Tutoriel Model Checking

Stephan Merz

INRIA Lorraine & LORIA
Nancy, France

Ecole d'été temps réel 2005
13 septembre 2005

Plan

Outline

Vérification automatique de logiciels

N.N.

Things like even software verification, this has been the Holy Grail of computer science for many decades

Vérification automatique de logiciels

Bill Gates, WinHEC 2002

Things like even software verification, this has been the Holy Grail of computer science for many decades but now in some very key areas, for example, driver verification we're building tools that can do actual proof about the software and how it works in order to guarantee the reliability.

Vérification automatique de logiciels

Bill Gates, WinHEC 2002

Things like even software verification, this has been the Holy Grail of computer science for many decades but now in some very key areas, for example, driver verification we're building tools that can do actual proof about the software and how it works in order to guarantee the reliability.

Représentation formelle : $\mathcal{K} \stackrel{?}{\models} \varphi$

- $\mathcal{K}$: (modèle du) programme sous observation
- φ : propriété à vérifier
- pré-requis : sémantique (opérationnelle), langage de spécification

Un peu d'histoire

- au début : systèmes transformationnelles
  - correction = correction partielle + terminaison
  - sémantique : relation entre états
  - spécification : logique de première ordre

Un peu d'histoire

- au début : systèmes transformationnelles
  - correction = correction partielle + terminaison
  - sémantique : relation entre états
  - spécification : logique de première ordre
- années 70 : arrivée des systèmes réactifs
  - interaction entre systèmes et environnement
  - terminaison pas important, ou même indésirable
  - correction = sûreté + progrès + équité + ...
  - sémantique : systèmes de transition, structures de Kripke
  - spécification : logiques temporelles

Logique temporelle

- Moyen Age : inférence temporelle en étude des langues

  Hier elle a dit «je viens demain», donc elle viendra aujourd'hui.

Logique temporelle

- Moyen Age : inférence temporelle en étude des langues

  Hier elle a dit «je viens demain», donc elle viendra aujourd'hui.

- 20ème siècle : formalisation de logiques modales et temporelles

  opérateurs : always, eventually, until, since, . . .
  A. Prior : Past, present, and future. Oxford Univ. Press, 1967.

Logique temporelle

- Moyen Age : inférence temporelle en étude des langues

  Hier elle a dit «je viens demain», donc elle viendra aujourd'hui.

- 20ème siècle : formalisation de logiques modales et temporelles

  opérateurs : always, eventually, until, since, . . .
  A. Prior : Past, present, and future. Oxford Univ. Press, 1967.

- 1977 : A. Pnueli utilise TL pour exprimer les propriétés

  interprétation de formules sur les systèmes de transition
  A. Pnueli : The Temporal Logic of Programs. FOCS'77

Logique temporelle

- Moyen Age : inférence temporelle en étude des langues

  Hier elle a dit «je viens demain», donc elle viendra aujourd'hui.

- 20ème siècle : formalisation de logiques modales et temporelles

  opérateurs : always, eventually, until, since, . . .
  A. Prior : Past, present, and future. Oxford Univ. Press, 1967.

- 1977 : A. Pnueli utilise TL pour exprimer les propriétés

  interprétation de formules sur les systèmes de transition
  A. Pnueli : The Temporal Logic of Programs. FOCS'77

Système	vérifie	propriété
	traduit en	
structure de Kripke $\mathcal{K}$	est un modèle de	formule temporelle φ

Mécanisation du problème de vérification

- Réduction au problème de validité (depuis les années 70)
  - point de départ : système de preuve pour la logique
  - générer un ensemble $\mathcal{F}$ de formules caractérisant $\mathcal{K}$
  - démontrer la validité de $\mathcal{F} \vdash \varphi$

problème : décision de la validité est trop coûteuse

Mécanisation du problème de vérification

- Réduction au problème de validité (depuis les années 70)
  - point de départ : système de preuve pour la logique
  - générer un ensemble $\mathcal{F}$ de formules caractérisant $\mathcal{K}$
  - démontrer la validité de $\mathcal{F} \vdash \varphi$

  problème : décision de la validité est trop coûteuse

- Réduction au problème de model checking (début années 80)
  - construire et mémoriser la structure de Kripke $\mathcal{K}$
    $\rightsquigarrow$ restriction aux structures $\mathcal{K}$ finies
  - évaluer la relation $\mathcal{K} \models \varphi$ suivant sa définition inductive

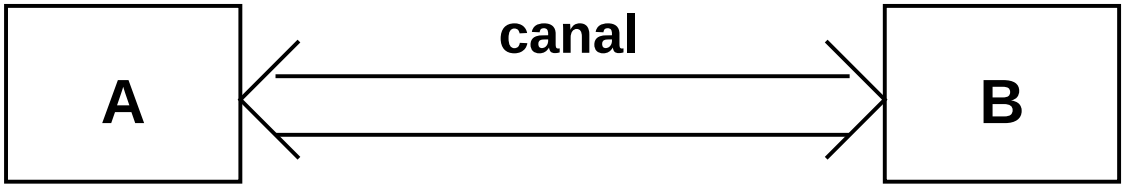
Clarke, Emerson : *Design and synthesis of synchronisation skeletons using branching time temporal logic.* Logics of Programs, LNCS 131, 1981.

Queille, Sifakis : *Specification and verification of concurrent systems in CESAR.*

Intl. Symp. Programming, LNCS 137, 1981.

Outline

Un protocole de communication



- communication bi-directionnelle sur canal non fiable
  - hypothèse : toute erreur de transmission est détectée
- protocole de communication
  - messages dans le format $\langle tag, data \rangle$
  - $tag \in \{\texttt{ack}, \texttt{nak}, \texttt{err}\}$
  - règles du protocole
    - ★ transmission correcte $\rightsquigarrow$ prochain message sera marqué `ack`
    - ★ transmission incorrecte $\rightsquigarrow$ prochain message sera marqué `nak`
    - ★ réception de `nak` ou err $\rightsquigarrow$ retransmission du message
    - ★ réception de `ack` $\rightsquigarrow$ transmission du message suivant
    - ★ début par un message `err`

Outline

Systèmes de transition

- Modèle sémantique des systèmes réactifs $\mathcal{T} = (Q, I, \delta)$
  - Q ensemble d'états
  - $I \subseteq Q$ états initiaux
  - $\delta \subseteq Q \times Q$ relation (totale) de transition

Systèmes de transition

- Modèle sémantique des systèmes réactifs $\mathcal{T} = (Q, I, \delta)$
  - Q ensemble d'états
  - $I \subseteq Q$ états initiaux
  - $\delta \subseteq Q \times Q$ relation (totale) de transition
- en pratique : $\mathcal{T}$ donné implicitement
  - Promela : points de contrôle, variables, canaux de messages
  - algèbres de processus : état = processus, transition = réduction
  - réseaux de Petri : état = marquage du réseau

Systèmes de transition

- Modèle sémantique des systèmes réactifs $\mathcal{T} = (Q, I, \delta)$
  - Q ensemble d'états
  - $I \subseteq Q$ états initiaux
  - $\delta \subseteq Q \times Q$ relation (totale) de transition

- en pratique : $\mathcal{T}$ donné implicitement
  - Promela : points de contrôle, variables, canaux de messages
  - algèbres de processus : état = processus, transition = réduction
  - réseaux de Petri : état = marquage du réseau

taille de $\mathcal{T}$ en général exponentielle en la taille de la description

Systèmes de transition

- Modèle sémantique des systèmes réactifs $\mathcal{T} = (Q, I, \delta)$
  - Q ensemble d'états
  - $I \subseteq Q$ états initiaux
  - $\delta \subseteq Q \times Q$ relation (totale) de transition

- en pratique : $\mathcal{T}$ donné implicitement
  - Promela : points de contrôle, variables, canaux de messages
  - algèbres de processus : état = processus, transition = réduction
  - réseaux de Petri : état = marquage du réseau

  taille de $\mathcal{T}$ en général exponentielle en la taille de la description

- extensions et variations
  - conditions d'équité
  - systèmes temporisés et hybrides

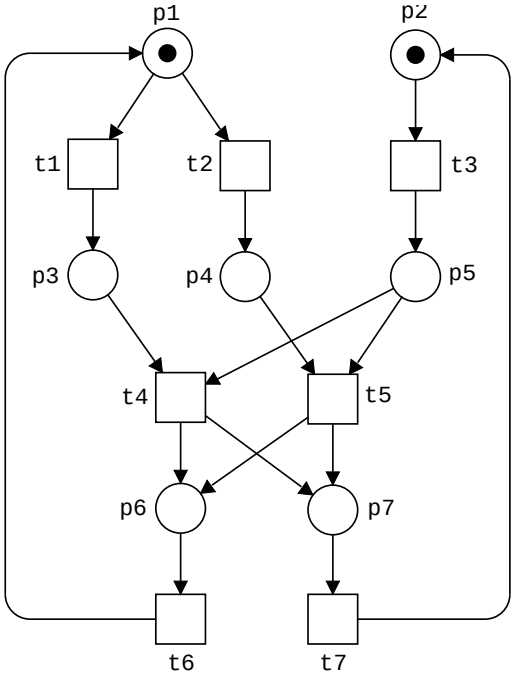
Structures de Kripke

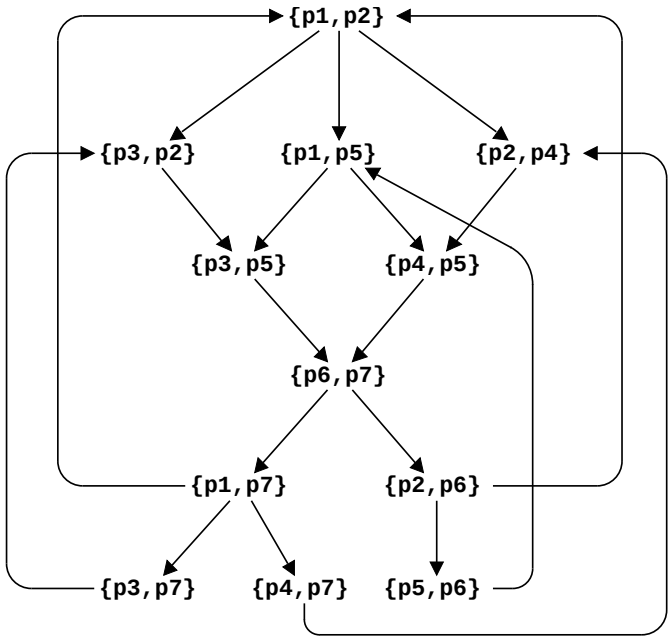
- système de transition + propositions $\mathcal{K} = (Q, I, \delta, \mathcal{V}, \lambda)$
  - $\mathcal{V}$ ensemble de propositions atomiques
  - $\lambda : Q \to 2^{\mathcal{V}}$ propositions vraies dans un état

Structures de Kripke

- système de transition + propositions $\mathcal{K} = (Q, I, \delta, \mathcal{V}, \lambda)$
  - $\mathcal{V}$ ensemble de propositions atomiques
  - $\lambda : Q \to 2^{\mathcal{V}}$ propositions vraies dans un état

- exemple : réseau de Petri et structure de Kripke





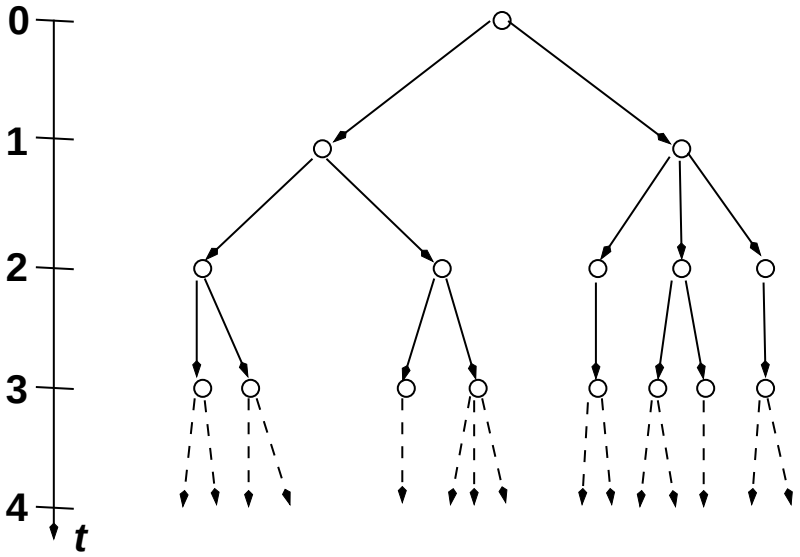
Exécutions

- exécution $\rho = q_0 q_1 \ldots$ d'un système de transitions $\mathcal{T} = (Q, I, \delta)$
  - $q_0 \in I$ état initial
  - $(q_i, q_{i+1}) \in \delta$ succession d'états

Exécutions

- exécution $\rho = q_0 q_1 \ldots$ d'un système de transitions $\mathcal{T} = (Q, I, \delta)$
  - $q_0 \in I$ état initial
  - $(q_i, q_{i+1}) \in \delta$ succession d'états

- arbre d'exécutions représente toutes les exécutions de $\mathcal{T}$



nœuds	états de $\mathcal{T}$
arètes	transitions
chemins	exécutions
arborescence	non-déterminisme (e.g., interleaving)

Outline

Invariants

Definition

Un prédicat Inv est un invariant de $\mathcal{K}$ si pour toute exécution $\rho = q_0 q_1 \ldots$ de $\mathcal{K}$ et pour tout $i \in \mathbb{N}$, on a $q_i \models Inv$.

Invariants

Definition

Un prédicat Inv est un invariant de $\mathcal{K}$ si pour toute exécution $\rho = q_0 q_1 \ldots$ de $\mathcal{K}$ et pour tout $i \in \mathbb{N}$, on a $q_i \models Inv$.

- propriétés de correction fondamentales
  - accès exclusif à des ressources
  - non-corruption des messages envoyés
  - préservation de l'ordre des messages

Invariants

Definition

Un prédicat Inv est un invariant de $\mathcal{K}$ si pour toute exécution $\rho = q_0 q_1 \ldots$ de $\mathcal{K}$ et pour tout $i \in \mathbb{N}$, on a $q_i \models Inv$.

- propriétés de correction fondamentales
  - accès exclusif à des ressources
  - non-corruption des messages envoyés
  - préservation de l'ordre des messages
- cas particulier : invariant inductif
  - vérifié par les états initiaux et préservé par toute transition

    $Init \Rightarrow Inv \qquad \{Inv\}\ \delta\ \{Inv\}$
  - à la base de l'approche déductive de vérification

Vérification algorithmique d'invariants (1)

- Idée
  - énumérer tous les états accessibles de $\mathcal{K}$
  - vérifier que tous ces états vérifient Inv

Vérification algorithmique d'invariants (1)

- Idée
  - énumérer tous les états accessibles de $\mathcal{K}$
  - vérifier que tous ces états vérifient Inv
- Algorithme (pseudo-code)

```
boolean verify_inv(KripkeStructure ks, Predicate inv) {
  Set visited = new Set();
  Set todo = ks.getInitials();
  while (!todo.isEmpty()) {
    State s = todo.someElement();
    todo.remove(s); visited.add(s);
    if (!s.satisfies(inv))
      return false;
    foreach (s' in ks.successors(s)) {
      if (!todo.contains(s') && !visited.contains(s'))
        todo.add(s')
  } }
  return true;
}
```

Vérification algorithmique d'invariants (2)

- recherche en profondeur (pile `todo`)
  - pile contient contre-exemple
  - contre-exemples potentiellement longs

Vérification algorithmique d'invariants (2)

- recherche en profondeur (pile `todo`)
  - pile contient contre-exemple
  - contre-exemples potentiellement longs
- recherche en largeur (file `todo`)
  - garantit contre-exemples de taille minimale
  - nécessite gestion supplémentaire de prédecesseurs

Vérification algorithmique d'invariants (2)

- recherche en profondeur (pile `todo`)
  - pile contient contre-exemple
  - contre-exemples potentiellement longs
- recherche en largeur (file `todo`)
  - garantit contre-exemples de taille minimale
  - nécessite gestion supplémentaire de prédecesseurs
- Problème : grands modèles ($> 10^6$–10^7 états accessibles)
  - utiliser le disque
  - mémoriser *signatures* au lieu d'états dans `visited`
    $\rightsquigarrow$ recherche incomplète en cas de collisions
  - réduire le nombre d'états ou de transitions à explorer

Vérification algorithmique d'invariants (2)

- recherche en profondeur (pile `todo`)
  - pile contient contre-exemple
  - contre-exemples potentiellement longs
- recherche en largeur (file `todo`)
  - garantit contre-exemples de taille minimale
  - nécessite gestion supplémentaire de prédecesseurs
- Problème : grands modèles ($> 10^6$–10^7 états accessibles)
  - utiliser le disque
  - mémoriser *signatures* au lieu d'états dans `visited`
    $\rightsquigarrow$ recherche incomplète en cas de collisions
  - réduire le nombre d'états ou de transitions à explorer
- Outil : Murφ `http://sprout.stanford.edu/dill/murphi.html`

Outline

Logique temporelle linéaire (LTL)

- formuler des propriétés d'exécutions
  - au-delà des invariants

Logique temporelle linéaire (LTL)

- formuler des propriétés d'exécutions
  - au-delà des invariants
- formules évaluées sur des séquences d'états de $\mathcal{K}$

type	formule		$\rho \models \varphi$ ssi ...
atomique	$p \in \mathcal{V}$		$p \in \lambda(\rho_0)$
booléenne	$\neg\varphi$		$\rho \not\models \varphi$
	$\varphi \vee \psi$		$\rho \models \varphi$ ou $\rho \models \psi$
temporelle	$\mathbf{X}\,\varphi$	$\bigcirc\varphi$	$\rho\|_1 \models \varphi$
	$\mathbf{G}\,\varphi$	$\Box\varphi$	$\rho\|_k \models \varphi$ pour tout $k \in \mathbb{N}$
	$\mathbf{F}\,\varphi$	$\Diamond\varphi$	$\rho\|_k \models \varphi$ pour un $k \in \mathbb{N}$
	$\varphi\ \mathbf{U}\ \psi$	φ **until** ψ	il existe $k \in \mathbb{N}$ tel que $\rho\|_k \models \psi$ et $\rho\|_i \models \varphi$ pour tout $i < k$

Logique temporelle linéaire (LTL)

- formuler des propriétés d'exécutions
  - au-delà des invariants
- formules évaluées sur des séquences d'états de $\mathcal{K}$

type	formule		$\rho \models \varphi$ ssi ...
atomique	$p \in \mathcal{V}$		$p \in \lambda(\rho_0)$
booléenne	$\neg\varphi$		$\rho \not\models \varphi$
	$\varphi \vee \psi$		$\rho \models \varphi$ ou $\rho \models \psi$
temporelle	$\mathbf{X}\,\varphi$	$\bigcirc\varphi$	$\rho\|_1 \models \varphi$
	$\mathbf{G}\,\varphi$	$\Box\varphi$	$\rho\|_k \models \varphi$ pour tout $k \in \mathbb{N}$
	$\mathbf{F}\,\varphi$	$\Diamond\varphi$	$\rho\|_k \models \varphi$ pour un $k \in \mathbb{N}$
	$\varphi\ \mathbf{U}\ \psi$	φ **until** ψ	il existe $k \in \mathbb{N}$ tel que $\rho\|_k \models \psi$ et $\rho\|_i \models \varphi$ pour tout $i < k$

$\mathcal{K} \models \varphi$ ssi $\rho \models \varphi$ pour toute exécution ρ de $\mathcal{K}$

LTL : exemples de propriétés

- Invariants $\quad \mathbf{G}\, p$

  $\mathbf{G}\, \neg(crit_1 \wedge crit_2)$ exclusion mutuelle

  $\mathbf{G}(gd_1 \vee \ldots \vee gd_n)$ absence de blocage

LTL : exemples de propriétés

- Invariants $\mathbf{G}\, p$

  $\mathbf{G}\, \neg(crit_1 \wedge crit_2)$ exclusion mutuelle

  $\mathbf{G}(gd_1 \vee \ldots \vee gd_n)$ absence de blocage

- réponse, récurrence $\mathbf{G}(p \Rightarrow \mathbf{F}\, q)$

  $\mathbf{G}(try_1 \Rightarrow \mathbf{F}\, crit_1)$ accès garanti à la section critique

  $\mathbf{G}\, \mathbf{F}\, \neg crit_1$ séjours finis en section critique

LTL : exemples de propriétés

- Invariants $\mathbf{G}\, p$

  $\mathbf{G}\, \neg(crit_1 \wedge crit_2)$ exclusion mutuelle

  $\mathbf{G}(gd_1 \vee \ldots \vee gd_n)$ absence de blocage

- réponse, récurrence $\mathbf{G}(p \Rightarrow \mathbf{F}\, q)$

  $\mathbf{G}(try_1 \Rightarrow \mathbf{F}\, crit_1)$ accès garanti à la section critique

  $\mathbf{G}\, \mathbf{F}\, \neg crit_1$ séjours finis en section critique

- réactivité $\mathbf{G}\, \mathbf{F}\, p \Rightarrow \mathbf{G}\, \mathbf{F}\, q$

  $\mathbf{G}\, \mathbf{F}(try_1 \wedge \neg crit_2) \Rightarrow \mathbf{G}\, \mathbf{F}\, crit_1$ équité forte

LTL : exemples de propriétés

- Invariants $\mathbf{G}\, p$

  $\mathbf{G}\, \neg(crit_1 \wedge crit_2)$ exclusion mutuelle

  $\mathbf{G}(gd_1 \vee \ldots \vee gd_n)$ absence de blocage

- réponse, récurrence $\mathbf{G}(p \Rightarrow \mathbf{F}\, q)$

  $\mathbf{G}(try_1 \Rightarrow \mathbf{F}\, crit_1)$ accès garanti à la section critique

  $\mathbf{G}\, \mathbf{F}\, \neg crit_1$ séjours finis en section critique

- réactivité $\mathbf{G}\, \mathbf{F}\, p \Rightarrow \mathbf{G}\, \mathbf{F}\, q$

  $\mathbf{G}\, \mathbf{F}(try_1 \wedge \neg crit_2) \Rightarrow \mathbf{G}\, \mathbf{F}\, crit_1$ équité forte

- précédence $p_1\ \mathbf{U} \ldots \mathbf{U}\ p_n$

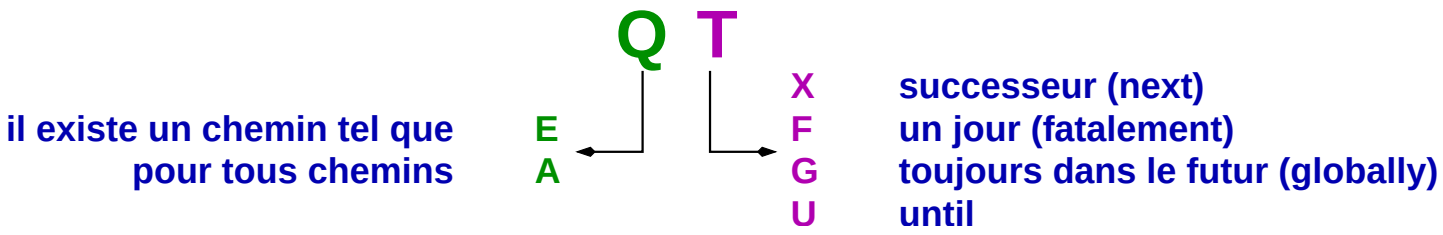
  $\mathbf{G}(try_1 \wedge try_2 \Rightarrow \neg crit_2\ \mathbf{U}\ crit_2\ \mathbf{U}\ \neg crit_2\ \mathbf{U}\ crit_1)$ 1-bounded overtaking

Logiques temporelles arborescentes

- Limites d'expressivité de LTL
  - formules expriment des propriétés de chemins
  - l'existence d'un chemin ne peut être formulé en LTL

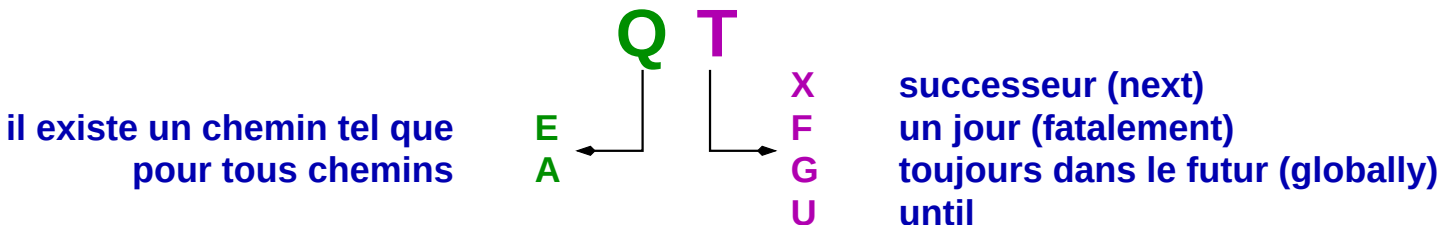
Logiques temporelles arborescentes

- Limites d'expressivité de LTL
  - formules expriment des propriétés de chemins
  - l'existence d'un chemin ne peut être formulé en LTL
- Logiques arborescentes : quantification sur les chemins
- Exemple : CTL (computation tree logic)



Logiques temporelles arborescentes

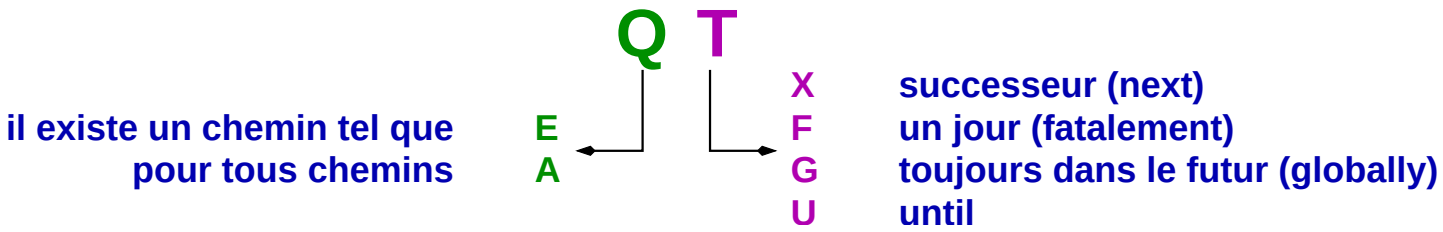
- Limites d'expressivité de LTL
  - formules expriment des propriétés de chemins
  - l'existence d'un chemin ne peut être formulé en LTL
- Logiques arborescentes : quantification sur les chemins
- Exemple : CTL (computation tree logic)



  - formules évaluées sur les sous-arbres $\mathcal{K}, q \models \varphi$
  - validité système $\mathcal{K} \models \varphi$ ssi $\mathcal{K}, q \models \varphi$ pour tout $q \in I$

Logiques temporelles arborescentes

- Limites d'expressivité de LTL
  - formules expriment des propriétés de chemins
  - l'existence d'un chemin ne peut être formulé en LTL
- Logiques arborescentes : quantification sur les chemins
- Exemple : CTL (computation tree logic)

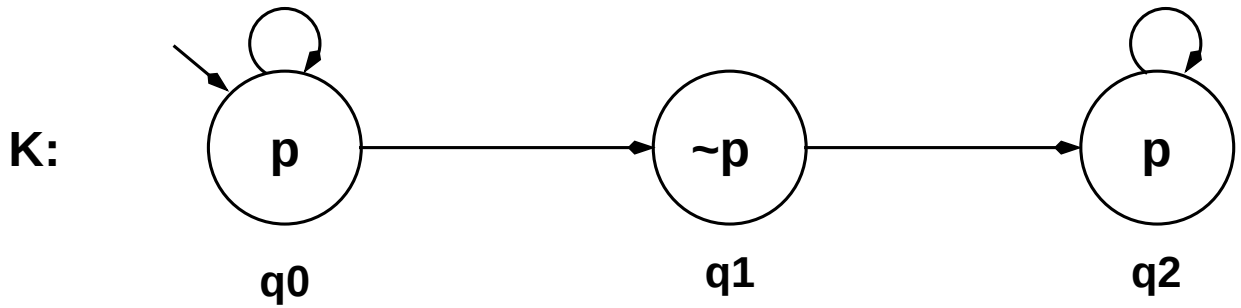


  - formules évaluées sur les sous-arbres $\mathcal{K}, q \models \varphi$
  - validité système $\mathcal{K} \models \varphi$ ssi $\mathcal{K}, q \models \varphi$ pour tout $q \in I$
- propriétés existentielles

AG EF *init* possibilité de réinitialisation

Expressivité : LTL vs. CTL

- Pouvoir expressif incomparable de LTL et CTL
  - impossibilité d'exprimer les propriétés existentielles en LTL
  - CTL ne sait pas exprimer $\mathbf{F}\,\mathbf{G}\,p$

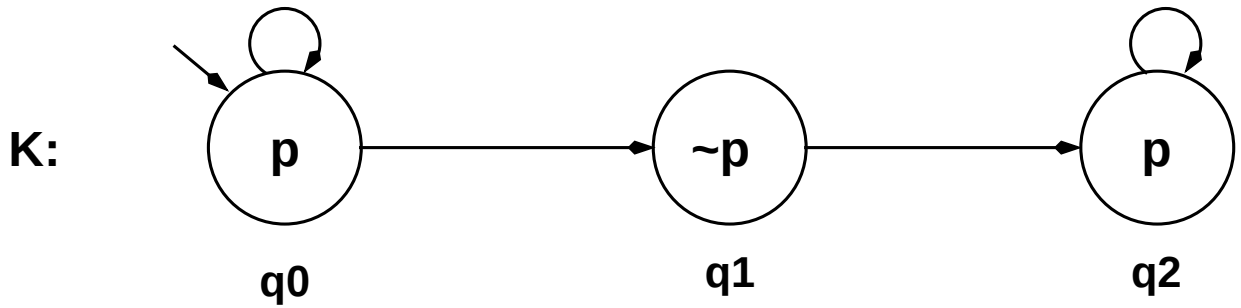


$\mathcal{K} \models \mathbf{F}\,\mathbf{G}\,p$ $\qquad$ $\mathcal{K} \not\models \mathbf{AF}\,\mathbf{AG}\,p$

  - problème similaire : équité forte

Expressivité : LTL vs. CTL

- Pouvoir expressif incomparable de LTL et CTL
  - impossibilité d'exprimer les propriétés existentielles en LTL
  - CTL ne sait pas exprimer $\mathbf{F}\,\mathbf{G}\,p$



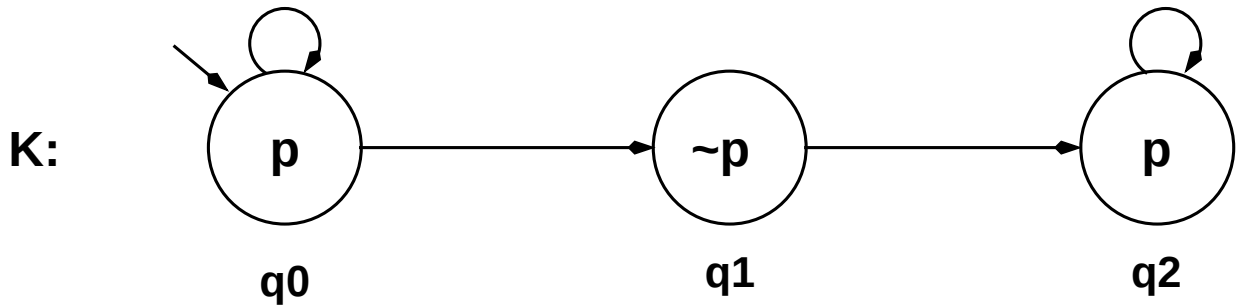
$\mathcal{K} \models \mathbf{F}\,\mathbf{G}\,p$ $\qquad$ $\mathcal{K} \not\models \mathbf{AF}\,\mathbf{AG}\,p$

  - problème similaire : équité forte

- Choix de la logique
  - en fonction des besoins du problème …
  - … et des outils qui conviennent

Expressivité : LTL vs. CTL

- Pouvoir expressif incomparable de LTL et CTL
  - impossibilité d'exprimer les propriétés existentielles en LTL
  - CTL ne sait pas exprimer $\mathbf{F}\,\mathbf{G}\,p$



$\mathcal{K} \models \mathbf{F}\,\mathbf{G}\,p$ $\qquad$ $\mathcal{K} \not\models \mathbf{AF}\,\mathbf{AG}\,p$

  - problème similaire : équité forte
- Choix de la logique
  - en fonction des besoins du problème …
  - … et des outils qui conviennent
- Logiques plus expressives : CTL*, μ-calcul

Outline

Algorithmes de vérification

- Extensions de l'algorithme pour les invariants
  - gérer les propriétés temporelles
  - réduction aux algorithmes de graphes

Algorithmes de vérification

- Extensions de l'algorithme pour les invariants
  - gérer les propriétés temporelles
  - réduction aux algorithmes de graphes

- Deux approches principales
  - récursion sur la structure de $\mathcal{K}$ — algorithme local
  - récursion sur les formules — algorithme global

Algorithmes de vérification

- Extensions de l'algorithme pour les invariants
  - gérer les propriétés temporelles
  - réduction aux algorithmes de graphes
- Deux approches principales
  - récursion sur la structure de $\mathcal{K}$ — algorithme local
  - récursion sur les formules — algorithme global
- Problème pour mise en œuvre
  - traiter les grands modèles : explosion combinatoire
  - techniques adaptées aux classes de systèmes

Algorithme local pour LTL

- Représenter les formules temporelles par des automates
  - $\mathcal{L}(\varphi)$ ensemble de modèles de φ
  - peut être décrit comme langage $\mathcal{L}(\mathcal{A}_\varphi)$ d'un automate $\mathcal{A}_\varphi$

Algorithme local pour LTL

- Représenter les formules temporelles par des automates
  - $\mathcal{L}(\varphi)$ ensemble de modèles de φ
  - peut être décrit comme langage $\mathcal{L}(\mathcal{A}_\varphi)$ d'un automate $\mathcal{A}_\varphi$

$$\mathcal{K} \models \varphi$$
ssi
$$\mathcal{L}(\mathcal{K}) \subseteq \mathcal{L}(\varphi)$$
ssi
$$\mathcal{L}(\mathcal{K}) \cap \mathcal{L}(\neg\varphi) = \emptyset$$
ssi
$$\mathcal{L}(\mathcal{K} \times \mathcal{A}_{\neg\varphi}) = \emptyset$$

Algorithme local pour LTL

- Représenter les formules temporelles par des automates
  - $\mathcal{L}(\varphi)$ ensemble de modèles de φ
  - peut être décrit comme langage $\mathcal{L}(\mathcal{A}_\varphi)$ d'un automate $\mathcal{A}_\varphi$

$$\mathcal{K} \models \varphi$$
ssi
$$\mathcal{L}(\mathcal{K}) \subseteq \mathcal{L}(\varphi)$$
ssi
$$\mathcal{L}(\mathcal{K}) \cap \mathcal{L}(\neg\varphi) = \emptyset$$
ssi
$$\mathcal{L}(\mathcal{K} \times \mathcal{A}_{\neg\varphi}) = \emptyset$$

- Deux sous-problèmes
  - traduction de formules $\varphi \rightsquigarrow \mathcal{A}_\varphi$
  - décision du vide $\mathcal{L}(\mathcal{A}) \stackrel{?}{=} \emptyset$

Automates de Büchi

- Automates finis sur mots infinis $\mathcal{B} = (Q, I, \delta, F)$

Q	ensemble fini d'états	structure d'un automate fini ordinaire
$I \subseteq Q$	états initiaux	
$\delta \subseteq Q \times 2^{\mathcal{V}} \times Q$	relation de transition	
$F \subseteq Q$	états d'acceptation	

Automates de Büchi

- Automates finis sur mots infinis $\mathcal{B} = (Q, I, \delta, F)$

Q	ensemble fini d'états	structure d'un automate fini ordinaire
$I \subseteq Q$	états initiaux	
$\delta \subseteq Q \times 2^{\mathcal{V}} \times Q$	relation de transition	
$F \subseteq Q$	états d'acceptation	

- Exécution d'un automate sur un mot $q_0 \xrightarrow{L_0} q_1 \xrightarrow{L_1} q_2 \ldots$
  - initialisation $q_0 \in I$
  - succession $(q_i, L_i, q_{i+1}) \in \delta$ pour tout $i \in \mathbb{N}$
  - acceptation $q_i \in F$ pour un nombre infini de $i \in \mathbb{N}$

Automates de Büchi

- Automates finis sur mots infinis $\mathcal{B} = (Q, I, \delta, F)$

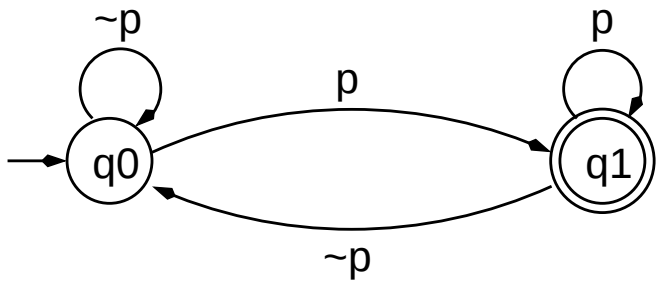
Q	ensemble fini d'états	structure d'un automate fini ordinaire
$I \subseteq Q$	états initiaux	
$\delta \subseteq Q \times 2^{\mathcal{V}} \times Q$	relation de transition	
$F \subseteq Q$	états d'acceptation	

- Exécution d'un automate sur un mot $q_0 \xrightarrow{L_0} q_1 \xrightarrow{L_1} q_2 \ldots$
  - initialisation $q_0 \in I$
  - succession $(q_i, L_i, q_{i+1}) \in \delta$ pour tout $i \in \mathbb{N}$
  - acceptation $q_i \in F$ pour un nombre infini de $i \in \mathbb{N}$

- Langages
  - $\mathcal{L}(\mathcal{B})$: langage de l'automate $\mathcal{B}$
    ensemble de mots pour lesquels il existe une exécution acceptée
  - langages ω-réguliers
    classe de langages définissables par les automates de Büchi

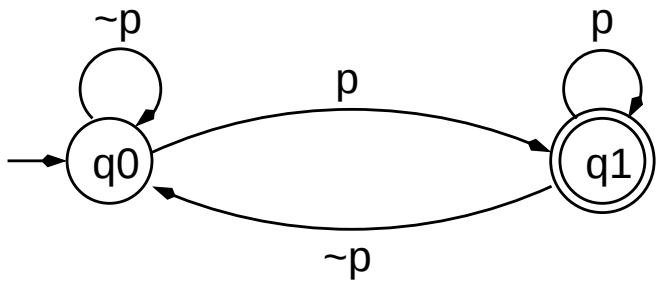
Automates de Büchi : exemples

- infiniment souvent p

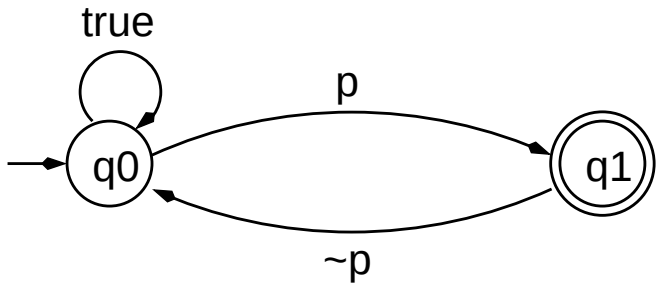


Automates de Büchi : exemples

- infiniment souvent p

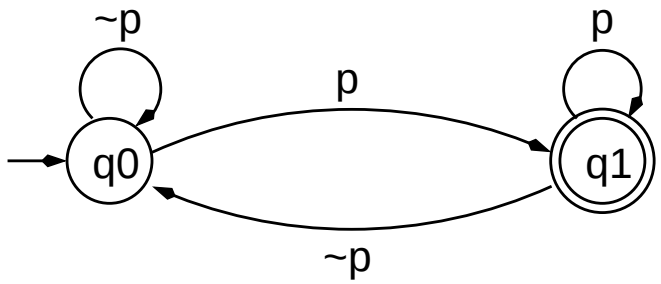


- infiniment souvent "p ; $\neg p$"

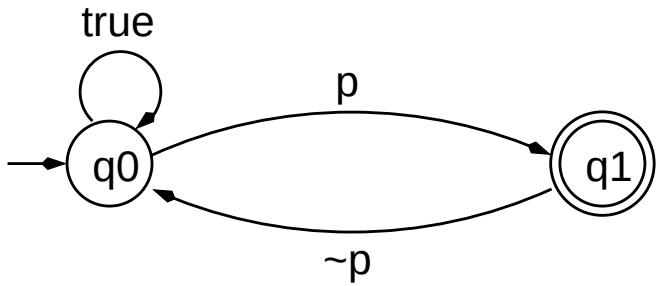


Automates de Büchi : exemples

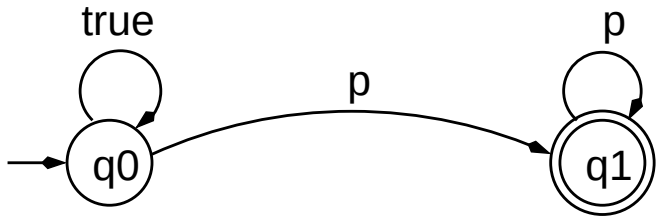
- infiniment souvent p



- infiniment souvent "p ; $\neg p$"

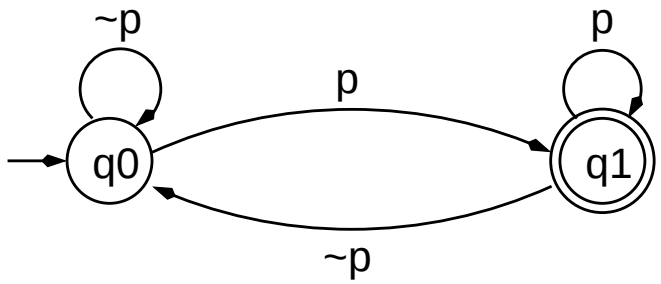


- presque toujours p

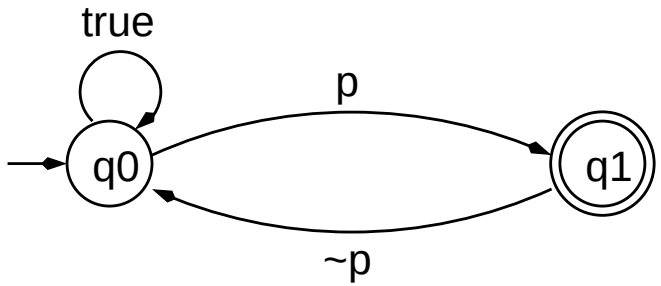


Automates de Büchi : exemples

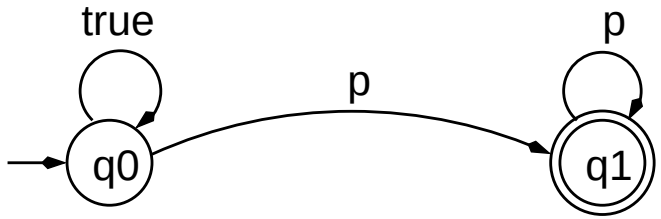
- infiniment souvent p



- infiniment souvent "p ; $\neg p$"



- presque toujours p



ne peut être défini par un automate déterministe

Automates de Büchi : model checking

- Traduction $\varphi \rightsquigarrow \mathcal{B}_\varphi$
  - états correspondent à des ensembles de sous-formules de φ
  - transitions vérifient les formules atomiques
  - états d'acceptation définis à partir des formules $p\ \mathbf{U}\ q$
  - algorithme de complexité exponentiel

Automates de Büchi : model checking

- Traduction $\varphi \rightsquigarrow \mathcal{B}_\varphi$
  - états correspondent à des ensembles de sous-formules de φ
  - transitions vérifient les formules atomiques
  - états d'acceptation définis à partir des formules $p\ \mathbf{U}\ q$
  - algorithme de complexité exponentiel

- Décision du vide
  - $\mathcal{L}(\mathcal{B}) \neq \emptyset$ ssi il existe $q_0 \in I, q \in F$ avec $q_0 \Longrightarrow^* q \Longrightarrow^+ q$
  - complexité linéaire en la taille de $\mathcal{B}$

Automates de Büchi : model checking

- Traduction $\varphi \rightsquigarrow \mathcal{B}_\varphi$
  - états correspondent à des ensembles de sous-formules de φ
  - transitions vérifient les formules atomiques
  - états d'acceptation définis à partir des formules $p\ \mathbf{U}\ q$
  - algorithme de complexité exponentiel

- Décision du vide
  - $\mathcal{L}(\mathcal{B}) \neq \emptyset$ ssi il existe $q_0 \in I, q \in F$ avec $q_0 \Longrightarrow^* q \Longrightarrow^+ q$
  - complexité linéaire en la taille de $\mathcal{B}$

- Complexité de la vérification $O(|\mathcal{K}| \cdot |\mathcal{B}_{\neg\varphi}|) = O(|\mathcal{K}| \cdot 2^{|\varphi|})$

Model checking pour CTL

- Algorithme global
  - marquer tout état par les formules qu'il vérifie

Model checking pour CTL

- Algorithme global
  - marquer tout état par les formules qu'il vérifie

- Rappel validité système

$$\begin{array}{lll} \mathcal{K} \models \varphi & \text{ssi} & \mathcal{K}, q \models \varphi \text{ pour tout } q \in I \\ & \text{ssi} & I \subseteq [\![\varphi]\!]_{\mathcal{K}} \\ \text{où} & & [\![\varphi]\!]_{\mathcal{K}} = \{q \in Q : \mathcal{K}, q \models \varphi\} \end{array}$$

Model checking pour CTL

- Algorithme global
  - marquer tout état par les formules qu'il vérifie

- Rappel validité système

$$\begin{array}{lll} \mathcal{K} \models \varphi & \text{ssi} & \mathcal{K}, q \models \varphi \text{ pour tout } q \in I \\ & \text{ssi} & I \subseteq [\![\varphi]\!]_{\mathcal{K}} \\ \text{où} & & [\![\varphi]\!]_{\mathcal{K}} = \{q \in Q : \mathcal{K}, q \models \varphi\} \end{array}$$

- Eléments algorithmiques
  - algorithme pour calculer $[\![\varphi]\!]_{\mathcal{K}}$ (par récursion sur φ)
  - structures de données efficaces pour manipuler les ensembles

Calcul de $[\![\varphi]\!]_{\mathcal{K}}$

- Observation : opérateurs de CTL de base **EX, EG, EU**

$$\mathbf{AX}\,\varphi \equiv \neg\,\mathbf{EX}\,\neg\varphi \qquad \mathbf{EF}\,\varphi \equiv \mathbf{true}\;\mathbf{EU}\;\varphi$$
$$\mathbf{AG}\,\varphi \equiv \neg\,\mathbf{EF}\,\neg\varphi \qquad \mathbf{AF}\,\varphi \equiv \neg\,\mathbf{EG}\,\neg\varphi$$

Calcul de $[\![\varphi]\!]_{\mathcal{K}}$

- Observation : opérateurs de CTL de base **EX, EG, EU**

$$\begin{array}{lcl@{\qquad\qquad}lcl} \mathbf{AX}\,\varphi & \equiv & \neg\,\mathbf{EX}\,\neg\varphi & \mathbf{EF}\,\varphi & \equiv & \mathbf{true}\ \mathbf{EU}\ \varphi \\ \mathbf{AG}\,\varphi & \equiv & \neg\,\mathbf{EF}\,\neg\varphi & \mathbf{AF}\,\varphi & \equiv & \neg\,\mathbf{EG}\,\neg\varphi \end{array}$$

- Calcul : les cas simples

$$\begin{array}{lcl} [\![p]\!]_{\mathcal{K}} & = & \{q \in Q : p \in \lambda(q)\} \\ [\![\neg\varphi]\!]_{\mathcal{K}} & = & Q \setminus [\![\varphi]\!]_{\mathcal{K}} \\ [\![\varphi \vee \psi]\!]_{\mathcal{K}} & = & [\![\varphi]\!]_{\mathcal{K}} \cup [\![\psi]\!]_{\mathcal{K}} \\ [\![\mathbf{EX}\,\varphi]\!]_{\mathcal{K}} & = & \delta^{-1}([\![\varphi]\!]_{\mathcal{K}}) \\ & =_{\text{def}} & \{q \in Q : q' \in [\![\varphi]\!]_{\mathcal{K}} \text{ pour un } q' \text{ tel que } (q, q') \in \delta\} \end{array}$$

Calcul de $[\![\varphi]\!]_{\mathcal{K}}$

- Observation : opérateurs de CTL de base **EX, EG, EU**

$$\begin{array}{lcl@{\qquad\qquad}lcl} \mathbf{AX}\,\varphi & \equiv & \neg\,\mathbf{EX}\,\neg\varphi & \mathbf{EF}\,\varphi & \equiv & \mathbf{true}\ \mathbf{EU}\ \varphi \\ \mathbf{AG}\,\varphi & \equiv & \neg\,\mathbf{EF}\,\neg\varphi & \mathbf{AF}\,\varphi & \equiv & \neg\,\mathbf{EG}\,\neg\varphi \end{array}$$

- Calcul : les cas simples

$$\begin{array}{lcl} [\![p]\!]_{\mathcal{K}} & = & \{q \in Q : p \in \lambda(q)\} \\ [\![\neg\varphi]\!]_{\mathcal{K}} & = & Q \setminus [\![\varphi]\!]_{\mathcal{K}} \\ [\![\varphi \vee \psi]\!]_{\mathcal{K}} & = & [\![\varphi]\!]_{\mathcal{K}} \cup [\![\psi]\!]_{\mathcal{K}} \\ [\![\mathbf{EX}\,\varphi]\!]_{\mathcal{K}} & = & \delta^{-1}([\![\varphi]\!]_{\mathcal{K}}) \\ & =_{\text{def}} & \{q \in Q : q' \in [\![\varphi]\!]_{\mathcal{K}} \text{ pour un } q' \text{ tel que } (q, q') \in \delta\} \end{array}$$

- Reste à définir $[\![\mathbf{EG}\,\varphi]\!]_{\mathcal{K}}$, $[\![\varphi\ \mathbf{EU}\ \psi]\!]_{\mathcal{K}}$

Sémantique : points fixes

- Lois «récursives»

$$\begin{aligned} \mathbf{EG}\,\varphi &\equiv \varphi \wedge \mathbf{EX}\,\mathbf{EG}\,\varphi \\ \varphi\ \mathbf{EU}\ \psi &\equiv \psi \vee (\varphi \wedge \mathbf{EX}(\varphi\ \mathbf{EU}\ \psi)) \end{aligned}$$

Sémantique : points fixes

- Lois «récursives»

$$\begin{aligned} \mathbf{EG}\,\varphi &\equiv \varphi \wedge \mathbf{EX}\,\mathbf{EG}\,\varphi \\ \varphi\ \mathbf{EU}\ \psi &\equiv \psi \vee (\varphi \wedge \mathbf{EX}(\varphi\ \mathbf{EU}\ \psi)) \end{aligned}$$

- Sémantique des opérateurs comme points fixes
  - $[\![\mathbf{EG}\,\varphi]\!]_{\mathcal{K}}$ plus grande solution de $X \equiv \varphi \wedge \mathbf{EX}\,X$ dans $(2^{Q}, \subseteq)$

Sémantique : points fixes

- Lois «récursives»

$$\begin{aligned} \mathbf{EG}\,\varphi &\equiv \varphi \wedge \mathbf{EX}\,\mathbf{EG}\,\varphi \\ \varphi\ \mathbf{EU}\ \psi &\equiv \psi \vee (\varphi \wedge \mathbf{EX}(\varphi\ \mathbf{EU}\ \psi)) \end{aligned}$$

- Sémantique des opérateurs comme points fixes
  - $[\![\mathbf{EG}\,\varphi]\!]_{\mathcal{K}}$ plus grande solution de $X \equiv \varphi \wedge \mathbf{EX}\,X$ dans $(2^Q, \subseteq)$
  - $[\![\mathbf{false}]\!]_{\mathcal{K}}$ en est la plus petite solution

Sémantique : points fixes

- Lois «récursives»

$$\begin{aligned} \mathbf{EG}\,\varphi &\equiv \varphi \wedge \mathbf{EX}\,\mathbf{EG}\,\varphi \\ \varphi\ \mathbf{EU}\ \psi &\equiv \psi \vee (\varphi \wedge \mathbf{EX}(\varphi\ \mathbf{EU}\ \psi)) \end{aligned}$$

- Sémantique des opérateurs comme points fixes
  - $[\![\mathbf{EG}\,\varphi]\!]_{\mathcal{K}}$ plus grande solution de $X \equiv \varphi \wedge \mathbf{EX}\,X$ dans $(2^{Q}, \subseteq)$
  - $[\![\mathbf{false}]\!]_{\mathcal{K}}$ en est la plus petite solution
  - $[\![\varphi\ \mathbf{EU}\ \psi]\!]_{\mathcal{K}}$ plus petite solution de $X \equiv \psi \vee (\varphi \wedge \mathbf{EX}\,X)$

Sémantique : points fixes

- Lois «récursives»

$$\begin{aligned} \mathbf{EG}\,\varphi &\equiv \varphi \wedge \mathbf{EX}\,\mathbf{EG}\,\varphi \\ \varphi\ \mathbf{EU}\ \psi &\equiv \psi \vee (\varphi \wedge \mathbf{EX}(\varphi\ \mathbf{EU}\ \psi)) \end{aligned}$$

- Sémantique des opérateurs comme points fixes
  - $[\![\mathbf{EG}\,\varphi]\!]_{\mathcal{K}}$ plus grande solution de $X \equiv \varphi \wedge \mathbf{EX}\,X$ dans $(2^Q, \subseteq)$
  - $[\![\mathbf{false}]\!]_{\mathcal{K}}$ en est la plus petite solution
  - $[\![\varphi\ \mathbf{EU}\ \psi]\!]_{\mathcal{K}}$ plus petite solution de $X \equiv \psi \vee (\varphi \wedge \mathbf{EX}\,X)$

- Calcul des points fixes
  - itérer l'équation de recurrence à partir de Q ou de $\emptyset$

Sémantique : points fixes

- Lois «récursives»

$$\begin{aligned} \mathbf{EG}\,\varphi &\equiv \varphi \wedge \mathbf{EX}\,\mathbf{EG}\,\varphi \\ \varphi\ \mathbf{EU}\ \psi &\equiv \psi \vee (\varphi \wedge \mathbf{EX}(\varphi\ \mathbf{EU}\ \psi)) \end{aligned}$$

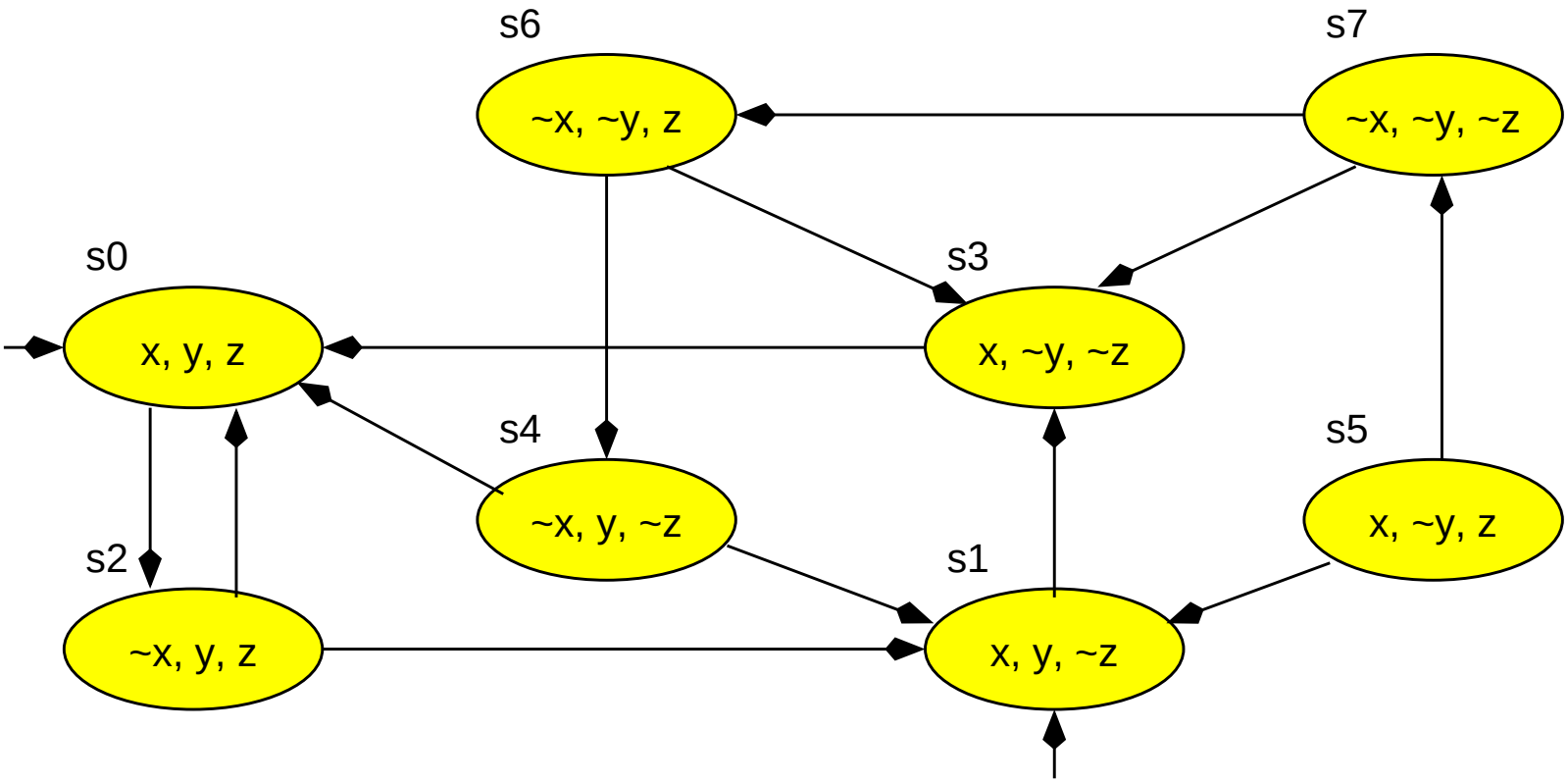
- Sémantique des opérateurs comme points fixes
  - $[\![\mathbf{EG}\,\varphi]\!]_{\mathcal{K}}$ plus grande solution de $X \equiv \varphi \wedge \mathbf{EX}\,X$ dans $(2^Q, \subseteq)$
  - $[\![\mathbf{false}]\!]_{\mathcal{K}}$ en est la plus petite solution
  - $[\![\varphi\ \mathbf{EU}\ \psi]\!]_{\mathcal{K}}$ plus petite solution de $X \equiv \psi \vee (\varphi \wedge \mathbf{EX}\,X)$

- Calcul des points fixes
  - itérer l'équation de recurrence à partir de Q ou de $\emptyset$

- Extension aux hypothèses d'équité

Calcul du plus grand point fixe (exemple)

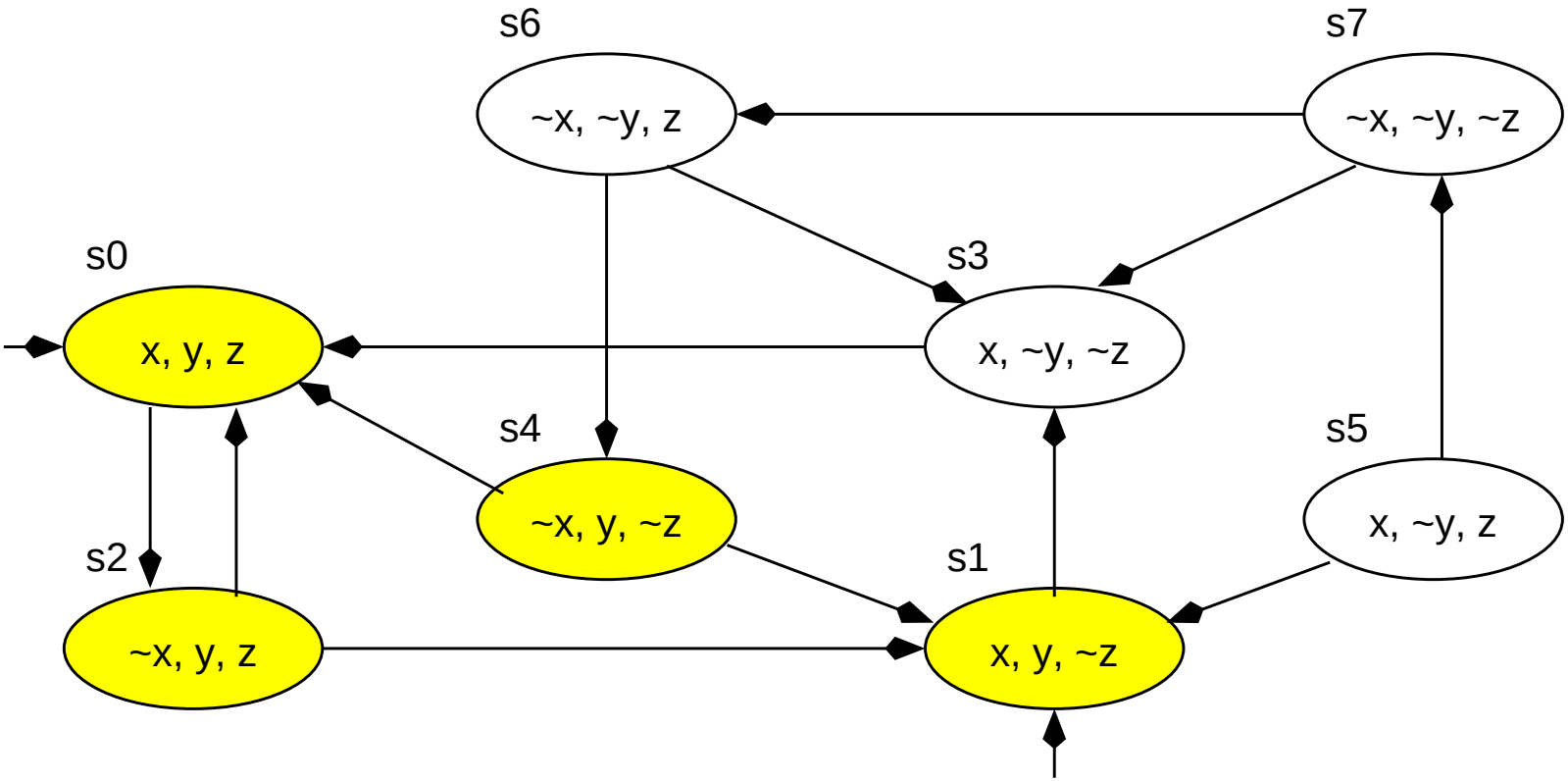
Calcul de $[\![\mathbf{EG}\, y]\!]$



$$\pi^0(Q) \;=\; Q$$

Calcul du plus grand point fixe (exemple)

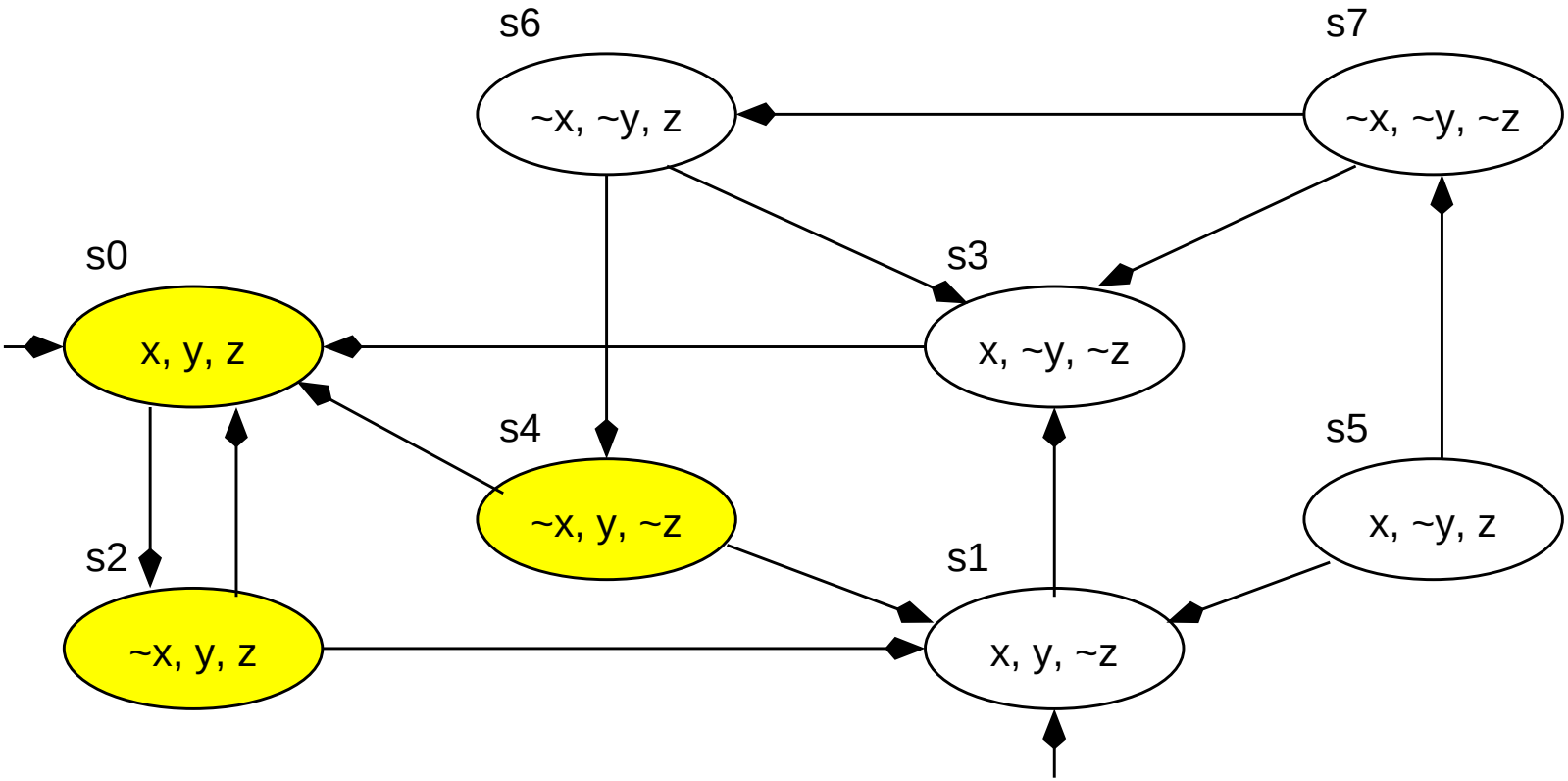
Calcul de $[\![\mathbf{EG}\, y]\!]$



$$\pi^1(Q) \;=\; [\![y]\!] \cap \delta^{-1}(Q)$$

Calcul du plus grand point fixe (exemple)

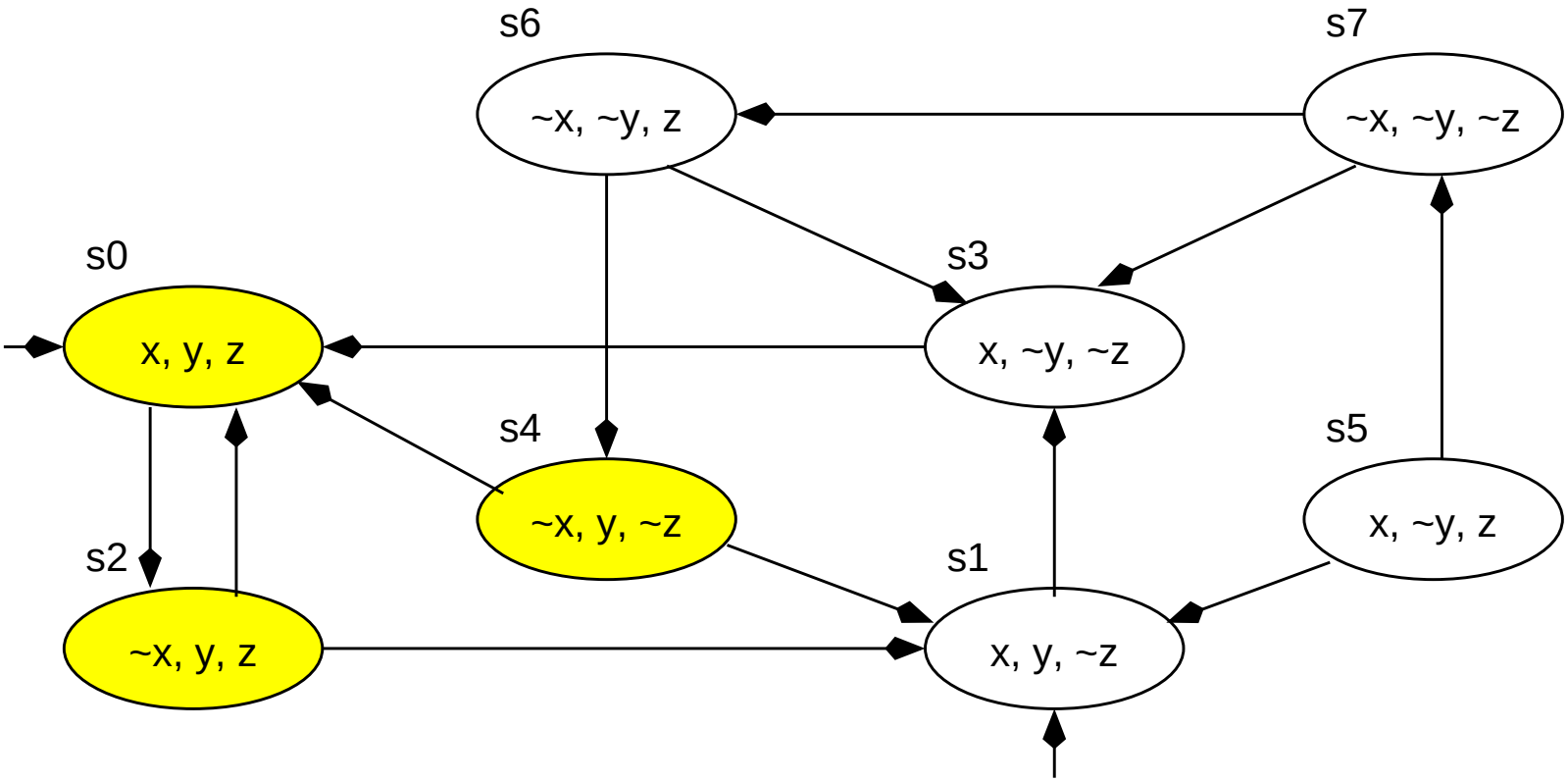
Calcul de $[\![\mathbf{EG}\, y]\!]$



$$\pi^2(Q) \;=\; [\![y]\!] \cap \delta^{-1}(\pi^1(Q))$$

Calcul du plus grand point fixe (exemple)

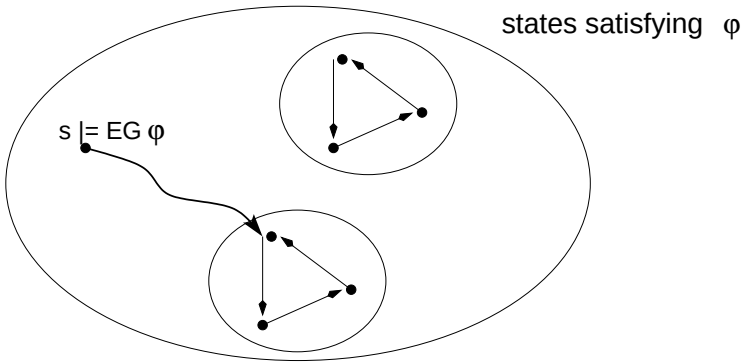
Calcul de $[\![\mathbf{EG}\, y]\!]$



$$\pi^3(Q) \;=\; [\![y]\!] \cap \delta^{-1}(\pi^2(Q)) \;=\; \pi^2(Q) \;=\; [\![\mathbf{EG}\, y]\!]$$

Vérification CTL : complexité

- Algorithme du point fixe $O(|\varphi| \cdot |Q| \cdot (|Q| + |\delta|))$
- Algorithme optimisé [Clarke, Emerson, Sistla 1986]
  - calcul de $[\![\mathbf{EG}\ \varphi]\!]_{\mathcal{K}}$
    1. restreindre aux états qui vérifient φ
    2. calculer les CFC
    3. utiliser la recherche en arrière pour trouver les états à partir desquels un CFC est accessible



  - calcul de $[\![\varphi\ \mathbf{EU}\ \psi]\!]_{\mathcal{K}}$: réduction similaire
- Complexité $O(|\varphi| \cdot (|Q| + |\delta|))$
  - linéaire en la taille du modèle et de la formule

Techniques d'optimisation

- Explosion combinatoire
  - les problèmes commencent à partir de 10^6–10^7 états
  - le nombre d'états est exponentielle en le nombre de composants
  - 10^{100} états correspondent à 300 bits

Techniques d'optimisation

- Explosion combinatoire
  - les problèmes commencent à partir de 10^6–10^7 états
  - le nombre d'états est exponentielle en le nombre de composants
  - 10^{100} états correspondent à 300 bits
- Comprimer : représentations efficaces
- Réduire : explorer des sous-modèles
- Abstraire : transformer le problème

Techniques d'optimisation

- Explosion combinatoire
- Comprimer : représentations efficaces
  - représentation compacte en mémoire
  - utilisation de signatures au lieu d'états
  - calcul symbolique : manipulations efficaces d'ensembles (BDD)
- Réduire : explorer des sous-modèles
- Abstraire : transformer le problème

Techniques d'optimisation

- Explosion combinatoire
- Comprimer : représentations efficaces
- Réduire : explorer des sous-modèles
  - exemple : actions A, B commutatives
    - ⋆ à partir de tout état où A et B sont possibles :
      - aucune des actions ne désactive l'autre
      - exécutions de $A; B$ et de $B; A$ aboutissent dans le même état
    - ⋆ propriété ne dépend pas d'états intermédiaires
    - ⋆ il suffit d'explorer un sous-ensemble d'actions possibles (mais trouver un ensemble minimal est NP-complet . . .)
  - autre exemple : stabilité sous permutations de valeurs
- Abstraire : transformer le problème

Techniques d'optimisation

- Explosion combinatoire
- Comprimer : représentations efficaces
- Réduire : explorer des sous-modèles
- Abstraire : transformer le problème
  - approche ad hoc :
    analyse de petits instance (2 processus, 3 valeurs, 1 ligne de cache)
  - rationnaliser cette démarche : préserver les propriétés
  - approche prometteuse : abstractions booléennes

Outline

Quelques problèmes actuels

- Systèmes ayant un nombre infini d'états
  - systèmes temps réel et hybrides
  - systèmes probabilistes
  - pushdown systems, regular model checking, ...
- Software model checking
  - analyser des programmes C, Java, ...
  - abstractions successives : CEGAR
  - outils prototypiques, dont
    - ★ BLAST `http://www-cad.eecs.berkeley.edu/~rupak/blast/`
    - ★ MAGIC `http://www.cs.cmu.edu/~chaki/magic`
    - ★ SLAM `http://www.research.microsoft.com/slam/`
- Intégration du model checking dans les processus de conception
  - vérification de modèles en langages largement utilisés (e.g., UML)
  - combinaison avec techniques d'analyse quantitative