Initiation au « model checking »

Serge Haddad

LSV, ENS Cachan & CNRS & INRIA

Ecole Temps-Réel, le 30 août 2011 à Brest

Plan

De la vérification de programmes à la vérification de systèmes

Vérification de programmes

- Le formalisme de spécification est un langage de programmation ou un pseudo-langage.
- Les propriétés significatives sont la *correction partielle* et la *terminaison*.
- La correction partielle est exprimée par une formule du logique du premier ordre incluant des opérateurs arithmétiques.

Vérification de systèmes réactifs

- De nouvelles caractéristiques : *concurrence*, *non déterminisme*, *temps*, *probabilité*, etc.
- De nouvelles propriétés : propriétés de *sûreté*, *équité*, *vivacité*, etc.
- Des nouveaux formalismes et de nouvelles logiques.

Modèles formels de systèmes pour la vérification

Expressivité versus décidabilité et complexité

- Les systèmes réels requièrent des modèles très expressifs ...
- ce qui conduit à l'indécidablité ou à une complexité élevée des problèmes.
- Les modèles formels pertinents nécessitent une étape d'abstraction conduite par le modélisateur *(intéressante aussi du point de vue de la modélisation)*.

Caractéristiques souhaitables d'un modèle

- Maîtrise de la complexité : modularité, hiérarchie, raffinement
- Portée des résultats : paramétrisation

Du modèle de conception au modèle de vérification : la sémantique formelle

- Garantie de la validité de la transformation
- Obtenue par des règles opérationnelles associées aux constructeurs

Systèmes de transitions étiquetés

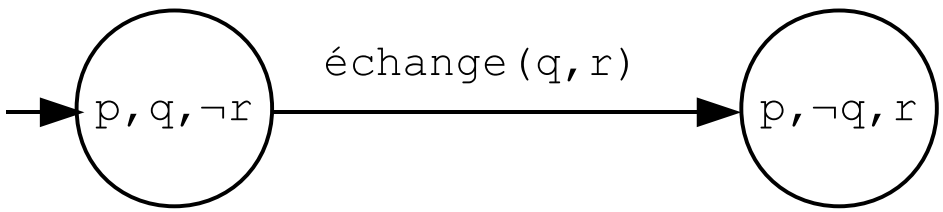
Un formalisme de bas-niveau représentant généralement la sémantique d'un formalisme de haut-niveau.

Un graphe (éventuellement infini) dont les sommets sont les états et les arcs sont les transitions. Lorsqu'il est fini, sa taille est beaucoup plus grande que la spécification de haut-niveau.

Dans un état, chaque *propriété atomique* est ou n'est pas satisfaite.

Une transition est (éventuellement) étiquetée par l'événement qui a déclenché la transition.

Un système de transitions a un sous-ensemble d'états initiaux.



Interprétation d'un système de transitions

Exemples de propriétés atomiques

- La variable x est égale à 9 (comprise entre 13 and 27).
- Le compteur ordinal pointe sur l'instruction Ins.
- Le processus p attend (a acquis) la ressource r.
- Le canal c est vide. Il y a deux occurrences du message m dans le canal c.

Exemples d'événements

- Le processus p remet à zéro la variable x.
- Le processus p libère la ressource r.
- Le processus p envoie (reçoit) le message m vers (depuis) le canal c.

Spécification de propriétés

Propriétés génériques *(jamais exhaustives)*

- Interprétation valide quelque soit le modèle (e.g. blocage)
- Algorithmes efficaces car spécialisés

Propriétés spécifiques *(plus difficiles à vérifier)*

- Propriétés relatives au modèle (e.g. toute requête sera servie en au plus 3 s.)
- exprimées dans des langages à base de logique.

Equivalence de modèles *(importance de l'étape de modélisation)*

- Valide le processus de modélisation.
- Par exemple, équivalence des modèles d'un service et du protocole associé.

Méthodes de vérification

Méthodes automatiques versus semi-automatiques

- Coût des méthodes automatiques faible mais champ d'application plus limité.
- Les méthodes semi-automatiques peuvent traiter des problèmes indécidables mais requièrent des experts et consomment plus de temps.

Méthodes structurelles

- Les résultats fournissent plus d'informations.
- Les algorithmes sont efficaces et intègrent souvent la paramétrisation.
- Semi-algorithmes ou algorithmes pour des sous-classes des formalismes.

Méthodes comportementales

- Implémentation facile.
- Applicables aux modèles finis et à certains modèles infinis.
- Algorithmes plus élaborés pour prendre en compte l'explosion combinatoire.

Plan

Propriétés génériques des systèmes de transitions

Accessibilité : Un état s' est-il accessible depuis un état (initial) s ?

Blocage : Existe-t-il un état sans successeur accessible depuis un état (initial) s ?

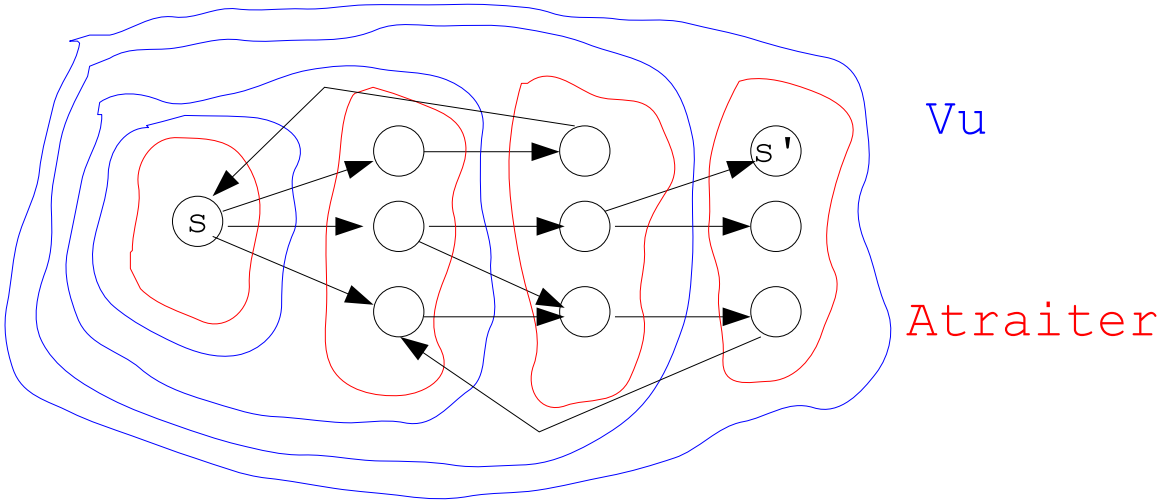
Quasi-vivacité : Un événement e peut-il apparaître à partir d'un état (initial) s ?

Vivacité : Un événement e peut-il apparaître à partir de tout état accessible depuis un état (initial) s ?

Equité : Existe-t-il une séquence infinie à partir d'un état (initial) s dont un suffixe ne rencontre jamais un sous-ensemble d'événements E ?

Accessibilité (1)

Une approche par saturation



$Search()$

$Atraiter \leftarrow Vu \leftarrow \{s\};$

While $s' \notin Vu$ **and** $Atraiter \neq \emptyset$ **do**

$Atraiter \leftarrow Successeur(Atraiter) \setminus Vu;$

$Vu \leftarrow Vu \cup Atraiter;$

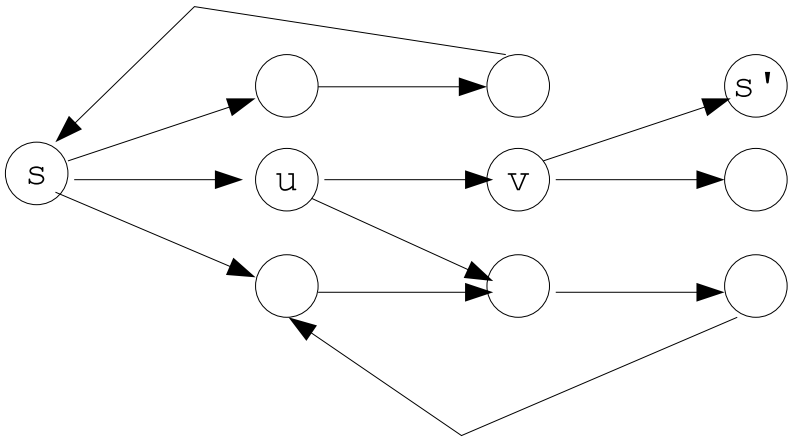
Return$(s' \in Vu);$

Accessibilité (2)

Une approche par exploration limitée

$Search()$
  Return$(search(s, s', |S| - 1))$;

$search(scur, snext, l)$
  If $scur = snext$
  or $snext \in Successor(scur)$ **then**
    Return$(true)$;
  Else If $l \leq 1$ **then**
    Return$(false)$;
  Else
    For $sint \in S \setminus \{scur, snext\}$ **do**
      If $search(scur, sint, \lfloor l/2 \rfloor)$
      and $search(sint, snext, \lceil l/2 \rceil)$ **then**
        Return$(true)$;
  Return$(false)$;



$search(s, s', 9)$

$search(s, u, 4)$

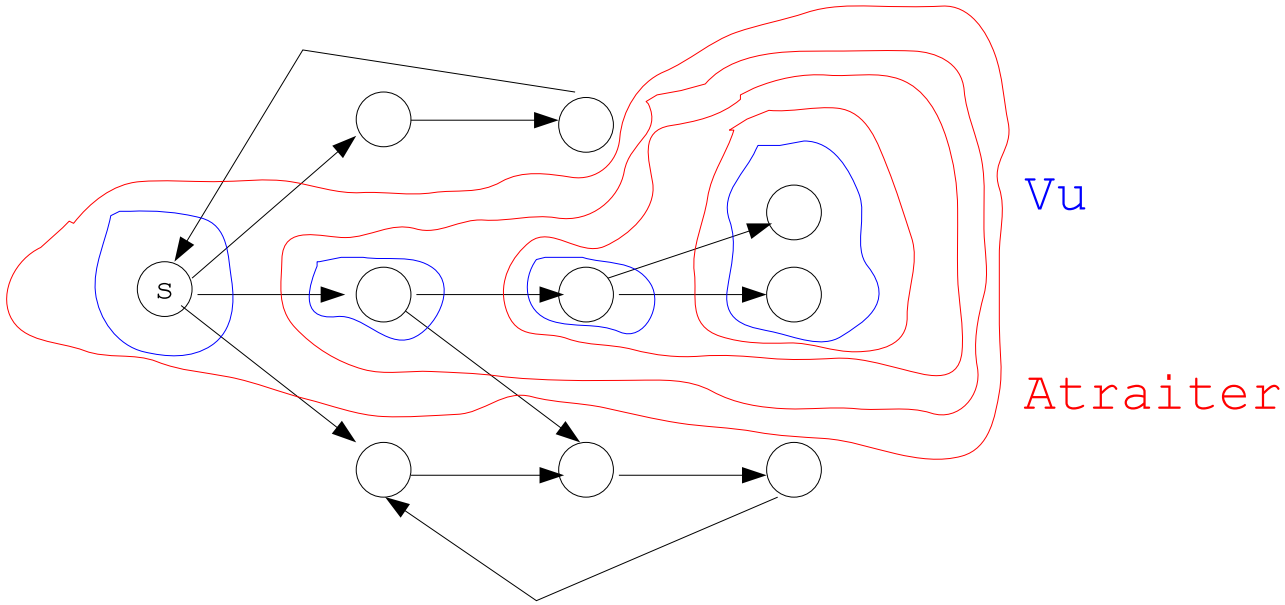
$search(u, s', 5)$

$search(u, v, 2)$

$search(v, s', 3)$

Recherche de blocage

Une approche par saturation avant ou arrière



$Search()$

$Atraiter \leftarrow Vu \leftarrow Morts$;

While $s \notin Vu$ **and** $Atraiter \neq \emptyset$ **do**

$Atraiter \leftarrow Prédecesseur(Atraiter) \setminus Vu$;

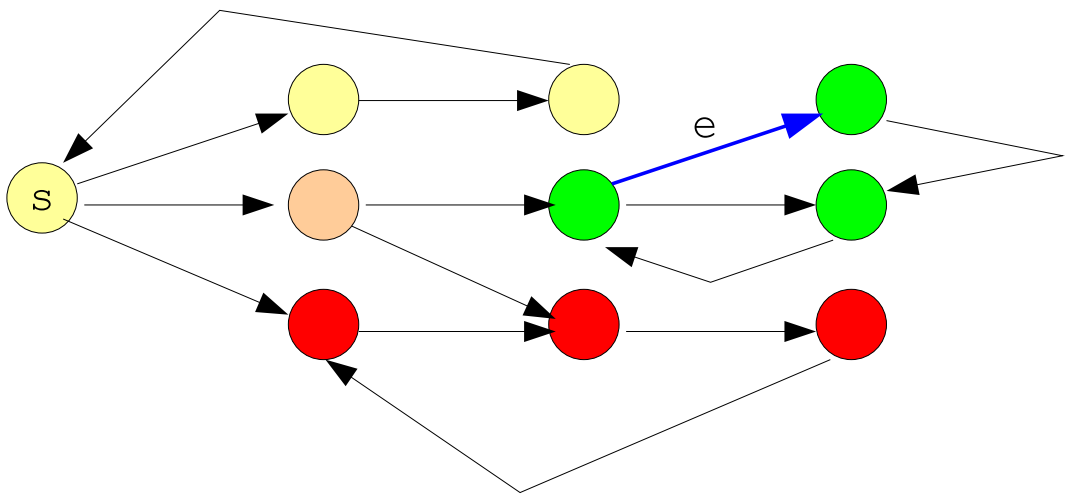
$Vu \leftarrow Vu \cup Atraiter$;

Return$(s \in Vu)$;

Quasi-vivacité et vivacité

Quasi-vivacité Une variante immédiate de l'accessibilité

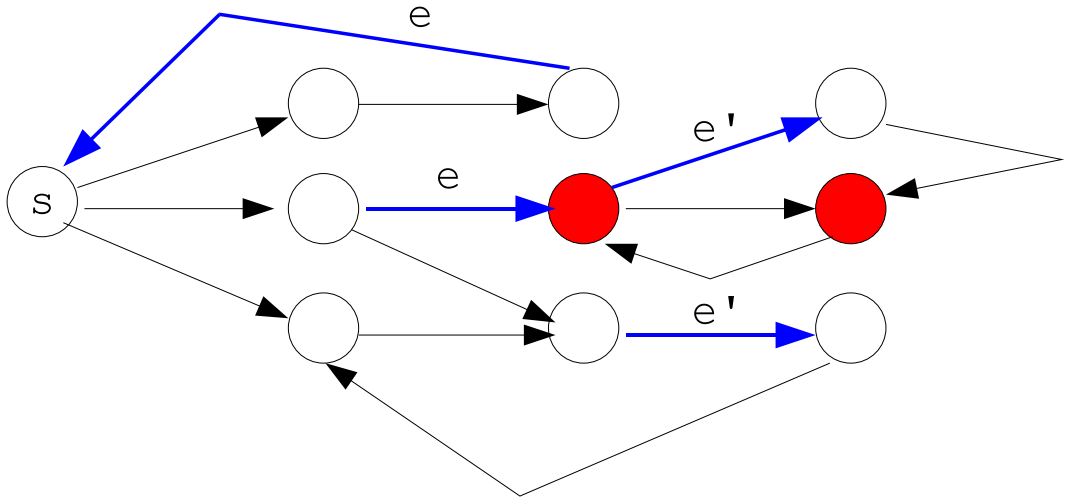
Vivacité Se décide à partir du calcul des composantes fortement connexes



Equité

Methode

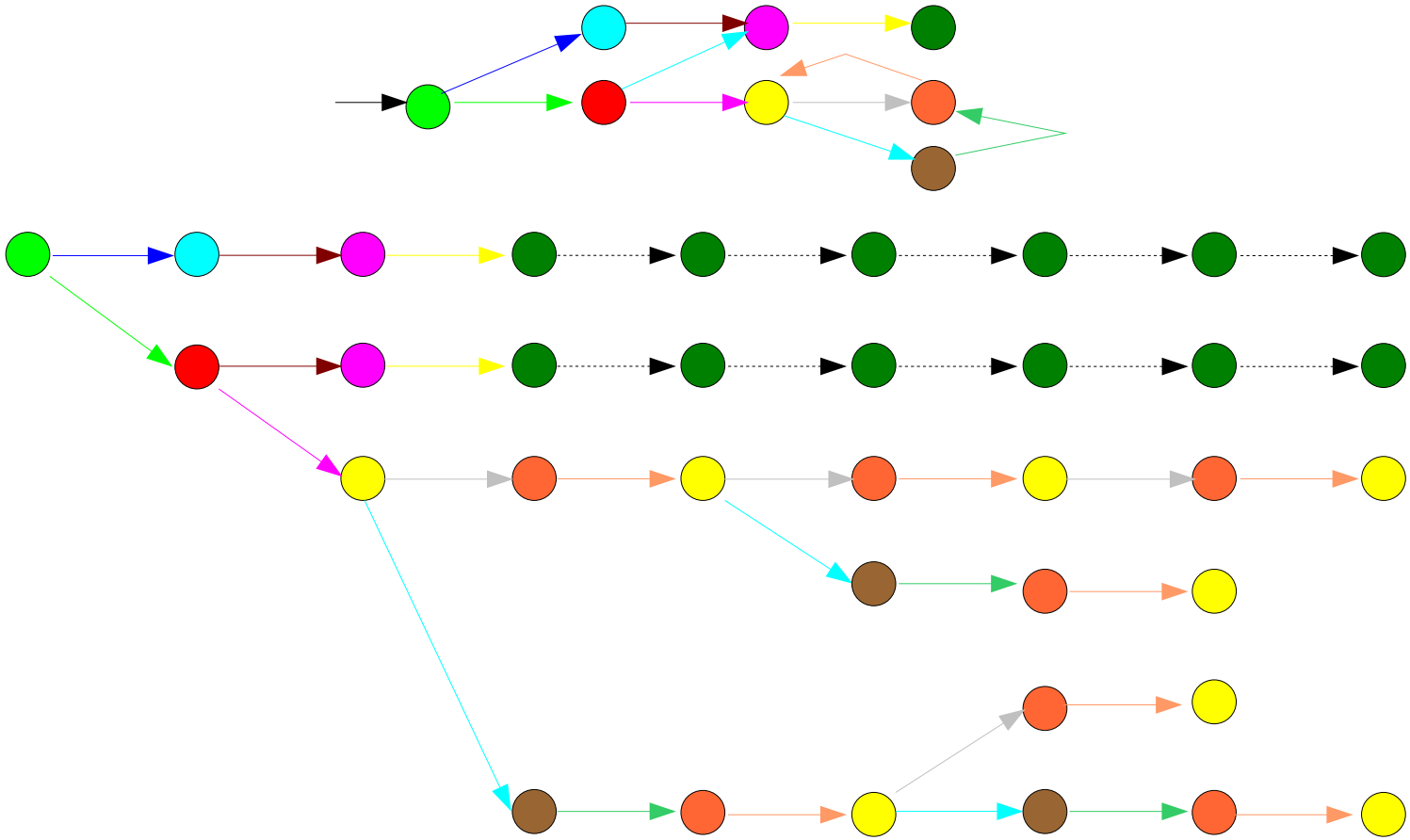
- Construire le graphe d'accessibilité
- Supprimer les arcs étiquetés par E
  (ci-dessous $E = \{e, e'\}$)
- Vérifier que le graphe est acyclique



Plan

Arbre d'exécution

CTL exprime des propriétés relatives à l'arbre d'exécution *infini*.

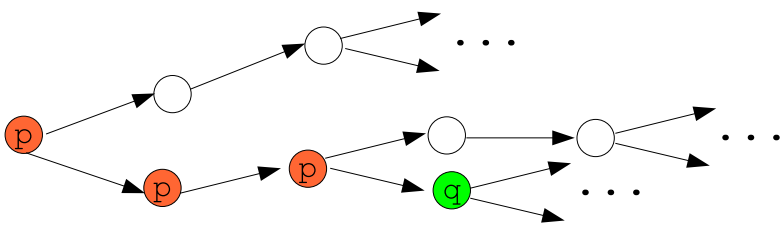


Les opérateurs de CTL (1)

$\mathbf{E}p\mathbf{U}q$

Il existe un chemin (infini) :

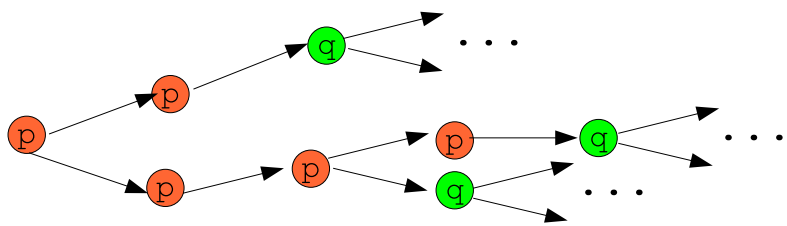
- sur lequel un état vérifie q
- et tous les états précédents vérifient p.



$\mathbf{A}p\mathbf{U}q$

Tous les chemins (infinis) :

- contiennent un état qui vérifie q
- tel que tous les états précédents vérifient p.



Les opérateurs de CTL (2)

$\mathbf{EX}p$ Il existe un chemin dont l'état qui suit l'état initial satisfait p.

$\mathbf{AX}p$ Il existe un chemin dont l'état qui suit l'état initial satisfait p.

Les opérateurs logiques $\vee, \wedge, \neg$

Les propositions atomiques peuvent porter :

- sur les états (logique propositionnelle)
- sur les transitions qui suivent les états dans le chemin (logique événementielle)

Les méthodes de vérification des systèmes finis sont similaires pour les logiques propositionnelle et événementielle.

Les opérateurs de CTL (3)

Quelques abréviations standard

- $\mathbf{EF}p \equiv \mathbf{E}\ true\ \mathbf{U}p$: Il existe un chemin qui contient un état qui vérifie p.
- $\mathbf{AF}p \equiv \mathbf{A}\ true\ \mathbf{U}p$: Tous les chemins contiennent un état qui vérifie p.
- $\mathbf{EG}p \equiv \neg\mathbf{AF}\neg p$: Il existe un chemin dont tous les états vérifient p.
- $\mathbf{AG}p \equiv \neg\mathbf{EF}\neg p$: Tous les états de tous les chemins vérifient p.

Quelques formules standard

- Il existe un état accessible qui vérifie p : $\mathbf{EF}p$
- Il existe un état accessible mort : $\mathbf{EF} \bigwedge_{e \in E} \neg Enabled(e)$
- La transition e est quasi-vivante : $\mathbf{EF}Enabled(e)$
- La transition e est vivante : $\mathbf{AG}\ \mathbf{EF}Enabled(e)$

Vérification de $\mathbf{E}p\mathbf{U}q$: l'algorithme

L'algorithme met à jour un tableau indicé par les états Ver initialisé à **false**.

Il décide pour les états qui satisfont q ...

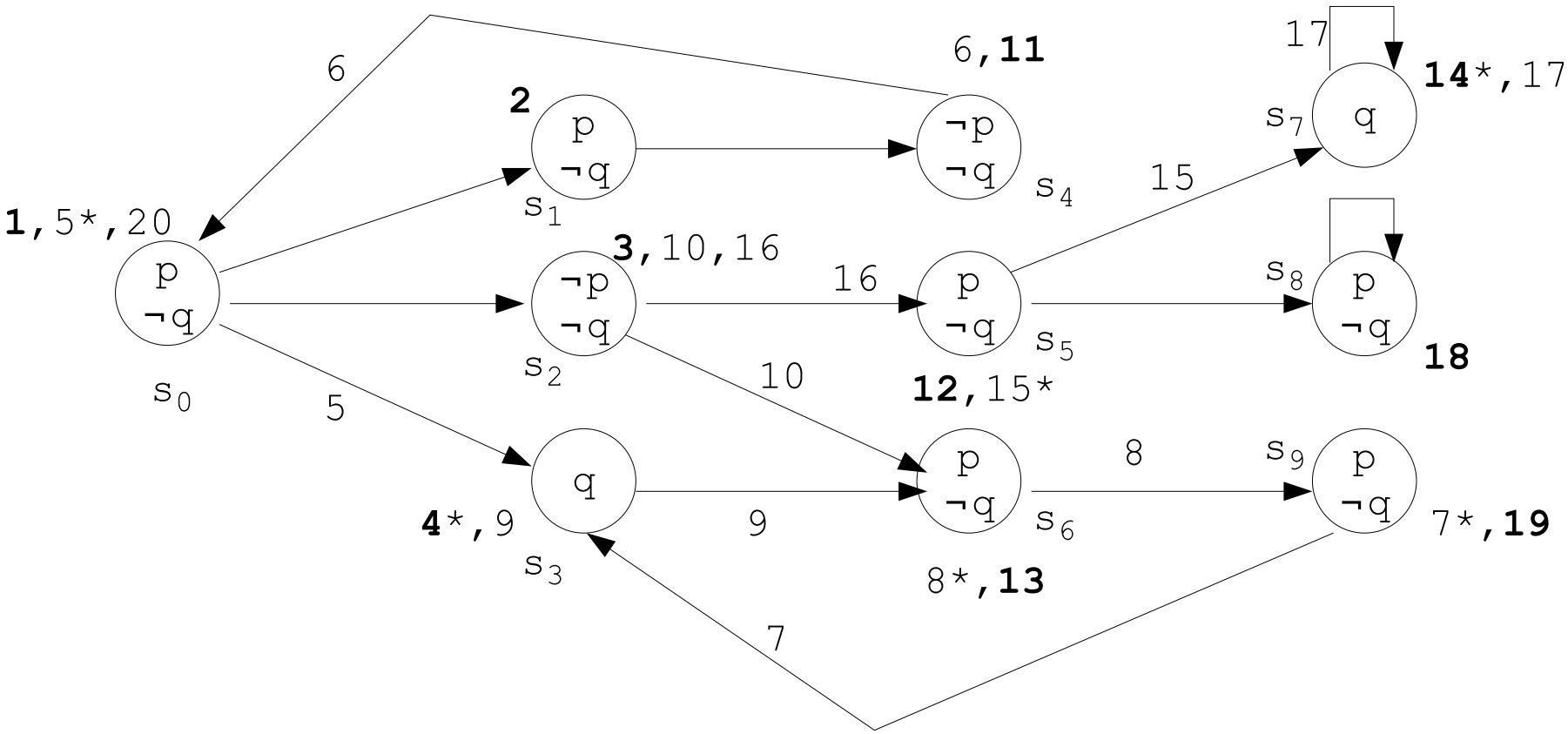
$Verification(p, q)$
For $s \in S$ **do**
If $s \models q \wedge Ver[s] =$ **false then**
$Ver[s] \leftarrow$ **true**
$Verifrec(s, p)$
Return(Ver)

et lance une exploration arrière le long des chemins satisfaisant p.

$Verifrec(s, p)$
For $s' \rightarrow s$ **do**
If $s' \models p \wedge Ver[s'] =$ **false then**
$Ver[s'] \leftarrow$ **true**
$Verifrec(s', p)$
Return

Chaque état est « visité » au plus une fois par chacune des deux procédures et chaque transition est « visitée » au plus une fois par la deuxième procédure.

Vérification de $\mathbf{E}p\mathbf{U}q$: illustration



Vérification de $\mathbf{A}p\mathbf{U}q$: l'algorithme

L'algorithme met à jour un tableau indicé par les états Ver initialisé à **inconnu**.

Il lance pour tous les états de statut inconnu ...

$Verification(p, q)$

For $s \in S$ **do if** $Ver[s] =$ **inconnu then** $Verifrec(s, p, q)$

Return(Ver)

une exploration avant reposant sur l'équivalence $\mathbf{A}p\mathbf{U}q \equiv q \vee (p \wedge \mathbf{AX}\ \mathbf{A}p\mathbf{U}q)$

La détection des circuits « $p \wedge \neg q$ » est assurée grâce au statut **visité**.

$Verifrec(s, p, q)$

$Ver[s] \leftarrow$ **visité**

If $s \models q$ **then** $Ver[s] \leftarrow$ **true**; **Return**$(Ver[s])$

If $s \models \neg p \wedge \neg q$ **then** $Ver[s] \leftarrow$ **false**; **Return**$(Ver[s])$

For $s \rightarrow s'$ **do**

If $Ver[s'] \in \{$**visité**, **false**$\}$ **then**

$Ver[s] \leftarrow$ **false**; **Return**(**false**)

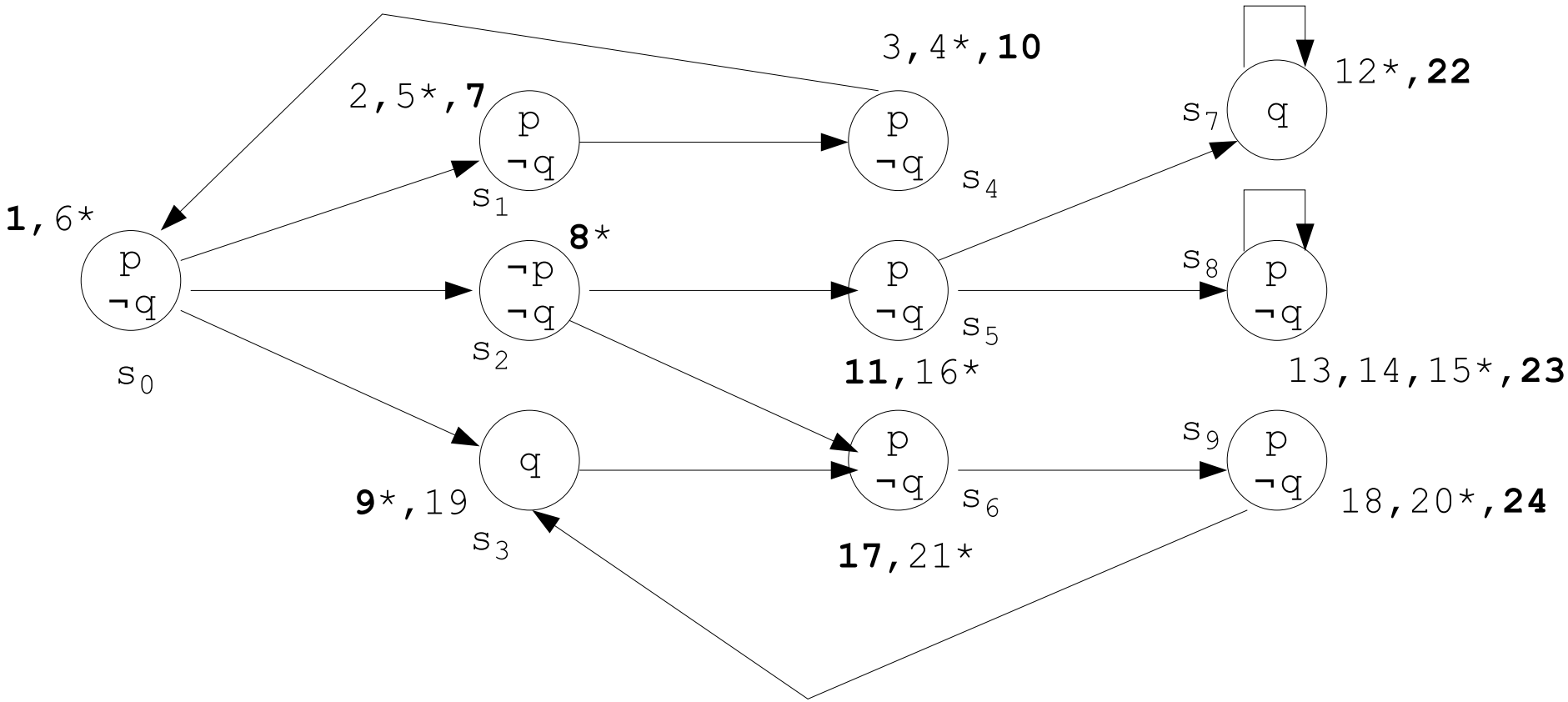
If $Ver[s'] =$ **inconnu and not** $Verifrec(s', p, q)$ **then**

$Ver[s] \leftarrow$ **false**; **Return**(**false**)

$Ver[s] \leftarrow$ **true**; **Return**(**true**)

Chaque état est « visité » au plus une fois par chacune des deux procédures et chaque transition est « visitée » au plus une fois par la deuxième procédure.

Vérification de $\mathrm{A}p\mathrm{U}q$: illustration

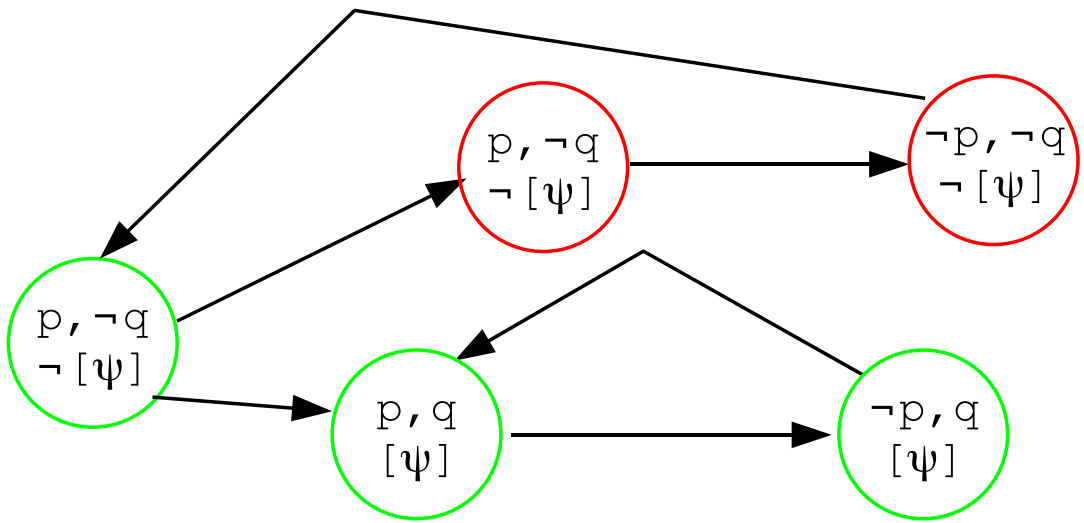


Vérification d'une formule de CTL

Soit une formule φ et un système de transitions $\mathcal{M}$

- Construction de l'arbre syntaxique de φ
- Vérification des sous-formules de φ « des feuilles à la racine »
- avec création de pseudo-propriétés atomiques $[\psi]$ pour chaque sous-formule ψ et étiquetage des états

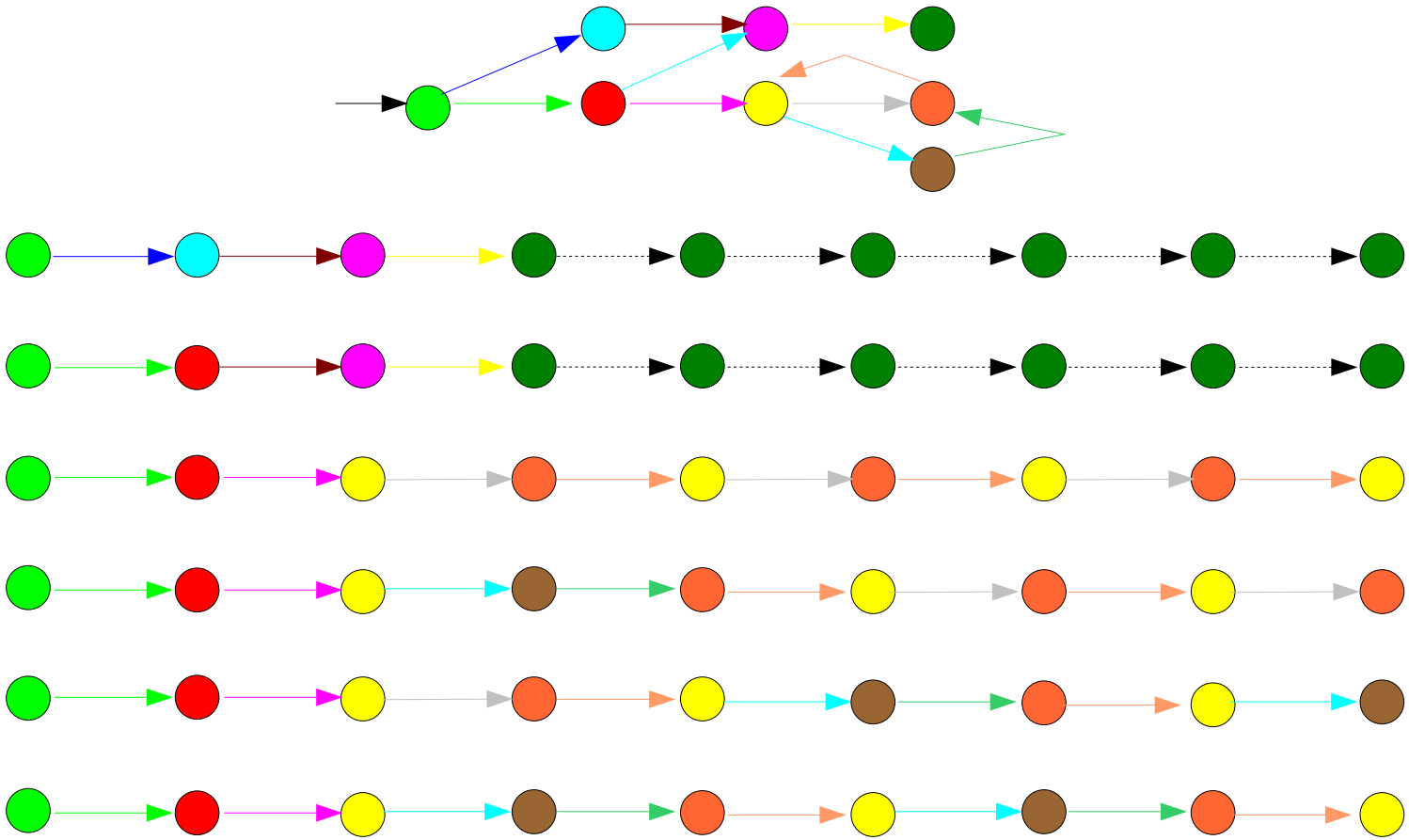
Illustration : $\varphi \equiv \mathbf{E}p\ \mathbf{U}\ \psi$ avec $\psi \equiv \mathbf{EG}q$



Plan

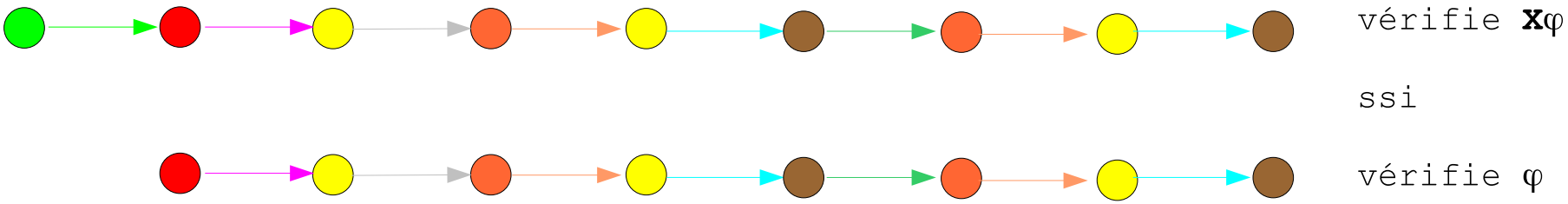
Séquences infinies

LTL exprime des propriétés relatives aux séquences infinies du STE.

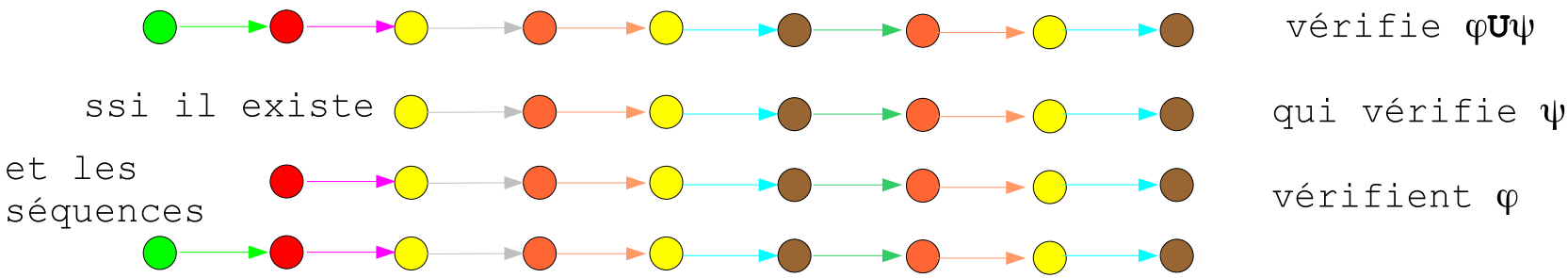


Les opérateurs de LTL (1)

Une séquence vérifie $\mathbf{X}\varphi$ si son plus grand suffixe propre vérifie φ.



Une séquence vérifie $\varphi\ \mathbf{U}\ \psi$ s'il existe un suffixe qui vérifie ψ et tous les plus grands suffixes vérifient φ.



Les opérateurs de LTL (2)

Les opérateurs logiques $\vee, \wedge, \neg$

Quelques abréviations standard

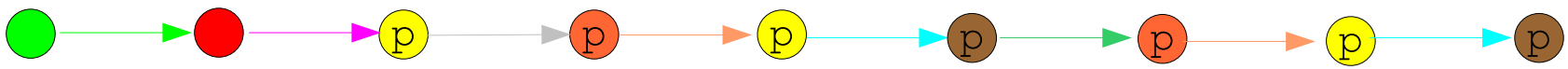
- $\mathbf{F}\varphi \equiv true\ \mathbf{U}\varphi$: Il existe un suffixe du chemin qui vérifie φ.
- $\mathbf{G}\varphi \equiv \neg\mathbf{F}\neg\varphi$: Tous les suffixes du chemin vérifient φ.

Une *formule d'état* de LTL est de la forme $\mathbf{A}\varphi$
où φ désigne une *formule de chemin* de LTL.

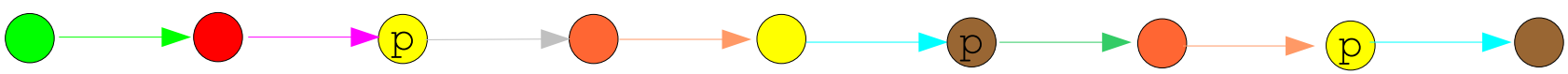
Quelques formules LTL

L'inaccessibilité $\mathbf{AG}\neg p$ (mais pas l'accessibilité)

$\mathbf{AFG}p$ Toute séquence d'exécution se termine.



$\mathbf{AGF}p$ Toute séquence d'exécution sert une infinité de requêtes.



Comment vérifier une formule de LTL ?

Observation

- Une formule de chemin de LTL « accepte » des séquences (infinies)
- Un automate fini « accepte » des mots (finis ou infinis)
- Si les lettres sont des sous-ensembles de propriétés alors les deux notions sont similaires.

Idée : Transformer la formule de LTL en un automate fini qui accepte le même ensemble de séquences.

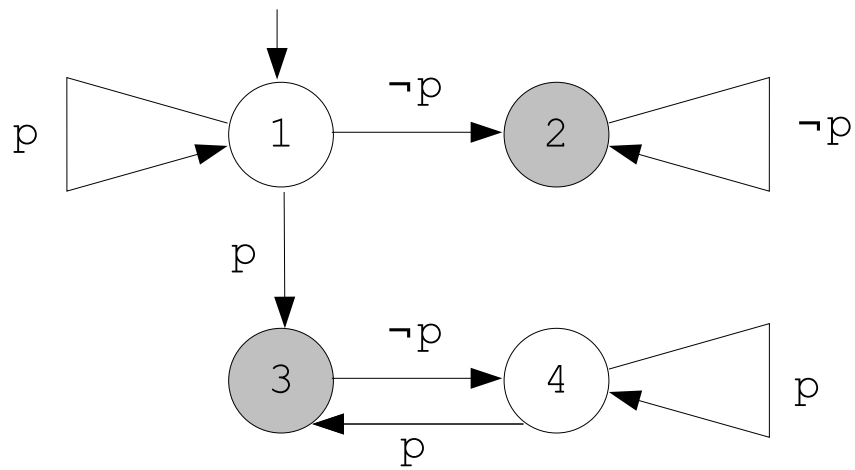
Problème : Quelle condition d'acceptation pour l'automate ?

Automate de Büchi

Un automate de Büchi est un automate dont la condition d'acceptation est spécifiée par un sous-ensemble d'états F.

Un mot infini est accepté par un automate de Büchi s'il existe un chemin étiqueté par ce mot dans l'automate qui rencontre une infinité de fois l'un des états de F.

Le mot $ppp(\neg p)^\omega$ est accepté par cet automate.
Le mot $ppp\neg p\neg p(p)^\omega$ est rejeté par cet automate.



Principe de la vérification de LTL

Construction

- Soit φ une formule de chemin de LTL,
  construire un automate de Büchi $\mathcal{B}$ correspondant à $\neg\varphi$
- Construire le produit synchronisé de $\mathcal{B}$ et du système de transitions $\mathcal{M}$:
  $\mathcal{B} \otimes \mathcal{M}$.
- Le produit synchronisé $\mathcal{B} \otimes \mathcal{M}$ est un automate de Büchi qui accepte les séquences infinies *invalidant* φ.

Vérification

Décider si le produit synchronisé accepte *au moins* une séquence.

De la formule à l'automate : exemple (1)

Soit $\varphi \equiv p\ \mathbf{U}\ (q \wedge\ \mathbf{XG}r)$.

Cette formule est équivalente à $(q \wedge\ \mathbf{XG}r) \vee (p \wedge\ \mathbf{X}\varphi)$.
Une séquence satisfait φ ssi :

- l'état initial de la séquence satisfait q
  et son plus grand suffixe propre satisfait $\mathbf{G}r$
- l'état initial de la séquence satisfait p
  le plus grand suffixe propre satisfait φ.
  (preuve retardée)

$\mathbf{G}r$ est équivalent à $r \wedge \mathbf{XG}r$.
Une séquence satisfait φ ssi
l'état initial de la séquence satisfait r
et son plus grand suffixe propre satisfait $\mathbf{G}r$

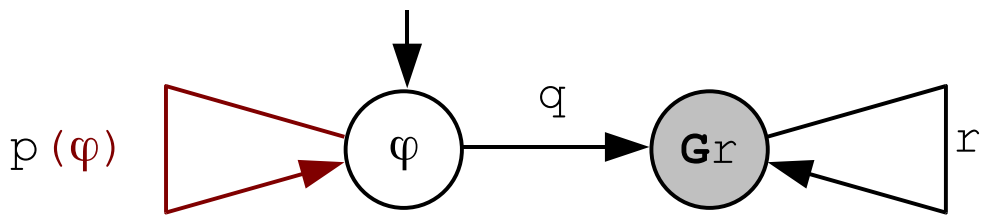
De la formule à l'automate : exemple (2)

Construction d'un automate intermédiaire

- Les états de l'automate sont caractérisés par des ensembles de formules à satisfaire.
- Les transitions de l'automates sont étiquetées par des propriétés atomiques et des preuves retardées.

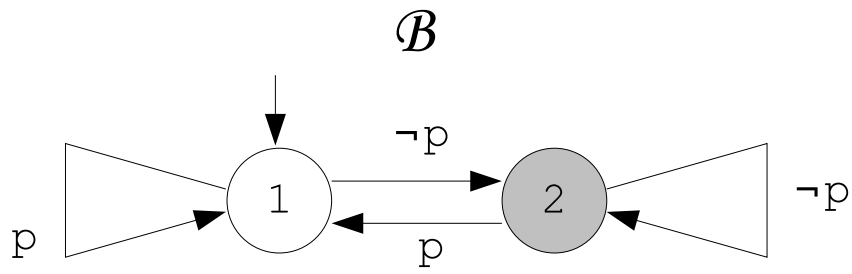
Une séquence est acceptée ssi il existe un chemin qui la reconnaît dans l'automate dans lequel une preuve n'est pas indéfiniment retardée.

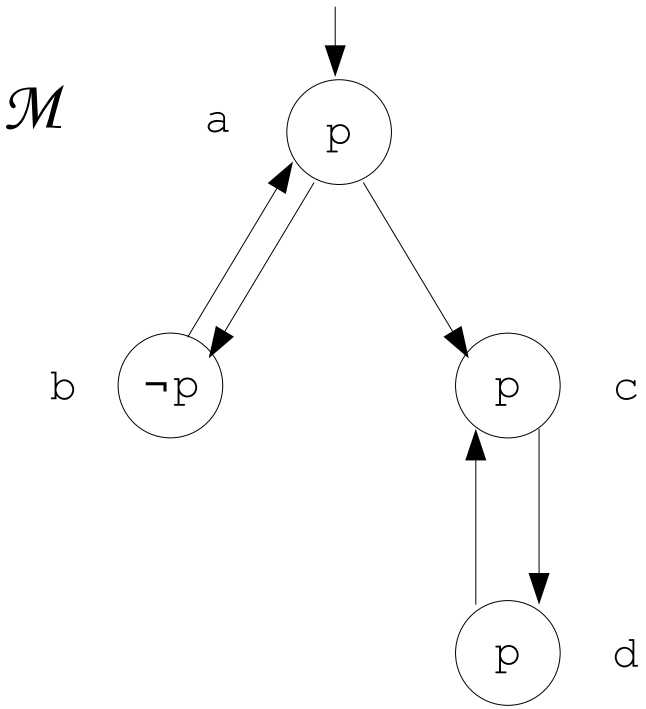
Moyennant une transformation (simple) de l'automate, cette deuxième condition est caractérisée par des états acceptants de Büchi.

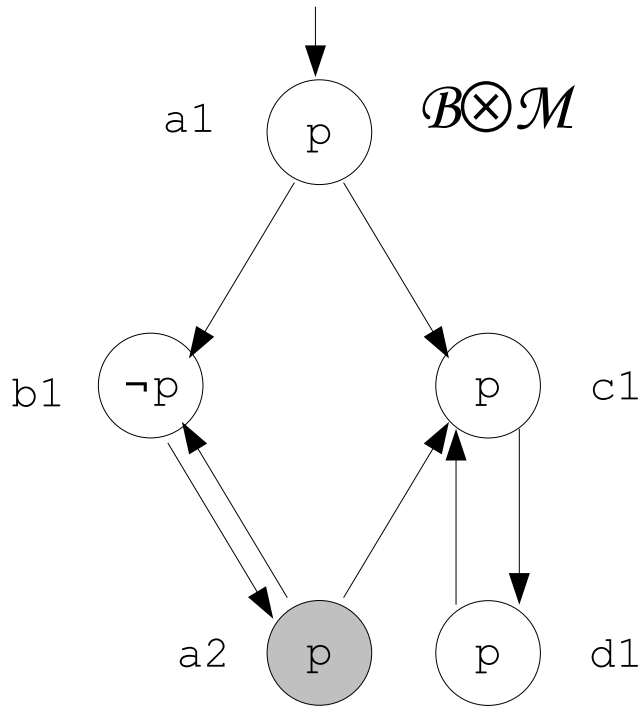


Le produit synchronisé

$\varphi = \mathbf{FG}p$, $\neg\varphi = \mathbf{GF}\neg p$



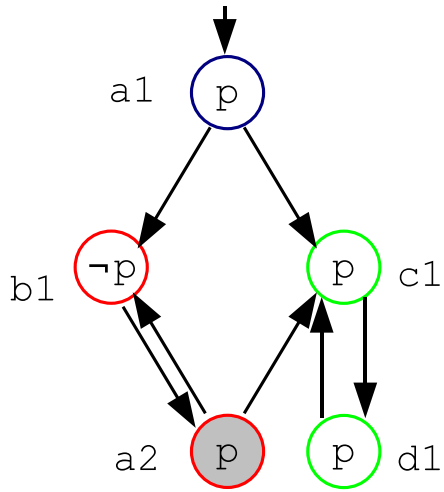




Test de vacuité du produit synchronisé

Il n'y a pas de séquence acceptée par $\mathcal{B} \otimes \mathcal{M}$
ssi
tous les états terminaux accessibles
sont contenus dans des composantes
fortement connexes *triviales*

(i.e. réduites à un état sans boucle)



Peut se décider par deux parcours en profondeur du produit synchronisé.
(sans calcul des composantes)

Complexité en temps

- Linéaire en fonction de la taille du modèle
- Exponentielle en fonction de la taille de la formule

Plan

CTL versus LTL : complexité du model checking

Il y a deux paramètres au problème du model checking :
une formule et un système de transitions

Il y a trois mesures de complexité possibles :

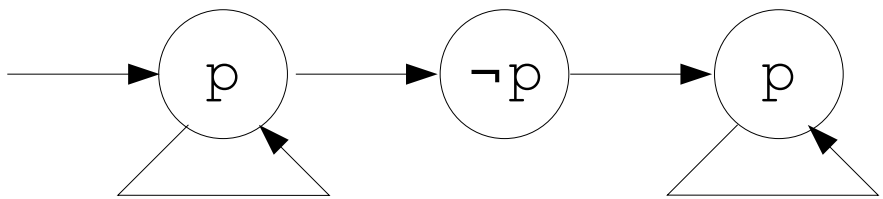
- La complexité vis à vis de la taille de la formule.
- La complexité vis à vis de la taille du système de transitions.
- La complexité conjointe

Logique	Formule	STE	Conjointe
CTL	LOGSPACE	NLOGSPACE-complet	PTIME-complet
LTL	PSPACE-complet	NLOGSPACE-complet	PSPACE-complet

Expressivité comparée : un exemple

AF AGp Sur tout chemin, il existe un état à partir duquel tout état accessible vérifie p.

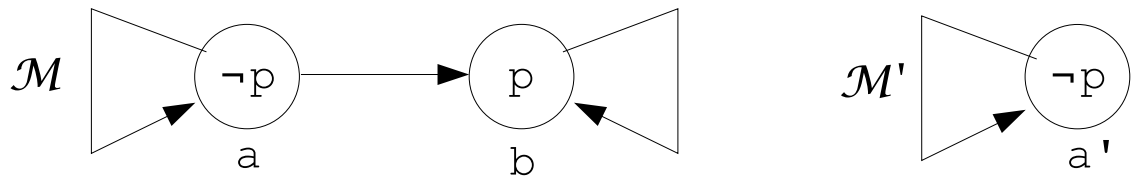
AFGp Sur tout chemin, il existe un état à partir duquel tous les états du chemin vérifient p.



AF AGp n'est pas satisfaite mais **AFG**p est satisfaite.

CTL n'est pas inclus dans LTL

L'accessibilité $\mathbf{EF}p$ n'est pas exprimable en LTL.



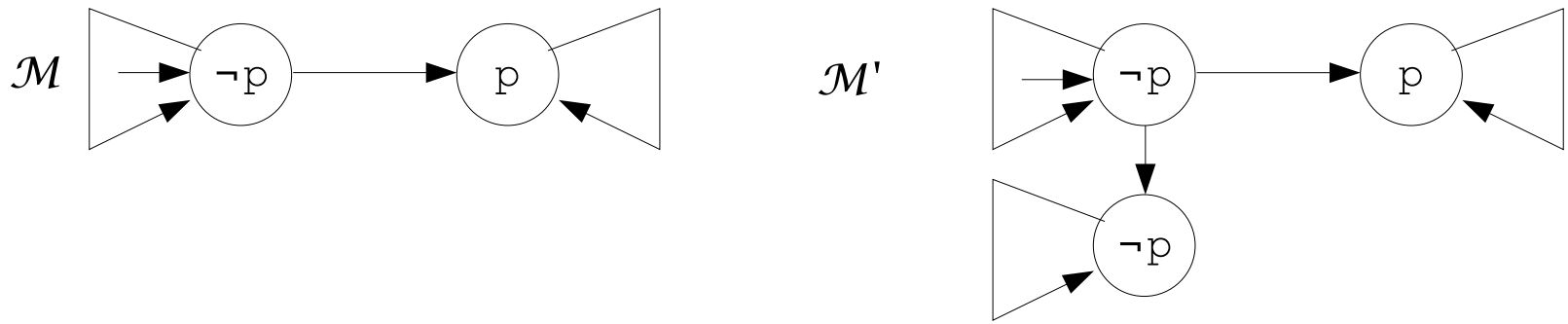
Sketch de preuve

- $\mathcal{M}, a \models \mathbf{EF}p$
- $\mathcal{M}', a' \not\models \mathbf{EF}p$
- Pour toute formule $\varphi \in LTL$, $\mathcal{M}, a \models \varphi$ implique $\mathcal{M}', a' \models \varphi$

Mais sa négation $\mathbf{AG}\neg p$ l'est.

CTL n'est inclus dans aucune logique temporelle linéaire

La vivacité $\mathbf{AG}\ \mathbf{EF}p$ n'est exprimable dans aucune logique temporelle linéaire.

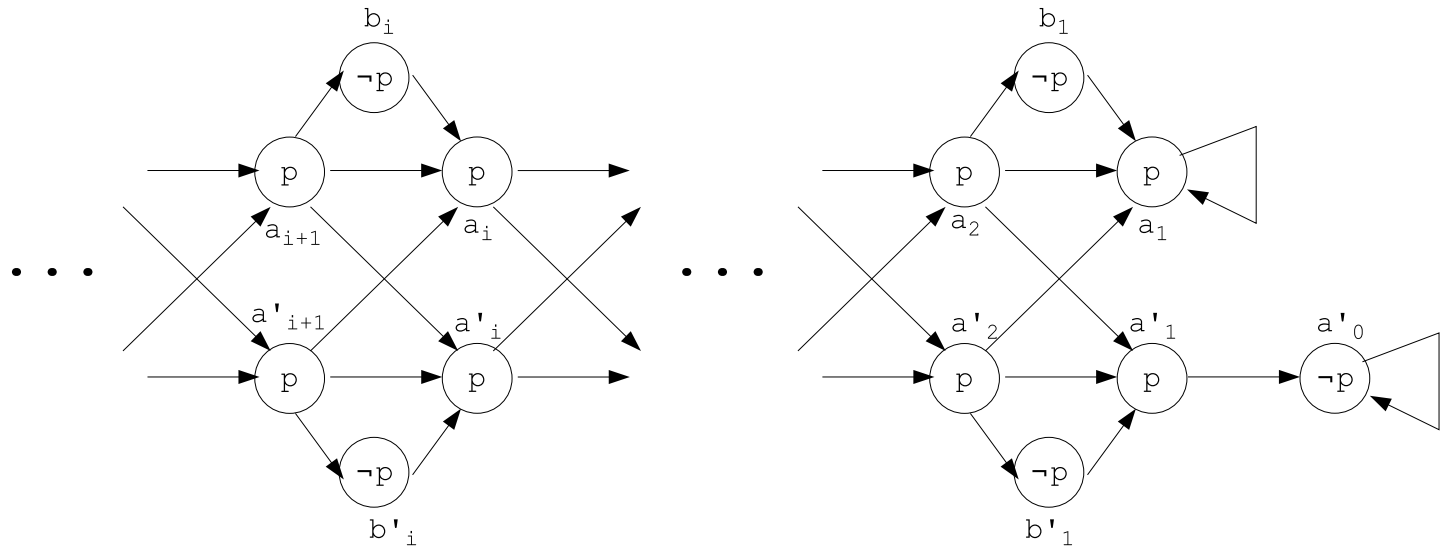


Sketch de preuve

- $\mathcal{M} \models \mathbf{AG}\ \mathbf{EF}p$
- $\mathcal{M}' \not\models \mathbf{AG}\ \mathbf{EF}p$
- $\mathcal{M}$ et $\mathcal{M}'$ ont les mêmes séquences infinies.

LTL n'est pas inclus dans CTL

La formule $\mathbf{AF}(p \wedge \mathbf{X}p)$ (tout chemin infini contient deux états consécutifs qui satisfont p) n'est pas exprimable en CTL.



Sketch de preuve (par induction sur i)

- Pour tout i, $a_i \models \mathbf{AF}(p \wedge \mathbf{X}p)$
- Pour tout i, $a'_i \not\models \mathbf{AF}(p \wedge \mathbf{X}p)$
- Pour toute formule $\varphi \in CTL$ de longueur inférieure ou égale à i, $a_i \models \varphi$ ssi $a'_i \models \varphi$ et $b_i \models \varphi$ ssi $b'_i \models \varphi$

CTL*

CTL* généralise à la fois CTL et LTL.

CTL* consiste en formules de chemin et d'état
définies de façon mutuellement inductive.

Une formule de chemin est :

- une formule d'état *(évaluée sur le premier état du chemin)*
- $\mathbf{X}\varphi$, $\varphi\ \mathbf{U}\ \psi$, où φ et ψ sont des formules de chemin
  (même sémantique que LTL)

Une formule d'état est :

- une propriété atomique
- $\mathbf{A}\varphi$, $\mathbf{E}\varphi$ où φ est une formule de chemin
  (même sémantique que CTL)

Vérification de formules CTL*

Méthode

- Elimination de l'opérateur $\mathbf{E}$ par l'équivalence $\mathbf{E}\varphi \equiv \neg\mathbf{A}\neg\varphi$
- Construction de l'arbre syntaxique de la formule
- Vérification de sous-formules maximales de LTL des feuilles à la racine
- Création de propriétés atomiques $[\mathbf{A}\psi]$ pour chaque sous-formule $\mathbf{A}\psi$ et étiquetage des états du système de transitions

Illustration : $\varphi = \mathbf{A}(\mathbf{FAG}p \wedge \mathbf{GAF}q)$

- Vérification de $\mathbf{AG}p$ et de $\mathbf{AF}q$
- Création des propriétés atomiques $[\mathbf{AG}p]$ et de $[\mathbf{AF}q]$ et étiquetage des états
- Vérification de $\mathbf{A}(\mathbf{F}[\mathbf{Ag}p] \wedge \mathbf{G}[\mathbf{AF}q])$

Complexité temporelle similaire à celle de LTL

- linéaire en fonction de la taille du système de transitions
- exponentielle en fonction de la taille de la formule

Complexité : récapitulatif

Logique	Formule	STE	Conjointe
CTL	LOGSPACE	NLOGSPACE-complet	PTIME-complet
LTL	PSPACE-complet	NLOGSPACE-complet	PSPACE-complet
CTL*	PSPACE-complet	NLOGSPACE-complet	PSPACE-complet

D'un point de vue théorique, il n'y a pas d'intérêt à se limiter à LTL.

D'un point de vue pratique, LTL permet le développement de nombreuses techniques exploitant la concurrence des systèmes.

D'un point de vue pratique, les techniques basées sur les diagrammes de décision (BDD, DDD, ...) s'appliquent naturellement à CTL.

D'autres logiques

CTL avec équité

- ECTL introduit les opérateurs **EGF** et **EFG** sans augmentation de complexité.
- CTL$^+$ et ECTL$^+$ permettent de « relativiser » les quantifications de chemin comme dans $\mathbf{A}(\mathbf{GF}req \Rightarrow \mathbf{GF}serv)$ avec une complexité intermédiaire.

Logique temporelle avec opérateurs du passé

- Deux opérateurs symétriques de **U** et **X** : **S** (*since*) et **Y** (*yesterday*).
- Dans le cas de LTL et de CTL*, même expressivité mais plus de *concision*. La complexité du model checking est identique.
- Dans le cas de CTL, ces opérateurs augmentent l'expressivité. La complexité du model checking devient celle de LTL.

Au-delà de LTL et CTL*. La propriété « p est satisfaite aux instants pairs » n'est pas exprimable en LTL. Plusieurs approches alternatives :

- Un langage à base de plus petit et plus grand point fixes : le mu-calcul.
- Un langage où les formules de chemin s'expriment par des automates : ECTL*.
- Des logiques sans opérateurs temporels mais avec des variables de position (*first-order*) et d'ensemble de positions (*monadic second-order*).

D'autres systèmes

Systèmes infinis

- Réseaux de Petri
- Automates à pile

Systèmes temporisés

- Automates temporisés
- Classes restreintes de systèmes hybrides

Systèmes probabilistes

- Chaînes de Markov
- Processus de décision markoviens

Pour approfondir

Principles of model checking

C. Baier, J-P. Katoen, The MIT Press, 2008

Model checking

E. M. Clarke, O. Grumberg, D. A. Peled, The MIT Press, 2000

Temporal Verification of Reactive Systems : Safety

Z. Manna and A. Pnueli. Springer-Verlag, New York, 1995

The Temporal Logic of Reactive and Concurrent Systems : Specification

Z. Manna and A. Pnueli. Springer-Verlag, 1991.