Vérification de propriétés CTL

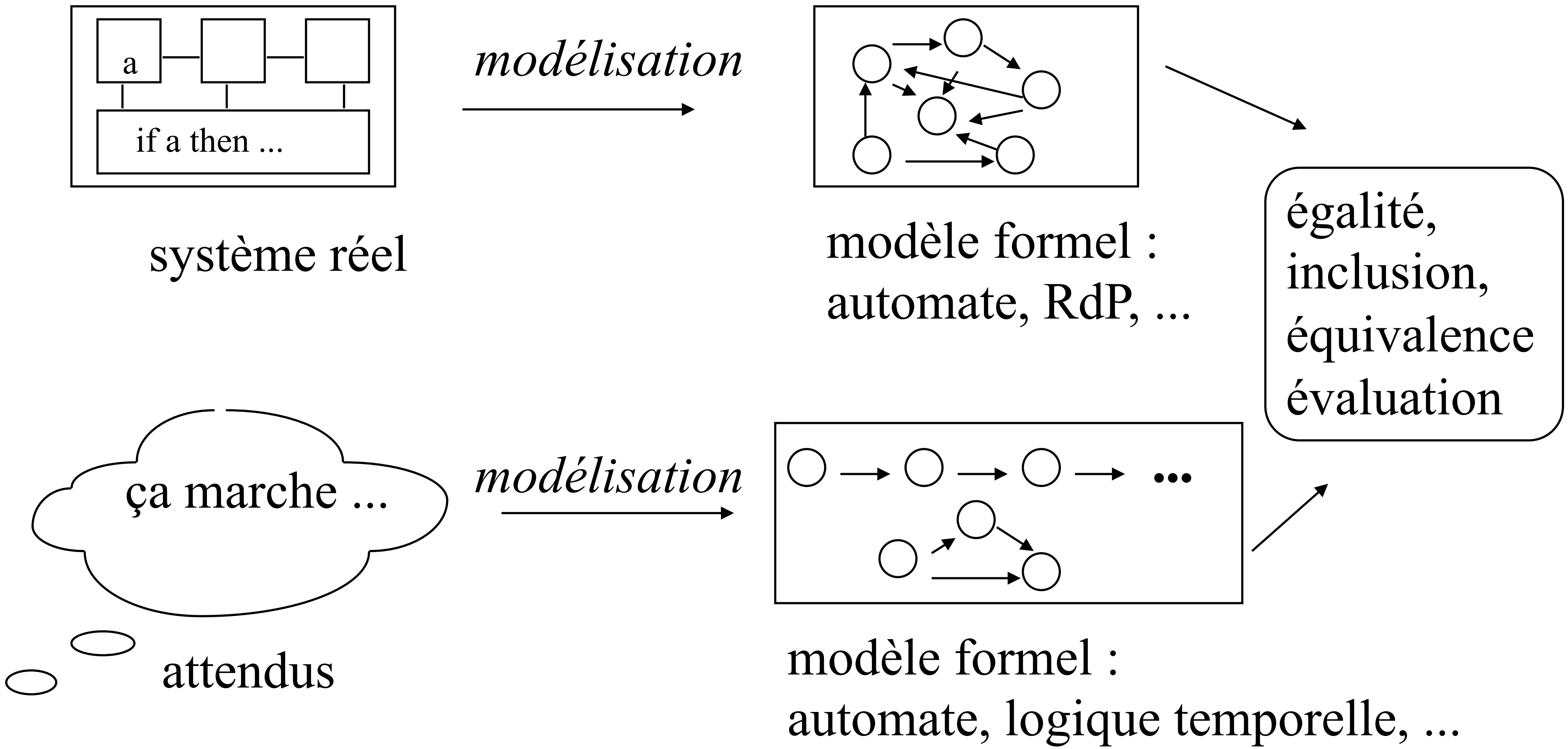
1- Rappel: définition de CTL

2- Algorithmes élémentaires du model-checking

3- Utilisation de NuSMV (TP)

E. Encrenaz-Tiphene

Processus de vérification



Modèle du système

Structure de Kripke = un système de transitions étiqueté sur états :
<S,T,So,AP,h>

S : ensemble fini d'états
T : relation de transition incluse dans $S \times S$
S_o : ensemble des états initiaux
AP : ensemble des propositions atomiques
h : fonction de satisfaction : $S \rightarrow 2^{AP}$ qui associe à chaque état les propriétés atomiques vraies dans cet état.

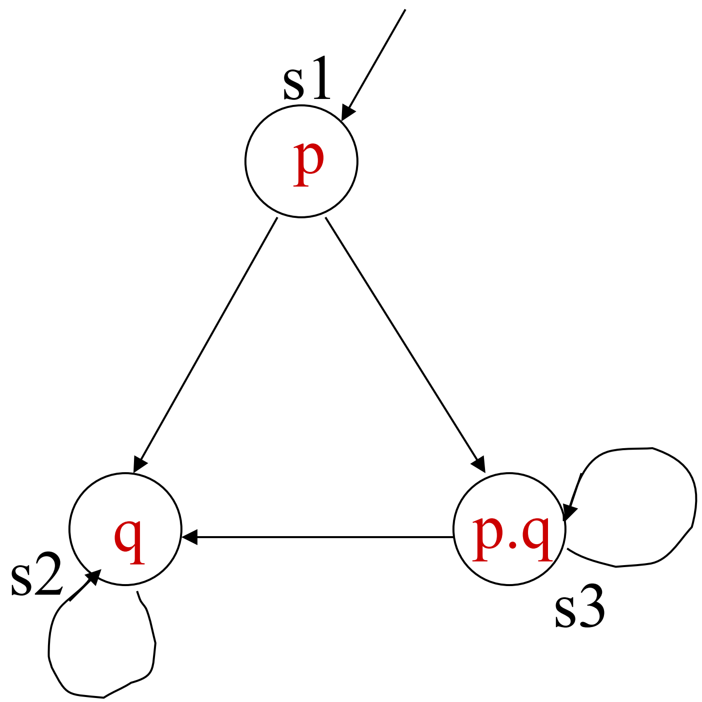
Une structure de Kripke

- <S,T,So,AP,h>
  - S = {s1, s2, s3}
  - T = {(s1,s2), (s1,s3), (s2,s2), (s3,s2),(s3,s3)}
  - So = {s1}
  - AP = {p,q}
  - h :

    s1 → p

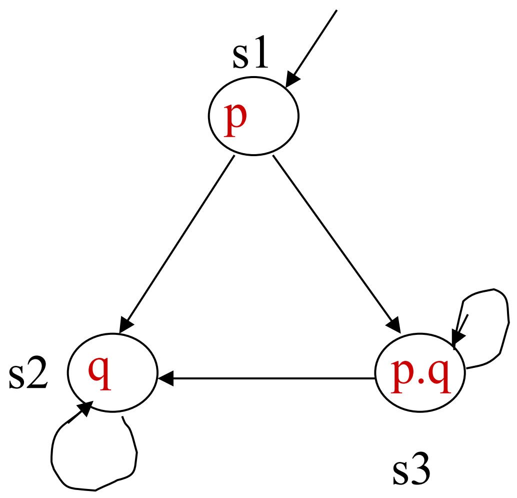
    s2 → q

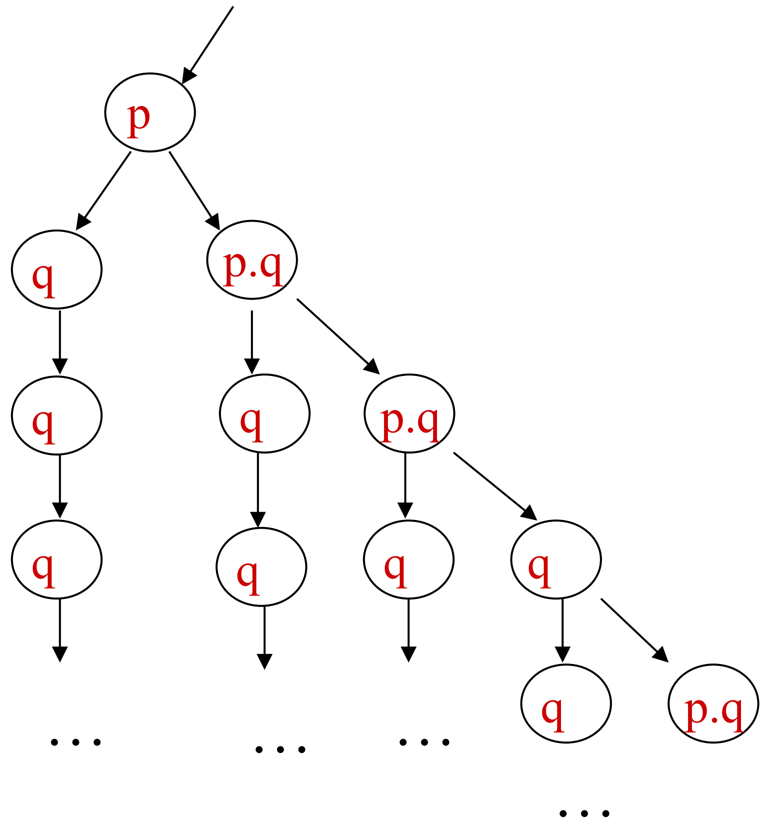
    s3 → p.q



Arbre d'exécution infini

- Obtenu par *déroulement* de la structure de Kripke





Modèle de la spécification

Logique temporelle CTL :

Une formule CTL f est définie inductivement (f_1, f_2 sont des formules CTL, AP un ens. des prop atomiques) :

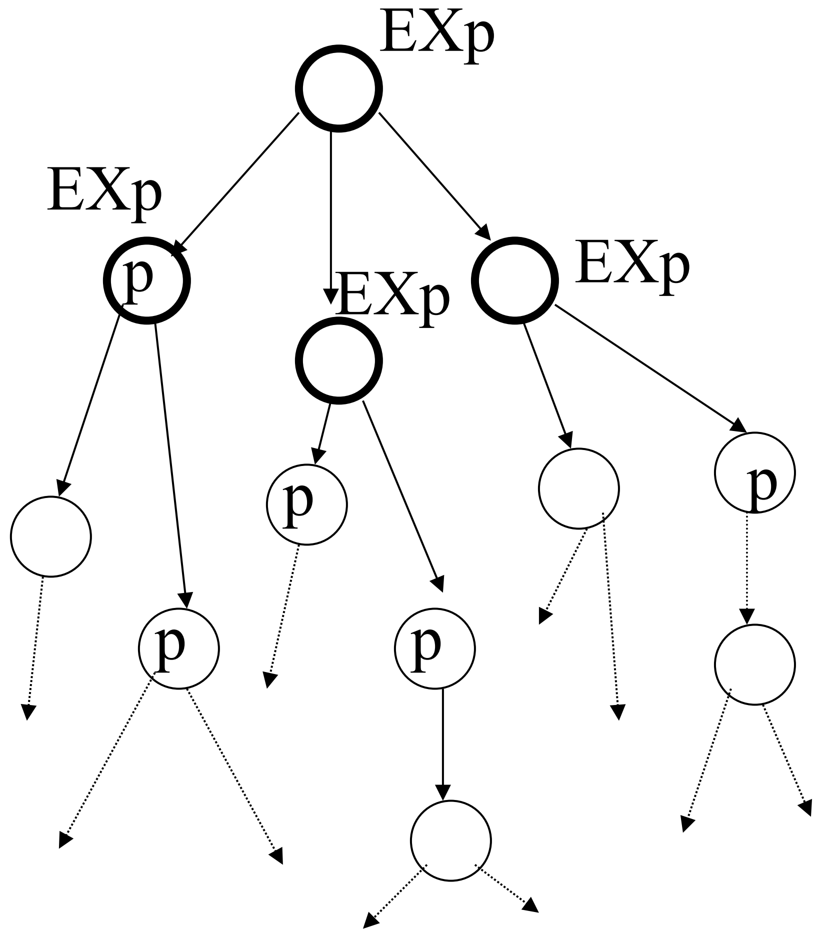
- une formule propositionnelle sur AP,
- f_1 `or` f_2, f_1 `and` f_2, `not` f_1,
- **EX** f_1, **AX** f_1,
- **E** f_1**U** f_2, **A** f_1 **U** f_2.

Modèles de formule CTL : états de l'arbre d'exécution infini obtenu par déroulement de la structure de Kripke K=<S,T,S_0, AP, h>

Opérateur EX

s |= EX p : il existe s' un successeur de s tel que s' |= p

Etats vérifiant EXp

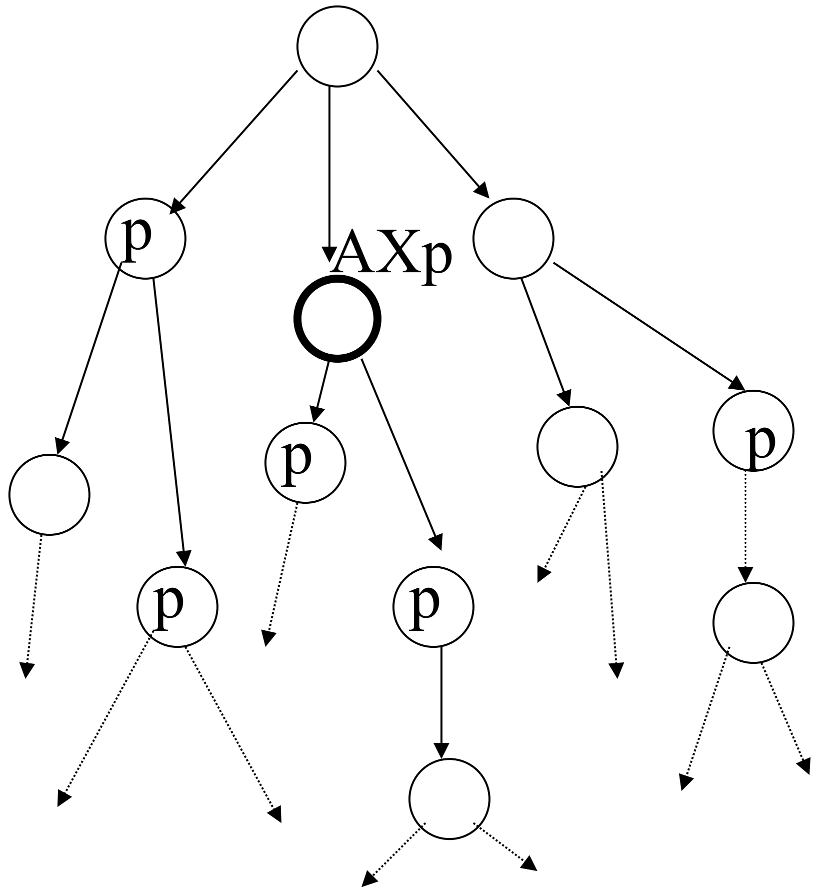


Rq : p est une proposition atomique ou une sous-formule CTL

Opérateur AX

s |= AX p : pour tout successeur s'
de s, on a s' |= p

Etats vérifiant AXp

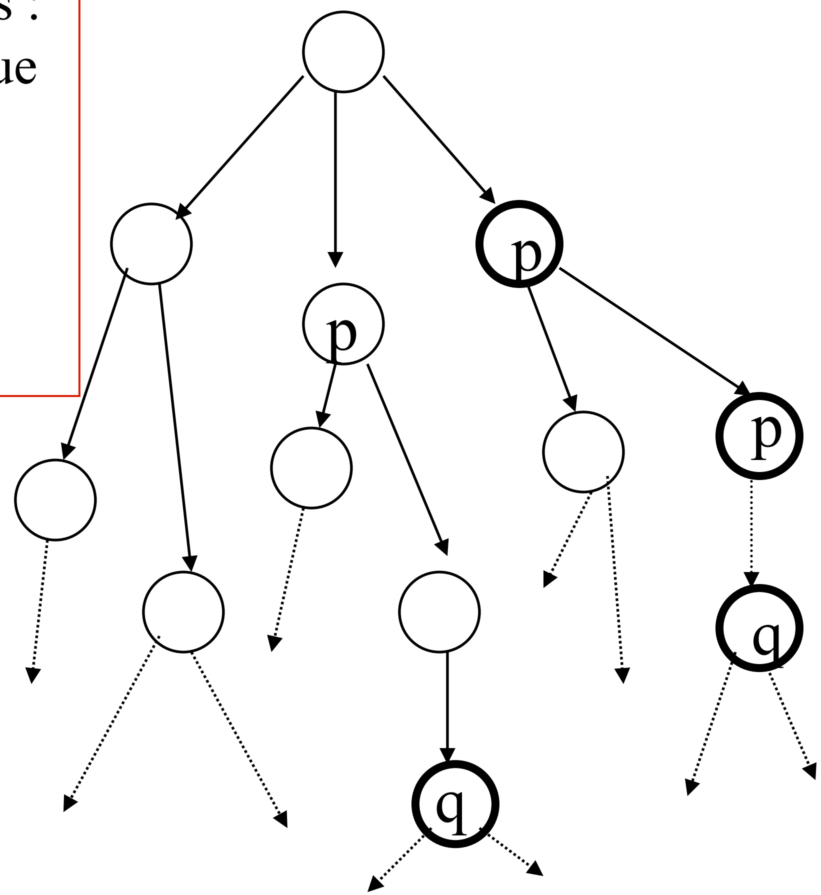


Opérateur EU

s |= E p U q : il existe une séquence σ issue de s :
σ = s→s'… →s" → … telle que

- s |= q ou alors
- s |= p et il existe s" |= q et
  quelquesoit s' sur σ entre s et s", s' |= p

Etats vérifiant E pUq

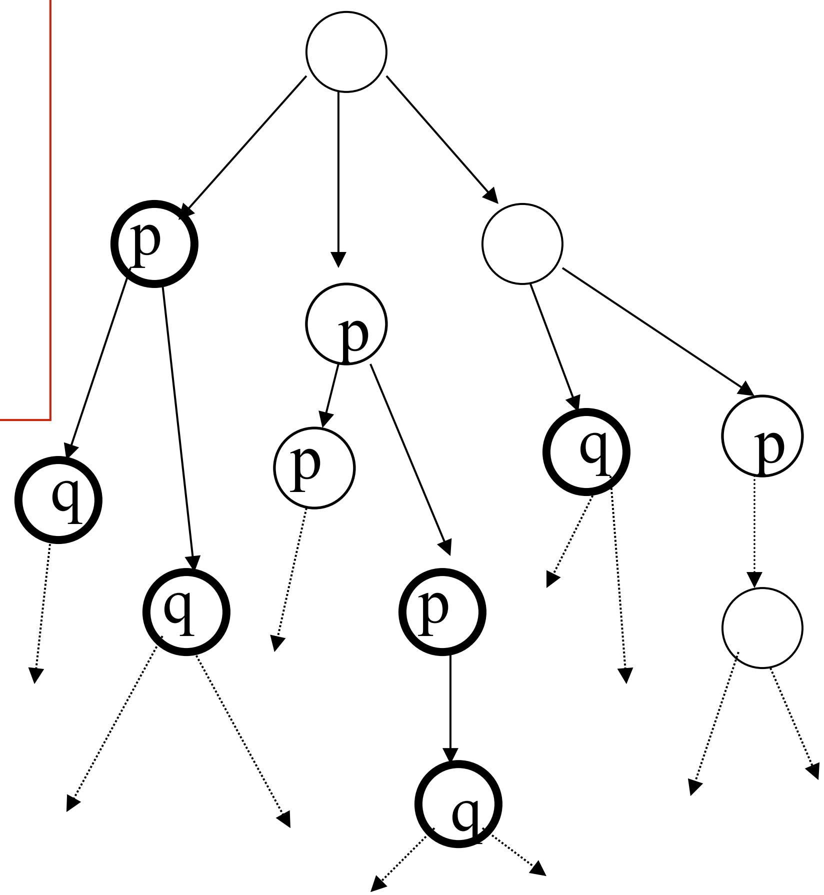


Opérateur AU

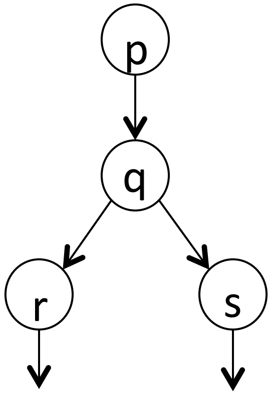
Etats vérifiant A pUq

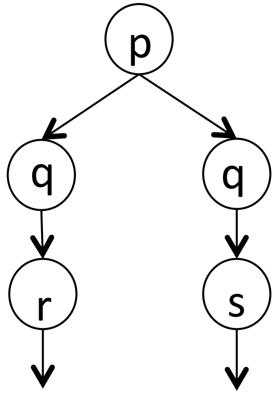
s |= A p U q : pour toute séquence σ issue de s :
σ = s→s'… →s" → … vérifie

- s |= q ou alors
- s |= p et il existe s" |= q et
  quelquesoit s' sur σ entre s et s", s' |= p



CTL permet de distinguer les points de branchement





« de p, on atteint **forcément** un état q, à partir duquel **il existe un état successeur** satisfaisant r **et il existe un état successeur** satisfaisant s »

$p \wedge AX\ (q \wedge (EX\ r \wedge EXs))$

« de p, on **peut** atteindre un état satisfaisant q, dont **tous les successeurs** satisfont r et un autre état q dont **tous les successeurs** satisfont s »

$p \wedge EX\ (q \wedge AX\ r) \wedge EX\ (q \wedge AX\ s)$

D'autres opérateurs non primitifs

$\mathbf{AF}\ p = \mathbf{A}\ (\mathrm{True}\ \mathbf{U}\ p)$,

$\mathbf{EF}\ p = \mathbf{E}\ (\mathrm{True}\ \mathbf{U}\ p)$,

$\mathbf{AG}\ p = \mathbf{A}\ (\neg\ \mathbf{F}\ (\neg\ p)) = \neg\ \mathbf{EF}\ (\neg\ p)$,

$\mathbf{EG}\ p = \mathbf{E}\ (\neg\ \mathbf{F}\ (\neg\ p)) = \neg\ \mathbf{AF}\ (\neg\ p)$,

Sémantique de CTL

Rq : p est une proposition atomique ou et f_1 et f_2 sont 2 formules CTL

$s \models p \Leftrightarrow$ p est vrai dans l'état s

$s \models f_1 \vee f_2 \Leftrightarrow s \models f_1$ ou $s \models f_2$

$s \models f_1 \wedge f_2 \Leftrightarrow s \models f_1$ et $s \models f_2$

$s \models \neg f_1 \Leftrightarrow s \not\models f_1$

$s \models EX\ f_1 \Leftrightarrow \exists s'$ tq $s \rightarrow s'$ et $s' \models f_1$

$s \models AX\ f_1 \Leftrightarrow \forall\ s'$ tq $s \rightarrow s'$ et $s' \models f_1$

$s \models EF\ f_1 \Leftrightarrow \exists \sigma = s \rightarrow s' \rightarrow \ldots s'' \ldots \rightarrow \ldots.$ et $s'' \models f_1$

$s \models AF\ f_1 \Leftrightarrow \forall\ \sigma = s \rightarrow s' \rightarrow \ldots s'' \ldots \rightarrow \ldots.$ et $s'' \models f_1$

$s \models EG\ f_1 \Leftrightarrow \exists \sigma = s \rightarrow s' \rightarrow \ldots s^k \ldots \rightarrow \ldots.$ et $\forall\ k \geq 0, s^k \models f_1$ (et $s = s^0$)

$s \models AG\ f_1 \Leftrightarrow \forall\ \sigma = s \rightarrow s' \rightarrow \ldots s^k \ldots \rightarrow \ldots.$ et $\forall\ k \geq 0, s^k \models f_1$ (et $s = s^0$)

$s \models E\ f_1\ U\ f_2 \Leftrightarrow \exists\ \sigma = s \rightarrow s^1 \rightarrow \ldots s^k \ldots \rightarrow \ldots.$ et $\exists$ k tq $s^k \models f_2$ et $\forall\ k > j \geq 0, s^j \models f_1$ (et $s = s^0$)

$s \models A\ f_1\ U\ f_2 \Leftrightarrow \forall\ \sigma = s \rightarrow s^1 \rightarrow \ldots s^k \ldots \rightarrow \ldots.$ et $\exists$ k tq $s^k \models f_2$ et $\forall\ k > j \geq 0, s^j \models f_1$ (et $s = s^0$)

Réécriture

On peut définir tous les opérateurs à partir de **EX**, **EG** et **EU** :

EF(p) = True **U** p

AX(p) = ¬ **EX** (¬p)

AF(p) = ¬ **EG** (¬p)

AG(p) = ¬ **E** True **U** ¬p

A p **U** q = ¬**E** ¬q **U** (¬p ∧ ¬q) ∧ ¬ **EG** (¬q)

Définitions expansives des opérateurs **EF**, **EG** et **EU** :

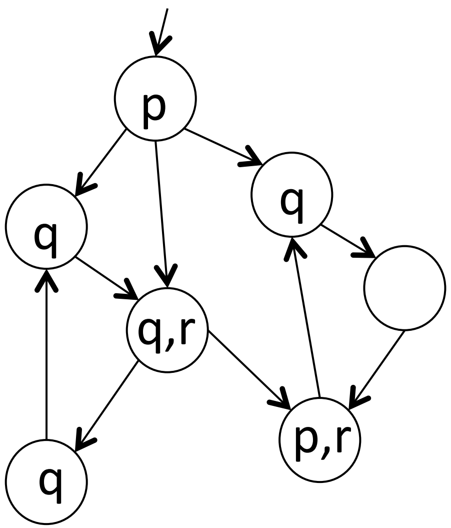
EF p = p ∨ **EX** **EF**(p)

EG p = p ∧ **EX** **EG**(p)

E p **U** q = q ∨ (p ∧ **EX** (**E** p **U** q))

Model-checking de propriété CTL

- Soit M un système décrit par une structure de Kripke et φ une propriété CTL; on dit que M $\models \varphi$ ssi s_0 l'état initial de S, satisfait φ : $s_0 \models \varphi$



A-t-on M $\models \varphi$?

$\varphi = p \wedge \mathrm{E}X\, q$

$\varphi = p \wedge \mathrm{A}X\, q$

$\varphi = EG\, q$

$\varphi = r \Rightarrow EX\, EG\, q$

$\varphi = AX\, (EG\, q)$

$\varphi = AG\, (r \Rightarrow A\, (p\, U\, q))$

$\varphi = AG\, EF\, q$

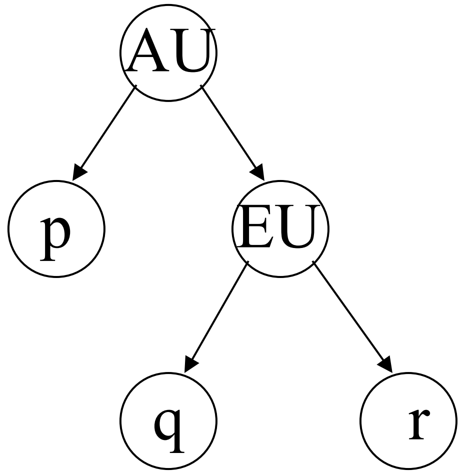
$\varphi = AG\, AF\, q$

Vérification de propriétés CTL

- Soit M un système décrit par une structure de Kripke et φ une propriété CTL;

  on dit que M $|= \varphi$ ssi s_0 l'état initial de S, satisfait φ : $s_0 |= \varphi$

1. Construire l'arbre syntaxique de la formule φ
   - –nœuds : opérateurs CTL
   - –feuilles : propriétés atomiques



A(p U E(q U r))

2. Réécrire l'arbre en utilisant uniquement les opérateurs
   EX, AX, EU, AU

3. Parcourir l'arbre et pour chaque nœud opérateur N, appliquer la procédure de marquage de la structure de Kripke adaptée (Algo1, Algo2, Algo3 ou Algo4 ci-après).
4. Si s_0 est marqué comme satisfaisant φ, alors M $|= \varphi$

Vérification de EXp et AXp

```
Algo 1 : EX p
Pour chaque état e faire
  Si ∃ e' un successeur  de e tel e' |= p Alors
      e |= EX p
  Sinon
      e |= NOT EX p

Algo 2 : AX p
Pour chaque état e faire
  Si ∀ e' un successeur de e , e' |= p Alors
      e |= AX p
  Sinon
      e |= NOT AX p
```

Vérification de EpUq (1)

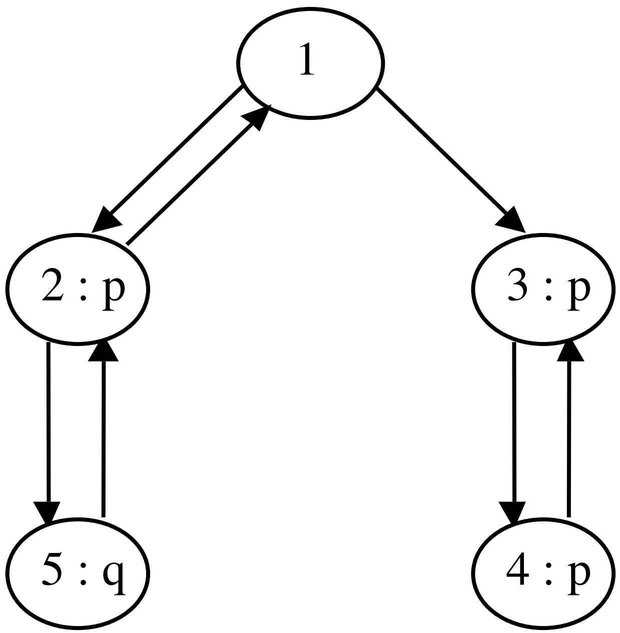
```
Algo 3 : E p U q
Début
   ∀e Démarquer(e)
   ∀e Vérifier_EU(e)
   ∀e Si e est non marqué Alors e |= NOT E p U q
Fin

Procédure Vérifier_EU(e)
Si e est non marqué Alors
   Si e|= q Alors
        Marquer(e)
        e|= E p U q
        ∀ e' un prédécesseur de e , Propager_EU(e')
   Sinon si e |= NOT p Alors
        Marquer(e)
        e |= NOT E p U q
   Finsi
Finsi
```

Vérification de ∃pUq (2)

```
Procédure Propager_EU(e)
Si e est non marqué Alors
   Marquer(e)
   Si e|= q ou e|= p Alors
        e|= E p U q
        ∀ e' un prédécesseur de e , Propager_EU(e')
   Sinon
        e |= NOT E p U q
   Finsi
Finsi
```

Vérification de E pUq - exemple



Vérification de ApUq (1)

Algo 4 : A p U q

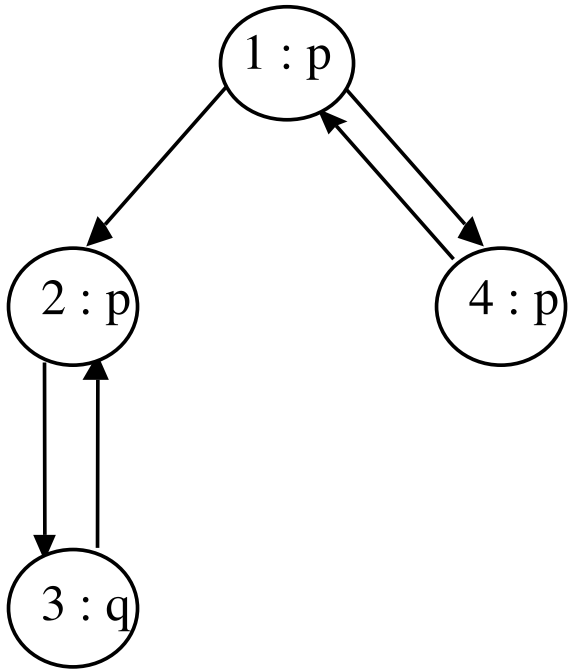
```
Début
  ∀e Démarquer(e)
  ∀e Vérifier_AU(e)
Fin

Procédure Vérifier_AU(e)
Si e est marqué Alors
  Si e|= A p U q Alors
      Retour(Vrai)
  Sinon
      Retour(Faux)
  Finsi
sinon ...
```

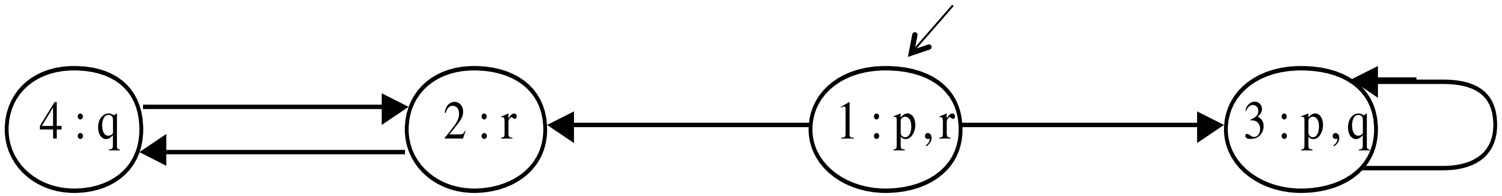
Vérification de ApUq (2)

```
...
Sinon
  Marquer(e)
  Si e|= q Alors
      e|= A p U q; Retour(Vrai)
  Sinon
      Si e |= NOT p Alors
            e |= NOT A p U q; Retour(Faux)
      Sinon
      Si ∀e' successeur de e, Vérifier_AU(e') Alors
                  e|= A p U q; Retour(Vrai)
            Sinon
                  e|= NOT A p U q; Retour(Faux)
      Finsi
  Finsi
Finsi
```

Vérification de ApUq - exemple



Vérification d'une formule avec opérateurs imbriqués - exemple



Evaluation de A p U (E q U r)

Conclusion

- Complexité de l'algorithme : $O(|S| \times |\varphi|)$
- Utilisable sur des systèmes à 10^6-10^7 états (20 bits de registres)
- Améliorations
  - Manipulation *d'ensembles* d'états (model-checking symbolique) (100 bits)
  - Restriction de la profondeur des traces d'exécution analysées et utilisation de SAT/SMT solvers (1000-10000 bits selon la propriété)
  - Abstractions :
    - Identifier une relation d'équivalence entre états et appliquer le model-checking sur le graphe quotient
    - Construction de la relation d'équivalence, préservant les propriétés intéressantes (pas tjs automatique)
  - Raisonnement compositionnel
    - Modèle structuré en composants, disposant de propriétés locales (vérifiables sur chaqu composant)
    - Vérifier des propriétés globales par assemblage des propriétés locales (pas tjs possible)

Des parcours plus efficaces

Manipulation unitaire d'ensembles d'états étiquetés par la même sous-formule plutôt que des états pris individuellement.

On note S_φ l'ensemble des états de la structure de Kripke satisfaisant φ

On note PRE(X) l'ensemble des états prédécesseurs en une transition d'un ensemble d'états X

$PRE(X) = \{ e \mid \exists e' : e \rightarrow e' \text{ et } e' \in X \}$

Quelques éléments sur les points fixes

[Tarski 50]

- Soient Q un ensemble, 2^Q l'ensemble des parties de cet ensemble

- Soit $f : 2^Q \rightarrow 2^Q$

  f est monotone si : X et $Y \subseteq 2^Q$ tq $X \subseteq Y$ alors $f(X) \subseteq f(Y)$

  f est $\cup$-continue (resp. $\cap$-continue) si $\cup f(Xi) = f(\cup Xi)$ (resp. $\cap f(Xi) = f(\cap Xi)$)

- Si f est monotone et continue, alors
  - $\cup f^i(\emptyset)$ est son plus petit point fixe, majoré par Q
  - $\cap f^i(\emptyset)$ est son plus grand point fixe, minoré par $\emptyset$ [Kleene]

- Pour $Y \subseteq Q$ et $F : 2^Q \rightarrow 2^Q$ continue,
  - PRE
  - $X \rightarrow Y \cup F(X)$
  - $X \rightarrow Y \cap F(X)$ sont continues : leur solution est leur point fixe

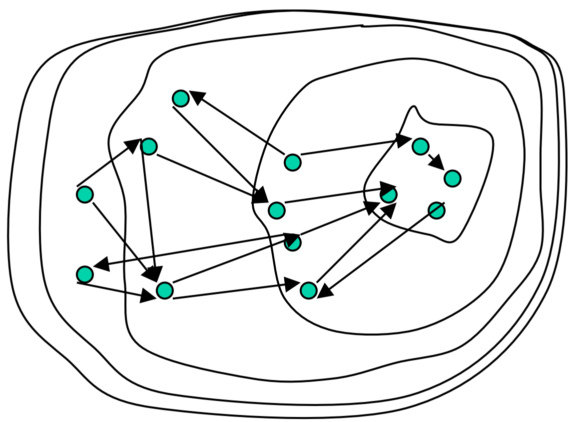
Calculs de point-fixe des opérateurs CTL (1)

- $S_{EX\,p} = \{e \mid e \rightarrow e' \text{ et } e' \models p\} = PRE\,(S_p)$

- $S_{EF\,p} = \{e \mid e \models p \text{ ou } e \rightarrow e' \ldots \rightarrow e^n \rightarrow \ldots \text{ et } e^n \models p\}$
  $= \{e \mid e \models p \text{ ou } e \rightarrow e' \text{ et } e' \models EFp\}$
  $= \{e \mid e \models p\} \cup \{e \mid e \rightarrow e' \text{ et } e' \models EFp\}$
  $= S_p \cup PRE\,(S_{EF\,p})$

Suite monotone croissante convergeant vers son plus petit point fixe

(On écrit aussi : $S_{EF\,p} = \mu.\ X = p \cup PRE\,(X)$)



Calcul du point-fixe de EF p

- $S_{EF\,p} = \underbrace{\underbrace{\underbrace{\{p\}}_{S_{EF^0}} \cup PRE\{p\}}_{S_{EF^1}} \cup PRE\,PRE\{p\}}_{S_{EF^n}} \ldots \cup PRE^n\{p\} \ldots \cup \ldots$

- $S_{EF^i\,p} = S_p \cup PRE\,(S_{EF^{i-1}\,p})$ si $S_{EF^i\,p} = S_{EF^{i-1}\,p}$, le point fixe est atteint

```
Calcul_point_fixe_EF(p,y) /* initialisé avec y = ∅ */
      y' ← {p} ∪ PRE(y)
      si y' = y alors retourner y
      sinon retourner Calcul_point_fixe_EF(p,y')
fin
```

Calcul de point-fixe des opérateurs CTL (2)

- $S_{EG\,p} = \{e \mid e \models p \text{ et } e \rightarrow e' \ldots \rightarrow e^n \rightarrow \ldots \text{ et } e' \models p \text{ et } e^n \models p\}$
  $= \{e \mid e \models p \text{ et } e \rightarrow e' \text{ et } e' \models EGp\}$
  $= \{e \mid e \models p\} \cap \{e \mid e \rightarrow e' \text{ et } e' \models EGp\}$
  $= S_p \cap PRE(S_{EG\,p})$

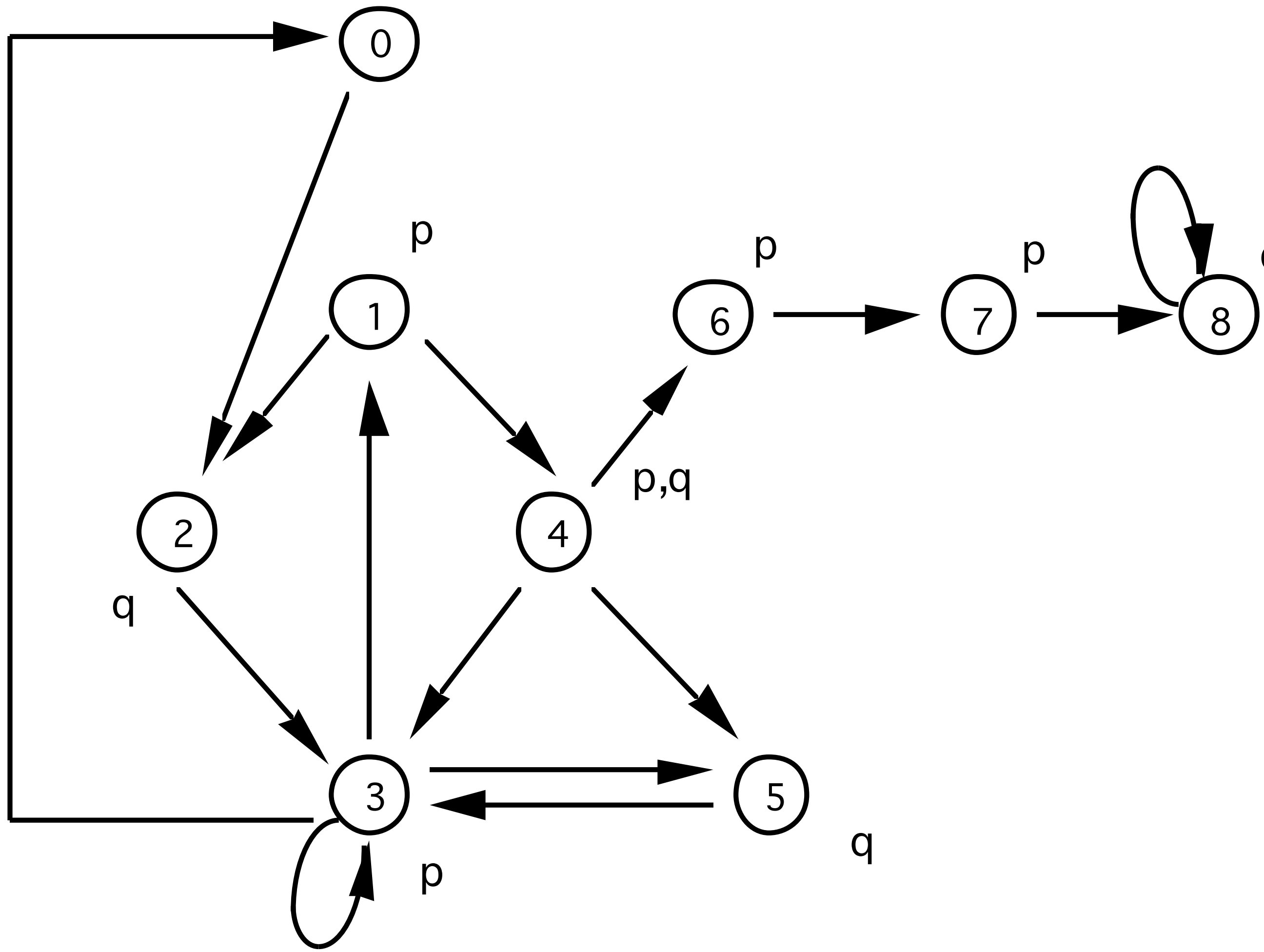
Suite monotone décroissante convergeant vers son plus grand point fixe

- $S_{E\,p\,U\,q} = \{e \mid e \models q \text{ ou } e \models p \text{ et } e \rightarrow e' \ldots \rightarrow e^n \rightarrow \ldots$
  $\text{et } \exists\, j \text{ tq } e^j \models q\ \forall\, i < j,\ e^i \models p\}$
  $= \{e \mid e \models q \text{ ou } e \models p \text{ et } e \rightarrow e' \text{ et } e' \models E\,p\,U\,q\}$
  $= \{e \mid e \models q\} \cup \{e \mid e \models p \text{ et } e \rightarrow e' \text{ et } e' \models E\,p\,U\,q\}$
  $= S_q \cup (S_p \cap PRE(S_{EpUq})$

Suite monotone croissante convergeant vers son plus petit point fixe

Calcul par point-fixe - exemple

Déterminez S_{EFp}, S_{EGp} et S_{EpUq} par calcul de point-fixe



Model-Checker symbolique (1)

Formule CTL

Ens des états vérifiant la formule

```
function Eval(φ)
   case
        φ : prop atomique p => retourner Sp
        φ = ¬ q        => retourner complement(Eval(q))
        φ = r ∨ s      => retourner Eval(r) ∪ Eval(s)
        φ = EXq        => retourner EvalEX(Eval(q))
        φ = EGq        => retourner EvalEG(Eval(q),true)
        φ = EU(r,s)    => retourner EvalEU(Eval(r), Eval(s),false)
   fin case
fin
```

Ens des états vérifiant φ

```
function EvalEX(Sφ)
   retourner PRE(Sφ)
fin
```

Ens des états vérifiant EXφ

Model-Checker symbolique (2)

Ens des états vérifiant EGφ
à l'itération courante

```
function EvalEG(Sφ,y)
   y' ← Sφ ∩ EvalEX(y)
   si y' = y alors
       retourner y
   sinon
       retourner EvalEG(Sφ,y')
fin
```

Ens des états vérifiant EφUψ
à l'itération courante

```
function EvalEU(Sφ,Sψ,y)
   y' ←  Sψ ∪ (Sφ ∩ EvalEX(y))
   si y' = y alors
       retourner y
   sinon
       retourner EvalEU(Sφ,Sψ,y')
fin
```

Model-checking symbolique de propriété CTL

$$M \models \varphi \text{ ssi } s_0 \in \texttt{\textbf{Eval}}(\varphi)$$

Model Checker NuSMV

- Successeur de SMV, développé par CMU en 1995, intégrant les différentes techniques développées sur les dernières décennies :
  - LTL et CTL,
  - Bounded Model-Checking
  - Algorithmes énumératif, BDD, SAT, équités

- http://nusmv.fbk.eu