# Cours - Systèmes de Transition

**Louis Thevenet**

## Table des matières

1. Mise en pratique : La factorielle ........ 2
2. Homme-Loup-Mouton-Chou ........ 2
3. Problème Lecteurs/Rédacteurs ........ 2
   3.1. Preuve axiomatique de ExclusionLR ........ 2
   3.2. Raffinement ........ 2
4. L'algorithme de Peterson ........ 2
5. Token Ring ........ 2
   5.1. Contre-exemple 1 ........ 2
   5.2. Contre-exemple 2 ........ 2
   5.3. Raffinage : Tableau de jetons ........ 2
   5.4. Raffinage : Modéliser les canaux de communication ........ 2

# 1. Mise en pratique : La factorielle

```
-------- MODULE Fact0 -------

EXTENDS Naturals
CONSTANT N
VARIABLE res

Init == res = Fact[N]
Next == UNCHANGED res (*ou FALSE*)
Spec == Init \land [Next]_res
==============
```

Liste 1. – 0 transition

```
-------- MODULE Fact1 -------

EXTENDS Naturals
CONSTANT N
ASSUME N \in Nat
VARIABLES res, i

Init ==
    /\ res = 1
    /\ i = 1

Mult ==
    /\ i <= N
    /\ res' = res * i
    /\ i' = i + 1

Next == Mult

Spec == Init \land [Next]_{res,i}
==============
```

Liste 2. – Avec transitions

```
-------- MODULE Fact1 -------

EXTENDS Naturals
CONSTANT N
ASSUME N \in Nat
VARIABLES res, factors

Init ==
    /\ res = 1
    /\ factors = 1..N

Mult(i) ==
    /\ res' = res * i
    /\ factors' = factors \ {i}


Next == \E i \in factors : Mult (i)

Spec == Init \land [Next]_{res,factors}
==============
```

Liste 3. – Sans ordre particulier

```
-------- MODULE Fact1 -------

EXTENDS Naturals
CONSTANT N
ASSUME N \in Nat
VARIABLES res, factors

Init ==
    /\ res = 1
    /\ factors = 1..N

Mult(I) ==
    /\ res' = (*on multiplie les éléments de I à res*)
    /\ factors = 1..N

Next == \E I \in SUBSET factors : Mult (i)
Spec == Init \land [Next]_{res,factors}
==============
```

Liste 4. – Sans ordre particulier

# 2. Homme-Loup-Mouton-Chou

On doit les faire passer d'une rive à l'autre d'une rivière.

- Il faut un homme pour ramer
- Sans la surveillance de l'homme
  - le mouton mange le chou
  - le loup mange le mouton

```
-------- MODULE hlmc -------

    VARIABLES h, m, c, l
    RIVES == {"G", "D"}

    Inv(r) ==
        IF r = "G"
            THEN "D"
            ELSE "G"

    TypeInvariant == {h, l, m,c} \subseteq RIVES

    Init ==
        /\ h = "G"
        /\ l = "G"
        /\ m = "G"
        /\ c = "G"
        (*/\ PasMiam*)

    PasMiam ==
        /\ (l = m => h = m)
        /\ (c = m => h = m)

    MoveH ==
        /\ h' = Inv(h)
        /\ UNCHANGED <<l, m, c>>
        /\ PasMiam'

    MoveHL ==
        /\ h' = Inv(h)
        /\ l' = Inv(l)
        /\ h = l
        /\ UNCHANGED << m, c >>
        /\ PasMiam'

    MoveHM ==
        /\ h' = Inv(h)
        /\ m' = Inv(m)
        /\ h = m
        /\ UNCHANGED << l, c >>
        /\ PasMiam'

    MoveHC ==
        /\ h' = Inv(h)
        /\ c' = Inv(c)
        /\ h = c
        /\ UNCHANGED << l, m >>
        /\ PasMiam'

    Next ==
        \/ MoveH
        \/ MoveHL
        \/ MoveHM
        \/ MoveHC

    Spec ==
        /\ Init
        /\ [Next]_<<h,l,m,c>>

    But == [](~ {h,l,m,c} = {"D"})
==================================================
```

Liste 5. – Sans ordre particulier

# 3. Problème Lecteurs/Rédacteurs

```
MODULE LR0
EXTENDS Naturals
VARIABLES nl, nr

TypeInvariant ==
    /\ nl \in Nat
    /\ nr \in 0..1

Initial ==
    /\ nl = 0
    /\ nr = 0

EntrerL ==
    /\ nr = 0
    /\ nl' = nl+1
    /\ UNCHANGED <<nr>>

SortirL ==
    /\ nl > 0
    /\ nl' = nl -1
    /\ UNCHANGED <<nr>>

EntrerR ==
    /\ nl = 0
    /\ nr = 0
    /\ UNCHANGED <<nl>>
    /\ nr' = 1

SortirR ==
    /\ nr = 1
    /\ UNCHANGED <<nl>>
    /\ nr' = 0

Next ==
    \/ EntrerL
    \/ SortirL
    \/ EntrerR
    \/ SortirR

Spec ==
    /\ Initial
    /\ [Next]_{nl, nr}
    /\ WF_{nl, nr}(SortirL)
    /\ WF_{nl, nr}(SortirR)

ExclusionLR ==
    [](nl = 0 /\ nr = 0)

(*EclusionR ==
    [](nr \in 0..1)
    (* déjà dans invariant de type*)
*)
```

Liste 6. – Lecteurs/Rédacteurs 0

## 3.1. Preuve axiomatique de ExclusionLR

- A l'état initial

$$\text{Initial} \Rightarrow \text{nl} = 0 \vee \text{nr} = 0 \vee$$

- A chaque transition

$$(\text{nl} = 0 \vee \text{nr} = 0) \wedge [\text{Next}]_{\text{nl, nr}} \stackrel{?}{\Rightarrow} \text{nl}' = 0 \vee \text{nr}' = 0$$

  - on étudie chaque transition séparément
    - bégaiement

$$(\text{nl} = 0 \vee \text{nr} = 0) \wedge \text{nl}' = \text{nl} \wedge \text{nr}' = \text{nr} \Rightarrow \text{nl}' = 0 \vee \text{nr}' = 0$$

    - EntrerL ✓

$$(\text{nl}{=}0 \vee \text{nr} = 0) \wedge \text{nr} = 0 \wedge \text{nl}' = \text{nl} + 1 \wedge \text{nr}' = \text{nr} + 1 \Rightarrow \text{nl}' = 0 \vee \text{nr}' = 0$$

    - SortirL ✓

$$(\text{nl}{=}0 \vee \text{nr} = 0) \wedge \text{nl} > 0 \wedge \text{nl}' = \text{nl} - 1 \wedge \text{nr}' = \text{nr} + 1 \Rightarrow \text{nl}' = 0 \vee \text{nr}' = 0$$

    - EntrerR ✓
    - SortirR ✓

## 3.2. Raffinement

```
MODULE LR1
EXTENDS Naturals
VARIABLES nl, nr, ndemr (*nombre demande rédacteurs*)


TypeInvariant ==
    /\ nl \in Nat
    /\ nr \in 0..1
    /\ ndemr \in Nat

Initial ==
    /\ nl = 0
    /\ nr = 0
    /\ ndemr = 0

EntrerL ==
    /\ nr = 0
    /\ nl' = nl+1
    /\ UNCHANGED <<nr>>
    /\ UNCHANEGD <<ndemr>>

SortirL ==
    /\ nl > 0
    /\ nl' = nl -1
    /\ UNCHANGED <<nr>>
    /\ UNCHANEGD <<ndemr>>

EntrerR ==
    /\ nl = 0
    /\ nr = 0
    /\ UNCHANGED <<nl>>
    /\ nr' = 1
    /\ ndemr > 0
    /\ ndemr' = ndemr - 1

SortirR ==
    /\ nr = 1
    /\ UNCHANGED <<nl>>
    /\ nr' = 0
    /\ UNCHANEGD <<ndemr>>

DemanderR ==
    /\ ndemr' = ndemr + 1
    /\ UNCHANGED <<nr, nl>>

Next ==
    \/ EntrerL
    \/ SortirL
    \/ EntrerR
    \/ SortirR
    \/ DemanderR

Spec ==
    /\ Initial
    /\ [Next]_{nl, nr}
    /\ WF_{nl, nr}(SortirL)
    /\ WF_{nl, nr}(SortirR)
    /\ WF_{nl, nr} (EntrerR)

ExclusionLR ==
    [](nl = 0 /\ nr = 0)

(*EclusionR ==
    [](nr \in 0..1)
    (* déjà dans invariant de type*)
*)
```

Liste 7. – Lecteurs/Rédacteurs 1

LR1 est-il un raffinage de LR0 ? Oui car les variables sont les mêmes et les actions sont aussi les mêmes (« raffinage de déterminisme ») ⇒ exclusion est préservée adns LR1

# 4. L'algorithme de Peterson

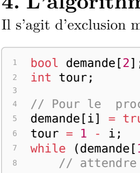
Il s'agit d'exclusion mutuellement entre deux processus.

```
bool demande[2];
int tour;

// Pour le  processus i dans {0, 1}
demande[i] = true;
tour = 1 - i;
while (demande[1-i] && tour == 1-i) {
    // attendre
}
demande[i] = false;
// section critique
```

Liste 8. – Algorithme de Peterson

Le tableau `demande` est « auxiliaire », sa valeur est déterminée par l'endroit du programme où on se trouve.

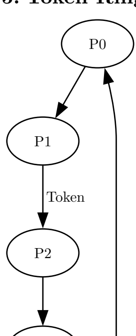
En général, ce type d'algorithme se décompose de la façon suivante :




```
MODULE Peterson
EXTENDS Naturals, FiniteSets


VARIABLES demande, tour, etat
TypeInvariant ==
    /\ demande \in [0..1 -> BOOLEAN]
    /\ tour \in 0..1
    /\ etat \in {"T", "H", "E"}

Initial ==
    /\ demande = [i \in 0..1 |-> FALSE]
    /\ tour = 0
    /\ etat = [ i \in 0..1 |-> "T"]

Demander(i) ==
    /\ etat[i] = "T"
    /\ etat' = [etat EXCEPT ![i] = "H"]
    /\ demande' = [demande EXCEPT ![i] = TRUE]
    /\ tour' = 1 - i

Sortir(i) ==
    /\ etat[i] = "E"
    /\ etat' = [etat EXCEPT ![i] = "T"]
    /\ demande' = [demande EXCEPT ![i] = FALSE]
    /\ tour' = tour

Entrer(i) ==
    /\ etat[i] = "H"
    /\ etat' = [etat EXCEPT ![i] = "E"]
    /\ (\neg demande[1-i]  \/ tour = i)

Next == \E i \in 0..1 : Entrer(i) \/ Demander(i) \/ Sortir(i)

Spec ==
    /\ Initial
    /\ [Next]_{demande, tour, etat}
    /\ \forall i \in 0..1 : WF_{demande, tour, etat}(Sortir(i))
                            /\ WF_{demande, tour, etat}(Demander(i))

ExclusionMutuelle ==
    [](~ (etat[0] = "E" /\ etat[1] = "E"))
    (* ou *)
    [](\forall i, j \in 0..1 : (etat[i] = "E" /\ etat[j] = "E" => i = j))
    (* ou *)
    [](Cardinality({i \in 0..1 : etat[i] = "E"}) <= 1)

AbsenceDeFamine ==
    \forall i \in 0..1 : etat[i] = "H" ~> etat[i] = "E"

DemandeMaintenue ==
    \forall i \in 0..1 : [](etat[i] = "H" => etat[i]' \in {"H", "E"})

AbsenceDeDeadlock ==
    {i \in {0..1} | etat[i] = "H"} != \emptyset ~> {i \in 0..1 | etat[i] = "E"} !
= \emptyset
    (* ou *)
    \forall i \in 0..1 : etat[i] = "H" => \E j \in 0..1 : etat[j] = "E"
```

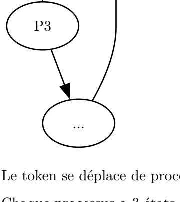
Liste 9. – Algorithme de Peterson en TLA+

# 5. Token Ring




Le token se déplace de processus en processus, c'est un problème de type exclusion mutuelle.

Chaque processus a 3 états : `Thinking`, `Hungry` et `Eating`




```
MODULE TokenRing0
EXTENDS Naturals, FiniteSets
CONSTANT N
ASSUME N \in Nat

ETAT == {"T", "H", "E"}

VARIABLES == etat, token

TypeInvariant ==
    /\ etat \in [0..N-1 -> ETAT]
    /\ token \in 0..N-1

Initial ==
    /\ etat = [i \in 0..N-1 |-> "T"]
    /\ token \in 0..N-1 (* ou juste 0 *)


Demander(i) ==
    /\ etat[i] = "T"
    /\ etat' = [etat EXCEPT ![i] = "H"]
    /\ UNCHANGED token

Sortir(i) ==
    /\ etat[i] = "E"
    /\ etat' = [etat EXCEPT ![i] = "T"]
    /\ UNCHANGED token

Entrer(i) ==
    /\ etat[i] = "H"
    /\ etat' = [etat = etat EXCEPT ![i] = "E"]
    /\ token = i
    /\ UNCHANGED token (* important *)

Bouger(i) ==
    /\ token = i
    /\ token' = (i+1) % N
    /\ UNCHANGED etat
    /\ etat[i] != "E"


Next ==
    \E i \in 0..N-1:
        \/ Demander(i)
        \/ Entrer(i)
        \/ Sortir(i)
        \/ Bouger(i)

Spec ==
    /\ Initial
    /\ [Next]_{etat, token}
    /\ \forall i \in 0..N-1:
        /\ WF_{etat, token}(Sortir(i))
        /\ WF_{etat, token}(Entrer(i))

ExclusionMutuelle == [(Cardinality({i \in 0..N-1: etat[i] = "E"}))]
AbsenceDeFamine == \forall i \in 0..N-1: etat[i] = "H" ~> etat[i] = "E"
Lien == \forall i \in 0..N-1: [etat[i] = "E" => token = i]
```

Liste 10. – Token Ring 0

## 5.1. Contre-exemple 1

Famine des processus autre que 0.

$$\text{token} = 0$$
$$(\text{Demander}(0) \to \text{Entrer}(0) \to \text{Sortir}(0))^{\omega}$$

On décide de pousser le token dehors.

```
Sortir(i) ==
    /\ etat[i] = "E"
    /\ etat' = [etat EXCEPT ![i] = "T"]
    (*/\ UNCHANGED token (*on vire ça*)*)
```

Liste 11. – Nouveau `Sortir(i)`

## 5.2. Contre-exemple 2

Le jeton bouge très vite et les processus ne peuvent pas accéder à leur ressource.

$$\text{jeton} = 0$$
$$(\text{Demander}(0) \to \text{Bouger}(0) \to \text{Bouger}(1) \to ... \to \text{Bouger}(N-1))^{\omega}$$

On ajoute une condition à `Bouger`

```
Bouger(i) ==
    /\ token = i
    /\ token' = (i+1) % N
    /\ UNCHANGED etat
    /\ etat[i] = "T"
    (*/\ etat[i] != "E" (*on vire ça*) *)
```

Liste 12. – Nouveau `Bouger(i)`

Au lieu de modifier `Sortir` et `Bouger`, on aurait pu ajouter de l'équité plus forte que faible pour forcer l'execution quand c'est nécessaire.

## 5.3. Raffinage : Tableau de jetons

```
    MODULE TokenRing0
    EXTENDS Naturals, FiniteSets
    CONSTANT N
    ASSUME N \in Nat

    ETAT == {"T", "H", "E"}

    VARIABLES == etat, token

    TypeInvariant ==
        /\ etat \in [0..N-1 -> ETAT]
        /\ token \in [0..N-1 -> BOOLEAN]

    Initial ==
        /\ etat = [i \in 0..N-1 |-> "T"]
        /\ \E i \in 0..N-1: token = [j \in 0..N-1 |-> i=j]


    Demander(i) ==
        /\ etat[i] = "T"
        /\ etat' = [etat EXCEPT ![i] = "H"]
        /\ UNCHANGED token

    Sortir(i) ==
        /\ etat[i] = "E"
        /\ etat' = [etat EXCEPT ![i] = "T"]
        /\ UNCHANGED token

    Entrer(i) ==
        /\ etat[i] = "H"
        /\ etat' = [etat = etat EXCEPT ![i] = "E"]
        /\ token[i]
        /\ UNCHANGED token (* important *)

    Bouger(i) ==
        /\ token[i]
        /\ token' = [token EXCEPT ![i]=FALSE, ![(i+1)%N] = TRUE]
        /\ UNCHANGED etat
        /\ etat[i] != "E"


    Next ==
        \E i \in 0..N-1:
            \/ Demander(i)
            \/ Entrer(i)
            \/ Sortir(i)
            \/ Bouger(i)

    Spec ==
        /\ Initial
        /\ [Next]_{etat, token}
        /\ \forall i \in 0..N-1:
            /\ WF_{etat, token}(Sortir(i))
            /\ WF_{etat, token}(Entrer(i))

    ExclusionMutuelle == [(Cardinality({i \in 0..N-1: etat[i] = "E"}))]
    AbsenceDeFamine == \forall i \in 0..N-1: etat[i] = "H" ~> etat[i] = "E"
    Lien == \forall i \in 0..N-1: [etat[i] = "E" => token[i]]
    TokenUnique == [\forall i, j \in 0..N-1: token[i] /\ token[j] => i = j
]
```

Liste 13. – Token Ring 1

Pour prouver le raffinage entre `TokenRing0` et `TokenRing1`, il faut au moins exhiber une fonction de mapping entre les variables de `TokenRing1` et sur celles de `TokenRing0`.

$$\text{etat1} = \text{etat0}$$
$$\text{token1} = \text{CHOOSE } iin0..N-1 : \text{token1}[i]$$

## 5.4. Raffinage : Modéliser les canaux de communication

On l'a pas fait