

Cours Modélisation et vérification des systèmes informatiques  
 Exercices (avec les corrections)

Modélisation TLA<sup>+</sup> (2)

par Dominique Méry

18 novembre 2025

**Exercice 1** Soit le réseau de Petri de la figure 1.

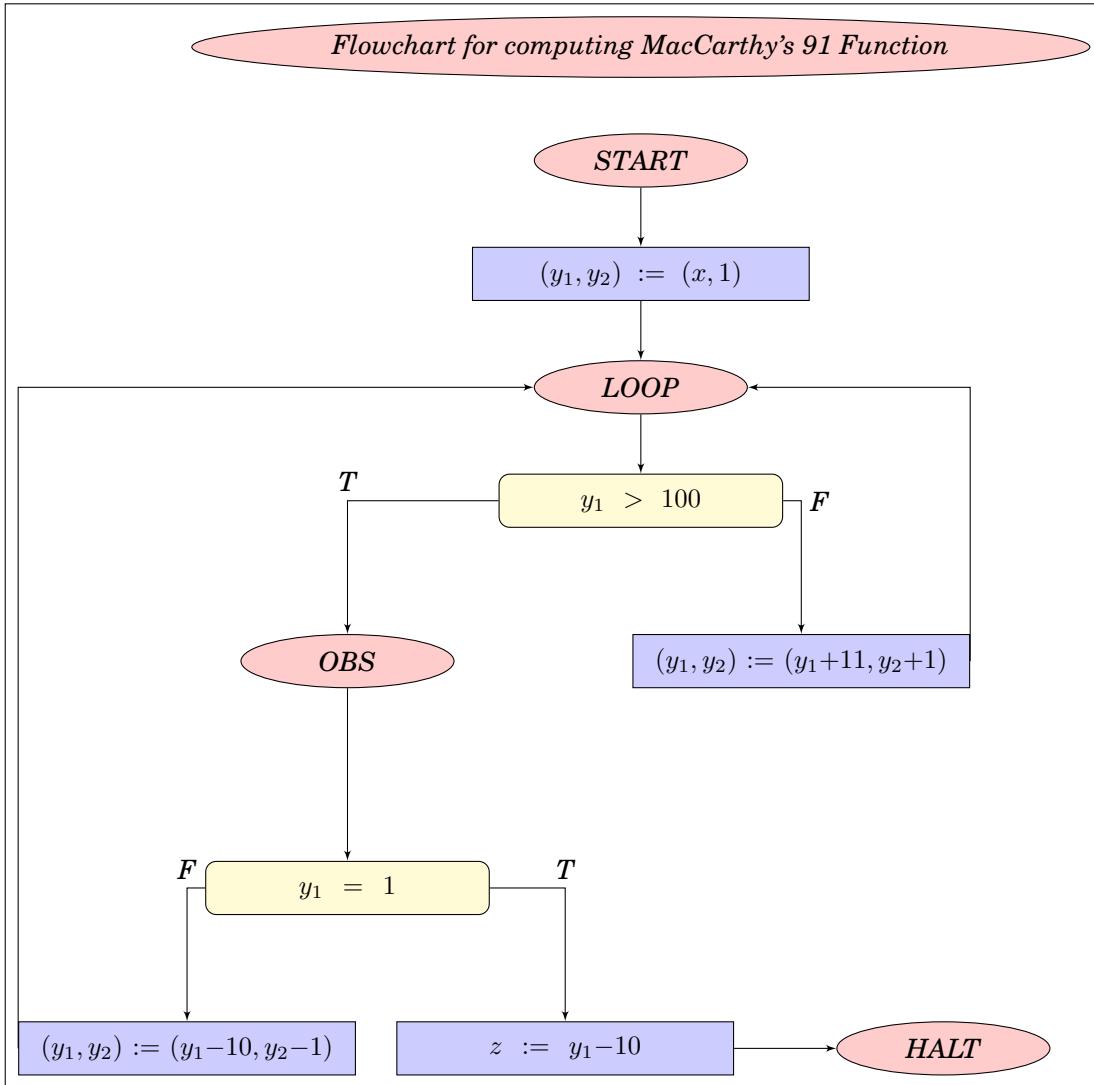
**Question 1.1** Déterminer les conditions initiales.

**Question 1.2** Déterminer les relations modélisant les transitions.

**Question 1.3** Valider les propriétés et les hypothèses que vous pourrez faire sur ce réseau de Petri.

Le fichier `appex2_1.tla` donne les détails de la modélisation avec TLA+.

**Exercice 2** Soit l'organigramme suivant proposé par Z. Manna dans son ouvrage *Mathematical Theory of Computation*.



**Question 2.1** Construire un module TLA<sup>+</sup> modélisant les différents pas de calcul.

**Question 2.2** Evaluer l'algorithme en posant des questions de sûreté suivantes :

1. l'algorithme est partiellement correct.
2. l'algorithme n'a pas d'erreurs à l'exécution.

Le fichier appex2\_2.tla donne les détails de la modélisation avec TLA<sup>+</sup>.

**Exercice 3** Soit le schéma suivant définissant un calcul déterminant si un nombre entier naturel est premier ou non.

Consider the flowchart program (Fig. 3-19) over the integers for deciding whether or not a given integer  $x$  is a prime number. Prove that it is totally correct wrt  $\varphi(x)$ :  $x \geq 2$  and  $\psi(x, z)$ :  $z \equiv \text{prime}(x)$ , where  $\text{prime}(x)$  is true if and only if  $x$  is a prime number; that is  $\forall k [2 \leq k < x \Rightarrow \text{rem}(x, k) \neq 0]$ .

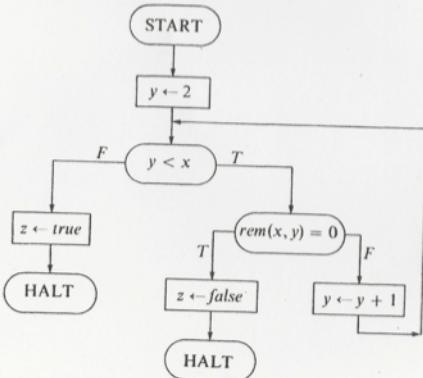


Figure 3-19 Flowchart program for deciding whether or not  $x$  is a prime number.

**Question 3.1** Ecrire un modèle TLA modélisant ce schéma de calcul.

**Question 3.2** Ecrire un invariant à partir d'annotations que vous définirez après avoir défini des points de contrôle.

**Question 3.3** Vérifier la correction partielle

**Question 3.4** Vérifier l'absence d'erreurs à l'exécution.

Le fichier appex2\_3.tla donne les détails de la modélisation avec TLA+.

**Exercice 4** 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 4.1** 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*.

**Question 4.2** 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 {}
      ]
  
```

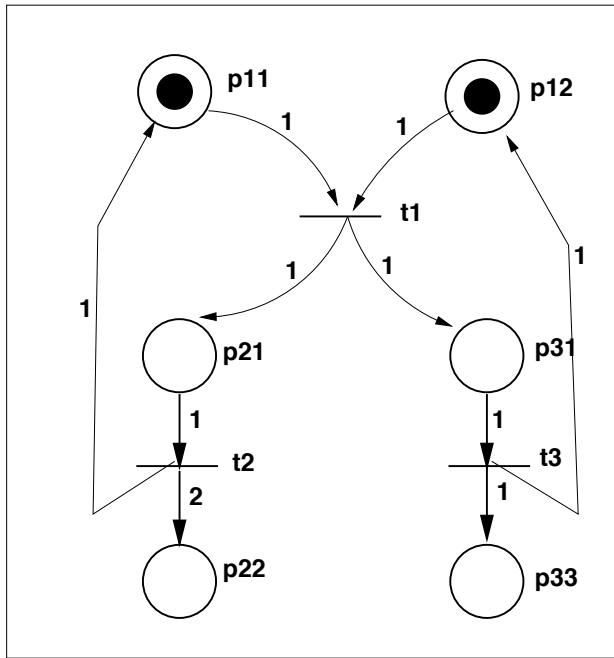


FIGURE 1 – Réseau de Petri

*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

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)

```

---

Le fichier `appex2_4.tla` donne les détails de la modélisation avec TLA+.

---

MODULE *Naturals*

---

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

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

---

MODULE *TLC*

---

LOCAL INSTANCE *Naturals*  
LOCAL INSTANCE *Sequences*

$$\begin{aligned}
 Print(out, val) &\triangleq val \\
 PrintT(out) &\triangleq \text{TRUE} \\
 Assert(val, out) &\triangleq \text{IF } val = \text{TRUE} \text{ THEN } \text{TRUE} \\
 &\quad \text{ELSE CHOOSE } v : \text{TRUE} \\
 JavaTime &\triangleq \text{CHOOSE } n : n \in \text{Nat} \\
 TLCGet(i) &\triangleq \text{CHOOSE } n : \text{TRUE} \\
 TLCSet(i, v) &\triangleq \text{TRUE}
 \end{aligned}$$

$$\begin{aligned}
 d :> e &\triangleq [x \in \{d\} \mapsto e] \\
 f @\!@ g &\triangleq [x \in (\text{DOMAIN } f) \cup (\text{DOMAIN } g) \mapsto \\
 &\quad \text{IF } x \in \text{DOMAIN } f \text{ THEN } f[x] \text{ ELSE } g[x]] \\
 Permutations(S) &\triangleq \\
 &\{f \in [S \rightarrow S] : \forall w \in S : \exists v \in S : f[v] = w\}
 \end{aligned}$$

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

$$\begin{aligned}
 SortSeq(s, Op(\_, \_)) &\triangleq \\
 \text{LET } Perm &\triangleq \text{CHOOSE } p \in Permutations(1 .. Len(s)) : \\
 &\quad \forall i, j \in 1..Len(s) : \\
 &\quad \quad (i < j) \Rightarrow Op(s[p[i]], s[p[j]]) \vee (s[p[i]] = s[p[j]]) \\
 \text{IN } &[i \in 1..Len(s) \mapsto s[Perm[i]]]
 \end{aligned}$$

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

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

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

$$TLCEval(v) \triangleq v$$

FIGURE 2 – Modules *Naturals.tla* et *TLC.tla*

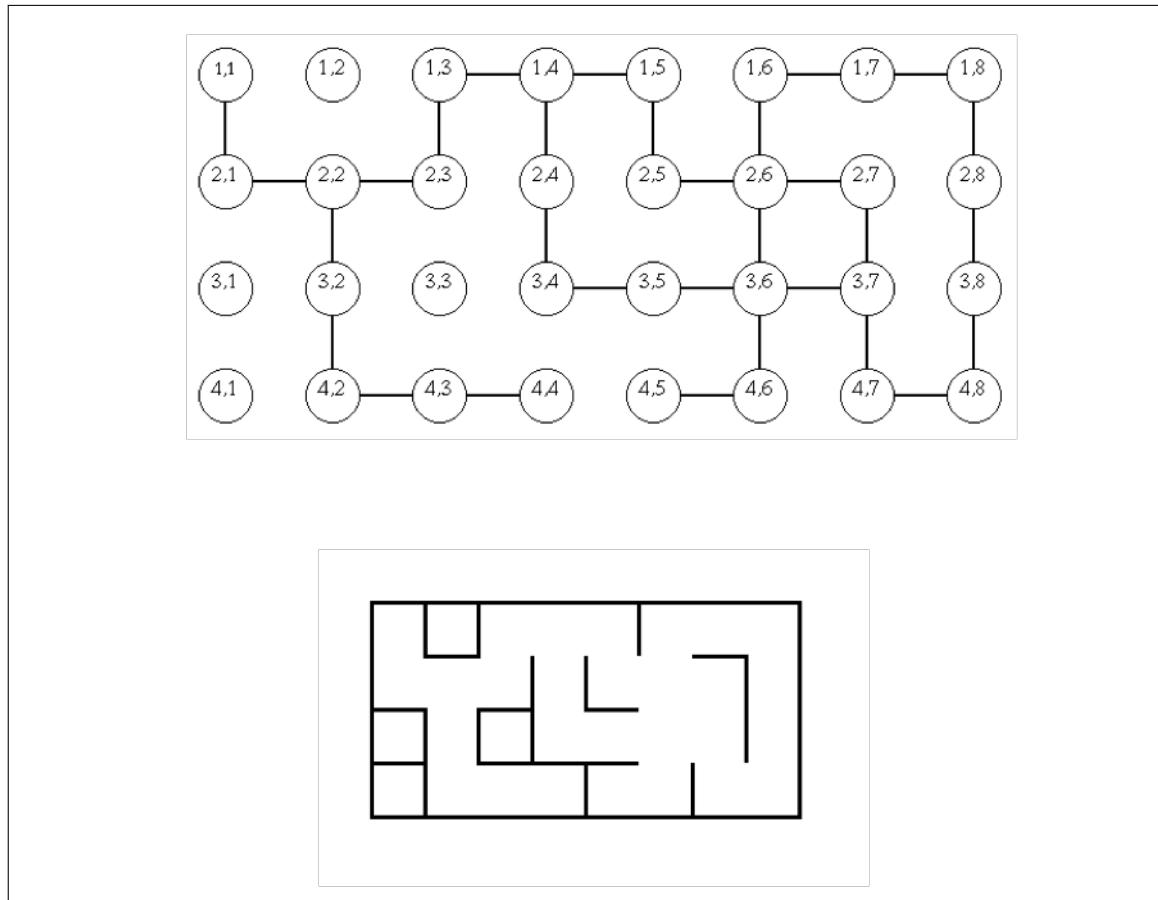


FIGURE 3 – Labyrinthe