

Cours Algorithmique des systèmes parallèles et distribués  
Exercices  
Série : PlusCal pour la programmation répartie ou concurrente (II)  
par Dominique Méry  
3 janvier 2026

**Exercice 1** *pluscalappspd22.tla*

Compléter le module *pluscalappspd22.tla* en proposant une assertion *Q1* correcte.

```
----- MODULE pluscalappspd22 -----
EXTENDS Integers, Sequences, TLC, FiniteSets
(*
--wf
--algorithm ex1{
variables x = 0;

process (one = 1)
variables u;
{
  A:
    u := x+1;
  AB:
    x := u;
  B:
    x := x +1;
};

process (two = 2)
{
  C:
    x := x - 1;
  D:
    assert E2;
};

}
end algorithm;

*)

=====
```

**Exercice 2** *pluscalappaspd33.tla*

Compléter le module *pluscalappaspd33.tla* en proposant deux assertions *R1* et *R2* correctes.

```
----- MODULE pluscalappaspd33 -----
EXTENDS Integers, Sequences, TLC, FiniteSets
(*
--wf
--algorithm ex3{
variables x = 0, y = 2;

process (one = 1)
variable u;
{
  A:
  u := x+1;
  AB:
  x := u;
  B:
  y := y -1;
  C:
  assert E31;
};

process (two = 2)
{
  D:
  x := x - 1;
  E:
  y:=y+2;
  F:
  x:= x+2;
  G:
  assert E32;
};

}
end algorithm;

*)
\
====
```

**Exercice 3** *qquestion1.tla* voir Figure 1

On considère un système formé de deux processus *one* et *two* assurant les calculs suivants :

- *one* : le processus envoie les entiers pairs entre 0 et  $N$  via un canal de communication à *two*.
- *two* : le processus reçoit les valeurs envoyées par *one* et ajoute la valeur reçue à la variable  $s$ .
- *three* : le processus fait un calcul de la somme des entiers de 0 à  $N/4$ .

On suppose que  $N$  est divisible par 4..

**Question 3.1** Afin de vérifier que le calcul effectué par les deux processus est correct, on décide de vérifier que, quand tous les processus ont terminé la variable  $result$  contient la somme des entiers pairs entre 0 et  $N$ .

En utilisant le fichier *qquestion1a.tla*, ajouter une propriété de sûreté *safety1* qui énonce la correction de cet algorithme.

**Question 3.2** On décide de calculer avec le processus *three* la somme des entiers de 0 à  $N\%4$ . Proposer une propriété à vérifier afin de montrer que le calcul du processus *two* est correct.

**Exercice 4** *qquestion2a.tla* voir Figure 2

Soit le petit module *qquestion2a.tla*.

Donner les deux expressions  $A1$  et  $A2$  à placer dans les parties *assert* afin que la vérification ne détecte pas d'erreurs dans cette assertion. Par exemple, on pourrait proposer  $(x = 1 \vee x = 2) \wedge (y = 0 \vee y = 5)$  mais il vous appartient de simuler le programme pluscal pour vérifier que jamais l'assertion que vous proposerez ne soit fausse. La solution *TRUE* fonctionne mais n'est pas autorisée et les expressions demandées doivent contenir une occurrence de  $x$  au moins et une occurrence de  $y$ .

**Exercice 5** *petri2023.tla*

La figure 3 est un réseau de Petri modélisant le système des philosophes qui mangent des spaghetti.

**Question 5.1** Traduire le réseau de Petri sous la forme d'un module TLA, en utilisant le fichier *petri2023.tla*. En particulier, il faut compléter l'initialisation.

**Question 5.2** Est-ce que le réseau peut atteindre un point de deadlock ? Expliquez votre réponse.

**Question 5.3** Proposer une propriété TLA pour répondre à la question suivante, en donnant des explications.

Est-ce que deux philosophes voisins peuvent manger en même temps ?

Listing 1 – qqquestion1.tla

```

----- MODULE question1a -----
EXTENDS Integers, Sequences, TLC, FiniteSets
CONSTANTS N
ASSUME N % 4 = 0
(*
--algorithm algo {
variable
    canal = <<>>;
    witness = -1;
    result = -1;

\* Macro for sending primitive: sending a message m on the fifo channel chan
macro Send(m, chan) {
    chan := Append(chan, m);
};

\* Macro for receiveinbg primitive: receiving
a message m on the fifo channel chan
macro Recv(v, chan) {
    await chan # <<>>;
    v := Head(chan);
    chan := Tail(chan);
};

process (one = 1)
variable
    x = 0;
{
    w:while (x <= N) {
        a:x := x + 1;
        b:if ( x % 4 = 0) {
            c: Send(x,canal);
        };
    };
    d: Send(-1,canal);
};

process (two = 2)
variable s = 0,mes;
{
    w:while (TRUE) {
        a: if (canal # <<>>) {
            b:Recv(mes,canal);
            c:if (mes # -1) { d: s := s +mes;}
            else {e: goto f;};
        };
        f: print <<s>>;
        g: result := s;
    };
};

process (three = 3)
variable
    i = 0;
    s = 0;
    b = N \div 4;
{
    w:while ( i<= b) {
        a:i := i + 1;
        b: s := s +i;
    };
};

```

Listing 2 – qqquestion2a.tla

```

----- MODULE qqquestion2a -----
EXTENDS Integers, Sequences, TLC, FiniteSets

(*
--wf
--algorithm ex3{
variables x = 0, y = 8;

process (one = 1)
{
  A:
    x := x + 1;
  B:
    y := y - 1;
  C:
    assert A1;
};

process (two = 2)
{
  D:
    x := x - 1;
  E:
    y:=y+2;
  F:
    x:= x+2;
    assert A2;
};

}
end algorithm;

*)
=====

```

FIGURE 2 – Programme

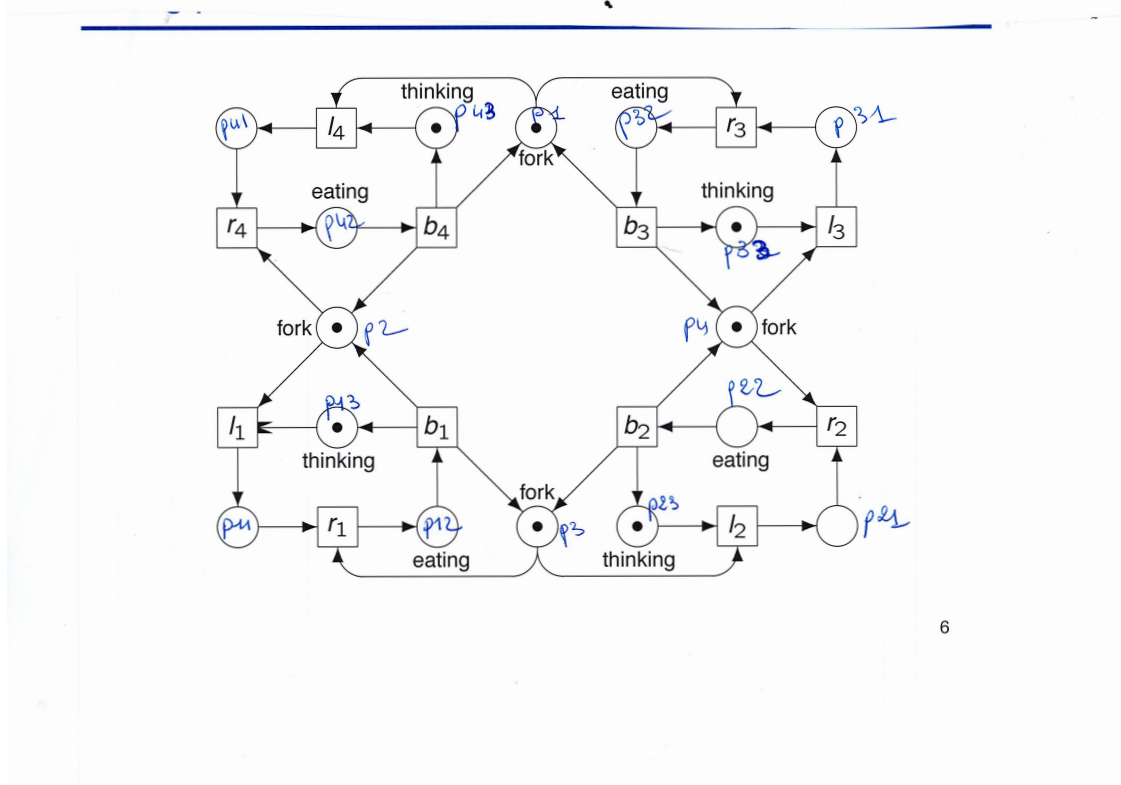


FIGURE 3 – Réseau de Petri

```

----- MODULE examen2023q1 -----
EXTENDS Naturals, TLC
CONSTANTS Places (* d'esigne l'ensemble des places du r'eseau de Petri *)

VARIABLES M (* la variable d'\etat indiquant o\'u se trouvent les jetons *)
-----
ASSUME
  Places \subseteqq {"p11", "p12", "p13", ...}
-----
l1 ==
r1 ==
b1 ==
.....

Init == M = [p \in Places |-> IF p \in {"p1", "p2", "p3", "p4"} THEN 1 ELSE IF .... ]

```

Next == t1 \/ t2 \/ t3 \/ t4 \/ t5

=====