

Cours Algorithmique des systèmes parallèles et distribués
 Exercices
 Série : PlusCal pour la programmation répartie ou concurrente (II)
 par Alessio Coltellacci et Dominique Méry
 12 mars 2025

Exercice 1 Compléter le module *pluscalappaspd22.tla* en proposant une assertion *Q1* correcte.

```
----- MODULE pluscalappaspd22 -----
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 Compléter le module *pluscalappaspd33.tla* en proposant deux as-

sertions 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 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 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 Voir figure 3

On considère des populations de clients $\mathcal{P}_i = \{P_{ij} : j \in 1..n_i\}$ avec $i \in 1..n$ et $Q_j, j \in 1..n$ associé à chaque population : le processus Q_i est le serveur de la population \mathcal{P}_i . L'algorithme de la figure 4 met en place la gestion d'une ressource R partagée par les processus $C_1 \dots C_p$ via un serveur S . On décide d'utiliser cet algorithme pour gérer une ressource R partagée par toutes les populations et attribuée aux populations par leur serveur respectif quand ce serveur a le jeton. Le réseau en anneau dans la figure 3 explique les liens possibles entre les processus serveurs Q_i et les populations. La gestion de l'anneau est réalisé comme indiqué dans le fichier *qring.tla* de la figure 3. La gestion d'une population est assurée par le programme du fichier de la figure 4.

Question 5.1 On souhaite tester le protocole ring du fichier *qring.tla* de la figure 3. Expliquer pourquoi le réseau ring de *qring.tla* est correct et effectuer des tests que vous indiquerez dans votre fichier soit en exprimant des propriétés de sûreté ou d'invariance, soit en vérifiant l'absence de blocage. n

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

Question 5.2 La ressource R ne peut être attribuée que par un processus serveur Q qui a le jeton c'est-à-dire que $v[Q] = \text{TOKEN}$ sinon NIL . Q est un des processus sur l'anneau et est numéroté de 0 à N . Modifier le processus Q afin de réaliser cette fonctionnalité d'attribution de la ressource R au processus de la population gérée par Q et répondant à une demande de la population selon le protocole de la figure 4.

Question 5.3 Détailler les propriétés de correction que doit satisfaire ce protocole et vérifier les. En particulier, il faudra montrer que le processus P d'une population donnée reçoit la ressource quand le serveur est en état de lui donner c'est-à-dire qu'il a le jeton.

Exercice 6

La figure 6 est un réseau de Petri modélisant le système des philosophes qui mangent des spaghetti.

Question 6.1 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 6.2 Est-ce que le réseau peut atteindre un point de deadlock ? Expliquez votre réponse.

Question 6.3 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 ?

```

----- 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 \subteq {"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

```

Listing 3 – anneau1

```

----- MODULE qring -----
EXTENDS Integers, Naturals, Sequences, TLC
CONSTANT N,NIL,TOKEN

Remove(i, seq) == [j \in 1..(Len(seq)-1) |-> IF j < i THEN seq[j] ELSE seq[j+1]]

v0 == [i \in 0..N |->NIL]

(*
--algorithm algo {

variable
  v = v0;
  port = [i \in 0..N |-> IF i # 0 THEN <<>> ELSE <<TOKEN>>];

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

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

process (q \in 0..N )
  variable mes;
  {
  s: while (TRUE) {
  cc1: Recv(mes,port[self]);
    test: if (self = 0) { pp: print <<"P[0]:",v>>;};
    cc4: v[self] := mes;
    rr: v[self]:= NIL;
    cc5: Send(TOKEN,port[(self+1) % N]);
  };
  };
};

} \* end algorithm

*)

```

=====

FIGURE 3 – Programme

Listing 4 – pop

```

----- MODULE  qquestion3a
-----
EXTENDS Naturals, Sequences, TLC
CONSTANT p

Remove(i, seq) == [j \in 1..(Len(seq)-1) |-> IF j < i THEN seq[j] ELSE seq[j+1]]

(*
--algorithm  algo {

variable
    requests = <<>>,  reply = [i \in 1..p |-><<>>], msgok = <<>>;

macro Send(m, chan) {
    chan := Append(chan, m);
};

macro Recv(v, chan) {
    await chan # <<>>;  \* could also do Len(chan) > 0 ??
    v := Head(chan);
    chan := Tail(chan);
};

process (C \in 1..p )
    variable request = 0, mes, cs = 0;
    {
    s:  while (TRUE) {
        c1: request := 1;
        c2: Send(self, requests);
        c3: Recv(mes, reply[self]);
        c4: cs := 1;
        c5: request := 0;
        c6: Send(self, msgok);
    };
}

process (Server = 0 )
    variables  cs=0,v;
    {
    r:  while (TRUE) {
        if (requests # <<>> /\ cs=0)
        {
            a: Recv(v, requests);
            b: cs := 1;          8
            c: Send(v, reply[v]);

        } else if (msgok # <<>>)
        {
            d: Recv(v, msgok);
            e: cs := 0;
        } else
        { v:skip;
        }
        ;

        };
    };
};

```

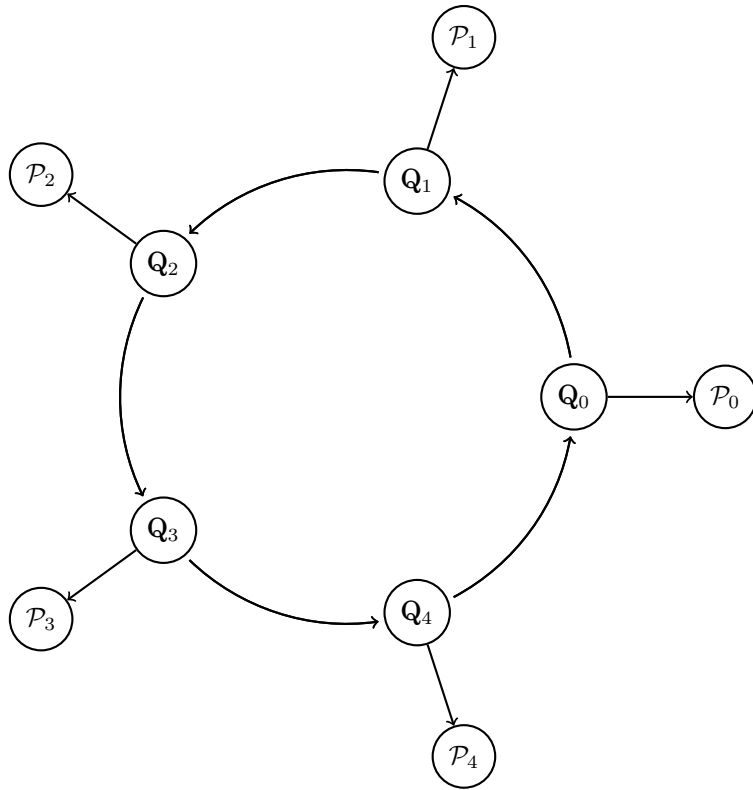
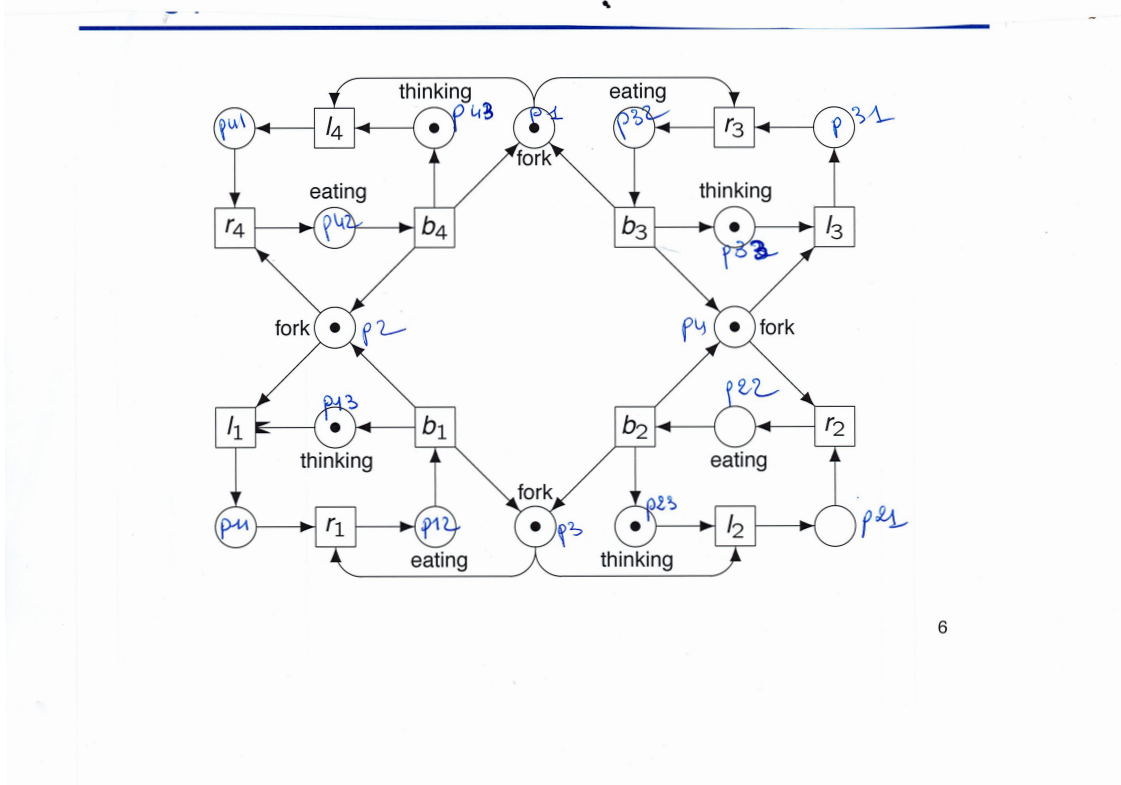



FIGURE 5 – Réseau global



6

FIGURE 6 – Réseau de Petri