Model Checking

INTRODUCTION

MODEL-CHECKING LTL

MODEL-CHECKING CTL

CONCLUSION

2020-2021

INTRODUCTION

Problème du Model Checking

Formule LTL, CTL,… φ

Modèle (structure de Kripke K)

Model Checking

$$K \overset{?}{\models} \varphi$$

Introduction

- Les formules LTL sont des formules qui portent sur les chemins d'exécution.
- Un automate, représentant le modèle à vérifier, même fini, donne souvent lieu à une infinité d'exécutions différentes de longueur infinie.
- Pour procéder au model checking de LTL, il faut se baser sur la théorie des langages.

Illustration

- Soient un modèle M et une formule ϕ : **GF***p* (*toujours il y aura un état tel que p*).

Illustration

- Soient un modèle M et une formule ϕ : **GF***p* (*toujours il y aura un état tel que p*).
- Une exécution de M $q_0, q_1, \ldots$ vérifiant ϕ doit contenir une infinité de positions $q_{n1}, q_{n2}, \ldots$ où *p* est vérifiée.

q_0
|
q_1
|
q_2
.
.
q_{n-1}
|
q_n
|
q_{n+1}
|
q_{n+2}
.
.

Illustration

- Soient un modèle M et une formule ϕ : **GF***p* (*toujours il y aura un état tel que p*).
- Une exécution de M $q_0, q_1, \ldots$ vérifiant ϕ doit contenir une infinité de positions $q_{n1}, q_{n2}, \ldots$ où *p* est vérifiée.

q_0	q_0
|	|
q_1	$q_1 p$
|	|
q_2	q_2
.	.
.	.
q_{n-1}	$q_{n-1} p$
|	|
q_n	q_n
|	|
q_{n+1}	
|	|
q_{n+2}	$q_{n+2} p$
.	.
.	.

Illustration

- Soient un modèle M et une formule ϕ : **GF**p (*toujours il y aura un état tel que p*).
- Une exécution de M $q_0, q_1, \ldots$ vérifiant ϕ doit contenir une infinité de positions $q_{n1}, q_{n2}, \ldots$ où p est vérifiée.
- Entre ces positions il peut y avoir un nombre fini d'états vérifiant $\neg p$. on dit que l'exécution est de la forme $((\neg p)^* p)^\omega$. ω *désigne un nombre infini.*

q_0	q_0
$\vert$	$\vert$
q_1	$q_1 p$
$\vert$	$\vert$
q_2	q_2
.	.
.	.
q_{n-1}	$q_{n-1} p$
$\vert$	$\vert$
q_n	q_n
$\vert$	$\vert$
q_{n+1}	
$\vert$	$\vert$
q_{n+2}	$q_{n+2} p$
.	.
.	.

Illustration

- Soient un modèle M et une formule ϕ : **GF**p (*toujours il y aura un état tel que p*).
- Une exécution de M $q_0, q_1, \ldots$ vérifiant ϕ doit contenir une infinité de positions $q_{n1}, q_{n2}, \ldots$ où p est vérifiée.
- Entre ces positions il peut y avoir un nombre fini d'états vérifiant $\neg p$. on dit que l'exécution est de la forme $((\neg p)^* p)^\omega$. ω *désigne un nombre infini.*

q_0	q_0	$q_0 \neg p$
q_1	$q_1 p$	$q_1 p$
q_2	q_2	$q_2 \neg p$
.	.	.
.	.	.
q_{n-1}	$q_{n-1} p$	$q_{n-1} p$
q_n	q_n	$q_n \neg p$
q_{n+1}	q_{n+1}	$q_{n+1} \neg p$
q_{n+2}	$q_{n+2} p$	$q_{n+2} p$
.	.	.
.	.	.

Illustration

- Une exécution ne vérifiant pas ϕ, doit, à partir d'une certaine position de l'exécution ne plus vérifier *p* et donc à partir de ce moment tous les états doivent vérifier $\neg p$.

Illustration

- Une exécution ne vérifiant pas ϕ, doit, à partir d'une certaine position de l'exécution ne plus vérifier *p* et donc à partir de ce moment tous les états doivent vérifier $\neg p$.
- Cette exécution est de la forme $(p|\neg p)^*(\neg p)^\omega$.

$q_0\ \neg p$

|

$q_1\ p$

|

$q_2\ \neg p$

.

.

$q_{n-1}\ p$

|

$q_n\ \neg p$

|

$q_{n+1}\ \neg p$

|

$q_{n+2}\ \neg p$

.

.

Illustration

- Une exécution ne vérifiant pas ϕ, doit, à partir d'une certaine position de l'exécution ne plus vérifier *p* et donc à partir de ce moment tous les états doivent vérifier $\neg p$.
- Cette exécution est de la forme $(p|\neg p)^*(\neg p)^\omega$.
- Il s'agit de ω-régulières, par analogie avec les expressions régulières, mais pour représenter des mots infinis.
- *ω signifie répétition un nombre infini de fois.*

$q_0 \neg p$
|
$q_1 p$
|
$q_2 \neg p$
.
.
$q_{n-1} p$
|
$q_n \neg p$
|
$q_{n+1} \neg p$
|
$q_{n+2} \neg p$
.
.

Mots infinis

- Un mot infini est un mot qui peut s'écrire sous la forme uv^{ω}.
- Un modèle M ne satisfait pas ϕ ($M \not\models \phi$) s'il existe toujours une trace uv^{ω} de M tel que $uv^{\omega} \not\models \phi$.

Mots infinis

- Un mot infini est un mot qui peut s'écrire sous la forme uv^{ω}.
- Un modèle M ne satisfait pas ϕ ($M \nvDash \phi$) s'il existe toujours une trace uv^{ω} de M tel que $uv^{\omega} \nvDash \phi$.
- Reprenons l'exemple avec ϕ=GF*p*, ici $uv^{\omega}=(p|\neg p)^*(\neg p)^{\omega}$, p n'apparait pas dans v $(\neg p)^{\omega}$ donc $M \nvDash GFp$.

q_0 $\neg p$
q_1 p
q_2 $\neg p$
:
q_{n-1} p
q_n $\neg p$
q_{n+1} $\neg p$
q_{n+2} $\neg p$

MODEL CHECKING LTL

Principe de base

- L'algorithme est du à Lichtenstein, Puneli, Vardi et Wolppoer.
- Associer à toute formule ϕ de LTL un ω-automate B_ϕ représenté par une expression ω-régulière décrivant la forme que doit respecter une exécution pour satisfaire ϕ.
- Représenter le modèle par un automate B_M de toutes les exécutions infinies de M.
- La question « A-t-on $B_M \models \phi$? » se ramène donc à la question « est-ce que toutes les exécutions maximales et initiales de B_M respectent B_ϕ ?».

Algorithme (informel)

- A partir d'une formule ϕ, construire un automate $B_{\neg\phi}$:
  - $L(\neg\phi)=\{\sigma\in(2^{AP}) \quad |\sigma \nvDash \phi\}$
- Construire un automate B_M de toutes les exécutions infinies de M.
- Construire l'automate reconnaissant $L(B_M)\cap L(B_{\neg\phi})$. C'est le résultat d'une synchronisation où les deux automates avancent simultanément. Le résultat est $B_\otimes=B_M\otimes B_{\neg\phi}$.
  - $L(B_\otimes)=\{\sigma|\sigma\in exec(M) \cap L(B_{\neg\phi})\}=\{\sigma \,|\sigma\in exec(M) \text{ et } \sigma\nvDash\phi\}$.
- Si $B_\otimes=\emptyset$ alors $M\vDash\phi$ sinon $M\nvDash\phi$.

Model checking LTL (schéma global)

Modèle M ➡ w-automate B_M

Propriété ϕ ➡ w-automate $B_{\neg\phi}$

Produit synchronisé
$B_M \otimes B_{\neg\phi}$

$L(B_M) \cap L(B_{\neg\phi}) = \varnothing$

oui

ϕ vérifie M
$M \models \phi$

non

Contre exemple
$M \in L(B_M \otimes B_{\neg\phi})$

Algorithme (informel)

- En appliquant cet algorithme, on obtient un automate dont les seuls comportements sont ceux de M qui sont tolérés par $B_{\neg\phi}$: *les exécutions de M ne vérifiant pas ϕ*.
- Le problème de MC :
  - $A \models \phi$? devient
  - $L(B_M \otimes B_{\neg\phi}) = \emptyset$?
- Mais pour appliquer cet algorithme on a besoin d'automates reconnaissant des mots infinis : automates de Büchi.

Automates de Büchi

- Un automate de Büchi est un n-uplet *B*=(Q, Σ, q_0, T, F) avec
  - Q un ensemble fini d'états.
  - Σ un alphabet fini.
  - $q_0 \in Q$ l'état initial.
  - $T \subseteq Q \times \Sigma \times Q$ la relation de transition.
  - $F \subseteq Q$ un ensemble d'états acceptants (ou répétés) appelés « *Büchi states* ».

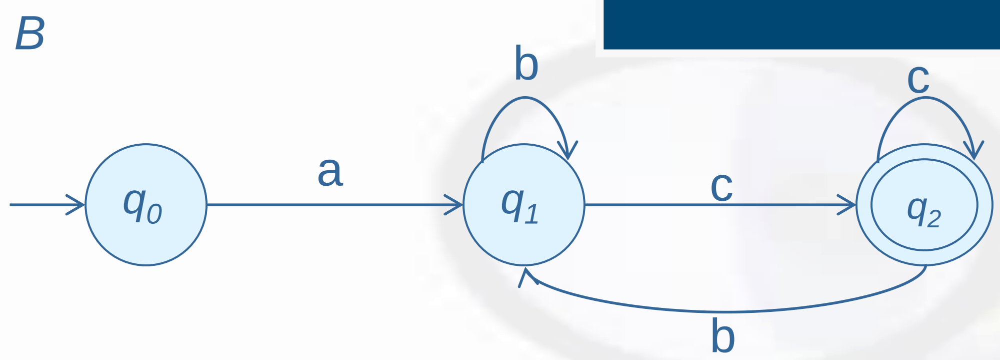
Automates de Büchi (exécution reconnaissance)

- Soient un automate de Büchi $B=(Q, \Sigma, q_0, T, F)$ et un mot infini $w \in \Sigma^w$.
- Une exécution σ de B sur $w=w_0w_1w_2\ldots$ est une séquence infinie $q_0q_1q_2\ldots$ d'états tel que pour tout $i \geq 0$, $(q_i,w_i,q_{i+1}) \in T$.
- Une exécution σ est acceptée si pour tout i ≥ 0, il existe j>i tel que $q_j \in B$.
- Un mot w est accepté s'il existe une exécution acceptée sur w.

Acceptation (informelle)

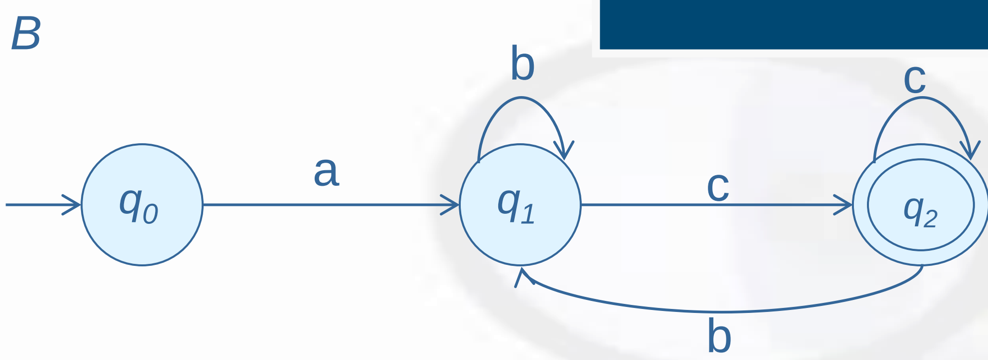
- Un mot infini w est accepté si l'automate commence par un état initial, respecte la relation de transition et passe *infiniment souvent par un état acceptant f* $\in$*B*.

Exemple



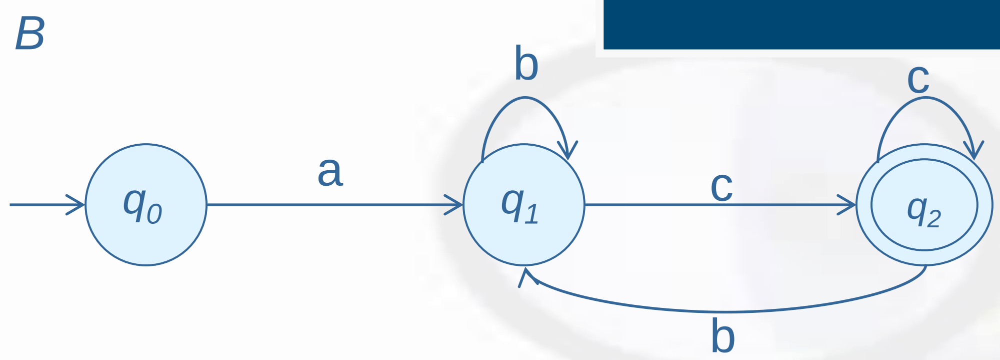
- w1=acccccccc…

Exemple



- w1=accccccc… Acceptée

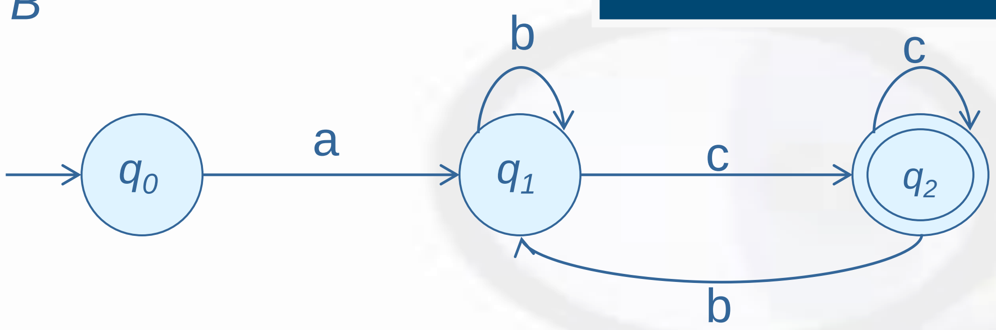
Exemple



- w1=acccccccc… Acceptée
- w2=acbcbcb…

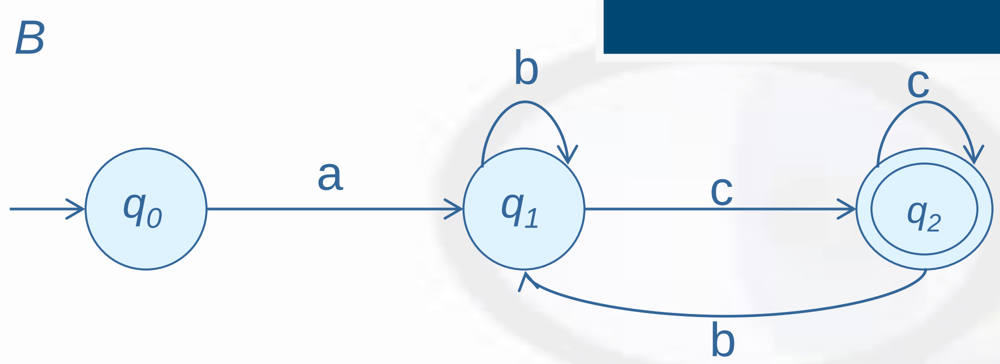
Exemple

B



- w1=acccccccc… Acceptée
- w2=acbcbcb… Acceptée

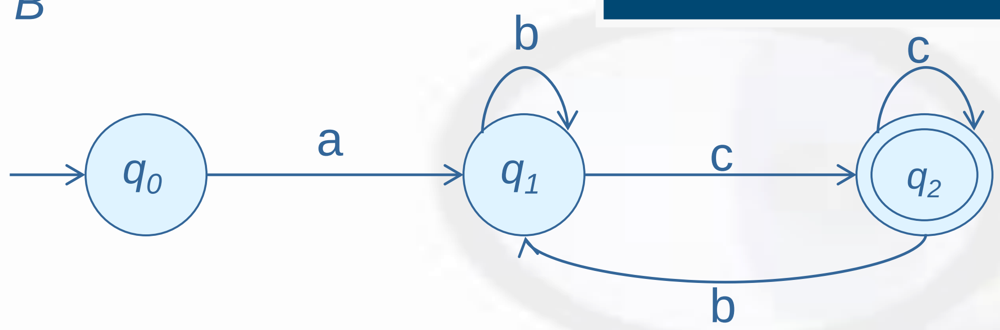
Exemple



- w1=accccccc… Acceptée
- w2=acbcbcb… Acceptée
- w3=acbbbbbb…

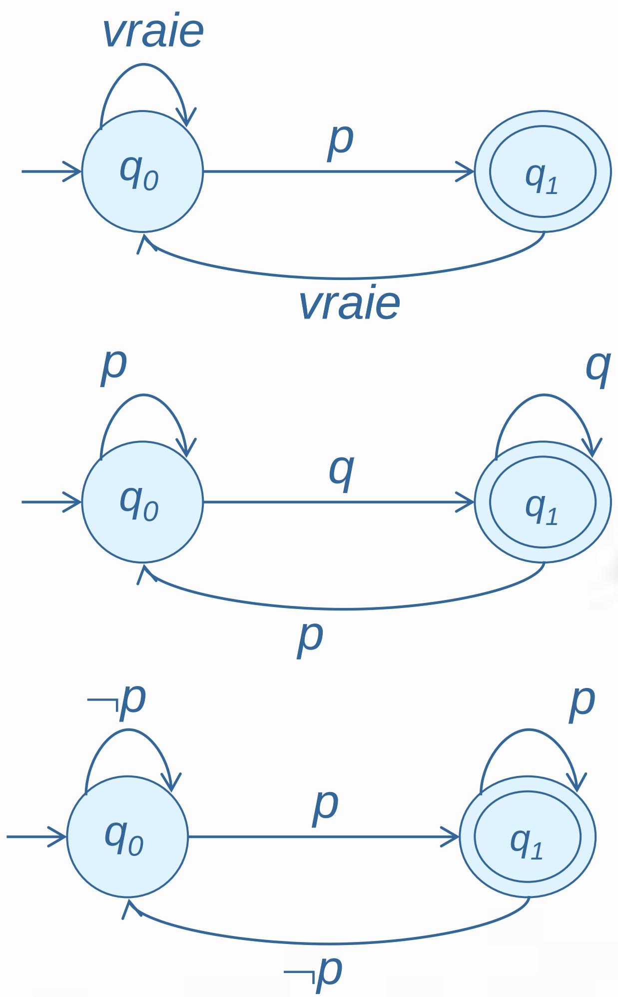
Exemple

B

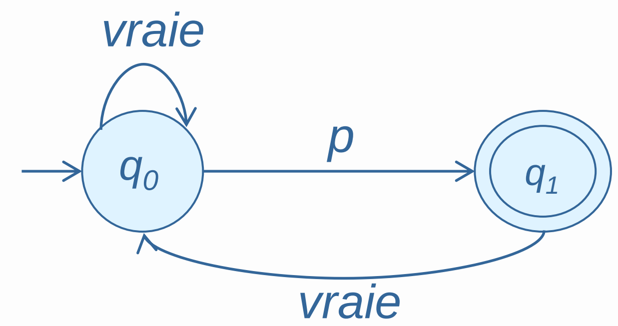


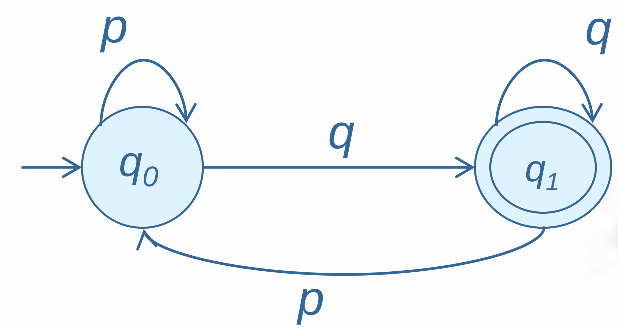
- w1=acccccccc… Acceptée
- w2=acbcbcb… Acceptée
- w3=acbbbbbbb… Rejetée

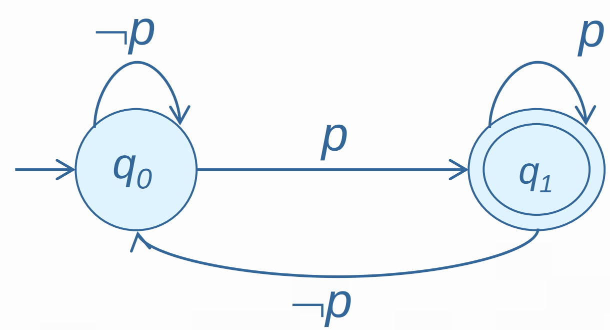
Quelques automates de Büchi



Quelques automates de Büchi

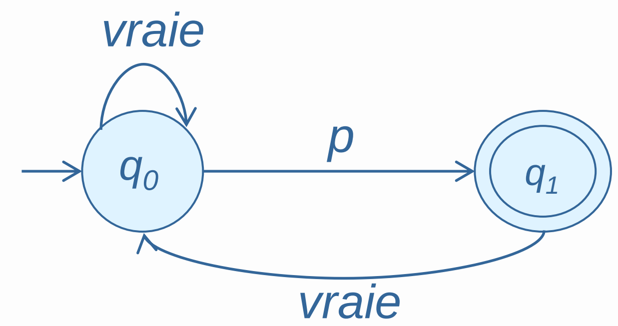




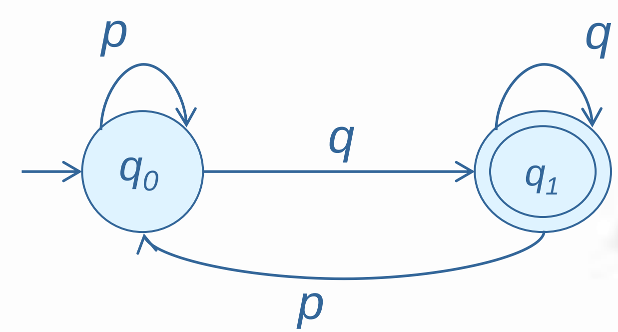


- GFp

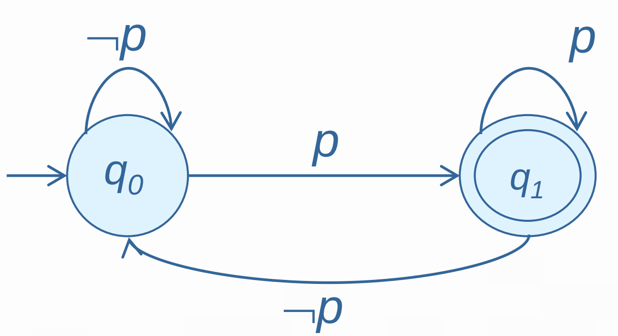
Quelques automates de Büchi



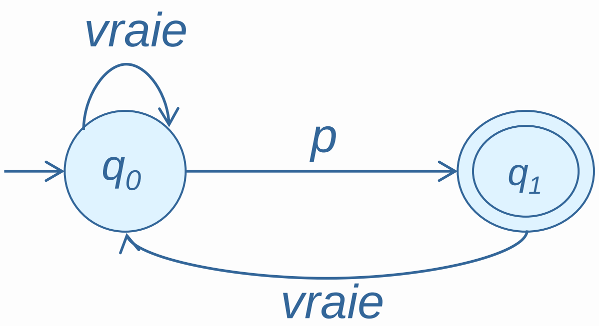
- **GF***p*



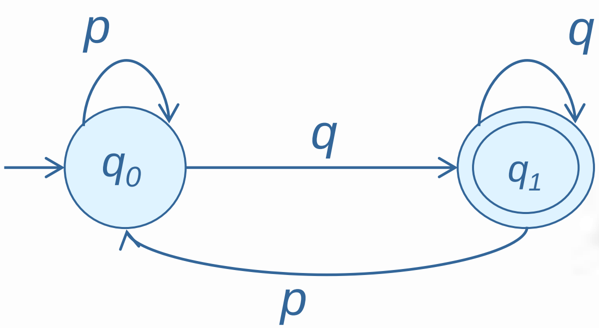
- **G(***p***U***q***)**



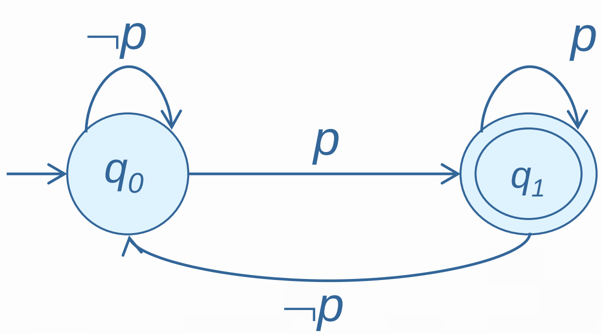
Quelques automates de Büchi



- **G F** p



- **G(**p**U**q**)**



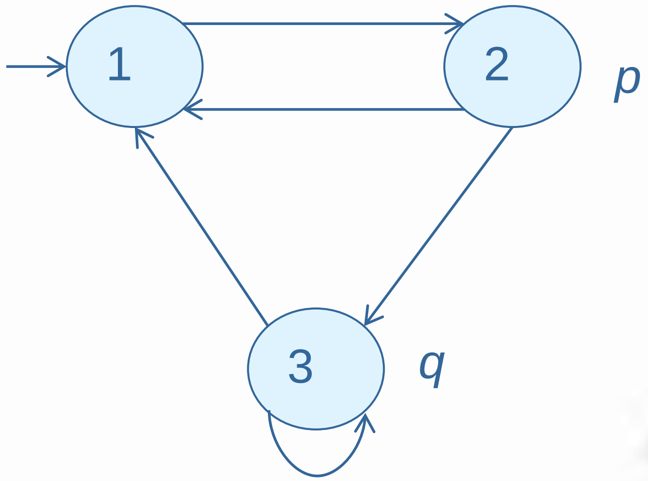
- **G F** p

Structure de Kripke et automate de Büchi

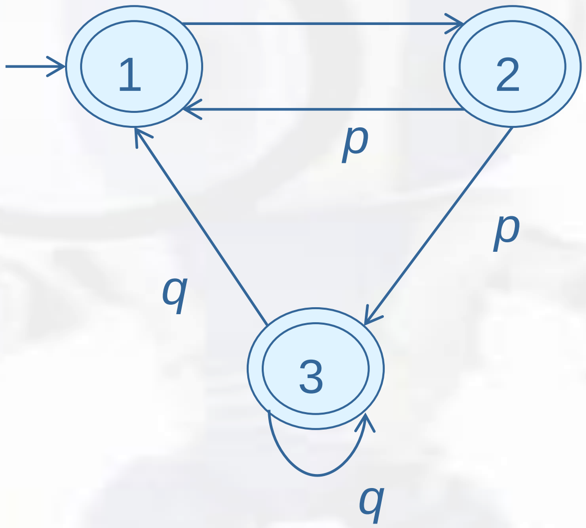
- Une structure de Kripke peut être vue comme un simple automate de Büchi.
- K=(S,s_0,$Prop$,T,l) correspond à l'automate de Büchi B=(Q, Σ, q_0, δ, F) avec :
  - $\Sigma=2^{Prop}$
  - Q=S
  - $q_0 = s_0$
  - (s,a,s')$\in$ δ ssi (s,s') $\in$T et a $\in$ l(s)
  - F=S

Exemple

Structure
de Kripke K



Automate
de Büchi B
généralisé



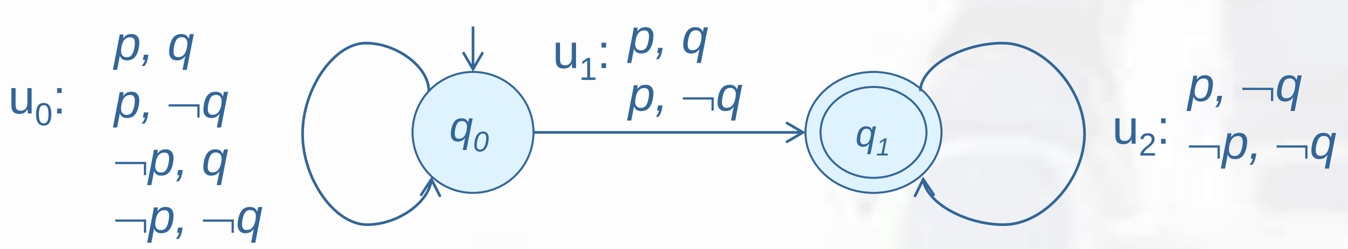
- $\Sigma=2^{\{p,q\}} =\{\emptyset,\{p\},\{q\},\{p,q\}\}$.

Model Checking LTL : exemple

- Soit la formule ϕ :
  - « *une occurrence de p est toujours suivie plus loin d'une occurrence de q tout au long de l'exécution* ».
- ϕ peut s'écrire en LTL :
  - $\mathbf{G}(p \Rightarrow \mathbf{XF}q)$.
- $\neg\phi$ signifie :
  - « *il existe une occurrence de p après laquelle on ne rencontrera plus jamais q* ».

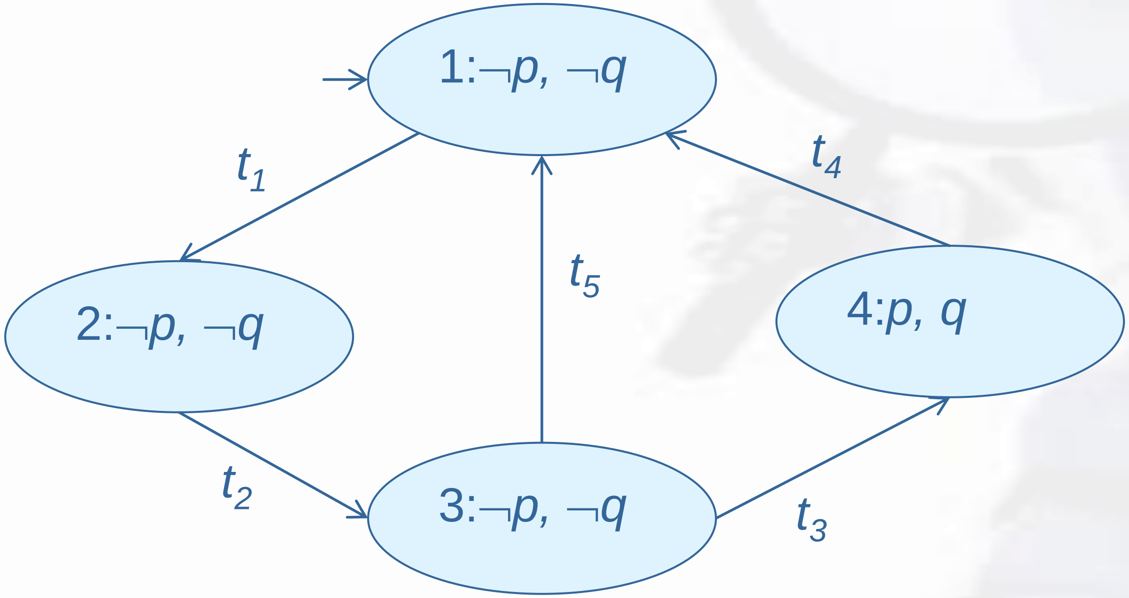
Model Checking LTL : exemple

- Soit la formule ϕ :
  - *« une occurrence de p est toujours suivie plus loin d'une occurrence de q tout au long de l'exécution ».*
- ϕ peut s'écrire en LTL :
  - $\mathbf{G}(p \Rightarrow \mathbf{XF}q)$.
- $\neg\phi$ signifie :
  - *« il existe une occurrence de p après laquelle on ne rencontrera plus jamais q ».*

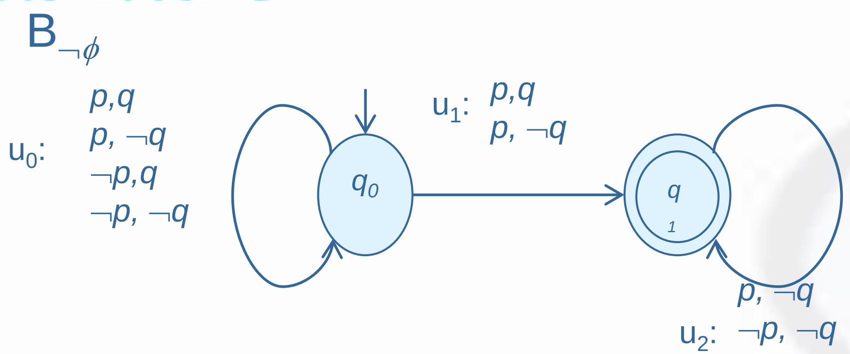


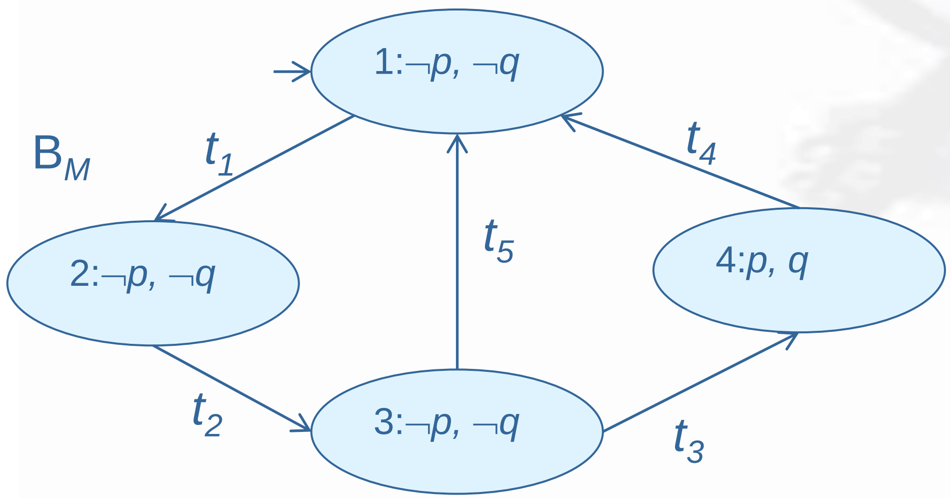
Model Checking LTL : exemple

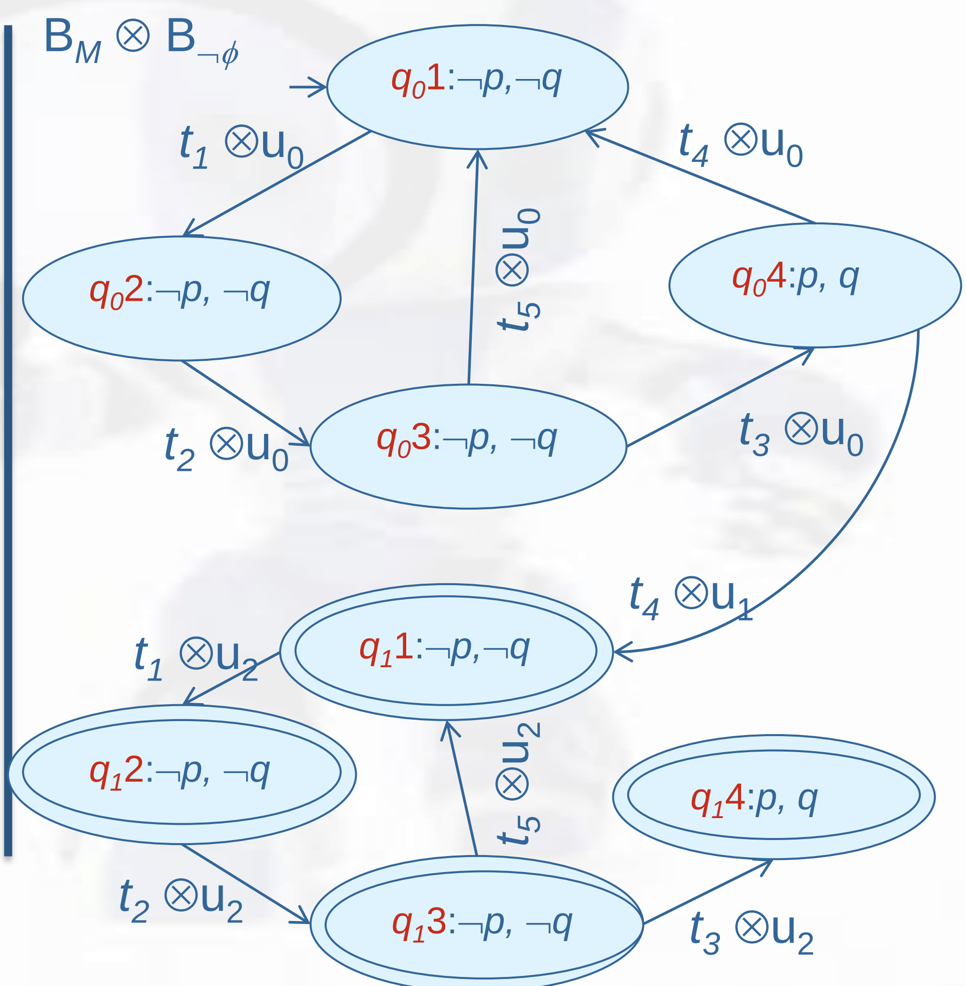
- Soit le modèle B_M



Model Checking LTL : exemple





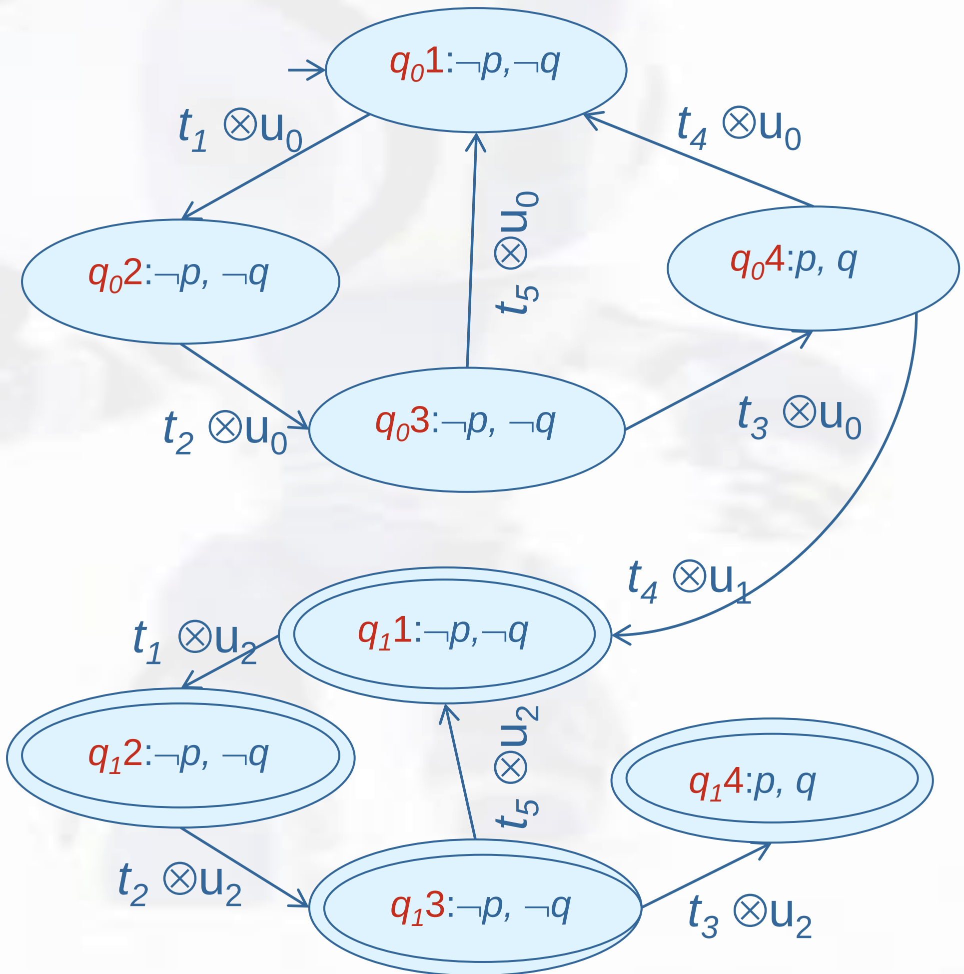


-Une transition $t \otimes u_1$ n'est possible que si t part d'un état vérifiant p

-Une transition $t \otimes u_2$ n'est possible que si t part d'un état vérifiant $\neg q$.

Model Checking LTL : exemple

- $L(B_M \otimes B_{\neg\phi}) \neq \emptyset$.
- Il existe une exécution qui visite les états acceptants une infinité de fois.
- $M \nvDash \phi$.



Complexité de LTL

- La complexité de la construction de l'automate intersection (dans les pires cas) :
  - $B_{\neg\phi}$ est de taille $O(2^{|\phi|})$.
  - $B_M \otimes B_{\neg\phi}$ est de taille $O(|B_M| \times |B_{\neg\phi}|)$.
  - Le model checking $M \models \phi$ peut être résolu en temps $O(|B_M| \times 2^{|\phi|})$.

LTL à Automate de Büchi (AB)

- Il existe plusieurs algorithmes pouvant être utilisés pour convertir une formule en un automate de Büchi.
- Dans le cadre de ce cours, nous utilisons une méthode simplifiée.

LTL à AB

- Forme normale négative (négation uniquement sur les propositions atomiques) :
  - $\neg(\varphi \mathbf{U} \psi) = \neg\varphi \mathbf{R} \neg\psi$
  - $\neg(\varphi \mathbf{R} \psi) = \neg\varphi \mathbf{U} \neg\psi$
  - $\neg \mathbf{G}(\varphi) = \mathbf{F}(\neg\varphi)$
  - $\neg \mathbf{F}(\varphi) = \mathbf{G}(\neg\varphi)$

<u>Rappel</u> $\varphi R \psi$ signifie ψ est vrai jusqu'à φ. ψ est vrai aussi dans l'état où φ est vrai.

$\varphi R \psi \equiv \neg(\neg\varphi U \neg\psi)$

LTL à AB

- Règles d'expansion
  - $\varphi \mathbf{U} \psi = \psi \vee (\varphi \wedge \mathbf{X}(\varphi \mathbf{U} \psi))$
  - $\varphi \mathbf{R} \psi = \psi \wedge (\varphi \vee \mathbf{X}(\varphi \mathbf{R} \psi))$
  - $\mathbf{G} \psi = \psi \wedge \mathbf{X}(\mathbf{G} \psi)$
  - $\mathbf{F} \psi = \psi \vee X(\mathbf{F} \psi)$
- On a aussi :
  - $(\mathbf{X} \varphi) \mathbf{U} (\mathbf{X} \psi) \equiv \mathbf{X} (\varphi \mathbf{U} \psi)$
  - $(\mathbf{X} \varphi) \wedge (\mathbf{X} \psi) \equiv \mathbf{X} (\varphi \wedge \psi)$
  - $\mathbf{GF} \varphi \wedge \mathbf{GF} \psi \equiv \mathbf{GF} (\varphi \wedge \psi)$

LTL à AB (algorithme simplifié)

1. L'état initial s est étiqueté par vrai$\wedge$Xϕ.
2. Développer ϕ jusqu'à obtenir une formule composée de sous-formules commençant par X, PAs (propositions atomiques ou une négation de PA) et des connecteurs booléens.
3. Réécrire sous forme normale disjonctive (disjonction de conjonctions), chaque disjonction est composée de deux sortes de formules :
   a. Formules atomiques (vraies ou fausses dans l'état courant).
   b. Formules portants sur le prochain état (**X**ψ) qui doivent être vraies au prochain état.
4. Créer un état s' pour chacune des disjonctions.
5. Ajouter une transition de s vers tout état s' pour chaque symbole t dans 2^{AP} vérifiant les formules de a) et une transition T (true) si **X**ψ = T.
6. Répéter les étapes 2 à 5 pour toute formule ψ de s'.
7. Ajouter les états d'acceptation B_i : tout état B_i ne doit pas être étiqueté par **X**ϕ_i tel que ϕ_i contient une **U**-sous-formule (une **U**-formule est une formule où **U** est l'opérateur racine de l'arbre syntaxique de la formule : exécuté en dernier).

Exemple

$\varphi = \mathbf{G}(a \Rightarrow \mathbf{F}b)$: tout a est suivi plus tard par un b.

1. $\rightarrow$ T,**X** φ
2. $\varphi \;= \mathbf{G}(a \Rightarrow \mathbf{F}b) = (a \Rightarrow \mathbf{F}b) \wedge \mathbf{X}\varphi$

   $= (\neg a \vee \mathbf{F}b) \wedge \mathbf{X}\varphi$

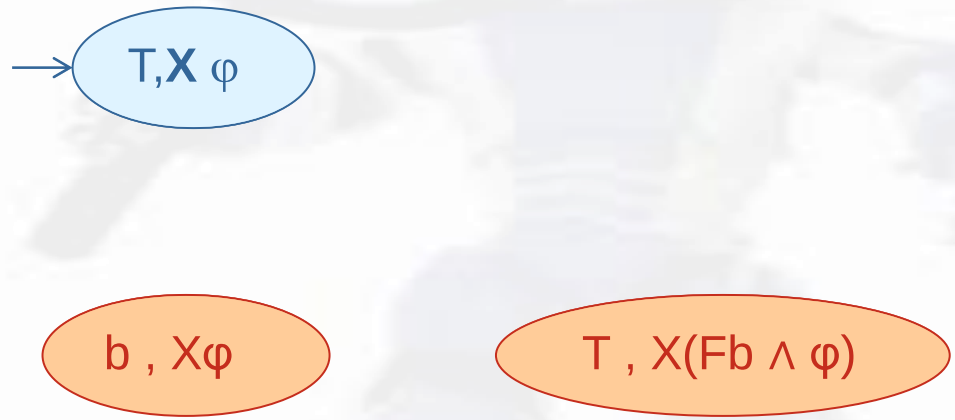
   $= (\neg a \vee b \vee XFb) \wedge X\varphi$
3. $\varphi \;= (\neg a \wedge X\varphi) \vee (b \wedge X\varphi) \vee (XFb \wedge X\varphi)$

Exemple

$\varphi = (\neg a \wedge X\varphi) \vee (b \wedge X\varphi) \vee (XFb \wedge X\varphi)$

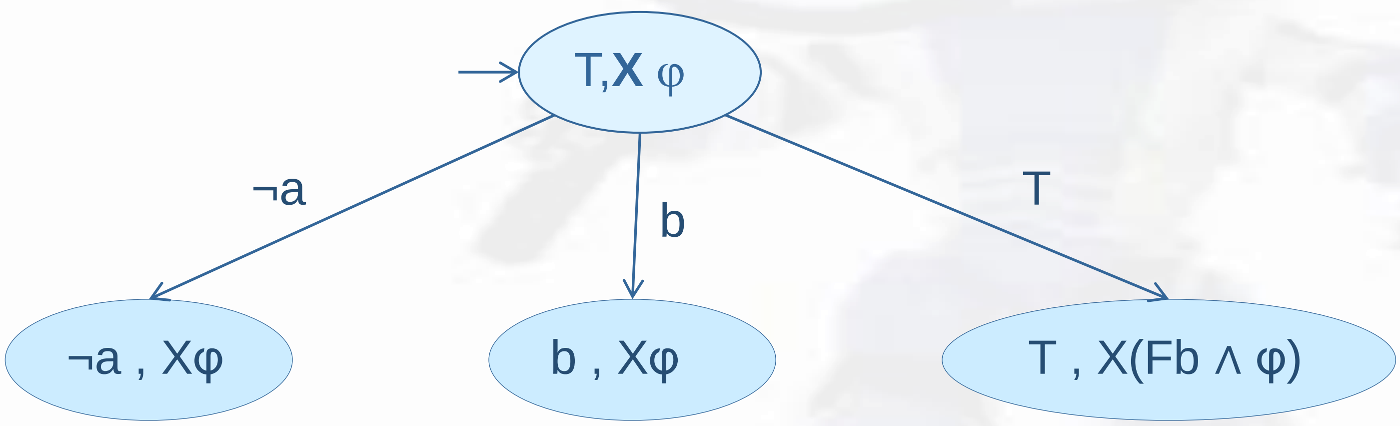
$\varphi = (\neg a \wedge X\varphi) \vee (b \wedge X\varphi) \vee (True \wedge X(Fb \wedge \varphi))$

4.



Exemple

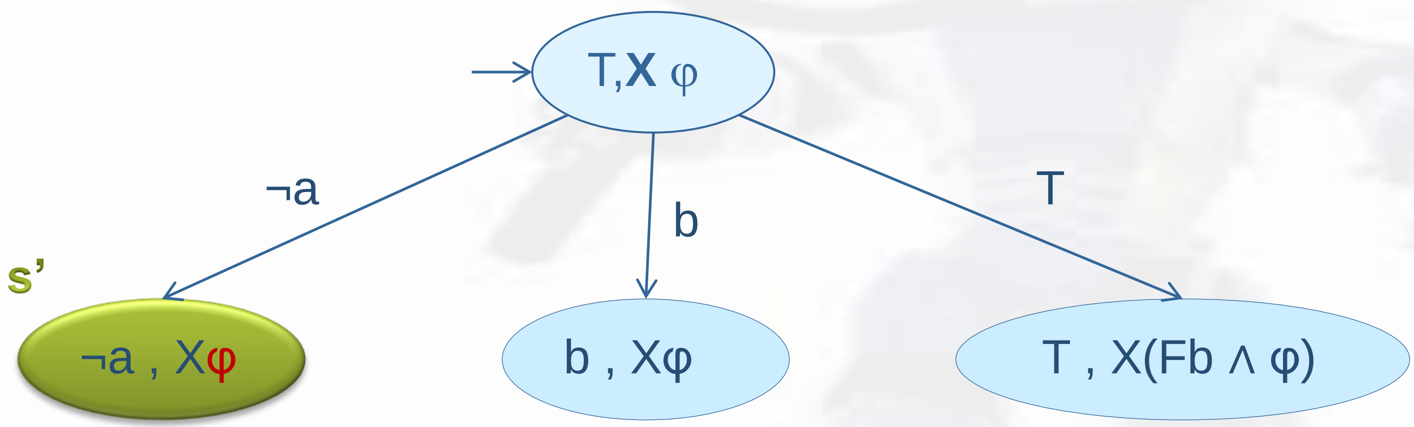
5.



Exemple

6. Itération sur s’

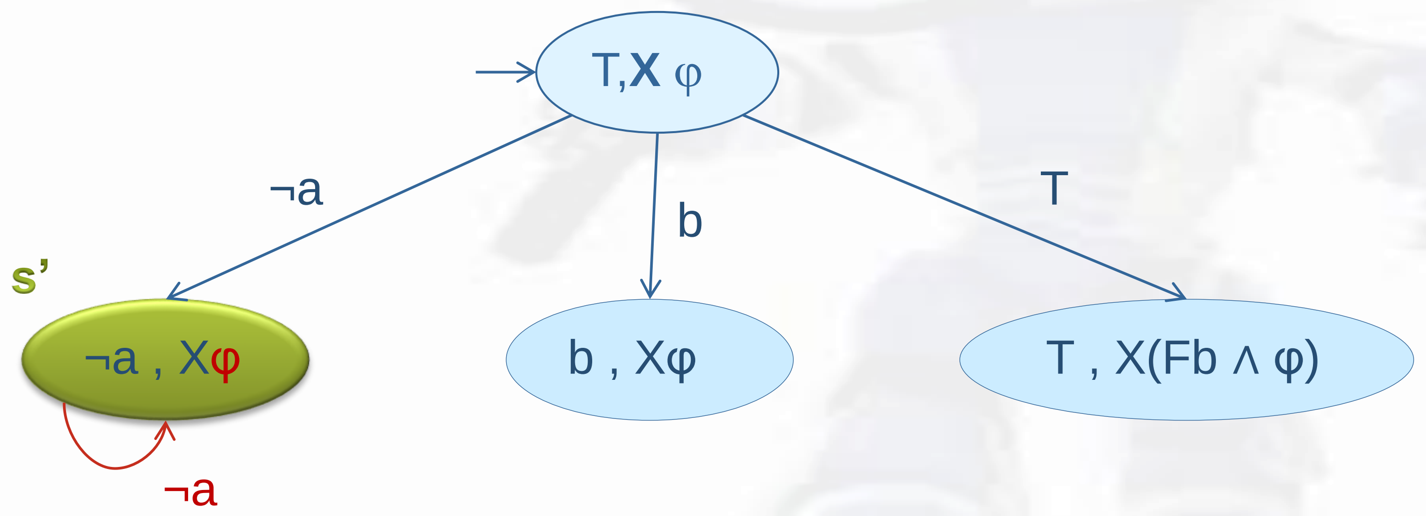
$\varphi = (\neg a \wedge X\varphi) \vee (b \wedge X\varphi) \vee (True \wedge X(Fb \wedge \varphi))$



Exemple

6. Itération sur s'

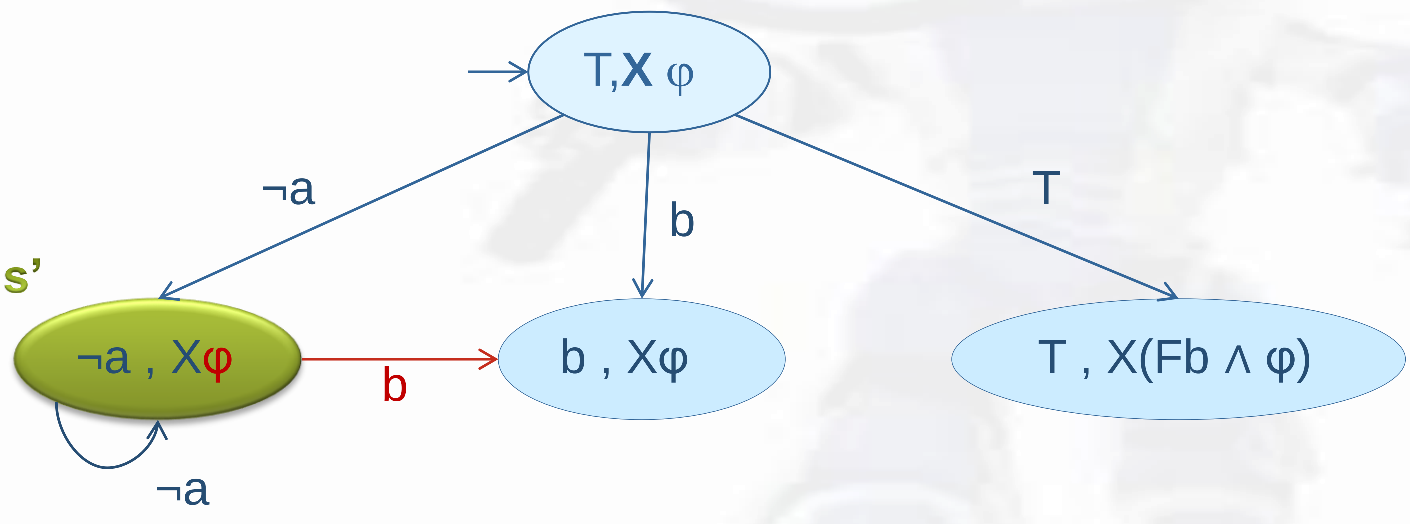
$\varphi = (\neg a \wedge X\varphi) \vee (b \wedge X\varphi) \vee (True \wedge X(Fb \wedge \varphi))$



Exemple

6. Itération sur s’

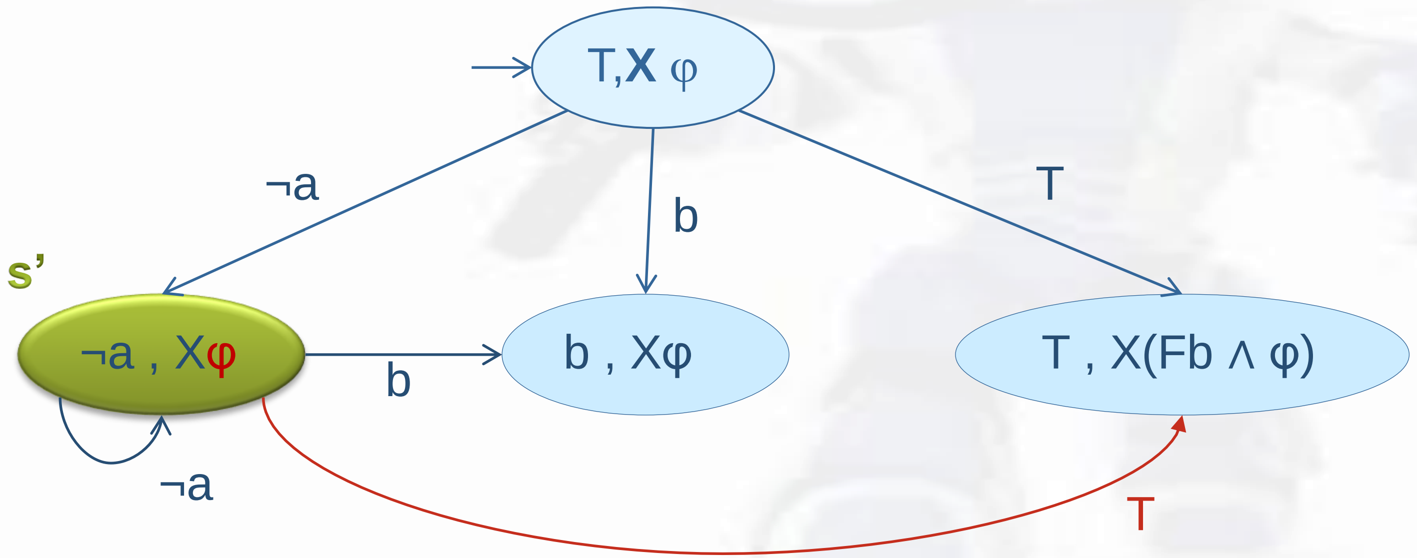
$\varphi = (\neg a \wedge X\varphi) \vee (b \wedge X\varphi) \vee (True \wedge X(Fb \wedge \varphi))$



Exemple

6. Itération sur s'

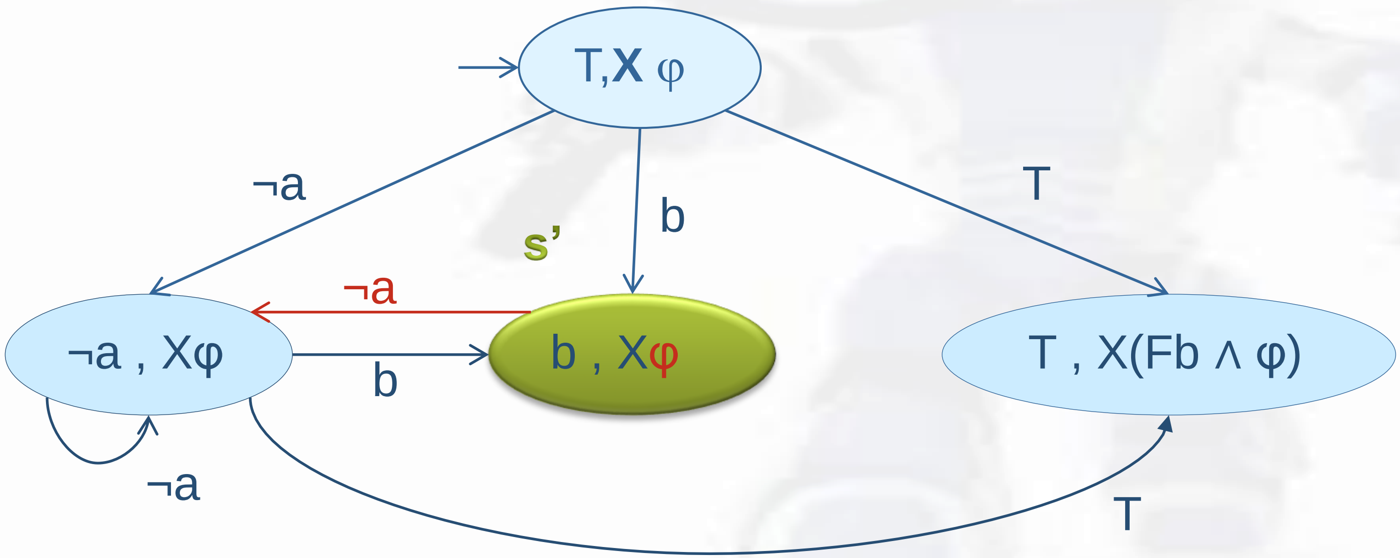
$$\varphi = (\neg a \wedge X\varphi) \vee (b \wedge X\varphi) \vee (True \wedge X(Fb \wedge \varphi))$$



Exemple

6. Itération sur s'

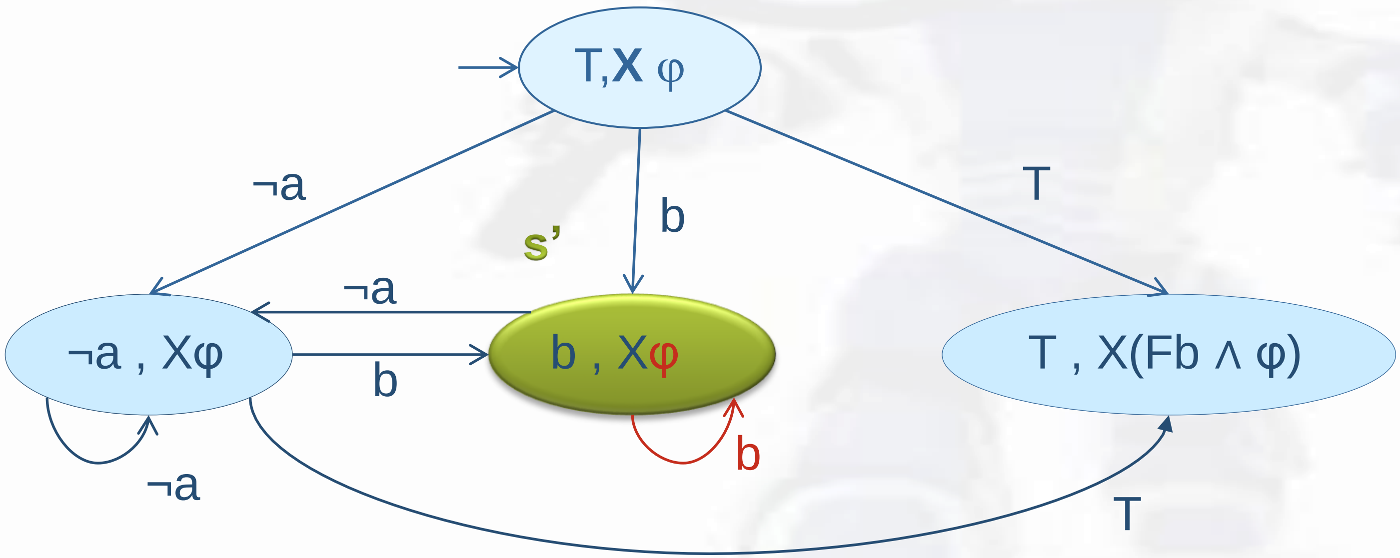
$\varphi = (\neg a \wedge X\varphi) \vee (b \wedge X\varphi) \vee (True \wedge X(Fb \wedge \varphi))$



Exemple

6. Itération sur s’

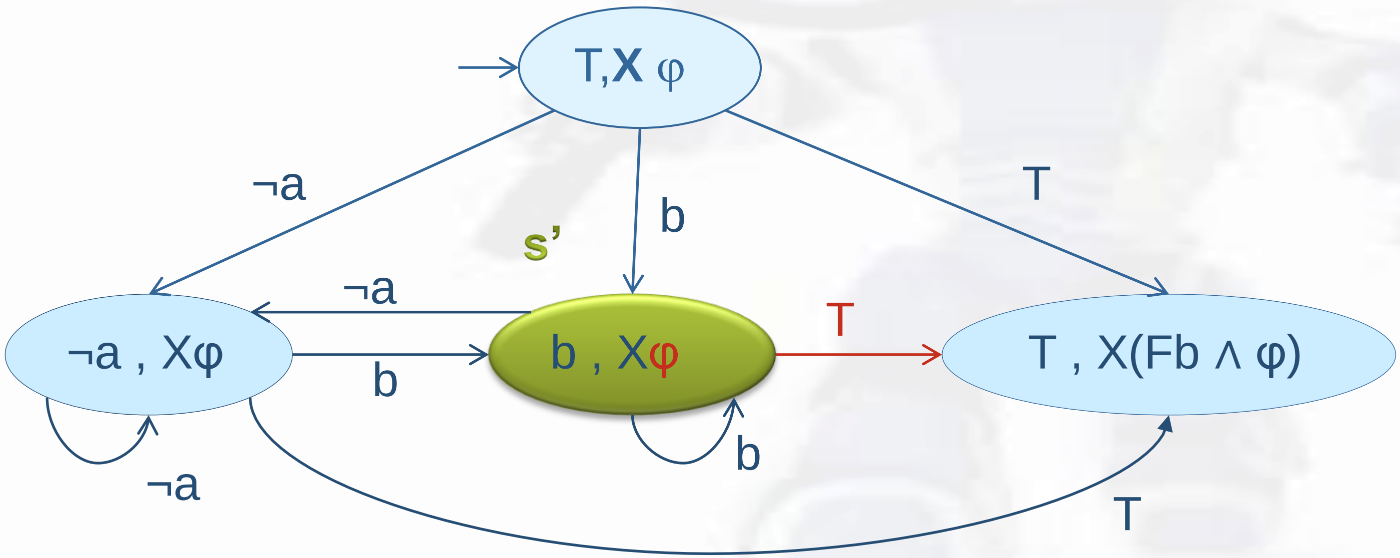
$\varphi = (\neg a \wedge X\varphi) \vee (b \wedge X\varphi) \vee (True \wedge X(Fb \wedge \varphi))$



Exemple

6. Itération sur s’

$\varphi = (\neg a \wedge X\varphi) \vee (b \wedge X\varphi) \vee (True \wedge X(Fb \wedge \varphi))$

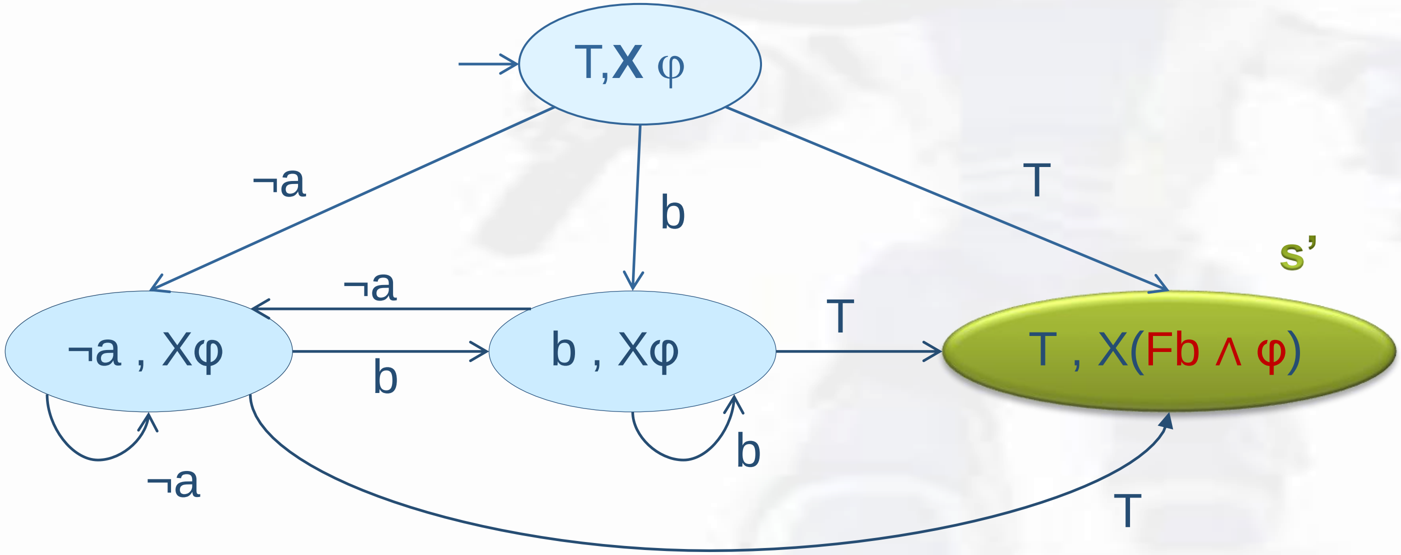


Exemple

6. Itération sur s'

ψ= Fb ∧ φ = Fb∧G(a ⇒ Fb) = Fb∧(a ⇒ Fb)∧ Xφ

= Fb∧Xφ = (b∨XFb)∧Xφ = (b∧Xφ) ∨ (X(Fb∧φ))

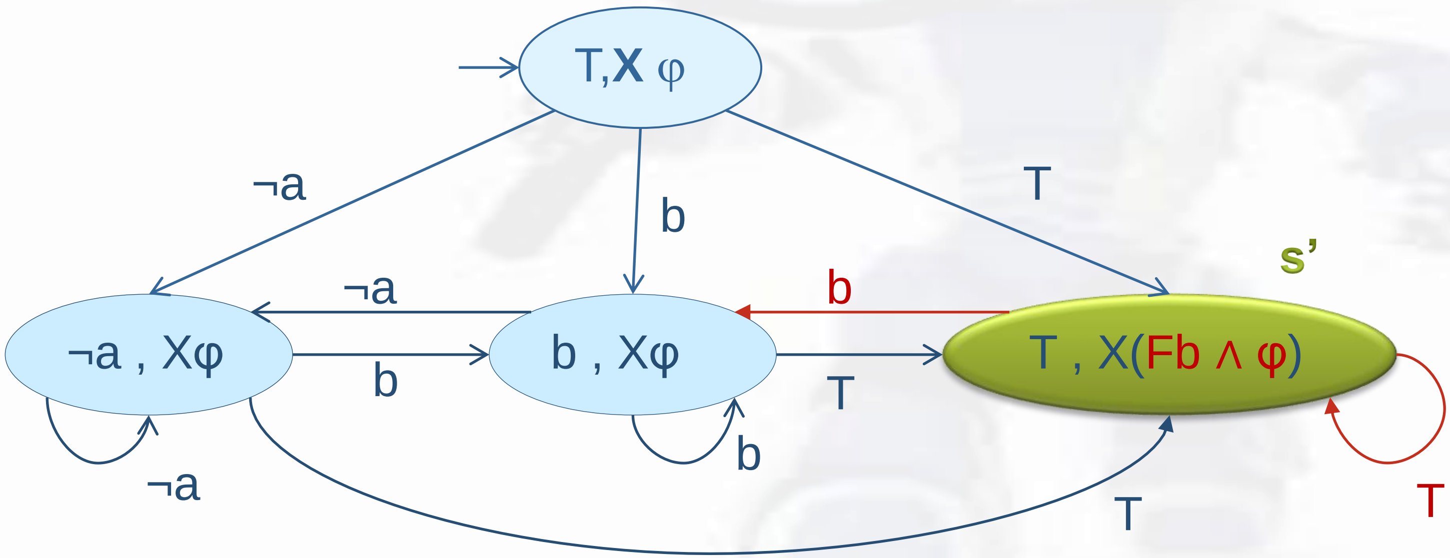


Exemple

6. Itération sur s'

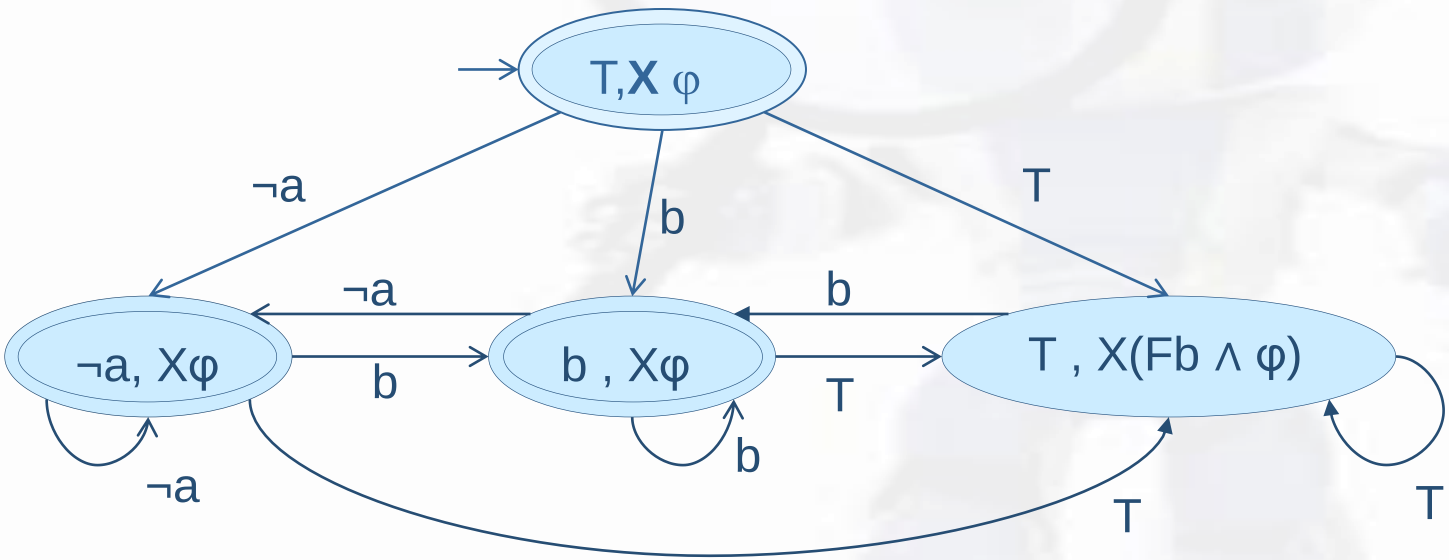
$\psi = Fb \wedge \varphi = Fb \wedge G(a \Rightarrow Fb) = Fb \wedge (a \Rightarrow Fb) \wedge X\varphi$

$= Fb \wedge X\varphi = (b \vee XFb) \wedge X\varphi = (b \wedge X\varphi) \vee (X(Fb \wedge \varphi))$

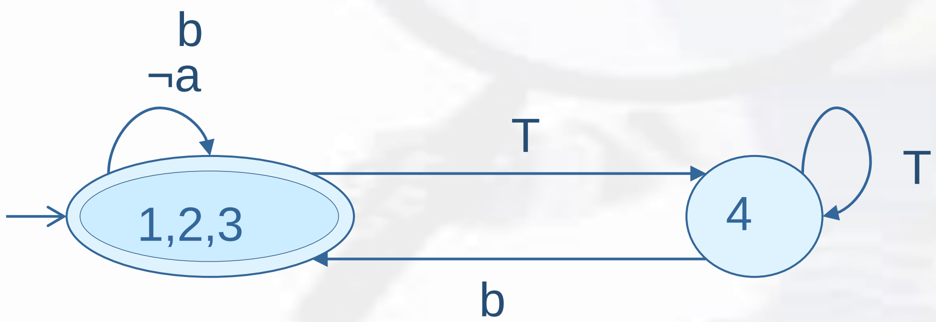


Exemple

7. Etats acceptants



Minimisation



MODEL CHECKING CTL

Introduction

- Soient M une structure de kripke et *p* une formule CTL.
- Il existe essentiellement deux techniques de model-checking CTL :
  - Technique de marquage : utilisation d'un algorithme d'étiquetage des états par les sous formules de *p* qu'ils satisfont.

Introduction

- Model-checking symbolique :
  - Technique de points fixes : calcul de l'ensemble caractéristique de *p* (l'ensemble des états du modèle M qui satisfont *p*).
  - Diagramme de décision binaire (BDD).

Model checking CTL par marquage des états

- Soient une structure de kripke M= (Q,q_0,E,T,$Prop$,l) et une formule CTL ϕ.
- L'algorithme est dû à Queille, Sifakis, Clarke, Emerson et Sistla, il consiste à :
  - pour chaque sous-formule ϕ' de ϕ, en commençant par la plus interne, marquer les états de M vérifiant ϕ' ($q.\phi'$) : $q \models \phi'$
  - procéder récursivement en réutilisant les marquages des sous-formules plus internes pour une sous-formule plus externe.
  - M satisfait ϕ ssi son état initial q_0 est marqué par ϕ ($q_0.\phi$).

Schéma global de l'algorithme de marquage

- Entrées : ϕ, M= ($Q, q_0, E, T, Prop, l$)
- Sortie : ensemble des états satisfaisant ϕ.
  - ϕ':=Normaliser(ϕ) (l'écrire en terme de AU, EU, EX, $\wedge$,$\neg$ et T(vrai)).
  - Marquage(ϕ',M).
  - Retourner $q_0.\phi'$.

Normalisation

- Utiliser les règle suivantes pour normaliser les formules :
  - $F = \quad T$
  - $\mathbf{AX}\phi = \quad \mathbf{EX}\neg\phi$
  - $\mathbf{AF}\phi = \mathrm{T}\mathbf{AU}\phi$
  - $\mathbf{AG}\phi = \quad \mathbf{EF}\neg\phi$
  - $\mathbf{EF}\phi = \mathrm{T}\mathbf{EU}\phi$
  - $\mathbf{EG}\phi = \quad \mathbf{AF}\neg\phi$

Algorithme *marquage*

- Entrées : formule CTL ϕ, M= (Q,q_0,E,T,$Prop$,l)
- Cas 1 : ϕ=p

  pour tout q$\in$Q faire

  si p $\in$l(q) alors q.ϕ:=vrai

  sinon q.ϕ:=faux fin si

  fin pour tout
- Cas 2 : $\phi=\neg\psi$

  faire *marquage*(ψ,M) ;

  pour tout q$\in$Q faire

  q.ϕ:=not(q.ψ) ;

  fin pour tout

Algorithme *marquage*

- Entrées : formule CTL ϕ, M= $(Q,q_0,E,T,Prop,l)$
- Cas 3 : $\phi= \psi_1 \wedge \psi_2$

  faire *marquage*(ψ_1,M) ; *marquage*(ψ_2,M) ;

  pour tout $q \in Q$ faire

  $q.\phi$:=et($q.\psi_1$,$q.\psi_2$)

  fin pour tout
- Cas 4 : ϕ=**EX** ψ

  faire *marquage*(ψ,M) ;

  pour tout $q \in Q$ faire $q.\phi$:=faux fin pour tout

  pour tout $(q,q') \in T$ faire

  si $q'.\psi$=vrai alors $q.\phi$:= vrai

  fin pour tout

Algorithme *marquage*

- Entrées : formule CTL ϕ, M= (Q,q_0,E,T,$Prop$,l)
- Cas 5 : ϕ= ψ_1**EU**ψ_2

faire *marquage*(ψ_1,M) ; *marquage*(ψ_2,M) ;
pour tout q$\in$Q faire
    q.ϕ:=faux;
    q.dejavu:=faux;
fin pour tout
L:=$\emptyset$
pour tout q$\in$Q faire si q.ψ_2=vrai alors L:=L$\cup${q} fin si fin pour tout
tant que L$\neq\emptyset$ faire
    prendre un q $\in$ L;
    L:=L\{q};
    q.ϕ:=vrai;
    pour tout (q',q) $\in$Tfaire
        si q'.dejavu=faux alors
            q'.dejavu := vrai;
            si q'.ψ_1=vrai alors L:=L$\cup${q'} finsi
        fin si
    fin pour tout
fin tant que

Algorithme *marquage*

- Entrées : formule CTL ϕ, M= $(Q,q_0,E,T,Prop,l)$
- Cas 6 : $\phi= \psi_1 \mathbf{AU} \psi_2$

faire *marquage*$(\psi_1$,M) ; *marquage*$(\psi_2$,M) ;

$L:=\varnothing$

pour tout $q \in Q$ faire

    $q.nb:=degre(q)$; $q.\phi:=faux$;

    si $q.\psi_2=vrai$ alors $L:=L\cup\{q\}$ fin si

fin pour tout

tant que $L\neq\varnothing$ faire

    prendre un $q \in L$;

    $L:=L\backslash\{q\}$;

    $q.\phi:=vrai$;

    pour tout $(q',q) \in T$faire

        $q'.nb:=q'.nb-1$

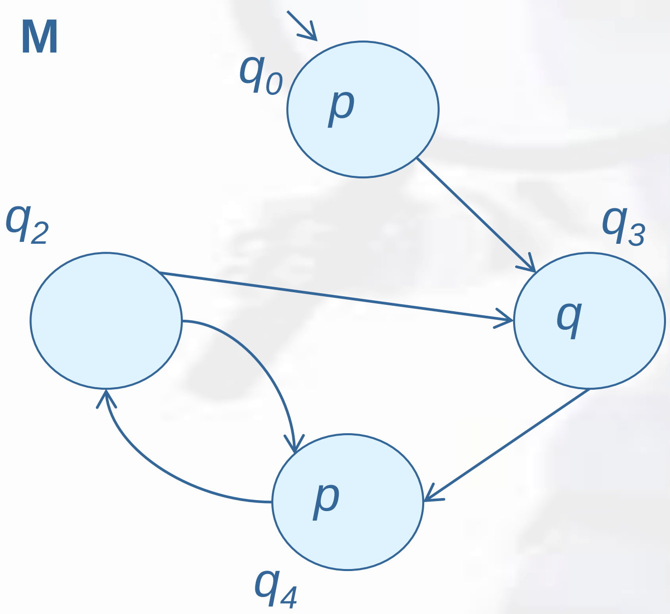
        si $(q'.nb=0)$ et $(q'.\psi_1=vrai)$ et $(q'.\phi=faux)$ alors $L:=L\cup\{q'\}$ fin si

    fin pour tout

fin tant que

Application

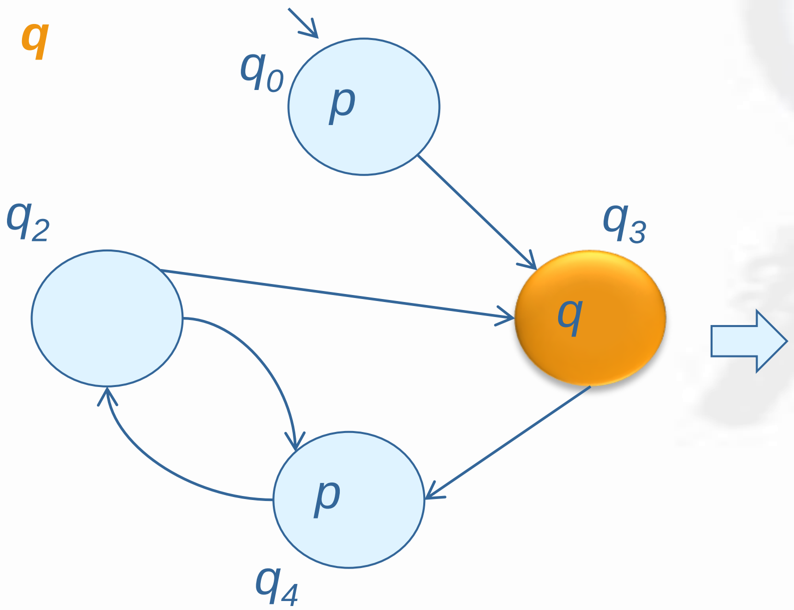
- Considérez le système M modélisé suivant et la formule $\phi = \mathbf{AG}(p \Rightarrow \mathbf{AF}q)$:



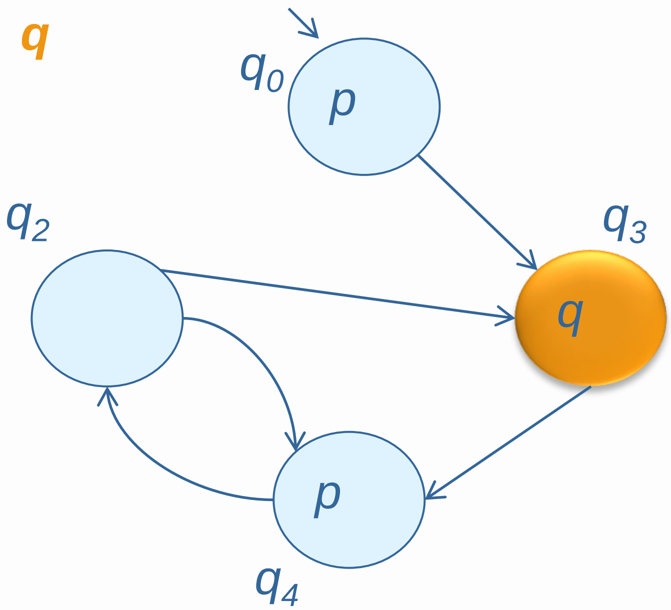
- Construire les états dans lesquels la formule ϕ est vérifiée.

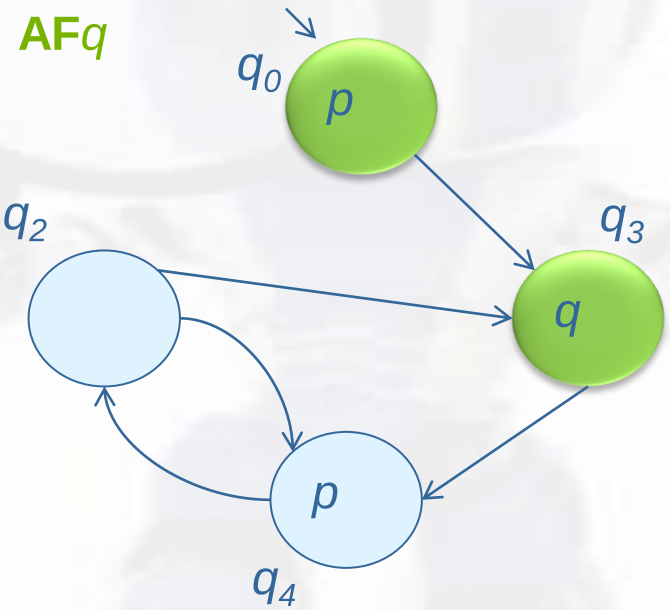
Application

q



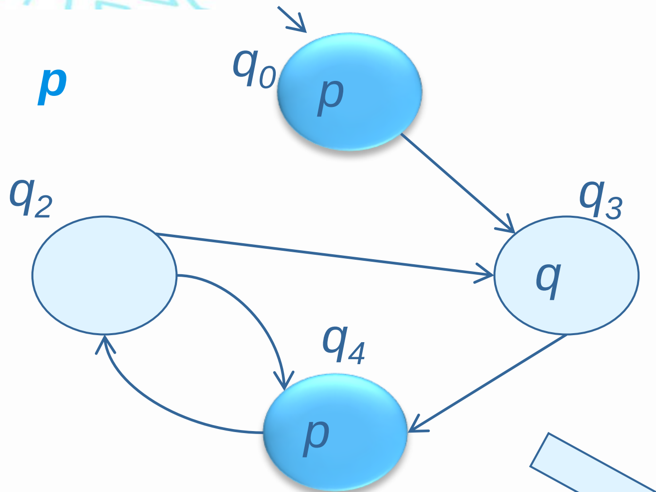
Application



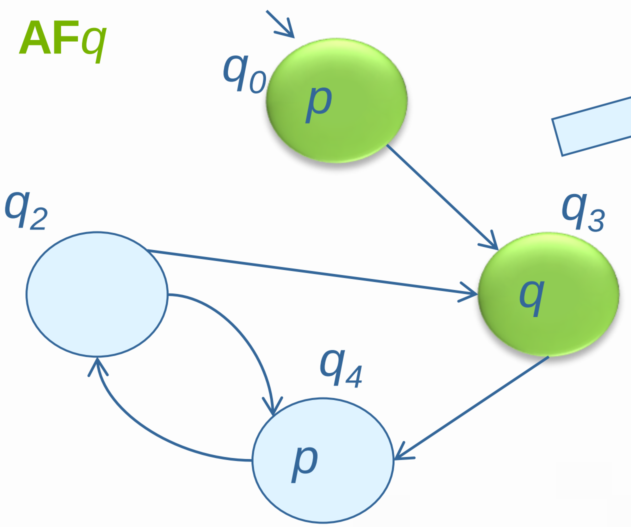


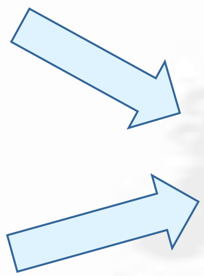
Application

p

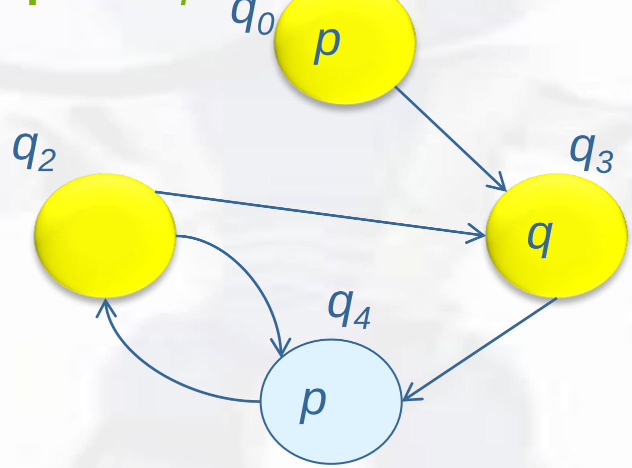


AF*q*

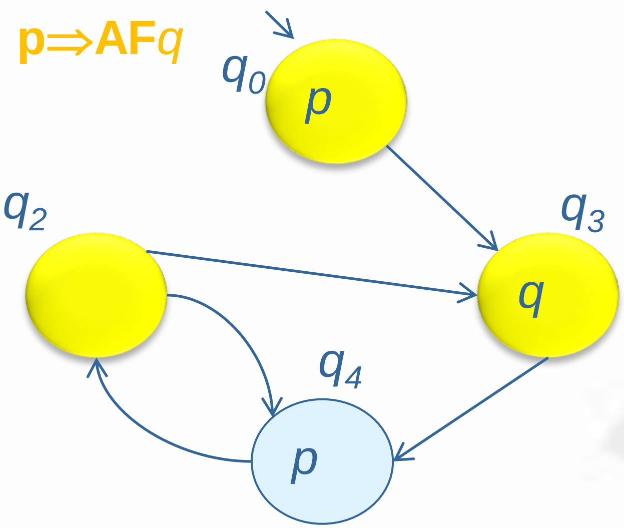


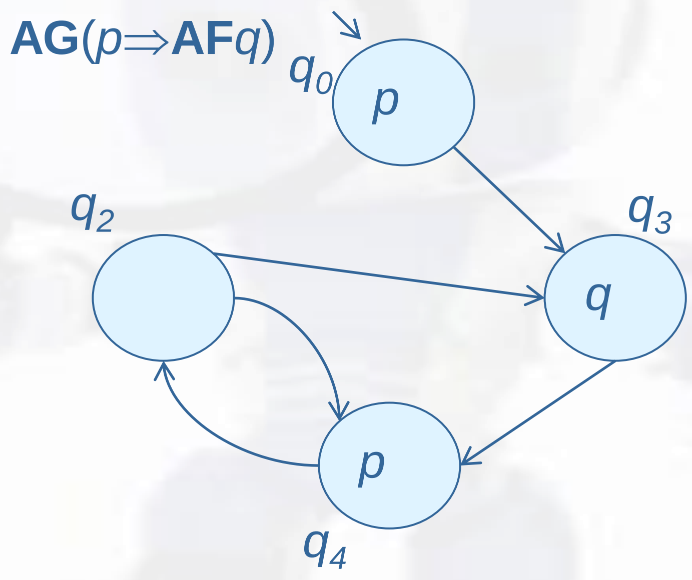


p⇒AF*q*



Application

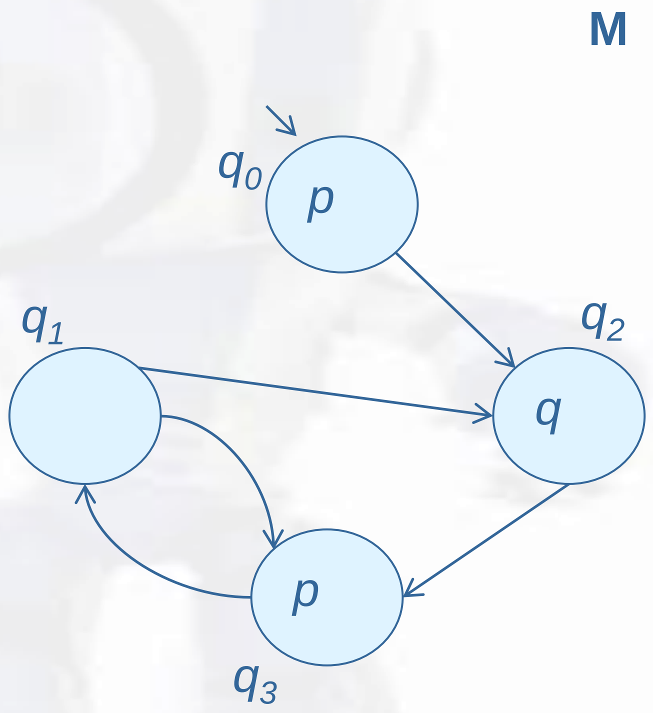




- Aucun état ne vérifie la formule : $M \not\models \phi$

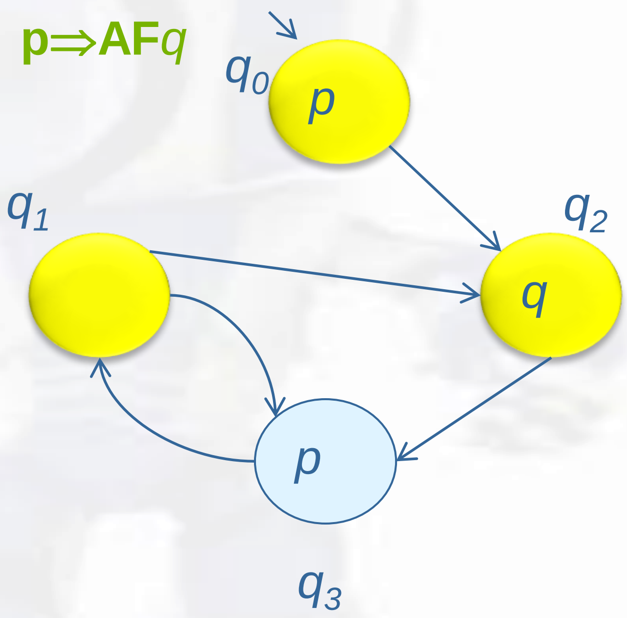
Algorithme de marquage pour l'application

- ϕ= **AG**($p \Rightarrow$ **AF**q)
- **AF**q = *T***AU**q
- $p \Rightarrow$ **AF**q = $\neg p \vee$ **AF**q = $\neg$(p$\wedge \neg$**AF**q)



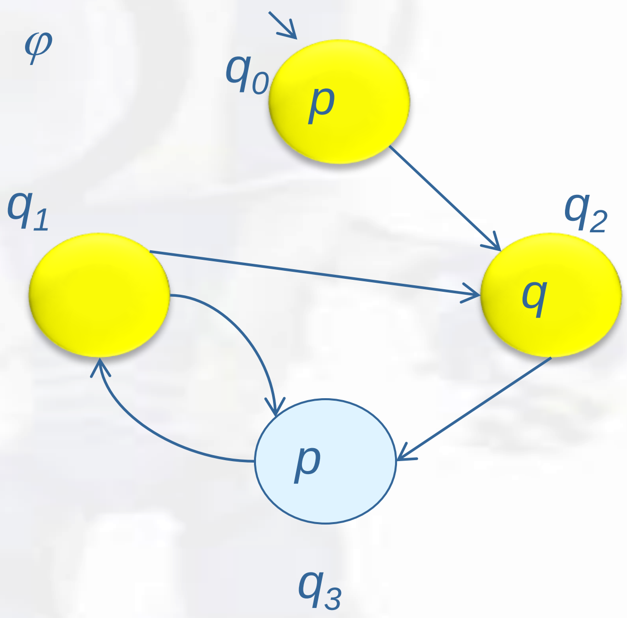
Algorithme de marquage pour l'application

- $\phi = \mathbf{AG}(p \Rightarrow \mathbf{AF}q)$
- $\mathbf{AF}q = T\mathbf{AU}q$
- $p \Rightarrow \mathbf{AF}q = \neg p \vee \mathbf{AF}q = \neg(p \wedge \neg \mathbf{AF}q)$
- Les états qui satisfont la formule $\varphi = \neg(p \wedge \neg \mathbf{AF}q)$ sont colorés en jaune.



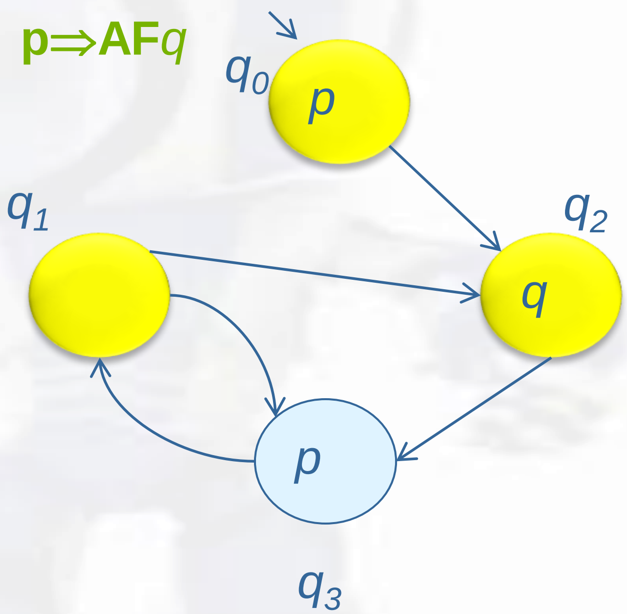
Algorithme de marquage pour l'application

- $\mathbf{AG}\varphi = \neg\mathbf{EF}\neg\varphi = \neg(T\mathbf{EU}\neg\varphi)$



Algorithme de marquage pour l'application

- ϕ= **AG**($p\Rightarrow$**AF**q)
- **AF**q = T**AU**q
- $p\Rightarrow$**AF**q = $\neg p\vee$ **AF**q = $\neg(p\wedge\neg$**AF**$q)$
- Les états qui satisfont la formule φ= $\neg(p\wedge\neg$**AF**$q)$ sont colorés en jaune.
- Trouver les états qui satisfont **AG** φ.



Complexité

- Le model checking CTL demande un temps linéaire en chacune des composants (automate et formule).
- Pourquoi est-il plus efficace que pour LTL:
  - CTL permet l'expression de formules d'états.
  - Il suffit donc de trouver quel état vérifie telle formule au lieu de considérer les exécutions.
- Le model checking d'une formule CTL par marquage peut se faire en $O(|M| \wedge |\phi|)$.

Conclusion

- Attention au problème d'*explosion du nombre d'états* (*state explosion problem*).
- Pour éviter ce problème il est possible de faire recours au model checking symbolique : représenter les états et les transitions du système de façon symbolique comme les diagrammes de décision binaire (BDD).
- Il est possible aussi d'appliquer des méthodes d'abstraction pour simplifier les modèles à vérifier.