Cours Algorithmique des systèmes parallèles et distribués
Exercices
Série 1 Modélisation, programmation et vérification en TLA$^+$
par Alessio Coltellacci et Dominique Méry
20 janvier 2025

Modélisation et vérification avec TLA$^+$

Exercice 1 *(disapp_td1_ex1.tla)*

Question 1.1 *Modéliser sous forme d'un module TLA$^+$ le réseau de Petri de la figure 1. Donner une instanciation possible des constantes. Préciser les conditions initiales correspondant à un système avec cinq (5) processeurs et deux (2) bus.*

Question 1.2 *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. naserie*

Question 1.3 *Est-ce que le réseau peut atteindre un point de deadlock ?*

Question 1.4 *Enoncez trois propriétés de sûreté de ce réseau établissant une relation entre au moins deux places.*

Exercice 2 *(disapp_td1_ex2.tla)*
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.*
- *$S \cap T = \varnothing$*
- *F est la relation du flÔt d'exécution : $F \subseteq S \times T \cup T \times S$*
- *K représente la capacité de chaque place : $K \in S \to Nat$.*
- *M représente le initial marquage chaque place :*
  $M \in S \to Nat$ et vérifie la condition $\forall s \in S : M(s) \leq K(s)$.
- *W représente le poids de chaque arc : $W \in F \to Nat$*
- *un marquage M pour R est une fonction de S dans Nat :*
  $M \in S \to Nat$ 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*
  1. *$\forall s \in \{ s' \in S \mid (s',t) \in F \} : M(s) \geq W(s,t)$.*
  2. *$\forall s \in \{ s' \in S \mid (t,s') \in F \} : M(s) \leq K(s) - W(s,t)$.*
- *Pour chaque transition t de T, Pre(t) est l'ensemble des places conduisant à t et Post(t) est l'ensemble des places pointées par un lien depuis t :*
  $Pre(t) = \{ s' \in S : (s',t) \in F \}$ et $Post(t) = \{ s' \in S : (t,s') \in F \}$

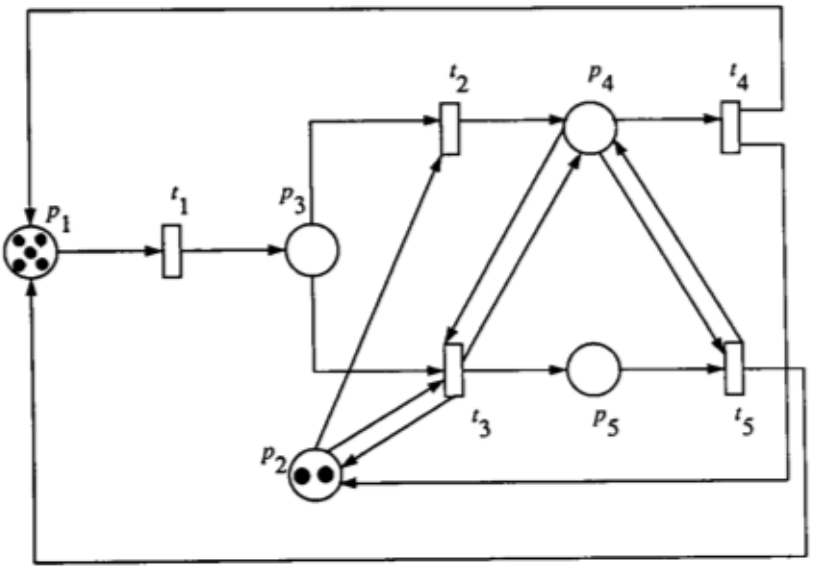


Fig. 14. A Petri-net model of a multiprocessor system, where tokens in p_1 represent active processors, p_2 available buses, p_3, p_4, and p_5 processors waiting for, having access to, queued for common memories, respectively.

FIGURE 1 – Réseau de Petri

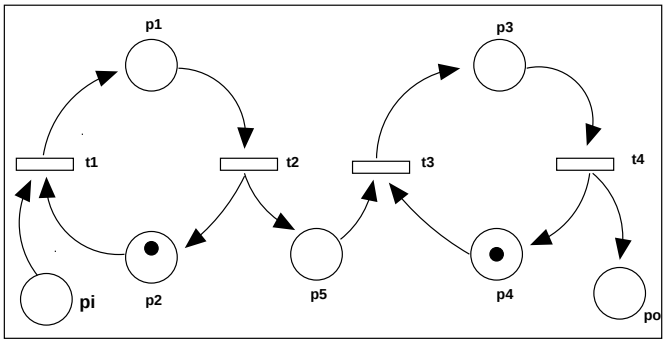
— *Soit une transition t de T activable à partir de M un marquage de R :*

1. $\forall s \in \{ s' \in S \mid (s',t) \in F \} : M(s) \geq W(s,t)$.
2. $\forall s \in \{ s' \in S \mid (t,s') \in F \} : M(s) \leq K(s) - W(s,t)$.

— *un nouveau marquage M' est défini à partir de M par :* $' \forall s \in S$,

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

On considère le réseau suivant :



MODULE $petri10$

EXTENDS $Naturals$, TLC
CONSTANTS $Places$, N, Q, B
VARIABLES M

$t1 \triangleq$
$t2 \triangleq$
$t3 \triangleq$
$t4 \triangleq$
$Init1 \triangleq \quad M = [p \in Places \mapsto \text{IF } p \in \{ \text{"p4"}, \text{"p2"} \} \text{ THEN } 1 \text{ ELSE}$
$\text{IF } p = \text{"pi"} \text{ THEN } N \text{ ELSE } 0]$

$Next \triangleq t1 \vee t2 \vee t3 \vee t4$

Question 2.1 *Traduire ce réseau en un module TLA$^+$ dont le squelette est donné dans le texte. Pour cela, on donnera la définition des quatre transitions* $t1$, $t2$, $t3$,$t4$. *On ne tiendra pas compte de la capacité des places : les places ont une capacité d'au plus un jeton, sauf la place pi qui peut contenir* N *jetons, la place p5 peut contenir au plus B jetons et la place po peut contenir au plus* Q.

Question 2.2 *Donner une relation liant les places po,p1,p3,p5,pi et la valeur* N. *Justifiez votre réponse.*

Question 2.3 *Si on suppose que la place po peut contenir au plus Q jetons, donnez une condition sur Q pour que tous les jetons de pi soient consommés un jour. Justifiez votre réponse.*

Question 2.4 *Expliquez ce que modélise ce réseau de Petri.*

PlusCal pour la programmation répartie ou concurrente

Exercice 3 *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,-2,0,1};
};

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) = "a26cf6c4" /\ chksum(tla) = "72d02274")
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 11, column 6.")
     /\ x' = x - 1
     /\ pc' = [pc EXCEPT ![1] = "B"]

B == /\ pc[1] = "B"
     /\ Assert(x \in {-1,0}, "Failure of assertion at line 13, column 6.")
     /\ x' = x * 3
     /\ pc' = [pc EXCEPT ![1] = "BB"]

BB == /\ pc[1] = "BB"
      /\ Assert(x \in {-3,-2,0,1},
                "Failure of assertion at line 15, 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 20, 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 23, 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 4 *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;

*)
\* BEGIN TRANSLATION (chksum(pcal) = "b54fa406" /\ chksum(tla) = "e84b4125")
\* Process variable temp of process one at line 10 col 11 changed to temp_
CONSTANT defaultInitValue
```

```
VARIABLES x, pc, temp_, temp

vars == << x, pc, temp_, temp >>

ProcSet == {1} \cup {2}

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

A == /\ pc[1] = "A"
     /\ temp_' = x + 1
     /\ x' = temp_'
     /\ pc' = [pc EXCEPT ![1] = "Done"]
     /\ temp' = temp

one == A

CC == /\ pc[2] = "CC"
      /\ temp' = x + 1
      /\ x' = temp'
      /\ pc' = [pc EXCEPT ![2] = "Done"]
      /\ temp_' = temp_

two == CC

(* 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
```

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

Exercice 5 *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;

*)
\* BEGIN TRANSLATION
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 -> "D"]

A == /\ pc[1] = "A"
     /\ x' = x + 1
     /\ pc' = [pc EXCEPT ![1] = "B"]

B == /\ pc[1] = "B"
     /\ x = 1
     /\ pc' = [pc EXCEPT ![1] = "C"]
     /\ x' = x

C == /\ pc[1] = "C"
     /\ PrintT(<<"x=",x>>)
     /\ pc' = [pc EXCEPT ![1] = "Done"]
     /\ x' = x

one == A \/ B \/ C

D == /\ pc[2] = "D"
     /\ x = 1
     /\ pc' = [pc EXCEPT ![2] = "E"]
     /\ x' = x

E == /\ pc[2] = "E"
     /\ Assert(x = 1, "Failure of assertion at line 22, column 5.")
     /\ pc' = [pc EXCEPT ![2] = "F"]
     /\ x' = x

F == /\ pc[2] = "F"
     /\ x' = x -2
     /\ pc' = [pc EXCEPT ![2] = "Done"]

two == D \/ E \/ F

(* 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

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

====
```

Exercice 6 *pluscalex1.tla*
Ecrire un programme PlusCal qui traduit le protocole suivant : S envoie une valeur à R

Exercice 7 *pluscalex2.tla*
Ecrire un programme PlusCal qui calcule la fonction factorielle de la façon suivante :

- *Un processus calcule* $1\times2\times3....\times k1$
- *Un processus calcule* $k2\times(k2+1)\times\ldots\times N$
- *Les processus stoppent quand la condition* $k1 < k2$ *est fausse*

Exercice 8 *pluscalex3.tla*
Ecrire un programme PlusCal qui calcule la fonction L^K *la façon suivante :*

- *Un processus calcule* $L\times\ldots\times L$ ***k1** fois.*
- *Un processus calcule* $L\times\ldots\times L$ ***k2** fois.*
- *Les processus stoppent quand la condition* $k1+K2 < L$ *est fausse*