MVSI – Modelling and Verification of Computer Systems

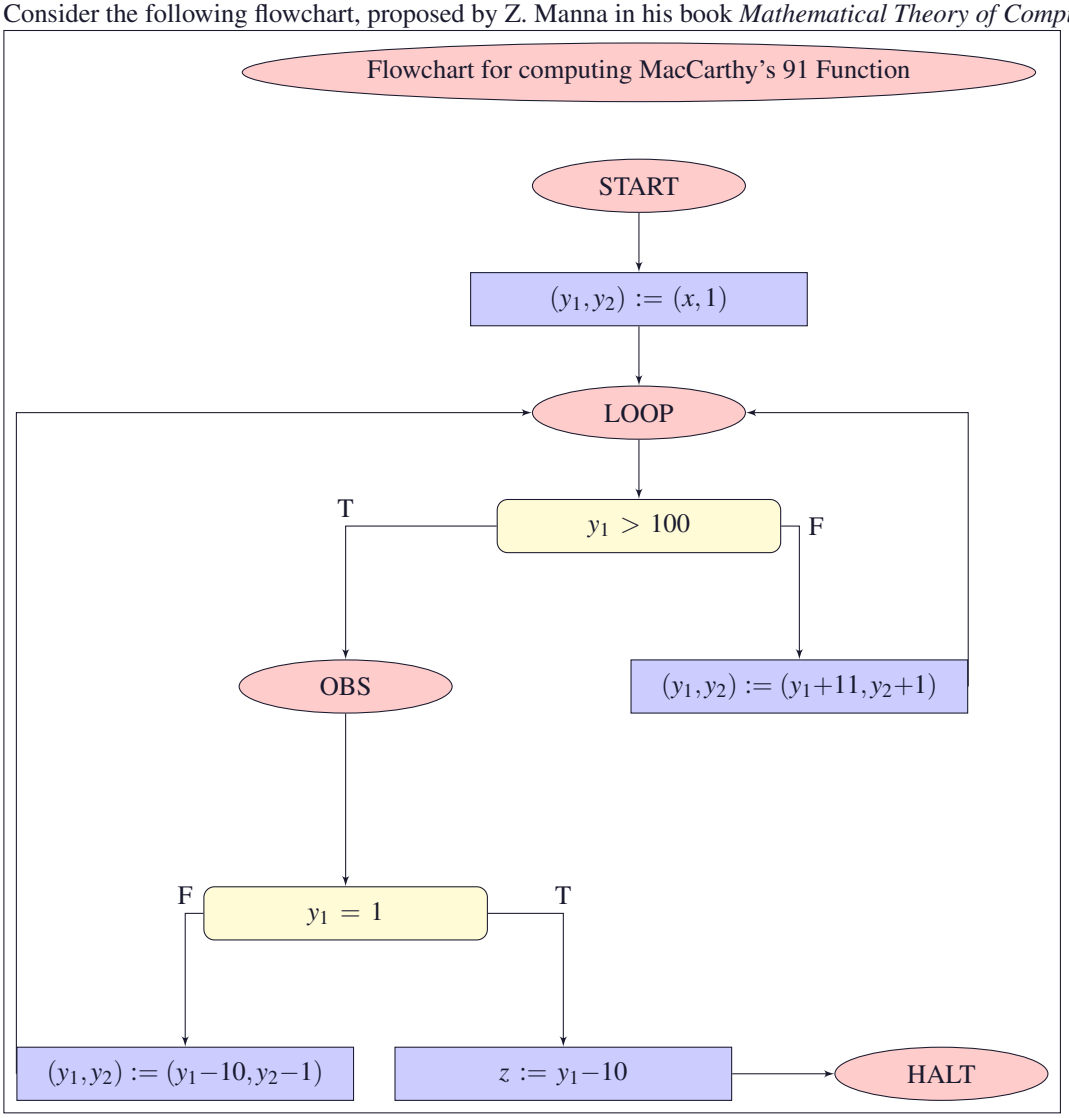
TLA$^+$ modelling (2)

Tutorial – with solutions

Dominique Méry – 20th November 2026

✎ Exercise 1 `appex2_2`

Consider the following flowchart, proposed by Z. Manna in his book *Mathematical Theory of Computation*.



Question 1.1 Build a TLA$^+$ module modelling the different computation steps.

Question 1.2 Evaluate the algorithm by asking the following safety questions:

1. the algorithm is partially correct.
2. the algorithm has no run-time errors.

✔ Solution of exercise 1

Listing 1: TLA$^+$ module `appex2_2.tla`

```
1 ------------------------------- MODULE appex2_2 -------------------------------
2 (* Calcul de la fonction de MacCarthy *)
3 EXTENDS Naturals,TLC,Integers
```

```
------------------
CONSTANTS x,max,u
min == -max
--------------------------------
VARIABLES  y1,y2,z,pc
---------------------------------
BF(Y) == Y#u => Y \in min..max

ASSUME  BF(x)

-----------------------------

start == pc="START" /\ y1'=x /\ y2'=1/\ pc'="LOOP"  /\ UNCHANGED <<z>>

case1 ==
    /\ pc="LOOP" /\  y1  \leq 100
    /\ y1'=y1+11 /\ y2'=y2+1
    /\  UNCHANGED <<z,pc>>

case2 ==
    /\ pc="LOOP" /\  y1  > 100
    /\ pc'="OBS"
    /\  UNCHANGED <<z,y1,y2>>

case21 ==
    /\ pc="OBS"  /\ y2#1
    /\ y1'=y1-10 /\ y2'=y2-1
    /\ pc'="LOOP"
    /\  UNCHANGED <<z>>

case22 ==
    /\ pc="OBS" /\ y2=1
    /\ z'=y1-10 /\ pc'="HALT"
    /\ UNCHANGED <<y1,y2>>
    --------------------

ePrint == pc="HALT" /\ PrintT(z)/\ UNCHANGED <<y1,y2,z,pc>>

-----------
Next == start  \/ case1 \/ case2 \/ case21 \/ case22 \/ UNCHANGED <<y1,y2,z,pc>> \/
    ePrint
init1 == y1 \in Int /\ y2 \in Int /\ z \in Int /\ pc = "START"
Init  == y1 =u  /\ y2 = u  /\ z =u  /\ pc = "START"

---------------------------

Q1 == pc#"HALT"  (* c prned la valeur HALT *)
Qpartialcorrectness == pc="HALT" => z=IF x>100 THEN x-10 ELSE 91

Qy1 == BF(y1)
Qrte == BF(y1) /\ BF(y2) /\ BF(z)
Question == Qpartialcorrectness /\ Qrte

QQQ== [] Question
QQ == 0 \leq y2 /\ y2 \leq 2

test == QQ

=============================================
```

Listing 2: TLC configuration appex2_2.cfg

```
\* SPECIFICATION
\* Uncomment the previous line and provide the specification name if it's declared
\* in the specification file. Comment INIT / NEXT parameters if you use SPECIFICATION.

CONSTANTS
    greeting = "Hello"
    max = 1000
    u = 1001
    x = 90

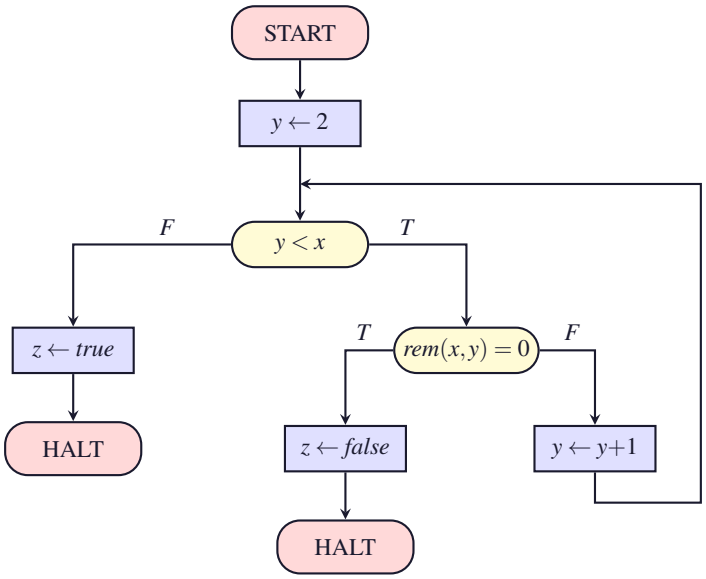
INIT Init
```

```
NEXT Next

PROPERTY
\* Uncomment the previous line and add property names
QQQ
INVARIANT
\* Uncomment the previous line and add invariant names
Question
```

✎ Exercise 2 appex2_3

Consider the following diagram, defining a computation which determines whether a natural number is prime or not (Z. Manna, *Mathematical Theory of Computation*, Fig. 3-19). The flowchart program works over the integers; it should be totally correct with respect to $\varphi(x) : x \geq 2$ and $\psi(x,z) : z \equiv prime(x)$, where $prime(x)$ is true if and only if x is a prime number, that is $\forall k\,.\,(2 \leq k < x \Rightarrow rem(x,k) \neq 0)$.



Question 2.1 Write a TLA model of this computation diagram.

Question 2.2 Write an invariant from annotations that you will define after defining control points.

Question 2.3 Check partial correctness.

Question 2.4 Check the absence of run-time errors.

✔ Solution of exercise 2

Listing 3: TLA$^+$ module `appex2_3.tla`

```
-------------------------------- MODULE appex2_3 --------------------------------
(* prime numbers *)
EXTENDS Naturals,TLC,Integers
CONSTANTS x0,max,u
min == -max
VARIABLES  x,y,z,pc
D == min..max
rte(X) == X#u => X \in D
--------------
(* Précondition *)
ASSUME x0 \in D /\  x0 \geq 2
----------------------------
(* définitions *)

diviseurs(X) == {  m \in 1..X :  X % m = 0}
```

```
prime(X) ==  (diviseurs (X) = {1,X}) /\ X # 1
Locs == {"START","HALT","POINT"}
-------------------------------------------------
start == pc="START" /\ y'=2 /\ pc'="POINT"  /\ UNCHANGED <<x,z>>

case1 ==
    /\ pc="POINT" /\  y \geq x
    /\ z'=TRUE
    /\ pc'="HALT"
    /\ PrintT(y)
    /\  UNCHANGED <<x,y>>

case21 ==
    /\ pc="POINT"  /\ y<x /\ (x % y = 0)
    /\ pc'="HALT"
    /\ z'=FALSE
    /\  UNCHANGED <<x,y>>

case22 ==
    /\ pc="POINT"  /\ y<x /\ (x % y # 0)
    /\ y'=y+1
    /\  UNCHANGED <<x,z,pc>>

eprint ==
    /\ pc="HALT"
    /\ PrintT(z)
    /\ PrintT(x)
    /\ UNCHANGED <<x,y,z,pc>>

  --------
Next ==
    \/ start  \/ case1 \/  case21 \/ case22
    \/ UNCHANGED <<x,y,z,pc>> \/ eprint


Init  == x=x0 /\ y = u /\ z = u  /\  pc = "START"
-------------------------


Q1 == pc # "HALT"  (* pc prned la valeur HALT *)
Q2 == pc ="HALT" => (x=x0) /\ (z <=> (diviseurs(x)={1,x} /\ x#1))
Q3 == pc ="HALT" => (x=x0) /\ (z = prime(x))
Q4 == pc \in Locs
Q5 ==  rte(x) /\ rte(y)

Q ==   Q2 /\ Q3 /\ Q4 /\ Q5


=============================================
```

Listing 4: TLC configuration appex2_3.cfg

```
\* SPECIFICATION
\* Uncomment the previous line and provide the specification name if it's declared
\* in the specification file. Comment INIT / NEXT parameters if you use SPECIFICATION.

CONSTANTS
    greeting = "Hello"
    x0 = 23
    max = 1000
    u = 100000

INIT Init
NEXT Next

\* PROPERTY
\* Uncomment the previous line and add property names

INVARIANT
\* Uncomment the previous line and add invariant names

Q
CHECK_DEADLOCK TRUE
```

✎ Exercise 3 appex2_4

The module truc solves a very classical problem in computer science: *find a path from an input vertex to output vertices, assumed to be exit vertices*.

Question 3.1 To find a path from *input* to one of the vertices of *output*, a safety question must be asked to our verification system. Give a safety question to ask, allowing us to find a path from *input* to a vertex of *output*.

Question 3.2 We want to use this technique to find a path in a maze. A maze is represented by a square matrix of size n. For each element $<< i, j >>$ of the matrix, we then define the communicating neighbours with the function lab, which maps $<< i, j >>$ to the elements that can be reached in one move. For instance, the possible move from $<< 1,1 >>$ is $<< 2,1 >>$; the possible move from $<< 2,1 >>$ is $<< 2,2 >>$ or $<< 1,1 >>$; the possible moves from $<< 2,2 >>$ are $<< 2,3 >>$, $<< 3,2 >>$ or $<< 2,1 >>$, ...

```
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 {}

                    ]
```

Modify the module truc to handle this problem, and give the question to ask in order to find an exit.

MODULE *truc*

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

$n \triangleq 10$
$nodes \triangleq 1..n$
$l \triangleq [i \in 1..n \mapsto$ 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
$\{\}$
$]$

$Init \triangleq p = 1$
$M(i) \triangleq \wedge i \in l[p]$
$\wedge p' = i$
$Next \triangleq \exists i \in 1..n : M(i)$

✔ Solution of exercise 3

Listing 5: TLA$^+$ module appex2_4.tla

```
1 ---------------------------------- MODULE appex2_4 ----------------------------------
2 EXTENDS Integers,TLC
3 VARIABLES  p
4 CONSTANTS input,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 $n : TRUE$
$TLCSet(i, v) \triangleq TRUE$

$d :> e \triangleq [x \in \{d\} \mapsto e]$
$f \,@@\, g \triangleq [x \in (\text{DOMAIN } f) \cup (\text{DOMAIN } g) \mapsto$
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 $x : TRUE$

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

$TLCEval(v) \triangleq v$

Figure 1: Modules Naturals.tla and TLC.tla

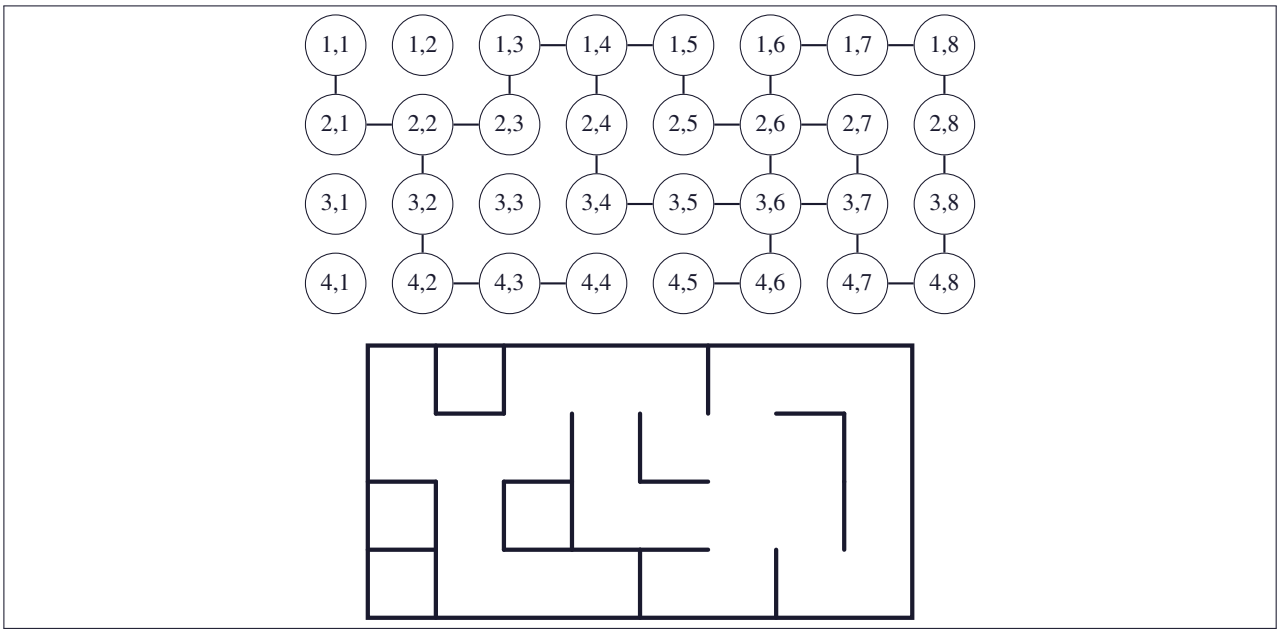


Figure 2: Maze

```
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 {<<1,2>>} ELSE
                     IF x=1 /\ y=2  THEN {<<1,1>>,<<1,3>>,<<2,2>>} ELSE
                     IF x=1 /\ y=3 THEN {<<1,2>>} ELSE
                     IF x=2 /\ y=2 THEN {<<1,2>>}
                     ELSE {}

                     ]

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

Initlab == p = input
ML(q) == /\ q \in lab[p]
         /\ p'=q
Nextlab ==  \E q \in  nodes \X nodes : ML(q)

Sortie == p \notin output

================================
================================================================================
```

Listing 6: TLC configuration `appex2_4.cfg`

```
\* SPECIFICATION
\* Uncomment the previous line and provide the specification name if it's declared
\* in the specification file. Comment INIT / NEXT parameters if you use SPECIFICATION.

CONSTANTS
```

```
    greeting = "Hello"
    input = {1}
    output = {5}

INIT Init
NEXT Next

\* PROPERTY
\* Uncomment the previous line and add property names

INVARIANT
\* Uncomment the previous line and add invariant names
Sortie
CHECK_DEADLOCK FALSE
```