Module TLA avec 2 variables x et y et deux actions:

- **1 qui transfère x dans y**
- **1 qui transfère y + 1 dans x**

MODULE xyplus1
EXTENDS Naturals
VARIABLES x, y

Init [=] x = 0 /\ y = 0
Act1 [=] /\ y’ = x
 /\ UNCHANGED x
Act2 [=] /\ x’ = y+1
 /\ UNCHANGED y
Next [=] Act1 \/ Act2
Spec [=] Init /\ [] [Next] <x,y>
 /\ WF(Act1)
 /\ WF(Act2)

Quelle propriétés spec vérifie-t-elle ?

Type OK [=] [] (x \in Nat /\ y \in Nat)
XSuperieur [=] (x >= y)
Ecart [=] [] (0 <= x - y <= 1)
CroissantX [=] [] (x’ >= x)
XSupY [=] <> (x >= y)
CroissantXBis [=] pourtout k \in Nat, [] (x >= k => [] (x >= k))
XCroitVivement [=] pourtout k \in Nat, <> (x >= k)
ToutEntierAtteint [=] pourtout k \in Nat, <> (x = k)

Preuve de [] (0 <= x - y <= 1):

Soit I [=] 0 <= x - y <= 1

- Init => I   x = 0 /\ y = 0 => 0 <= x - y <= 1 trivial
- I /\ Next => I’
  I /\ Act1 => I’
  0 <= x - y <= 1 /\ y’ = x /\ x’ = x => 0 <= x’ - y’ <= 1 ok
- I /\ Act2 => I’
  0 <= x - y <= 1 /\ x’ = y + 1 /\ y’ = y => 0 <= x’ - y’ <= 1 ok
- I /\ UNCHANGED (x, y) => I’ ok

Preuve de pourtout k \in Nat, <> (x >= k):

pourtout k, x >= k → x >= k+1

- x >= k → y >= k   Act1 + WF (Act1)
- y >= k → x >= k+1   Act2 + WF(Act2)
- x >= k → x >= k+1   transitivité de →
- x >= 0 → x >= k+1
- true → x >= k+1 par type ok
  = [] <> (x >= k +1) => <> (x>= k+1)

Etude d'un algo d'exclusions mutuelle:
Algorithme de Peterson, 2 processus, variables partagées: demande [0..1] et tour

Algo pour P0:

<u>demande d'entrée</u>
demande[0] ← true
tour ← 1
while (demande[1] /\ tour = 1)

<u>sortie</u>
// section critique
demande[0] ← false

1) **<u>Modéliser le problème de l'exclusion mutuelle</u>**
a) **<u>Comportement des processus:</u>**

Etats: en section critique, Ei (eating)
hors section critique, Ti (thinking)
demandeur en attente, Hi (hungry)

Propriétés d'un processus en isolation:

- [ ] (Ti (+) Hi (+) Ei)   (+) = ou exclusif
- [ ] (Ti => Ti \/ Hi) donc [] (Ti => Ti' \/ Hi')
- [ ] (Hi => [] Hi)   un processus ne contrôle pas l'entrée mais ili ne retire pas sa demande tant qu'elle n'est aps satisfaite
- [ ] (Ei => Ei \/ Ti)
  Ei → Tiun processus ne peut pas rester définitivement en Ei

b) Propriétés du système (les 2 processus et l'algo d'exclusion mutuelle)

- Exclusion mutuelle:
  ☐ (non(E0 /\ E1))
  pourtout i pourtout j, [] (Ei /\ Ej => i = j)
- Absence de famine:
  pourtout i [] (Hi => <> Ei)
  pourtout i Hi → Ei
- Absence d'interblocage: (plus faible)
  (\E i, Hi) → (\E j, Ej)

Codage de l'algorithme de Peterson:

```
MODULE Peterson
CONSTANT N
ASSUME N = 2
VARIABLES
        etats, (* pour chaque processus \in {“T”, “H”, “E”}*)
        tour (* \in 0..1*)
Init [=] /\ etat = [i \in 0..N-1 |→ “T”]
        /\ tour \in 0..N-1 (* ou {0..1} si N=2*)
demander(i) [=] etat[i] = “T”
                /\ etat’ = [etat EXCEPT ![i] = “H”]
                /\ tour’ = 1 - i
entier(i) [=] /\ etat[i] = “H”
        /\ (etat[i] = “T” \/ tour = i)
        /\ etat’ = [etat EXCEPT ![i] = “E”]
        /\ UNCHANGED tour
sortir(i) [=] /\ etat[i] = “E”
        /\ etat’ = [etat EXCEPT ![i] = “T”]
        /\ UNCHANGED tour
Next [=] \E i \in 0..N-1, \/ demander[i]
                        \/ entrer[i]
                        \/ sortir[i]
Fairness [=] pourtout i \in 0..N-1, /\ WF (sortir(i))
                                   /\ WF (entrer(i))
Spec [=] Init /\ [] [Next] /\ Fairness
```

Exclusion Mutuelle en réparti
Réalisation avec un jeton circulant:
Le jeton visite les sites
Le site qui a le jeton peut entrer en mutex

MODULE jeton
CONSTANT N   // Nb de sites
Thinking [=] 1
Hungry [=] 2
Eating [=] 3
VARIABLE
etat // \in [0..N-1 |→ {Thinking, Hungry, Eating}]
jeton // \in 0..N-1
Init [=] etat = [i \in 0..N-1 |→ Thinking] /\ jeton \in 0..N-1
demander(i) [=] /\ etat[i] = Thinking
                    /\ etat' = [etat EXCEPT ![i] = Hungry]
                    /\ UNCHANGED jeton
entrer(i) [=] /\ etat[i] = Hungry
          /\ pourtout j \in 0..N-1, etat[j] ≠ Eating
          /\ etat' = [etat EXCEPT ![i] = Eating
          /\ jeton = i
          /\ UNCHANGED jeton
sortir(i) [=] /\ etat'[i] = Thinking
          /\ etat' = [etat EXCEPT ![i] = Thinking]
bouger(i) [=] /\ jeton = i
                    /\ etat[i] ≠ Eating
                    /\ jeton' = (i+1)%Eating
                    /\ UNCHANGED etat
Equite [=] pourtout in \in 0..N-1, SF(entrer(i)) /\n WF(sortie(i))
 // SF = Soft Fairness   WF = weak Fairness
Next [=] pourtout i \in 0..N-1, demander(i) \/ entrer(i) \/ sortir(i) \/ bouger(i)
Spec [=] Init AND [] [Next] /\ Equite

Propriétés attendues:
ExclusionMutuelle [=] [] (pourtouti, j \in 0..N-1, etat[i] = Eating
                                                            /\ etat[j] = Eating
                                                            => i = j)
                              [] (Cardianlity({i \in 0..N-1, etat[i] = Eating}) <= 1)
AbsenceFamine [=] pourtout i \in 0..N-1, etat[i] = Hungry → etat[i] = Eating
AbsenceInterbloquage [=] (\E i, etat[i] = Hungry) → (\E j etat[j] = Eating)
          [] (pourtout i \in 0..N-1, etat[i] = Eating => jeton = 1)
JetonAnneau [=] pourtout i [] (jeton = i => jeton = i OR jeton = (i+1)%N)
JetonVaPartout [=] pourtout i [] <> (jeton = i)
<u>Equité nécessaire pour absence de famine:</u>
pourtout i \in 0..N-1, SF(entrer(i)) /\ SF(bouegr(i)) /\ WF (sortir(i))
<u>Equité nécessaire pour absence d'interbloquage:</u>
pourtout i \in 0..N-1, SF(entrer(i)) /\ WF(bouger(i)) /\ WF(sortir(i))