

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

**Exercice 1** (*pluscaltut1.tla*)

Etudier le programme *PlusCal* suivant :

```

----- MODULE pluscaltut1 -----
EXTENDS Integers , Sequences , TLC, FiniteSets
(*
--wf
--algorithm Tut1 {
variables x = 0;

process (one = 1)
{
    A: assert x \in {0,1};
      x := x - 1;
    B: assert x \in {-1,0} ;
      x := x * 3;
    BB: assert x \in {-3,0};
};

process (two = 2)
{
    C: assert x \in {-3,-2,-1,0,1};
      x := x + 1;
    D:
      assert x \in {-2,-1,0,1,2};
};

}
end algorithm;

*)
\* BEGIN TRANSLATION (chksum(pcal) = "6bb757bc" /\ chksum(tla) = "ad730de7")
VARIABLES x, pc

vars == << x, pc >>

ProcSet == {1} \cup {2}

Init == (* Global variables *)

```

```

/\ x = 0
/\ pc = [self \in ProcSet |-> CASE self = 1 -> "A"
                        [] self = 2 -> "C"]

A == /\ pc[1] = "A"
     /\ Assert(x \in {0,1}, "Failure of assertion at line 10, column 6.")
     /\ x' = x - 1
     /\ pc' = [pc EXCEPT ![1] = "B"]

B == /\ pc[1] = "B"
     /\ Assert(x \in {-1,0}, "Failure of assertion at line 12, column 6.")
     /\ x' = x * 3
     /\ pc' = [pc EXCEPT ![1] = "BB"]

BB == /\ pc[1] = "BB"
     /\ Assert(x \in {-3,0}, "Failure of assertion at line 14, column 8.")
     /\ pc' = [pc EXCEPT ![1] = "Done"]
     /\ x' = x

one == A \ / B \ / BB

C == /\ pc[2] = "C"
     /\ Assert(x \in {-3,-2,-1,0,1},
               "Failure of assertion at line 19, column 6.")
     /\ x' = x + 1
     /\ pc' = [pc EXCEPT ![2] = "D"]

D == /\ pc[2] = "D"
     /\ Assert(x \in {-2,-1,0,1,2},
               "Failure of assertion at line 22, column 5.")
     /\ pc' = [pc EXCEPT ![2] = "Done"]
     /\ x' = x

two == C \ / D

(* Allow infinite stuttering to prevent deadlock on termination. *)
Terminating == /\ \A self \in ProcSet: pc[self] = "Done"
              /\ UNCHANGED vars

Next == one \ / two
       \ / Terminating

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

Termination == <>(\A self \in ProcSet: pc[self] = "Done")

```

\\* *END TRANSLATION*

====

**Exercice 2** (*pluscaltut2.tla*)

*Etudier le programme PlusCal suivant :*

----- *MODULE pluscaltut2* -----  
*EXTENDS Integers, Sequences, TLC, FiniteSets*

```
(*  
--algorithm Tut2 {  
  variables x = 0;  
  
  process (one = 1)  
  
    variables temp  
    {  
  
      A:  
        temp := x + 1;  
  
        x := temp;  
  
    };  
  
  process (two = 2)  
  
    variables temp  
    {  
      CC:  
        temp := x + 1;  
  
        x := temp;  
  
    };  
  }  
end algorithm;  
*)
```

====

**Exercice 3** (*pluscaltut3.tla*)

Etudier le programme PlusCal suivant :

```

----- MODULE pluscaltut3 -----
EXTENDS Integers, Sequences, TLC, FiniteSets
(*
--algorithm Tut3 {
variables x = 0;

process (one = 1)
{
  A:
    x := x + 1;
  B:
    await x = 1;
  C:
    print <<"x=",x>>;
};

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

}
end algorithm;

*)

test == (\A i \in ProcSet : pc[i]="Done") => x \in {1, 2}

```

=====

**Exercice 4** *pluscalex1.tla*

Ecrire un programme PlusCal qui traduit le protocole suivant : S envoie une valeur à R

**Exercice 5** *pluscalex2.tla*

*Ecrire un programme PlusCal qui calcule la fonction factorielle de la façon suivante :*

- Un processus calcule  $1 \times 2 \times 3 \dots \times k_1$*
- Un processus calcule  $k_2 \times (k_2 + 1) \times \dots \times N$*
- Les processus stoppent quand la condition  $k_1 < k_2$  est fausse*

**Exercice 6** *pluscalex3.tla*

*Ecrire un programme PlusCal qui calcule la fonction  $L^K$  la façon suivante :*

- Un processus calcule  $L \times \dots \times L$   $k_1$  fois.*
- Un processus calcule  $L \times \dots \times L$   $k_2$  fois.*
- Les processus stoppent quand la condition  $k_1 + k_2 < L$  est fausse*

Cours Algorithmique des systèmes parallèles et distribués  
 Exercices  
 Série : PlusCal pour la programmation répartie ou concurrente (II)  
 par Dominique Méry  
 21 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

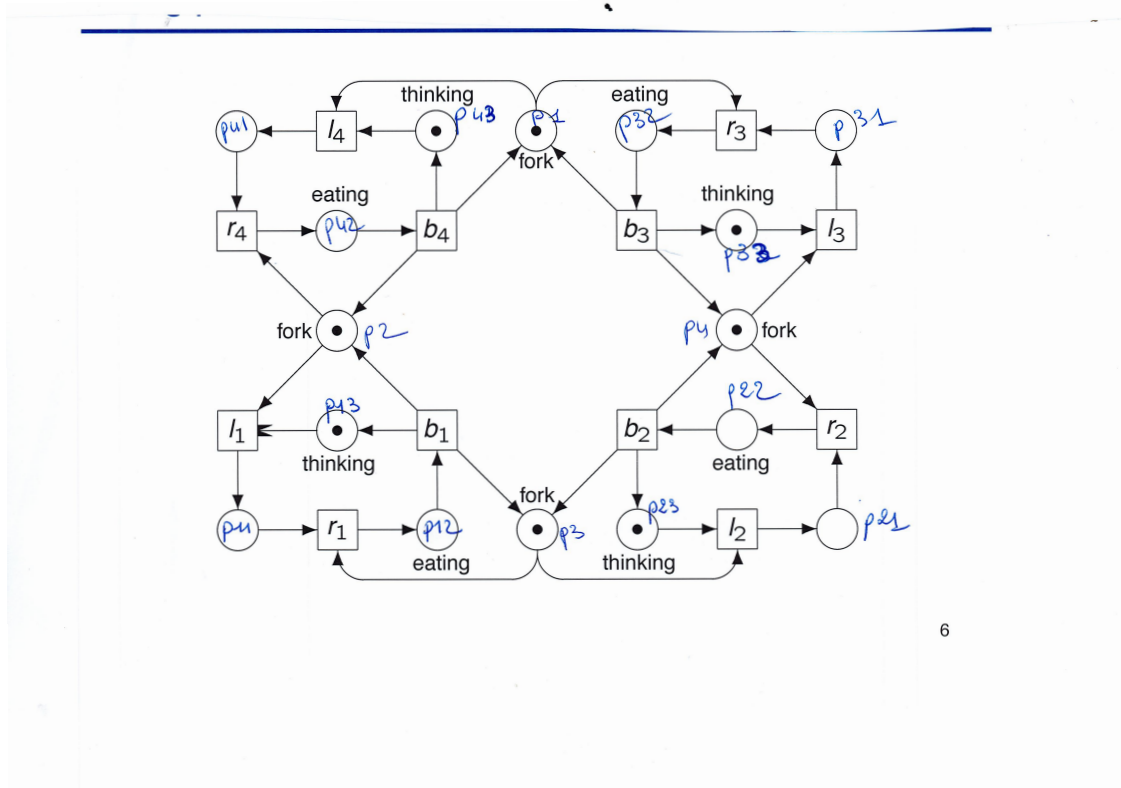


FIGURE 3 – Réseau de Petri

```

----- 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 \subseteq {"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

=====