

Cours Algorithmique des systèmes parallèles et distribués
Exercices

Série :PlusCal pour la programmation répartie ou concurrente (I)
par Dominique Méry
12 février 2026

Exercice 1 (*pluscaltut1.tla*)

Etudier, compléter et analyser le programme PlusCal suivant :

----- MODULE *pluscaltut1q* -----

EXTENDS Integers , Sequences , TLC, FiniteSets

(*

--wf

--algorithm Tut1 {

variables x = 0;

process (one = 1)

{

 A: assert x \in ???;

 x := x - 1;

 B: assert x \in ???? ;

 x := x * 3;

 BB: assert x \in ???;

};

process (two = 2)

{

 C: assert x \in ???;

 x := x + 1;

 D:

 assert x \in ??;

};

}

end algorithm;

*)

safepc == pc[1] = "Done" /\ pc[2] = "Done" => ??

=====

Exercice 2 (*pluscaltut2.tla*)

Etudier, compléter et analyser le programme PlusCal suivant :

```

----- MODULE pluscaltut2q -----
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
{
B:
temp := x + 1;

x:= temp;

};

}
end algorithm;

*)

saferpc == pc[1] = "Done" /\ pc[2] = "Done" => ??
=====
```

Exercice 3 (*pluscaltut3.tla*)

Etudier le programme PlusCal suivant :

```

----- MODULE pluscaltut3q -----
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;
*)

=====

```

Exercice 4 pluscaltut4.tla

Ecrire un programme PlusCal qui traduit le protocole suivant :

- S envoie une valeur val à R
- R reçoit la même valeur val

Exercice 5 pluscaltut5.tla

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

- Un processus P1 calcule $1 \times 2 \times 3 \dots \times k_1$
- Un processus P2 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 pluscaltut6.tla

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

- Un processus P_1 calcule $L \times \dots \times L$ k_1 fois.
- Un processus P_2 calcule $L \times \dots \times L$ k_2 fois.
- Les processus P_3 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
12 février 2026

Exercice 1 pluscaltut7.tla

Compléter le module pluscaltut7q.tla en proposant une assertion Q1 correcte.

```
----- MODULE pluscaltut7q -----
EXTENDS Integers, Sequences, TLC, FiniteSets
(*

--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 x \in {-1,0,1,2};
    \*E2;
};

}
end algorithm;

*)
\* BEGIN TRANSLATION (chksum(pcal) = "9ff5fa47" /\ chksum(tla) = "69a4c6ee")
CONSTANT defaultInitValue
VARIABLES x, pc, u
```

```

vars == << x, pc, u >>

ProcSet == {1} \cup {2}

Init == (* Global variables *)
  /\ x = 0
  (* Process one *)
  /\ u = defaultInitValue
  /\ pc = [self \in ProcSet |-> CASE self = 1 -> "A"
           [] self = 2 -> "C"]

A == /\ pc[1] = "A"
  /\ u' = x+1
  /\ pc' = [pc EXCEPT !{1} = "AB"]
  /\ x' = x

AB == /\ pc[1] = "AB"
  /\ x' = u
  /\ pc' = [pc EXCEPT !{1} = "B"]
  /\ u' = u

B == /\ pc[1] = "B"
  /\ x' = x +1
  /\ pc' = [pc EXCEPT !{1} = "Done"]
  /\ u' = u

one == A \vee AB \vee B

C == /\ pc[2] = "C"
  /\ x' = x - 1
  /\ pc' = [pc EXCEPT !{2} = "D"]
  /\ u' = u

D == /\ pc[2] = "D"
  /\ Assert(x \in {-1,0,1,2},
            "Failure of assertion at line 25, column 5.")
  /\ pc' = [pc EXCEPT !{2} = "Done"]
  /\ UNCHANGED << x, u >>

two == C \vee 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 pluscaltut8.tla

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

```

----- MODULE pluscaltut8q -----
EXTENDS Integers, Sequences, TLC, FiniteSets
CONSTANTS
  k,l
(*

--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 x \in -k..k /\ y \in -l..l;
  \*{0,1,2,3} /\ y \in {1,3}; \* E31;
};

process (two = 2)
{
  D:
  x := x - 1;
  E:
  y:=y+2;
```

```

F:
  x:= x+2;
G:
  assert x \in -k..k /\ y \in -l..l; \*x \in {0,1,2,3} /\ y \in {1,3,4};    \* E3
};

}
end algorithm;

*)
\* BEGIN TRANSLATION (chksum(pcal) = "43ec0ed4" /\ chksum(tla) = "4bfca968")
CONSTANT defaultValue
VARIABLES x, y, pc, u

vars == << x, y, pc, u >>

ProcSet == {1} \cup {2}

Init == (* Global variables *)
  /\ x = 0
  /\ y = 2
  (* Process one *)
  /\ u = defaultValue
  /\ pc = [self \in ProcSet |-> CASE self = 1 -> "A"
           [] self = 2 -> "D"]

A == /\ pc[1] = "A"
  /\ u' = x+1
  /\ pc' = [pc EXCEPT ![1] = "AB"]
  /\ UNCHANGED << x, y >>

AB == /\ pc[1] = "AB"
  /\ x' = u
  /\ pc' = [pc EXCEPT ![1] = "B"]
  /\ UNCHANGED << y, u >>

B == /\ pc[1] = "B"
  /\ y' = y -1
  /\ pc' = [pc EXCEPT ![1] = "C"]
  /\ UNCHANGED << x, u >>

C == /\ pc[1] = "C"
  /\ Assert(x \in -k..k /\ y \in -l..l,
            "Failure of assertion at line 20, column 6.")
  /\ pc' = [pc EXCEPT ![1] = "Done"]

```

```

/\ UNCHANGED << x, y, u >>

one == A \/ AB \/ B \/ C

D == /\ pc[2] = "D"
/\ x' = x - 1
/\ pc' = [pc EXCEPT !(2) = "E"]
/\ UNCHANGED << y, u >>

E == /\ pc[2] = "E"
/\ y' = y+2
/\ pc' = [pc EXCEPT !(2) = "F"]
/\ UNCHANGED << x, u >>

F == /\ pc[2] = "F"
/\ x' = x+2
/\ pc' = [pc EXCEPT !(2) = "G"]
/\ UNCHANGED << y, u >>

G == /\ pc[2] = "G"
/\ Assert(x \in -k..k /\ y \in -l..l,
           "Failure of assertion at line 33, column 5.")
/\ pc' = [pc EXCEPT !(2) = "Done"]
/\ UNCHANGED << x, y, u >>

two == D \/ E \/ F \/ G

(* 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 3 pluscaltut9.tla Fig : ??

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

- one : le processus envoie les entiers multiples de 4 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 Ecrire un programme PlusCal qui effectue ces tâches.

Question 3.2 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.

Ajouter une propriété de sûreté safety1 qui énonce la correction de cet algorithme en vérifiant que la valeur de result et la valeur de witness sont égales en fin de calcul.

Exercice 4 pluscaltut10.tla voir Figure 1

Soit le petit module pluscaltut10.tla.

Donner les expressions ???? 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.

Cours Algorithmique des systèmes parallèles et distribués
 Exercices
 Série 1 Modélisation, programmation et vérification en TLA⁺
 par Dominique Méry
 12 février 2026

Modélisation et vérification avec TLA⁺

RAPPELS

Un réseau de Petri est un uple $R=(S,T,F,K,M,W)$ tel que

- S est l'ensemble (fini) des places.
- T est l'ensemble (fini) des transitions.
- $S \cap T = \emptyset$
- F est la relation du flôt d'exécution : $F \subseteq S \times T \cup T \times S$
- K représente la capacité de chaque place : $K : S \rightarrow \text{Nat}$.
- M représente le initial marquage chaque place :
- $M : S \rightarrow \text{Nat}$ et vérifie la condition $\forall s \in S : M(s) \leq K(s)$.
- W représente le poids de chaque arc : $W : F \rightarrow \text{Nat}$

Listing 1 – pluscaltut10q.tla

```
----- MODULE pluscaltut10 -----
EXTENDS Integers , Sequences , TLC, FiniteSets
CONSTANTS k, l

(*

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

process (one = 1)
{
  A:
    x := x + 1;
  B:
    y := y -1;
  C:
    assert x \in ?? /\ y \in ???;
};

process (two = 2)
{
  D:
    x := x - 1;
  E:
    y:=y+2;
  F:
    x:= x+2;
  assert x \in ?? /\ y \in ??;
};

}

end algorithm;

*)

=====
```

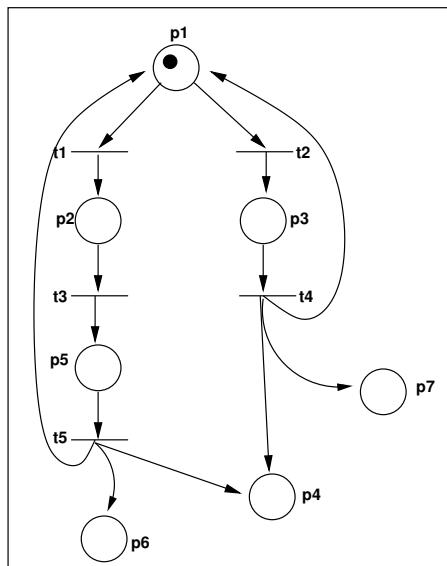
FIGURE 1 – Programme PlusCal pluscaltut10.tla

- un marquage M pour R est une fonction de S dans Nat :
 $M \in S \rightarrow \text{Nat}$ et respectant la condition $\forall s \in S : M(s) \leq K(s)$.
 - une transition t de T est activable à partir de M un marquage de R si
 - $\forall s \in \{ s' \in S \mid (s',t) \in F \} : M(s) \geq W(s,t)$.
 - $\forall s \in \{ s' \in S \mid (t,s') \in F \} : M(s) \leq K(s) - W(s,t)$.
 - Pour chaque transition t de T , $\text{Pre}(t)$ est l'ensemble des places conduisant à t et $\text{Post}(t)$ est l'ensemble des places pointées par un lien depuis t :
 $\text{Pre}(t) = \{ s' \in S : (s',t) \in F \}$ et $\text{Post}(t) = \{ s' \in S : (t,s') \in F \}$
 - Soit une transition t de T activable à partir de M un marquage de R :
 - $\forall s \in \{ s' \in S \mid (s',t) \in F \} : M(s) \geq W(s,t)$.
 - $\forall s \in \{ s' \in S \mid (t,s') \in F \} : M(s) \leq K(s) - W(s,t)$.
 - un nouveau marquage M' est défini à partir de M par : $\forall s \in S$,

$$M'(s) = \begin{cases} M(s) - W(s,T), & \text{SI } s \in \text{PRE}(T) - \text{POST}(T) \\ M(s) + W(T,s), & \text{SI } s \in \text{POST}(T) - \text{PRE}(T) \\ M(s) - W(s,T) + W(T,s), & \text{SI } s \in \text{PRE}(T) \cap \text{POST}(T) \\ M(s), & \text{SINON} \end{cases}$$
-

Exercice 1 (*petri13.tla*)

Soit le réseau de Petri suivant :

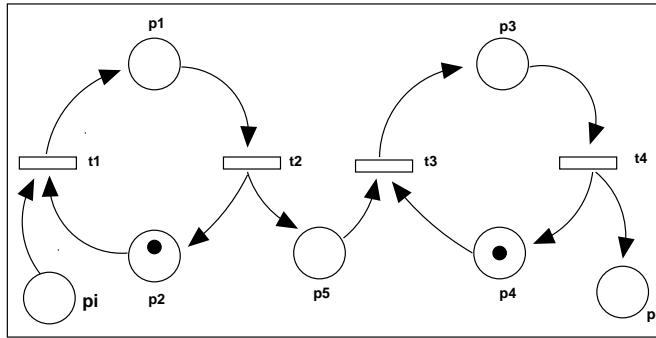


Question 1.1 Modéliser ce réseau de Petri avec TLA^+ .

Question 1.2 Etudier ce réseau en proposant et en vérifiant des invariants à l'aide des outils.

Exercice 2 (*petri10.tla*)

On considère le réseau suivant :



Question 2.1 Traduire ce réseau en un module TLA⁺.. Pour cela, on donnera la définition des quatre transitions t_1, t_2, t_3, t_4 . On ne tiendra pas compte de la capacité des places : les places ont une capacité d'au plus un jeton, sauf la place p_i qui peut contenir N jetons, la place p_5 peut contenir au plus B jetons et la place p_o peut contenir au plus Q .

Question 2.2 Donner une relation liant les places p_o, p_1, p_3, p_5, p_i et la valeur N . Justifier la réponse.

Question 2.3 Si on suppose que la place p_o peut contenir au plus Q jetons, donnez une condition sur Q pour que tous les jetons de p_i soient consommés un jour. Justifier la réponse.

Question 2.4 Expliquer ce que modélise ce réseau de Petri.

Exercice 3 (*petri14.tla*)

La figure 2 est un réseau de Petri modélisant le système des philosophes qui mangent des spaghetti.

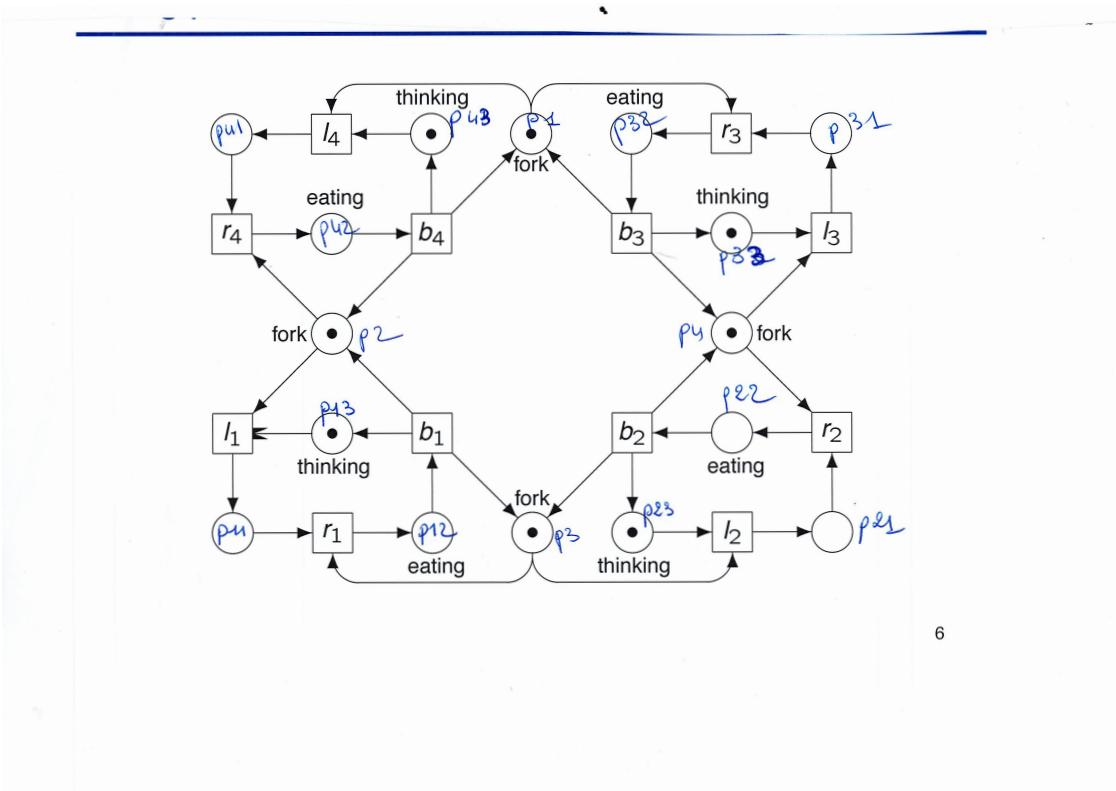
Question 3.1 Traduire le réseau de Petri sous la forme d'un module TLA.

Question 3.2 Est-ce que le réseau peut atteindre un point de deadlock ? Expliquez votre réponse.

Question 3.3 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 ?

Exercice 4 (*disapp_td1_ex1.tla*)



6

FIGURE 2 – Réseau de Petri

Question 4.1 Modéliser sous forme d'un module TLA⁺ le réseau de Petri de la figure 3. Donner une instantiation possible des constantes. Préciser les conditions initiales correspondant à un système avec cinq (5) processeurs et deux (2) bus.

Question 4.2 On désire analyser le comportement de ce réseau et, pour cela, on souhaite savoir si la place p₅ contiendra au moins un jeton. Expliquer comment on doit procéder pour obtenir une réponse en un temps fini. Préciser le message donné par le système TLAPS.

Question 4.3 Est-ce que le réseau peut atteindre un point de deadlock ?

Question 4.4 Enoncez trois propriétés de sûreté de ce réseau établissant une relation entre au moins deux places.

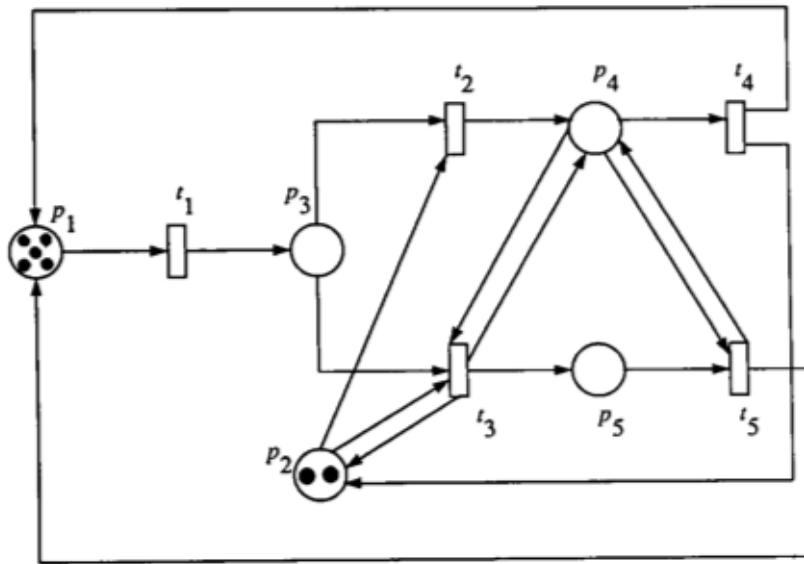


Fig. 14. A Petri-net model of a multiprocessor system, where tokens in p_1 represent active processors, p_2 available buses, p_3 , p_4 , and p_5 processors waiting for, having access to, queued for common memories, respectively.

FIGURE 3 – Réseau de Petri