langle\langle 2, 3 \rangle\rangle$  ou  $\langle\langle 3, 2 \rangle\rangle$  ou  $\langle\langle 2, 1 \rangle\rangle$ , ...

```
lab == [<<x,y>> \in (nodes \X nodes) |->
      IF x=1 /\ y=1 THEN {<<2,1>>} ELSE
      IF x=2 /\ y=1 THEN {<<2,2>>}
      IF x=1 /\ y=2 THEN {} ELSE
      IF x=2 /\ y=2 THEN {<<3,2>>, <<2,3>>} ELSE
      ELSE {}
    ]
```

Modifier le module `truc` pour traiter ce problème et donner la question à poser pour trouver une sortie.

MODULE *truc*

EXTENDS *Integers, TLC*  
 VARIABLES *p*  
 CONSTANTS *input, output*

$$\begin{aligned}
 n &\triangleq 10 \\
 nodes &\triangleq 1..n \\
 l &\triangleq [i \in 1..n \mapsto \text{IF } i = 1 \text{ THEN } \{4, 5\} \text{ ELSE} \\
 &\quad \text{IF } i = 2 \text{ THEN } \{6, 7, 10\} \text{ ELSE} \\
 &\quad \text{IF } i = 4 \text{ THEN } \{7, 8\} \text{ ELSE} \\
 &\quad \text{IF } i = 5 \text{ THEN } \{\} \text{ ELSE} \\
 &\quad \text{IF } i = 6 \text{ THEN } \{4\} \text{ ELSE} \\
 &\quad \text{IF } i = 7 \text{ THEN } \{5\} \text{ ELSE} \\
 &\quad \text{IF } i = 8 \text{ THEN } \{5, 2\} \text{ ELSE} \\
 &\quad \{\} \\
 &\quad ]
 \end{aligned}$$

$$\begin{aligned}
 Init &\triangleq p = 1 \\
 M(i) &\triangleq \wedge i \in l[p] \\
 &\quad \wedge p' = i \\
 Next &\triangleq \exists i \in 1..n : M(i)
 \end{aligned}$$

#### ◇- Solution de l'exercice 7

Modèle solution TLA<sup>+</sup> :  
 ../tlamodels/malgtdlex7.tla

----- MODULE malgtdlex7 -----

EXTENDS Integers, TLC

VARIABLES p

CONSTANTS input, output

n == 10

nodes == 1..n

```

l == [i \in 1..n |->
  IF i=1 THEN {4,5} ELSE
  IF i=2 THEN {6,7,10} ELSE
  IF i=4 THEN {7,8} ELSE
  IF i=5 THEN {} ELSE
  IF i=6 THEN {4} ELSE
  IF i = 7 THEN {5} ELSE
  IF i = 8 THEN {5, 2} ELSE
  {}
]
```

```

lab == [<x,y> \in (nodes \X nodes) |->
  IF x=1 /\ y=1 THEN {<<2,1>>} ELSE
  IF x=1 /\ y=2 THEN {<<1,3>>, <<2,2>>} ELSE
  IF x=2 /\ y=1 THEN {<<1,1>>, <<2,2>>} ELSE
  IF x=2 /\ y=2 THEN {<<1,2>>, <<2,1>>} ELSE
  IF x=1 /\ y=3 THEN {<<2,3>>, <<1,4>>} ELSE
  IF x=1 /\ y=4 THEN {<<2,4>>, <<1,5>>} ELSE
  IF x=1 /\ y=5 THEN {<<2,5>>, <<1,4>>} ELSE
  IF x=1 /\ y=6 THEN {<<2,6>>, <<1,7>>} ELSE
  IF x=1 /\ y=7 THEN {<<1,6>>, <<1,8>>} ELSE
  IF x=1 /\ y=8 THEN {<<2,8>>, <<1,7>>}
  ELSE {}
]
```

```

Init == p = 1
M(i) == /\ i \in l[p]
        /\ p'=i
Next == \E i \in 1..n: M(i)

Initlab == p = <<1,1>>
ML(q) == /\ q \in lab[p]
        /\ p'=q
Nextlab == \E q \in nodes \X nodes : ML(q)

Sortie == p \notin output

```

**Fin 7**

**Exercice 8** (*malgtd1ex10,malgtd1ex10bis,malgtd1ex10ter,malgtd1ex10last*)

Pour montrer que chaque annotation est correcte ou incorrecte, on propose de procéder comme suit :

- Traduire cette annotation sous la forme d'un contrat.
- Vérifier les conditions de vérification du contrat

$$\begin{array}{l} \ell_1 : P_{\ell_1}(v) \\ v := f(v, c) \\ \ell_2 : P_{\ell_2}(v) \end{array}$$

variables v
requires $pre(v_0)$
ensures $post(v_0, v_f)$
begin
$\ell_1 : Q_1(v_0, v)$
$v := f(v, c)$
$\ell_2 : Q_2(v_0, v)$
end

- $pre(v_0) \equiv P_{\ell_1}(v_0)$
- $post(v_0, v_f) \equiv P_{\ell_2}(v_f)$ .
- $Q_{\ell_1}(v_0, v) \equiv P_{\ell_1}(v) \wedge v = v_0$
- $Q_{\ell_2}(v_0, v) \equiv P_{\ell_2}(v)$

On rappelle qu'un contrat est valide si les trois conditions suivantes sont valides :

- (*init*)  $pre(v_0) \wedge v = v_0 \Rightarrow Q_1(v_0, v)$
- (*concl*)  $pre(v_0) \wedge Q_2(v_0, v) \Rightarrow post(v_0, v)$
- (*induct*)  $pre(v_0) \wedge Q_1(v_0, v) \wedge cond_{\ell_1, \ell_2}(v) \wedge v' = f(v, c) \Rightarrow Q_2(v_0, v')$

Les deux propriétés (*init*) et (*concl*) sont valides par construction et la seule propriété à montrer correcte ou incorrecte est la propriété (*induct*).

**Question 8.1** (*malgtd1ex10*)

$$\begin{array}{l} \ell_1 : x = 3 \wedge y = z+x \wedge z = 2 \cdot x \\ y := z+x \\ \ell_2 : x = 3 \wedge y = x+6 \end{array}$$

◇— **Solution de la question 8.1**

Modèle solution TLA<sup>+</sup> :

../tlamodels/malgtd1ex10.tla

```

----- MODULE malgtd1ex10 -----
EXTENDS Naturals, Integers, TLC, TLAPS
CONSTANTS x0, y0, z0
VARIABLES x, y, z, pc

```

```

-----
(* Auxiliary definitions *)
typeInt(u) == u \in Int
pre(u,v,w) ==
    /\ u \in Int /\ v \in Int /\ w \in Int
    /\ u=3 /\ v=w+u /\ w=2*u

L == {"l1","l2"}

ppre == pre(x0,y0,z0)
-----
(* Interpretation: we assume that the precondition can hold and we have to find pos
ASSUME ppre
-----

(* Action for transition of the algorithm *)
all12 ==
    /\ pc="l1"
    /\ pc'="l2"
    /\ y'=z+x
    /\ z'=z /\ x'=x
-----

(* Computations *)
vars == <<x,y,z,pc>>
Next == all12 \/ UNCHANGED vars
Init == pc="l0" /\ x=x0 /\ y=y0 /\ z = z0 /\ pre(x0,y0,z0)
-----

(* Checking the annotation by checking the invariant i derived from the annotation
i ==
    /\ typeInt(x) /\ typeInt(y) /\ typeInt(z) /\ pc \in L
    /\ pc="l1" => x=x0 /\ y=y0 /\ z=z0 /\ pre(x0,y0,z0)
    /\ pc="l2" => x=3 /\ y = x +6 /\ pre(x0,y0,z0)

Safe == i

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

```

## Fin 8.1

On définit un contrat comme suit :

variables  $x, y, z$   
 requires  $x_0 = 3 \wedge y_0 = z_0 + x_0 \wedge z_0 = 2 \cdot x_0$   
 ensures  $x_f = 3 \wedge y_f = x_f + 6$

begin  
 $\ell_1 : x = 3 \wedge y = z + x \wedge z = 2 \cdot x$   
 $y := z + x$   
 $\ell_2 : x = 3 \wedge y = x + 6$   
 end

On pose les assertions suivantes à partir de l'annotation :

- $pre(x_0, y_0, z_0) \stackrel{def}{=} x_0 = 3 \wedge y_0 = z_0 + x_0 \wedge z_0 = 2 \cdot x_0$
- $prepost(x_0, y_0, z_0, x, y, z) \stackrel{def}{=} x = 3 \wedge y = x + 6$
- $Q_1(x_0, y_0, z_0, x, y, z) \stackrel{def}{=} x = 3 \wedge y = z + x \wedge z = 2 \cdot x$
- $Q_2(x_0, y_0, z_0, x, y, z) \stackrel{def}{=} x = 3 \wedge y = x + 6$

On établit les trois conditions pour valider le contrat :

- *(init)*  $pre(x_0, y_0, z_0) \wedge (x, y, z) = (x_0, y_0, z_0) \Rightarrow Q_1(x_0, y_0, z_0, x, y, z)$
- *(concl)*  $pre(v_0) \wedge Q_2(x_0, y_0, z_0, x, y, z) \Rightarrow prepost(x_0, y_0, z_0, x, y, z)$
- *(induct)*  $pre(x_0, y_0, z_0) \wedge Q_1(x_0, y_0, z_0, x, y, z) \wedge TRUE \wedge (x', y', z') = (x, z+x, z) \Rightarrow Q_2(x_0, y_0, z_0, x', y', z')$
- $pre(x_0, y_0, z_0) \wedge Q_1(x_0, y_0, z_0, x, y, z) \wedge TRUE \wedge (x', y', z') = (x, z+x, z) \Rightarrow Q_2(x_0, y_0, z_0, x', y', z')$
- $\vdash pre(x_0, y_0, z_0) \wedge Q_1(x_0, y_0, z_0, x, y, z) \wedge TRUE \wedge (x', y', z') = (x, z+x, z) \Rightarrow Q_2(x_0, y_0, z_0, x', y', z')$
- $pre(x_0, y_0, z_0) \wedge Q_1(x_0, y_0, z_0, x, y, z) \wedge TRUE \wedge (x', y', z') = (x, z+x, z) \vdash Q_2(x_0, y_0, z_0, x', y', z')$
- $pre(x_0, y_0, z_0), Q_1(x_0, y_0, z_0, x, y, z), TRUE, (x', y', z') = (x, z+x, z) \vdash Q_2(x_0, y_0, z_0, x', y', z')$
- $x_0 = 3 \wedge y_0 = z_0 + x_0, z_0 = 2 \cdot x_0, Q_1(x_0, y_0, z_0, x, y, z), TRUE, (x', y', z') = (x, z+x, z) \vdash$   
 $Q_2(x_0, y_0, z_0, x', y', z')$
- $x_0 = 3 \wedge y_0 = z_0 + x_0, z_0 = 2 \cdot x_0, x = 3 \wedge y = z + x \wedge z = 2 \cdot x, TRUE, (x', y', z') =$   
 $(x, z+x, z) \vdash Q_2(x_0, y_0, z_0, x', y', z')$
- $x_0 = 3 \wedge y_0 = z_0 + x_0, z_0 = 2 \cdot x_0, x = 3 \wedge y = z + x \wedge z = 2 \cdot x, TRUE, (x', y', z') =$   
 $(x, z+x, z) \vdash x' = 3 \wedge y' = x' + 6$
- $x_0 = 3, y_0 = z_0 + x_0, z_0 = 2 \cdot x_0, x = 3, y = z + x, z = 2 \cdot x, TRUE, (x', y', z') = (x, z+x, z) \vdash$   
 $x' = 3 \wedge y' = x' + 6$
- $x_0 = 3, y_0 = z_0 + x_0, z_0 = 2 \cdot x_0, x = 3, y = z + x, z = 2 \cdot x, TRUE, (x', y', z') = (x, z+x, z) \vdash$   
 $x' = 3$
- $x_0 = 3, y_0 = z_0 + x_0, z_0 = 2 \cdot x_0, x = 3, y = z + x, z = 2 \cdot x, TRUE, (x', y', z') = (x, z+x, z) \vdash$   
 $x' = 3$
- $x_0 = 3, y_0 = z_0 + x_0, z_0 = 2 \cdot x_0, x = 3, y = z + x, z = 2 \cdot x, TRUE, (x', y', z') = (x, z+x, z) \vdash$   
 $x = 3$
- $x = 3$  est une hypothèse à gauche. Le séquent est valide.
- $x_0 = 3, y_0 = z_0 + x_0, z_0 = 2 \cdot x_0, x = 3, y = z + x, z = 2 \cdot x, TRUE, (x', y', z') = (x, z+x, z) \vdash$   
 $y' = x' + 6$
- $x_0 = 3, y_0 = z_0 + x_0, z_0 = 2 \cdot x_0, x = 3, y = z + x, z = 2 \cdot x, TRUE, (x', y', z') = (x, z+x, z) \vdash$   
 $y' = x' + 6$
- $x_0 = 3, y_0 = z_0 + x_0, z_0 = 2 \cdot x_0, x = 3, y = z + x, z = 2 \cdot x, TRUE, (x', y', z') = (x, z+x, z) \vdash$   
 $y' = x + 6$
- $x_0 = 3, y_0 = z_0 + x_0, z_0 = 2 \cdot x_0, x = 3, y = z + x, z = 2 \cdot x, TRUE, (x', y', z') = (x, z+x, z) \vdash$   
 $z + x = x + 6$
- $x_0 = 3, y_0 = z_0 + x_0, z_0 = 2 \cdot x_0, x = 3, y = z + x, z = 2 \cdot x, TRUE, (x', y', z') = (x, z+x, z) \vdash$   
 $2 \cdot x + x = x + 6$
- $x_0 = 3, y_0 = z_0 + x_0, z_0 = 2 \cdot x_0, x = 3, y = z + x, z = 2 \cdot x, TRUE, (x', y', z') = (x, z+x, z) \vdash$   
 $2 \cdot 3 + 3 = 3 + 6$
- $x_0 = 3, y_0 = z_0 + x_0, z_0 = 2 \cdot x_0, x = 3, y = z + x, z = 2 \cdot x, TRUE, (x', y', z') = (x, z+x, z) \vdash$   
 $9 = 9$
- *Réflexivité de l'égalité.*

### Question 8.2 (malgtd1ex10bis)

Pour les deux exemples qui suivent, on considère deux cas et on doit donner une interprétation.

$$\begin{aligned} \ell_1 : x = 2^4 \wedge y = 2 \wedge x \cdot y = 2^6 \\ x := y + x + 2^x \\ \ell_2 : x = 2^{10} \wedge y = 2 \end{aligned}$$

$$\begin{aligned} \ell_1 : x = 2^4 \wedge y = 2 \wedge x \cdot y = 2^5 \\ x := y + x + 2^x \\ \ell_2 : x = 2^{10} \wedge y = 2 \end{aligned}$$

### ◊ Solution de la question 8.2

Modèle solution TLA<sup>+</sup> :

../tlamodels/malgtd1ex10bis.tla

----- MODULE malgtd1ex10bis -----

```

EXTENDS Naturals, Integers, TLC
CONSTANTS x0, y0
VARIABLES  x, y, pc
-----

(* Interpretation: w assume that the precondition can hold and we have to find poss
ASSUME /\ x0 \in Int /\ y0 \in Int
      /\  x0=2^4 /\ y0=2 /\ x0*y0=2^5
-----

(* Auxiliary definitions *)
typeInt(u) == u \in Int
pre == /\ x0 \in Int /\ y0 \in Int
      /\  x0=2^4 /\ y0=2 /\ x0*y0=2^5
-----

(* Action for transitioon of the algorithm *)
all12 ==
  /\ pc="l1"
  /\ pc'="l2"
  /\ x'=y+x+2^x
  /\ y'=y
-----

(* Computations *)
Next == all12  \/ UNCHANGED <<x,y,pc>>
Init == pc="l1" /\ x=x0 /\ y=y0 /\ pre
-----

(* Checking the annotation by checking the invariant i derived from the annotation
i ==
  /\ typeInt(x) /\ typeInt(y)
  /\ pc="l1" =>  x=x0 /\ y=y0 /\ pre
  /\ pc="l2" =>  x=2^10 /\ y = 2 /\ PrintT(x)

safe ==  i
=====

\* Modification History
\* Last modified Tue Feb 07 11:35:34 CET 2023 by mery
\* Created Wed Sep 09 18:19:08 CEST 2015 by mery

```

**Fin 8.2**

### Question 8.3 (*malgtd1ex10ter.tla*)

$$\begin{aligned} \ell_1 : x = 1 \wedge y = 12 \\ x := 2 \cdot y + x \\ \ell_2 : x = 1 \wedge y = 25 \end{aligned}$$

### ◇ Solution de la question 8.3

Modèle solution  $TLA^+$  :

../tlamodels/malgtd1ex10ter.tla

```

----- MODULE malgtd1ex10ter -----

```

```

EXTENDS Naturals, Integers, TLC
CONSTANTS x0, y0
VARIABLES  x, y, pc

```



```

-----
(* Interpretation: w assume that the precondition can hold and we have to find poss
ASSUME /\ x0 \in Int /\ y0 \in Int
      /\ x0=1 /\ y0=12
-----

(* Auxiliary definitions *)
typeInt(u) == u \in Int
pre == /\ x0 \in Int /\ y0 \in Int
      /\ x0=1 /\ y0=12
-----

(* Action for transitioon of the algorithm *)
all12 ==
  /\ pc="l1"
  /\ pc'="l2"
  /\ x'=2*y+x
  /\ y'=y
-----

(* Computations *)
Next == all12  \/ UNCHANGED <<x,y,pc>>
Init == pc="l1" /\ x=x0 /\ y =y0  /\ pre
-----

(* Checking the annotation by checking the invariant i derived from the annotation
i ==
  /\ typeInt(x) /\ typeInt(y)
  /\ pc="l1" =>  x=x0 /\ y=y0 /\ pre
  /\ pc="l2" =>  x=25 /\ y =  y0 /\ PrintT(x)

safe ==  i
=====

\* Modification History
\* Last modified Wed Mar 06 10:41:19 CET 2024 by mery
\* Created Wed Sep 09 18:19:08 CEST 2015 by mery

```

**Fin 8.3**

#### Question 8.4 (*malgtd1ex10last.tla*)

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

#### ◇— Solution de la question 8.4

Modèle solution  $TLA^+$  :

*../tlamodels/malgtd1ex10last.tla*

```

----- MODULE malgtd1ex10last -----

EXTENDS Naturals, Integers, TLC
CONSTANTS x0,y0,z0,UND
VARIABLES  x,y,z,pc

```

```

-----
(* Auxiliary definitions *)
typeInt(u) == u \in Int

```

```

pre == /\ x0 \in Int /\ y0 \in Int
      /\  x0=11 /\ y0=13 /\ z0 = UND
-----
(* Interpretation: we assume that the precondition can hold and we have to find poss
ASSUME  pre

-----

(* Action for transition of the algorithm *)
all12 ==
  /\ pc="l1"
  /\ pc'="l2"
  /\ z'=x
  /\ x' = z
  /\ y' = z'

-----

(* Computations *)
Next == all12  \/ UNCHANGED <<x,y,z,pc>>
Init == pc="l1" /\ x=x0 /\ y=y0 /\ z = z0 /\ pre
-----
(* Checking the annotation by checking the invariant i derived from the annotation
i ==
  /\ pc="l1" =>  x=x0 /\ y=y0 /\ pre
  /\ pc="l2" =>  x=26 \div 2 /\ y = 33 \div 3

safe ==  i

=====
\* Modification History
\* Last modified Wed Feb 23 08:31:14 CET 2022 by mery
\* Created Wed Sep 09 18:19:08 CEST 2015 by mery

```

**Fin 8.4**

## TD4

### Exercice 9 (*malgtd1ex11,pluscal\_max.tla*)

**Question 9.1** *Ecrire un module  $TLA^+$  qui traduit la relation de transition de cet algorithme selon les instructions. Pour cela, vous utiliserez la fonctionnalité offerte par la traduction d'un algorithme PlusCal en  $TLA^+$ .*

**Question 9.2** *Compléter l'algorithme 9 en l'annotant.*

◇— **Solution de la question 9.2** \_\_\_\_\_

**Annotation** *L'annotation de cet algorithme est donnée à la référence d'algorithme 9 et la figure est placée au gré de  $\LaTeX$  sous le numéro 9.*

**Fin 9.2**

**Question 9.3** *Vérifier que l'annotation est correcte.*

◇— **Solution de la question 9.3** \_\_\_\_\_

Modèle solution  $TLA^+$  :  
*../tla/models/malgtd1ex11.tla*

```
Variables : X,Y,Z
Requires :  $x_0, y_0 \in \mathbb{N} \wedge z_0 \in \mathbb{Z}$ 
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:** maximum de deux nombres non annotée

```
Variables : X,Y,Z
Requires :  $x_0, y_0 \in \mathbb{N} \wedge z_0 \in \mathbb{Z}$ 
Ensures :  $z_f = \max(x_0, y_0)$ 

 $\ell_0 : \{x = x_0 \wedge y = y_0 \wedge z = z_0 \wedge x_0, y_0 \in \mathbb{N} \wedge z_0 \in \mathbb{Z}\}$ 
if  $X < Y$  then
  |  $\ell_1 : \{x < y \wedge x = x_0 \wedge y = y_0 \wedge z = z_0 \wedge x_0, y_0 \in \mathbb{N} \wedge z_0 \in \mathbb{Z}\}$ 
  |  $Z := Y;$ 
  |  $\ell_2 : \{x < y \wedge x = x_0 \wedge y = y_0 \wedge x_0, y_0 \in \mathbb{N} \wedge z_0 \in \mathbb{Z} \wedge z = y_0\}$ 
else
  |  $\ell_3 : \{x \geq y \wedge x = x_0 \wedge y = y_0 \wedge z = z_0 \wedge x_0, y_0 \in \mathbb{N} \wedge z_0 \in \mathbb{Z}\}$ 
  |  $Z := X;$ 
  |  $\ell_4 : \{x \geq y \wedge x = x_0 \wedge y = y_0 \wedge x_0, y_0 \in \mathbb{N} \wedge z_0 \in \mathbb{Z} \wedge z = x_0\}$ 
  |
;
 $\ell_5 : \{z = \max(x_0, y_0) \wedge x = x_0 \wedge y = y_0 \wedge x_0, y_0 \in \mathbb{N} \wedge z_0 \in \mathbb{Z}\}$ 
```

**Algorithme 2:** maximum de deux nombres non annotée

**Variables** : X,Y,Z

**Requires** :  $x_0, y_0 \in \mathbb{N} \wedge z_0 \in \mathbb{Z}$

**Ensures** :  $z_f = \max(x_0, y_0)$

$\ell_0 : \{x = x_0 \wedge y = y_0 \wedge z = z_0 \wedge x_0, y_0 \in \mathbb{N} \wedge z_0 \in \mathbb{Z}\}$

**if**  $X < Y$  **then**

$\ell_1 : \{x < y \wedge x = x_0 \wedge y = y_0 \wedge z = z_0 \wedge x_0, y_0 \in \mathbb{N} \wedge z_0 \in \mathbb{Z}\}$

$Z := Y;$

$\ell_2 : \{x < y \wedge x = x_0 \wedge y = y_0 \wedge x_0, y_0 \in \mathbb{N} \wedge z_0 \in \mathbb{Z} \wedge z = y_0\}$

**else**

$\ell_3 : \{x \geq y \wedge x = x_0 \wedge y = y_0 \wedge z = z_0 \wedge x_0, y_0 \in \mathbb{N} \wedge z_0 \in \mathbb{Z}\}$

$Z := X;$

$\ell_4 : \{x \geq y \wedge x = x_0 \wedge y = y_0 \wedge x_0, y_0 \in \mathbb{N} \wedge z_0 \in \mathbb{Z} \wedge z = x_0\}$

;

$\ell_5 : \{z = \max(x_0, y_0) \wedge x = x_0 \wedge y = y_0 \wedge x_0, y_0 \in \mathbb{N} \wedge z_0 \in \mathbb{Z}\}$

### Algorithme 3: maximum de deux nombres non annotée

----- MODULE malgtdlex11 -----

EXTENDS Naturals, Integers

CONSTANTS x0,y0,z0,mini0,maxi0

VARIABLES x,y,z,pc

typeInt(u) == u \in Int

maxi(u,v) == IF u < v THEN v ELSE u

pre == x0 \in Nat /\ y0 \in Nat /\ z0 \in Int

ASSUME pre

-----

a1011 ==

    /\ pc="10"

    /\ pc'="11"

    /\ x<y

    /\ z'=z /\ x'=x /\ y'=y

a1112 ==

    /\ pc="11"

    /\ pc'="12"

    /\ z'=y

    /\ x'=x /\ y'=y

a1215 ==

    /\ pc="12"

    /\ pc'="15"

    /\ z'=z /\ x'=x /\ y'=y

a1013 ==

    /\ pc="10"

    /\ pc'="13"

    /\ x \geq y

    /\ UNCHANGED <<z,x,y>>

a1314 ==

    /\ pc="13"

    /\ pc'="14"

    /\ z'=x

    /\ x'=x /\ y'=y

a1415 ==

```

/\ pc="l4"
/\ pc'="l5"
/\ z'=z /\ x'=x /\ y'=y
-----
Next == a10l1 /\ a1l12 /\ a12l5 /\ a10l3 /\ a13l4 /\ a14l5 /\ UNCHANGED <<x,y,z,pc
Init == pc="l0" /\ x=x0 /\ y=y0 /\ z=z0
-----
i ==
  /\ typeInt(x) /\ typeInt(y) /\ typeInt(z)
  /\ pc="l0" => x=x0 /\ y=y0 /\ z=z0 /\ pre
  /\ pc="l1" => x<y /\ x=x0 /\ y=y0 /\ z=z0 /\ pre
  /\ pc="l2" => x<y /\ x=x0 /\ y=y0 /\ z=y0 /\ pre
  /\ pc="l3" => x \geq y /\ x=x0 /\ y=y0 /\ z=z0 /\ pre
  /\ pc="l4" => x \geq y /\ x=x0 /\ y=y0 /\ z=x0 /\ pre
  /\ pc="l5" => z = maxi(x0,y0) /\ x=x0 /\ y=y0 /\ pre
safe pc == pc="l5" => z = maxi(x0,y0)
safe ab == x=x0 /\ y=y0
safe rte ==
  /\ mini0 <= x /\ x <= maxi0
  /\ mini0 <= y /\ y <= maxi0
=====
\* Modification History
\* Last modified Wed Feb 08 10:52:27 CET 2023 by mery
\* Created Wed Sep 09 18:19:08 CEST 2015 by mery

```

### Fin 9.3

**Question 9.4** Compléter le module  $TLA^+$  en définissant l'invariant construit avec les annotations et vérifier le contrat.

**Exercice 10** Montrer que chaque annotation est correcte ou incorrecte selon les conditions de vérifications énoncées comme suit

$\forall x, y, x', y'. P_\ell(x, y) \wedge \text{cond}_{\ell, \ell'}(x, y) \wedge (x', y') = f_{\ell, \ell'}(x, y) \Rightarrow P_{\ell'}(x', y')$

—	$\begin{aligned} \ell_1 : x = 10 \wedge y = z + x \wedge z = 2 \cdot x \\ y := z + x \\ \ell_2 : x = 10 \wedge y = x + 2 \cdot 10 \end{aligned}$	—	$\begin{aligned} \ell_1 : x = 1 \wedge y = 12 \\ x := 2 \cdot y \\ \ell_2 : x = 1 \wedge y = 24 \end{aligned}$
—	<p>On suppose que <math>p</math> est un nombre premier :</p> $\begin{aligned} \ell_1 : x = 2^p \wedge y = 2^{p+1} \wedge x \cdot y = 2^{2 \cdot p + 1} \\ x := y + x + 2^x \\ \ell_2 : x = 5 \cdot 2^p \wedge y = 2^{p+1} \end{aligned}$	—	$\begin{aligned} \ell_1 : x = 11 \wedge y = 13 \\ z := x; x := y; y := z; \\ \ell_2 : x = 26/2 \wedge y = 33/3 \end{aligned}$

On rappelle qu'un contrat pour la correction partielle d'un petit programme est donné par les éléments ci-dessous en colonne de gauche et que les conditions de vérification associées sont définies par le texte de la colonne de droite.

Contrat de la correction  
partielle

variables <i>type</i> $X$	
definitions	
$def1 \stackrel{def}{=} text1$	
requires $pre(x_0)$	
ensures $post(x_0, x_f)$	
<div> <div>begin</div> <div> <math>0 : P_0(x_0, x)</math>  <math>instruction_0</math>  <math>1 : P_i(x_0, x)</math>  <math>instruction_1</math>  <math>f : P_f(x_0, x)</math>  end </div> </div>	<div> <div>Conditions de vérification</div> <div> <ul style="list-style-type: none"> <li>— <math>pre(x_0) \wedge x = x_0 \Rightarrow P_0(x_0, x)</math></li> <li>— <math>pre(x_0) \wedge P_f(x_0, x) \Rightarrow post(x_0, x)</math></li> <li>— Pour toutes les paires <math>\ell, \ell'</math>, telles que <math>\ell \longrightarrow \ell'</math>, on vérifie que, pour toutes les valeurs <math>x, x' \in \text{MEMORY}</math> <math display="block">\left( \begin{array}{l} pre(x_0) \wedge P_\ell(x_0, x) \\ \wedge cond_{\ell, \ell'}(x) \wedge x' = f_{\ell, \ell'}(x) \end{array} \right) \Rightarrow P_{\ell'}(x_0, x')</math> </li> </ul> </div> </div>

**Exercice 11** (prog23-4.c)

Soit le contrat suivant qui met en jeu les variables  $X, Y, Z, C, R$ .

<b>VARIABLES</b> int $X, Y, Z, C, R$
<b>REQUIRES</b> $x_0, y_0, z_0, c_0, r_0 \in \mathbb{Z}$
<b>ENSURES</b> $r_f = 0$
<div> <div><b>BEGIN</b></div> <div> <math>0 : x = x_0 \wedge y = y_0 \wedge z = z_0 \wedge c = c_0 \wedge r = r_0 \wedge x_0, y_0, z_0, c_0, r_0 \in \mathbb{Z}</math>  <math>(X, Z, Y) := (49, 2 \cdot C, (2 \cdot C + 1) \cdot (2 \cdot C + 1));</math>  <math>1 : x = 49 \wedge z = 2 \cdot c \wedge y = (z + 1) \cdot (z + 1)</math>  <math>Y := X + Z + 1;</math>  <math>2 : x = 49 \wedge z = 2 \cdot c \wedge y = (c + 1) \cdot (c + 1)</math>  <div><b>END</b></div> </div> </div>

**Question 11.1** Ecrire les conditions de vérification associée au contrat ci-dessus en vous aidant du rappel de la définition de ces conditions de vérification.

**Question 11.2** Simplifier les conditions de vérification et préciser les conditions que doivent vérifier les valeurs initiales des variables  $X, Y, Z, C, R$  pour que les conditions de vérification soient toutes vraies. En particulier, il faudra s'assurer que la précondition est satisfaisable.

**Exercice 12** ()

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  $n_0, x_0, b_0 \in \mathbb{Z}$  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 12.1** Donner l'assertion Requies en complétant ce qui est déjà mentionné et en reportant le texte complet de cette assertion Requies dans votre copie.

On rappelle que la relation de transition de  $\ell$  vers  $\ell'$ , notée  $a(\ell, \ell')$ , est définie par une relation de la forme  $cond_{\ell, \ell'}(v) \wedge v' = f_{\ell, \ell'}(v)$ .

**Question 12.2** Ecrire les relations de transition entre les étiquettes successives :  $a(\ell_0, \ell_1)$ ,  $a(\ell_1, \ell_2)$ ,  $a(\ell_2, \ell_3)$ ,  $a(\ell_3, \ell_6)$ ,  $a(\ell_1, \ell_4)$ ,  $a(\ell_4, \ell_5)$ ,  $a(\ell_5, \ell_6)$ .

**VARIABLES**  $int\ N, X, B$

**REQUIRES**  $n_0, x_0, b_0 \in \mathbb{Z}$

**ENSURES**  $\left( \begin{array}{l} n_0 < b_0 \Rightarrow x_f = \text{question1} \\ n_0 \geq b_0 \Rightarrow x_f = \text{question1} \\ n_f = n_0 \wedge b_f = b_0 \end{array} \right)$

**BEGIN**

$\ell_0 :$

$X := N;$

$\ell_1 :$

**IF**  $X < B$  **THEN**

$\ell_2 :$

$X := X \cdot X + 2 \cdot B \cdot X + B \cdot B;$

$\ell_3 :$

**ELSE**

$\ell_4 :$

$X := B;$

$\ell_5 :$

**FI**

$\ell_6 :$

**END**

**Exercice 13** (squareroot)

On considère l'algorithme squareroot calculant la racine carrée entière d'un nombre naturel  $x \in \mathbb{N}$ .

**VARIABLES**  $X, Y1, Y2, Y3, Z$

$pre(x0, y10, y20, y30, z0) \stackrel{def}{=}$

$U \stackrel{def}{=} (X, Y1, Y2, Y3, Z)$

$u0 \stackrel{def}{=} (x0, y10, y20, y30, z0)$

$post(x0, y10, y20, y30, z0, xf, y1f, y2f, y3f, zf) \stackrel{def}{=}$

**REQUIRES**  $pre(x0, y10, y20, y30, z0)$

**ENSURES**  $post(x0, y10, y20, y30, z0, xf, y1f, y2f, y3f, zf)$

$\ell_0 : pre(u0) \wedge u = u0$

$(Y1, Y2, Y3) := (0, 1, 1)$

$\ell_1 : pre(u0) \wedge x = x0 \wedge z = z0 \wedge y2 = (y1+1) \cdot (y1+1) \wedge y3 = 2 \cdot y1+1 \wedge y1 \cdot y1 \leq x$

**WHILE**  $Y2 \leq X$  **DO**

$\ell_2 :$

$(Y1, Y2, Y3) := (Y1+1, Y2+Y3+2, Y3+2);$

$\ell_3 : OD;$

$\ell_4 :$

$Z := Y1;$

$\ell_5 :$

**Question 13.1** Définir les deux assertions pre et post qui établissent le contrat de cet algorithme.

**Question 13.2** Complétez cet algorithme en proposant trois assertions :

- $P_{\ell_2}(u0, u)$
- $P_{\ell_3}(u0, u)$
- $P_{\ell_4}(u0, u)$
- $P_{\ell_5}(u0, u)$

Pour cela, le plus efficace est de définir clairement les conditions de vérifications : pour chaque paire  $(\ell, \ell')$  d'étiquettes correspondant à un pas élémentaire ; on vérifie la propriété suivante :

$$P_{\ell}(u0, u) \wedge \text{cond}_{\ell, \ell'}(u) \wedge u' = f_{\ell, \ell'}(u) \Rightarrow P_{\ell'}(u')$$

Enoncez et vérifiez cette propriété pour les paires d'étiquettes suivantes :  $(\ell_1, \ell_2); (\ell_1, \ell_4); (\ell_2, \ell_3); (\ell_3, \ell_2); (\ell_3, \ell_4); (\ell_4, \ell_5);$

**Question 13.3** Finalisez les vérifications en montrant que les conditions de vérification pour un contrat sont toutes vérifiées.

**Question 13.4** On suppose que toutes les conditions de vérifications associées aux paires d'étiquettes successives de l'algorithme sont vérifiées. Quelles sont les deux conditions à montrer pour déduire que l'algorithme est partiellement correct par rapport aux pré et post conditions ? Vous donnerez explicitement les conditions et vous expliquerez pourquoi elles sont correctes. <

**Question 13.5** Expliquer que cet algorithme est sans erreurs à l'exécution, si les données initiales sont dans un domaine à définir inclus dans le domaine des entiers informatiques c'est-à-dire les entiers codables sur  $n$  bits. L'ensemble des entiers informatiques sur  $n$  bits est l'ensemble noté  $\mathbb{Z}_n$  et défini par  $\{i | i \in \mathbb{Z} \wedge -2^{n-1} \leq i \wedge i \leq 2^{n-1}-1\}$ .

L'algorithme annoté est décrit par l'algorithme ??

VARIABLES  $X, Y1, Y2, Y3, Z$

$$\text{pre}(x0, y10, y20, y30, z0) \stackrel{\text{def}}{=} \begin{cases} x0 \in \mathbb{N} \wedge x0 \neq 0 \\ y10, y20, y30, z0 \in \mathbb{Z} \end{cases}$$

$$U \stackrel{\text{def}}{=} (X, Y1, Y2, Y3, Z)$$

$$u0 \stackrel{\text{def}}{=} (x0, y10, y20, y30, z0)$$

$$\text{post}(x0, y10, y20, y30, z0, xf, y1f, y2f, y3f, zf) \stackrel{\text{def}}{=} (zf^2 \leq x0 \wedge x0 \leq (zf+1)^2)$$

$$\text{pre}(u0) \stackrel{\text{def}}{=} \text{pre}(x0, y10, y20, y30, z0)$$

$$(u = u0) \stackrel{\text{def}}{=} x = x0 \wedge y1 = y10 \wedge y2 = y20 \wedge y3 = y30 \wedge z = z0$$

REQUIRES  $\text{pre}(x0, y10, y20, y30, z0)$

ENSURES  $\text{post}(x0, y10, y20, y30, z0, xf, y1f, y2f, y3f, zf)$

$$\ell_0 : \text{pre}(u0) \wedge u = u0$$

$$(Y1, Y2, Y3) := (0, 1, 1)$$

$$\ell_1 : \text{pre}(u0) \wedge x = x0 \wedge z = z0 \wedge y2 = (y1+1) \cdot (y1+1) \wedge y3 = 2 \cdot y1+1 \wedge y1 \cdot y1 \leq x$$

WHILE  $Y2 \leq X$  DO

$$\ell_2 : P_{\ell_1}(u0, u) \wedge y2 \leq x$$

$$(Y1, Y2, Y3) := (Y1+1, Y2+Y3+2, Y3+2);$$

$$\ell_3 : P_{\ell_1}(u0, u)$$

OD;

$$\ell_4 : P_{\ell_1}(u0, u) \wedge y2 > x$$

$$Z := Y1;$$

$$\ell_5 : \text{pre}(u0) \wedge x = x0 \wedge y2 = (y1+1) \cdot (y1+1) \wedge y3 = 2 \cdot y1+1 \wedge y1 \cdot y1 \leq x \wedge x < y2$$

Fin de la série 1