

Cours Algorithmique des systèmes parallèles et distribués
Exercices

Série : PlusCal pour la programmation répartie ou concurrente (II)
par Dominique Méry
11 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 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*.

Listing 1 – pluscaltut10.tla

```
----- MODULE pluscaltut10 -----
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 1 – Programme