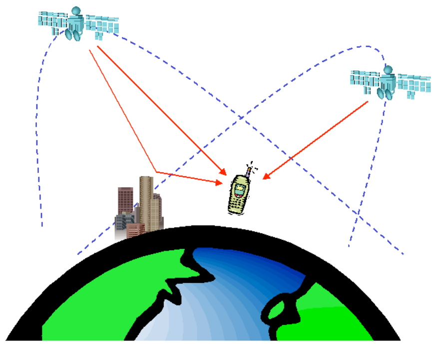
Vérification formelle de systèmes par Model-Checking

Nathalie Sznajder
Université Pierre et Marie Curie, LIP6

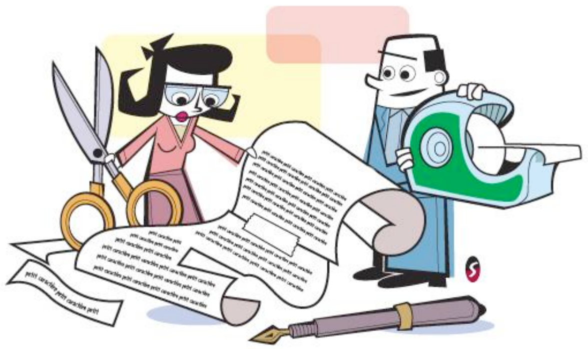
Les méthodes formelles

- Preuve assistée par ordinateur
- Test
- Model-Checking

Model-Checking

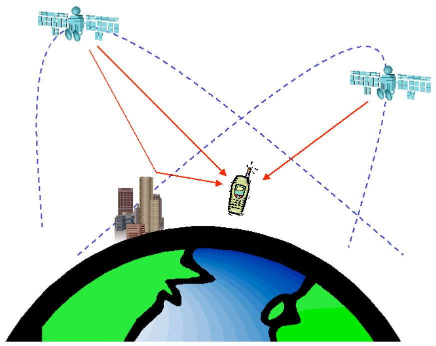
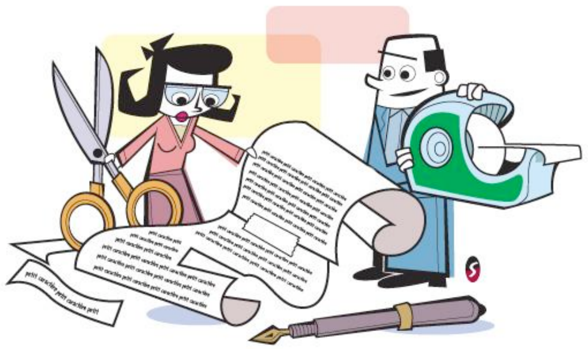


système



spécification

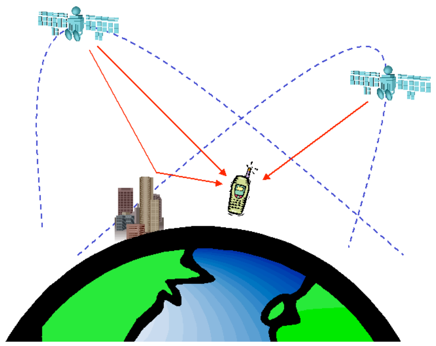
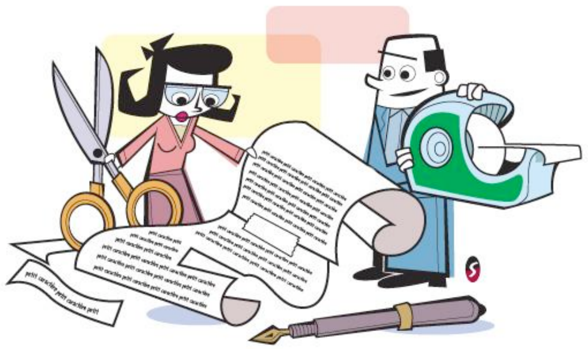
Model-Checking

Est-ce que le  satisfait  ?

système

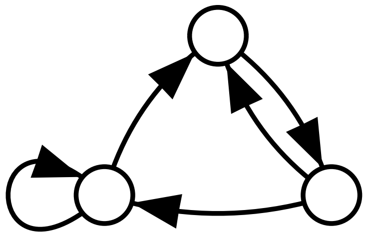
spécification

Model-Checking

Est-ce que le  satisfait  ?

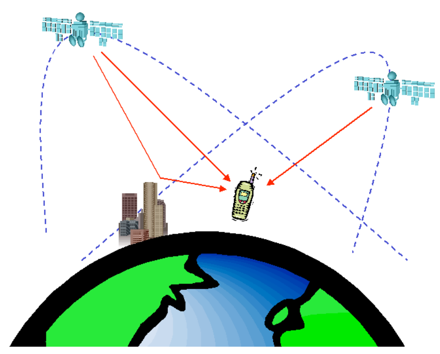
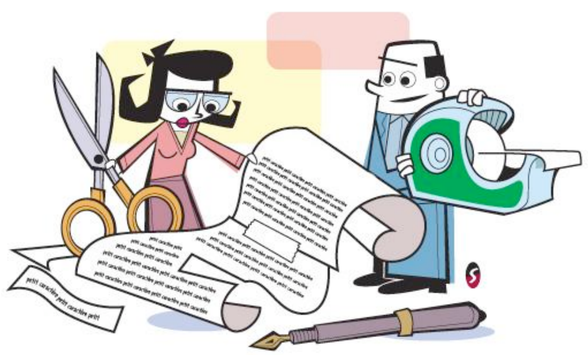
système

spécification



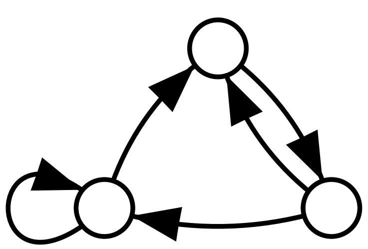
modèle

Model-Checking

Est-ce que le  satisfait  ?

système

spécification

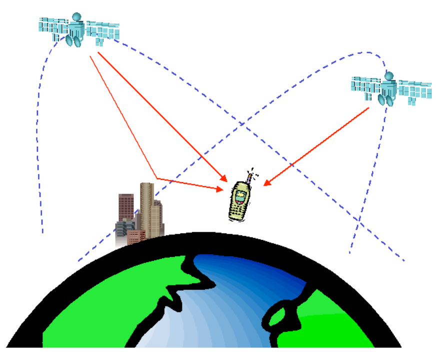
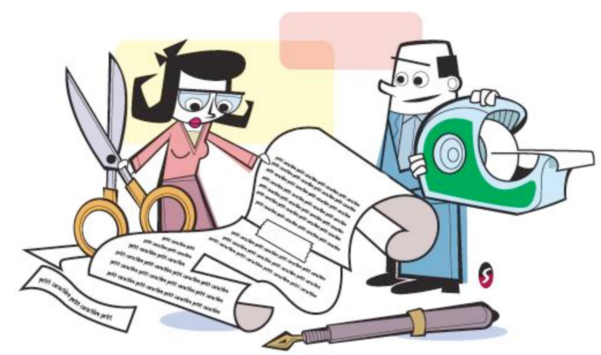


φ

modèle

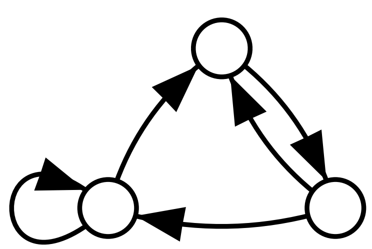
formule

Model-Checking

Est-ce que le  satisfait  ?

système

spécification

 $\models$ φ ?

modèle

algorithme de Model Checking

formule

Références bibliographiques

- *Model Checking*, E. Clarke, O. Grumberg, D. Peled, MIT Press 99
- *Vérification de logiciels : techniques et outils du model-checking*, P. Schnoebelen, B. Bérard, M. Bidoit, F. Laroussinie, A. Petit, Vuibert 99
- *Principles of Model-Checking*, C. Baier, J.-P. Katoen, MIT Press 08

Plan

1. Modélisation
2. Spécifications par logiques temporelles
3. Algorithmes de Model-Checking
    3.1. LTL
    3.2. CTL
    3.3. Inclure des notions d'équité

I. Modélisation

- On veut vérifier comportement du système au cours du temps.
- Notion d'état à un instant donné
- Actions du système →changement d'état.
- →Système de transition
- Informations supplémentaires sur
  - communication (notion d'action)
  - propriétés vérifiées par les états (propositions atomiques)

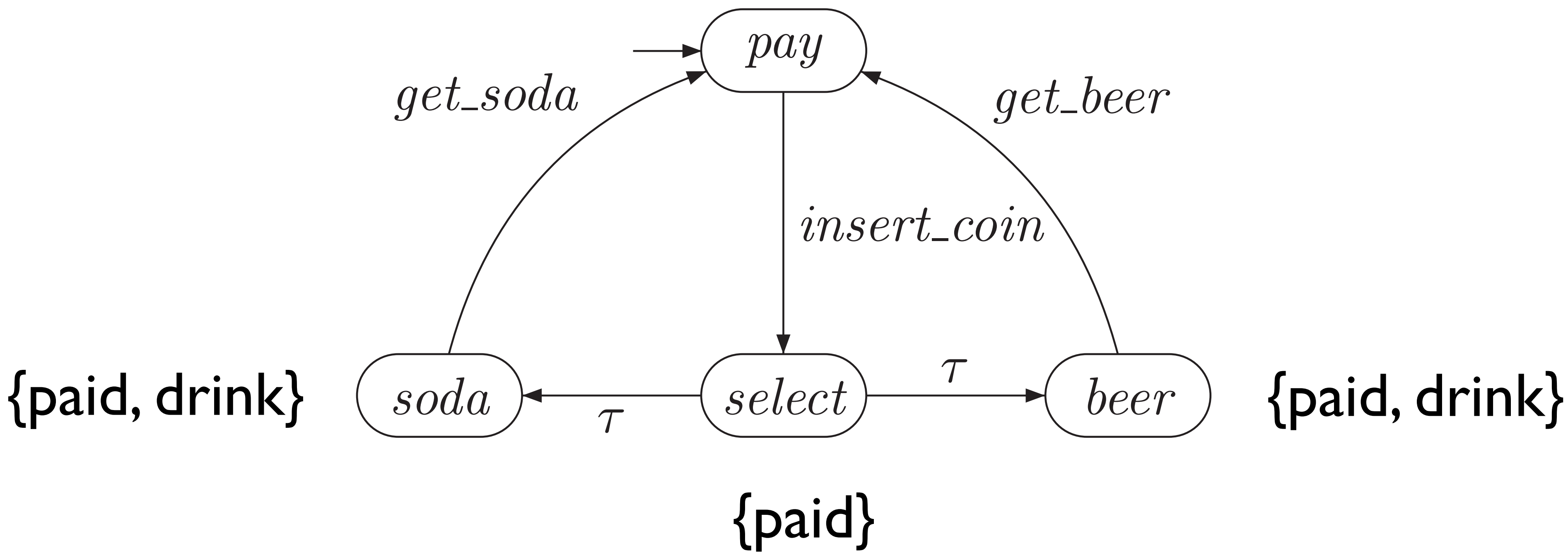
Structure de Kripke

- Définition: $M=(Q,T,A,q_0,AP,l)$
  - Q : ensemble fini d'états
  - A : alphabet d'actions (facultatif)
  - T : relation de transitions entre états
  - q_0 : état initial
  - AP : ensemble de propositions atomiques
  - $l : Q \rightarrow 2^{AP}$, étiquetage des états

Structure de Kripke

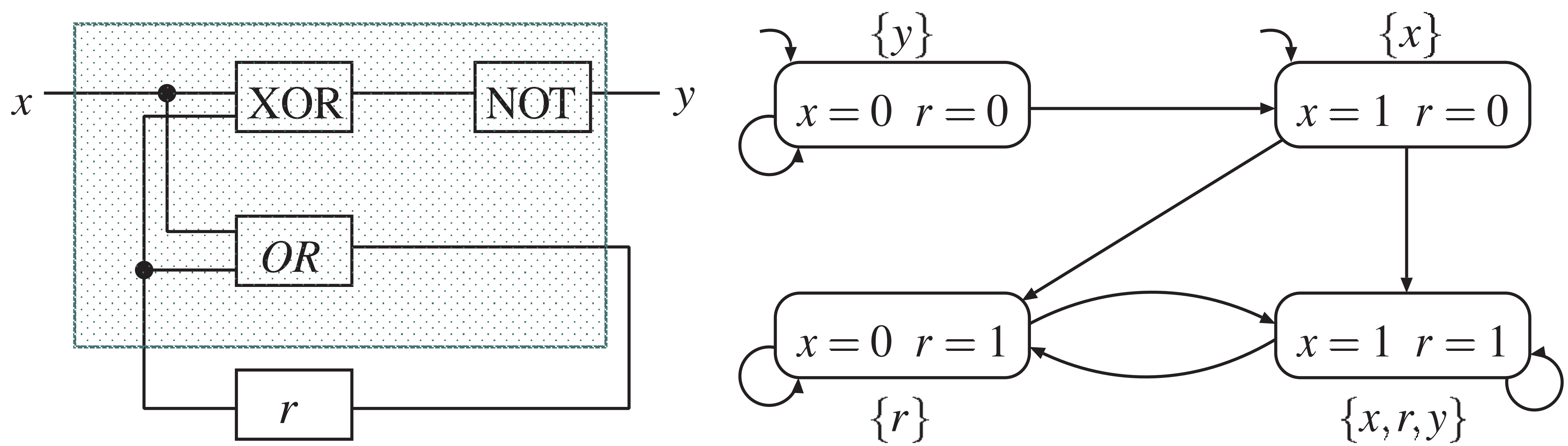
- Soit $M=(Q, T, A, q_0, AP, l)$ une structure de Kripke.
- Soit q un état. L'ensemble $\{q' \in Q \mid$ il existe $a \in A, (q,a,q') \in T\}$ est l'ensemble des successeurs de q.

Exemple: distributeur



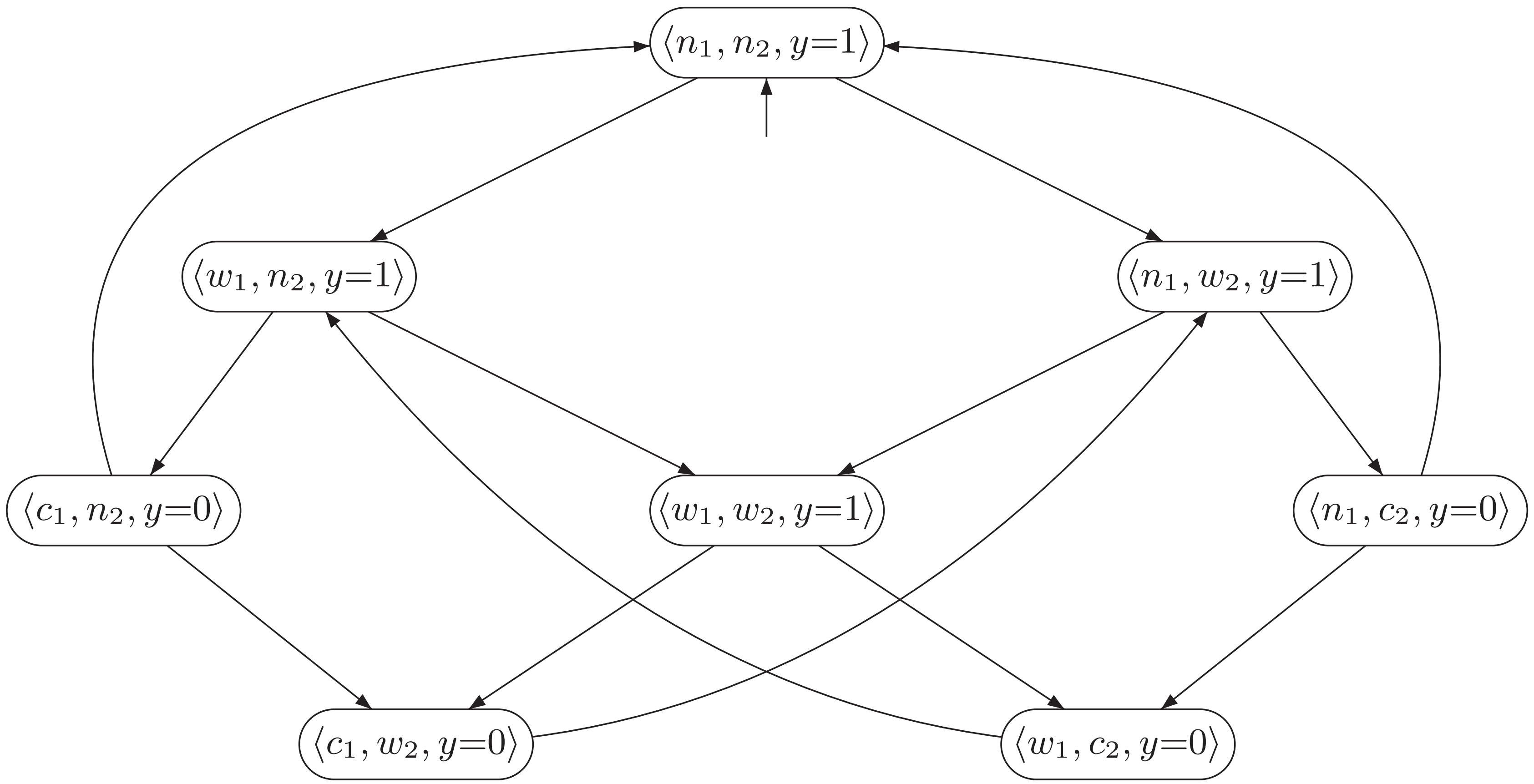
[Principles of Model-Checking,
C. Baier, J.-P. Katoen]

Exemple: circuit



[Principles of Model-Checking,
C. Baier, J.-P. Katoen]

Exemple: exclusion mutuelle



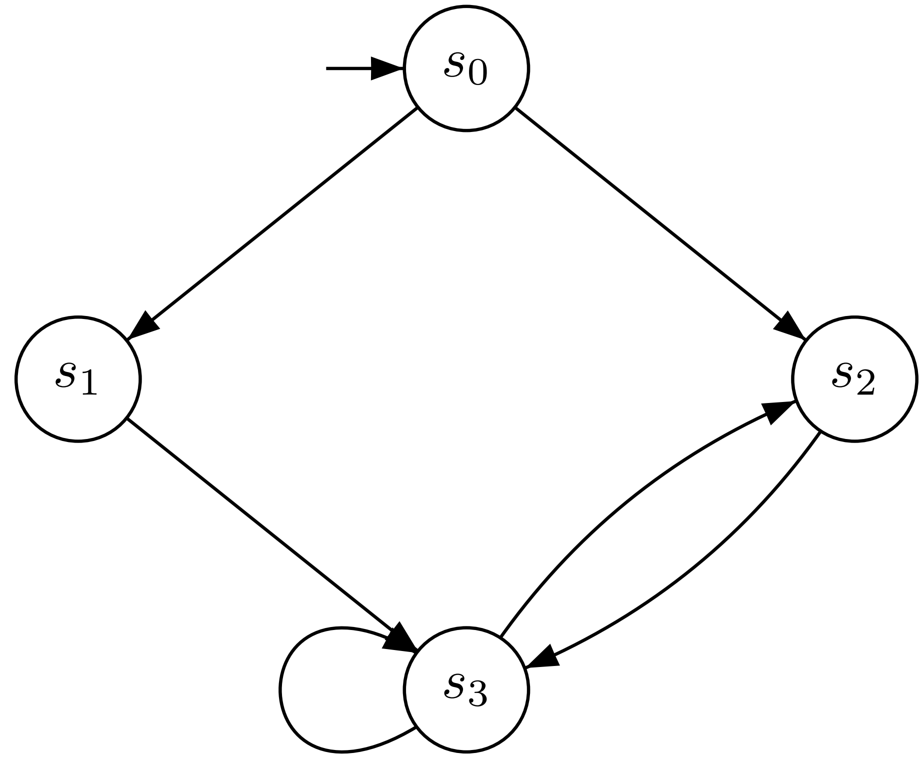
Structure de Kripke

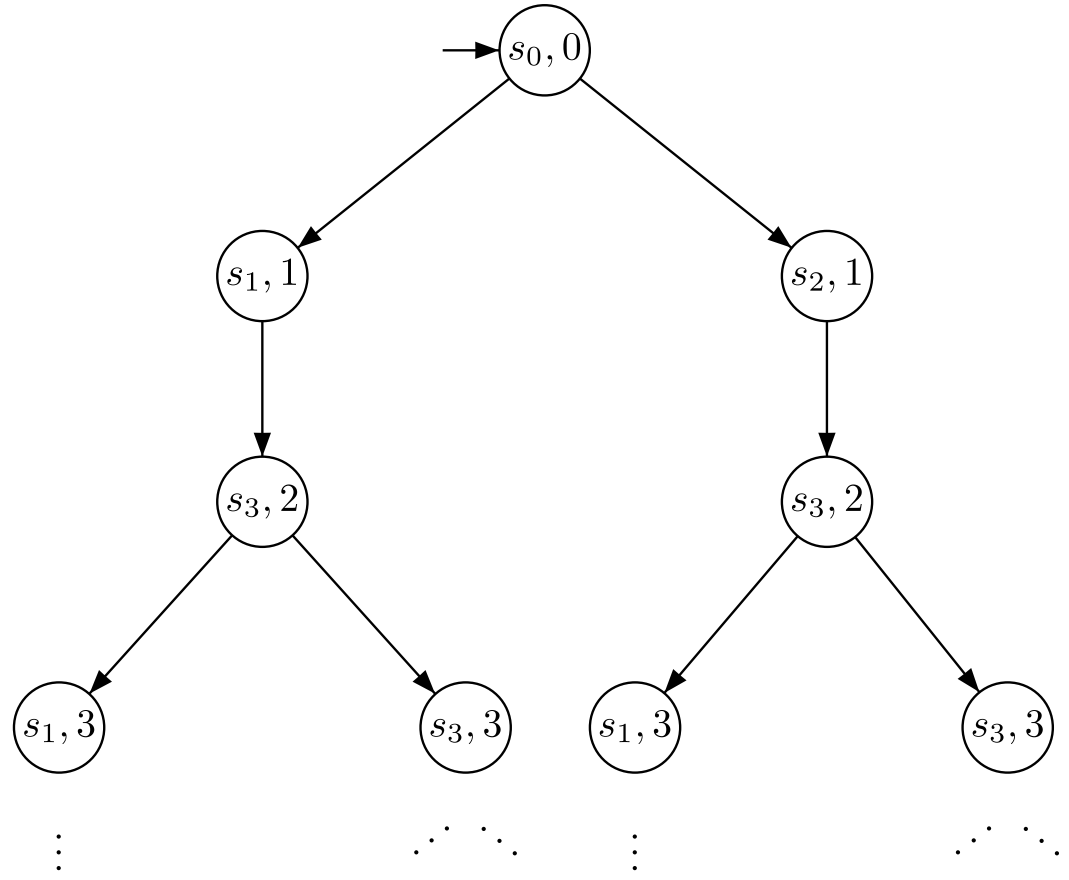
- Soit M= (Q, T, A, q_0, AP, l).
- On supposera que T est totale, i.e., chaque état a au moins un successeur.
- On peut compléter une structure de Kripke : on rajoute un état puits successeur des états dead-lock.
- Exemple

Exécutions et traces

- Soit $M=(Q,T,A, q_0, AP, l)$. Une exécution de M est une séquence infinie $r=q_0a_0q_1a_1q_2a_2\ldots$ telle que $(q_i,a_i,q_{i+1})\in T$, pour tout $i\geq 0$.
- On peut omettre l'étiquetage des transitions : $r=q_0q_1q_2\ldots$
- Une trace d'exécution de M est l'étiquetage d'une exécution: $l(r)=l(q_0)l(q_1)l(q_2)\ldots$

Arbre d'exécutions d'une structure de Kripke





Arbre d'exécutions d'une structure de Kripke

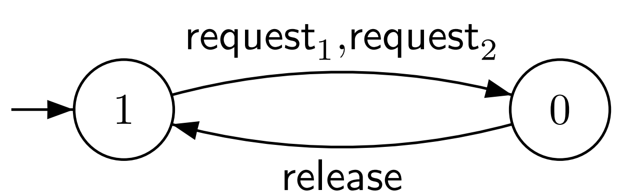
- Correspond au «dépliage» de la structure de Kripke
- Sa racine est l'état initial de la structure de Kripke
- Au niveau i, les fils d'un noeud sont les états successeurs au niveau i+1
- La relation de transition est totale : l'arbre est infini

Systèmes concurrents

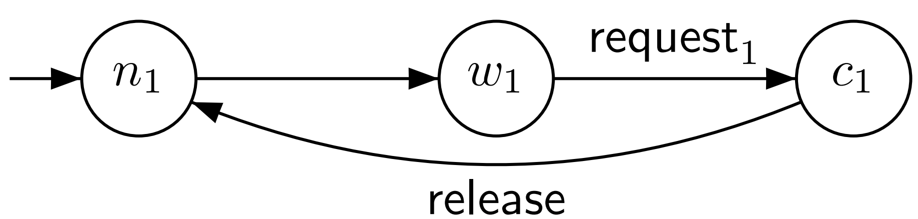
- Compositionnalité des modèles, description modulaire
- Différents modes de synchronisation
  - entrelacement
  - variables partagées
  - communication par rendez-vous (synchrone)
  - communication par canaux de communication (asynchrone)
  - produit synchrone
  - ...
- explosion combinatoire

Exemple : exclusion mutuelle II

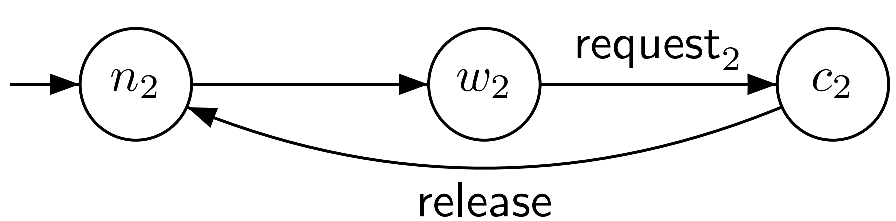
y :

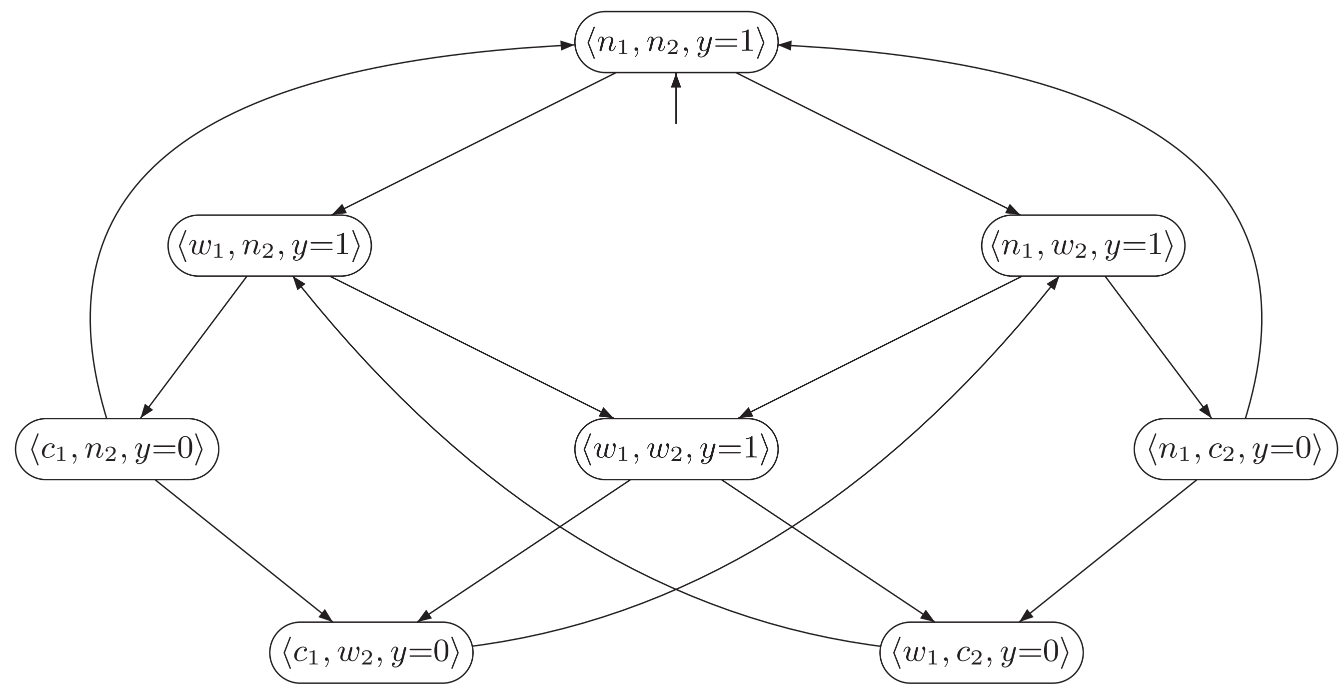


P1



P2



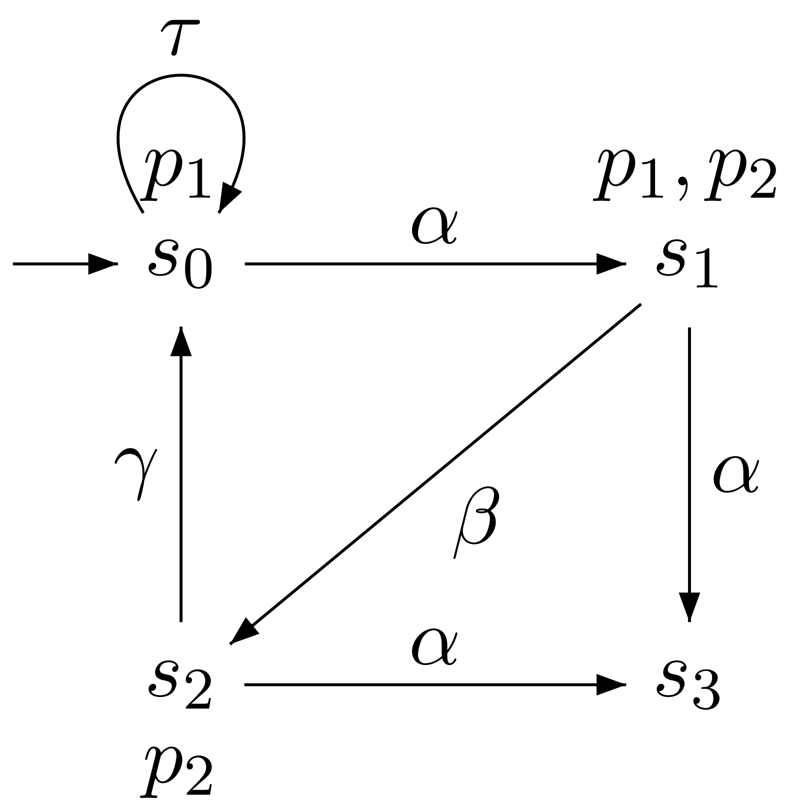


Descriptifs de haut niveau

- Programmes séquentiels
- Programmes concurrents
- Réseaux de Petri
- ...

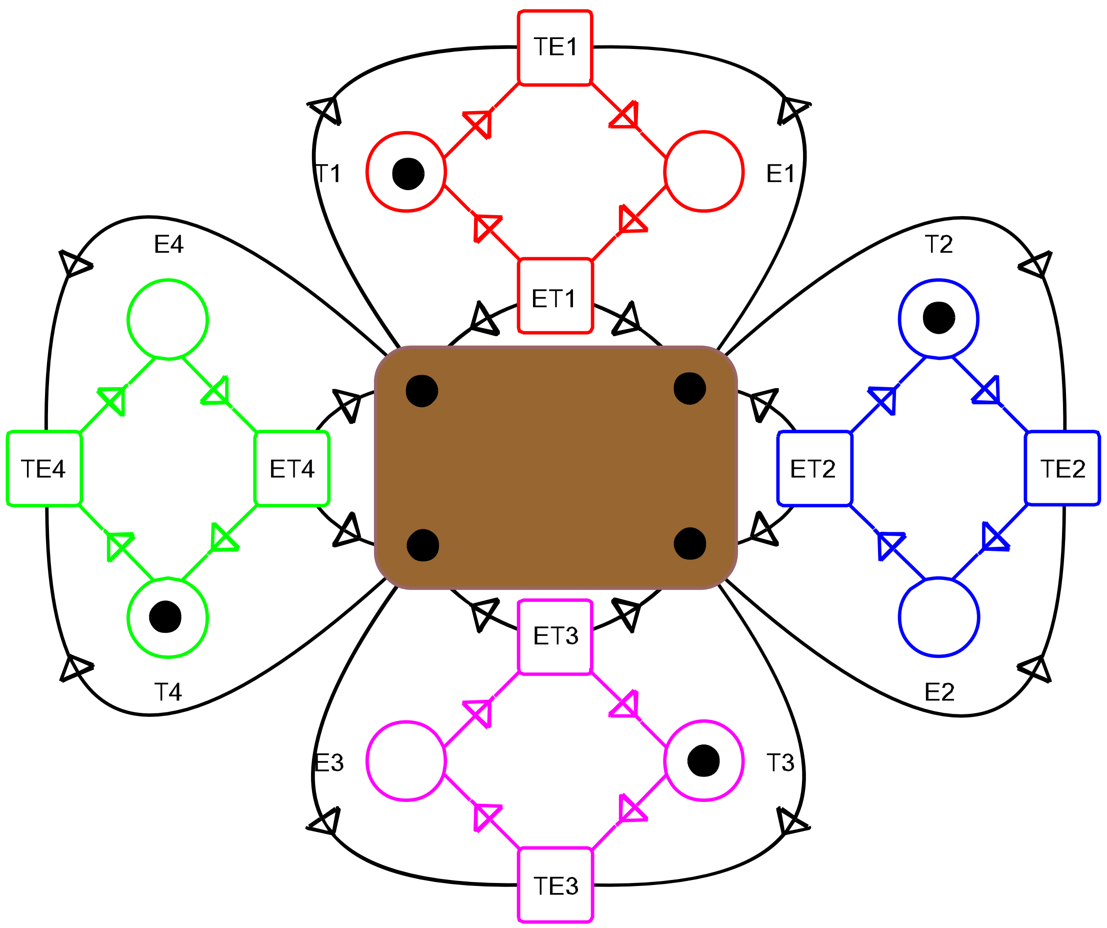
Exemple des feux de signalisation

Exercice



- Décrire formellement la structure de Kripke ci-dessus.
- Donner une exécution, une trace d'exécution
- Dessiner l'arbre d'exécutions associé (3 premiers niveaux).

Exercice : le dîner des philosophes



Dessiner la structure de Kripke sous-jacente

2. Généralités sur les spécifications

Propriétés sur les systèmes de transition (I)

- Accessibilité : un état donné est accessible depuis l'état initial
- Invariance : tous les états du système vérifient une certaine propriété
- Sûreté : quelque chose de mauvais n'arrive jamais

Propriétés sur les systèmes de transition (II)

- Vivacité : Quelque chose de «bon» finira par arriver
- Equité : Quelque chose se produira infiniment souvent

Logiques temporelles

- Permettent d'exprimer propriétés sur séquences d'observations
- Utilisation de connecteurs temporels et de quantificateurs sur les chemins

Logiques temporelles : pourquoi?

- On pourrait utiliser logique du premier ordre.
- Exemple : «toute requête sera un jour satisfaite»

$$\forall t \cdot \big(\text{requete} \rightarrow \exists t' \geq t \cdot (\text{reponse})\big)$$

Logiques temporelles : pourquoi?

- On pourrait utiliser l… …emier ordre.
- Ex… …equête sera un jour s…

Difficile à écrire/comprendre
Vérification peu efficace

$$\forall t \cdot \big(\text{requete} \rightarrow \exists t' \geq t \cdot (\text{reponse})\big)$$

Logiques temporelles

- Pas de variable (instants implicites), mais modalités.
- Temporel ≠ temporisé : logiques temporelles ne quantifient pas écoulement du temps.

Logiques temporelles linéaires ou arborescentes

- 2 approches :
  - temps linéaire : propriétés des séquences d'exécutions (futur déterminé)
  - temps arborescent : propriétés de l'arbre d'exécutions (tous les futurs possibles)

3. Algorithmes de Model-Checking

Logique temporelle linéaire : LTL

[Pnueli 77]

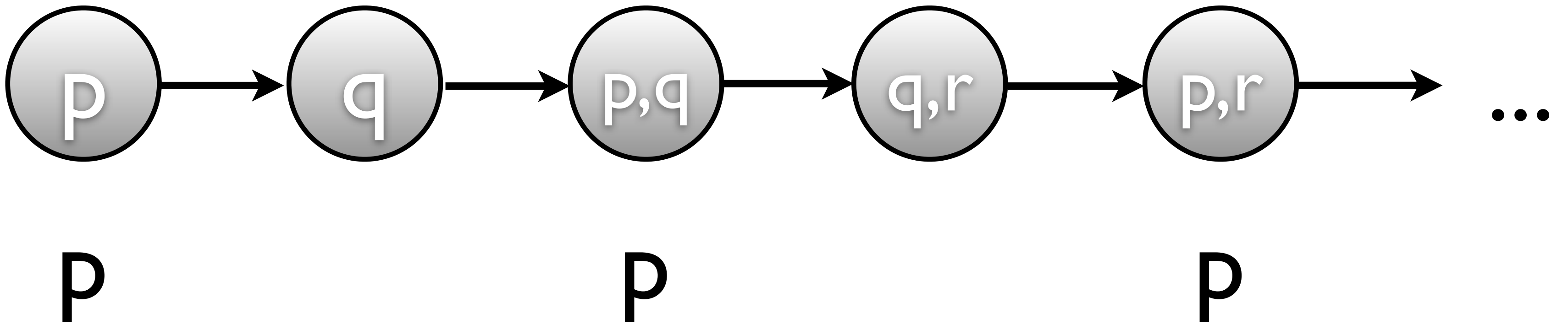
- Modèle des formules : une trace d'exécution infinie.
- $t, i \models \varphi$ ssi la formule φ est vérifiée à la position i de la trace.
- Défini inductivement sur la formule

Logique temporelle linéaire : LTL

- Rappel : une trace d'exécution ≡ exécution dans laquelle seul l'étiquetage des états est visible
- → c'est un mot (infini) sur l'alphabet 2^{AP}.
- Soit t une trace, on note t(i) la «lettre» à la position i≥0, i.e. l'ensemble des propositions atomiques vraies.

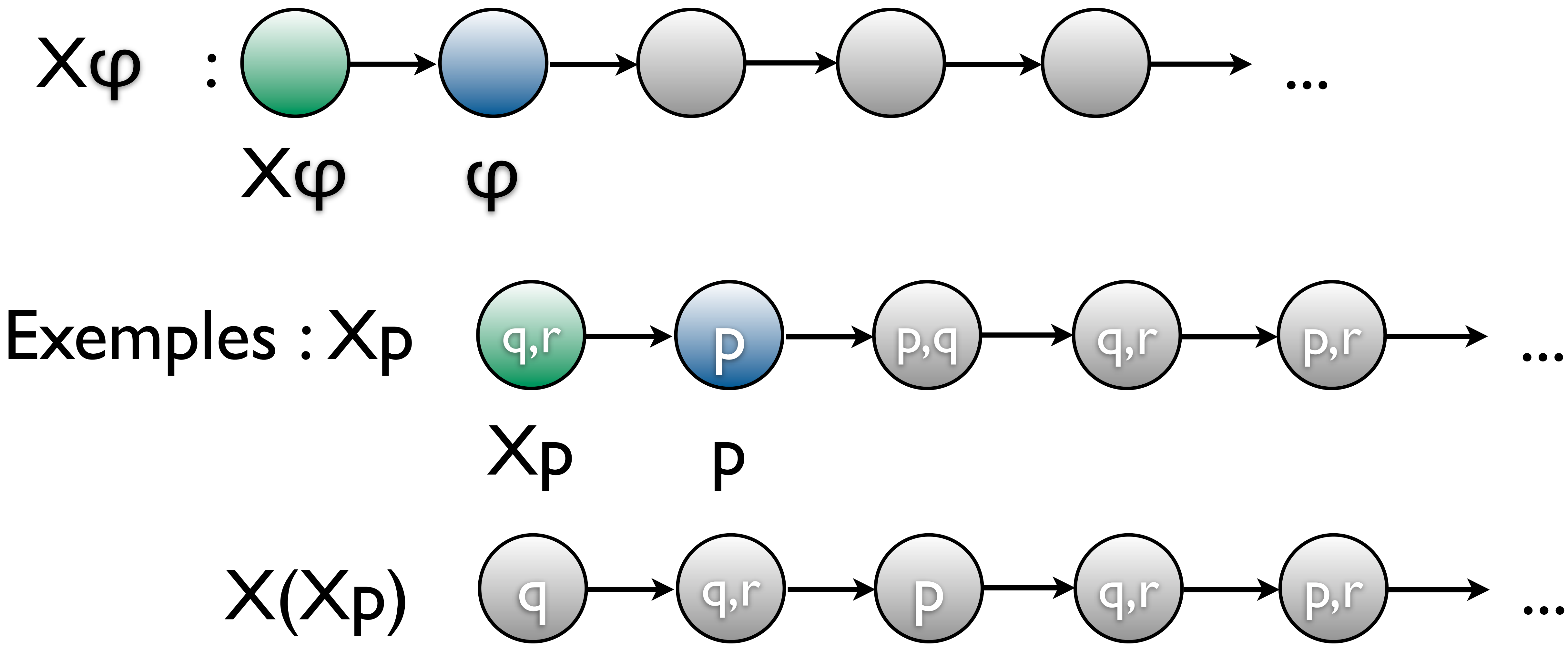
Logique temporelle linéaire : LTL

p :

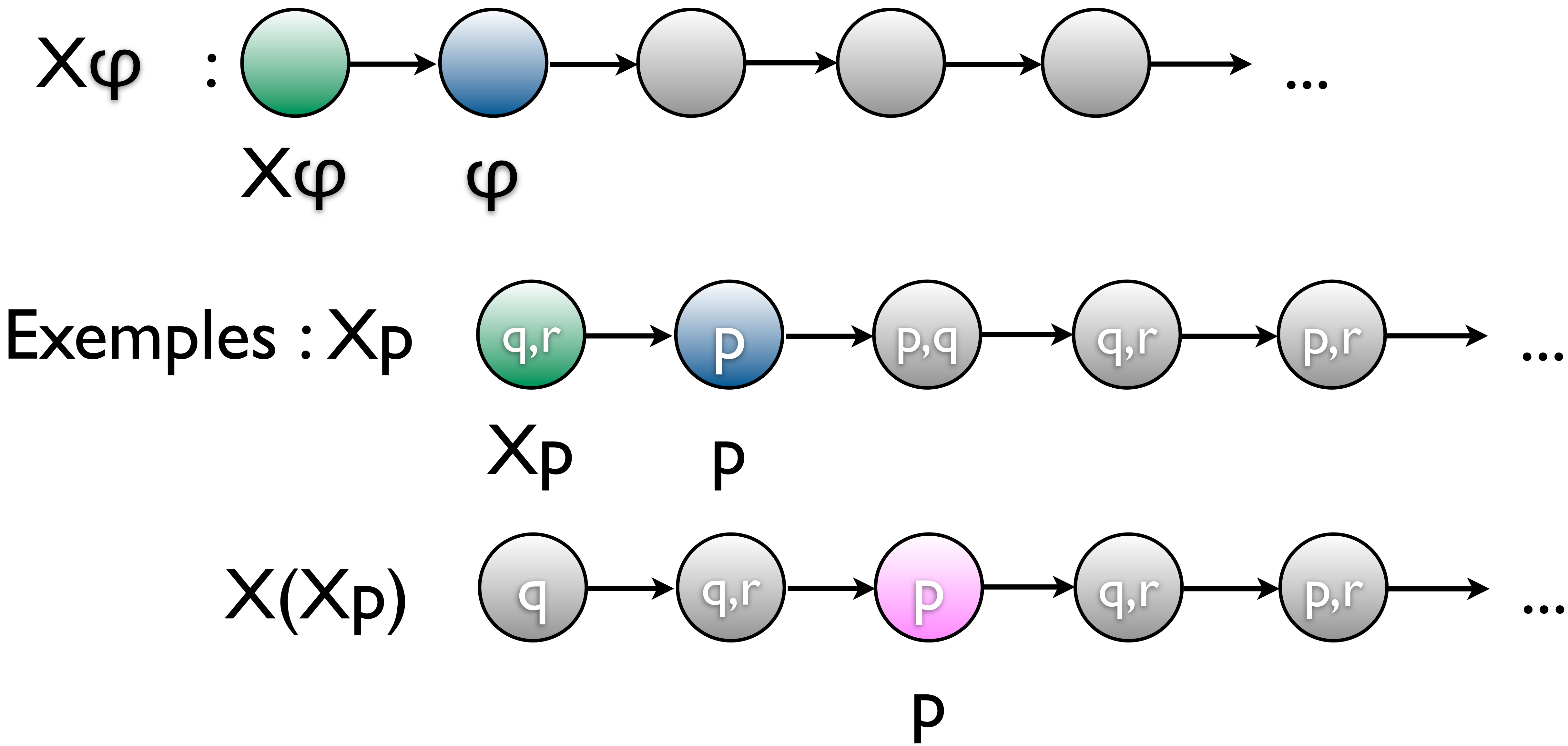


t,i ⊧ p ssi p∈t(i)

Logique temporelle linéaire : LTL



Logique temporelle linéaire : LTL

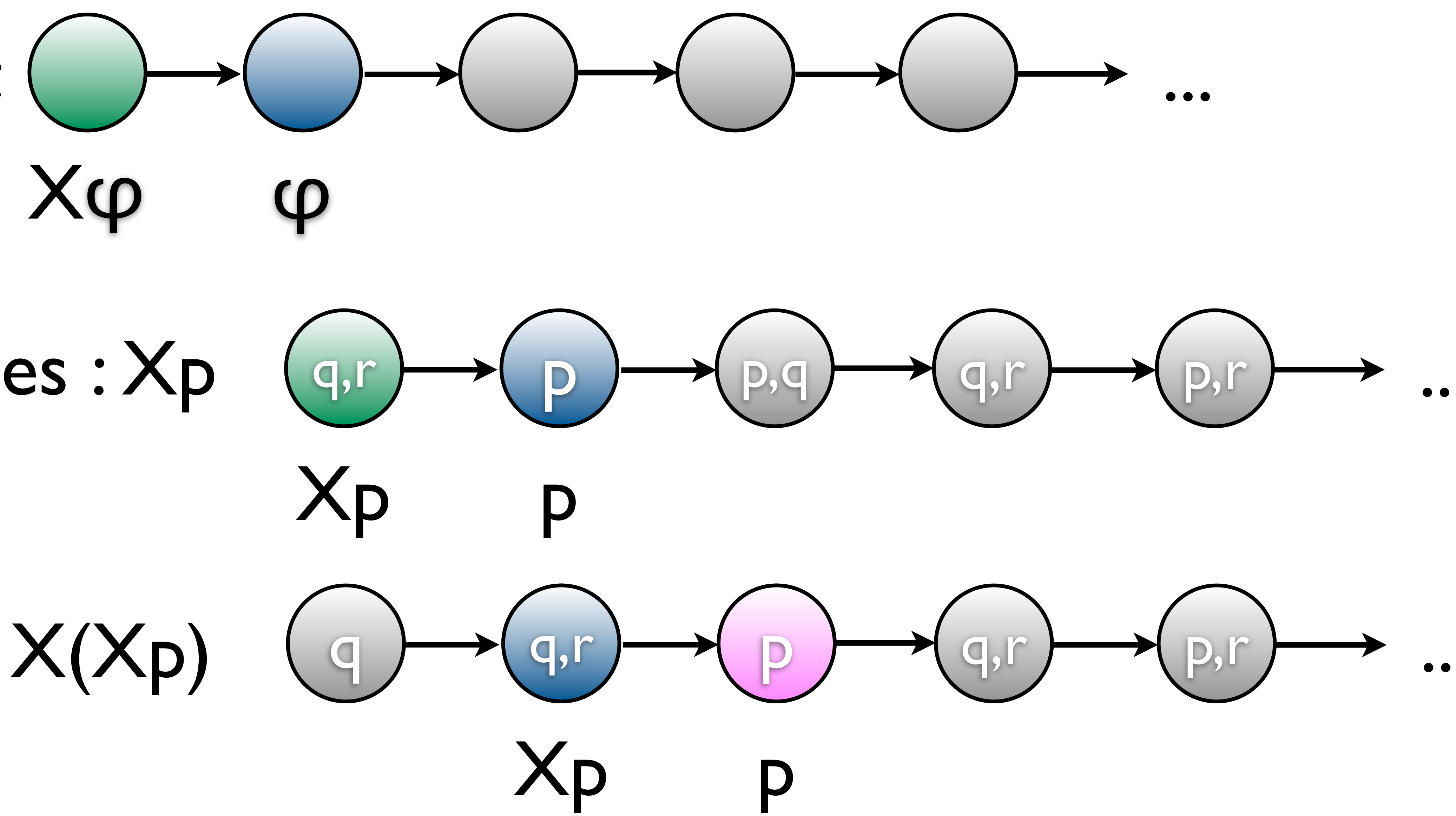


Logique temporelle linéaire : LTL

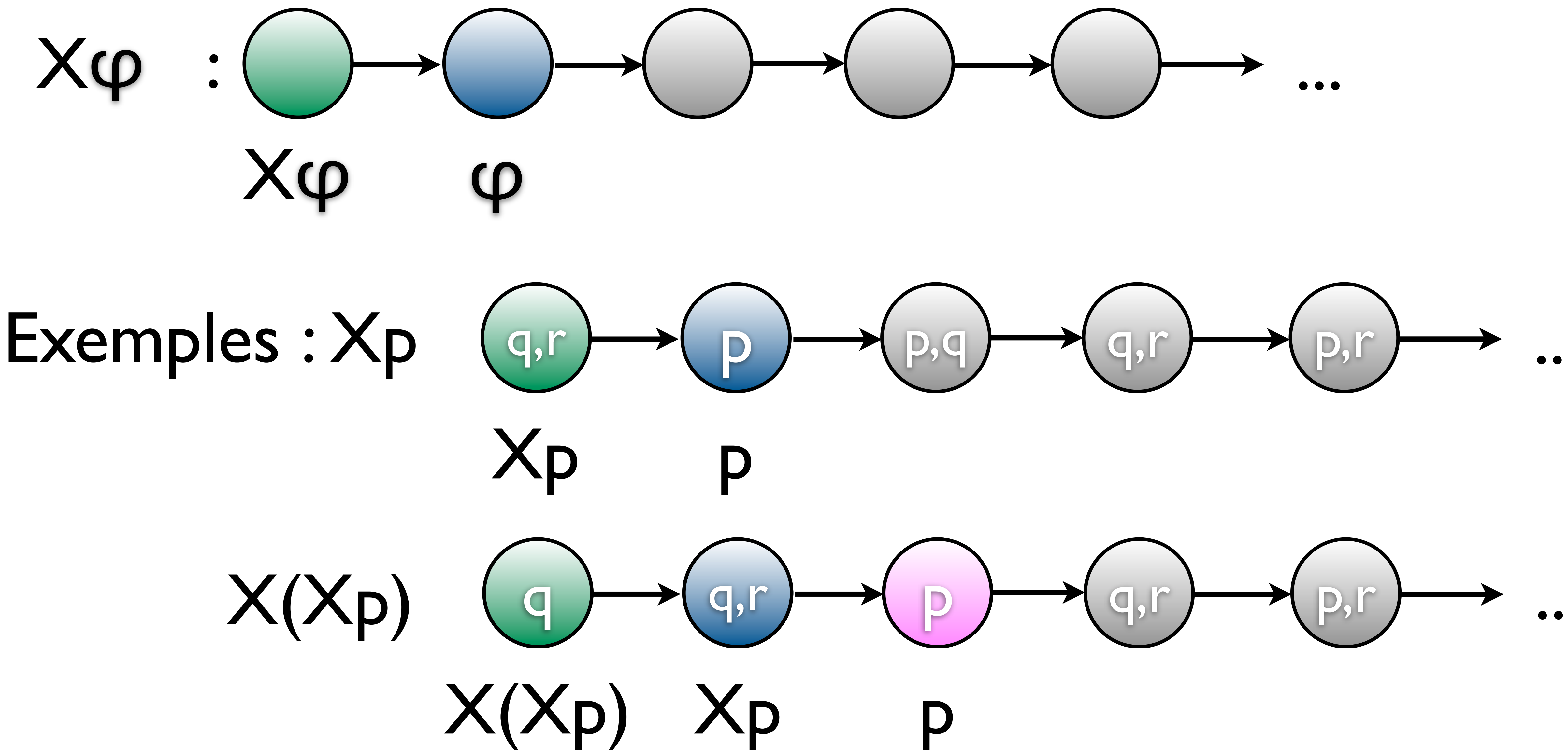
$X\varphi$:

Exemples : Xp

$X(Xp)$

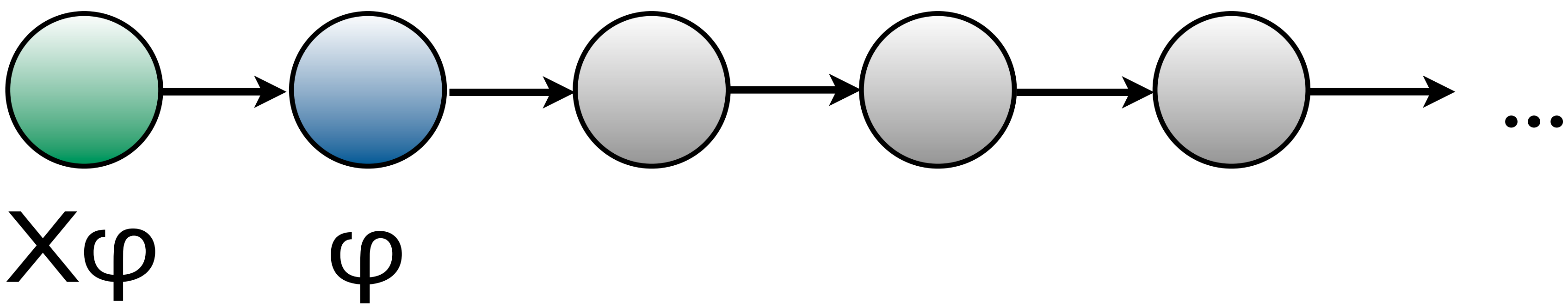


Logique temporelle linéaire : LTL



Logique temporelle linéaire : LTL

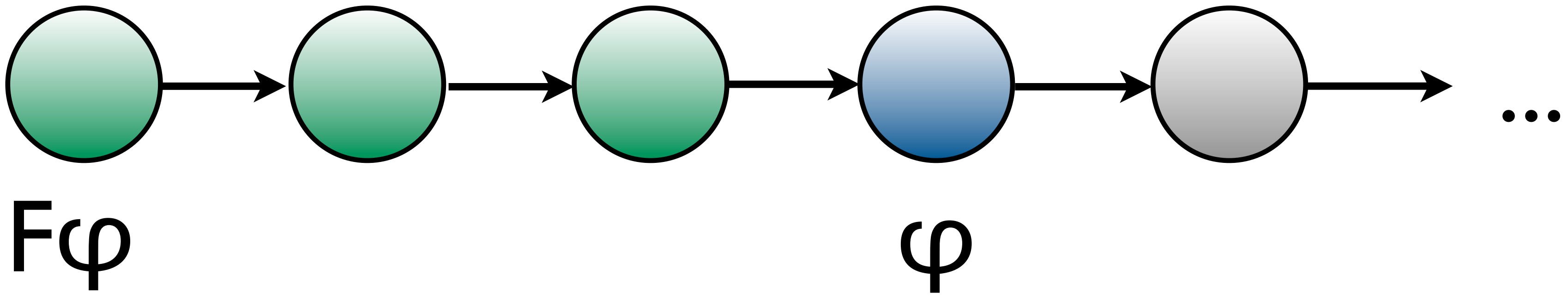
Xφ :



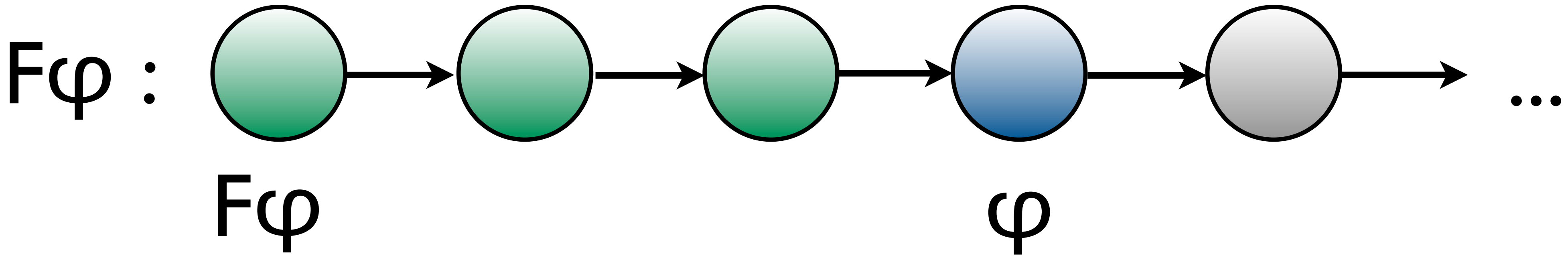
t,i ⊧ Xφ ssi t,i+1⊧φ

Logique temporelle linéaire : LTL

Fφ :



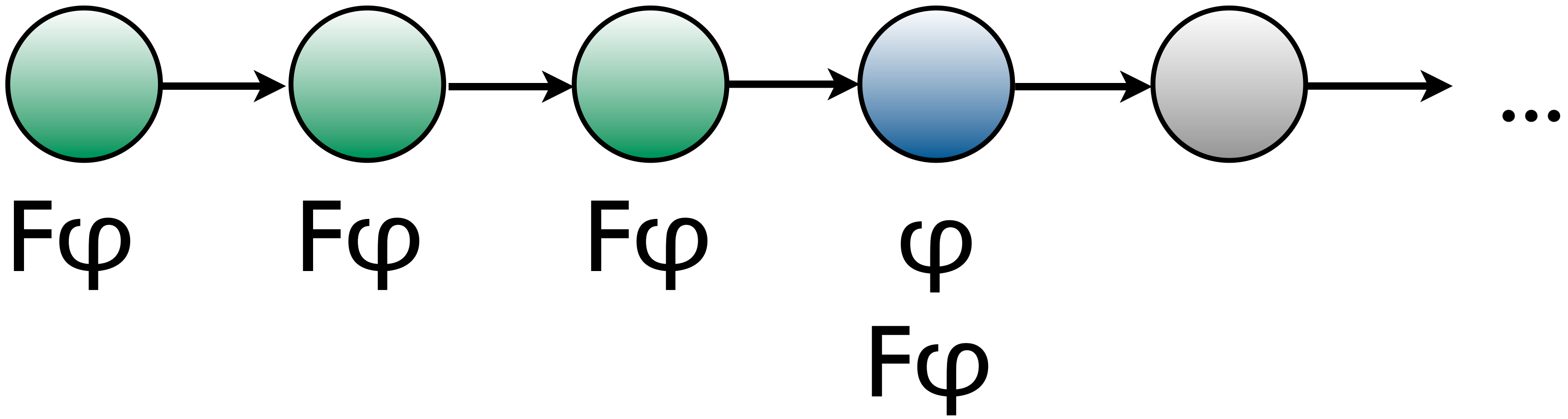
Logique temporelle linéaire : LTL



$t,i \models F\varphi$ ssi il existe $j \geq i$ tel que $t,j \models \varphi$

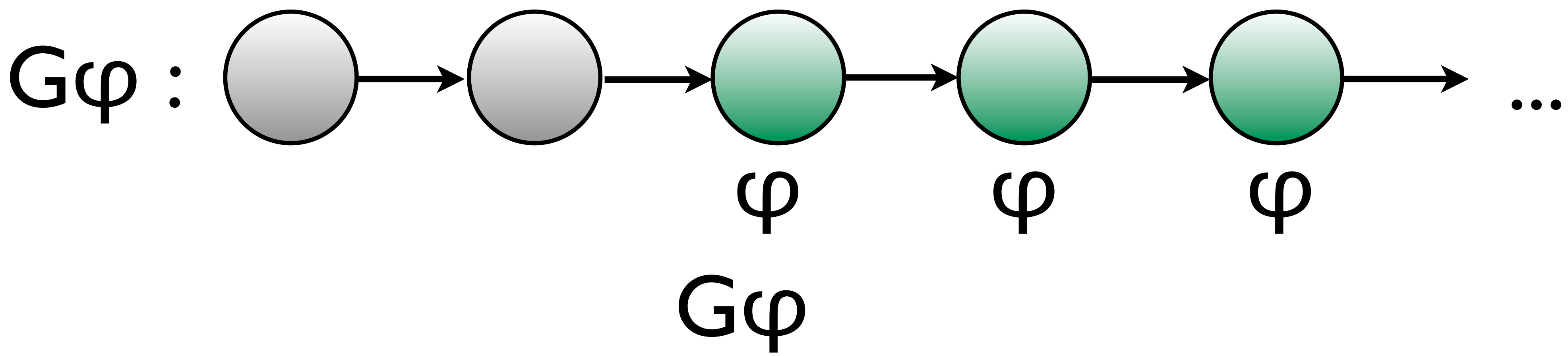
Logique temporelle linéaire : LTL

Fφ :

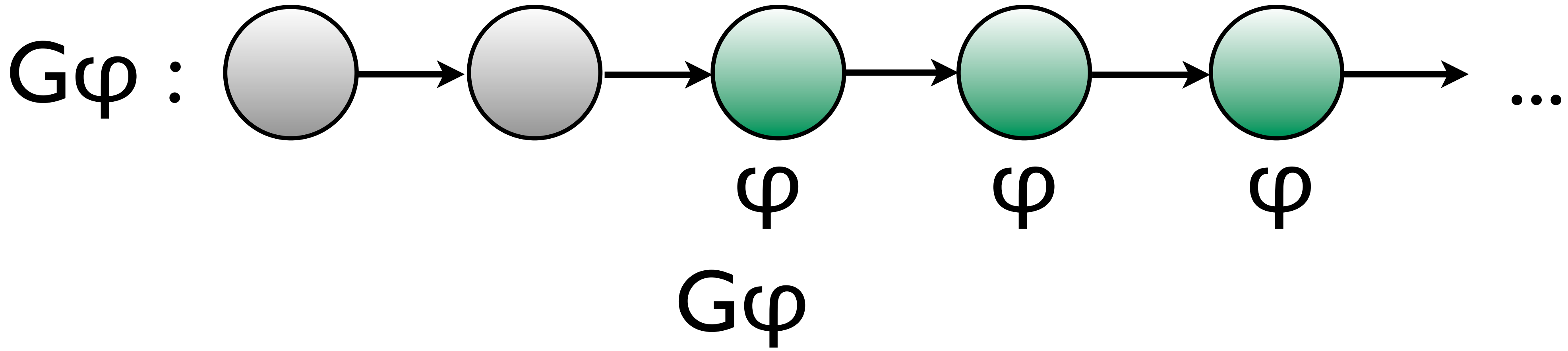


$t,i \models F\varphi$ ssi il existe $j \geq i$ tel que $t,j \models \varphi$

Logique temporelle linéaire : LTL

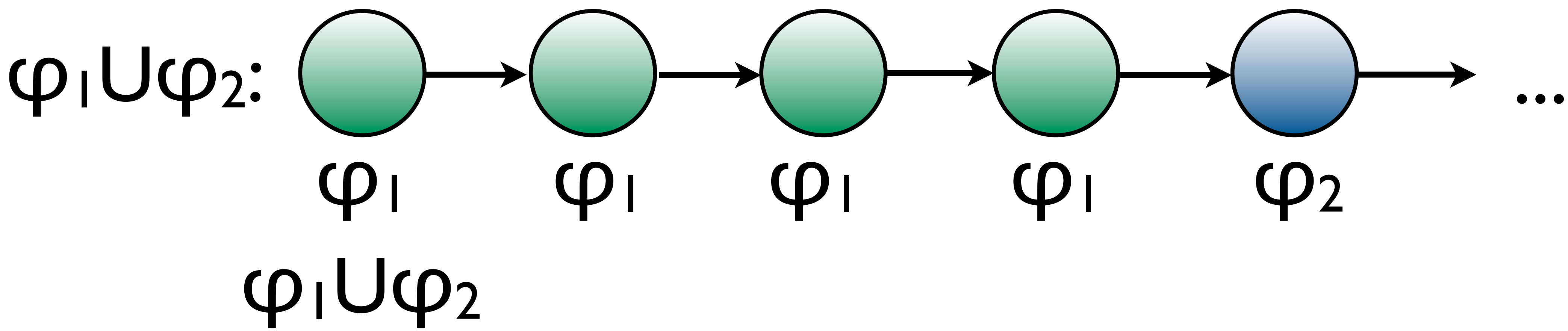


Logique temporelle linéaire : LTL

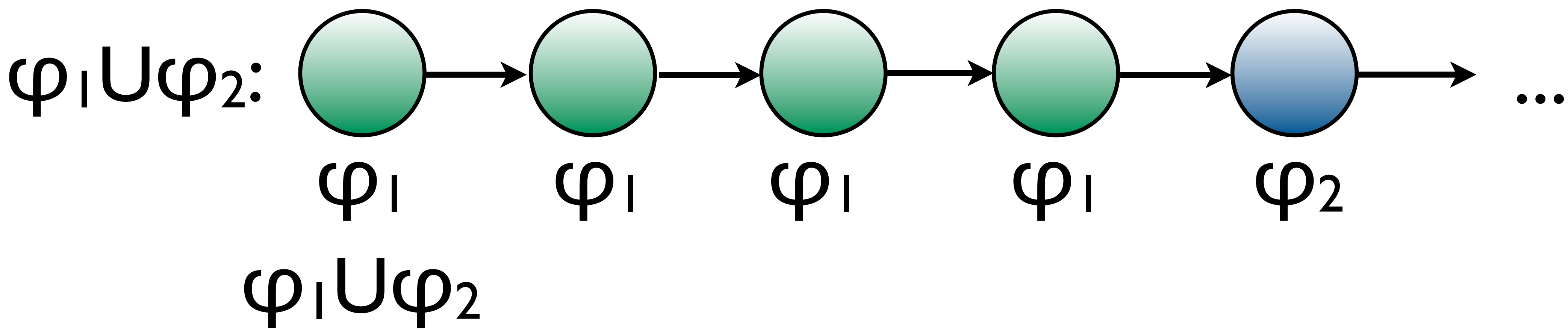


t,i ⊧ Gφ ssi pour tout j≥i, t,j⊧φ

Logique temporelle linéaire : LTL



Logique temporelle linéaire : LTL



t,i ⊧ $\varphi_1 U \varphi_2$ ssi il existe $j \geq i$, t,j ⊧ φ_2 et, pour tout $i \leq k < j$, t,k ⊧ φ_1

Logique temporelle linéaire : LTL

$\varphi ::= p \in AP \mid \neg\varphi \mid \varphi \vee \varphi$
$\mid X\varphi \mid F\varphi \mid G\varphi \mid \varphi U \varphi$

$t,i \models p$ ssi $p \in t(i)$

$t,i \models \neg\varphi$ ssi $t,i \not\models \varphi$

$t,i \models \varphi_1 \vee \varphi_2$ ssi $t,i \models \varphi_1$ ou $t,i \models \varphi_2$

$t,i \models X\varphi$ ssi $t,i+1 \models \varphi$

$t,i \models F\varphi$ ssi il existe $j \geq i$ tel que $t,j \models \varphi$

$t,i \models G\varphi$ ssi pour tout $j \geq i$, $t,j \models \varphi$

$t,i \models \varphi_1 U \varphi_2$ ssi il existe $j \geq i$, $t,j \models \varphi_2$ et, pour tout $i \leq k < j$, $t,k \models \varphi_1$

Logique temporelle linéaire : LTL

$\varphi ::= p \in AP \mid \neg\varphi \mid \varphi_1 \vee \varphi_2$
$\mid X\,\varphi \mid F\,\varphi \mid G\varphi \mid \varphi U \varphi$

$\top \equiv p \vee \neg p$, pour $p \in AP$ quelconque

$\bot \equiv \neg \top$

$\varphi_1 \wedge \varphi_2 \equiv \neg\varphi_1 \vee \neg\varphi_2$

$\varphi_1 \rightarrow \varphi_2 \equiv \neg\varphi_1 \vee \varphi_2$

Logique temporelle linéaire : LTL

En fait, $F\varphi$ et $G\varphi$ macros aussi :

- $F\varphi \equiv \top U \varphi$
- $G\varphi \equiv \neg F(\neg\varphi)$

Exercice : vérifier.

Logique temporelle linéaire : LTL

$\varphi ::= p \in AP \mid \neg\varphi \mid \varphi_1 \vee \varphi_2$
$\mid X\,\varphi \mid \varphi U \varphi$

- Autres macros utiles :
  - (Weak until) $\varphi_1 W \varphi_2 \equiv G\varphi_1 \vee \varphi_1 U \varphi_2$
  - (Release) $\varphi_1 R \varphi_2 \equiv \varphi_2 W (\varphi_1 \wedge \varphi_2) \equiv G\varphi_2 \vee \varphi_2 U (\varphi_1 \wedge \varphi_2)$

LTL : Exemples

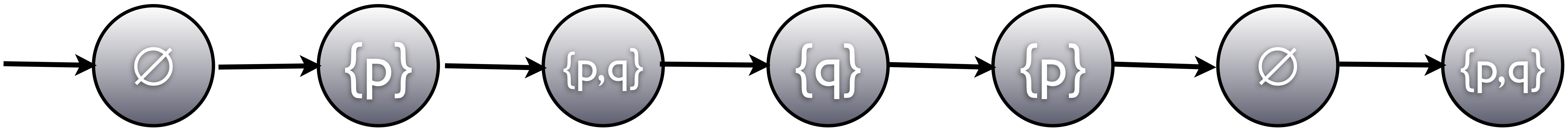
- Accessibilité : F (x=0)
- Invariance : G ¬(x=0)
- Vivacité : GF (active)
- Equité forte : GF enabled → GF scheduled
- Equité faible : FG enabled → GF scheduled
- Relâchement de contrainte : reset R alarm

LTL : Exercice 1

- Toute requête sera un jour satisfaite (AP = {requete, reponse})
- A chaque fois que de l'argent a été retiré, le code pin a été fourni (AP={cash-withdraw, pin-ok})
- Deux processus ne sont jamais en section critique en même temps (AP= {$critique_1$, $critique_2$})
- Si un processus demande l'accès en section critique, il l'obtiendra un jour (AP = {demande_crit, acc_crit})
- Une fois que le feu est vert, il ne peut pas devenir rouge immédiatement (AP= {vert, rouge})
- Lorsque le feu est rouge, il deviendra vert un jour
- Lorsque le feu est vert, il deviendra rouge un jour, après avoir été orange (AP= {vert, rouge, orange})

LTL : Exercice II

- Vérifier que $\neg X\varphi \equiv X\neg\varphi$, $\neg(\varphi_1 U \varphi_2) \equiv \neg\varphi_1 R \neg\varphi_2$
- Dites si, à chaque position de la trace ci-dessous, les propositions suivantes sont vérifiées : $p \wedge q$, $F(p \wedge q)$, pUq.



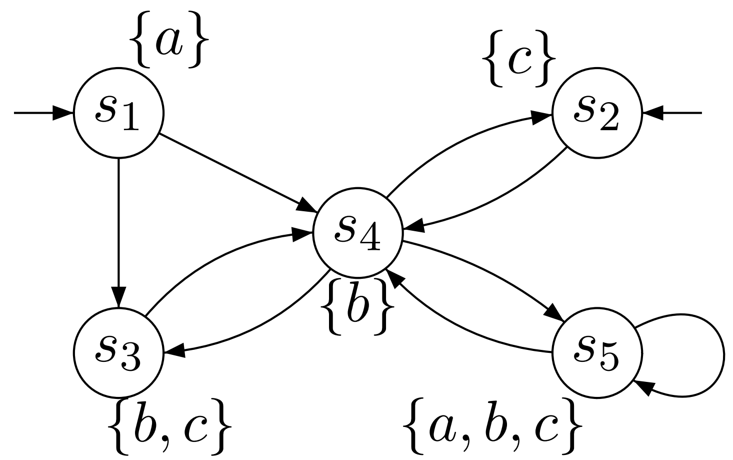
LTL : Exercice III

- Les équivalences suivantes sont-elles vraies?
  - $G(Fp \wedge Fq) \leftrightarrow GFp \wedge GFq$
  - $F(Gp \wedge Gq) \leftrightarrow FGp \wedge FGq$
  - $G(Fp \vee Fq) \leftrightarrow GFp \vee GFq$
  - $F(Gp \vee Gq) \leftrightarrow FGp \vee FGq$
  - $GF(p \wedge q) \leftrightarrow GFp \wedge GFq$
  - $GF(p \vee q) \leftrightarrow GFp \vee GFq$
  - $FG(p \wedge q) \leftrightarrow FGp \wedge FGq$
  - $FG(p \vee q) \leftrightarrow FGp \vee FGq$

Model-Checking de LTL

- Données : Une structure de Kripke $M=(Q, T, A, q_0, AP, l)$ et une formule LTL φ.
- Question : Est-ce que $M \models \varphi$?
  - $M \models \varphi$ ssi $t,0 \models \varphi$ pour toute trace initiale t de M.

Exercice



M⊨φ?

- φ=FGc
- φ=GFc
- φ=Ga
- φ=aU(G(b∨c))
- X¬c→XXc

Model-Checking de LTL: principe

- Soit Σ un alphabet. On note Σ^* l'ensemble des mots finis et Σ^ω les mots infinis.
- Modèles de φ = mots infinis. Soit $[\![\varphi]\!]$ le langage des modèles de la formule : $[\![\varphi]\!]=\{t\in(2^{AP})^\omega \mid t, 0\models \varphi\}$
- Soit $[\![M]\!]$ le langage des traces initiales de M : $[\![M]\!]=\{t\in (2^{AP})^\omega \mid t \text{ est une trace initiale de } M\}$
- Le problème du model-checking revient donc à vérifier si : $[\![M]\!]\subseteq[\![\varphi]\!]$

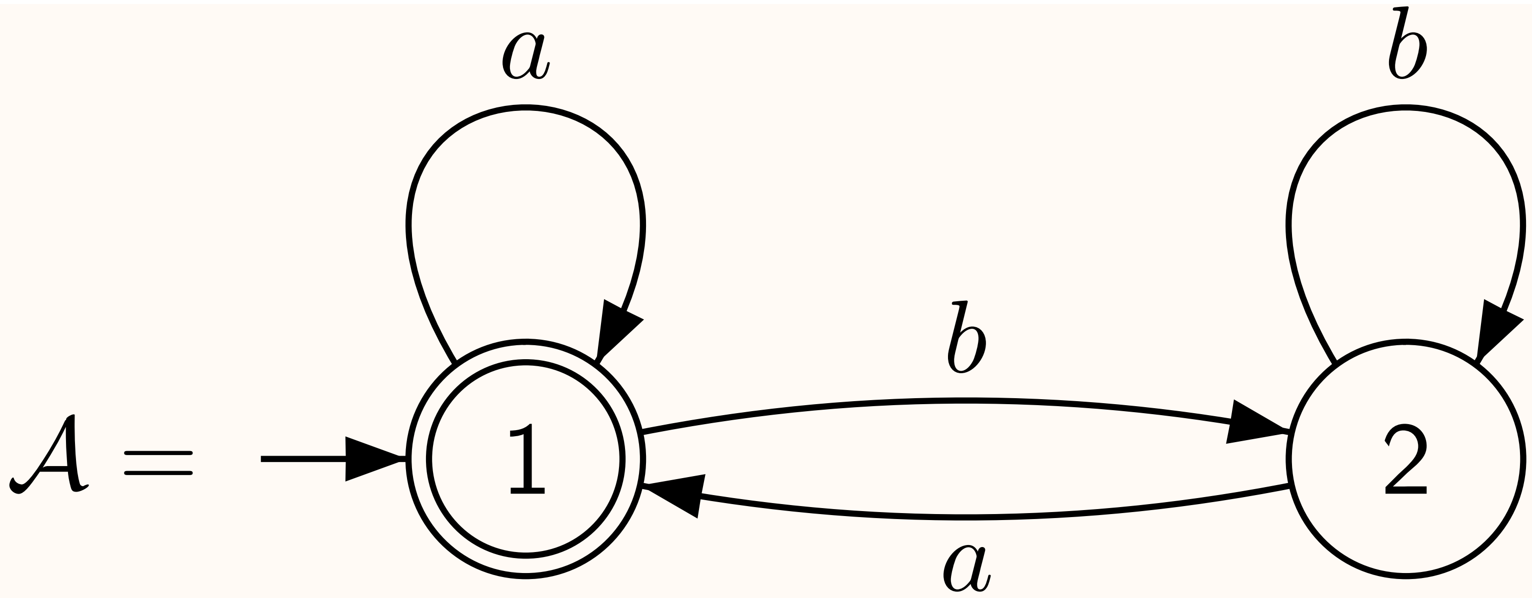
Outil : les automates de Büchi

- Définition : Un automate de Büchi est un n-uplet $A=(Q, \Sigma, I, T, F)$ avec
  - Q un ensemble fini d'états
  - Σ un alphabet fini
  - $I \subseteq Q$ les états initiaux
  - $T \subseteq Q \times \Sigma \times Q$ la relation de transition
  - $F \subseteq Q$ un ensemble d'états acceptants (ou répétés)

Outil : les automates de Büchi

- Une exécution de A sur un mot infini $w=w_0w_1w_2\ldots$ de Σ^{ω} est une séquence $r=q_0q_1q_2q_3\ldots$ telle que $q_0 \in I$ et $(q_i,w_i,q_{i+1}) \in T$, pour tout $i \geq 0$.
- r est acceptante si $q_i \in F$ pour un nombre infini de i.
- w est accepté par A s'il existe une exécution acceptante de A sur w.
- $L(A)=\{w \in \Sigma^{\omega} \mid w \text{ accepté par } A\}$.

Automate de Büchi: exemple



$$\mathcal{L}(\mathcal{A}) = \{w \in \{a,b\}^\omega \mid |w|_a = \omega\}$$

Les automates de Büchi pour LTL

- Définition : Un automate de Büchi est un n-uplet $A=(Q, \Sigma, I, T, F)$ avec
  - Q un ensemble fini d'états
  - Σ un alphabet fini
  - $I \subseteq Q$ les états initiaux
  - $T \subseteq Q \times \Sigma \times Q$ la relation de transition
  - $F \subseteq Q$ un ensemble d'états acceptants (ou répétés)

Les automates de Büchi pour LTL

- Définition : Un automate de Büchi est un n-uplet $A=(Q, \Sigma, I, T, F)$ avec
  - Q un ensemble fini d'états
  - Σ un alphabet fini $\boxed{\Sigma = 2^{AP}}$
  - $I \subseteq Q$ les états initiaux
  - $T \subseteq Q \times \Sigma \times Q$ la relation de transition
  - $F \subseteq Q$ un ensemble d'états acceptants (ou répétés)

Exercice

- Exemple : automate de Büchi reconnaissant p, Xp.
- Construire des automates de Büchi reconnaissant Fp, XXp, Gp, FGp, GFp, pUq, pRq.

Automates de Büchi

Théorème : Les automates de Büchi sont clos par union, intersection, et complément.

Théorème : on peut tester le vide d'un automate de Büchi.

Model-Checking LTL : approche par automates

- Donnée: Structure de Kripke M, formule LTL φ.
- Etapes de l'algorithme :
  - Transformer M en un automate A_M tel que $L(A_M)=[\![M]\!]$ (assez facile)
  - Transformer φ en un automate A_φ tel que $L(A_\varphi)=[\![\varphi]\!]$ (plus difficile)
  - Tester si $L(A_M) \subseteq L(A_\varphi)$, i.e., si $L(A_M) \cap L(A_\varphi)^c=\varnothing$.

Model-Checking LTL : approche par automates

- Donnée: Structure de Kripke M, formule LTL φ.
- Etapes de l'algorithme :
  - Transformer M en u … A_M tel que $L(A_M)=[\![M]\!]$ (a …
  - Trans … un automate A_φ tel que $L(A$ … plus difficile)
  - Tester si $L(A_M)\subseteq L(A_\varphi)$, i.e., si $L(A_M)\cap L(A_\varphi)^c=\emptyset$.

Difficile de complémenter un automate de Büchi!!

Model-Checking LTL : approche par automates

- Donnée: Structure de Kripke M, formule LTL φ.
- Etapes de l'algorithme :
  - Transformer M en un automate A_M tel que $L(A_M)=[\![M]\!]$
  - Transformer φ en un automate $A_{\neg\varphi}$ tel que $L(A_{\neg\varphi})=[\![\neg\varphi]\!]$
  - Tester si $L(A_M) \cap L(A_{\neg\varphi})=\emptyset$.

Transformer φ en un automate de Büchi

I. Automates de Büchi généralisés

II. Réduire la formule

  II.I. Forme normale négative

  II.II. Réduire les connecteurs temporels

III. Construire un graphe

IV. Transformation en automate de Büchi

Transformer φ en un automate de Büchi

I. Automates de Büchi généralisés

II. Réduire la formule

    II.I. Forme normale négative

    II.II. Réduire les connecteurs temporels

III. Transformation en automate de Büchi généralisé

Automates de Büchi généralisés

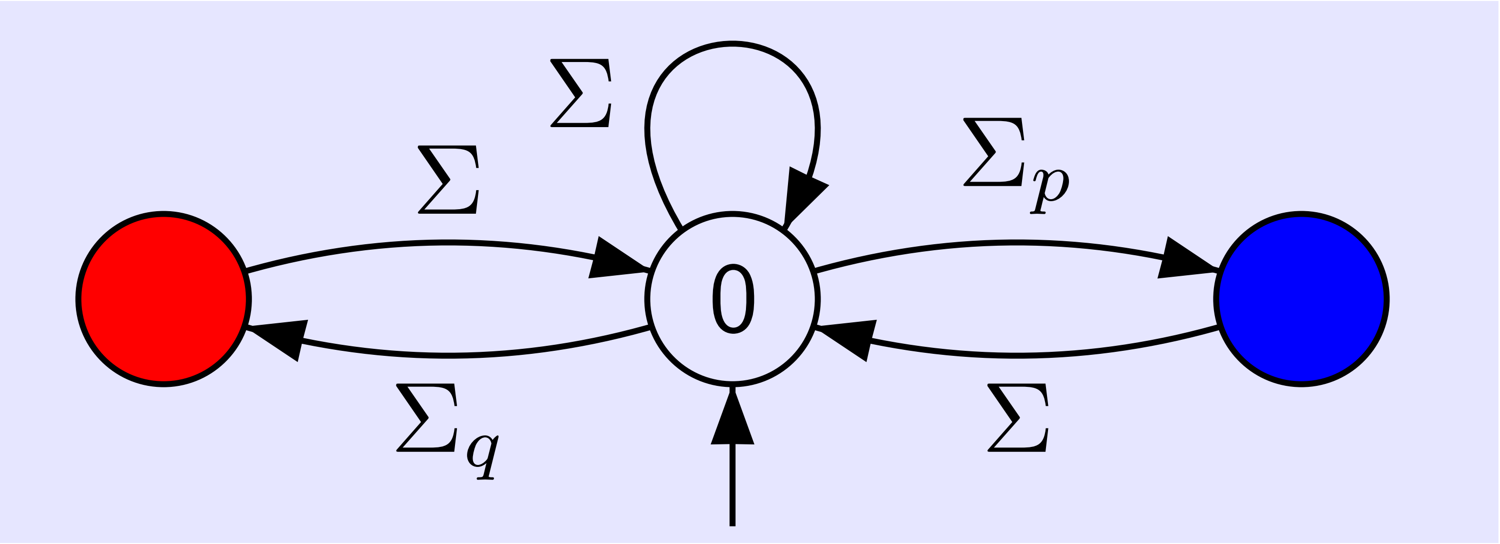
- Définition : Un automate de Büchi généralisé est un n-uplet $A=(Q, \Sigma, I, T, F)$ avec
  - Q un ensemble fini d'états
  - Σ un alphabet fini
  - $I \subseteq Q$ les états initiaux
  - $T \subseteq Q \times \Sigma \times Q$ la relation de transition
  - $F=\{F_1,F_2,\ldots,F_k\} \subseteq 2^Q$ un ensemble d'ensemble d'états acceptants (ou répétés)

Automates de Büchi généralisés

- Une exécution de A sur un mot infini $w=w_0w_1w_2\ldots$ de Σ^ω est une séquence $r=q_0q_1q_2q_3\ldots$ telle que $q_0 \in I$ et $(q_i,w_i,q_{i+1}) \in T$, pour tout $i \geq 0$.
- r est acceptante si pour tout $\mathcal{F} \in F$, $q_i \in \mathcal{F}$ pour un nombre infini de i.
- w est accepté par A s'il existe une exécution acceptante de A sur w.
- $L(A)=\{w \in \Sigma^\omega \mid w \text{ accepté par } A\}$.

Automates de Büchi généralisés : exemple

GFp∧GFq:



Des ABG aux AB

Théorème : Tout automate de Büchi généralisé A peut être transformé en un automate de Büchi A' tel que L(A)=L(A')

Transformer φ en un automate de Büchi

I. Automates de Büchi généralisés

II. Réduire la formule

II.I. Forme normale négative

II.II. Réduire les connecteurs temporels

III. Transformation en automate de Büchi généralisé

Forme normale négative

$$\varphi ::= \bot \mid \top \mid p \mid \neg p \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid$$
$$X\varphi \mid \varphi U \varphi \mid \varphi R \varphi$$

- $\neg\neg p = p$
- $\neg(\varphi_1 \vee \varphi_2) = \neg\varphi_1 \wedge \neg\varphi_2$
- $\neg(\varphi_1 \wedge \varphi_2) = \neg\varphi_1 \vee \neg\varphi_2$
- $\neg(X\varphi) = X(\neg\varphi)$
- $\neg(\varphi_1 U \varphi_2) = \neg\varphi_1 R \neg\varphi_2$
- $\neg(\varphi_1 R \varphi_2) = \neg\varphi_1 U \neg\varphi_2$

Exercice

- Transformer $G(p \rightarrow Fq)$ en forme normale négative

Réduire les connecteurs temporels

- Idée : Un état de notre graphe va représenter l'ensemble des propositions atomiques vérifiées au prochain instant de la séquence, et l'ensemble des sous-formules qu'il «promet» de vérifier à l'état suivant.
- Pour cela, on ne veut que des propositions atomiques (ou négations), et des sous-formules commençant par X (next).

Réduire les connecteurs temporels

- Un ensemble Z de formules en forme normale négative est réduit si

(1) pour tout $z \in Z$, z est de la forme p, $\neg p$ ou $X(z')$

(2) il est cohérent : $\bot \notin Z$, $\{p, \neg p\} \not\subseteq Z$, pour tout $p \in AP$.

Réduire les connecteurs temporels

- On utilise les équivalences suivantes :
  - $\varphi U \varphi' \equiv \varphi' \vee (\varphi \wedge X(\varphi U \varphi'))$
  - $\varphi R \varphi' \equiv (\varphi \wedge \varphi') \vee (\varphi' \wedge X(\varphi R \varphi'))$

Construction du graphe de réduction

Construction du graphe de réduction

$$\begin{tikzcd} & Z \cup \{\varphi \vee \varphi'\} \arrow[dl] \arrow[dr] & \\ Z \cup \{\varphi\} & & Z \cup \{\varphi'\} \end{tikzcd}$$

Construction du graphe de réduction

$$
\begin{tikzcd}
 & Z \cup \{\varphi \vee \varphi'\} \arrow[dl] \arrow[dr] & \\
Z \cup \{\varphi\} & & Z \cup \{\varphi'\}
\end{tikzcd}
\qquad\qquad
\begin{tikzcd}
Z \cup \{\varphi \wedge \varphi'\} \arrow[d] \\
Z \cup \{\varphi, \varphi'\}
\end{tikzcd}
$$

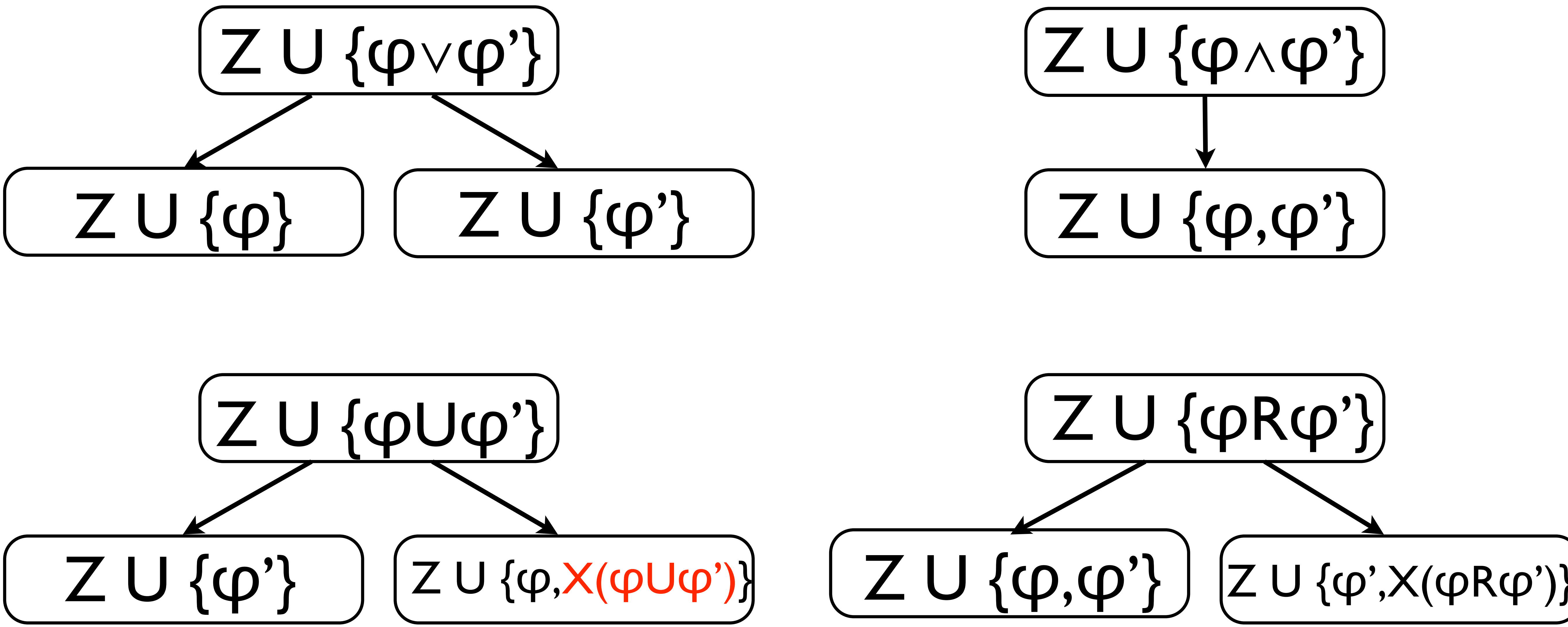
Construction du graphe de réduction

$$\begin{tikzcd}
 & Z \; U \; \{\varphi \vee \varphi'\} \arrow[dl] \arrow[dr] & \\
Z \; U \; \{\varphi\} & & Z \; U \; \{\varphi'\}
\end{tikzcd}$$

$$\begin{tikzcd}
Z \; U \; \{\varphi \wedge \varphi'\} \arrow[d] \\
Z \; U \; \{\varphi, \varphi'\}
\end{tikzcd}$$

$$\begin{tikzcd}
 & Z \; U \; \{\varphi U \varphi'\} \arrow[dl] \arrow[dr] & \\
Z \; U \; \{\varphi'\} & & Z \; U \; \{\varphi, X(\varphi U \varphi')\}
\end{tikzcd}$$

Construction du graphe de réduction



Exemple de réduction d'une formule

$(pUq) \wedge (\neg rUs)$

$\boxed{(pUq) \wedge (\neg rUs)}$

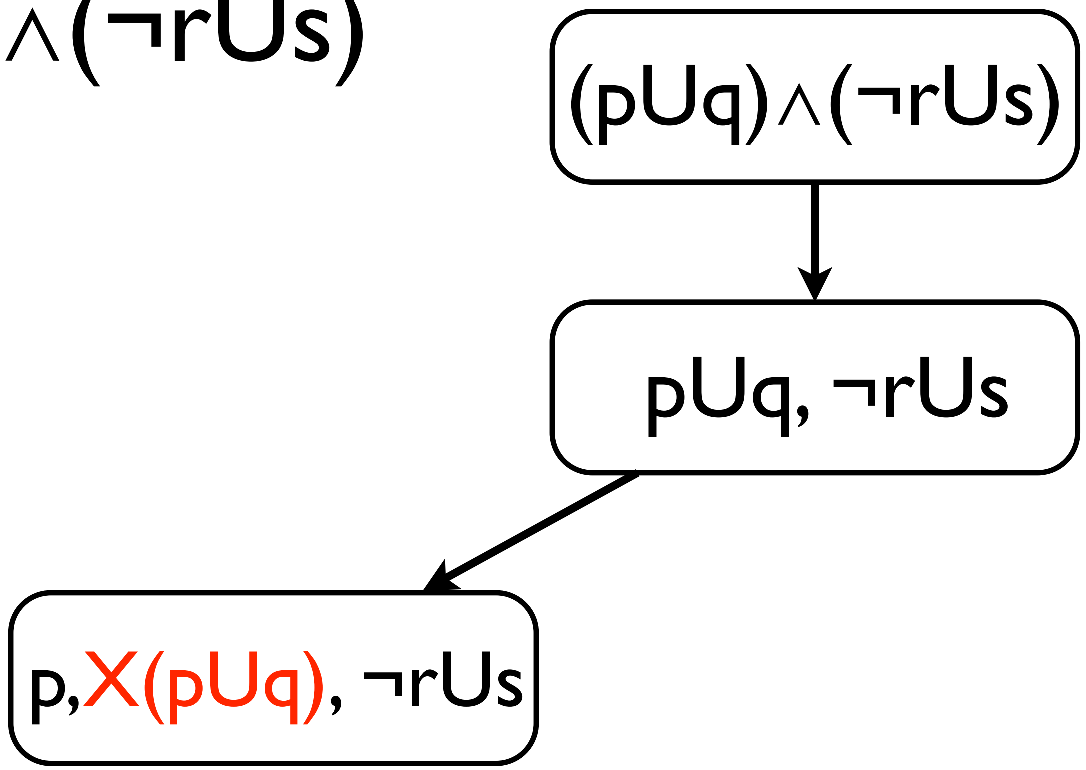
Exemple de réduction d'une formule

$(pUq) \wedge (\neg rUs)$

$$
\begin{tikzcd}
\boxed{(pUq) \wedge (\neg rUs)} \arrow[d] \\
\boxed{pUq,\ \neg rUs}
\end{tikzcd}
$$

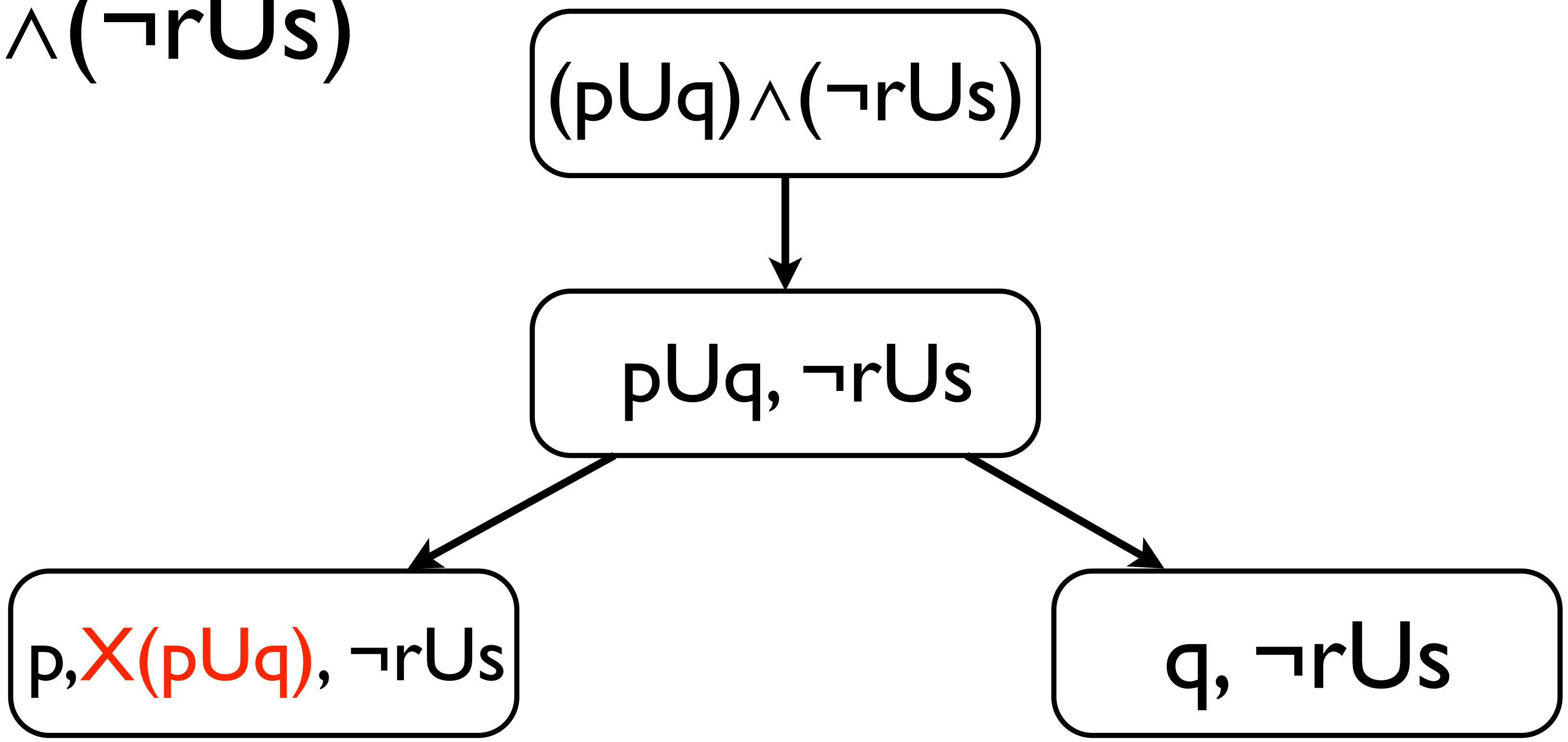
Exemple de réduction d'une formule

$(pUq) \wedge (\neg rUs)$



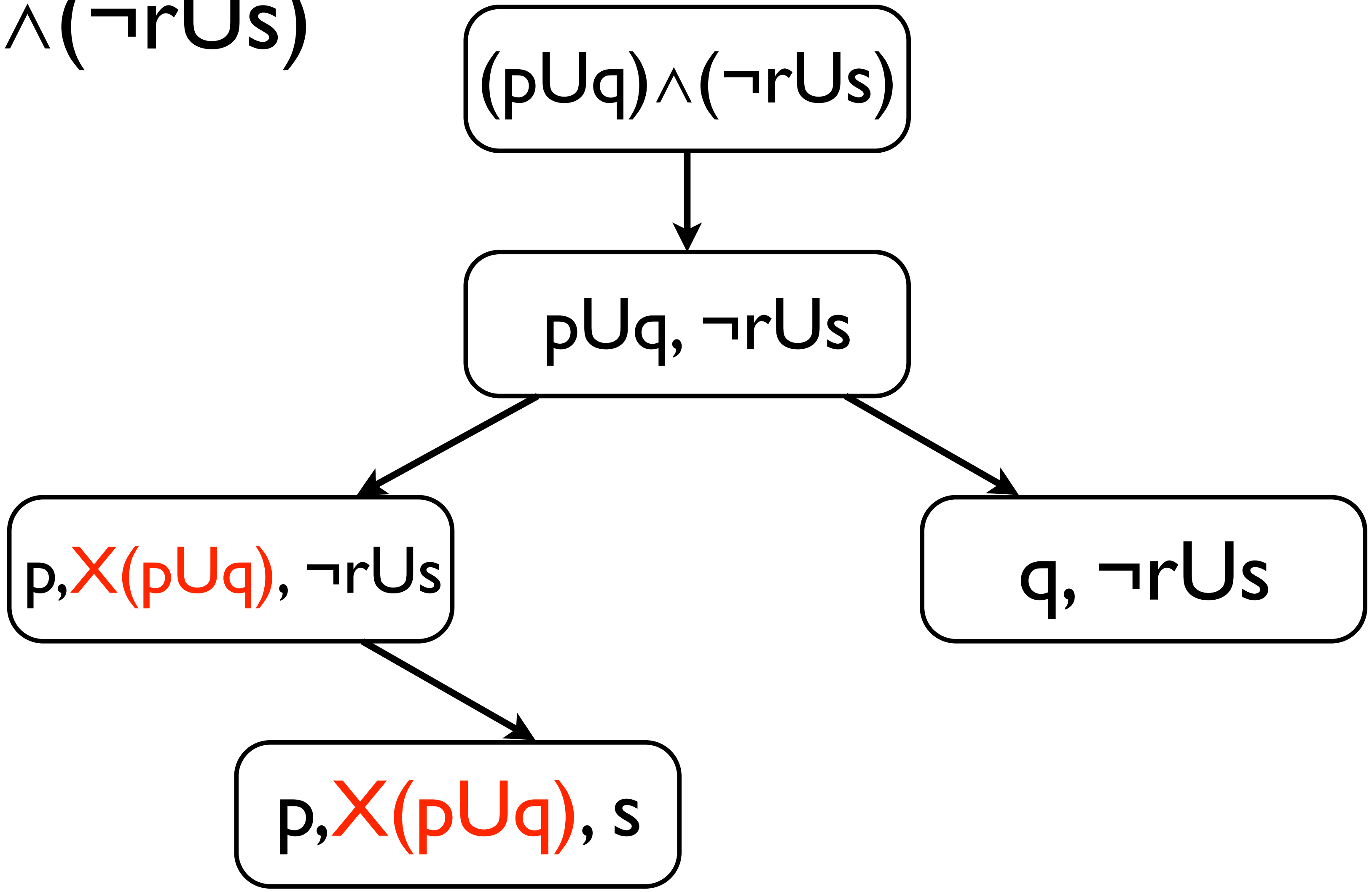
Exemple de réduction d'une formule

$(pUq) \wedge (\neg rUs)$



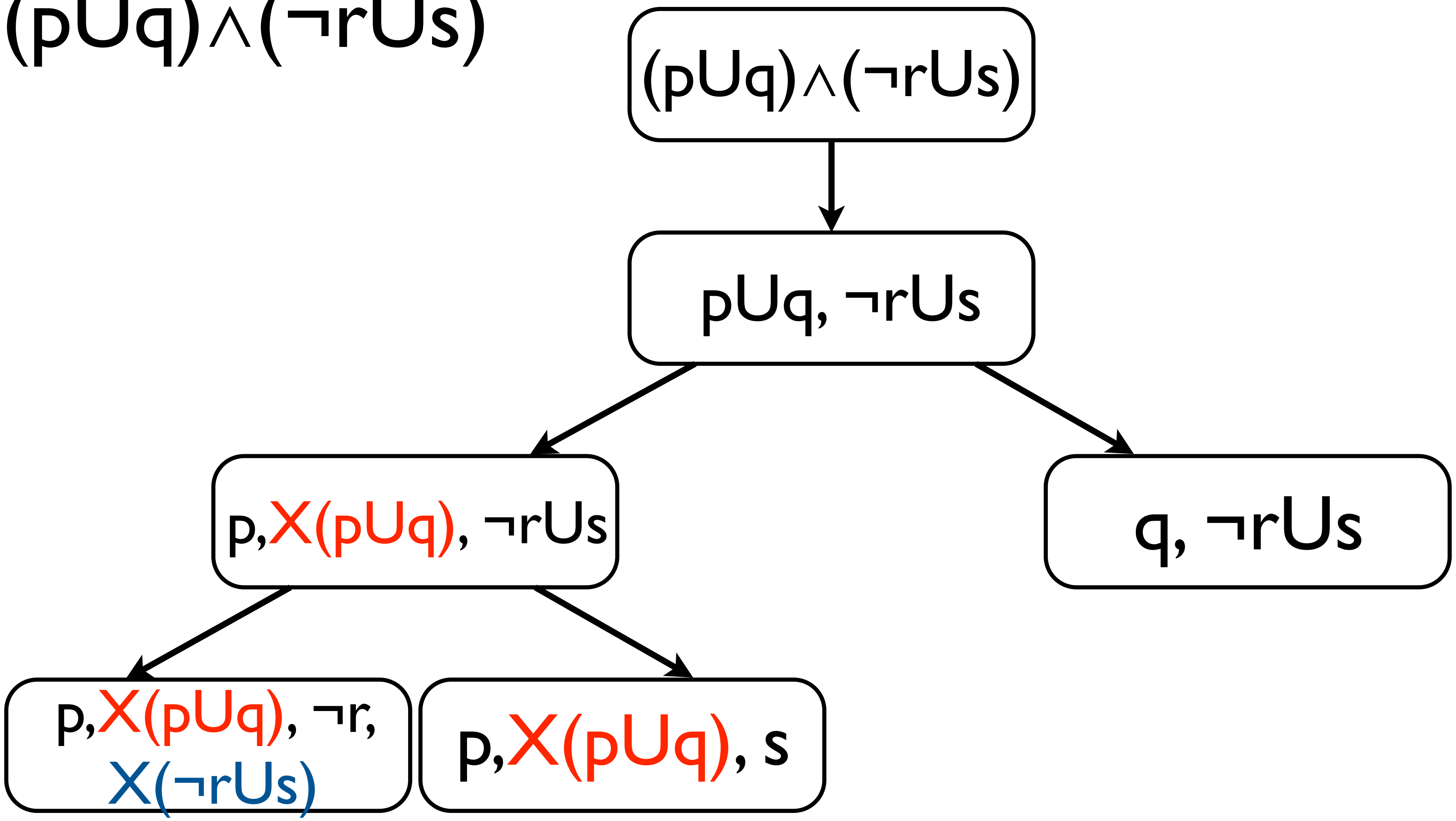
Exemple de réduction d'une formule

$(pUq) \wedge (\neg rUs)$



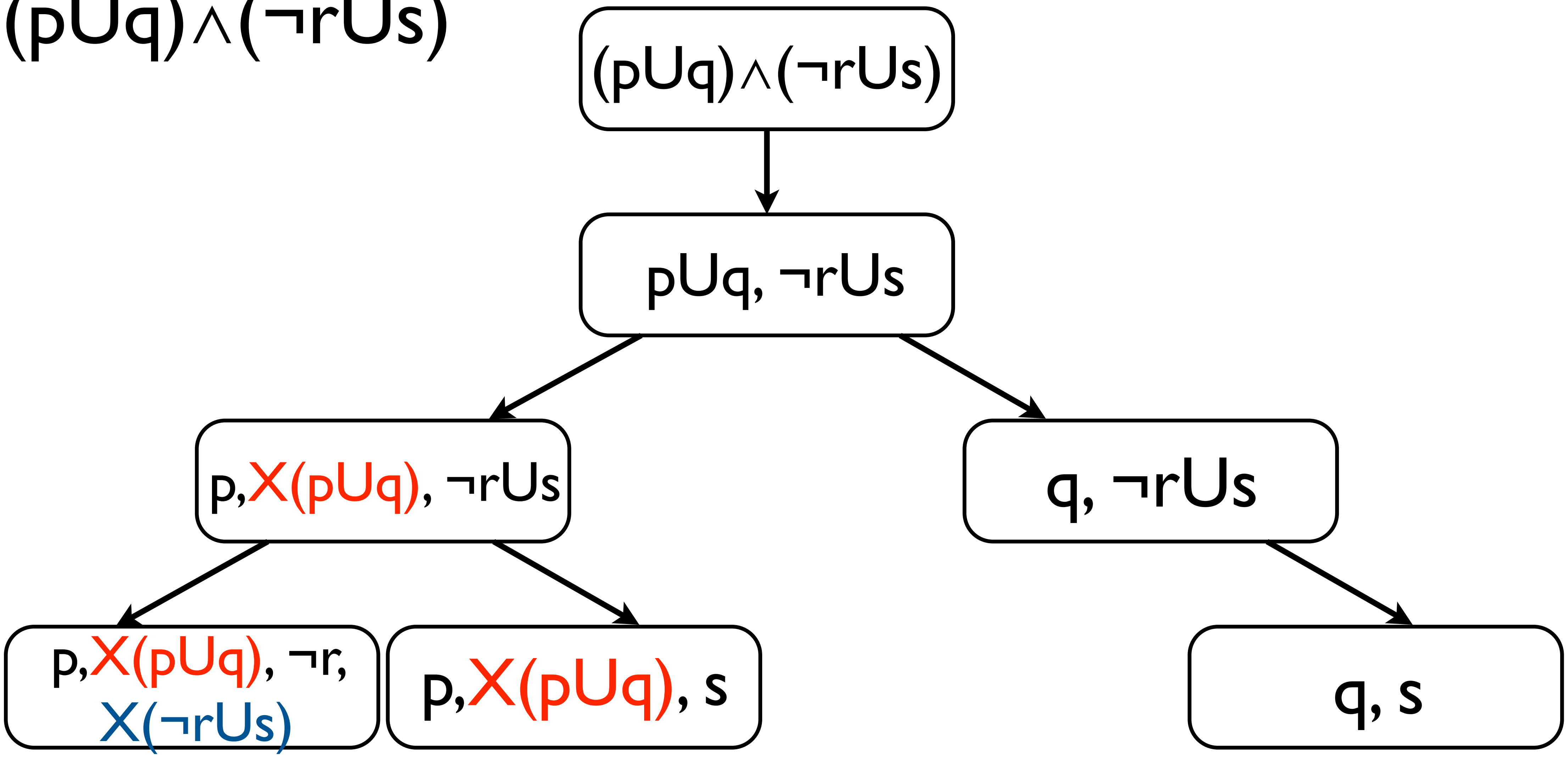
Exemple de réduction d'une formule

$(pUq) \wedge (\neg rUs)$



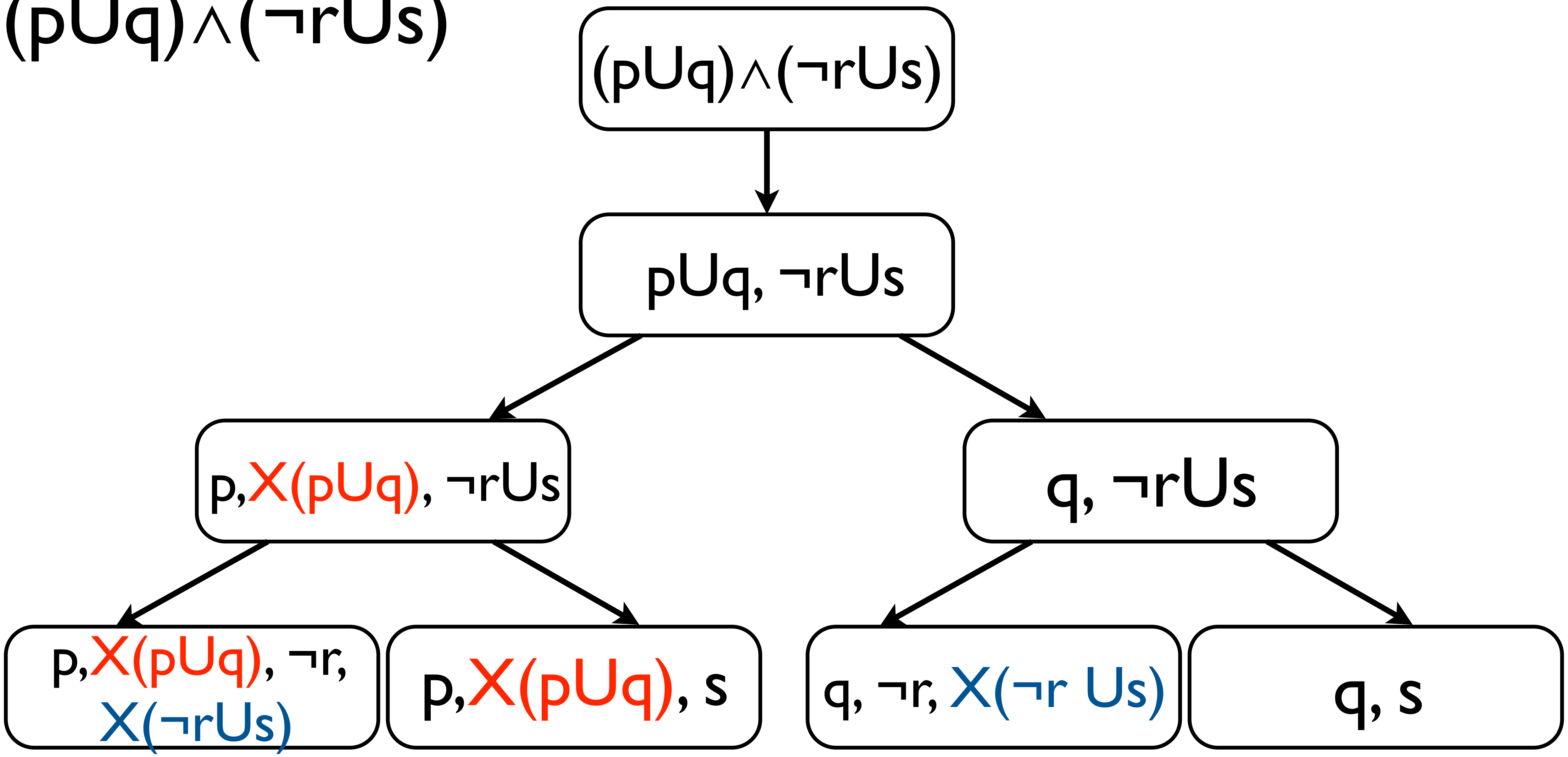
Exemple de réduction d'une formule

$(pUq) \wedge (\neg rUs)$



Exemple de réduction d'une formule

$(pUq) \wedge (\neg rUs)$



Transformer φ en un automate de Büchi

I. Automates de Büchi généralisés

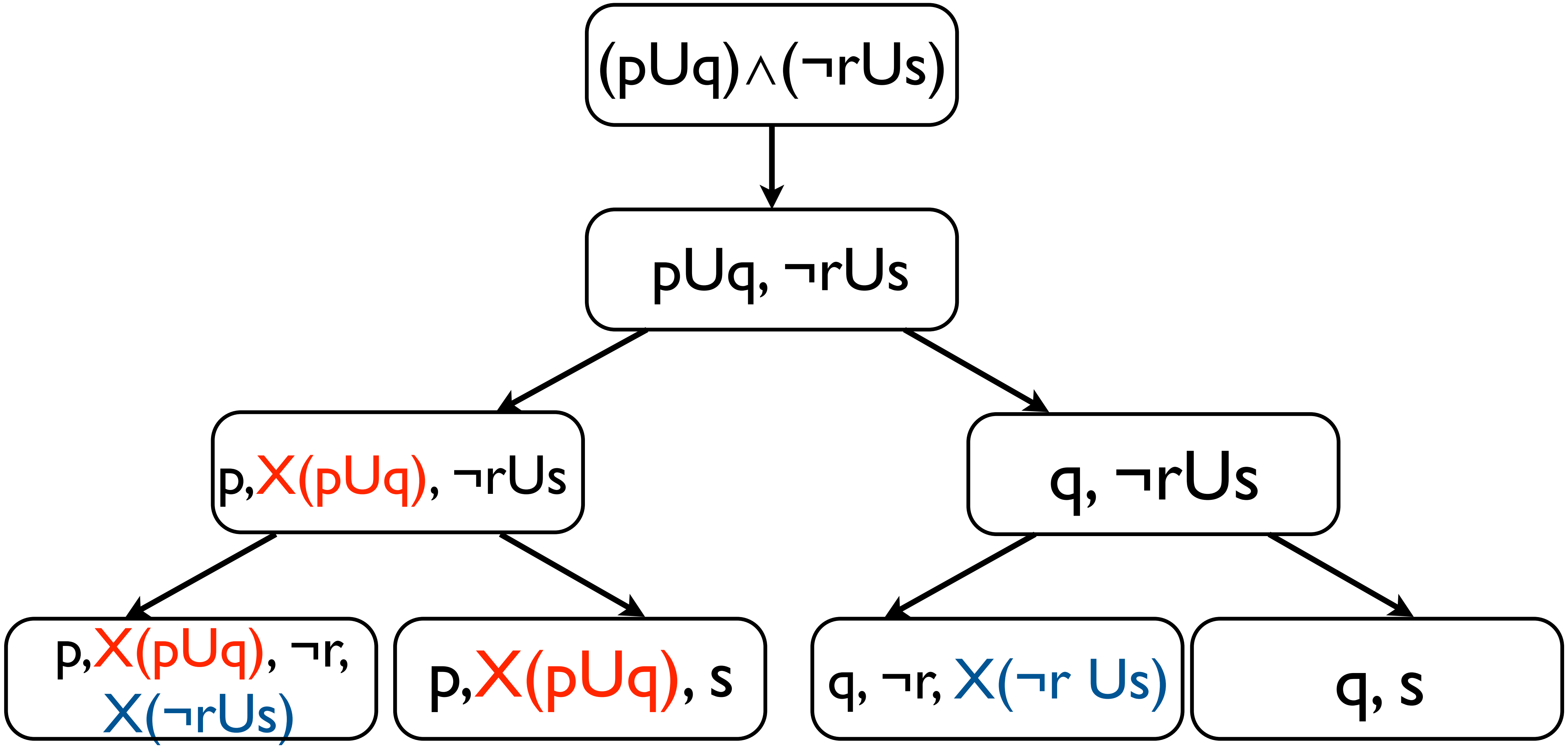
II. Réduire la formule

  II.I. Forme normale négative

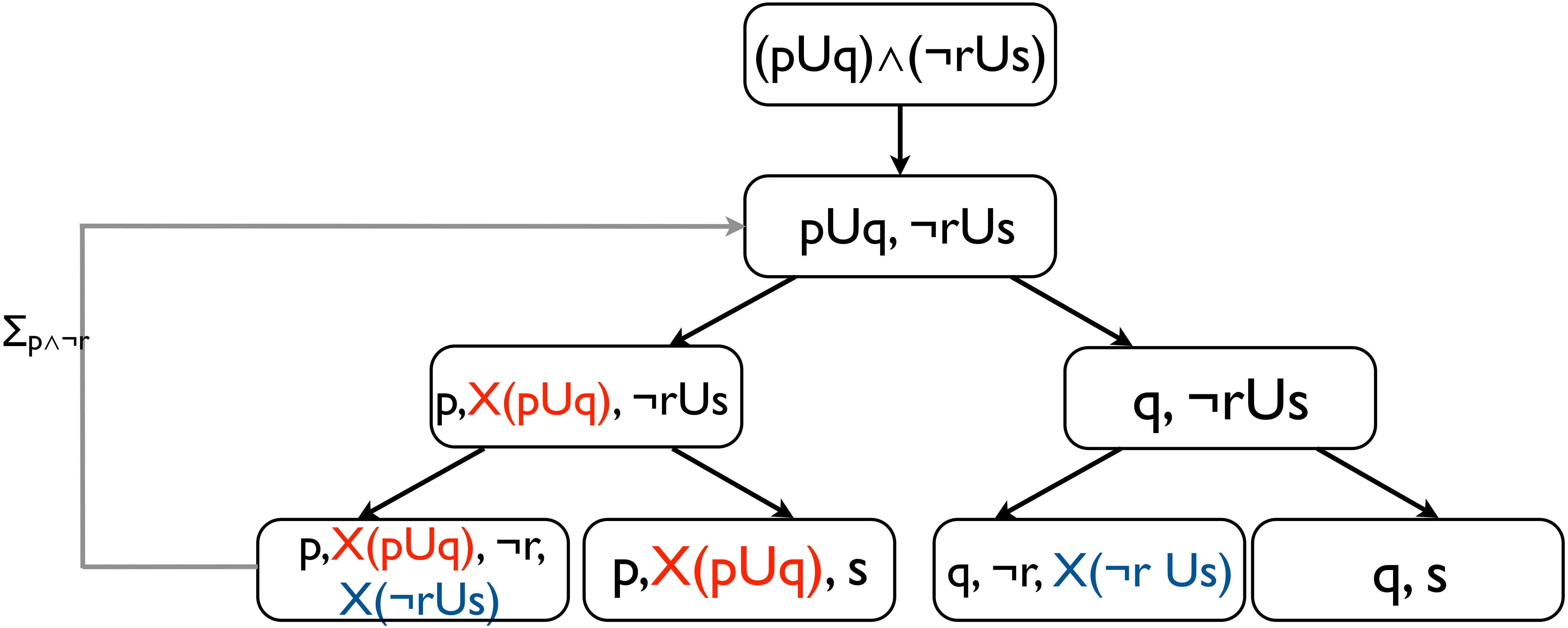
  II.II. Réduire les connecteurs temporels

III. Transformation en automate de Büchi généralisé

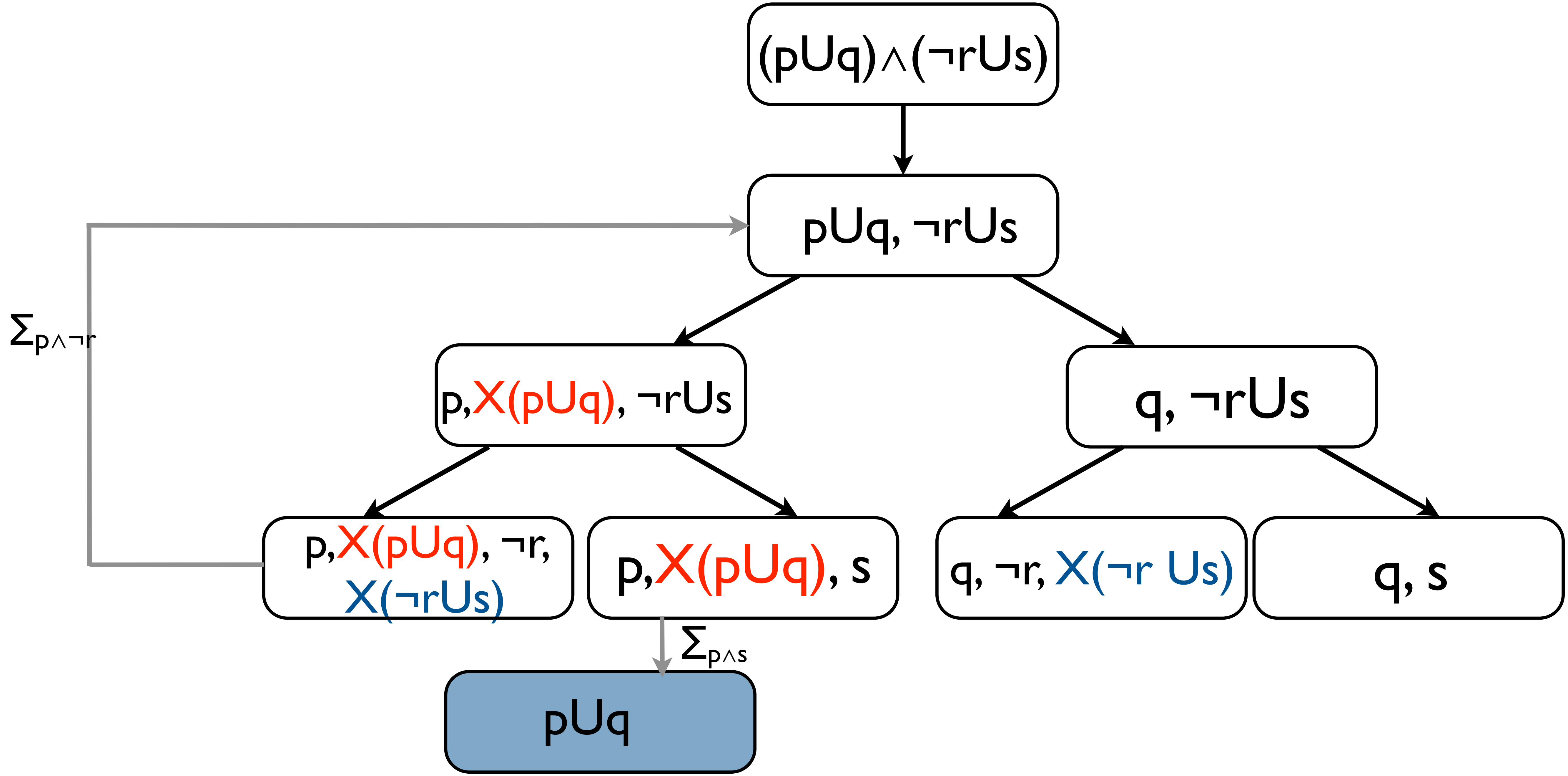
Exemple (suite)



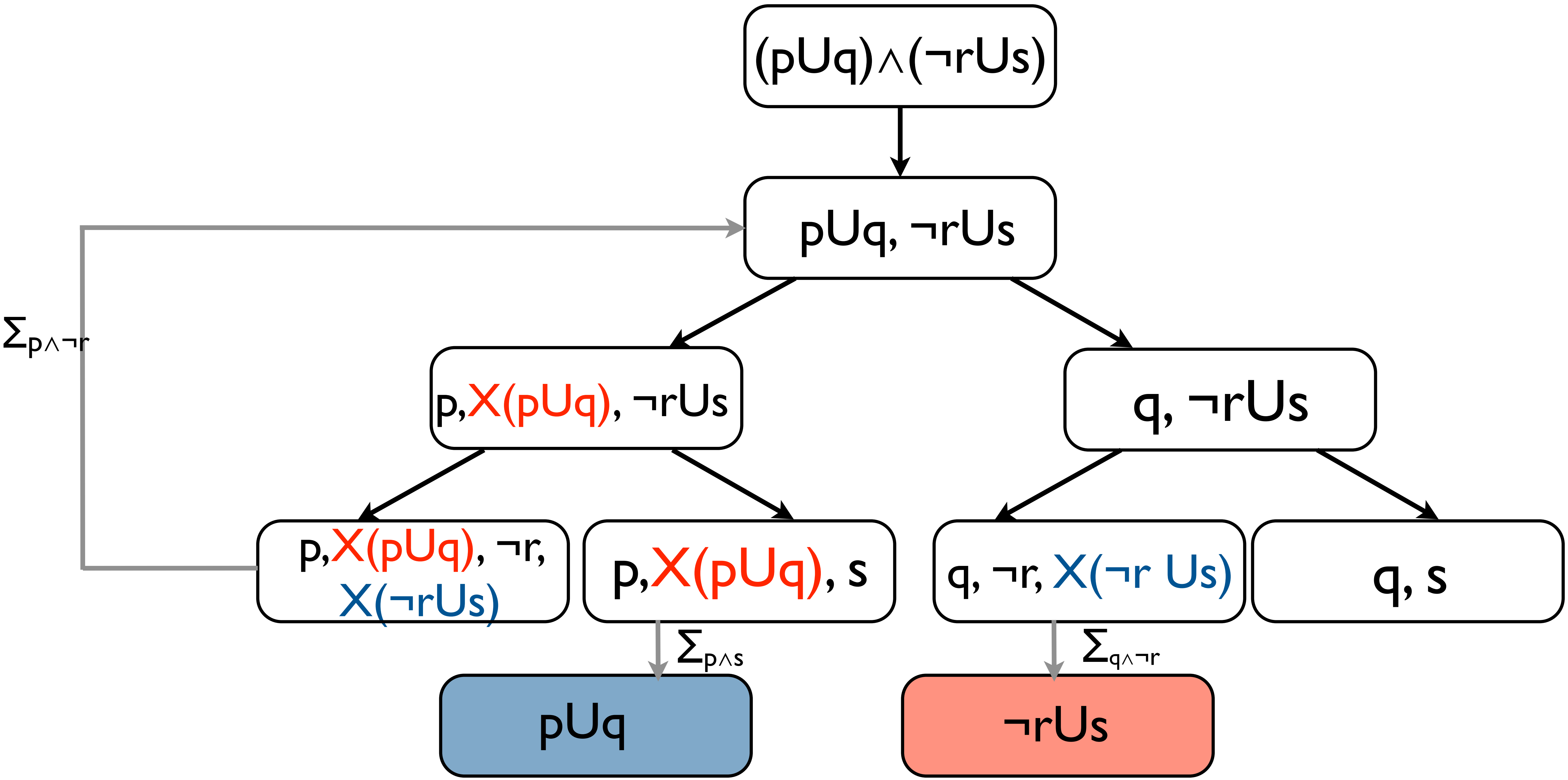
Exemple (suite)



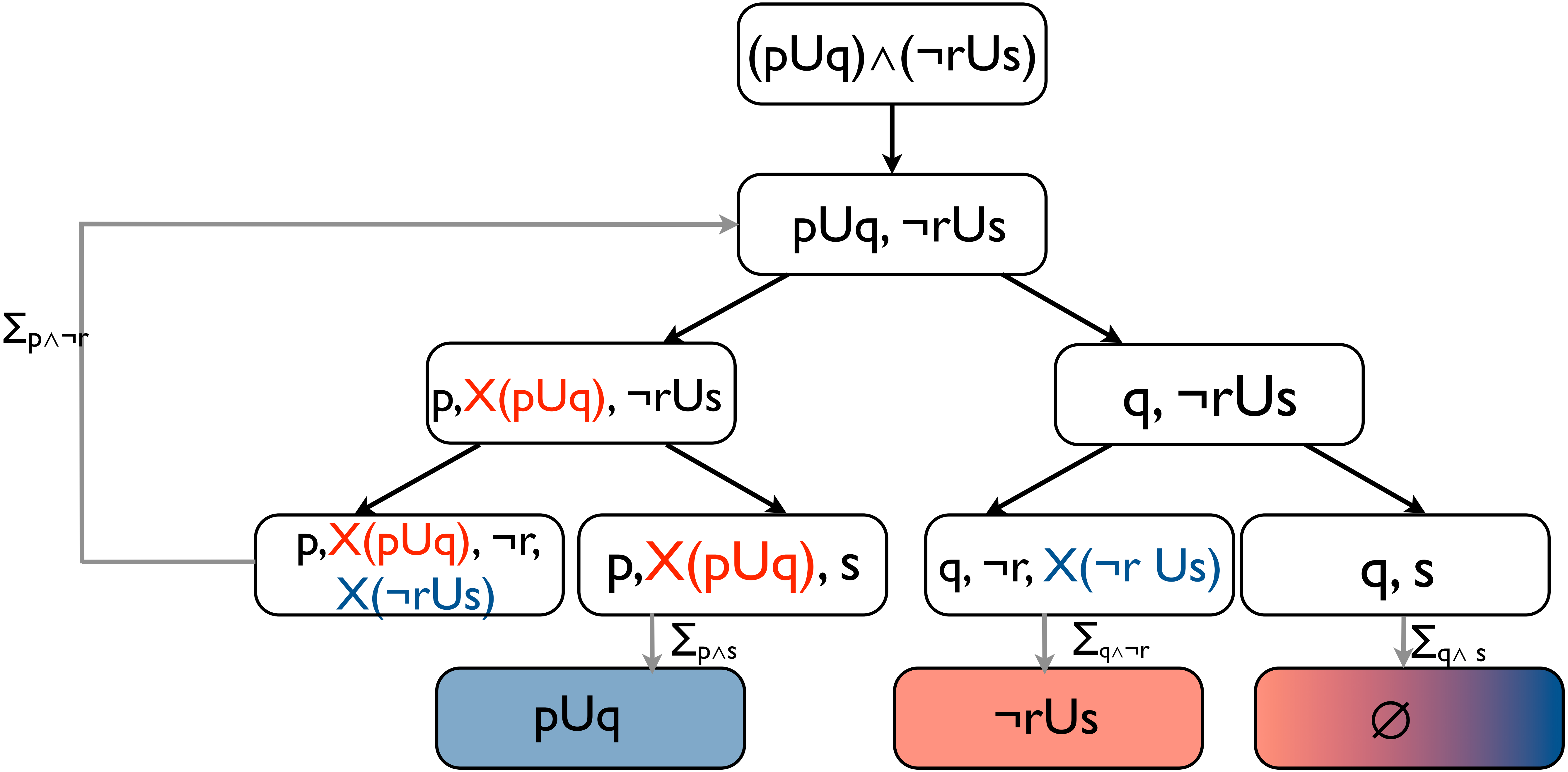
Exemple (suite)



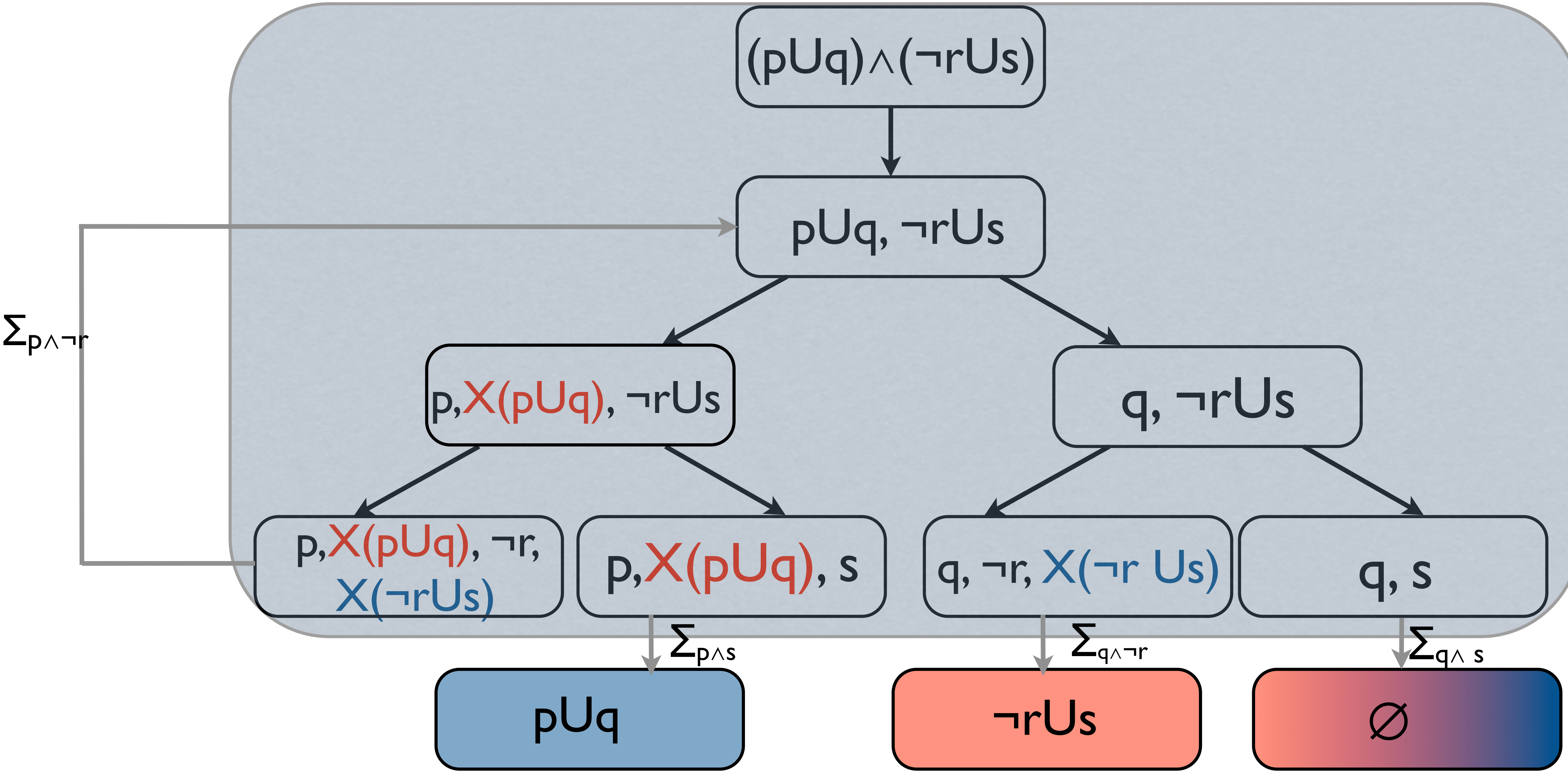
Exemple (suite)



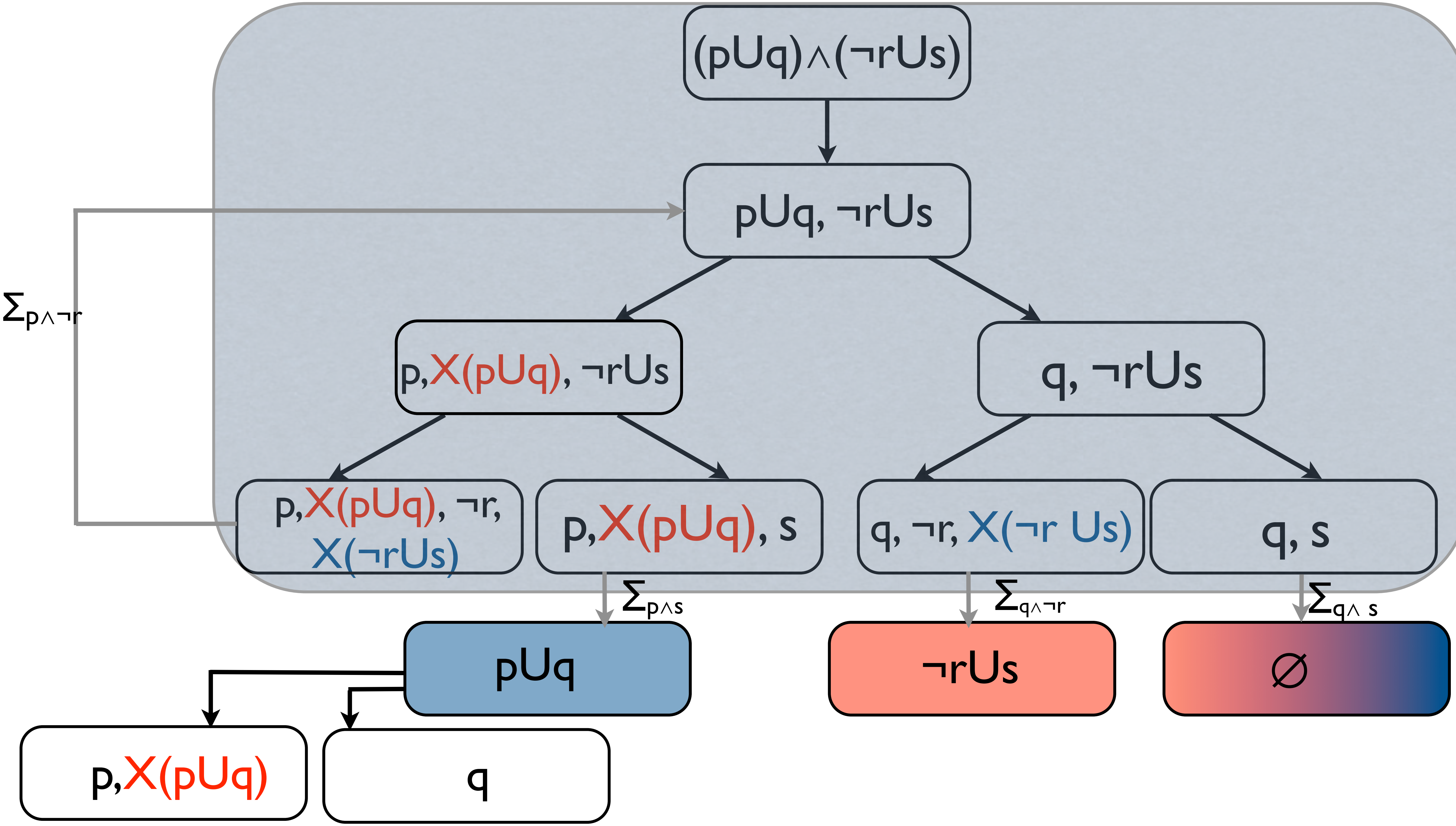
Exemple (suite)



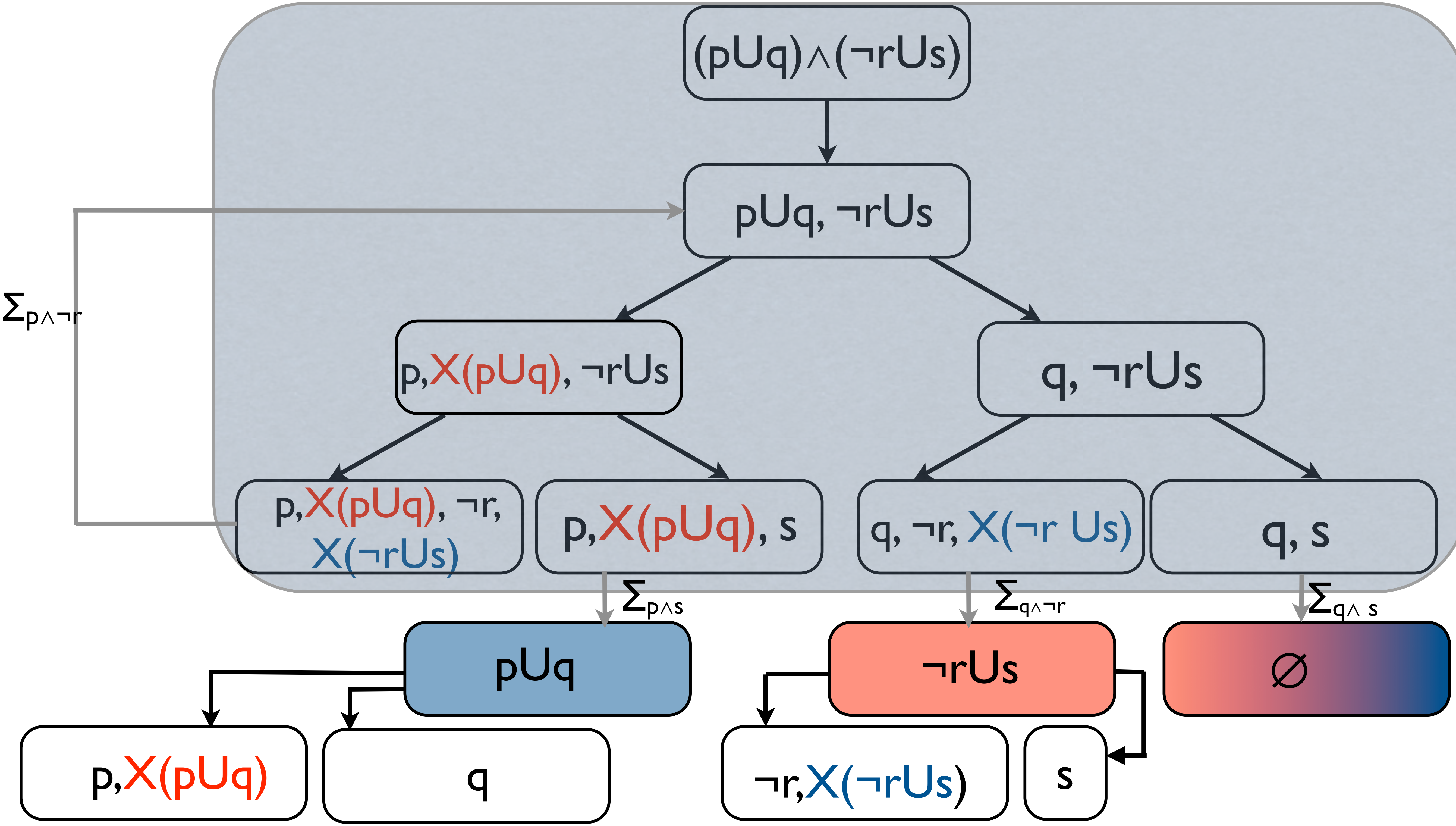
Exemple (suite)



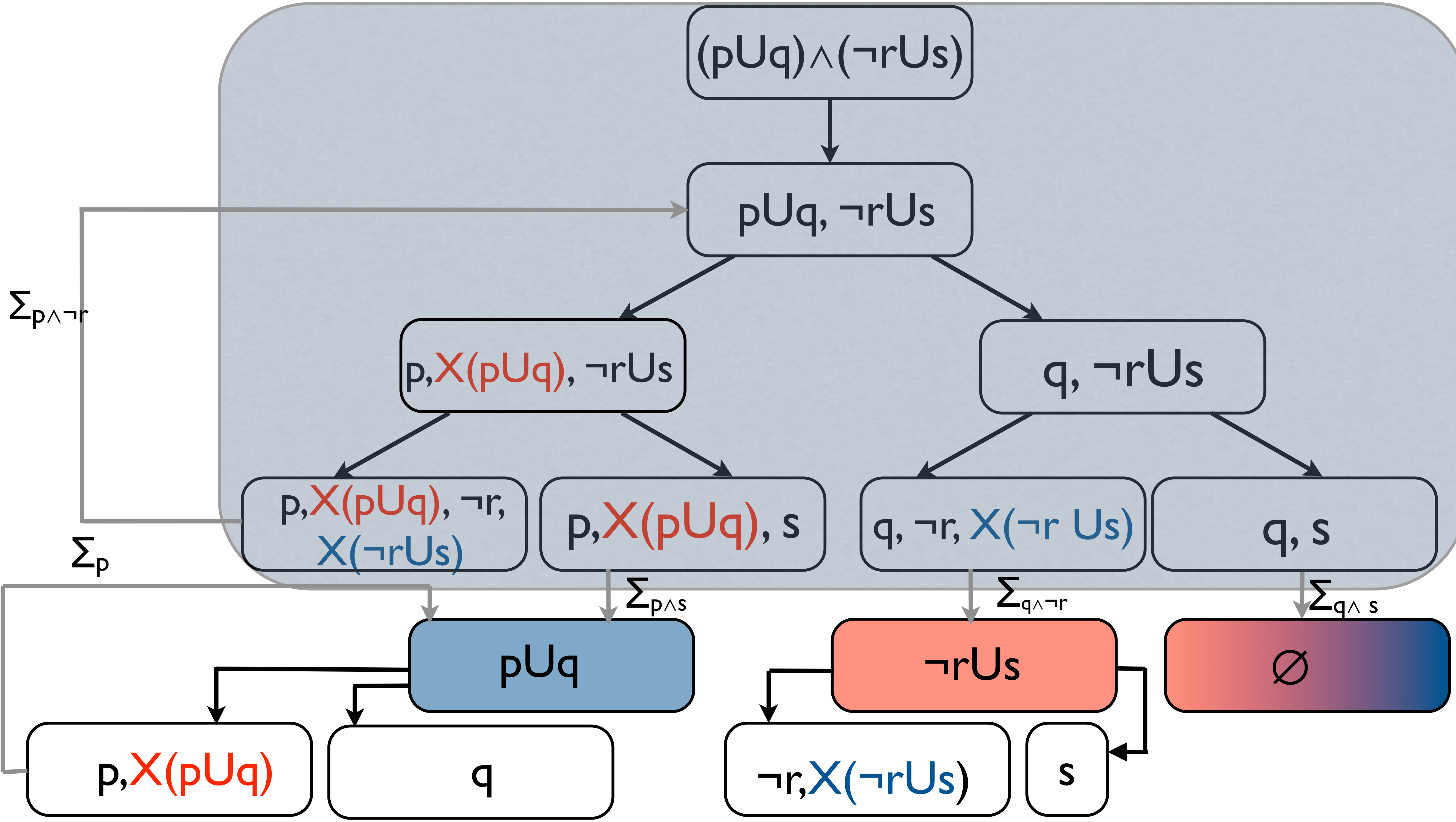
Exemple (suite)



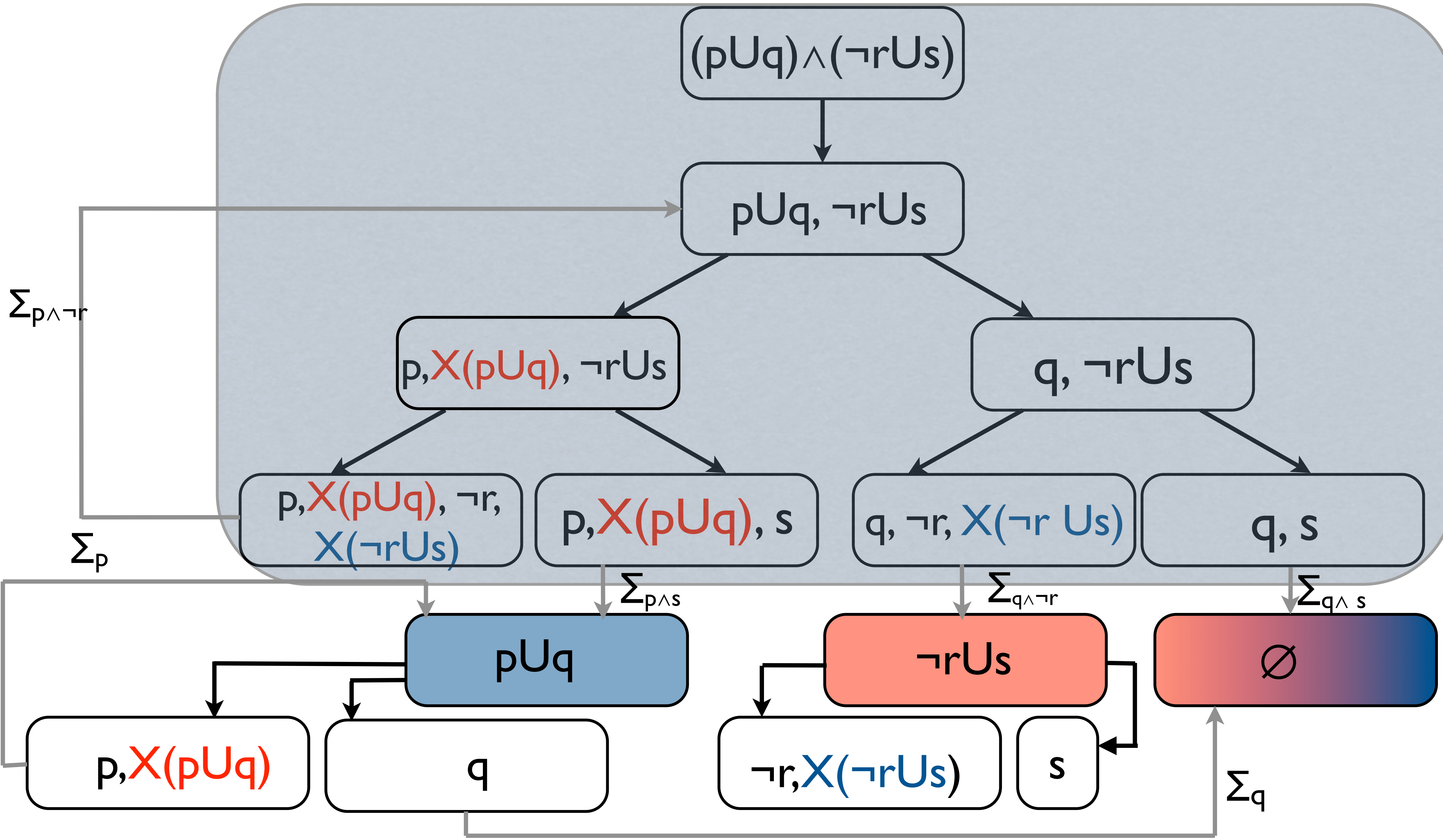
Exemple (suite)



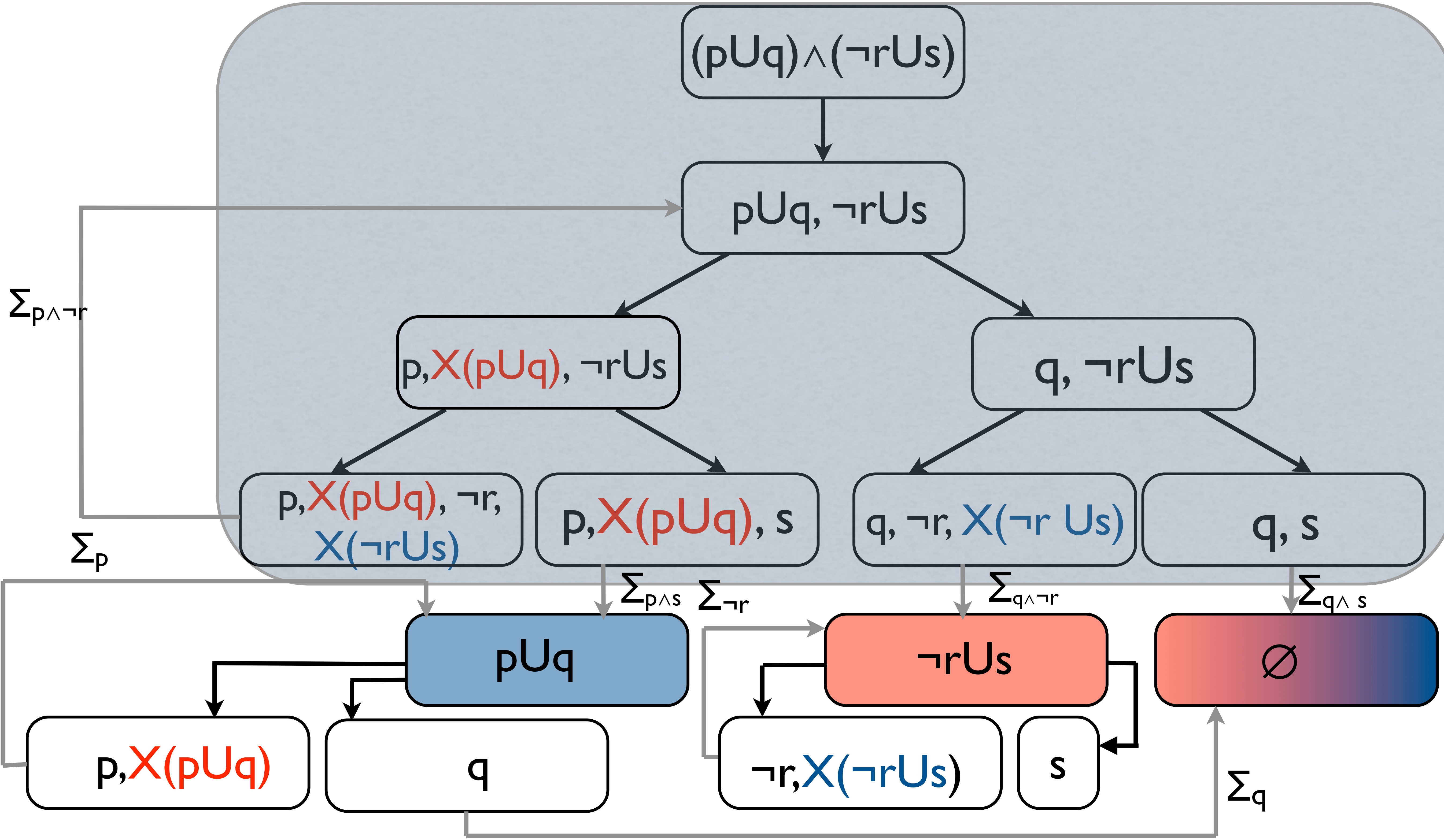
Exemple (suite)



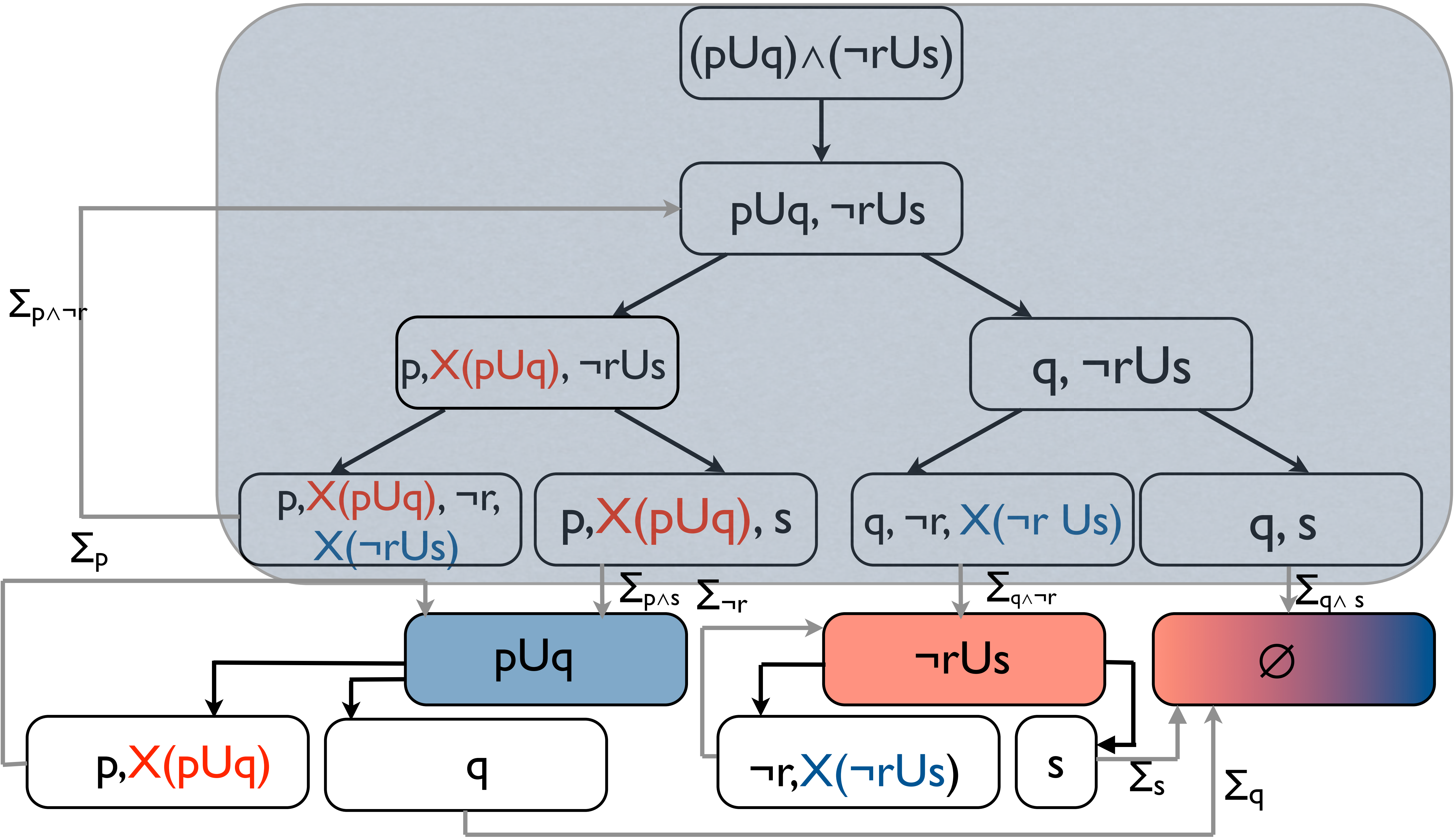
Exemple (suite)



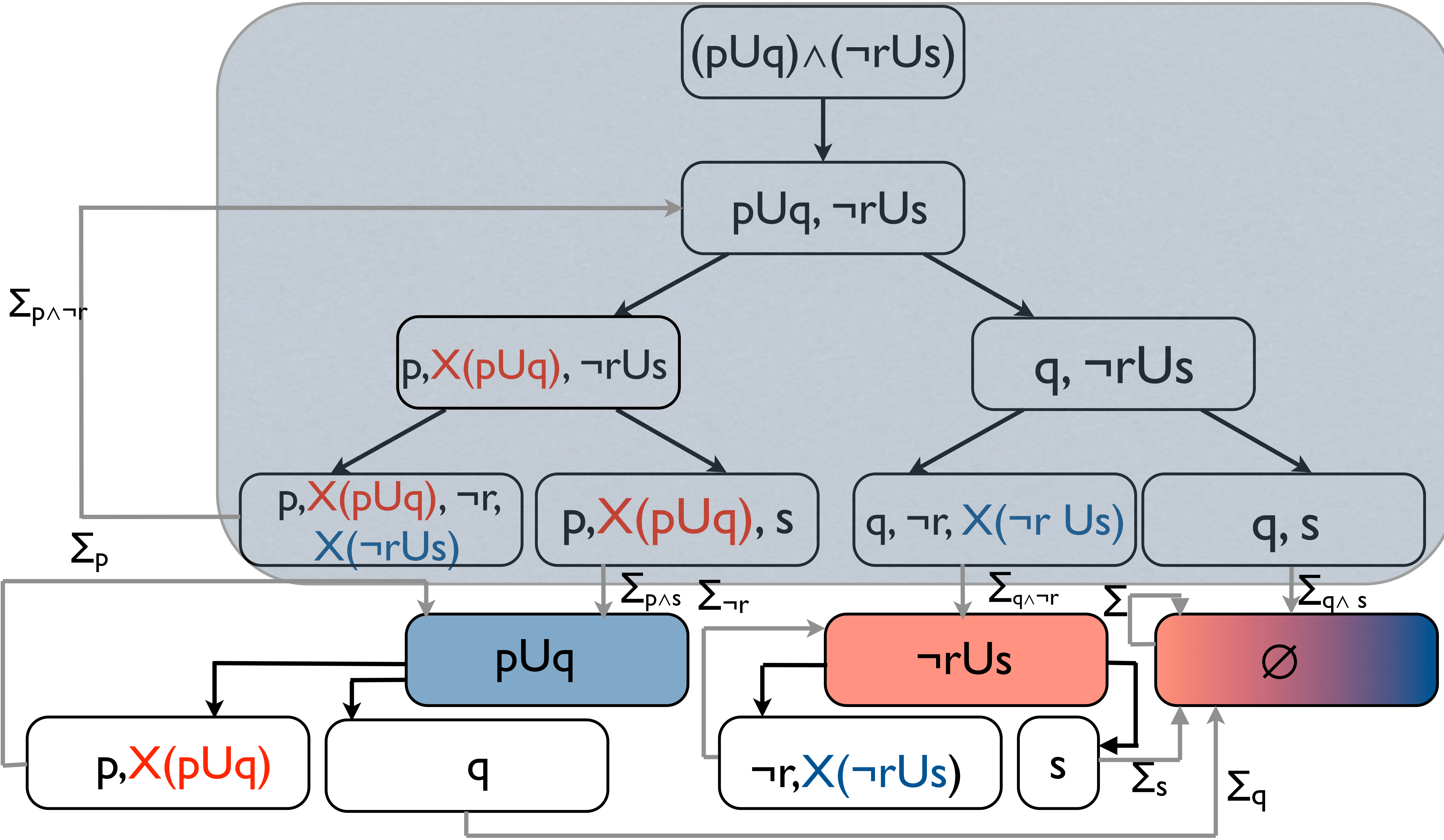
Exemple (suite)



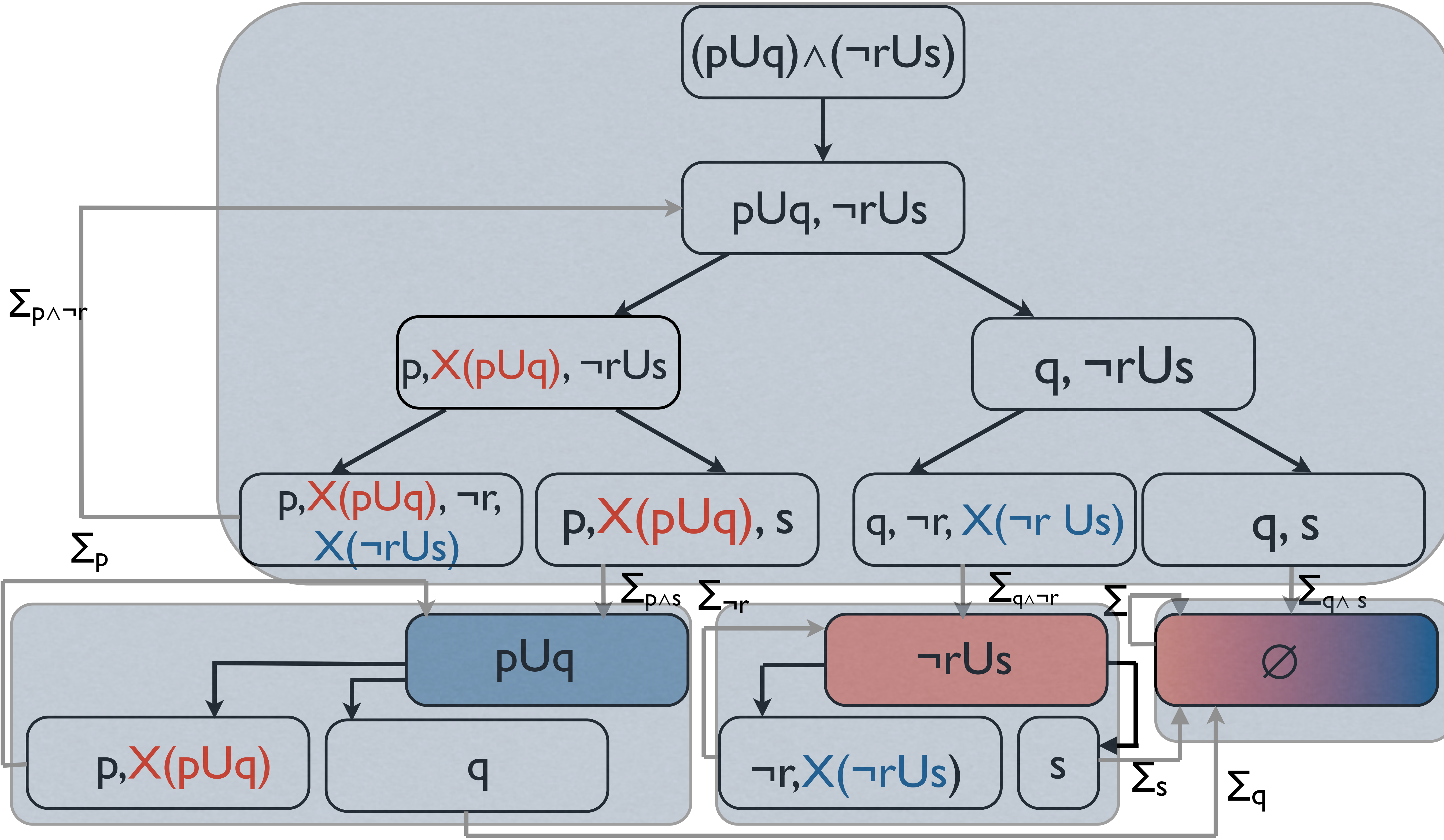
Exemple (suite)



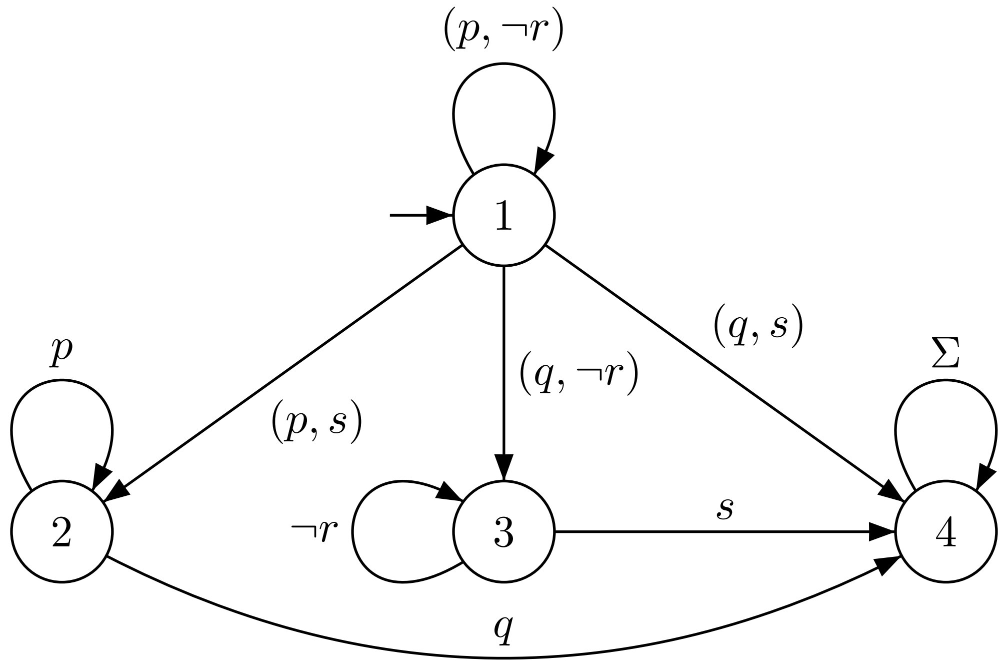
Exemple (suite)



Exemple (suite)



Exemple (fin)



$F_{pUq}=\{3,4\}$ $F_{\neg rUs}=\{2,4\}$

Model-Checking LTL : approche par automates

- Donnée: Structure de Kripke M, formule LTL φ.
- Etapes de l'algorithme :
  - Transformer M en un automate A_M tel que $L(A_M)=[\![M]\!]$
  - Transformer φ en un automate $A_{\neg\varphi}$ tel que $L(A_{\neg\varphi})=[\![\neg\varphi]\!]$
  - Tester si $L(A_M)\cap L(A_{\neg\varphi})=\emptyset$.

Model-Checking LTL : approche par automates

- Donnée: Structure de Kripke M, formule LTL φ.
- Etapes de l'algorithme :
  - Transformer M en un automate A_M tel que $L(A_M)=[\![M]\!]$
  - Transformer φ en un automate $A_{\neg\varphi}$ tel que $L(A_{\neg\varphi})=[\![\neg\varphi]\!]$ ✔
  - Tester si $L(A_M)\cap L(A_{\neg\varphi})=\emptyset$.

Model-Checking LTL : approche par automates

- Donnée: Structure de Kripke M, formule LTL φ.
- Etapes de l'algorithme :
  - Transformer M en un automate A_M tel que $L(A_M)=[\![M]\!]$
  - Transformer φ en un automate $A_{\neg\varphi}$ tel que $L(A_{\neg\varphi})=[\![\neg\varphi]\!]$ ✔
  - Tester si $L(A_M)\cap L(A_{\neg\varphi})=\varnothing$.

Transformer M en un automate de Büchi

- Soit $M=(Q, T, A, q_0, AP, l)$ une structure de Kripke. On construit un automate de Büchi $B= (Q', \Sigma, q'0, T', F)$ tel que $L(B)=[\![M]\!]$:
- Idée: on fait «basculer» les étiquettes des états vers les transitions + tous les états sont acceptants
  - $\Sigma=2^{AP}$
  - $Q'=T \cup \{q_0'\}$
  - $F=Q'$
  - Soit $t=(q_0,q)\in T$, alors $(q_0',l(q_0),t)\in T'$
  - Soient $t=(q,q')$ et $t'=(q',q'')\in T$, alors $(t,l(q'),t')\in T'$

Exemple

- au tableau

Model-Checking LTL : approche par automates

- Donnée: Structure de Kripke M, formule LTL φ.
- Etapes de l'algorithme :
  - Transformer M en un automate A_M tel que $L(A_M)=[\![M]\!]$
  - Transformer φ en un automate $A_{\neg\varphi}$ tel que $L(A_{\neg\varphi})=[\![\neg\varphi]\!]$ ✔
  - Tester si $L(A_M)\cap L(A_{\neg\varphi})=\varnothing$.

Model-Checking LTL : approche par automates

- Donnée: Structure de Kripke M, formule LTL φ.
- Etapes de l'algorithme :
  - Transformer M en un automate A_M tel que $L(A_M)=[\![M]\!]$ ✔
  - Transformer φ en un automate $A_{\neg\varphi}$ tel que $L(A_{\neg\varphi})=[\![\neg\varphi]\!]$ ✔
  - Tester si $L(A_M)\cap L(A_{\neg\varphi})=\emptyset$.

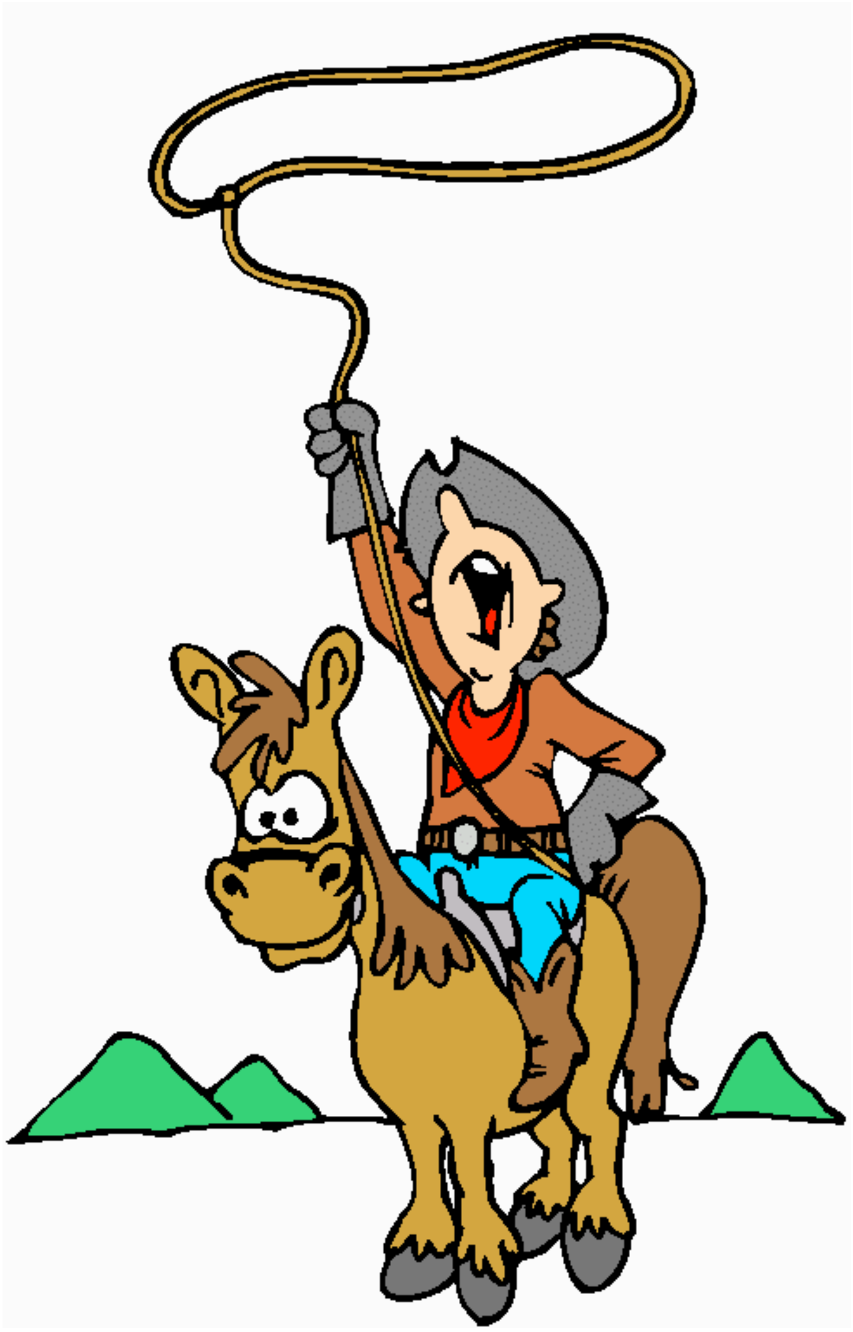
Model-Checking LTL : approche par automates

- Donnée: Structure de Kripke M, formule LTL φ.
- Etapes de l'algorithme :
  - Transformer M en un automate A_M tel que $L(A_M)=[\![M]\!]$ ✔
  - Transformer φ en un automate $A_{\neg\varphi}$ tel que $L(A_{\neg\varphi})=[\![\neg\varphi]\!]$ ✔
  - Tester si $L(A_M) \cap L(A_{\neg\varphi})=\emptyset$.

Tester le vide de l'intersection

- Construire l'automate $A_M \otimes A_{\neg\varphi}$ tel que $L(A_M \otimes A_{\neg\varphi}) = L(A_M) \cap L(A_{\neg\varphi})$.
- Rechercher s'il existe un mot accepté par $A_M \otimes A_{\neg\varphi}$ ≡ rechercher un chemin d'un état initial vers un état acceptant + boucle sur cet état acceptant.

Model-Checking LTL: catching bugs with a lasso



Model-Checking LTL : approche par automates

- Donnée: Structure de Kripke M, formule LTL φ.
- Etapes de l'algorithme :
  - Transformer M en un automate A_M tel que $L(A_M)=[\![M]\!]$ ✔
  - Transformer φ en un automate $A_{\neg\varphi}$ tel que $L(A_{\neg\varphi})=[\![\neg\varphi]\!]$ ✔
  - Tester si $L(A_M)\cap L(A_{\neg\varphi})=\varnothing$.

Model-Checking LTL : approche par automates

- Donnée: Structure de Kripke M, formule LTL φ.
- Etapes de l'algorithme :
  - Transformer M en un automate A_M tel que $L(A_M)=[\![M]\!]$ ✔
  - Transformer φ en un automate $A_{\neg\varphi}$ tel que $L(A_{\neg\varphi})=[\![\neg\varphi]\!]$ ✔
  - Tester si $L(A_M) \cap L(A_{\neg\varphi})=\emptyset$. ✔

Model-Checking LTL : approche par automates

- Donnée: Structure de Kripke M, formule LTL φ.
- Etapes de l'algorithme :
  - Transformer M en un automate A_M tel que $L(A_M)=[\![M]\!]$ ✔ O(|M|)
  - Transformer φ en un automate $A_{\neg\varphi}$ tel que $L(A_{\neg\varphi})=[\![\neg\varphi]\!]$ ✔
  - Tester si $L(A_M) \cap L(A_{\neg\varphi})=\emptyset$. ✔

Model-Checking LTL : approche par automates

- Donnée: Structure de Kripke M, formule LTL φ.
- Etapes de l'algorithme :
  - Transformer M en un automate A_M tel que $L(A_M)=[\![M]\!]$ ✔ $O(|M|)$
  - Transformer φ en un automate $A_{\neg\varphi}$ tel que $L(A_{\neg\varphi})=[\![\neg\varphi]\!]$ ✔ $O(2^{|\varphi|})$
  - Tester si $L(A_M) \cap L(A_{\neg\varphi})=\emptyset$. ✔

Model-Checking LTL : approche par automates

- Donnée: Structure de Kripke M, formule LTL φ.
- Etapes de l'algorithme :
  - Transformer M en un automate A_M tel que $L(A_M)=[\![M]\!]$ ✔ O(|M|)
  - Transformer φ en un automate $A_{\neg\varphi}$ tel que $L(A_{\neg\varphi})=[\![\neg\varphi]\!]$ ✔ $O(2^{|\varphi|})$
  - Tester si $L(A_M)\cap L(A_{\neg\varphi})=\varnothing$. ✔ $O(|M|.2^{|\varphi|})$

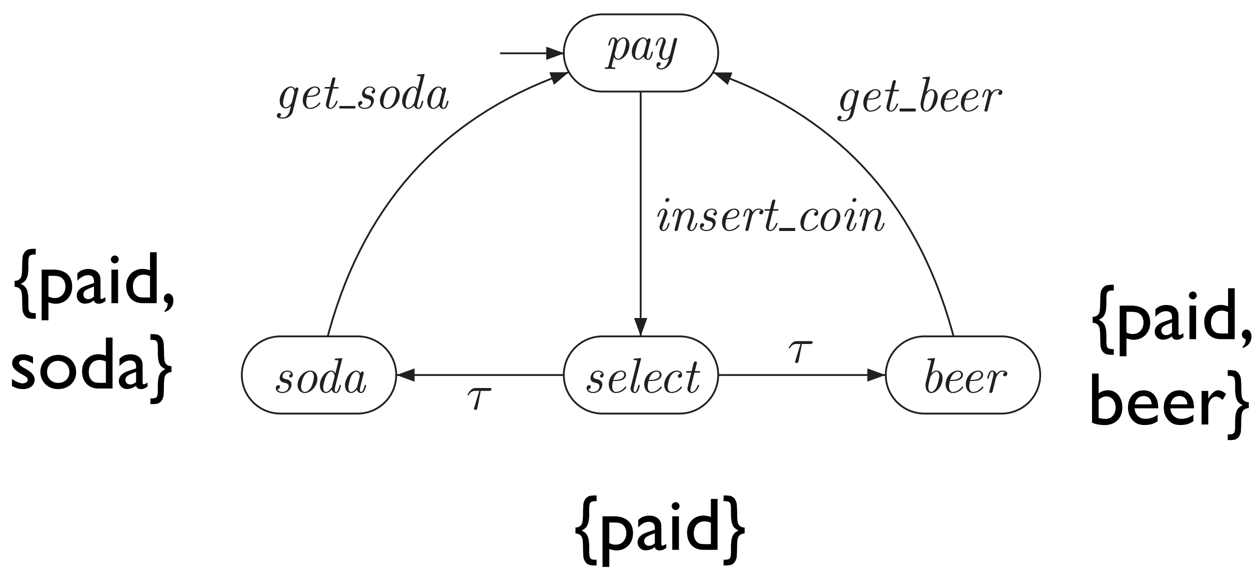
Model-Checking LTL: techniques à la volée

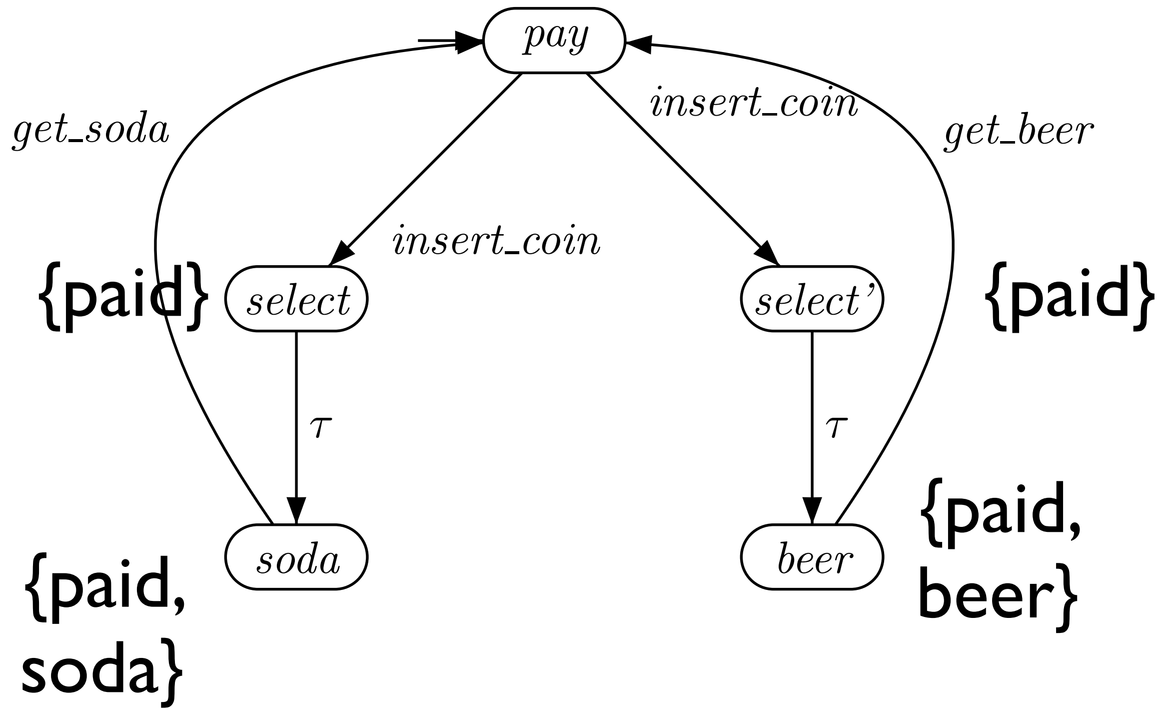
- Pas nécessaire de construire l'automate produit en entier
- On construit pas à pas, et on s'arrête lorsqu'on trouve un cycle (=contre-exemple).

3.2 CTL

Exprimer la possibilité

- La propriété «à chaque fois que *paid* est vérifié, il est possible d'obtenir une bière» n'est pas exprimable en LTL!





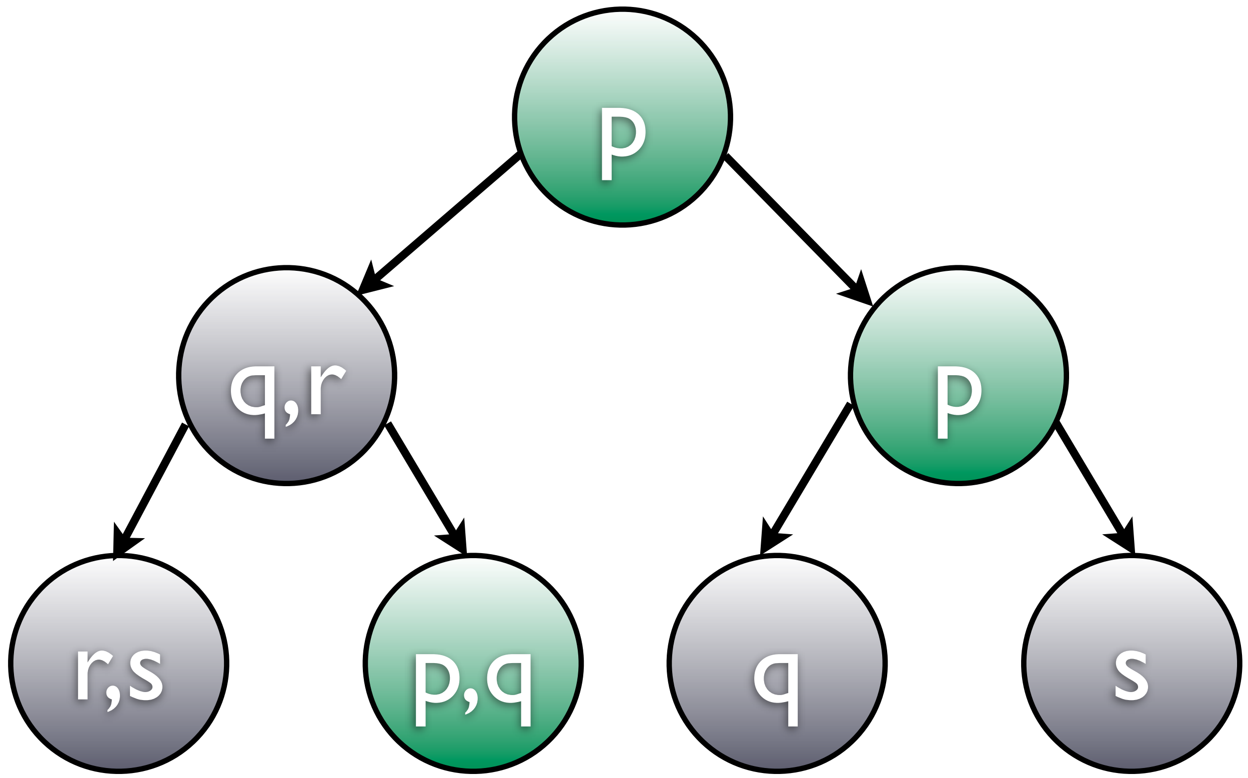
Les deux systèmes vérifient les mêmes propriétés LTL!!

Computational Tree Logic : CTL

[Clarke, Emerson 81]

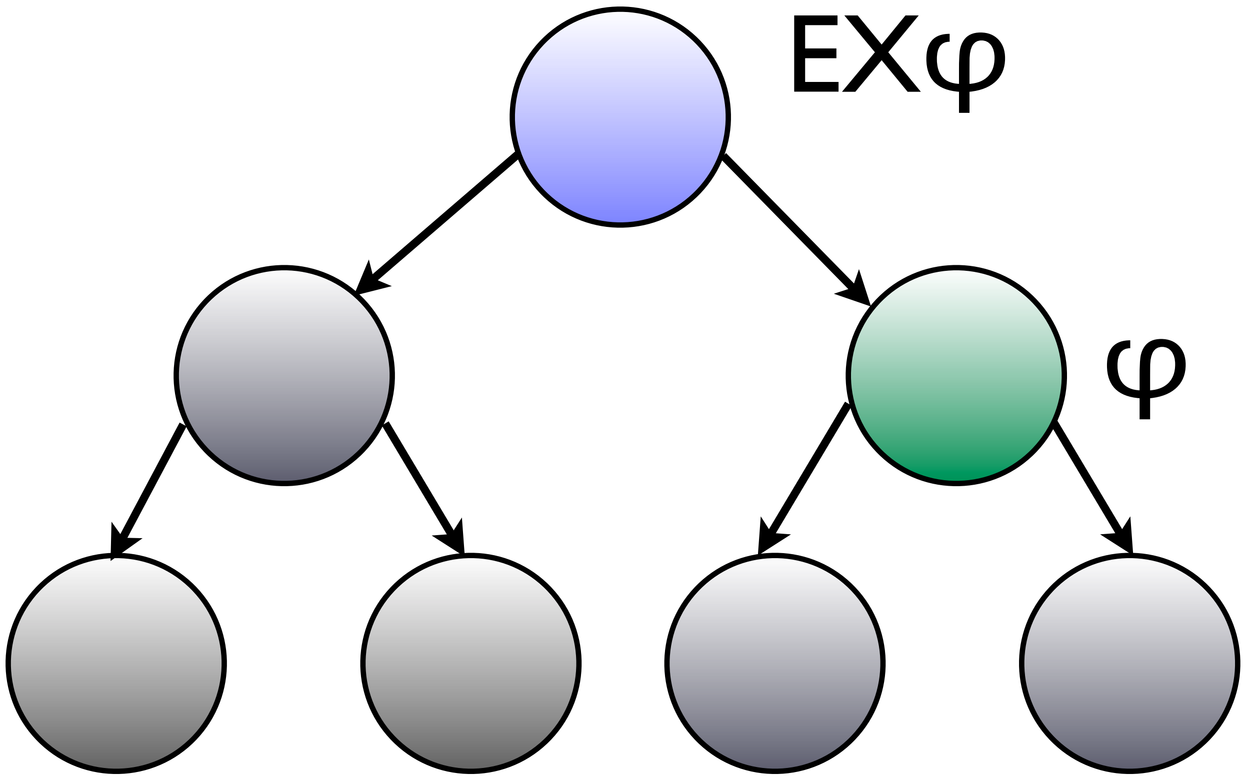
- Modèle des formules : état de l'arbre d'exécutions infini.
- M,s ⊧ φ ssi la formule φ est vérifiée à l'état s de la structure de Kripke M.
- On note S(φ) l'ensemble des états s t.q. M,s ⊧ φ
- Ajout de quantificateurs sur les chemins dans l'arbre : E et A.
- Défini inductivement sur la formule.

CTL: syntaxe et sémantique



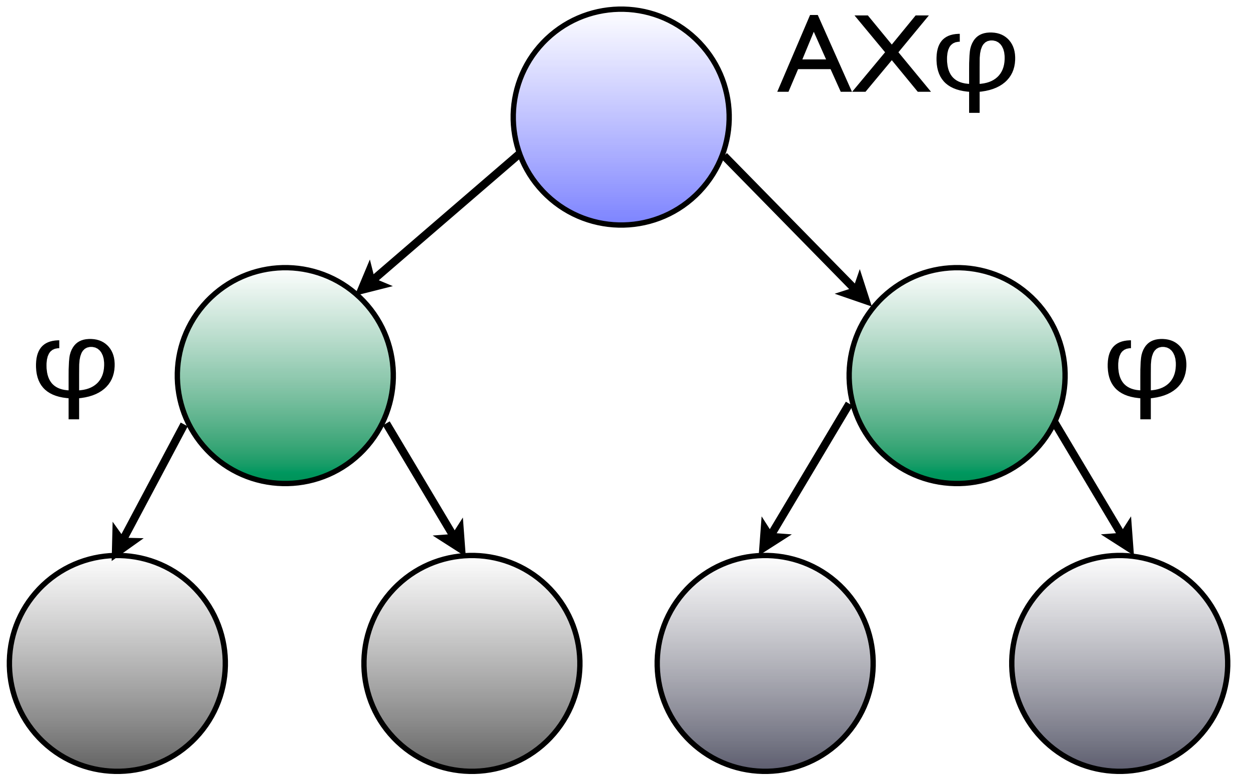
s ⊧ p ssi p∈l(s)

CTL: syntaxe et sémantique



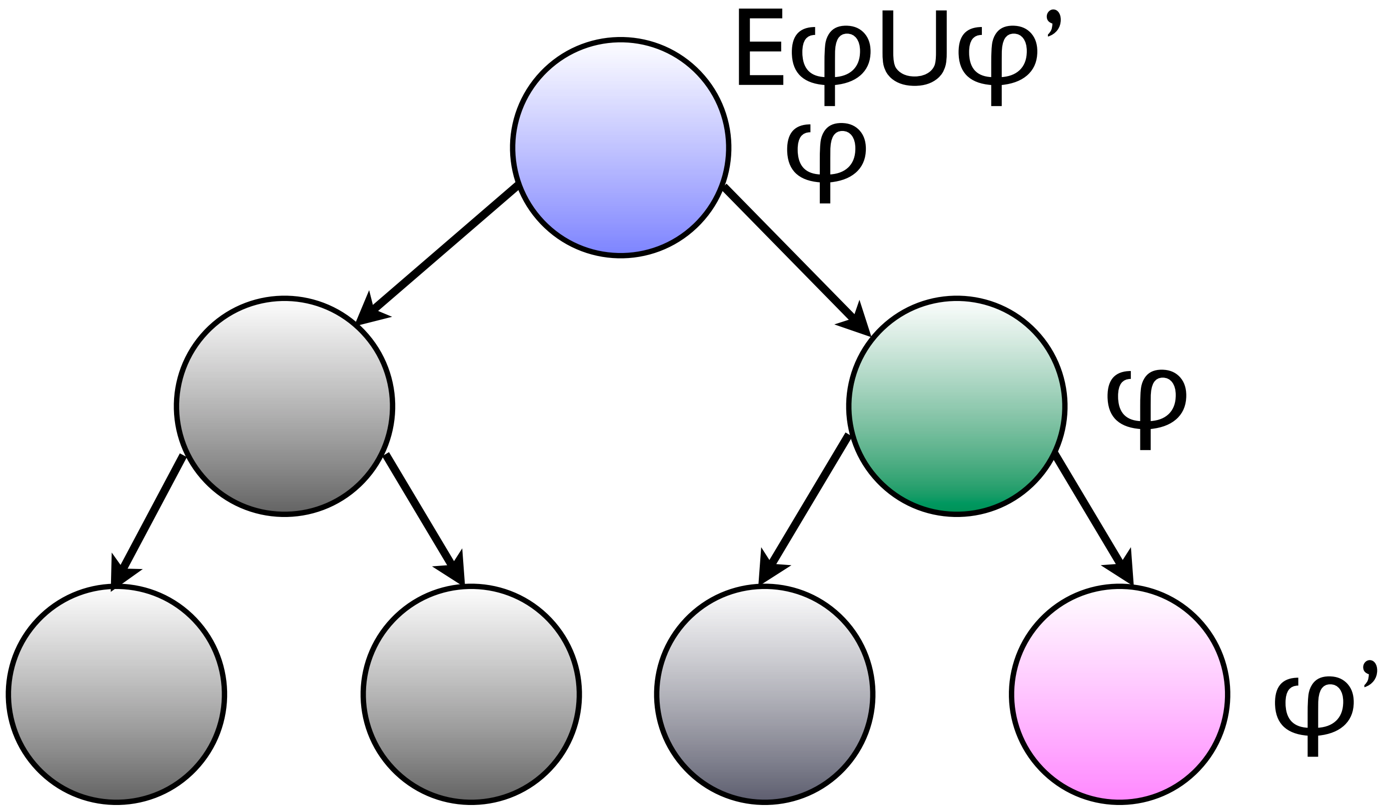
s ⊨ EXφ ssi il existe s', successeur de s t.q. s' ⊨φ

CTL: syntaxe et sémantique



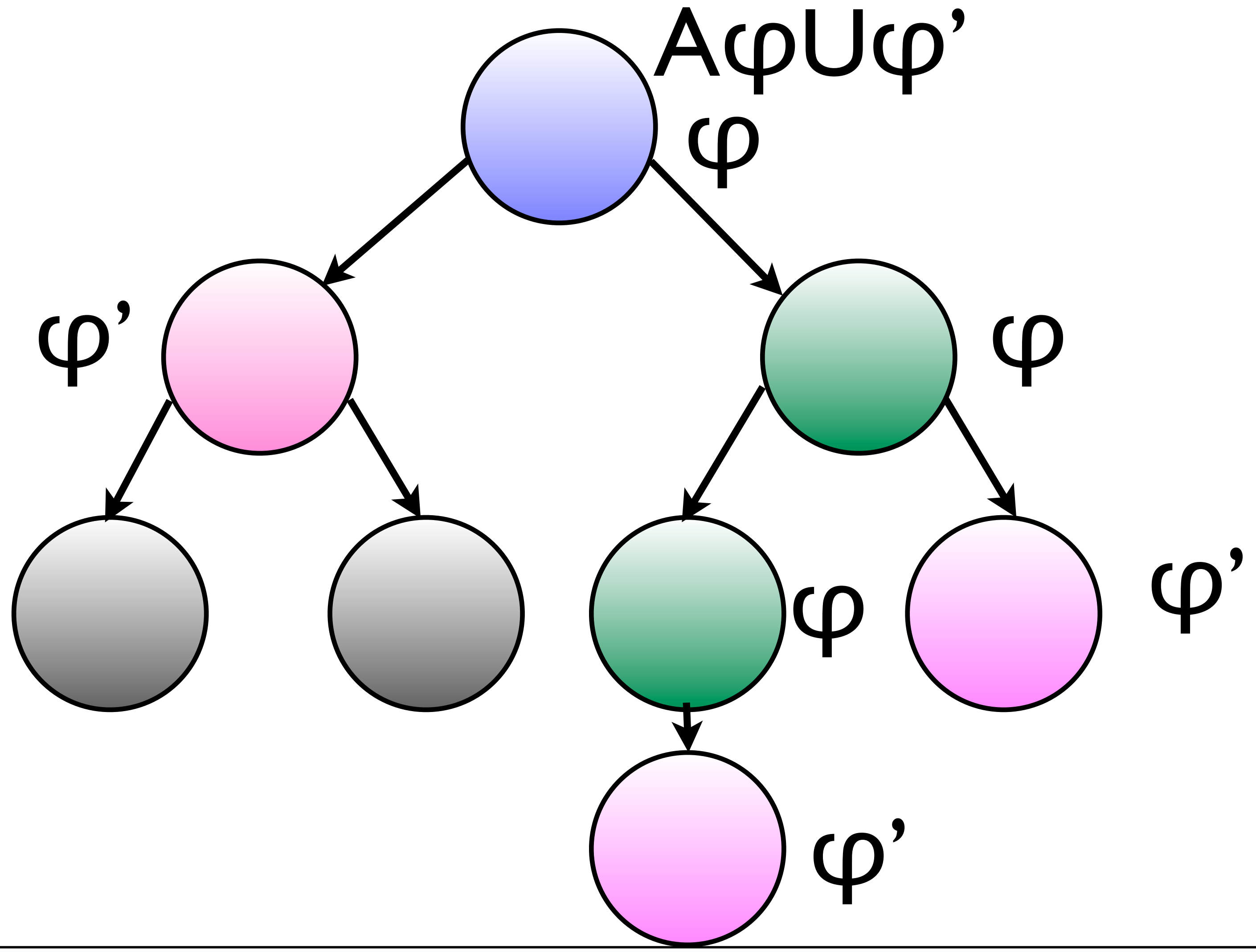
s ⊧ AXφ ssi pour tout s', successeur de s, s' ⊧φ

CTL: syntaxe et sémantique



$s \models E\varphi U\varphi'$ ssi il existe une exécution $s_0 s_1 \ldots s_k$ telle que $s_0=s$, $s_k \models \varphi'$ et pour tout $0 \leq i < k$, $s_i \models \varphi$.

CTL: syntaxe et sémantique



$s \models A\varphi U\varphi'$ ssi pour toute exécution $s_0 s_1 \ldots$ telle que $s_0 = s$, $\exists$ k t.q. $s_k \models \varphi'$ et pour tout $0 \leq i < k$, $s_i \models \varphi$.

CTL: syntaxe et sémantique

$\varphi ::= p \in AP \mid \neg\varphi \mid \varphi \vee \varphi$
$\mid EX\varphi \mid AX\varphi \mid E\varphi U\varphi \mid A\varphi U\varphi$

$s \models p$ ssi $p \in l(s)$

$s \models \neg\varphi$ ssi $s \not\models \varphi$

$s \models \varphi_1 \vee \varphi_2$ ssi $s \models \varphi_1$ ou $s \models \varphi_2$

$s \models EX\varphi$ ssi il existe s', successeur de s, t.q. $s' \models \varphi$

$s \models AX\varphi$ ssi s', pour tout s', successeur de s, $s' \models \varphi$

$s \models E\varphi_1 U\varphi_2$ ssi il existe une exécution $s_0 s_1 \ldots s_k$ tel que $s_0 = s$, $s_k \models \varphi_2$ et pour tout $0 \leq i \leq k$, $s_i \models \varphi_1$.

$s \models A\varphi_1 U\varphi_2$ ssi pour toute exécution $s_0 s_1 \ldots$ telle que $s_0 = s$, il existe k t.q. $s_k \models \varphi_2$ et pour tout $0 \leq i \leq k$, $s_i \models \varphi_1$.

CTL : macros

- $EF\varphi \equiv E\top U\varphi$
- $AF\varphi \equiv A\top U\varphi$
- $EG\varphi \equiv \neg AF\neg\varphi$
- $AG\varphi \equiv \neg EF\neg\varphi$

CTL : Equivalences de formules

- $AX\varphi = \neg EX\neg\varphi$
- $A\varphi U\varphi' = \neg E\neg(\varphi U\varphi') = \neg E(G\neg\varphi' \vee \neg\varphi' U(\neg\varphi \wedge\neg\varphi')) = \neg EG\neg\varphi' \wedge \neg E(\neg\varphi' U(\neg\varphi \wedge \neg\varphi'))$

CTL : Lois d'expansion

- $A\varphi U\varphi' = \varphi' \vee (\varphi \wedge AX(A\varphi U\varphi'))$
- $AF\varphi = \varphi \vee (AXAF\varphi)$
- $AG\varphi = \varphi \wedge AXAG\varphi$
- $E\varphi U\varphi' = \varphi' \vee (\varphi \wedge EXE(\varphi U\varphi'))$
- $EF\varphi = \varphi \vee EXEF\varphi$
- $EG\varphi = \varphi \wedge EXEG\varphi$

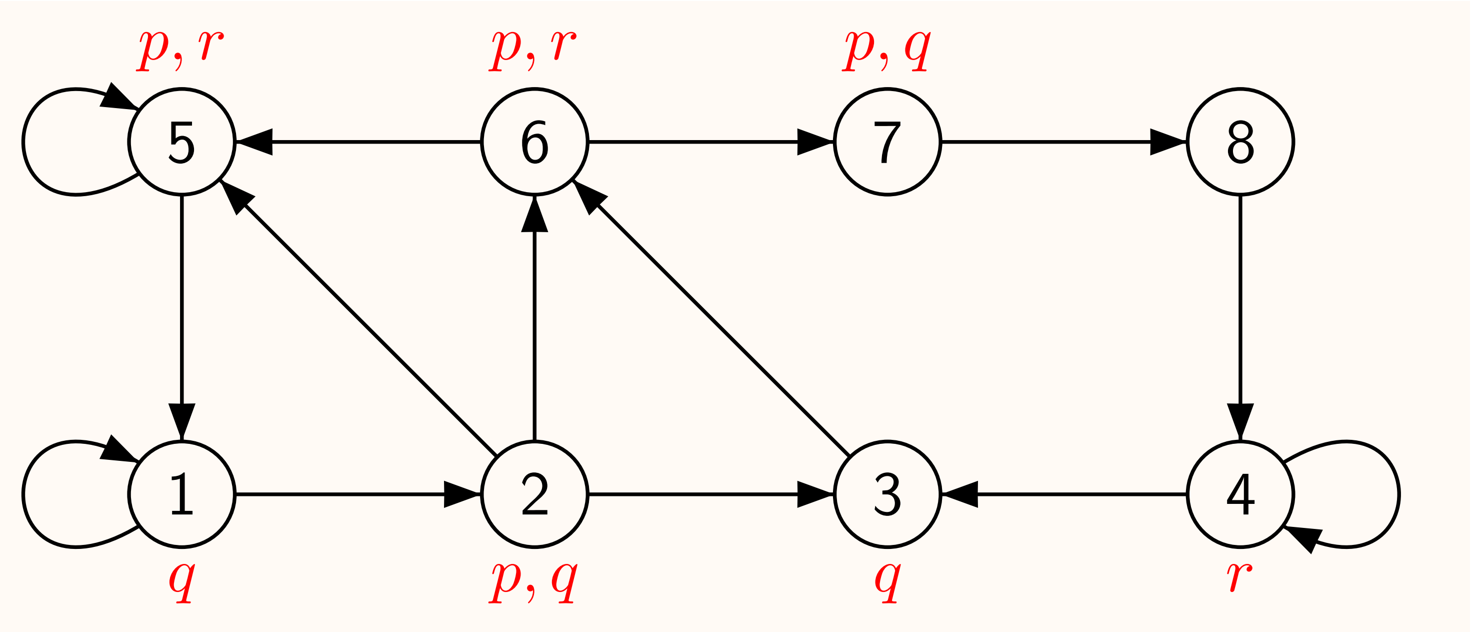
CTL : lois distributives

- $AG(\varphi \wedge \varphi') = AG\varphi \wedge AG\varphi'$
- $EF(\varphi \vee \varphi') = EF\varphi \vee EF\varphi'$

Exemples

- Accessibilité : $EF(x=0)$
- Invariance : $AG\neg(x=0)$
- Vivacité : $AGAF(active)$

Exercice



S(EXp)? S(AXp)?
S(EFp)? S(AFp)?
S(EqUr)?S(AqUr)?

Exercice

- Toute fraude est susceptible d'être détectée un jour(AP={fraude, detect})
- Deux processus ne sont jamais en section critique en même temps (AP={crit1,crit2})
- Toute requête sera un jour satisfaite (AP = {requete, reponse})
- Le processus est activé infiniment souvent (AP= {active})
- Il est possible qu'à partir d'un moment, l'alarme sonne continuellement (AP= {alarm})
- La lumière finit toujours par s'éteindre (AP= {off})
- La lumière finit toujours par s'éteindre et la ventilation tourne tant que la lumière est allumée (AP= {ventilation,off})

Comparaison LTL/CTL

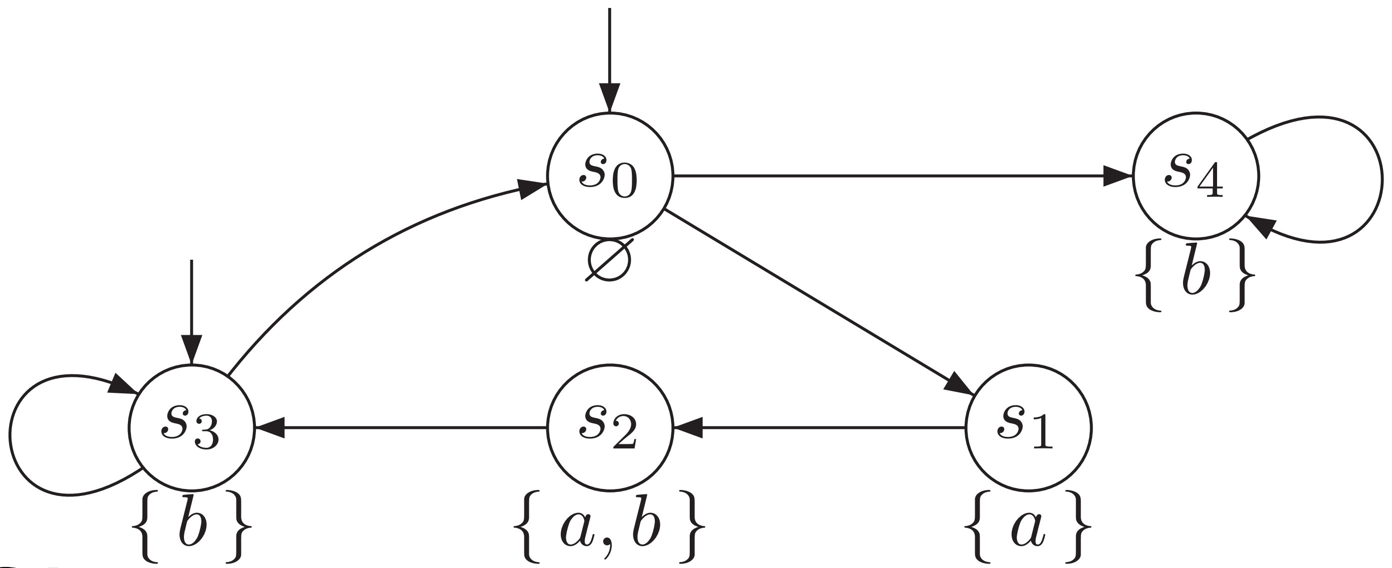
- La formule CTL $AF(a \wedge EXa)$ n'est pas exprimable en LTL
- La formule LTL $FG\ request \rightarrow GF\ response$ n'est pas exprimable en CTL
- LTL et CTL incomparables!
- LTL et CTL inclus dans CTL*

Model-Checking de CTL

- Données : Une structure de Kripke $M=(Q, T, A, Q_0, AP, l)$ et une formule CTL φ.
- Question : Est-ce que $M \models \varphi$?
  - $M \models \varphi$ ssi $Q_0 \subseteq S(\varphi)$.

Exercice

$M \models \varphi$?



$\varphi = A(aUb) \vee EXEGb$

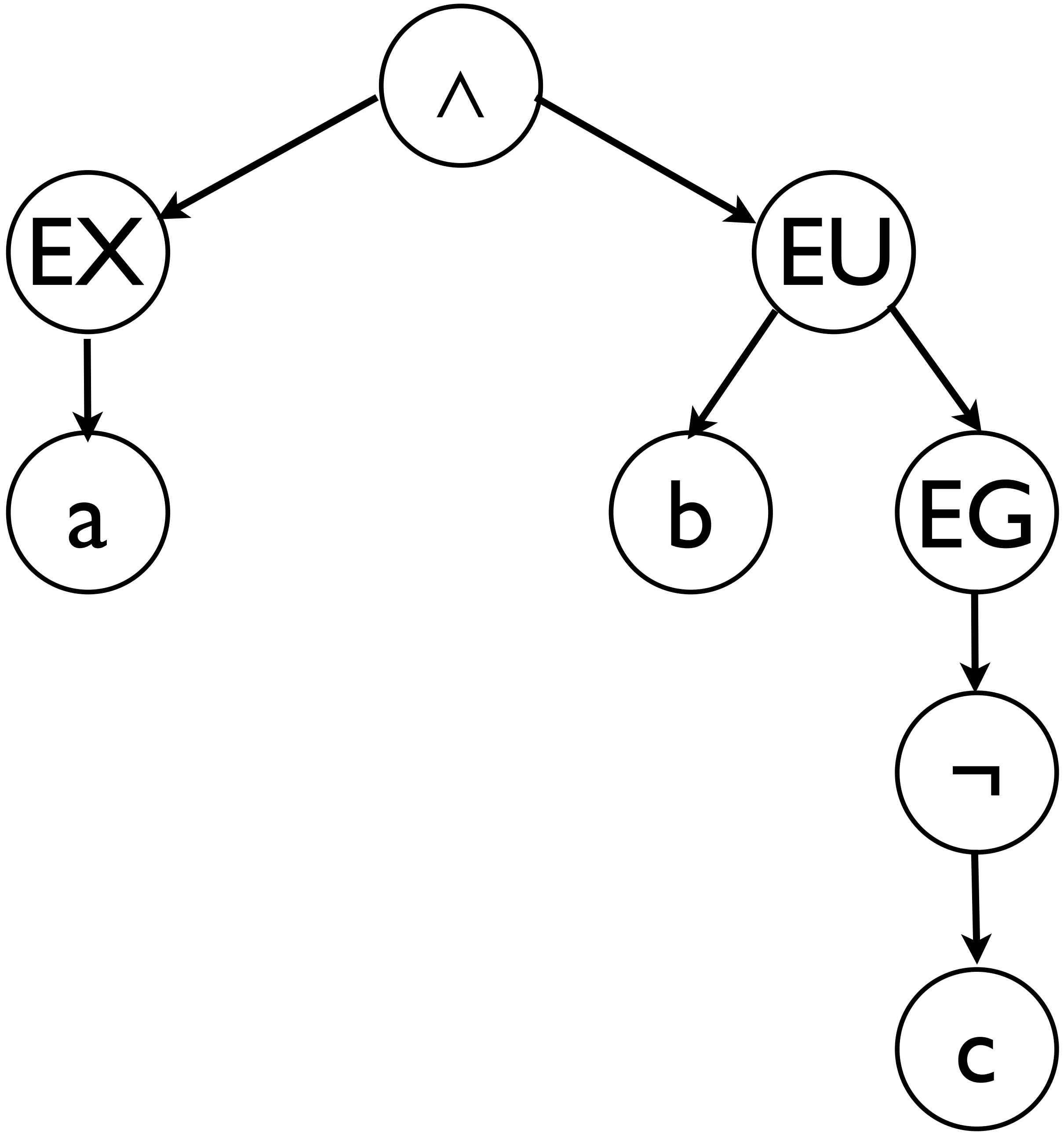
$\varphi = AG(A(aUb))$

Model-Checking de CTL : principe

- Procédure de marquage des états par les sous-formules de φ.
- Induction sur les sous-formules
- Arbre syntaxique de φ

Arbre syntaxique : exemple

$\varphi = EXa \wedge E(bU(EG\neg c))$



Procédure de marquage: mark(φ)

- si φ=p :

  pour tout s$\in$Q s.φ:=(p$\in$ l(s));

- si φ=$\neg\varphi$':

  mark(φ');

  pour tout s$\in$Q s.φ:=$\neg$s.φ';

- si $\varphi=\varphi_1\vee\varphi_2$

  mark(φ_1); mark(φ_2);

  pour tout s$\in$Q, s.φ:=s.$\varphi_1\vee$s.φ_2;

- si φ=EXφ':

  mark(φ');

  pour tout s$\in$Q, s.φ:=false;

  pour tout (t,s)$\in$T, si s.φ', alors t.φ:=true;

- si φ=AXφ':

  mark(φ');

  pour tout s$\in$Q, s.φ:=true;

  pour tout (t,s)$\in$T, si $\neg$s.φ', alors t.φ:=false;

Procédure de marquage: mark(φ)

- cas $E\varphi_1 U\varphi_2$: idée = $E\varphi_1 U\varphi_2 \equiv \varphi_2 \vee (\varphi_1 \wedge EX(E\varphi_1 U\varphi_2))$

  $X = Sat(\varphi_2) \cup (Sat(\varphi_1) \cap Pre(X))$

- si $\varphi = E\varphi_1 U\varphi_2$:

  mark(φ_1);mark(φ_2);

  L:=$\varnothing$;

  pour tout s$\in$Q, s.φ:=s.φ_2; si s.φ, L:=L U{s};

  tant que L$\neq\varnothing$

  pour tout s$\in$L, L:=L$\setminus${s};

  pour tout prédécesseur t de s, si t. $\varphi_1 \wedge \neg$t. φ, alors t. φ:=true ; L:=LU{t};

Procédure de marquage: mark(φ)

- cas $A\varphi_1 U\varphi_2$: idée = $A\varphi_1 U\varphi_2 \equiv \varphi_2 \vee (\varphi_1 \wedge AX(A\varphi_1 U\varphi_2))$

  $X = Sat(\varphi_2) \cup (Sat(\varphi_1) \cap \{t \in S \mid \forall s, (t,s) \in T, s \in X\})$

- si $\varphi = A\varphi_1 U\varphi_2$:

  mark(φ_1);mark(φ_2);

  L:=$\emptyset$;

  pour tout s$\in$Q, s.φ:=s.φ2; s.nb=degree(s);si s.φ, L:=L U{s};

  tant que L$\neq\emptyset$

  pour tout s$\in$L, L:=L$\setminus${s};

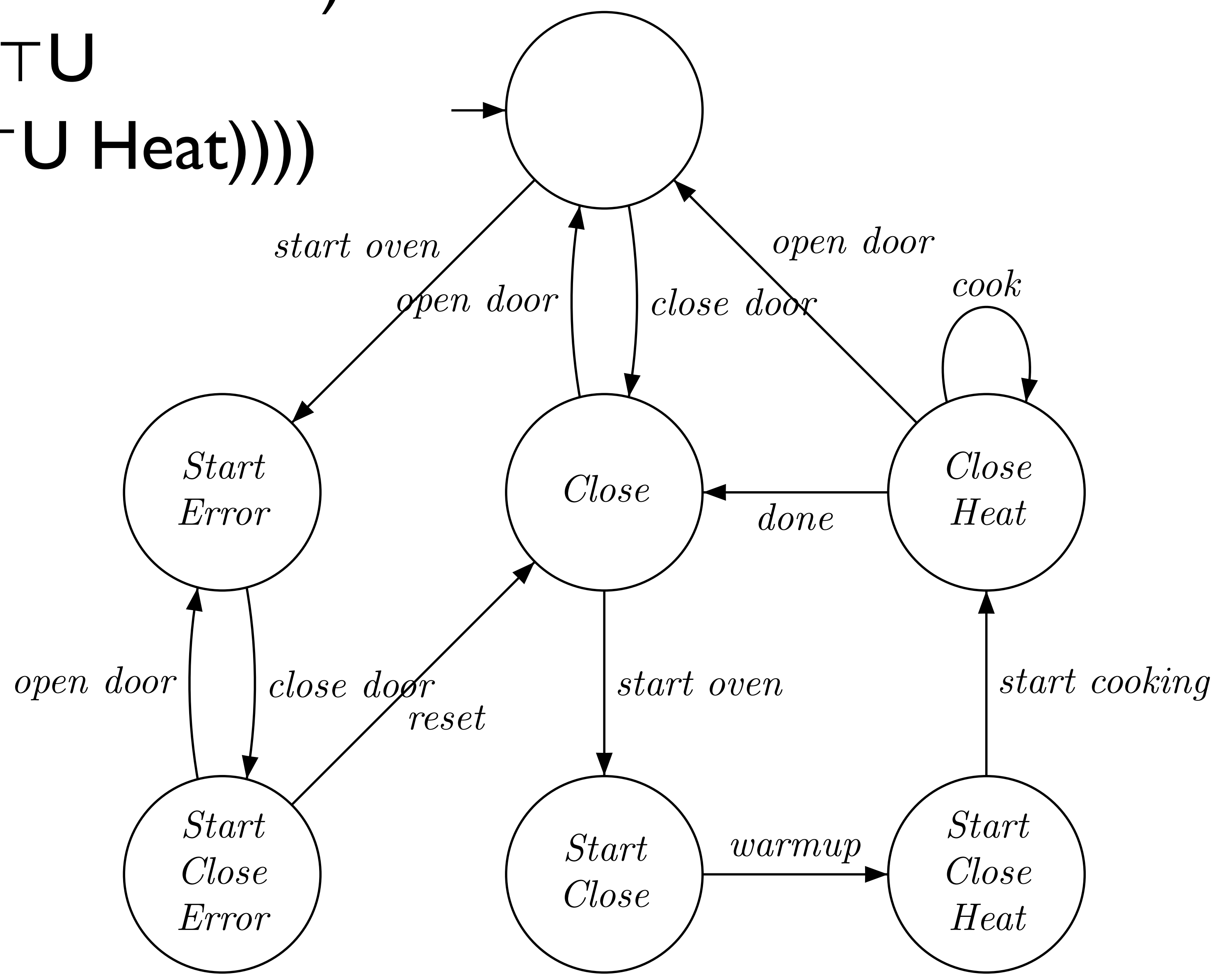
  pour tout prédécesseur t de s,

  t.nb:=t.nb-1;

  si t.nb:=0$\wedge$t. φ1$\wedge\neg$t. φ, alors t. φ:=true ; L:=LU{t};

Exemple : un four à microondes

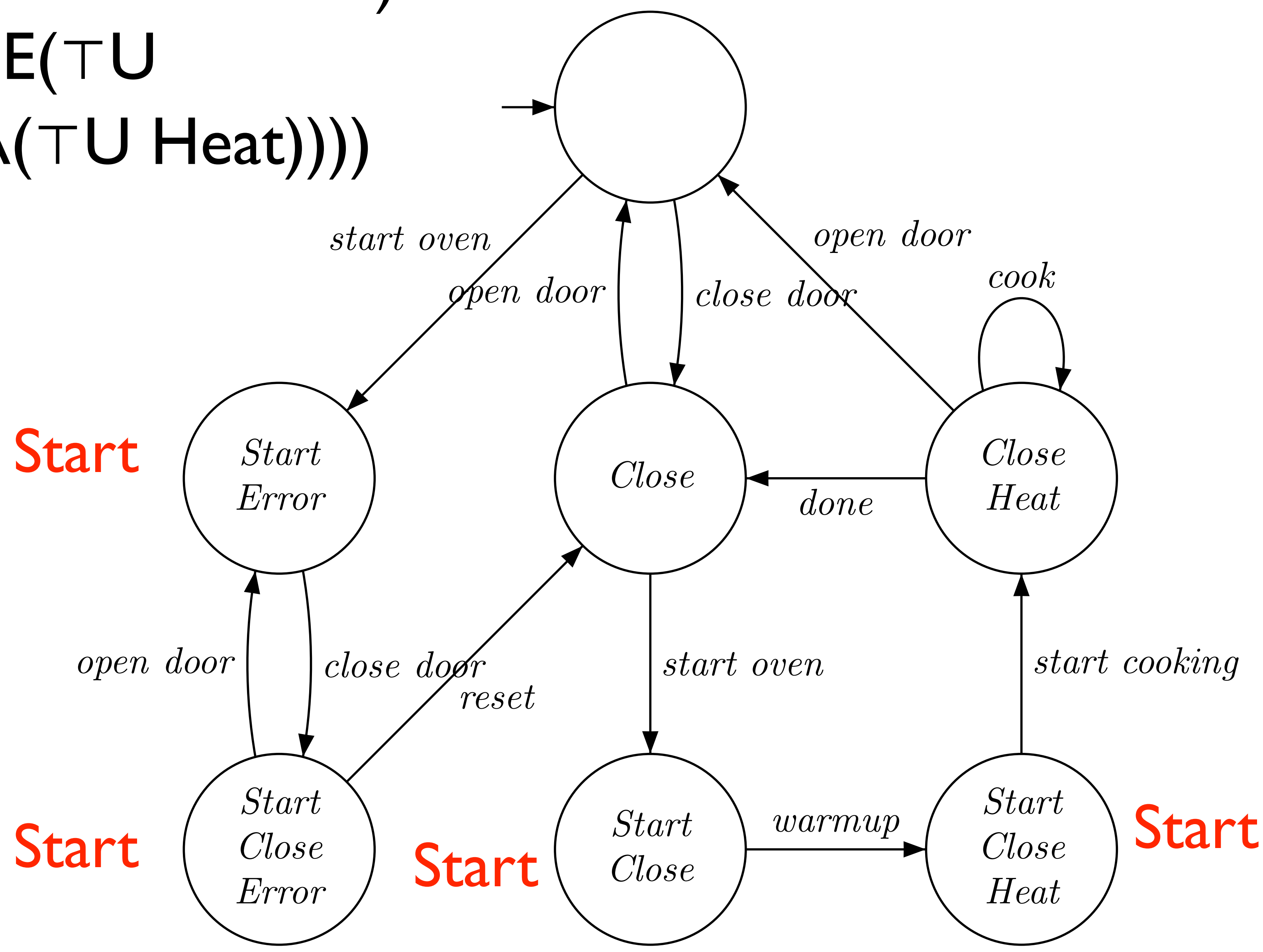
φ =AG(Start→AF Heat)
=¬E(⊤U
(Start∧¬(A(⊤U Heat))))



[Model-Checking, Clarke, Grumberg, Peled]

Exemple : un four à microondes

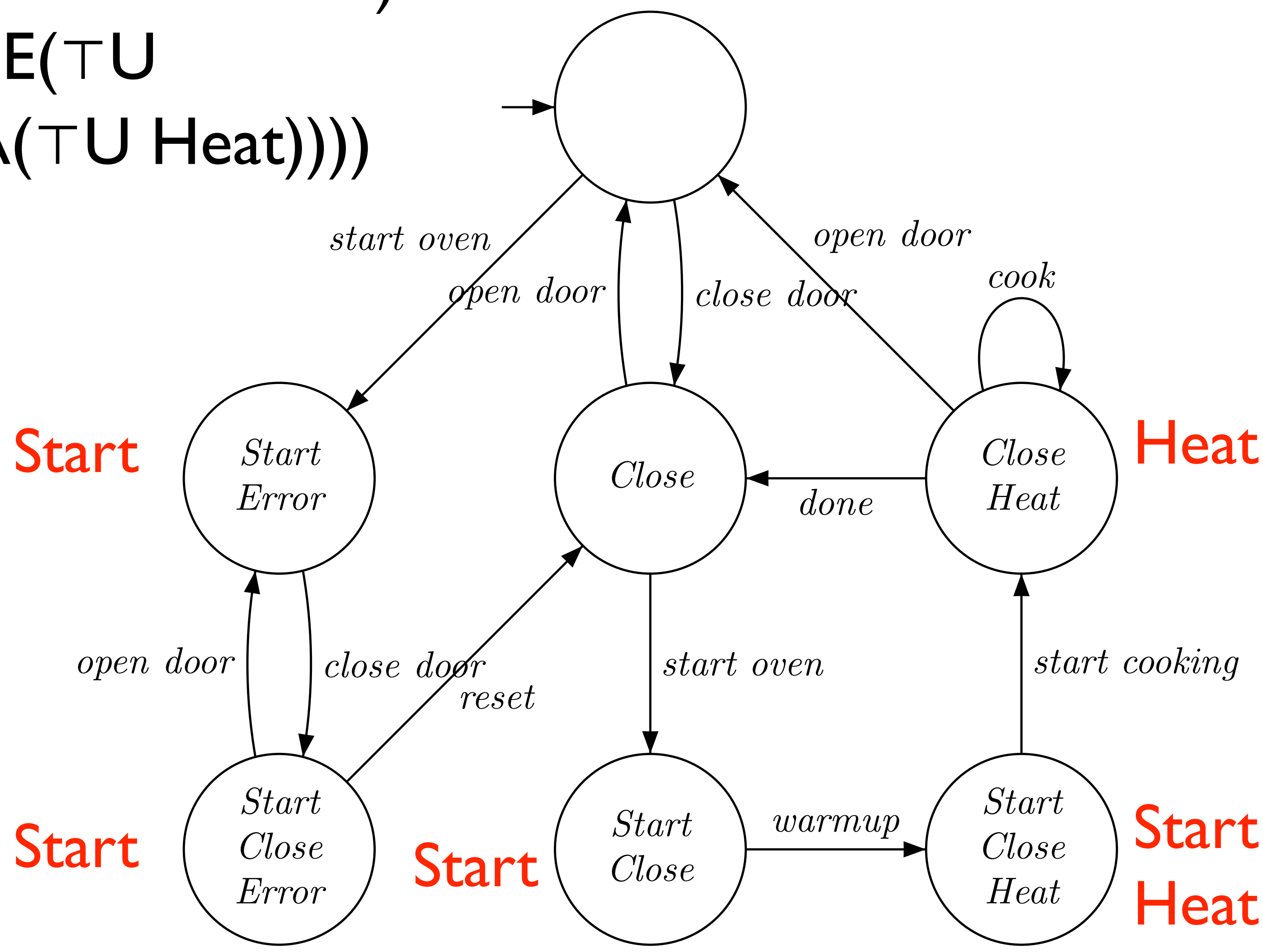
φ =AG(Start→AF Heat)
=¬E(⊤U
(Start∧¬(A(⊤U Heat))))



[Model-Checking, Clarke, Grumberg, Peled]

Exemple : un four à microondes

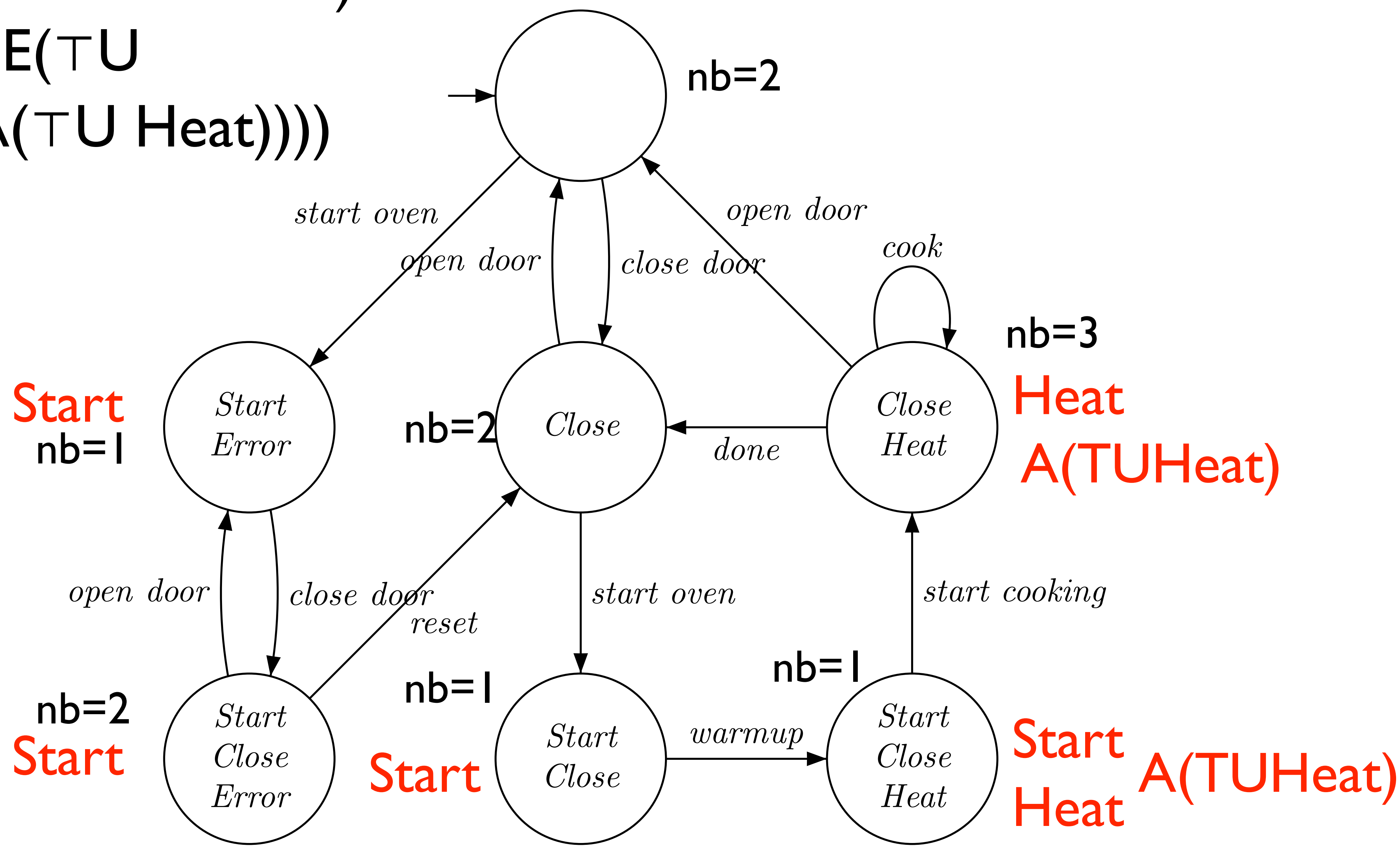
φ $= AG(Start \rightarrow AF\ Heat)$
$= \neg E(\top U$
$(Start \wedge \neg (A(\top U\ Heat))))$



[Model-Checking, Clarke, Grumberg, Peled]

Exemple : un four à microondes

φ =AG(Start→AF Heat)
=¬E(⊤U
(Start∧¬(A(⊤U Heat))))



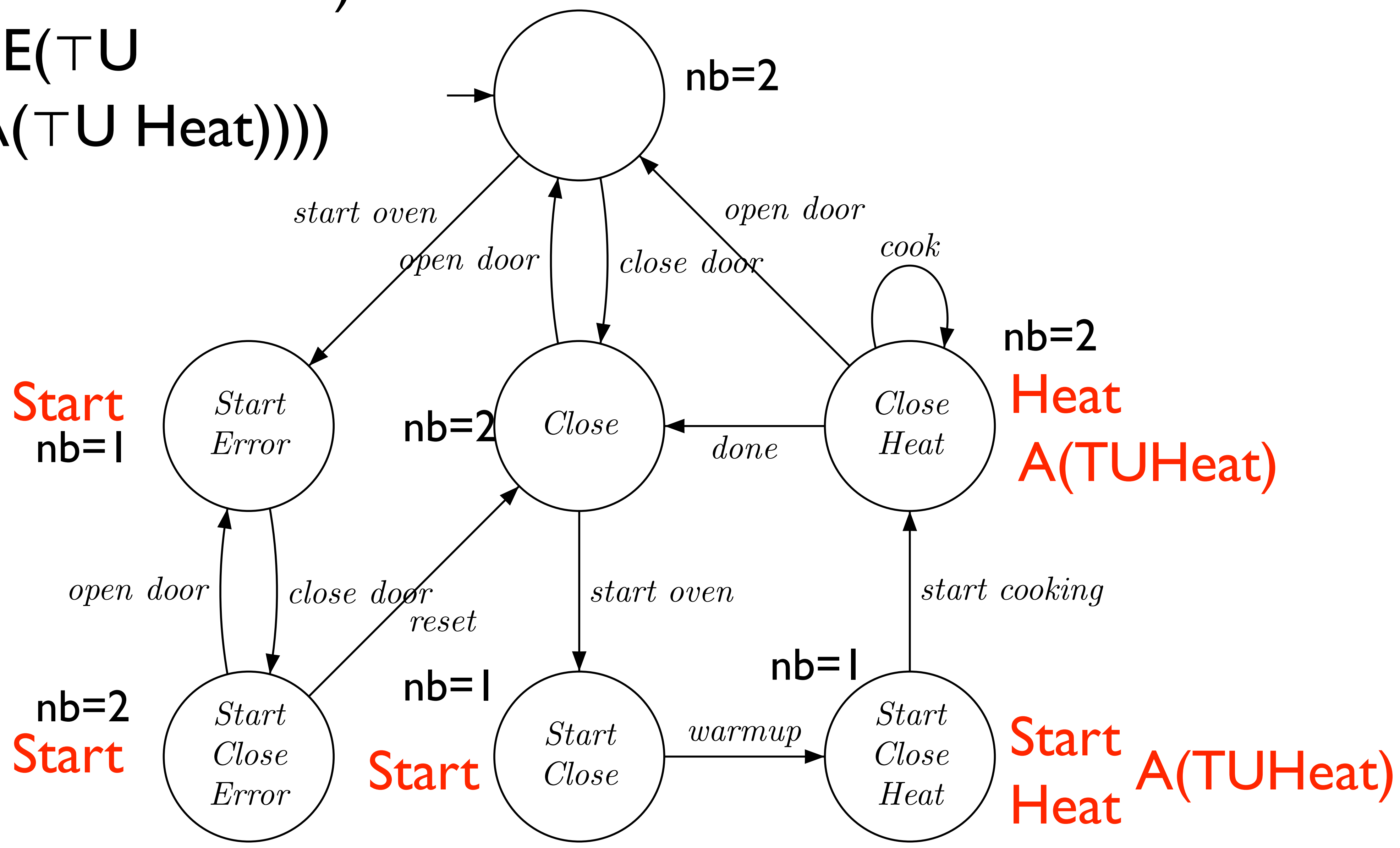
[Model-Checking, Clarke, Grumberg, Peled]

Exemple : un four à microondes

φ $= AG(Start \rightarrow AF\ Heat)$

$= \neg E(\top U$

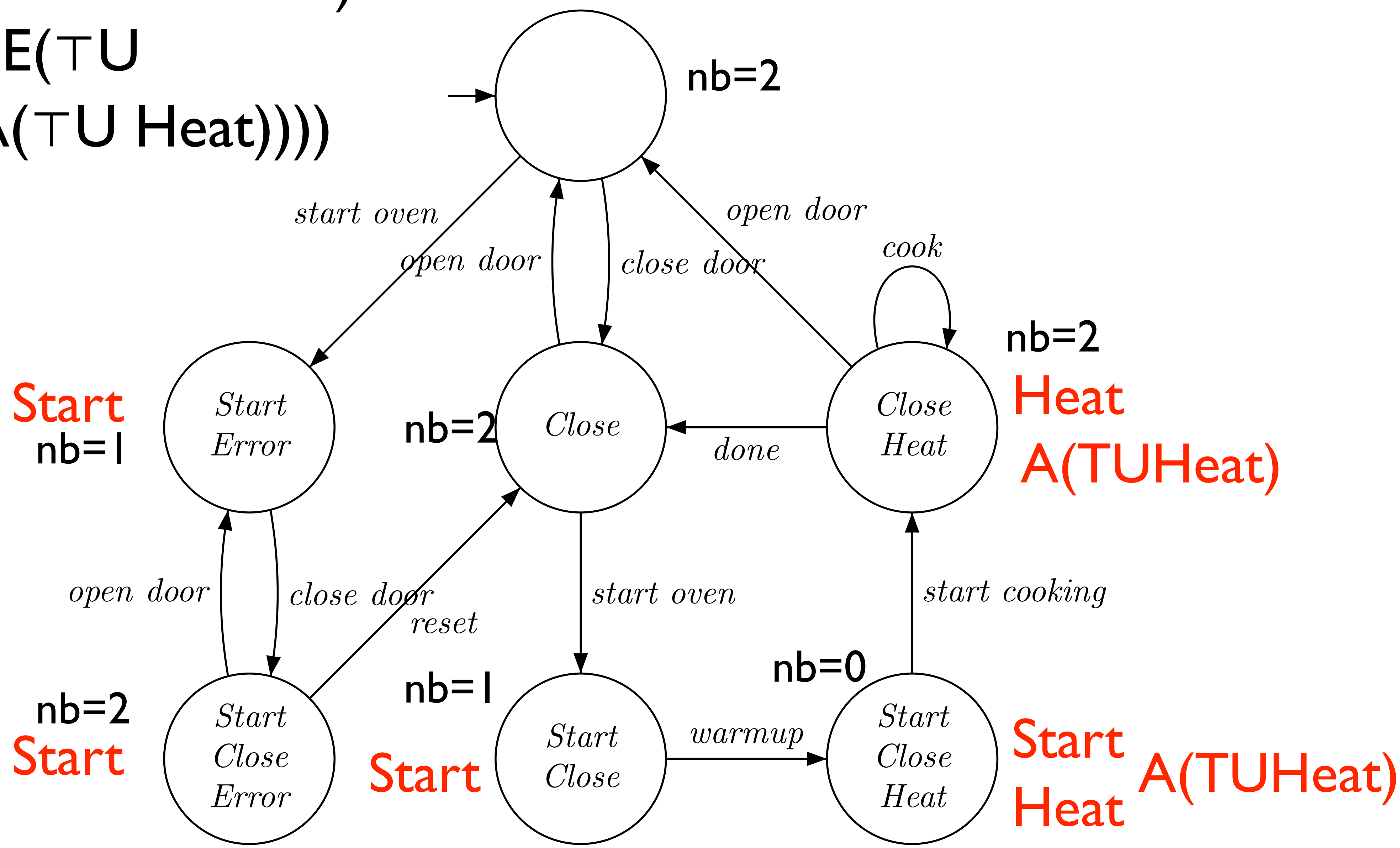
$(Start \wedge \neg(A(\top U\ Heat))))$



[Model-Checking, Clarke, Grumberg, Peled]

Exemple : un four à microondes

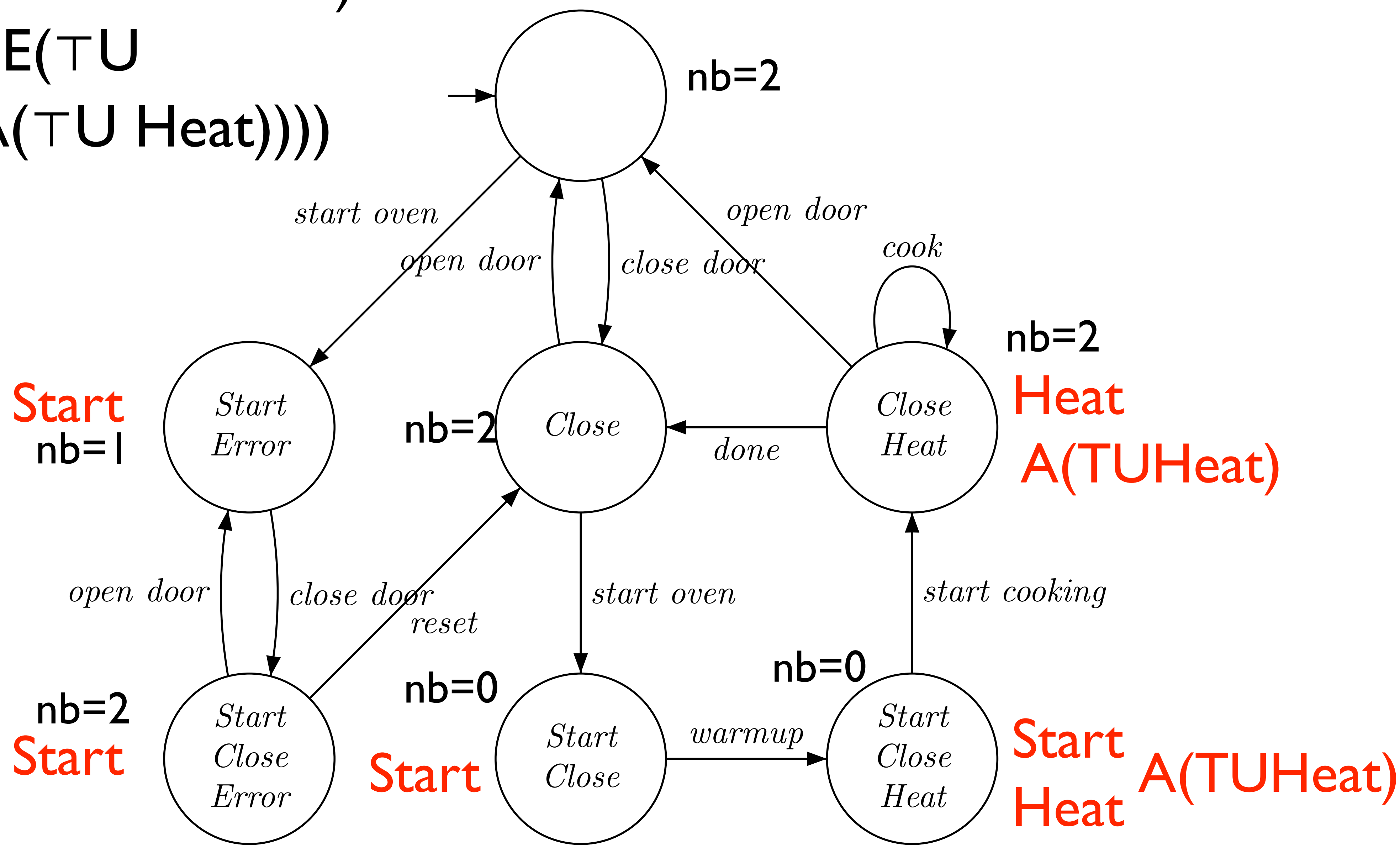
φ =AG(Start→AF Heat)
=¬E(⊤U
(Start∧¬(A(⊤U Heat))))



[Model-Checking, Clarke, Grumberg, Peled]

Exemple : un four à microondes

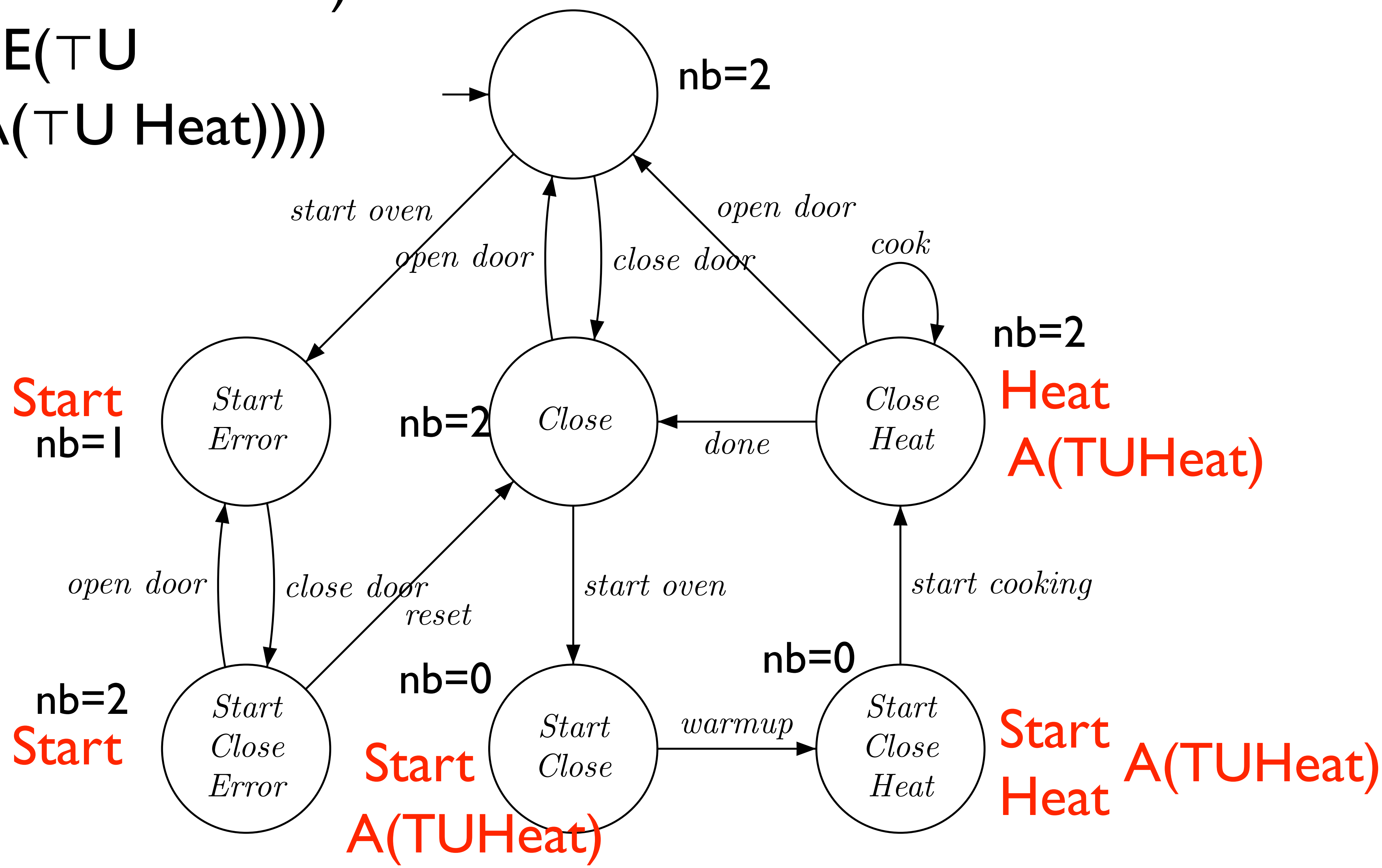
$$\varphi = AG(Start \rightarrow AF\ Heat)$$
$$= \neg E(\top U\ (Start \wedge \neg(A(\top U\ Heat))))$$



[Model-Checking, Clarke, Grumberg, Peled]

Exemple : un four à microondes

φ =AG(Start→AF Heat)
=¬E(⊤U
(Start∧¬(A(⊤U Heat))))



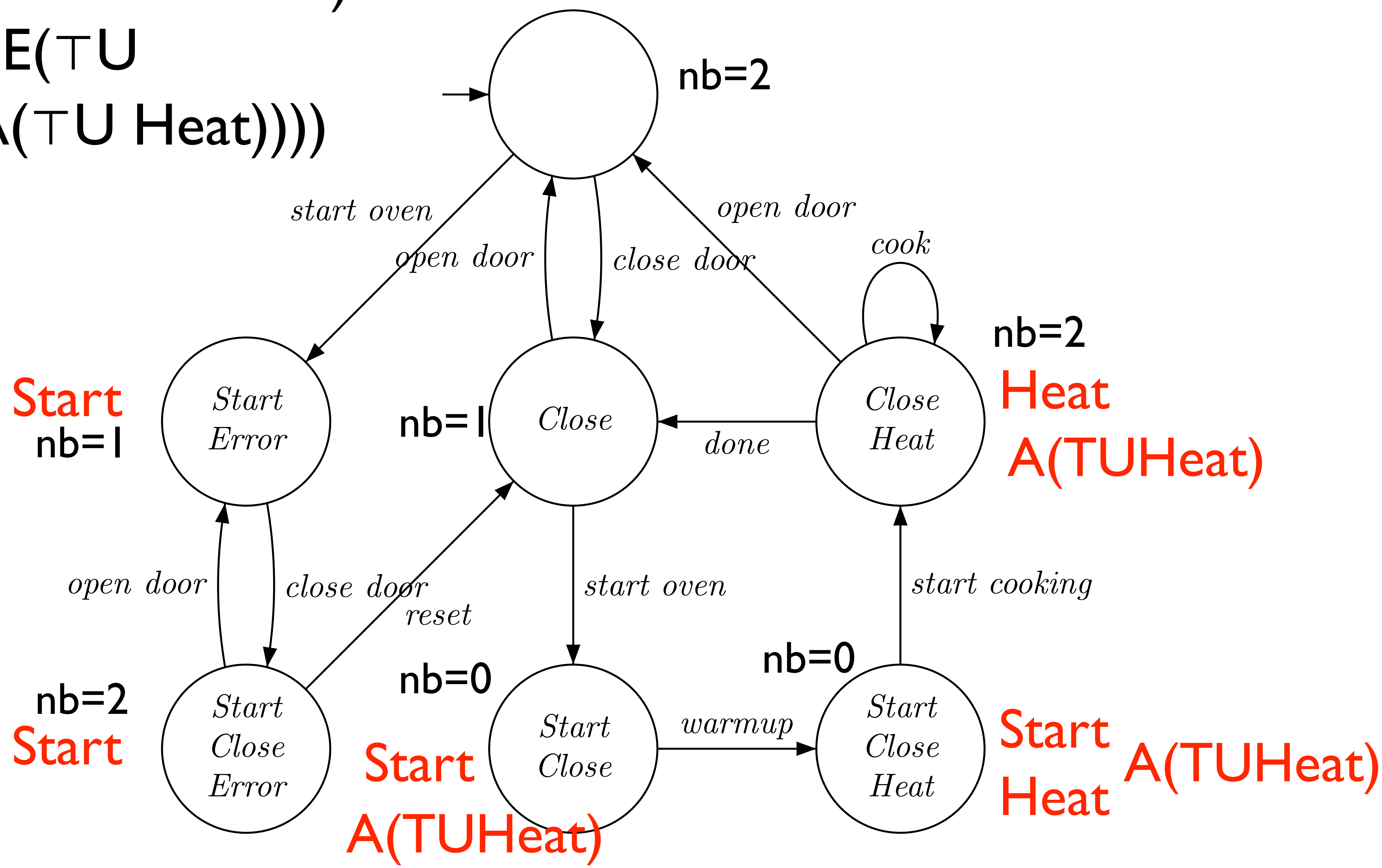
[Model-Checking, Clarke, Grumberg, Peled]

Exemple : un four à microondes

φ =AG(Start→AF Heat)

=¬E(⊤U

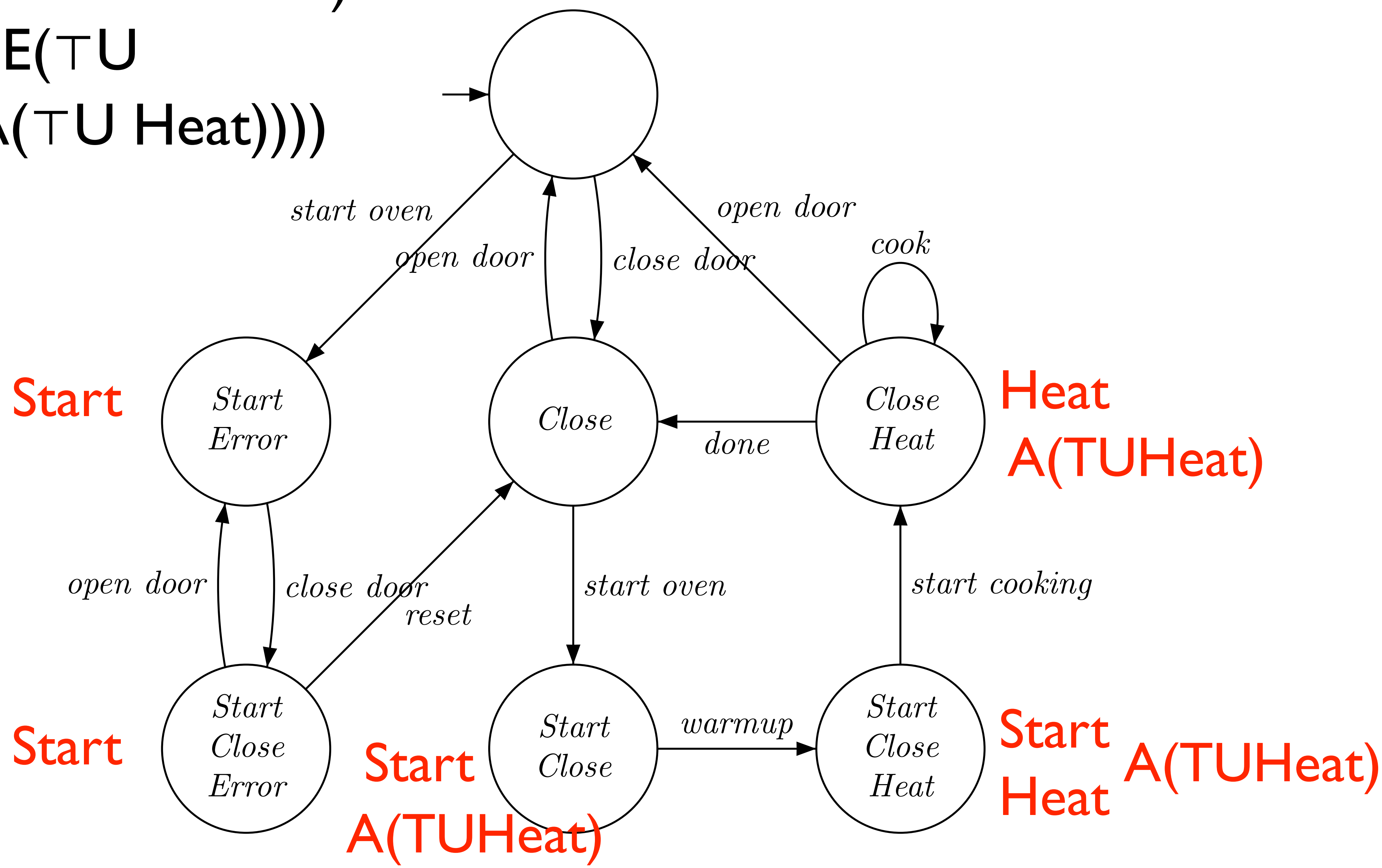
(Start∧¬(A(⊤U Heat))))



[Model-Checking, Clarke, Grumberg, Peled]

Exemple : un four à microondes

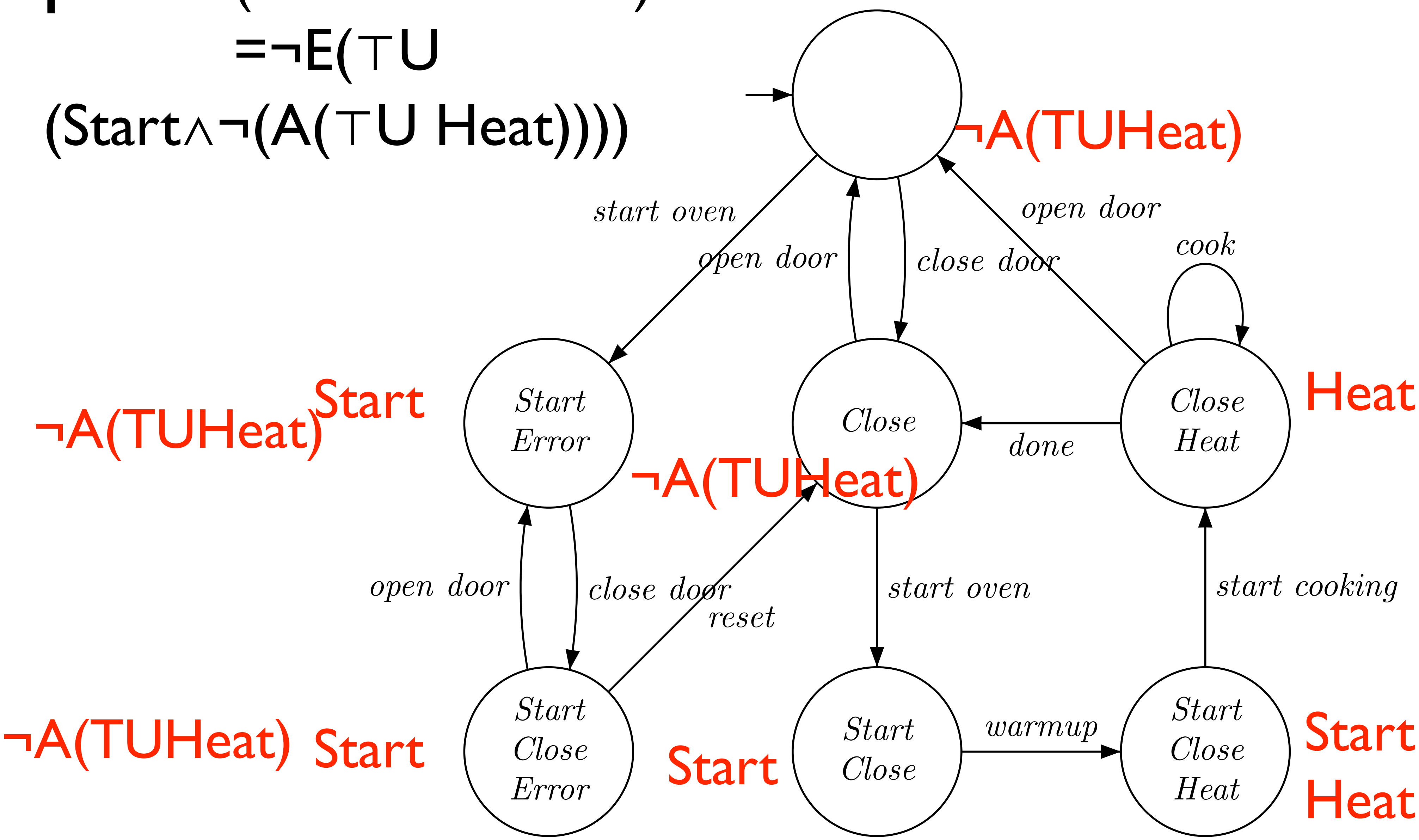
$\varphi = AG(Start \rightarrow AF\ Heat)$
$= \neg E(\top U$
$(Start \wedge \neg(A(\top U\ Heat))))$



[Model-Checking, Clarke, Grumberg, Peled]

Exemple : un four à microondes

φ =AG(Start→AF Heat)
=¬E(⊤U
(Start∧¬(A(⊤U Heat))))



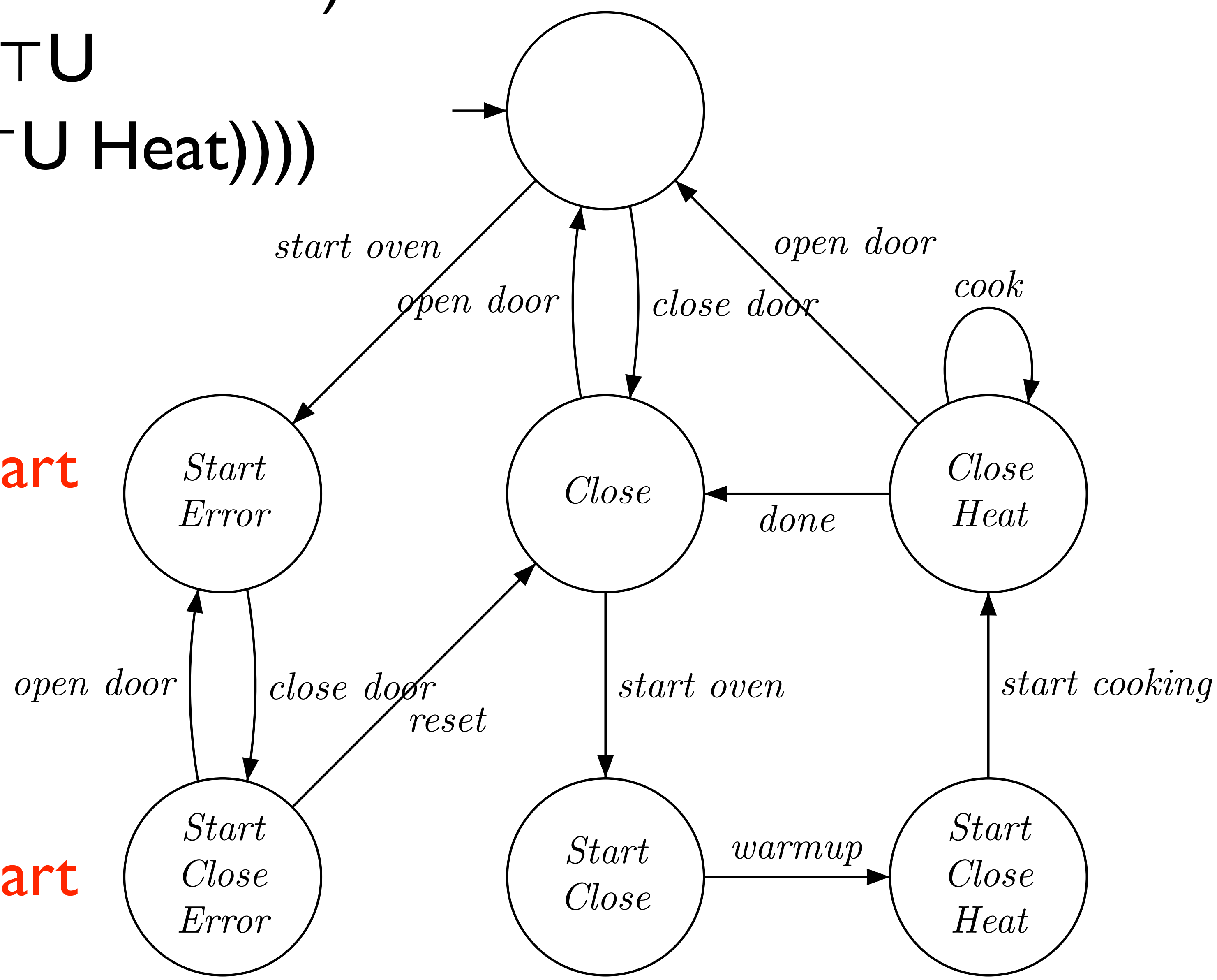
[Model-Checking, Clarke, Grumberg, Peled]

Exemple : un four à microondes

φ =AG(Start→AF Heat)
=¬E(⊤U
(Start∧¬(A(⊤U Heat))))

¬A(TUHeat) Start

¬A(TUHeat) Start



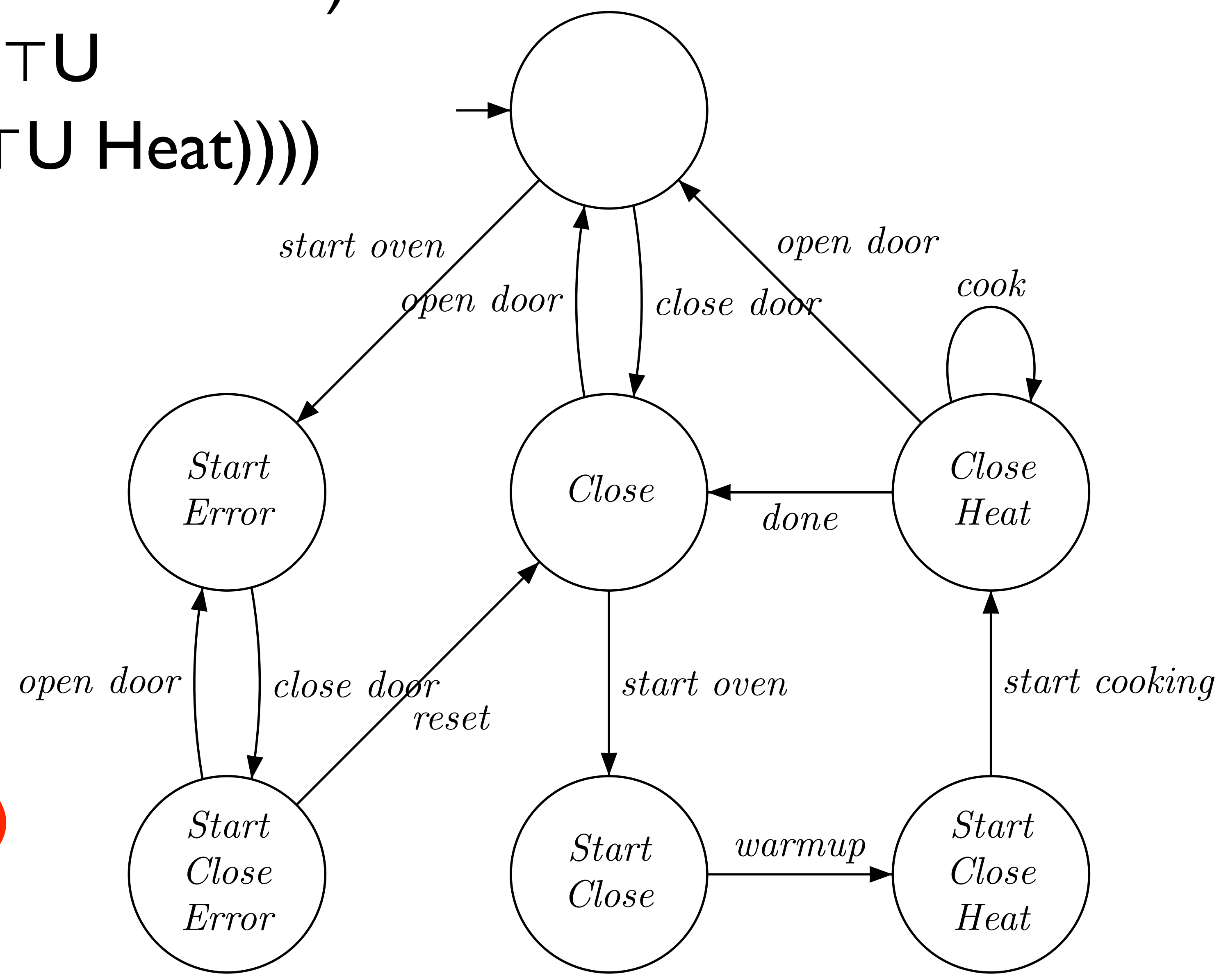
[Model-Checking, Clarke, Grumberg, Peled]

Exemple : un four à microondes

φ =AG(Start→AF Heat)
=¬E(⊤U
(Start∧¬(A(⊤U Heat))))

E(TUf)

E(TUf)



[Model-Checking, Clarke, Grumberg, Peled]

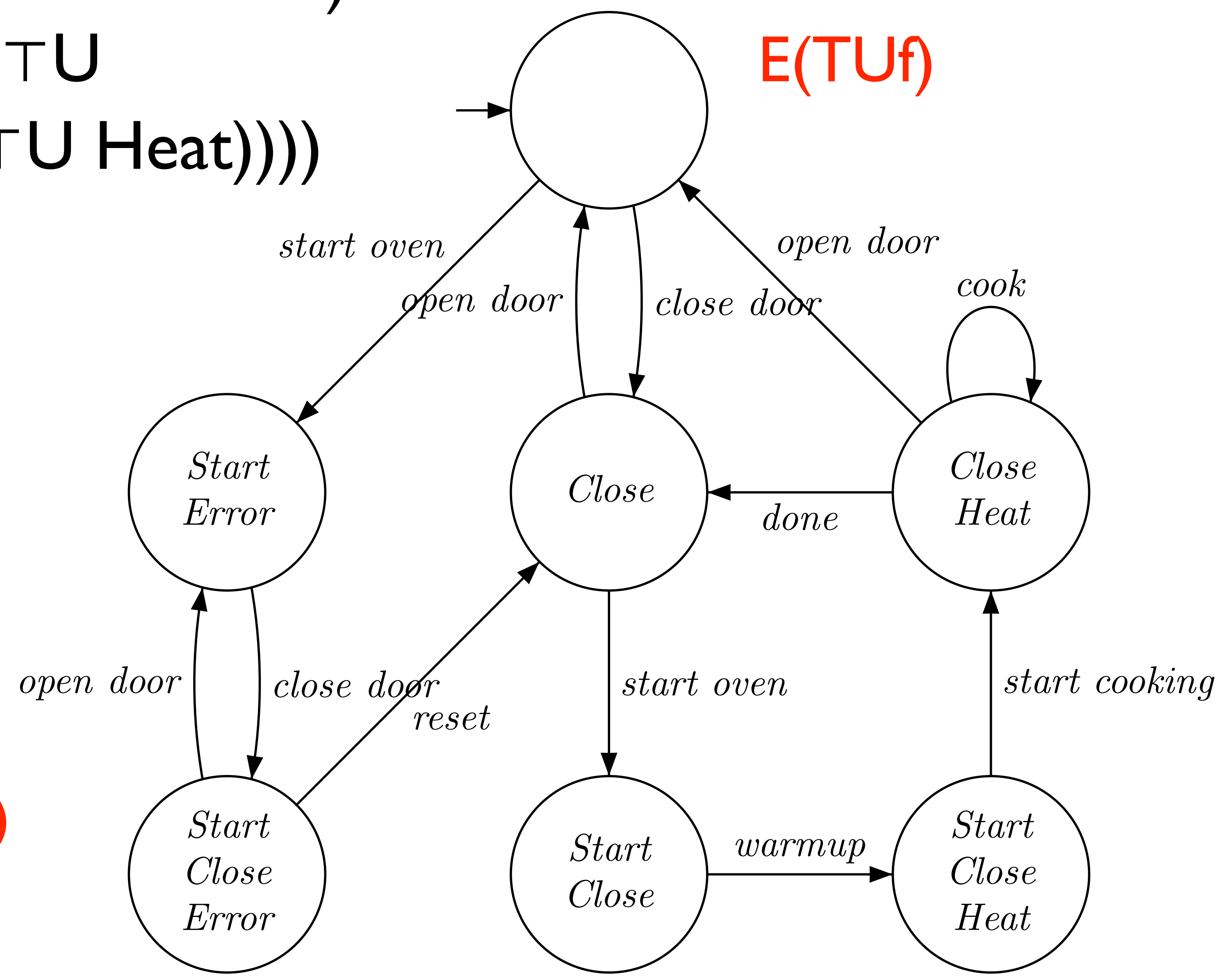
Exemple : un four à microondes

φ =AG(Start→AF Heat)
=¬E(⊤U
(Start∧¬(A(⊤U Heat))))

E(TUf)

E(TUf)

E(TUf)

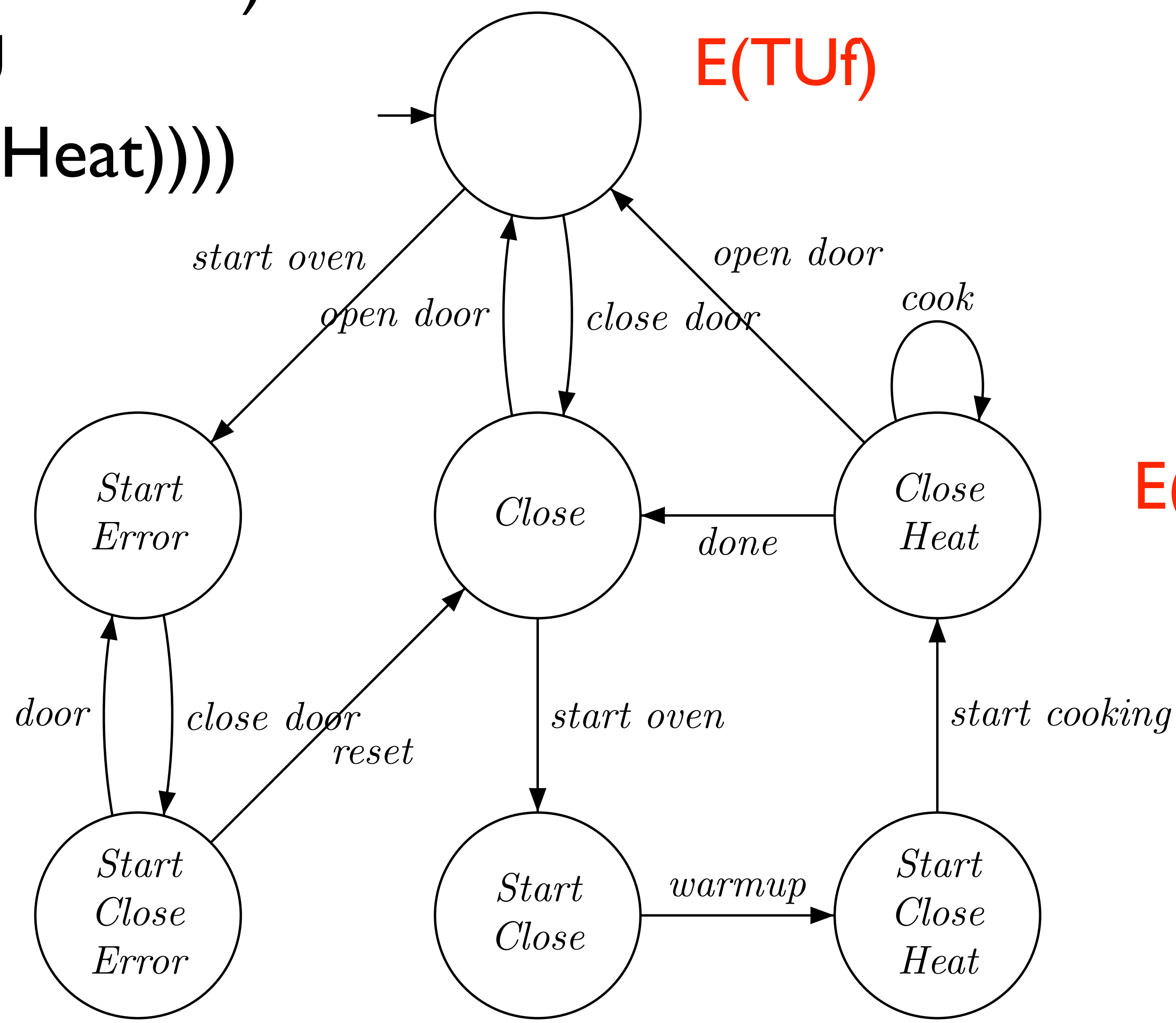


[Model-Checking, Clarke, Grumberg, Peled]

Exemple : un four à microondes

φ =AG(Start→AF Heat)
=¬E(⊤U
(Start∧¬(A(⊤U Heat))))

E(⊤Uf)



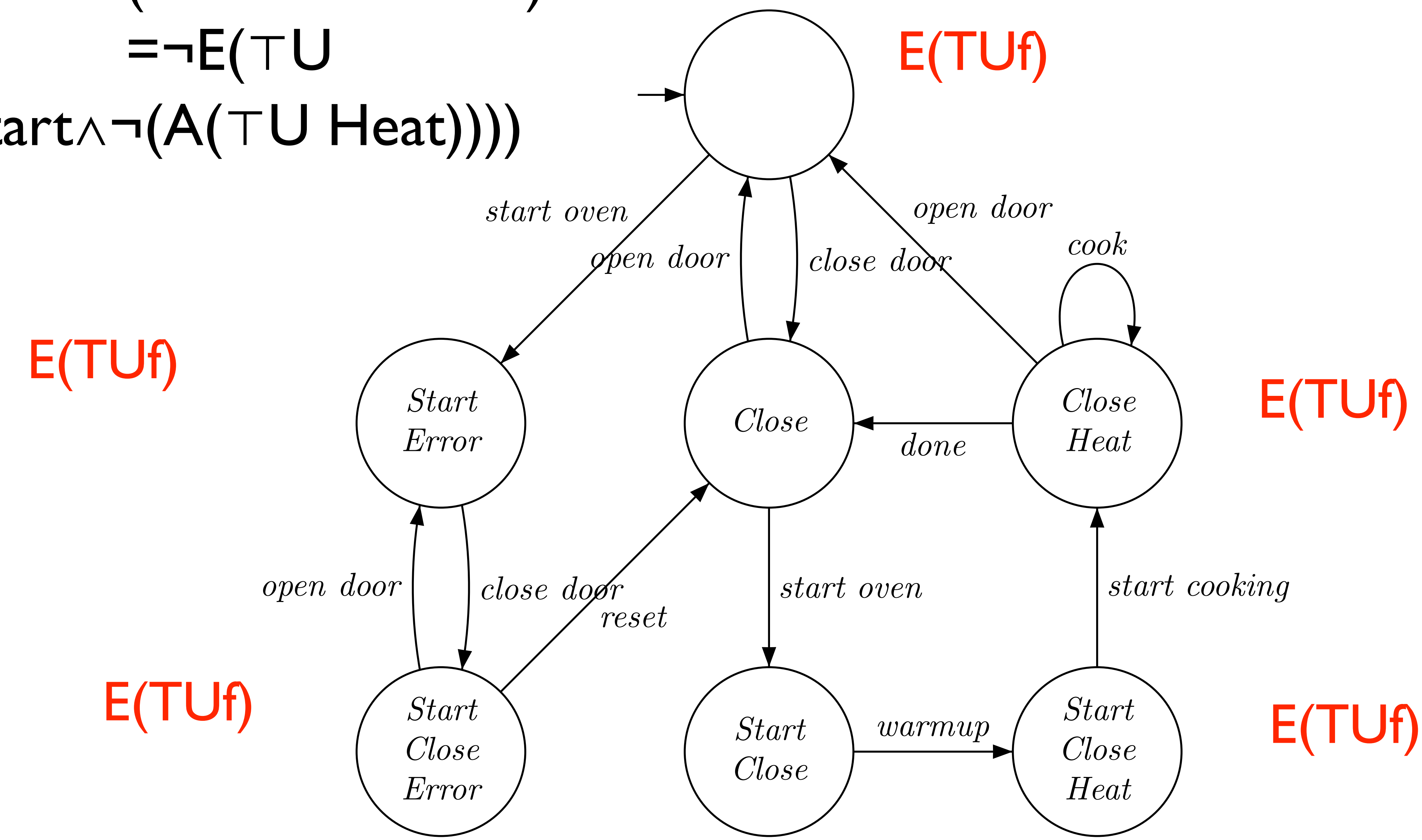
E(⊤Uf)

E(⊤Uf)

[Model-Checking, Clarke, Grumberg, Peled]

Exemple : un four à microondes

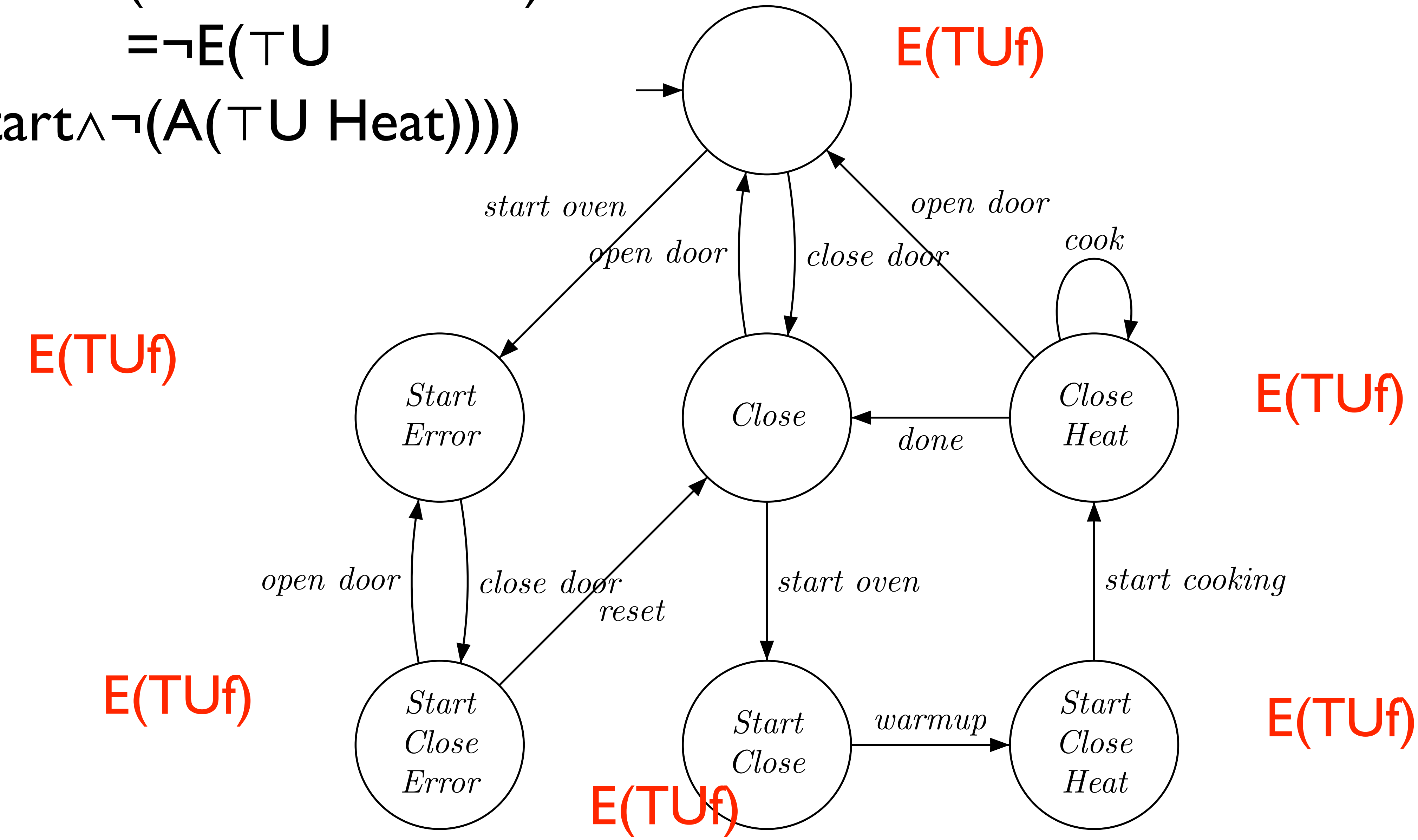
φ =AG(Start→AF Heat)
=¬E(⊤U
(Start∧¬(A(⊤U Heat))))



[Model-Checking, Clarke, Grumberg, Peled]

Exemple : un four à microondes

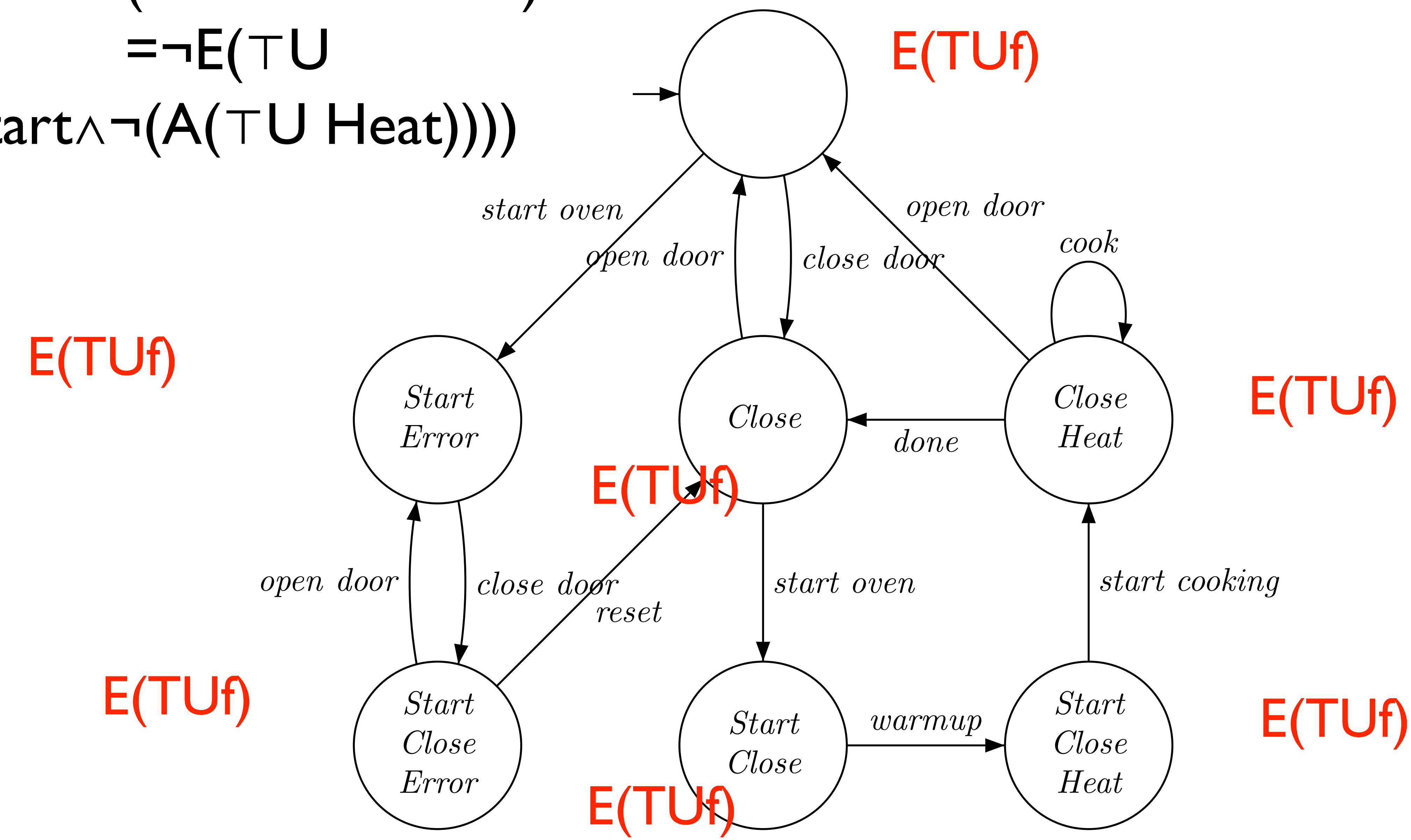
φ =AG(Start→AF Heat)
=¬E(⊤U
(Start∧¬(A(⊤U Heat))))



[Model-Checking, Clarke, Grumberg, Peled]

Exemple : un four à microondes

φ =AG(Start→AF Heat)
=¬E(⊤U
(Start∧¬(A(⊤U Heat))))

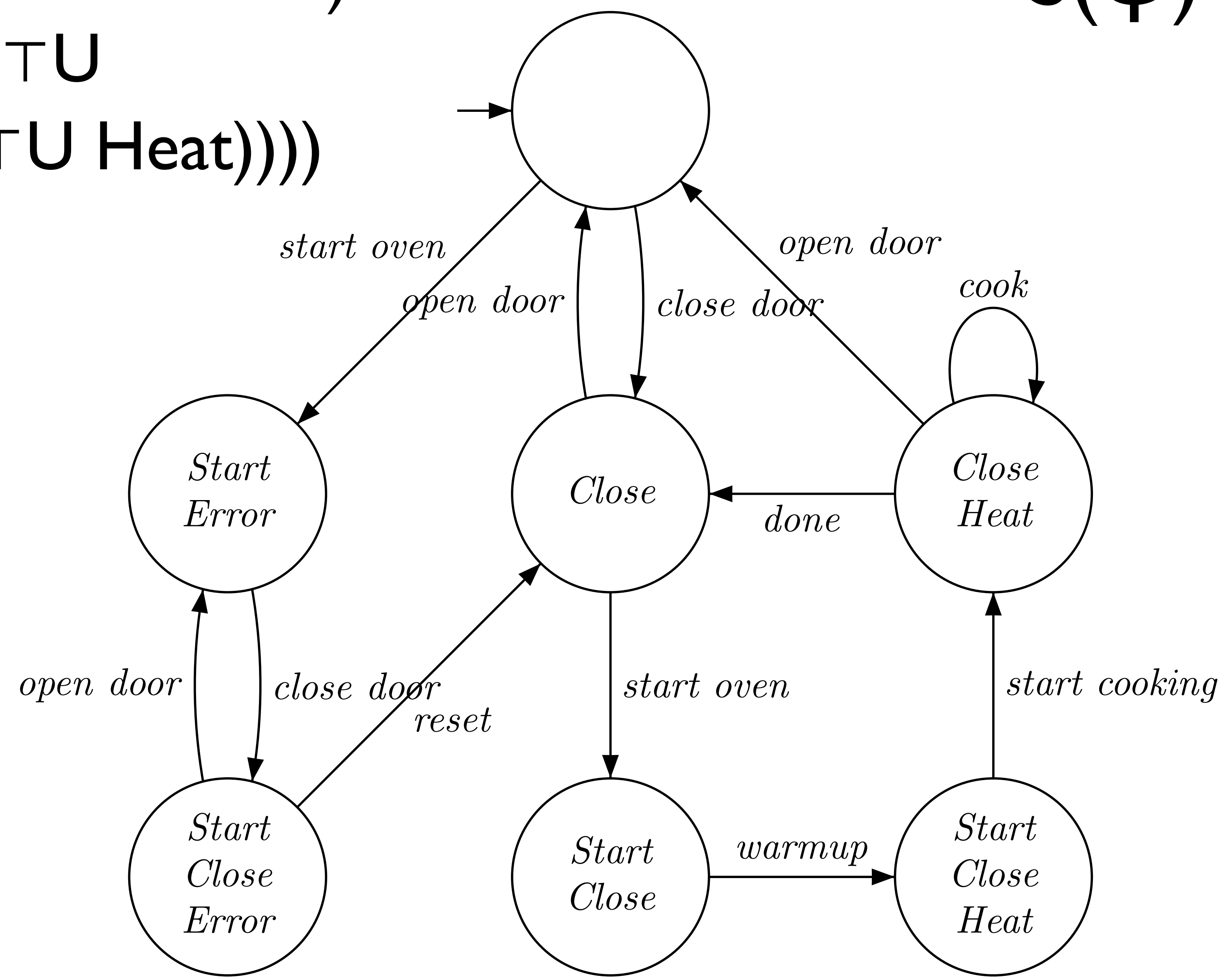


[Model-Checking, Clarke, Grumberg, Peled]

Exemple : un four à microondes

φ =AG(Start→AF Heat)
=¬E(⊤U
(Start∧¬(A(⊤U Heat))))

S(φ)=∅



[Model-Checking, Clarke, Grumberg, Peled]

Complexité

- Complexité en temps est $O(|M|.|\varphi|)$
- MAIS formules CTL peuvent être plus grosses que formules LTL!

3.3 Inclure des notions d'équité

Exécutions équitables

- Chaque processus est activé infiniment souvent : $\wedge_i$ (GF enabled$_i$)
- Aucun processus ne reste infiniment dans la section critique : $\wedge_i\neg(\mathrm{FGcritic}_i)= \wedge_i\mathrm{GF}(\neg\mathrm{critic}_i)$

Contraintes d'équité

- Contrainte d'équité inconditionnelle : $GF\varphi$
- Contrainte d'équité forte : $GF\varphi \rightarrow GF\varphi'$
- Contrainte d'équité faible : $FG\varphi \rightarrow GF\varphi'$

Conditions d'équité

- Une condition d'équité est une conjonction de contraintes d'équité
- Une condition d'équité est une formule LTL!

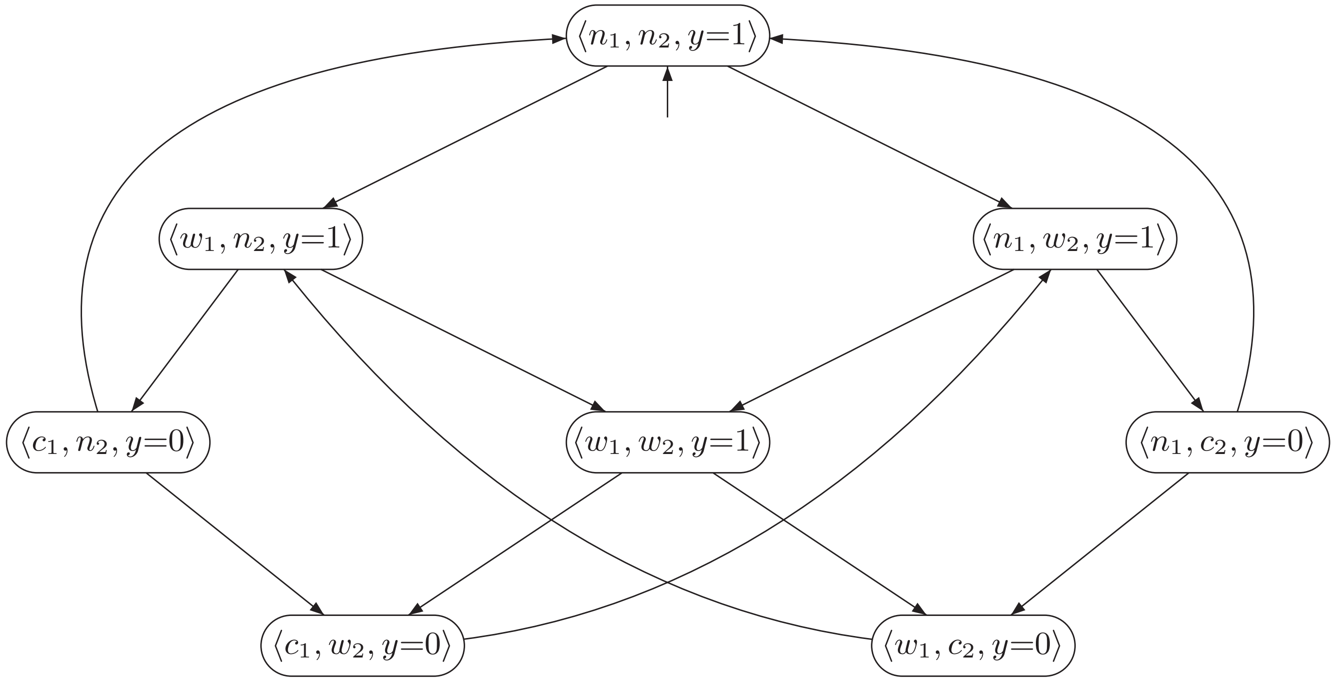
Exécutions équitables

- Soit t une trace d’exécution d’une structure de Kripke M, *fair* une condition d’équité
- t est équitable si $t,0 \models$ *fair*

LTL équitable

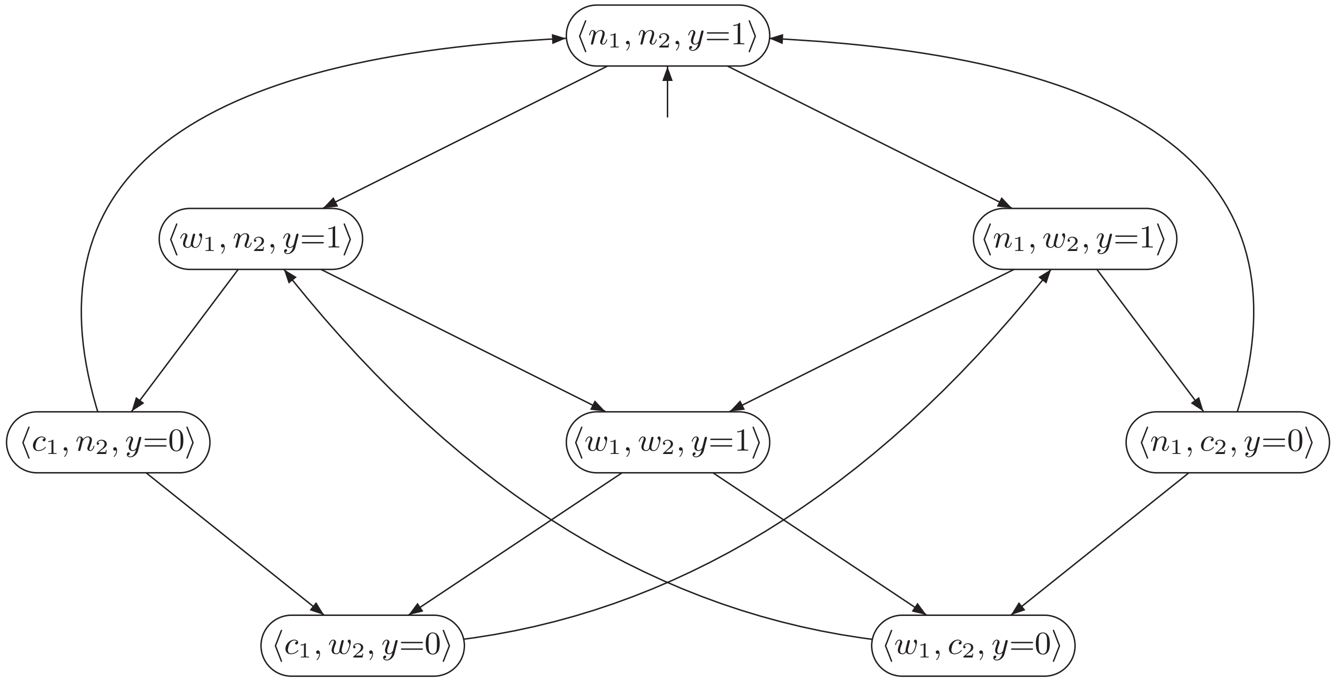
- Soit une structure de Kripke M, *fair* une condition d'équité et φ une formule LTL.
- $M \models_{fair} \varphi$ ssi $t,0 \models_{fair} \varphi$ pour toute trace initiale t de M ssi $t,0 \models \varphi$ pour toute trace initiale équitable de M.

Exemple


$\langle n_1, n_2, y{=}1\rangle$
$\langle w_1, n_2, y{=}1\rangle$
$\langle n_1, w_2, y{=}1\rangle$
$\langle c_1, n_2, y{=}0\rangle$
$\langle w_1, w_2, y{=}1\rangle$
$\langle n_1, c_2, y{=}0\rangle$
$\langle c_1, w_2, y{=}0\rangle$
$\langle w_1, c_2, y{=}0\rangle$

Exemple

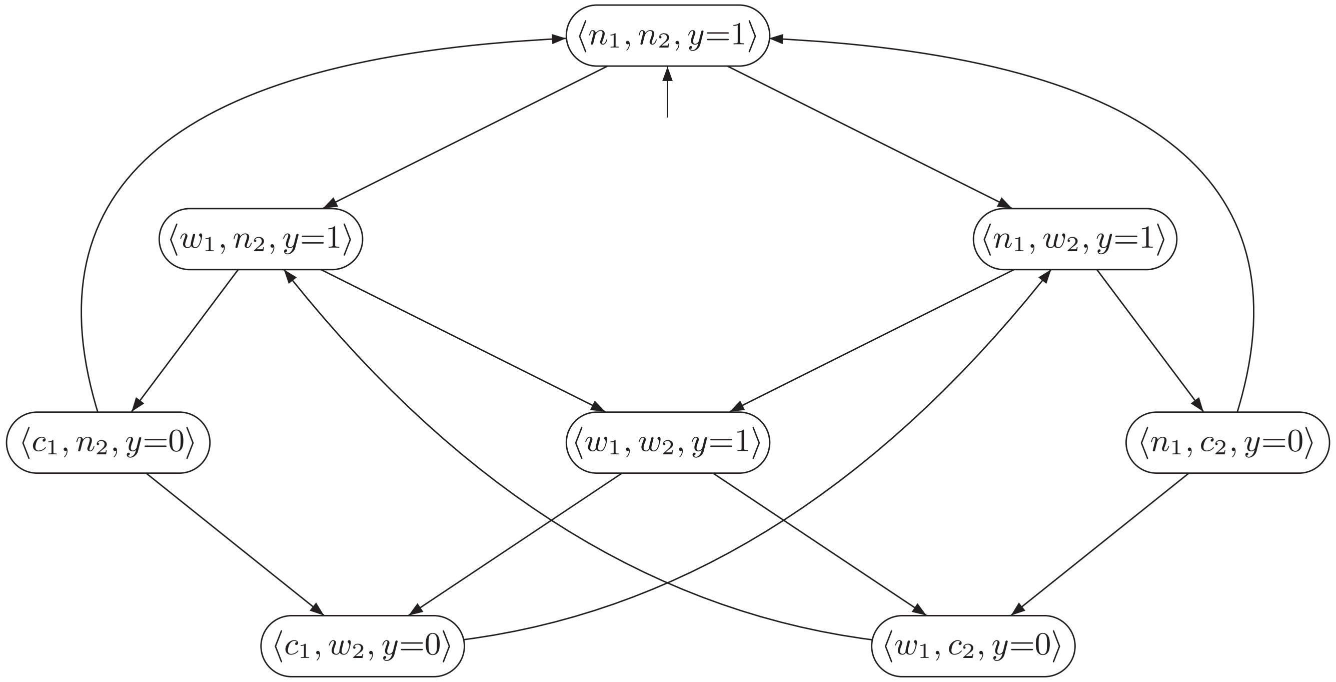
$GF(w_1 \land \neg c_2) \rightarrow GFc_1 \land GF(w_2 \land \neg c_1) \rightarrow GFc_2$



Exemple

$GF(w_1 \wedge \neg c_2) \rightarrow GFc_1 \wedge GF(w_2 \wedge \neg c_1) \rightarrow GFc_2$

$\wedge$

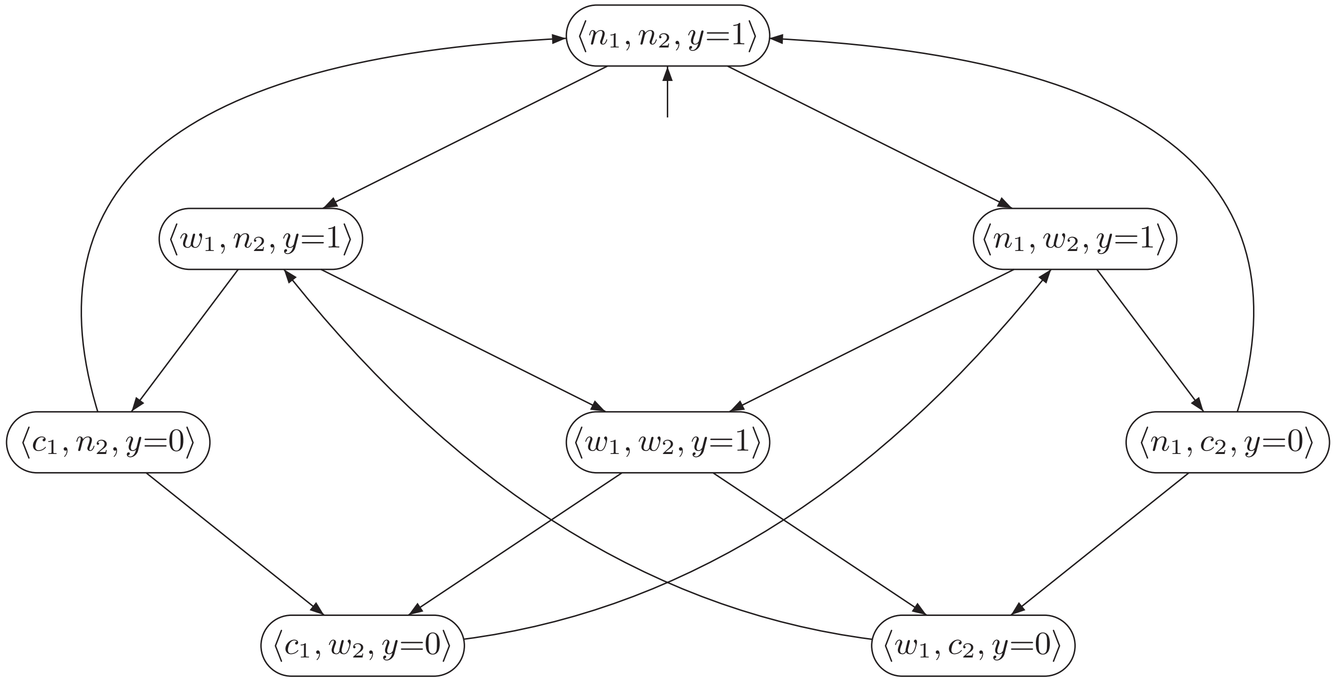


Exemple

$GF(w_1 \wedge \neg c_2) \rightarrow GFc_1 \wedge GF(w_2 \wedge \neg c_1) \rightarrow GFc_2$

$\wedge$

$(FGn_1 \rightarrow GFw_1) \wedge (FGn_2 \rightarrow GFw_2)$

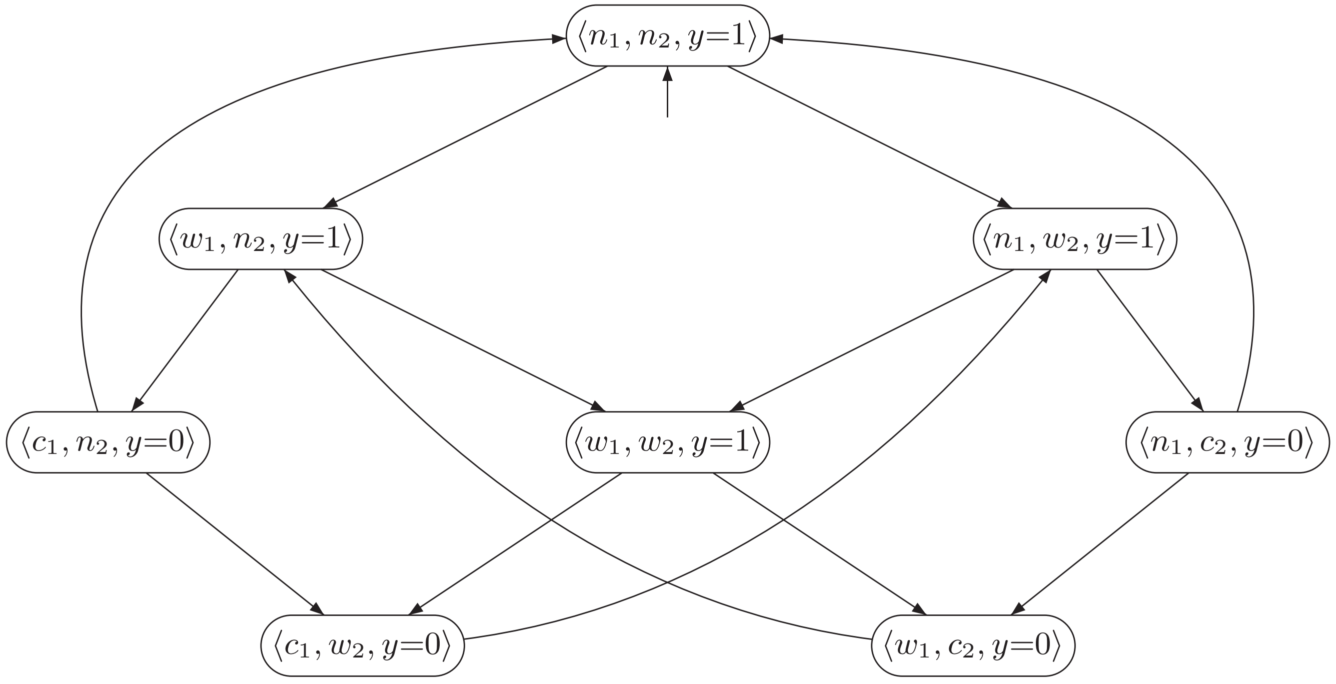


Exemple

$GF(w_1 \wedge \neg c_2) \rightarrow GFc_1 \wedge GF(w_2 \wedge \neg c_1) \rightarrow GFc_2$

$\wedge$

$(FGn_1 \rightarrow GFw_1) \wedge (FGn_2 \rightarrow GFw_2)$



$M \models_{fair} GFc_1 \wedge GFc_2$

Model-Checking LTL équitable

Théorème : $M \models_{fair} \varphi$ ssi $M \models \mathit{fair} \rightarrow \varphi$.

CTL équitable

- Conditions d'équité ne peuvent pas s'écrire en CTL
- On voudrait dire $A(\mathit{fair} \rightarrow \varphi)$ ou $E(\mathit{fair} \wedge \varphi)$ mais ce sont des formules CTL*

CTL équitable

$\varphi ::= p \in AP \mid \neg\varphi \mid \varphi \vee \varphi$
$\mid E_f X\varphi \mid A_f X\varphi \mid E_f \varphi U \varphi \mid A_f \varphi U \varphi$

$s \models p$ ssi $p \in l(s)$

$s \models \neg\varphi$ ssi $s \not\models \varphi$

$s \models \varphi_1 \vee \varphi_2$ ssi $s \models \varphi_1$ ou $s \models \varphi_2$

$s \models E_f X\varphi$ ssi il existe une exécution équitable $s_0 s_1 \ldots$ tel que $s_0 = s$, t.q. $s_1 \models \varphi$

$s \models A_f X\varphi$ ssi s', pour toute exécution équitable $s_0 s_1 \ldots$ telle que $s_0 = s$, $s_1 \models \varphi$

$s \models E_f \varphi_1 U \varphi_2$ ssi il existe une exécution équitable $s_0 s_1 \ldots s_k$ tel que $s_0 = s$, $s_k \models \varphi_2$ et pour tout $0 \leq i \leq k$, $s_i \models \varphi_1$.

$s \models A_f \varphi_1 U \varphi_2$ ssi pour toute exécution équitable $s_0 s_1 \ldots$ telle que $s_0 = s$, il existe k t.q. $s_k \models \varphi_2$ et pour tout $0 \leq i \leq k$, $s_i \models \varphi_1$.

Model-Checking de CTL équitable

- On suppose qu'on a étiqueté les états avec une nouvelle AP fair, qui indique s'il existe une exécution équitable partant de l'état
- $s \models E_f X\varphi$ ssi $s \models EX(\varphi \wedge fair)$
- $s \models A_f X\varphi$ ssi $s \models AX(\neg\, fair \vee \varphi)$
- $s \models E_f \varphi U \varphi'$ ssi $s \models E\varphi U(fair \wedge \varphi')$
- $s \models A_f \varphi U \varphi'$ ssi $s \models \neg E_f G \neg \varphi' \wedge \neg E_f(\neg \varphi' U(\neg \varphi \wedge \neg \varphi'))$
  ssi $s \models \neg E_f G \neg \varphi' \wedge \neg E(\neg \varphi' U(fair \wedge \neg \varphi \wedge \neg \varphi'))$

Model-Checking de CTL équitable

- 1er problème : Comment calculer fair?
- Rappel : s⊨fair ssi il existe une exécution équitable partant de s
- → Dépend de la condition d'équité!
- 2ème problème : calculer $E_fG\varphi$

Calculer fair : les composantes fortement connexes

Définition : Dans un graphe, une composante fortement connexe (SCC) est un sous-graphe maximal tel que pour toute paire de noeuds (s,s’) s’ est accessible depuis s, et s est accessible depuis s’

L’algorithme de Tarjan permet de calculer les SCC d’un graphe en temps linéaire.

Calculer fair : le cas inconditionnel

- On considère une condition d'équité de la forme $\wedge_i GF\varphi_i$, avec φ_i formule CTL.
- On marque les états par les φ_i.
- On calcule les SCC de M par l'algorithme de Tarjan.
- Soit S' l'union des SCC qui intersecte $S(\varphi_i)$, pour tout i.
- fair est l'ensemble des états pouvant atteindre S'.
- (accessibilité se calcule en temps linéaire)

Calculer $E_f G\varphi$: le cas inconditionnel

- Effectuer mark(φ).
- Soit $M(\varphi)$ la restriction de M aux états de $S(\varphi)$.
- Calculer les SCC de $M(\varphi)$ (algo de Tarjan).
- Soit S' l'union de SCC de $M(\varphi)$ intersectant $S(\varphi_i)$, pour tout i.
- $M,s \models E_f G\varphi$ ssi $M,s \models E\varphi U S'$ ssi $M(\varphi),s \models EFS'$.
- $\rightarrow$problème d'accessibilité.

Model-Checking de CTL équitable

- On suppose qu'on a étiqueté les états avec une nouvelle AP fair, qui indique s'il existe une exécution équitable partant de l'état

- $s \models E_f X\varphi$ ssi $s \models EX(\varphi \wedge fair)$

- $s \models A_f X\varphi$ ssi $s \models AX(\neg\, fair \vee \varphi)$

- $s \models E_f \varphi U \varphi'$ ssi $s \models E\varphi U(fair \wedge \varphi')$

- $s \models A_f \varphi U \varphi'$ ssi $s \models \neg E_f G \neg\varphi' \wedge \neg E_f(\neg\varphi' U(\neg\varphi \wedge \neg\varphi'))$
  ssi $s \models \neg E_f G \neg\varphi' \wedge \neg E(\neg\varphi' U(fair \wedge \neg\varphi \wedge \neg\varphi'))$

Model-Checking de CTL équitable

- On suppose qu'on a étiqueté les états avec une nouvelle AP fair, qui indique s'il existe une exécution équitable partant de l'état ✔
- $s \models E_f X\varphi$ ssi $s \models EX(\varphi \wedge fair)$
- $s \models A_f X\varphi$ ssi $s \models AX(\neg\, fair \vee \varphi)$
- $s \models E_f \varphi U \varphi'$ ssi $s \models E\varphi U(fair \wedge \varphi')$
- $s \models A_f \varphi U \varphi'$ ssi $s \models \neg E_f G \neg\varphi' \wedge \neg E_f(\neg\varphi' U(\neg\varphi \wedge \neg\varphi'))$ ssi $s \models \neg E_f G \neg\varphi' \wedge \neg E(\neg\varphi' U(fair \wedge \neg\varphi \wedge \neg\varphi'))$

Model-Checking de CTL équitable

- On suppose qu'on a étiqueté les états avec une nouvelle AP fair, qui indique s'il existe une exécution équitable partant de l'état ✔
- $s \models E_f X \varphi$ ssi $s \models EX(\varphi \wedge fair)$
- $s \models A_f X \varphi$ ssi $s \models AX(\neg\, fair \vee \varphi)$
- $s \models E_f \varphi U \varphi'$ ssi $s \models E \varphi U (fair \wedge \varphi')$
- $s \models A_f \varphi U \varphi'$ ssi $s \models \neg E_f G \neg \varphi' \wedge \neg E_f(\neg \varphi' U (\neg \varphi \wedge \neg \varphi'))$
  ssi $s \models \neg E_f G \neg \varphi' \wedge \neg E(\neg \varphi' U (fair \wedge \neg \varphi \wedge \neg \varphi'))$ ✔

Et aussi...

- Autres logiques temporelles : CTL*,mu-calcul... (plus expressives), ForSpec, PSL, Sugar... (industrie)
- Méthodes efficaces : méthodes symboliques, techniques de réduction (ordres partiels...)