

Cours Algorithmique des systèmes parallèles et distribués
Exercices

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