# Validation des systèmes embarqués
# Model checking

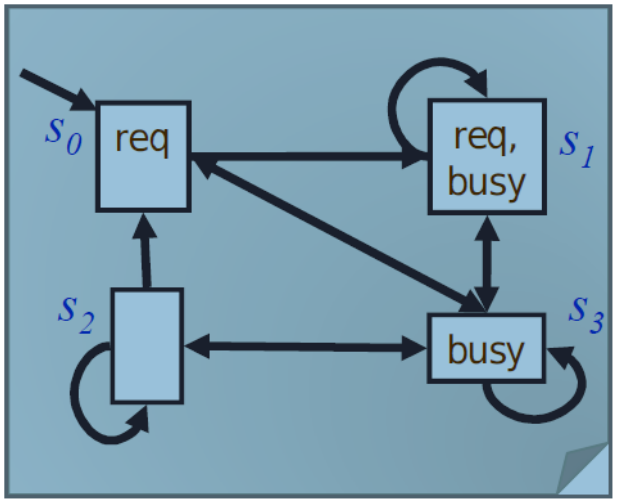
Nga Nguyen

# Cours 2 : Modélisation des systèmes réactifs

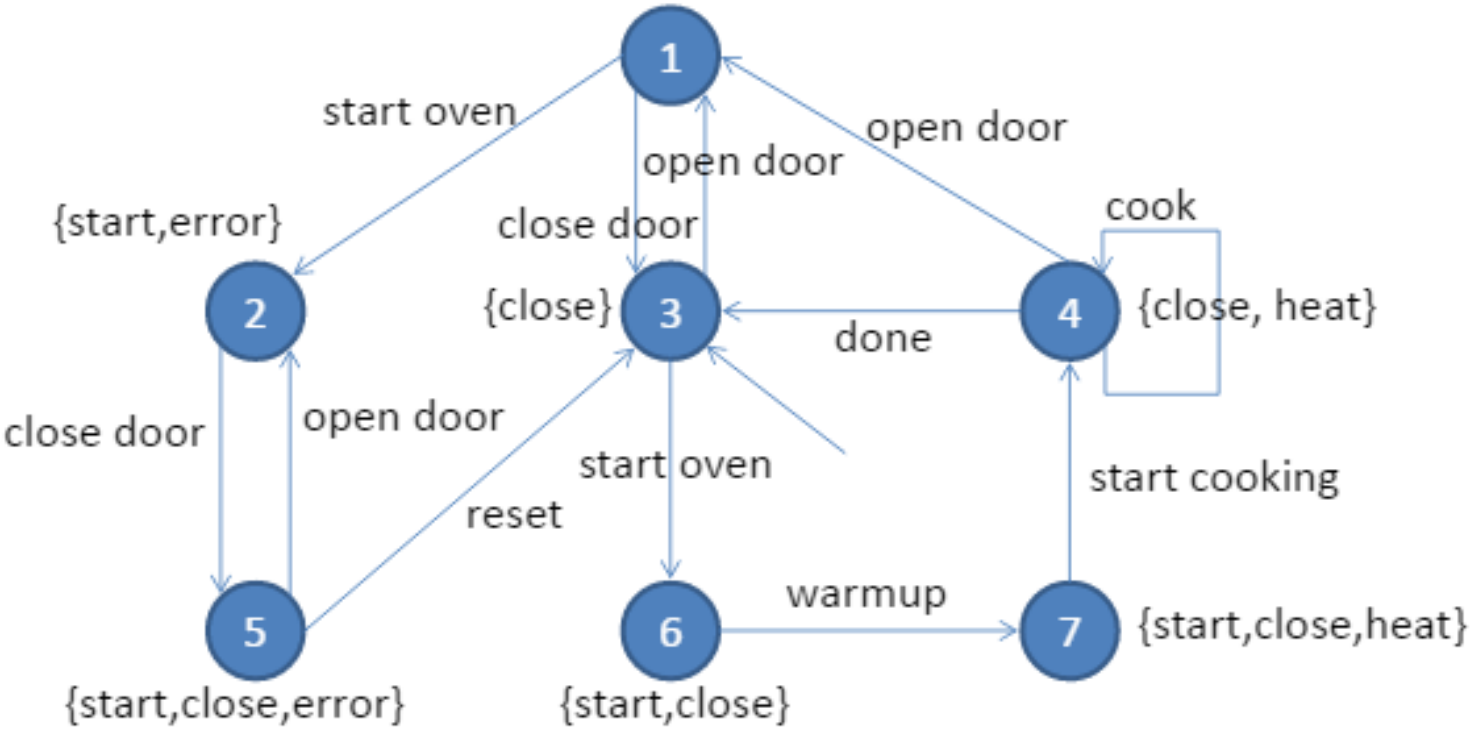
- Structure de Kripke
- Sémantique du LTL et CTL
- Algorithme model checking CTL par marquage

# Structure de Kripke

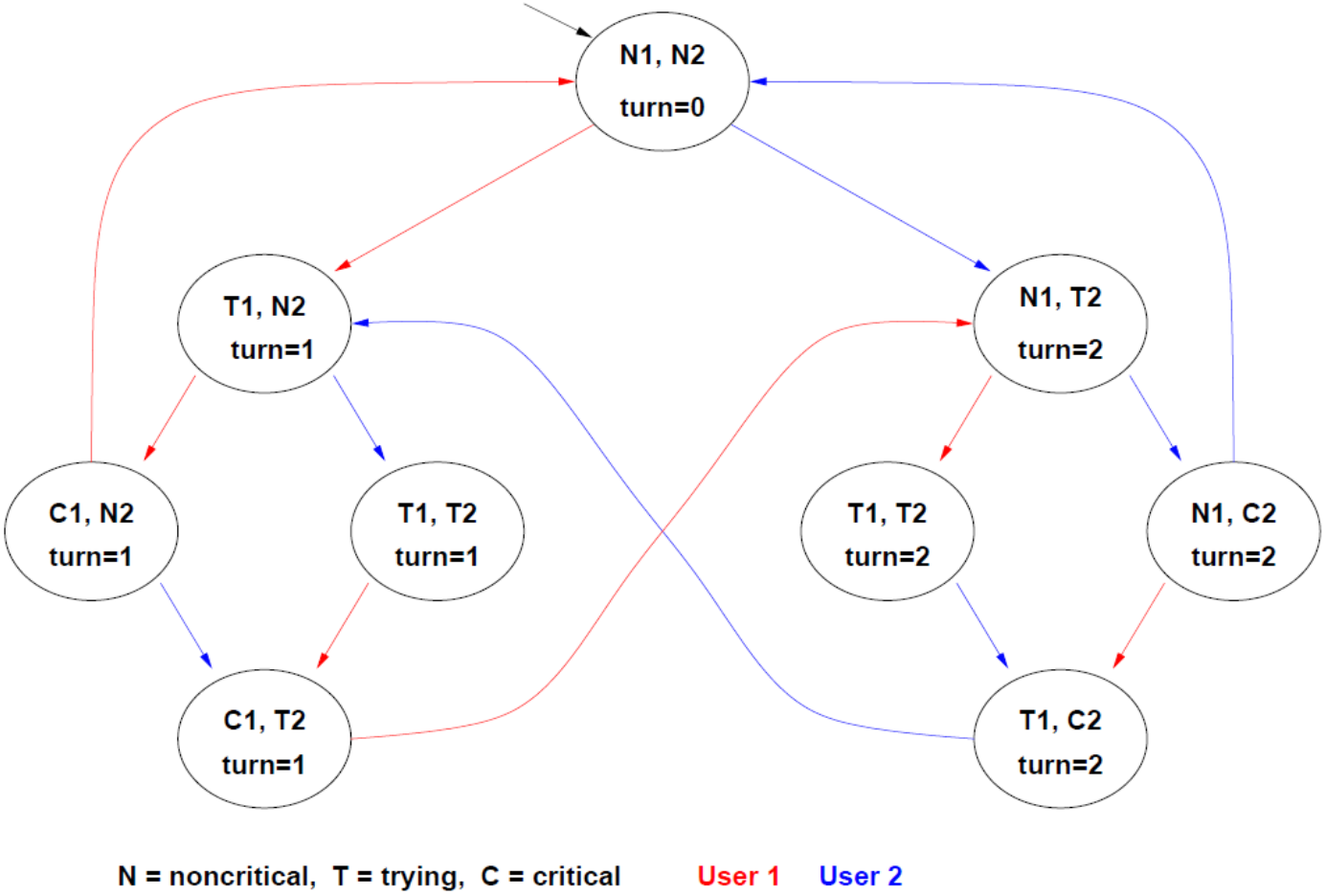
- Une structure de Kripke est un ensemble K = (V, S, $s_0$, I, R):
  - V : ensemble (fini) de propositions atomiques
  - S : ensemble (fini) d'états
  - $s_0 \in S$ : état initial
  - I : S -> $2^V$ : fonction associant chaque état à un ensemble de propositions qui sont vraies dans cet état
  - R : S x S : relation de transition



# Structure de Kripke d'un micro-onde



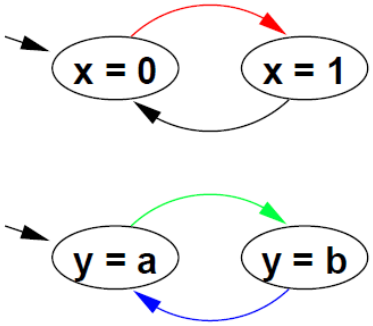
# Structure de Kripke pour l'exclusion mutuelle

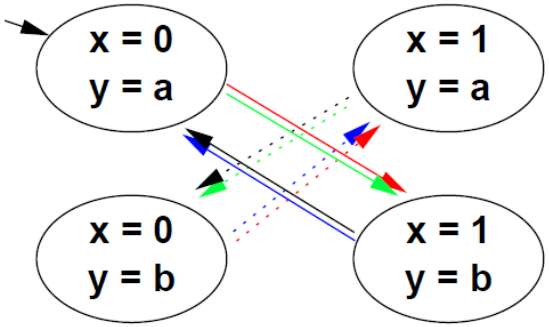



*A. Cimatti, M. Pistore, and M. Roveri, http://nusmv.irst.itc.it/courses*

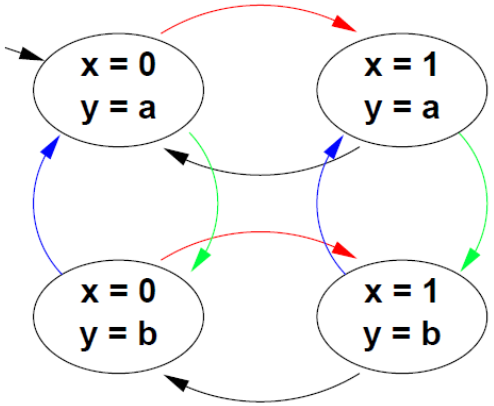
# Systèmes complexes

Chaque composant est représenté par une structure de Kripke. Les composants peuvent être combinés via :




- Composition synchrone :
  - Les composants évoluent en parallèle
  - A chaque instance, chaque composant réalise sa transition
- Composition asynchrone
  - Entrelacement entre les composants
  - A chaque instance, un seul composant est choisi pour réaliser sa transition

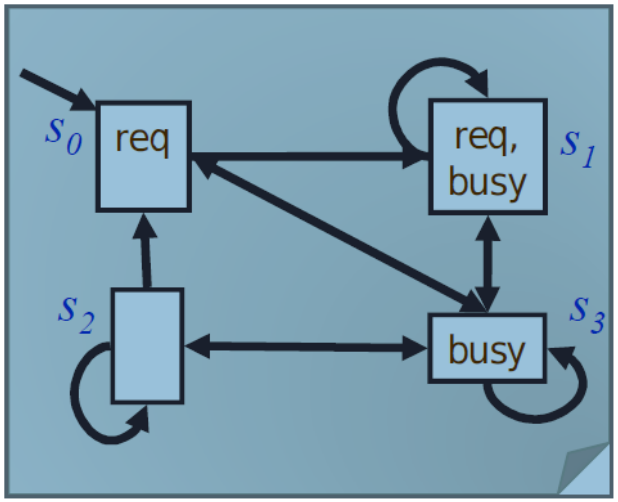



*A. Cimatti, M. Pistore, and M. Roveri, http://nusmv.irst.itc.it/courses*

# Chemin d'exécution

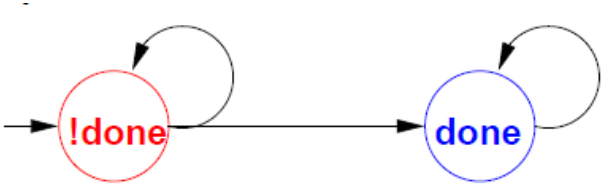
- Un chemin en K est une séquence infinie d'état $\pi = s_0, s_1, \ldots$ tels que $(s_i, s_{i+1}) \in R$ pour tout $i >= 0$
- On note $\pi^i$ le suffixe de $\pi$ à partir de l'état $s_i$

- Un état s est atteignable dans K s'il existe un chemin de l'état initial $s_0$ à s



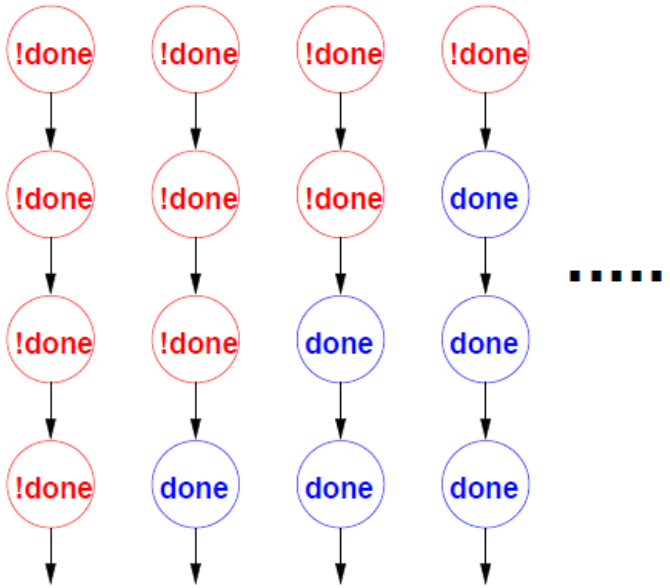
# Chemin de calcul vs Arbre de calcul

Considérer la structure de Kripke suivante :

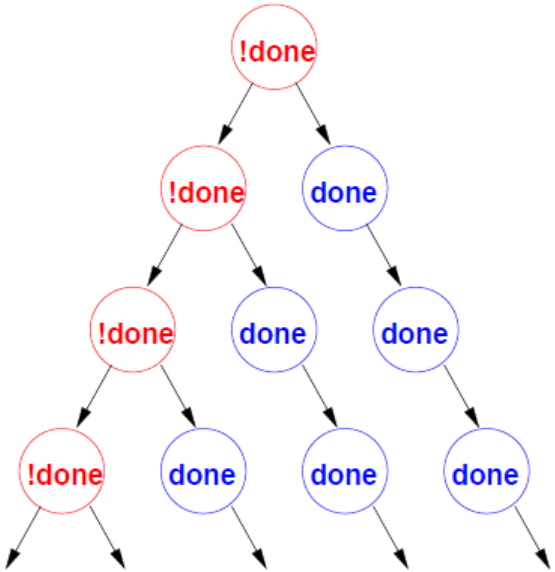



Son exécution peut être vue comme :

- Un ensemble de chemins de calcul infini(s)




- Un arbre de calcul infini

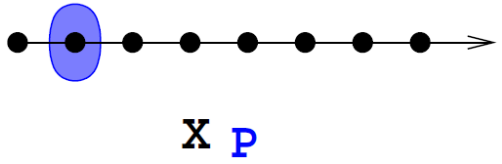



*A. Cimatti, M. Pistore, and M. Roveri, http://nusmv.irst.itc.it/courses*

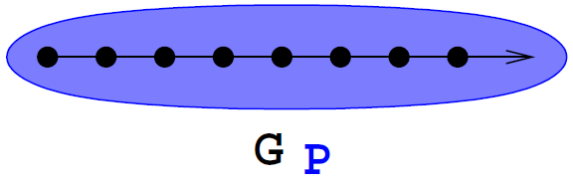
# Sémantique du LTL

Si on note $\pi \models p$ formule p est vraie en $\pi$, alors :

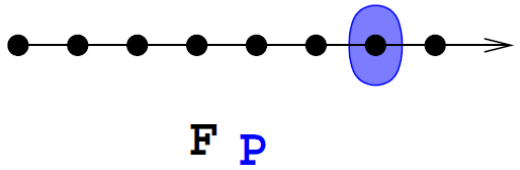
- $\pi \models X\ p$ ssi $\pi^1 \models p$



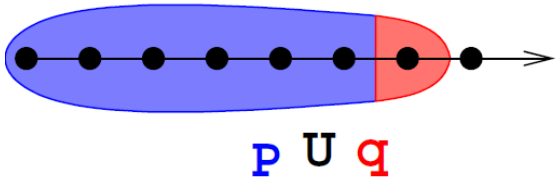

- $\pi \models G\ p$ ssi pour tout i : $\pi^i \models p$




- $\pi \models F\ p$ ssi existe i : $\pi^i \models p$




- $\pi \models p\ U\ q$ ssi existe i : $\pi^i \models q$ et $\pi^j \models p$ pour tout $0 <= j < i$

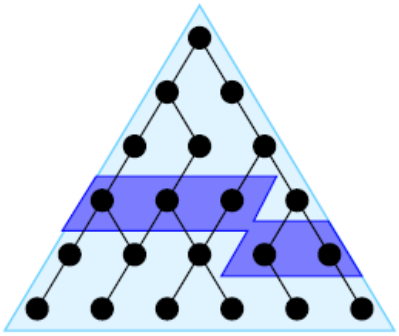


# Sémantique du CTL

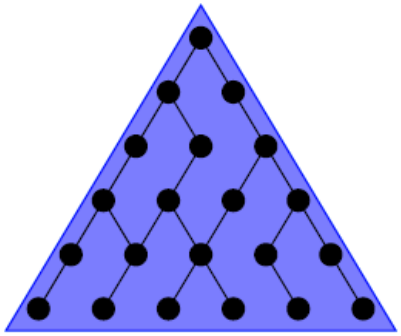
Si on note $\pi \models p$ formule p est vraie en $\pi$, alors :

- AX p — pour tout $\pi$ : $\pi^1 \models p$
- EX p — existe $\pi$ : $\pi^1 \models p$
- AG p — pour tout $\pi$, pour tout i : $\pi^i \models p$
- EG p — existe $\pi$, pour tout i : $\pi^i \models p$
- AF p — pour tout $\pi$, existe i : $\pi^i \models p$
- EF p — existe $\pi$, existe i : $\pi^i \models p$
- A[p U q] — pour tout $\pi$, existe i : $\pi^i \models q$ et $\pi^j \models p$ pour tout $0 <= j < i$
- E[p U q] — existe $\pi$, existe i : $\pi^i \models q$ et $\pi^j \models p$ pour tout $0 <= j < i$

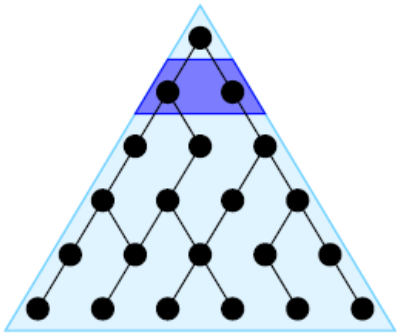
# Arbres de calcul



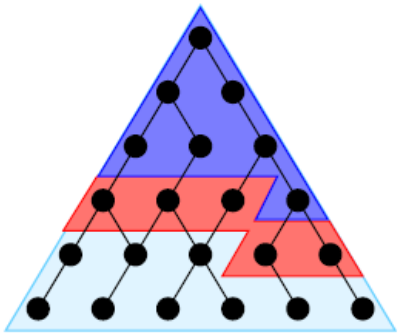
AF p



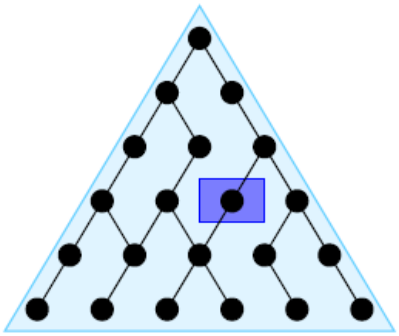
AG p



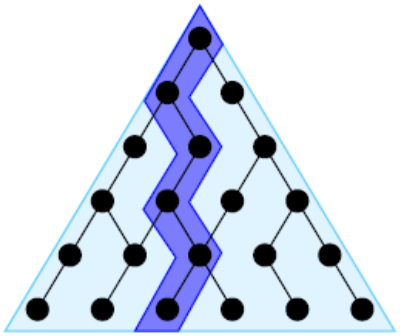
AX p



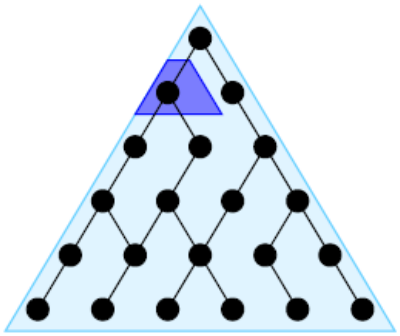
A [ p U q ]



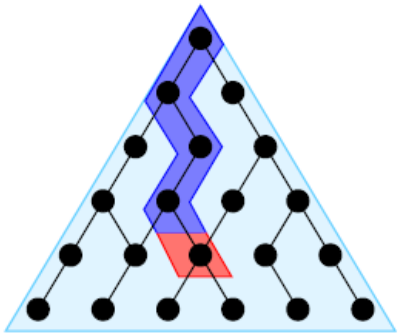
EF p



EG p

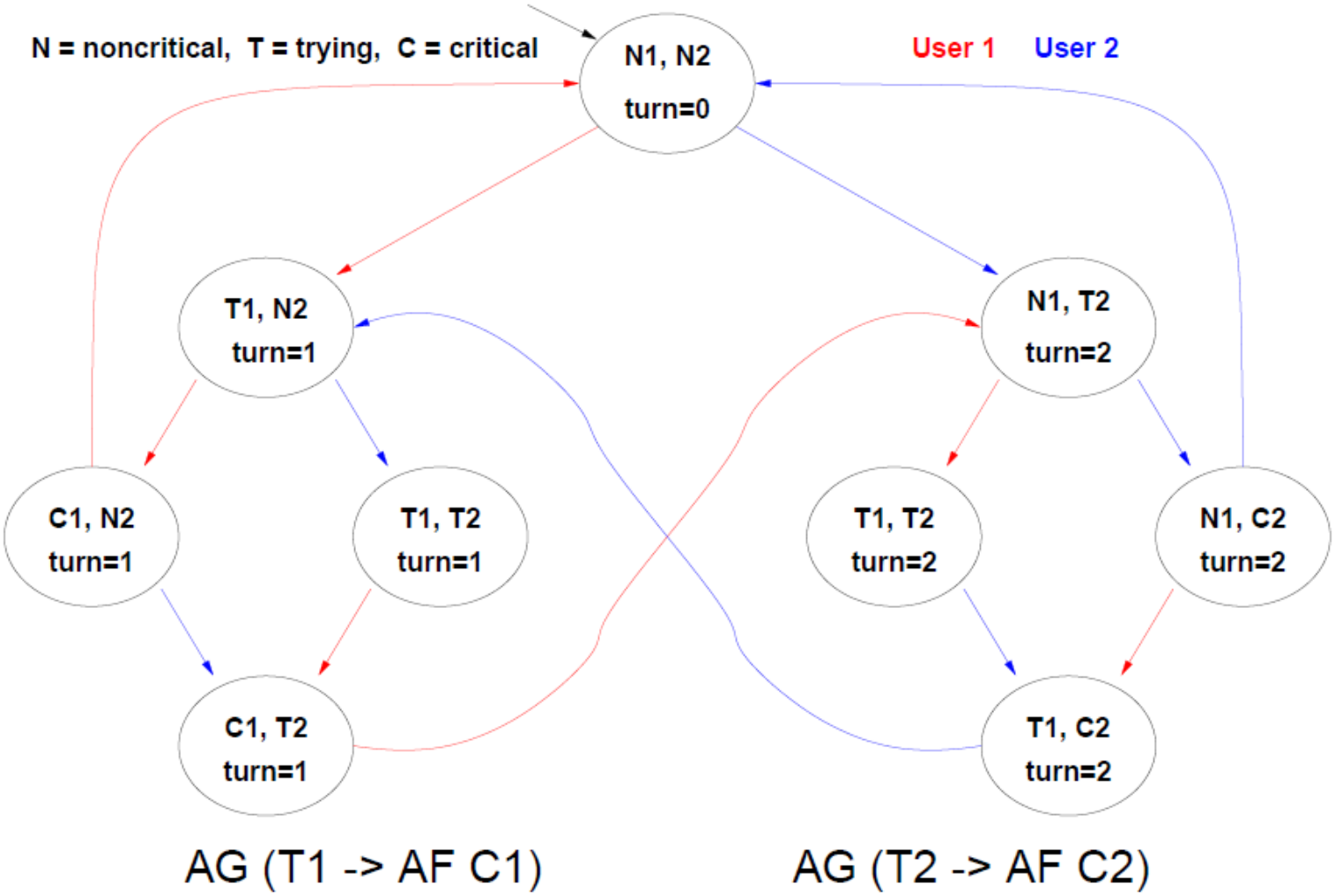


EX p

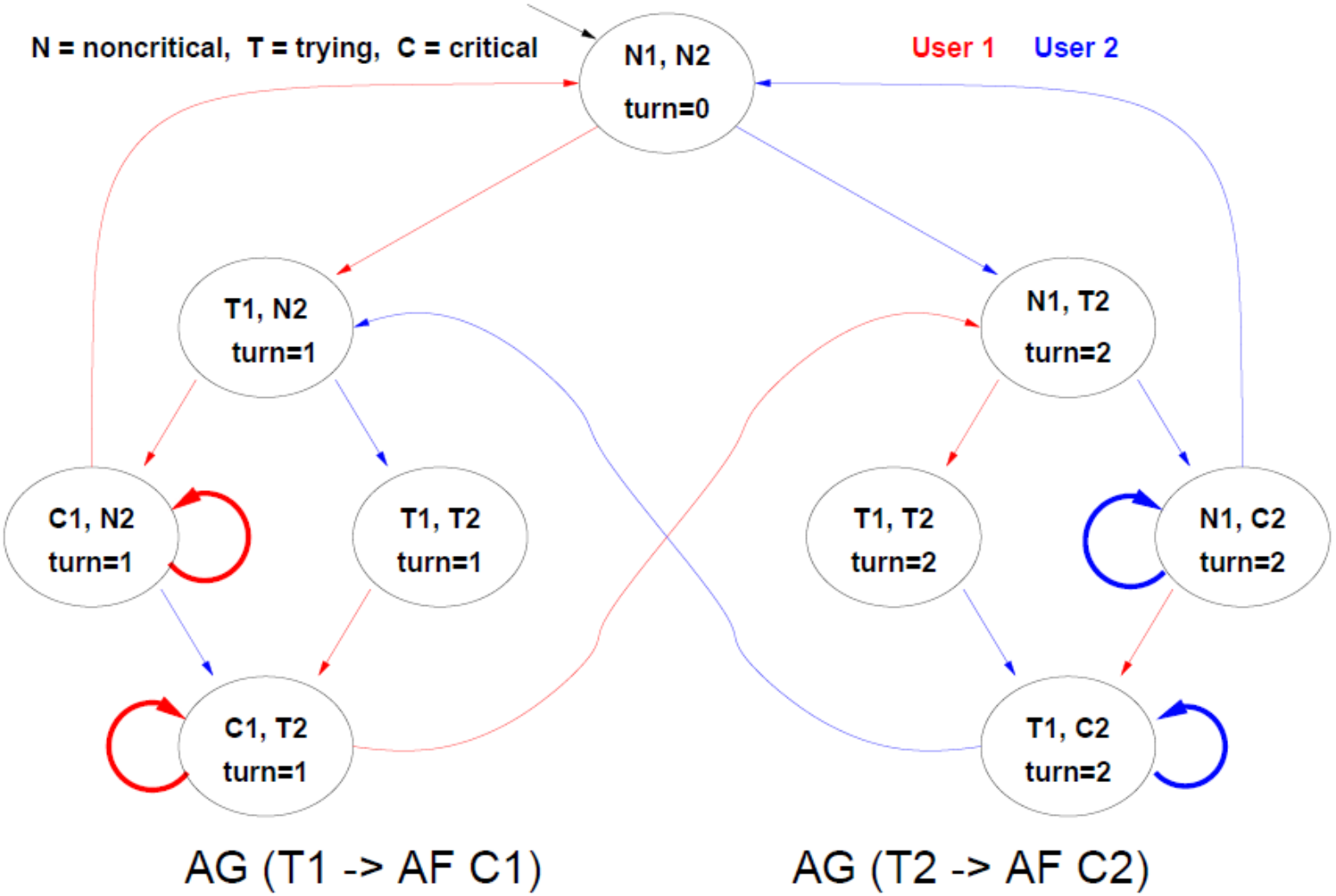


E [ p U q ]

# Hypothèse d'équité (fairness)



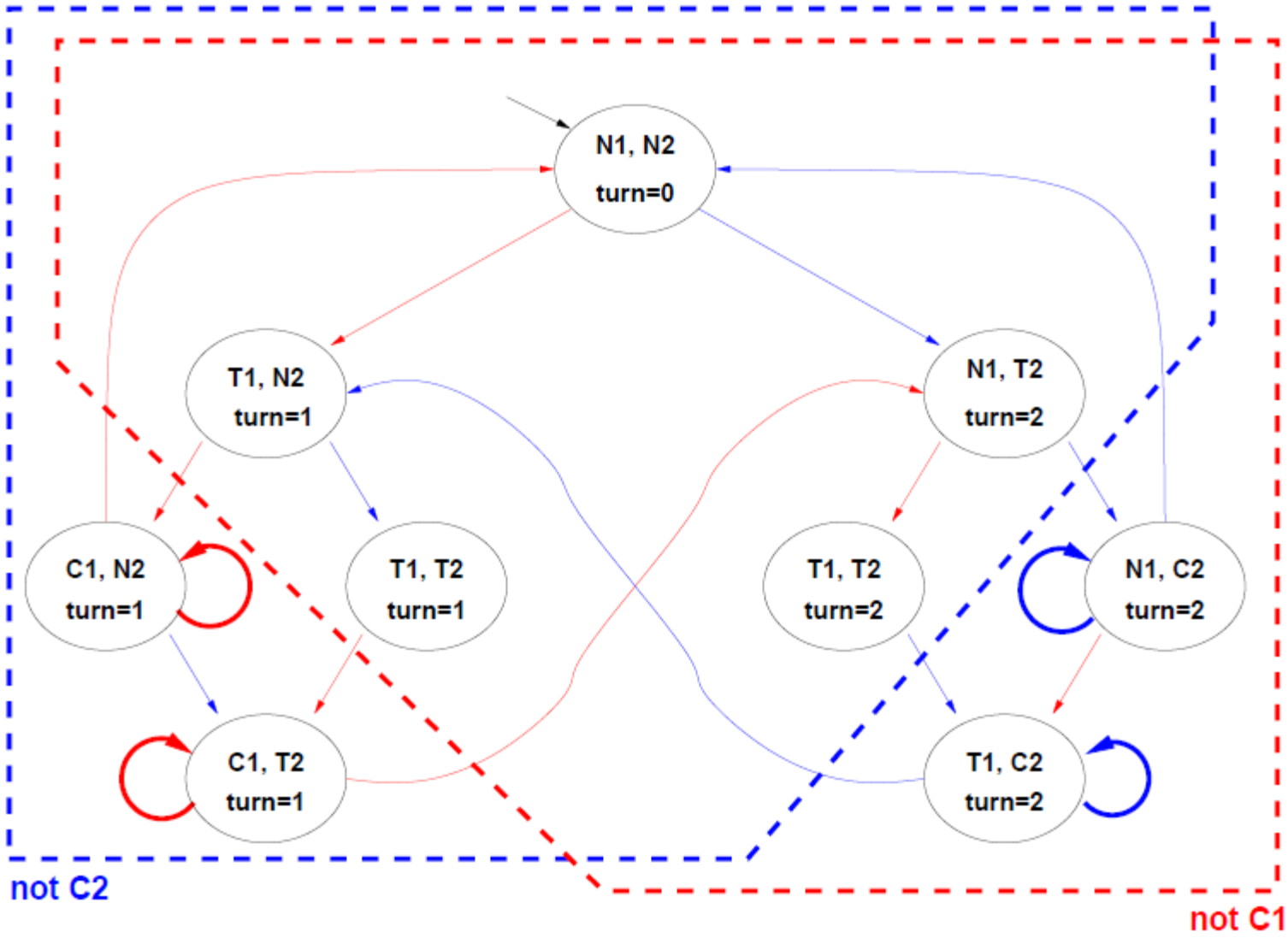
# Hypothèse d'équité (fairness)



# Besoin d'hypothèse d'équité (fairness)

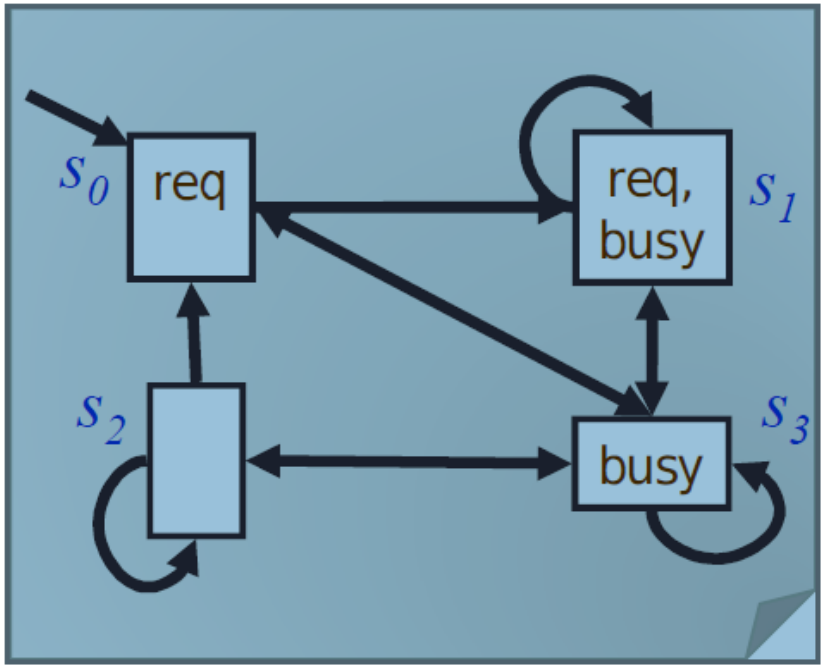
- Des hypothèses d'équité sont utilisées pour éliminer des comportements où une condition ne tient jamais :
  - Quand un processus entre en section critique, il n'en sort jamais
- Un ensemble de contraintes d'équité F = $\{f_1, f_2, \ldots\}$ où $f_i \in V$ :
  - Une exécution « fair » du système est une exécution passant infiniment souvent par chaque $f_i$

# Fairness = {{not C1}, {not C2}}



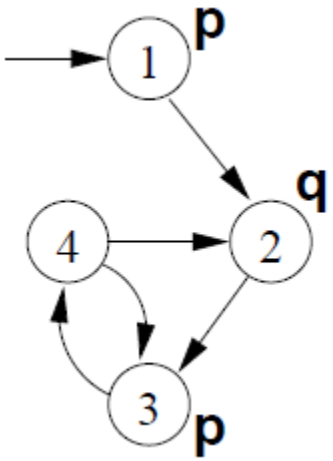
# CTL et Structure de Kripke

- Vrai ou Faux ?
  1. (AX busy) ($s_0$)
  2. (EG busy) ($s_3$)
  3. A(req U busy) ($s_0$)
  4. E(~req U busy) ($s_1$)
  5. AG(req ->AF busy) ($s_0$)
  6. (AX (req v busy)) ($s_3$)



# Algorithme de model checking

1. Le système est représenté comme une machine d'états




2. Les propriétés sont exprimées en logique temporelle :

   LTL : G(p -> Fq)  CTL : AG(p -> AFq)

3. L'algorithme vérifie si toutes les exécutions du modèle satisfont la formule

# Algorithme Model checking CTL par marquage (complexité linéaire)

- Entrée :
  - Structure Kripke K = (V, S, $s_0$, I, R)
  - Formule CTL f
- Principe :
  - Pour chaque sous-formule f’ de f, marquer les états de K qui satisfont f’
  - Procédure récursive utilisant les marquages des sous-formules les plus internes pour marquer une sous-formule plus externe (bottom-up)
  - K satisfait f ssi l’état initial $s_0$ est marqué par f

# Exemple

- AG(p -> AFq)

  q

  AFq

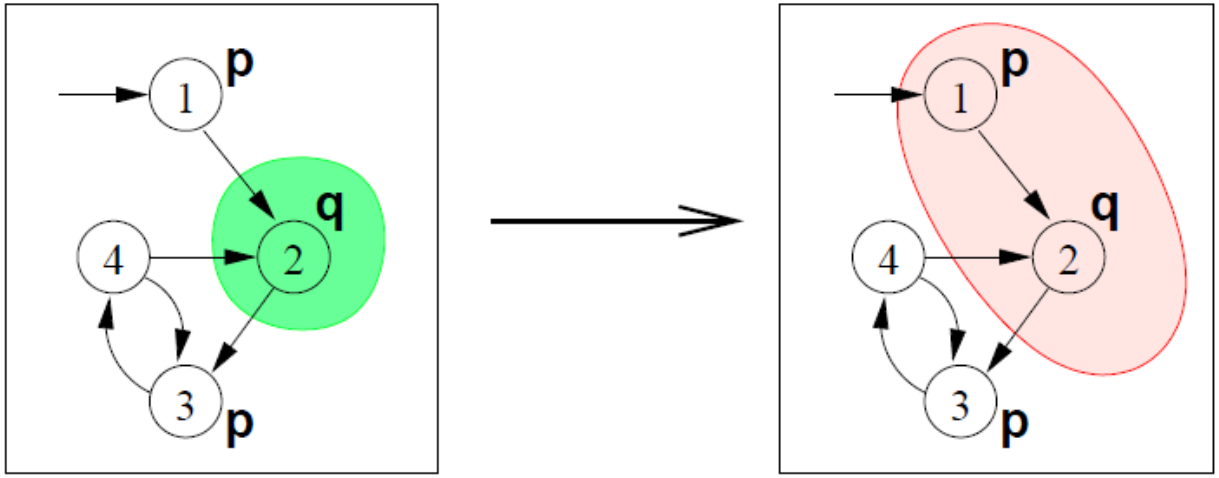
  p

  p -> AFq

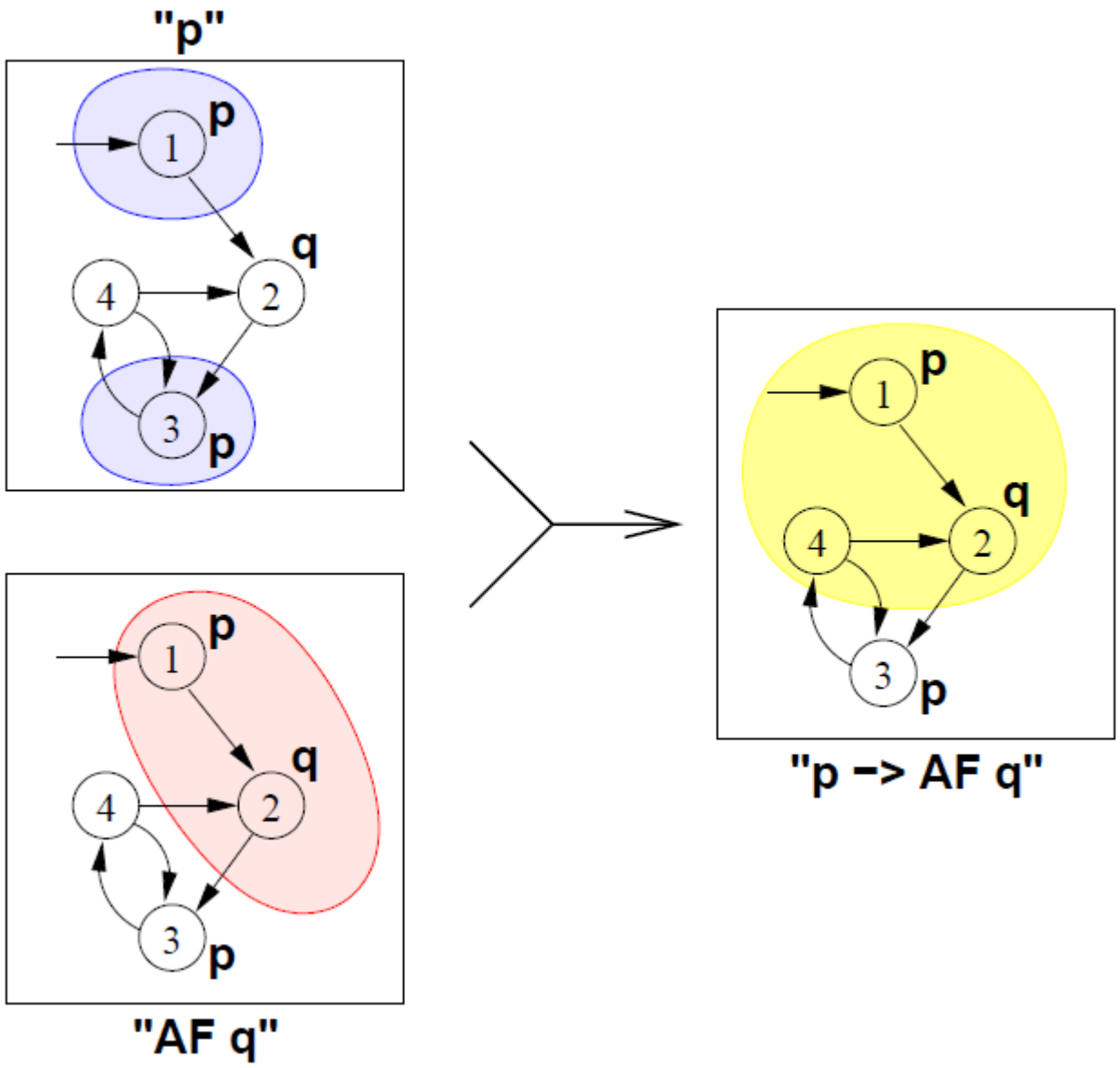
  AG(p -> AFq)

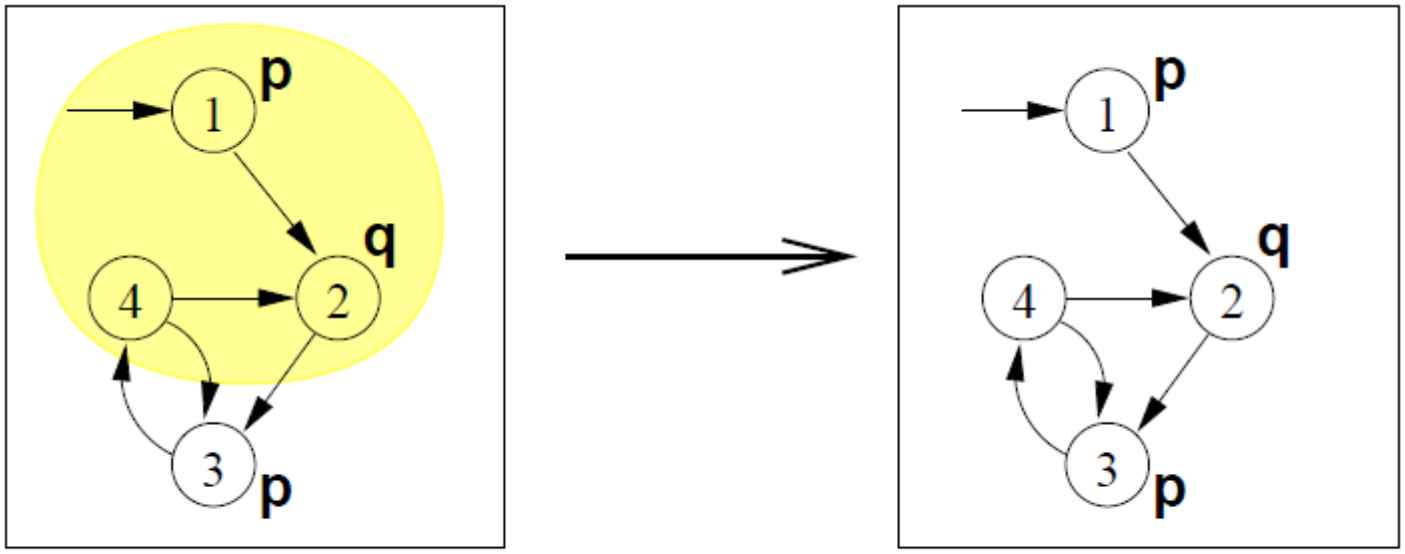
# Exemple

- AG(p -> AFq)




- AFq est l'union de q, AX q, AX AX q, …


"p"
p
1
q
4
2
3
p
"AF q"
p
1
q
4
2
3
p
p
1
q
4
2
3
p
"p -> AF q"




"p –> AF q"

"AG(p –> AF q)"

- L'ensemble d'états où la formule tient est vide !

**Algorithme MARKING(f, K)**

input : formule f normalisée, K = (V, S, $s_0$, I, R)

**Case 1 : f = p**

for all s ∈ S do
    if p ∈ l(s) then s.f := true
    else s.f := false
end for

**Case 2 : f = !f'**

do MARKING(f',K) ;
for all s ∈ S do s.f := not(s.f') end for

**Case 3 : f = f1 ∧ f2**

do MARKING(f1,K) ; MARKING(f2,K) ;
for all s ∈ S do s.f := and(s.f1,s.f2) end for

**Case 4 : f = EX f’**

do MARKING(f’,K) ;

for all s ∈ S do s.f := false end for

for all (s, s’) ∈ R do

if s’.f’=true then s.f := true

end for

**Case 5 : f = E f1Uf2**

```
    do MARKING(f1,K) ; MARKING(f2,K) ;
    for all s ∈ S do
            s.f := false ;
            s.seenbefore := false
    end for
    L := ∅
    for all s ∈ S do if s.f2=true then L :=L + {s} end for
    while L != ∅ do
            choose s ∈ L ; L := L - {s} ;
            s.f := true ;
    for all (s',s) ∈ R do // s' predecessor of s
            if s'.seenbefore = false then
                    s'.seenbefore := true ;
                    if s'.f1 = true then L := L + {s'} ;
            end if
    end for
end while
....
```