Cours MOdélisation, Vérification et EXpérimentations
Exercices (avec les corrections)
Série 1 Annotation, modélisation, vérification - Validation en TLA$^+$
par Dominique Méry
12 février 2025

TD1

Exercice 1 *(malgtd1ex1)*
Le PGCD de deux nombres vérifie les propriétés suivantes :

- $\forall a, b \in \mathbb{N}.pgcd(a, b) = pgcd(b, a)$
- $\forall a, b \in \mathbb{N}.pgcd(a, a+b) = pgcd(a, b)$
- *Ecrire une spécification TLA$^+$ 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$^+$:
../tlamodels/malgtd1ex1.tla

```
------------------- MODULE malgtd1ex1 -----------------
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

-----------------------------------
(* PropriÃ©tÃ©s de sÃ»retÃ© Ã  vÃ©rifier *)
test == x # y
prop1 == x \geq 0 \* ok
prop2 == x+y \leq a+b \* ok
tocheck == prop1 /\ prop2
=========================================
```

Fin 1

Exercice 2 *(malgtd1ex2)*
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/malgtd1ex2.tla

```
------------------------------- MODULE malgtd1ex2 ------------------------------
(* 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 *(malgtd1ex3)*
On considère l'algorithme suivant décrit par un organigramme ou flowchart de la figure 1.

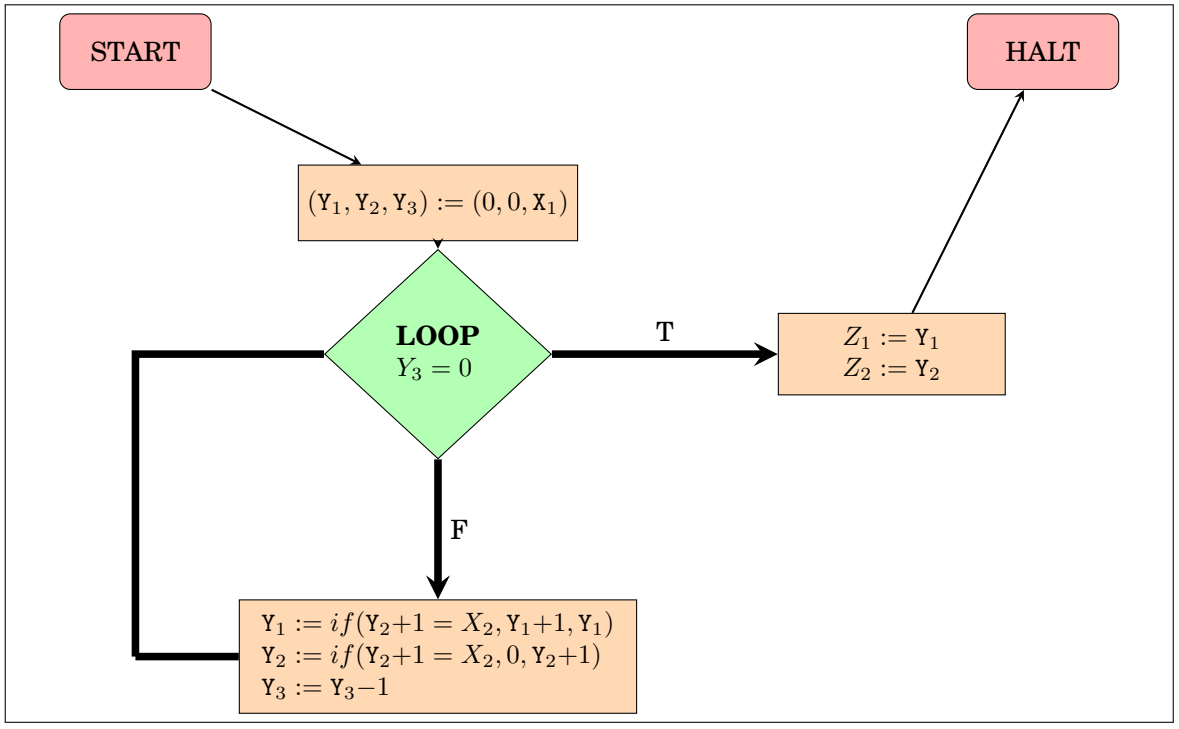


FIGURE 1 – Organigramme de calcul de la division entière

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 $f91$ *est définie pour tout entier* x $f91(x) = if\ x > 100\ then\ x{-}10\ else\ 91\ fi$.

Question 4.1 *Définir le contrat é'tablissant la correction partielle de l'algorithme* $ALG91$ *de la figure 2 qui est réputé calculer la fonction* $f91$

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.*

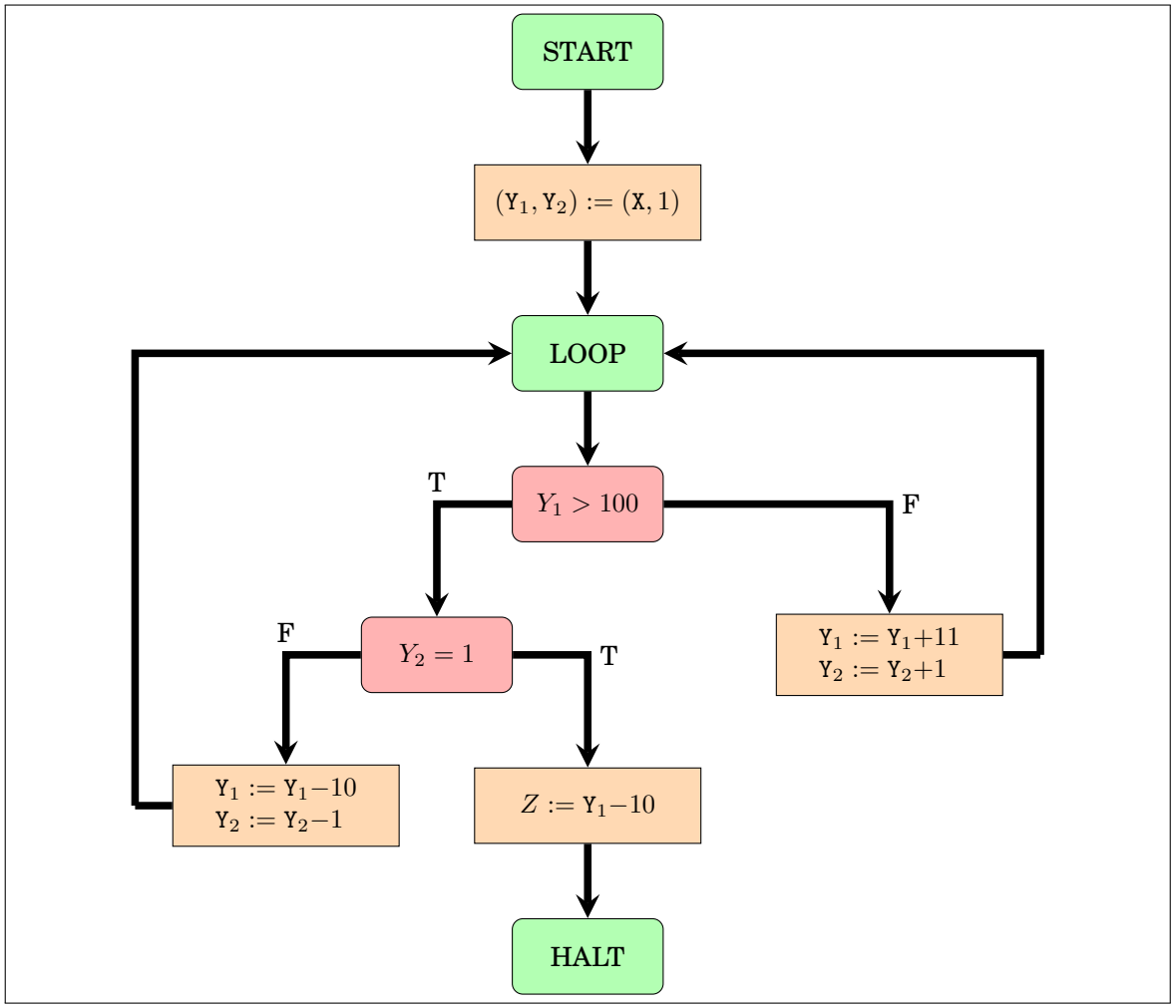


FIGURE 2 – Flowchart du calcul de la fonction de McCarthy

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

◇– Solution de l'exercice 4

Modèle solution TLA$^+$:
../tlamodels/malgtd1ex4.tla

```
------------------------------- MODULE malgtd1ex4 -------------------------
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+11 /\ 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>>
-------------------------------------------------------------------------
(* spcification *)
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"
-------------------------------------------------------------------------
```

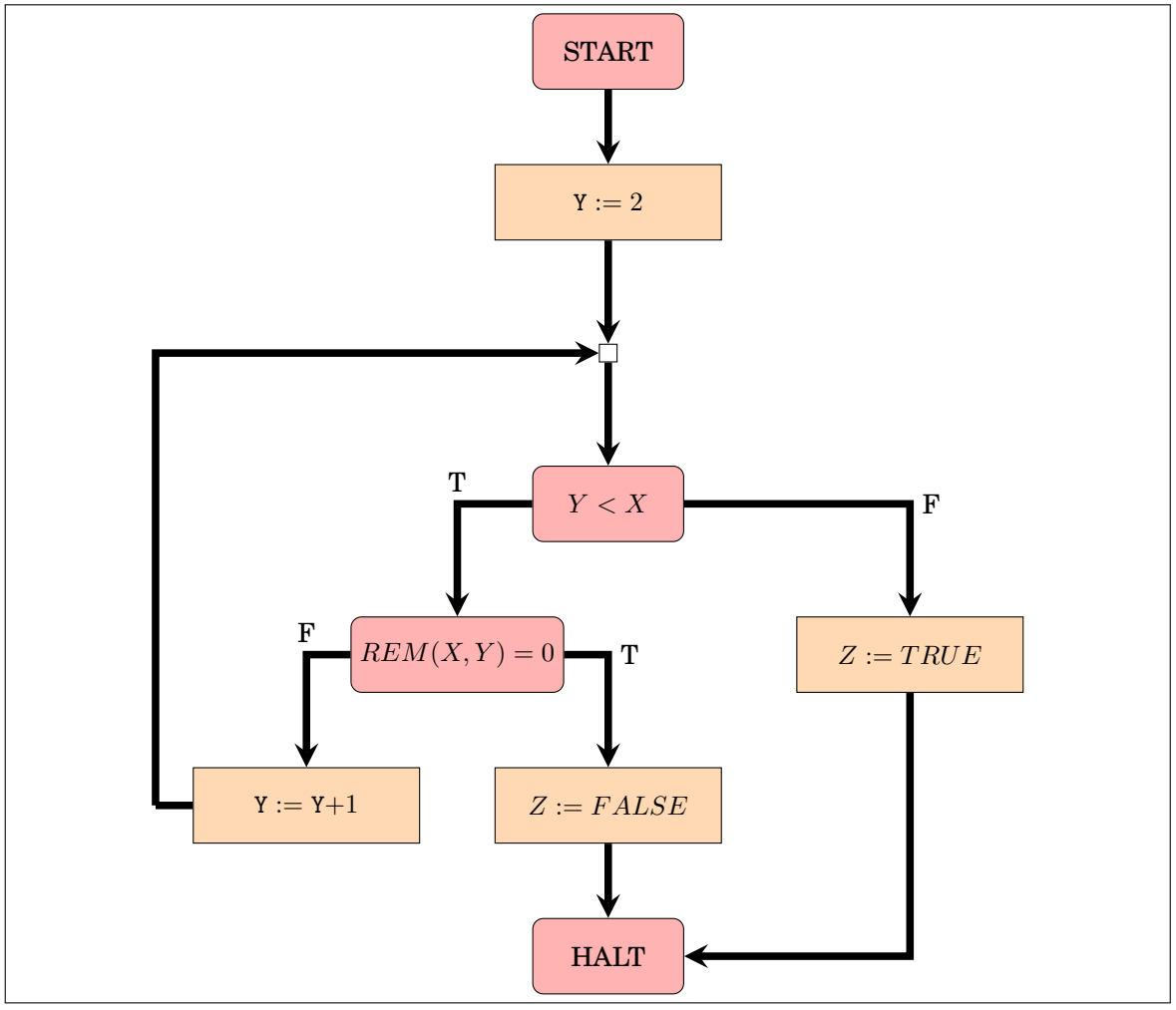


FIGURE 3 – Flowchart pour le test de primalité

```
(* 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$^+$ 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 $zf = prime(x0) \wedge xf = x0$

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$^+$:
../tlamodels/malgtd1ex5.tla

```
------------MODULE malgtd1ex5 --------

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$^+$:
../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*

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

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

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

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

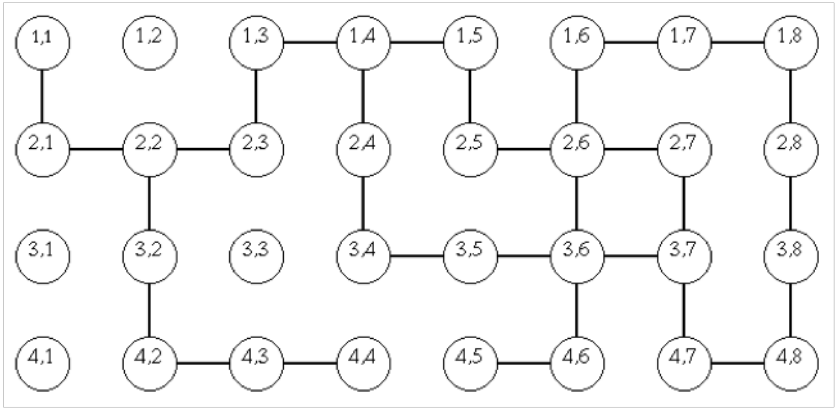
$RandomElement(s) \triangleq$ CHOOSE $x \in s : TRUE$

$Any \triangleq$ 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



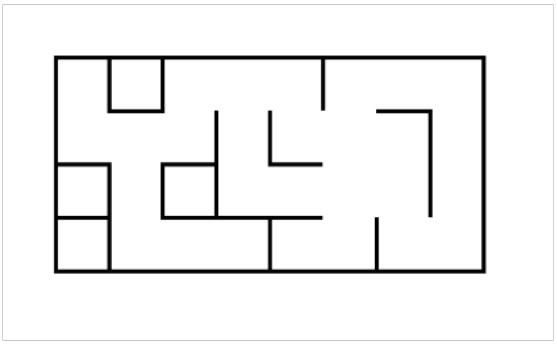


FIGURE 5 – Labyrinthe

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* $<< i, j >>$ *de la matrice les voisins communiquant à l'aide de la fonction lab qui associe à* $<< i, j >>$ *les éléments qui peuvent être atteints en un coup. Par exemple, le mouvement possible à partir de* $<< 1, 1 >>$ *est* $<< 2, 1 >>$*, ou le mouvement possible à partir de* $<< 2, 1 >>$ *est* $<< 2, 2 >>$ *ou* $<< 1, 1 >>$*, ou le mouvement possible à partir de* $<< 2, 2 >>$ *est* $<< 2, 3 >>$ *ou* $<< 3, 2 >>$ *ou* $<< 2, 1 >>$*, ...*

```
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*

$$
\begin{array}{l}
n \triangleq 10 \\
\mathit{nodes} \triangleq 1..n \\
l \triangleq [i \in 1..n \mapsto \text{IF } i = 1 \text{ THEN } \{4,5\} \text{ ELSE} \\
\qquad\qquad\qquad\quad \text{IF } i = 2 \text{ THEN } \{6,7,10\} \text{ ELSE} \\
\qquad\qquad\qquad\quad \text{IF } i = 4 \text{ THEN } \{7,8\} \text{ ELSE} \\
\qquad\qquad\qquad\quad \text{IF } i = 5 \text{ THEN } \{\} \text{ ELSE} \\
\qquad\qquad\qquad\quad \text{IF } i = 6 \text{ THEN } \{4\} \text{ ELSE} \\
\qquad\qquad\qquad\quad \text{IF } i \ = 7 \text{ THEN } \{5\} \text{ ELSE} \\
\qquad\qquad\qquad\quad \text{IF } i \ = 8 \text{ THEN } \{5,\ 2\} \text{ ELSE} \\
\qquad\qquad\qquad\quad \{\} \\
\qquad\qquad\qquad\quad] \\
\\
\mathit{Init} \triangleq p = 1 \\
M(i) \triangleq \land i \in l[p] \\
\qquad\quad \land p' = i \\
\mathit{Next} \triangleq \exists\, i \in 1..n : \ M(i)
\end{array}
$$

◇– Solution de l'exercice 7

Modèle solution TLA$^+$:
../tlamodels/malgtd1ex7.tla

```
------------------------------- MODULE malgtd1ex7 -------------------------------
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 {}
    ]
```

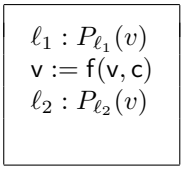
```
Init == p = 1
M(i) == /\ i \in l[p]
        /\ p'=i
Next == \E i \in 1..n: M(i)

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
```

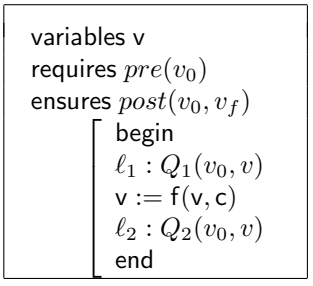
==========

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*





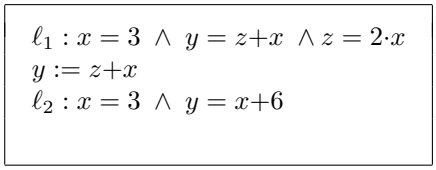
- $pre(v_0) \equiv P_{\ell_1}(v_0)$
- $post(v_0, v_f) \equiv P_{\ell_2}(v_f)$.
- $Q_{\ell_1}(v_0, v) \equiv P_{\ell_1}(v) \wedge v = v_0$
- $Q_{\ell_2}(v_0, v) \equiv P_{\ell_2}(v)$

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)*



◇– Solution de la question 8.1
Modèle solution TLA$^+$:
../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 *)
al1l2 ==
    /\ pc="l1"
    /\ pc'="l2"
    /\ y'=z+x
    /\ z'=z /\ x'=x
-----------------------------------------------------------------
(* Computations *)
vars ==  <<x,y,z,pc>>
Next == al1l2  \/ 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

Question 8.2 *(malgtd1ex10bis)*
Pour les deux exemples qui suivent, on considère dex cas et on doit donner une interprétation.

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

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

◇– **Solution de la question 8.2**
Modèle solution TLA$^+$:
../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 *)
al1l2 ==
    /\ pc="l1"
    /\ pc'="l2"
    /\ x'=y+x+2^x
    /\ y'=y
-----------------------------------------------------------
(* Computations *)
Next == al1l2  \/ 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)*

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

◇– 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 *)
al1l2 ==
    /\ pc="l1"
    /\ pc'="l2"
    /\ x'=2*y+x
    /\ y'=y
-----------------------------------------------------------------
(* Computations *)
Next == al1l2  \/ 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: w assume that the precondition can hold and we have to find poss
ASSUME  pre

------------------------------------------------------------------------
(* Action for transitioon of the algorithm *)
al1l2 ==
    /\ pc="l1"
    /\ pc'="l2"
    /\ z'=x
    /\ x' = z
    /\ y' = z'

------------------------------------------------------------------
(* Computations *)
Next == al1l2  \/ 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

Exercice 9 *(malgtd1ex11)*

Question 9.1 *Ecrire un module TLA$^+$ qui traduit la relation de transition de cet algorithme selon les instructions.*

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 LaTeXsous le numéro 9 .*

Fin 9.2

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

◇– **Solution de la question 9.3**
Modèle solution TLA$^+$:
../tlamodels/malgtd1ex11.tla

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

$\ell_0 : \{\ldots\}$
if $X < Y$ **then**
  $\ell_1 : \{\ldots\}$
  $Z := Y$;
  $\ell_2 : \{\ldots\}$
else
  $\ell_3 : \{\ldots\}$
  $Z := X$;
  $\ell_4 : \{\ldots\}$
;
$\ell_5 : \{\ldots\}$

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

```
------------------------------- MODULE malgtd1ex11 ------------------------------
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
-------------------------------------------------------------------------
al0l1 ==
    /\ pc="l0"
    /\ pc'="l1"
    /\ x<y
    /\ z'=z /\ x'=x /\ y'=y
al1l2 ==
    /\ pc="l1"
    /\ pc'="l2"
    /\ z'=y
    /\ x'=x /\ y'=y
al2l5 ==
    /\ pc="l2"
    /\ pc'="l5"
    /\ z'=z /\ x'=x /\ y'=y
al0l3 ==
    /\ pc="l0"
    /\ pc'="l3"
    /\ x \geq y
    /\ UNCHANGED <<z,x,y>>
al3l4 ==
    /\ pc="l3"
    /\ pc'="l4"
    /\ z'=x
    /\ x'=x /\ y'=y
  al4l5 ==
    /\ pc="l4"
    /\ pc'="l5"
    /\ z'=z /\ x'=x /\ y'=y
---------------------
Next == al0l1 \/ al1l2 \/ al2l5  \/ al0l3 \/ al3l4 \/ al4l5 \/ 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
safepc ==  pc="l5" =>  z = maxi(x0,y0)
safeab == x=x0 /\ y=y0
saferte ==
    /\ mini0 <= x /\ x <= maxi0
    /\ mini0 <= y /\ y <= maxi0
=========================================================================
\* Modification History
```

Dominique Méry le 12 février 2025

```
\* 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*
$\forall x, y, , x', y'. P_\ell(x, y) \ \wedge \ cond_{\ell,\ell'}(x, y) \ \wedge \ (x', y') = f_{\ell,\ell'}(x, y) \Rightarrow P_{\ell'}(x', y')$

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

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

— *On suppose que p est un nombre premier :*
$\ell_1 : x = 2^p \ \wedge \ y = 2^{p+1} \ \wedge \ x{\cdot}y = 2^{2{\cdot}p+1}$
$x := y{+}x{+}2^x$
$\ell_2 : x = 5{\cdot}2^p \ \wedge \ y = 2^{p+1}$

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

Exercice 11 *Montrer que chaque annotation est correcte ou incorrecte selon les conditions de vérifications énoncées comme suit*
$\forall x, y, , x', y'. P_\ell(x, y) \ \wedge \ cond_{\ell,\ell'}(x, y) \ \wedge \ (x', y') = f_{\ell,\ell'}(x, y) \Rightarrow P_{\ell'}(x', y')$

— *(1)* $\ell_1 : x = 9 \ \wedge \ y = z{+}x$
$y := x{+}9$
$\ell_2 : x = 9 \ \wedge \ y = x{+}9$

— $\ell_1 : x = 3 \ \wedge \ y = 3$
$x := y{+}x$
$\ell_2 : x = 6 \ \wedge \ y = 3$

— *(2)* $\ell_1 : x = 1 \ \wedge \ y = 3 \ \wedge \ x{+}y = 12$
$x := y{+}x$
$\ell_2 : x = 567 \ \wedge \ y = 34$

— $\ell_1 : x = 1 \ \wedge \ y = 3$
$z := x; x := y; y := z;$
$\ell_2 : x = 3 \ \wedge \ y = 1$

1. $c = \ell_1 \wedge x = 9 \ \wedge \ y = z{+}x \ \wedge$ **TRUE** $\wedge \ (x', y', c') = (x, x{+}9, \ell_2) \Rightarrow c' = \ell_2 \wedge x' = 9 \ \wedge \ y' = x'{+}9$ *:*
   - *(a)* $c = \ell_1 \wedge x = 9 \ \wedge \ y = z{+}x \ \wedge c' = \ell_2 \Rightarrow c' = \ell_2 \wedge x = 9 \ \wedge \ x{+}9 = x{+}9$
   - *(b)* $c = \ell_1 \wedge x = 9 \ \wedge \ y = z{+}x \ \wedge c' = \ell_2 \Rightarrow x = 9 \ \wedge \ x{+}9 = x{+}9$
   - *(c) CORRECT*
2. $c = \ell_1 \wedge x = 1 \ \wedge \ y = 3 \ \wedge \ x{+}y = 12 \ \wedge$ **TRUE** $\wedge \ (x', y', c') = (y{+}x, y, \ell_2) \Rightarrow c' = \ell_2 \wedge x' = 567 \ \wedge \ y' = 34$ *:*
   - *(a)* $c = \ell_1 \wedge x = 1 \ \wedge \ y = 3 \ \wedge \ x{+}y = 12 \ \wedge \ (x', y', c') = (y{+}x, y, \ell_2) \Rightarrow c' = \ell_2 \wedge y{+}x = 567 \ \wedge \ y = 34$
   - *(b)* $c = \ell_1 \wedge x = 1 \ \wedge \ y = 3 \ \wedge \ x{+}y = 12 \ \wedge \ c' = \ell_2 \Rightarrow c' = \ell_2 \wedge y{+}x = 567 \ \wedge \ y = 34$
   - *(c)* $c = \ell_1 \wedge x = 1 \ \wedge \ y = 3 \ \wedge \ x{+}y = 12 \ \wedge \ c' = \ell_2 \Rightarrow x{+}y = 4 \wedge x{+}y = 12$
   - *(d)* $c = \ell_1 \wedge x = 1 \ \wedge \ y = 3 \ \wedge \ x{+}y = 12 \ \wedge \ c' = \ell_2 \Rightarrow$ **FALSE**
   - *(e)* **FALSE** $\Rightarrow c' = \ell_2 \wedge y{+}x = 567 \ \wedge \ y = 34$
   - *(f) CORRECT*
3. $c = \ell_1 \wedge x = 3 \ \wedge \ y = 3 \wedge$ **TRUE** $\wedge \ (x', y', c') = (y{+}x, y, \ell_2) \Rightarrow c' = \ell_2 \wedge x' = 6 \ \wedge \ y' = 3$

(a) $c = \ell_1 \wedge x = 3 \ \wedge \ y = 3 \ \wedge \ c' = \ell_2 \Rightarrow c' = \ell_2 \wedge y{+}x = 6 \ \wedge \ y = 3$
(b) $c = \ell_1 \wedge x = 3 \ \wedge \ y = 3 \ \wedge \ c' = \ell_2 \Rightarrow c' = \ell_2 \wedge y{+}x = 6 \ \wedge \ y = 3$
(c) $c = \ell_1 \wedge x = 3 \ \wedge \ y = 3 \ \wedge \ c' = \ell_2 \Rightarrow c' = \ell_2 \wedge 6 = 6 \ \wedge \ y = 3$
(d) CORRECT

4. $c = \ell_1 \wedge x = 1 \ \wedge \ y = 3$ **TRUE** $\wedge \ (x', y', z', c') = (y, x, x, \ell_2) \Rightarrow c' = \ell_2 \wedge x' = 3 \ \wedge \ y' = 1$

(a) $c = \ell_1 \wedge x = 1 \ \wedge \ y = 3$ **TRUE** $\wedge \ (x', y', z', c') = (y, x, x, \ell_2) \Rightarrow c' = \ell_2 \wedge y = 3 \ \wedge \ x = 1$
(b) CORRECT

TD4

Exercice 12 *(malgtd1ex12) Dans l'algorithm 12, on calcule le maximum d'une suite de valeurs entières. On vous demande :*
— *Définir la précondition et la postcondition.*
— *Annoter cet algorithme*
— *Vérifier les conditions de vérification pour la correction partielle*
— *Vérifier les conditions pour l'absence d'erreurs à l'exécution*
— *Ecrire un module TLA pour valider ce qui a été prouvé.*

Variables : F,N,M,I

Requires : $\left(\begin{array}{l} n_0 \in \mathbb{N} \wedge \\ n_0 \neq 0 \wedge \\ f_0 \in 0 \mathinner{..} n_0{-}1 \to \mathbb{N} \end{array} \right)$

Ensures : $\left(\begin{array}{l} m_f \in \mathbb{N} \wedge \\ m_f \in ran(f_0) \wedge \\ (\forall j \cdot j \in 0 \mathinner{..} n_0{-}1 \Rightarrow f_0(j) \leq m_f) \end{array} \right)$

$M := F(0)$;
$I := 1$;
while $I < N$ **do**
  if ***F(i)>M*** **then**
    $M := F(I)$;
  ;
  $I{+}{+}$;
;

Algorithme 3: Algorithme du maximum d'une liste non annotée

◇– **Solution de l'exercice 12**
La solution de cette annotation est dans l'algorithme annoté.
Modèle solution TLA$^+$:
../tlamodels/malgtd1ex12.tla

```
------------------------------ MODULE malgtd1ex12 ------------------------------
(* computing the maximum value of an array f *)
EXTENDS Naturals, TLC,Integers
CONSTANTS undef,n0,f0,i0,m0,min,max
--------------------------------
VARIABLES n,f,m,i,pc
--------------------------------
(* Auxiliary defintions *)
(* an exampe for an array *)
def0 == [j \in 0..n0-1 |-> n0-j]
```

/* algorithme de calcul du maximum avec une boucle while de l'exercice 12 */
Variables : F,N,M,I

Requires : $\left(\begin{array}{l} n_0 \in \mathbb{N} \wedge \\ n_0 \neq 0 \wedge \\ f_0 \in 0\,..\,n_0{-}1 \to \mathbb{N} \end{array}\right)$

Ensures : $\left(\begin{array}{l} m_f \in \mathbb{N} \wedge \\ m_f \in ran(f_0) \wedge \\ (\forall j \cdot j \in 0\,..\,n_0{-}1 \Rightarrow f_0(j) \leq m_f) \end{array}\right)$

$\ell_0 : \{\left(\begin{array}{l} n_0 \in \mathbb{N} \wedge \\ n_0 \neq 0 \wedge \\ f_0 \in 0\,..\,n_0{-}1 \to \mathbb{N} \\ m_0, i_0 \in \mathbb{Z} \end{array}\right) \wedge n = n_\wedge f = f0 \wedge i = i_0 \wedge m = m_0\}$

$M := F(0);$

$\ell_1 : \{\left(\begin{array}{l} n_0 \in \mathbb{N} \wedge \\ n_0 \neq 0 \wedge \\ f_0 \in 0\,..\,n_0{-}1 \to \mathbb{N} \\ m_0, i_0 \in \mathbb{Z} \end{array}\right) \wedge n = n_\wedge f = f0 \wedge i = i_0 \wedge m = f(0)\}$

$I := 1;$

$\ell_2 : \{\left(\begin{array}{l} n_0 \in \mathbb{N} \wedge \\ n_0 \neq 0 \wedge \\ f_0 \in 0\,..\,n_0{-}1 \to \mathbb{N} \\ m_0, i_0 \in \mathbb{Z} \\ n = n_0 \wedge f = f0 \end{array}\right) \wedge i = 1 \wedge \left(\begin{array}{l} m \in \mathbb{N} \wedge \\ m \in ran(f[0..i{-}1]) \wedge \\ (\forall j \cdot j \in 0\,..\,i{-}1 \Rightarrow f(j) \leq m) \end{array}\right)\}$

while $I < N$ **do**

$\ell_3 : \{\left(\begin{array}{l} n_0 \in \mathbb{N} \wedge \\ n_0 \neq 0 \wedge \\ f_0 \in 0\,..\,n_0{-}1 \to \mathbb{N} \\ m_0, i_0 \in \mathbb{Z} \\ n = n_0 \wedge f = f0 \end{array}\right) \wedge i \in 1..n{-}1 \wedge \left(\begin{array}{l} m \in \mathbb{N} \wedge \\ m \in ran(f[0..i{-}1]) \wedge \\ (\forall j \cdot j \in 0\,..\,i{-}1 \Rightarrow f(j) \leq m) \end{array}\right)\}$

if *F(I)>M* **then**

$\ell_4 : \{\left(\begin{array}{l} n_0 \in \mathbb{N} \wedge \\ n_0 \neq 0 \wedge \\ f_0 \in 0\,..\,n_0{-}1 \to \mathbb{N} \\ m_0, i_0 \in \mathbb{Z} \\ n = n_0 \wedge f = f0 \end{array}\right) \wedge i \in 1..n{-}1 \wedge \left(\begin{array}{l} m \in \mathbb{N} \wedge \\ m \in ran(f[0..i{-}1]) \wedge \\ (\forall j \cdot j \in 0\,..\,i{-}1 \Rightarrow f(j) \leq m) \end{array}\right) \wedge$
$f(i) > m\}$

$M := F(I);$

$\ell_5 : \{\left(\begin{array}{l} n_0 \in \mathbb{N} \wedge \\ n_0 \neq 0 \wedge \\ f_0 \in 0\,..\,n_0{-}1 \to \mathbb{N} \\ m_0, i_0 \in \mathbb{Z} \\ n = n_0 \wedge f = f0 \end{array}\right) \wedge i \in 1..n{-}1 \wedge \left(\begin{array}{l} m \in \mathbb{N} \wedge \\ m \in ran(f[0..i]) \wedge \\ (\forall j \cdot j \in 0\,..\,i \Rightarrow f(j) \leq m) \end{array}\right)\}$

;

$\ell_6 : \{\left(\begin{array}{l} n_0 \in \mathbb{N} \wedge \\ n_0 \neq 0 \wedge \\ f_0 \in 0\,..\,n_0{-}1 \to \mathbb{N} \\ m_0, i_0 \in \mathbb{Z} \\ n = n_0 \wedge f = f0 \end{array}\right) \wedge i \in 1..n{-}1 \wedge \left(\begin{array}{l} m \in \mathbb{N} \wedge \\ m \in ran(f[0..i]) \wedge \\ (\forall j \cdot j \in 0\,..\,i \Rightarrow f(j) \leq m) \end{array}\right)\}$

$I{+}{+};$

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

;

$\ell_8 : \{\left(\begin{array}{l} n \in \mathbb{N} \wedge \\ n \neq 0 \wedge \\ f \in 0\,..\,n{-}1 \to \mathbb{N} \end{array}\right) \wedge i \in \mathbb{Z} \wedge \wedge i = n \wedge \left(\begin{array}{l} m \in \mathbb{N} \wedge \\ m \in ran(f[0..n{-}1]) \wedge \\ (\forall j \cdot j \in 0\,..\,n{-}1 \Rightarrow f(j) \leq m) \end{array}\right)\}$

Algorithme 4: Algorithme du maximum d'une liste annoté

```
(* defintion of the range of a function *)
ran(g) == { u  \in Nat : ( \E j \in DOMAIN g : g[j]=u) }
(* defintion of the restriction of a function *)
Rest(g,l) == [k \in 0..l |-> g[k]]
(* precondition *)
pre ==
    /\ n0 \in Nat /\ n0 # 0
    /\ f0 = def0
    /\ i0 \in Int /\ m0 \in Int
pre1 == f=f0 /\ n=n0 /\ pre
(* Integers for your computer *)
zinf == min..max
(* Naturals  for your computer *)
ninf == 0..max
------------------------------
(* assuming precondition  over initial values of variables *)
ASSUME pre
------------------------------
(* Initialisaton for tyhe TLA model *)
Init == /\ i = i0
        /\ m = m0
        /\ f=f0
        /\ n=n0
        /\ pc = "l0"
--------------------------------
(* actions for transition *)
l0l1 == /\ pc = "l0"
        /\ m' = f[0]
        /\ pc'= "l1"
        /\ UNCHANGED <<n,f,i>>

l1l2 == /\ pc = "l1"
        /\ i' = 1
        /\ pc'= "l2"
        /\ UNCHANGED <<n,f,m>>

l2l3 == /\ pc = "l2"
        /\ i < n
        /\ pc'= "l3"
        /\ UNCHANGED <<n,f,m,i>>


l2l8 == /\ pc = "l2"
        /\ (i \geq n)
        /\ m' = m
        /\ i' = i
        /\ pc'= "l8"
        /\ UNCHANGED <<n,f>>

l3l4 == /\ pc = "l3"
        /\ f[i] > m
        /\ m' = m
        /\ i' = i
        /\ pc'= "l4"
        /\ UNCHANGED <<n,f>>
```

```
l3l6 == /\ pc = "l3"
        /\ (f[i] \leq m)
        /\ m' = m
        /\ i' = i
        /\ pc'= "l6"
        /\ UNCHANGED <<n,f>>

l4l5 == /\ pc = "l4"
        /\ m' = f[i]
        /\ i' = i
        /\ pc'= "l5"
        /\ UNCHANGED <<n,f>>

l5l6 == /\ pc = "l5"
        /\ m' = m
        /\ i' = i
        /\ pc'= "l6"
        /\ UNCHANGED <<n,f>>

l6l7 == /\ pc = "l6"
        /\ m' = m
        /\ i' = i + 1
        /\ pc'= "l7"
       /\ UNCHANGED <<n,f>>

l7l3 == /\ pc = "l7"
        /\ i < n
        /\ m' = m
        /\ i' = i
        /\ pc'= "l3"
        /\ UNCHANGED <<n,f>>

 l7l8 ==
        /\ pc = "l7"
        /\ i \geq n
        /\ m' = m
        /\ i' = i
        /\ pc'= "l8"
        /\ UNCHANGED <<n,f>>
-------------------------------
(* Next relatioon over  values variables *)
 Next == \/ l0l1
        \/ l1l2
        \/ l2l3
        \/ l2l8
        \/ l3l4
        \/ l3l6
        \/ l4l5
        \/ l5l6
        \/ l6l7
        \/ l7l3
        \/ l7l8
        \/ UNCHANGED <<n,m,i,f,pc>>

  Dl0l1 ==  0\leq 0 /\ 0 \leq n0-1
  Dl1l2 == 1 \in zinf
```

```
inv ==
   /\ pc \in {"l0","l1","l2","l3","l4","l5","l6","l7","l8"}
   /\ n \in Int /\ f = def0 /\ i \in Int /\ m \in Int
   /\ pc="l0" =>   f=f0 /\ n=n0 /\ m=m0 /\ i = i0/\ pre
   /\ pc="l1" =>  f=f0 /\ n=n0 /\ m=f[0] /\ i = i0 /\ pre
   /\ pc ="l2" =>  i=1 /\  m \in Nat /\ (m \in ran(Rest(f,i-1))) /\ (\A k  \in 0..i
   /\ pc="l3" =>     (i \in 1..n-1) /\  m \in Nat /\ (m \in ran(Rest(f,i-1))) /\ (\
   /\ pc="l4" =>    f[i] > m /\ (i \in 1..n-1) /\  m \in Nat /\ (m \in ran(Rest(f,i
   /\ pc="l5"  =>   f[i] > m /\ (i \in 1..n-1) /\  m \in Nat /\ (m \in ran(Rest(f,i
   /\ pc="l6" =>    (i \in 1..n-1) /\  m \in Nat /\ (m \in ran(Rest(f,i))) /\ (\A k
   /\ pc="l7" =>    (i \in 1..n) /\  m \in Nat /\ (m \in ran(Rest(f,i-1))) /\ (\A k
   /\ pc="l8" =>   i=n /\  m \in Nat /\ (m \in ran(Rest(f,i-1))) /\ (\A k  \in 0..i

partialcorrectness == pc="l8" => m \in Nat /\ (m \in ran(Rest(f,n-1))) /\ (\A k  \i

runtimeerrors ==  m \in zinf /\ i \in zinf /\ n \in zinf

safe ==  inv  /\ runtimeerrors  /\ partialcorrectness
==============================================================================
```

Fin 12

Exercice 13 *(malgtd1ex13)*
Soit l'algorithme 13 de la boucle bornée. On demande

1. *de définir le contrat de cet algorithme*
2. *d'annoter l'algorithme.*
3. *de vérifier les conditions de vérification.*
4. *de proposer un modèle TLA$^+$ pour vérifier les annotations et la correction partielle*

Variables : X
Requires : $x_0 \in \mathbb{N}$
Ensures : $x_f = 0$

$\ell_0 : \{\ldots\}$
while $0 < X$ **do**
  $\ell_1 : \{\ldots\}$
  $X := X-1$;
  $\ell_2 : \{\ldots\}$
;
$\ell_3 : \{\ldots\}$

Algorithme 5: Exemple de la boucle bornée

◇– **Solution de l'exercice 13**

Annotation L'annotation (cf algorithme) est construite par propagation des assertions selon les instructions. Il faut ensuite vérifier que les conditions sont vraies.

Modèle TLA$^+$ pour vérifier la bonne annotation Modèle solution TLA$^+$:
../tlamodels/malgtd1ex13.tla

Variables : X
Requires : $x_0 \in \mathbb{N}$
Ensures : $x_f = 0$

$\ell_0 : \{ x = x_0 \wedge x_0 \in \mathbb{N}\}$
while $0 < X$ **do**
  $\ell_1 : \{0 < x \leq x_0 \wedge x_0 \in \mathbb{N}\}$
  $X := X-1$;
  $\ell_2 : \{0 \leq x \leq x_0 \wedge x_0 \in \mathbb{N}\}$
;
$\ell_3 : \{x = 0\}$

Algorithme 6: exemple annoté

```
------------------------------- MODULE malgtd1ex13 ------------------------------
EXTENDS Naturals, Integers
CONSTANTS x0
VARIABLES  x,pc
ASSUME x0 \in Nat
typeInt(u) == u \in Int

---------------------------------------------------------------------------
al0l1 ==
    /\ pc="l0"
    /\ pc'="l1"
    /\ 0<x
    /\ x' = x
al1l2 ==
    /\ pc="l1"
    /\ pc'="l2"
    /\ x'=x-1

al2l3 ==
    /\ pc="l2"
    /\ pc'="l3"
    /\ 0 \geq x
    /\ x'=x

al2l1 ==
    /\ pc="l2"
    /\ pc'="l1"
    /\ 0  <  x
    /\ x'=x
al0l3 ==
    /\ pc="l0"
    /\ pc'="l3"
    /\ 0 \geq x
    /\  x'=x



---------------------
Next == al0l1 \/ al1l2 \/ al2l3  \/ al0l3 \/ al2l1 \/ UNCHANGED <<x,pc>>
```

```
Init == pc="l0" /\ x=x0
----------------------
inv ==
    /\ typeInt(x)
    /\ pc \in {"l0","l1","l2","l3"}
    /\ pc="l0" =>  x=x0 /\ x0 \in Nat
    /\ pc="l1" => 0 < x /\ x \leq x0 /\ x0 \in Nat
    /\ pc="l2" => 0 \leq x  /\  x  < x0 /\ x0 \in Nat
    /\ pc="l3" => x = 0

safe ==  pc="l3" => x=0

============================================================================
\* Modification History
\* Last modified Thu Sep 23 11:52:02 CEST 2021 by mery
\* Created Wed Sep 09 18:19:08 CEST 2015 by mery
```

Fin 13

On rappelle qu'un contrat pour la correction partielle d'un petit programme est donné par les éléments ci-dessou 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 $type\ X$

definitions
$def1 \stackrel{def}{=} text1$

requires $pre(x_0)$

ensures $post(x_0, x_f)$

begin
$0 : P_0(x_0, x)$
$\texttt{instruction}_0$
$1 : P_i(x_0, x)$
$\texttt{instruction}_1$
$f : P_f(x_0, x)$
end

Conditions de vérification
- $pre(x_0) \wedge x = x_0 \Rightarrow P_0(x_0, x)$
- $pre(x_0) \wedge P_f(x_0, x) \Rightarrow post(x_0, x)$
- Pour toutes les paires ℓ, ℓ', telles que $\ell \longrightarrow \ell'$, on vÃ©rifie que, pour toutes les valeurs $x, x' \in$ MEMORY
$$\left(\begin{array}{l} \left(\begin{array}{l} pre(x_0) \wedge P_\ell(x_0, x)) \\ \wedge cond_{\ell,\ell'}(x) \wedge x' = f_{\ell,\ell'}(x) \end{array} \right) \\ \Rightarrow P_{\ell'}(x_0, x') \end{array} \right),$$

Exercice 14 *(6 points)*
Soit le contrat suivant qui met en jeu les variables X,Y, Z,C,R.

VARIABLES $\texttt{int}\ X, Y, Z, C, R$

REQUIRES $x_0, y_0, z_0, c_0, r_0 \in \mathbb{Z}$

ENSURES $r_f = 0$

BEGIN
$0 : x = x_0 \wedge y = y_0 \wedge z = z_0 \wedge c = c_0 \wedge r = r_0 \wedge x_0, y_0, z_0, c_0, r_0 \in \mathbb{Z}$
$(X, Z, Y) := (49, 2{\cdot}C, (2{\cdot}C{+}1){\cdot}(2{\cdot}C{+}1));$
$1 : x = 49 \wedge z = 2{\cdot}c \wedge y = (z{+}1){\cdot}(z{+}1)$
$Y := X{+}Z{+}1;$
$2 : x = 49 \wedge z = 2{\cdot}c \wedge y = (c{+}1){\cdot}(c{+}1)$
END

Question 14.1 *Ecrire les conditions de vérification associée au contrat ci-dessus en vous aidant du rappel de la définition de ces conditions de vérification.*

Question 14.2 *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 prÃ©condition est satisfaisable.*

Exercice 15 *(6 points)*

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 15.1 *Donner l'assertion Requires en complétant ce qui est déjà mentionné et en reportant le texte complet de cette assertion Requires dans votre copie.*

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

Question 15.2 *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 $\begin{cases} n_0 < b_0 \Rightarrow x_f = \textbf{\textit{question1}} \\ n_0 \geq b_0 \Rightarrow x_f = \textbf{\textit{question1}} \\ n_f = n_0 \wedge b_f = b_0 \end{cases}$

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 16 *(8 points)*

VARIABLES N, V, S, I

DEFINITIONS

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

REQUIRES $\begin{cases} n_0 \in \mathbb{N} \wedge n_0 \neq 0 \\ v_0 \in 0..n{-}1 \longrightarrow \mathbb{Z} \end{cases}$

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

BEGIN
$\ell_0 : \begin{cases} pre(n_0, v_0, s_0, i_0) \\ (n, v, s, i) = (n_0, v_0, s_0, i_0) \end{cases}$
$S := V(0)$
ℓ_1 :?????????
$I := 1$
$\ell_2 : \begin{cases} pre(n_0, v_0, s_0, i_0) \\ s = \bigcup_{k=0}^{i-1} v(k) \wedge i = 1 \\ (n, v) = (n_0, v_0) \end{cases}$
WHILE $I < N$ ***DO***
$\ell_3 : \begin{cases} pre(n_0, v_0, s_0, i_0) \\ s = \bigcup_{k=0}^{i-1} v(k) \wedge i \in 1..n{-}1 \\ (n, v) = (n_0, v_0) \end{cases}$
$S := S \oplus V(I)$
ℓ_4 :?????
$I := I{+}1$
ℓ_5 :?????
OD;
ℓ_6 :?????
END

La notation $\bigcup_{k=i}^{j} v(k)$ *désigne la valeur maximale des éléments* $\{v(k) | k \in i..j\}$ *et on suppose que l'opérateur* $\oplus$ *renvoie pur deux valeurs entières, la valeut maximale.*

Question 16.1 *Compléter les annotations incomplètes* ℓ_1, ℓ_4 , ℓ_5 *et* ℓ_6.

Question 16.2 *Vérifier les conditions de vérification associées aux transitions suivantes :*

1. ℓ_0, ℓ_1
2. ℓ_2, ℓ_3
3. ℓ_3, ℓ_4
4. ℓ_5, ℓ_6

Question 16.3 *Donner et vérifier les points pour assurer la correction partielle de cet algorithme.*

Question 16.4 *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.*

Fin de la série 1