Cours de Model Checking

Leçon 2.2 : Algorithmes de Model Checking

Sébastien Bardin

CEA-LIST, Laboratoire de Sûreté Logicielle

sebastien.bardin@cea.fr
http://sebastien.bardin.free.fr/

Plan

Cours 2.1 : logiques temporelles

Cours 2.2 : algorithmes

- Introduction
- Prélude 1 : accessibilité, invariance, sûreté
- Prélude 2 : SCC
- CTL model checking
- LTL model checking
- Fair CTL model checking
- Conclusion
- Bonus techniques

Rappels & Contenu

Model Checking

Technique de vérification automatique de systèmes réactifs

Ingrédients

- $\mathcal{M}$ = système de transitions
- φ = formule temporelle
- MC = est-ce que $\mathcal{M} \models \varphi$?

Trois algorithmes de model checking

- (cas simple : accessibilité, invariance, sûreté)
- CTL : technique de marquage des états
- LTL : transformation de la formule en automate de Büchi
- Fair CTL : marquage de CTL + détection des états "fair"

Remarque : importance des composantes fortements connexes (SCC) pour chaque algo

Plan

Propriétés sur les systèmes réactifs

- Accessibilité *Une certaine situation peut être atteinte*
- Invariance *Chaque état local respecte une bonne propriété*
- Sûreté *Quelquechose de mauvais n'arrive jamais*
- Vivacité *Quelquechose de bon finit par arriver*
- Équité *Quelquechose de bon se répète infiniment souvent*
- Équivalence comportementale *Est-ce que 2 systèmes sont équivalents ?*

Le même algorithme permet de traiter toutes ces propriétés

Propriétés sur les systèmes réactifs

- Accessibilité *Une certaine situation peut être atteinte*
- Invariance *Chaque état local respecte une bonne propriété*
- Sûreté *Quelquechose de mauvais n'arrive jamais*
- Vivacité *Quelquechose de bon finit par arriver*
- Équité *Quelquechose de bon se répète infiniment souvent*
- Équivalence comportementale *Est-ce que 2 systèmes sont équivalents ?*

Le même algorithme permet de traiter toutes ces propriétés

États accessibles

Un état $q \in Q$ est accessibles à partir de q_0 si il existe un chemin de $\mathcal{M} = \langle Q, \rightarrow, \mathrm{AP}, l, s_0 \rangle$ menant de q_0 à q.

Accessibilité en un coup : $\mathrm{post} = \{(q, q') | q \rightarrow q'\}$

Ensemble d'accessibilité à partir de q_0 : $\mathrm{post}^*(q_0)$

- facile à calculer dans le cas fini
- itération de $X \mapsto \mathrm{post}(X) \cup X$ à partir de $\{q_0\}$ jusqu'à stabilisation
- permet de vérifier accessibilité et invariance

Invariance et accessibilité

Invariance : tous les états ont une bonne propriété

ex : on a toujours $x \neq 0$

Accessibilité : un certain ensemble d'états est accessible

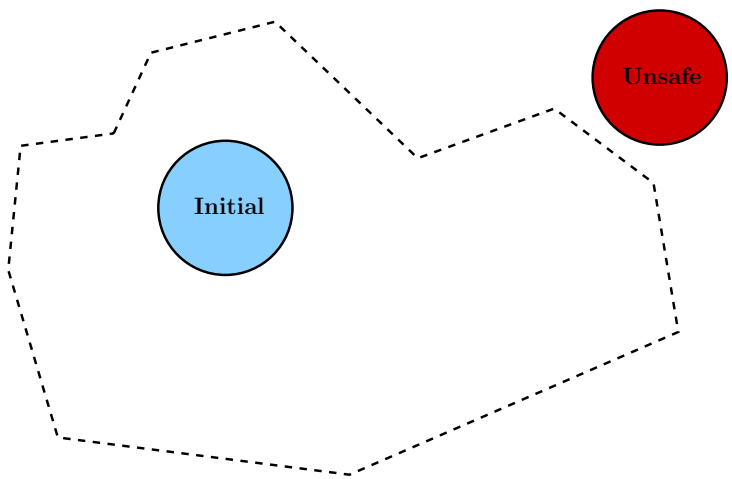
ex : tout point du programme peut être atteint

Propriétés simples mais fondamentales.

Facile à vérifier grâce à $\text{post}^*(q_0)$

- Accessibilité : est-ce que $\text{post}^*(q_0) \cap A \neq \emptyset$?
- Invariance : est-ce que $\text{post}^*(q_0) \subseteq I$?

Idée : itération de post + tests ensemblistes



Invariance et accessibilité

Invariance : tous les états ont une bonne propriété

ex : on a toujours $x \neq 0$

Accessibilité : un certain ensemble d'états est accessible

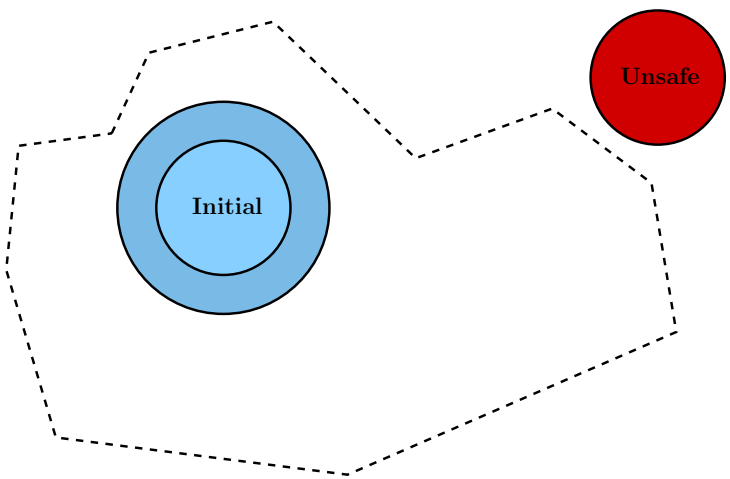
ex : tout point du programme peut être atteint

Propriétés simples mais fondamentales.

Facile à vérifier grâce à $\text{post}^*(q_0)$

- Accessibilité : est-ce que $\text{post}^*(q_0) \cap A \neq \emptyset$?
- Invariance : est-ce que $\text{post}^*(q_0) \subseteq I$?

Idée : itération de post + tests ensemblistes



Invariance et accessibilité

Invariance : tous les états ont une bonne propriété

ex : on a toujours $x \neq 0$

Accessibilité : un certain ensemble d'états est accessible

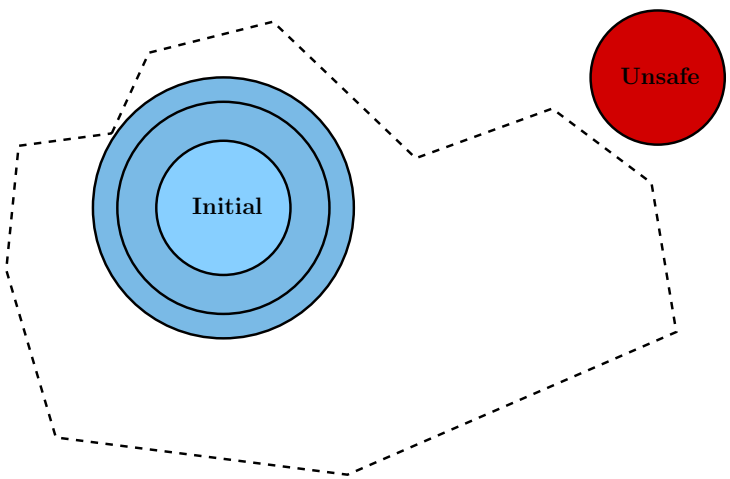
ex : tout point du programme peut être atteint

Propriétés simples mais fondamentales.

Facile à vérifier grâce à $\text{post}^*(q_0)$

- Accessibilité : est-ce que $\text{post}^*(q_0) \cap A \neq \emptyset$?
- Invariance : est-ce que $\text{post}^*(q_0) \subseteq I$?

Idée : itération de post + tests ensemblistes



Invariance et accessibilité

Invariance : tous les états ont une bonne propriété

ex : on a toujours $x \neq 0$

Accessibilité : un certain ensemble d'états est accessible

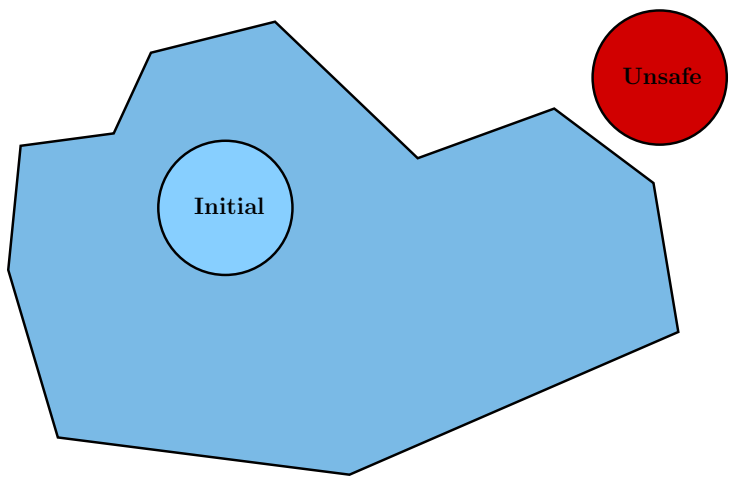
ex : tout point du programme peut être atteint

Propriétés simples mais fondamentales.

Facile à vérifier grâce à $\text{post}^*(q_0)$

- Accessibilité : est-ce que $\text{post}^*(q_0) \cap A \neq \emptyset$?
- Invariance : est-ce que $\text{post}^*(q_0) \subseteq I$?

Idée : itération de post + tests ensemblistes



Calcul de $\text{post}^*(q_0)$

Entrées : structure $\mathcal{M}$ avec état initial s_0

Sorties : ensemble d'accessibilité $\text{post}^*(s_0)$

$R := \emptyset$ /* *états atteints et traîtés* */
$Q := \{s_0\}$ /* *états atteints non traîtés* */
Tant que Q non vide **faire**
  choisir $q \in Q$, $Q := Q - q$
  Si $q \notin R$ **alors**
    $R := R + \{q\}$, $post_q := \emptyset$
    Pour tout q' **tq** $q \rightarrow q'$ **faire** /* *calcul de post*(q) */
      $post_q := post_q + \{q'\}$
    Fin Pour tout
    $Q := Q \cup \text{post}_q$
  Fin Si
Fin Tant que
retourner R

Complexité : linéaire en $|\mathcal{M}|$

(mais $|\mathcal{M}|$ exponentiel si description succinte)

Sûreté

Sûreté : quelquechose de mauvais n'arrive jamais.

Quand j'accède à mon compte, j'ai entré le bon password avant.

Autre définition : contre-exemples finis.

Plus général que invariance.

Comment vérifier : se ramener à accessibilité dans un système modifié

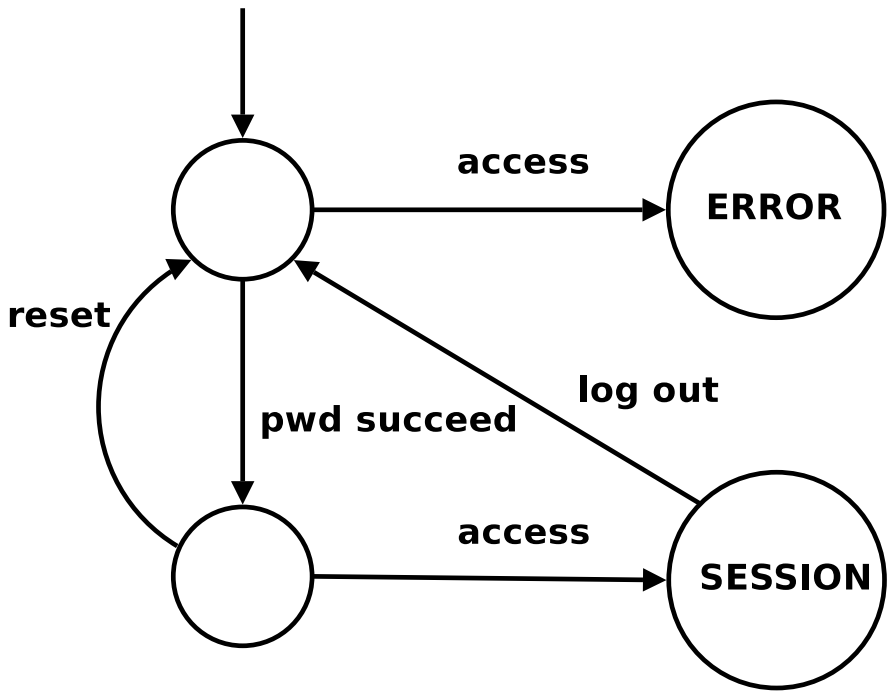
- Automate observeur
- Synchronisé avec $\mathcal{M}$
- Évalue le comportement (ne modifie pas les configurations)

Les techniques de calcul de $\text{post}^*(q_0)$ permettent de vérifier l'accessibilité, l'invariance et la sûreté.

Sûreté II

Propriété simple : Quand j'accède à mon compte, j'ai entré le bon password avant.

Automate observateur $\mathcal{O}$ pour la proriété



Vérifier que $(\_, \mathit{ERROR})$ n'est pas accessible

Plan

Avant-propos

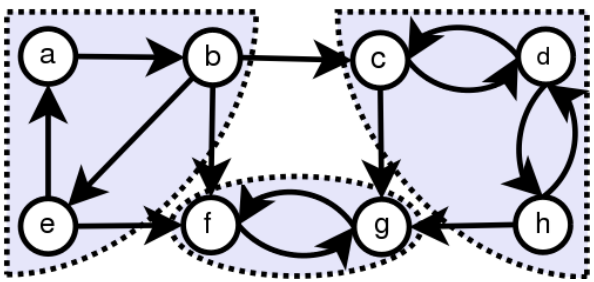
Composante connexe d'un graphe $G = \langle Q, T \rangle$

- un ensemble de noeuds tel que tous les noeuds sont reliés les uns aux autres (pas forcément directement)
- $C \subseteq Q$ tq $c_i \xrightarrow{*} c_j$ pour tout $c_i, c_j \in C$

Quelques définitions

- fortement connexe : composante connexe maximale
- non triviale : le sous-graphe associé a au moins un arc

Décomposer un graphe en SCC : linéaire (Tarjan, Kosaraju)



Avant-propos (2)

Liens entre SCC et les algorithmes de model checking

CTL	:	gérer le cas **EG**
Fair CTL	:	décider les chemins fair
LTL	:	test du vide des automates de Büchi

Plan

Marquage des états

Algorithme de marquage des états (labeling)

- Entrées : $\mathcal{M} = \langle Q, \rightarrow, P, I, s_0 \rangle$ et $\varphi \in$ CTL
- Sortie : est-ce que $\mathcal{M} \models \varphi$?
- Importance historique (1981, Clarke et Emerson)
- Efficace : linéaire en chacune des entrées

Principe : raisonner sur les états plutôt que sur les traces

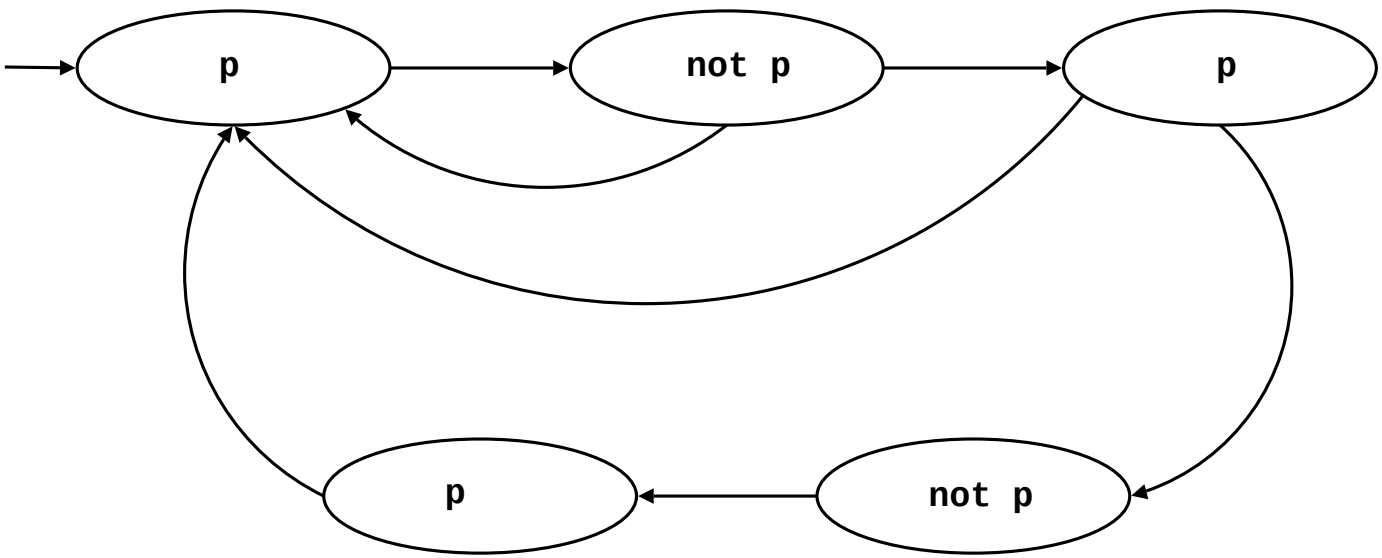
- Marquer les états de $\mathcal{M}$ vérifiant les sous formules de φ
- Marquage récursif, les sous-formules d'abord
- $\mathcal{M} \models \varphi$ ssi s_0 est marqué pour φ

Remarques

- raisonner en terme d'états plutôt que d'exécutions
- très spécifique à CTL

Exemple

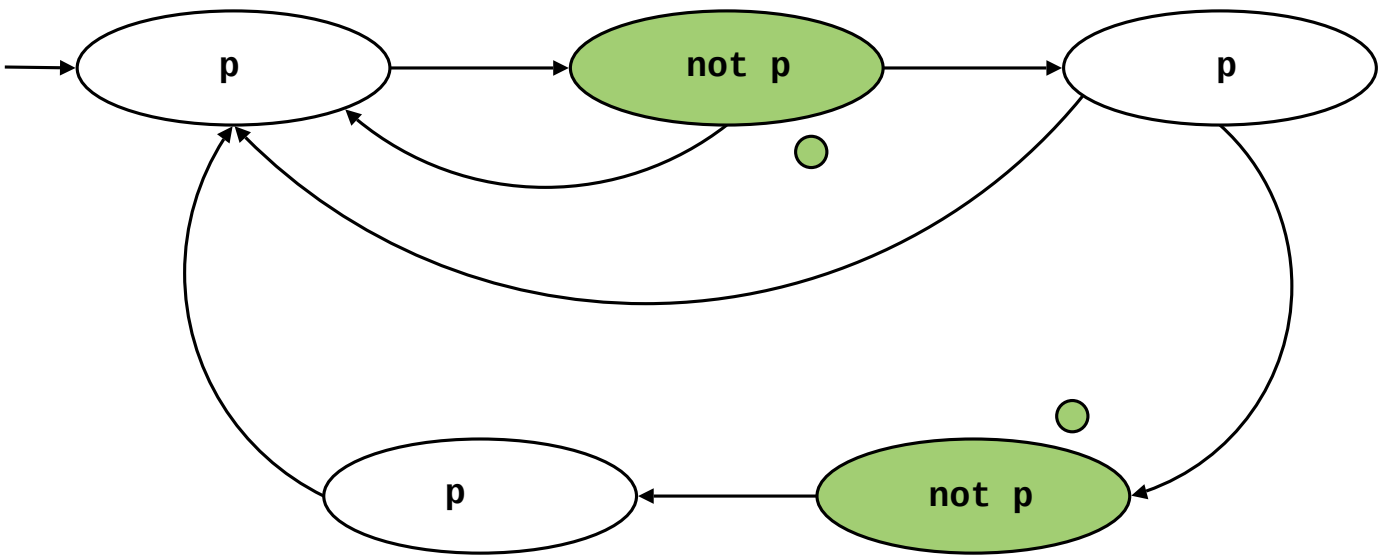
On veut vérifier $(\neg \mathbf{EX} \neg p) \wedge p$



Exemple

On veut vérifier $(\neg \mathbf{EX} \neg p) \wedge p$

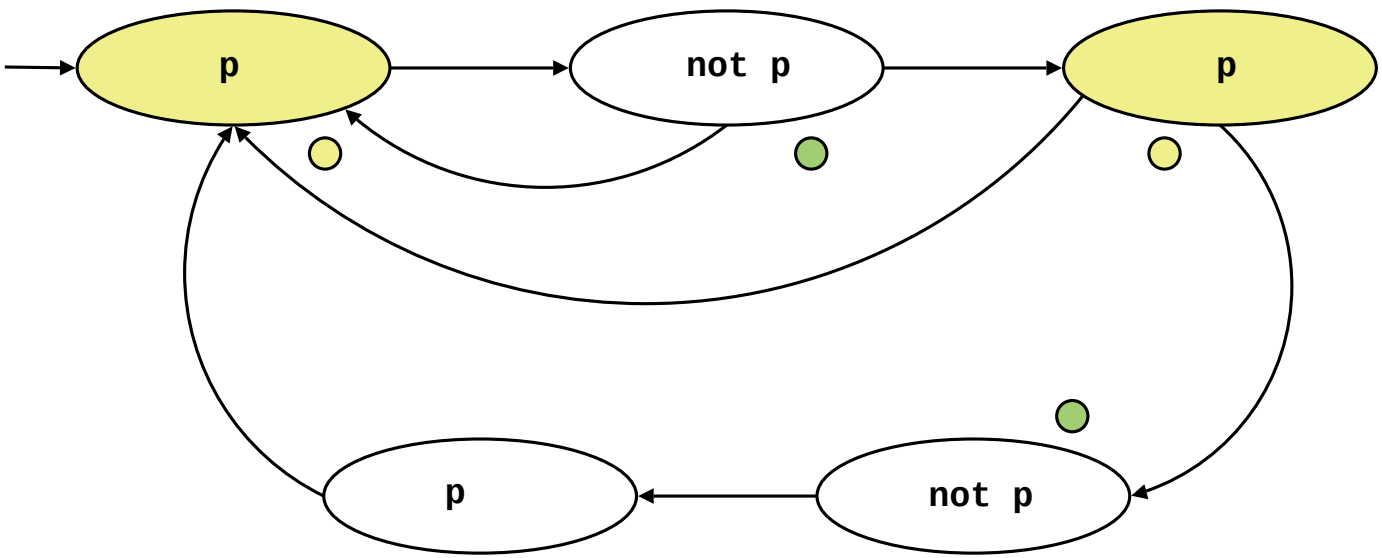
$\neg p$



Exemple

On veut vérifier $(\neg \mathbf{EX} \neg p) \wedge p$

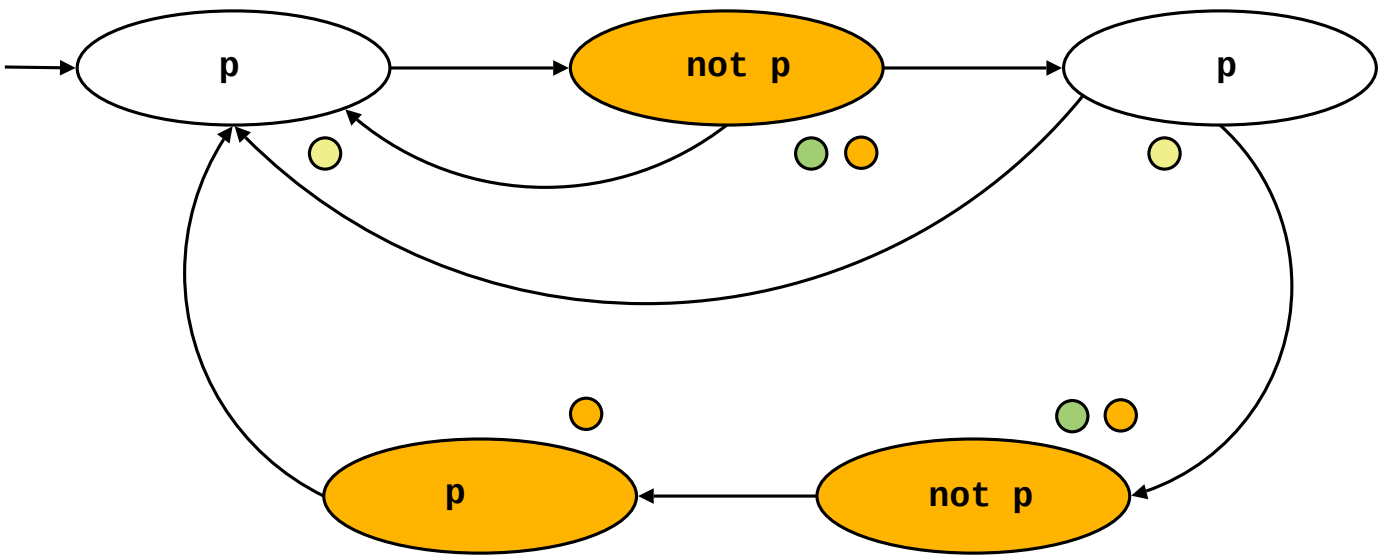
$\mathbf{EX} \neg p$



Exemple

On veut vérifier $(\neg \mathbf{EX} \neg p) \wedge p$

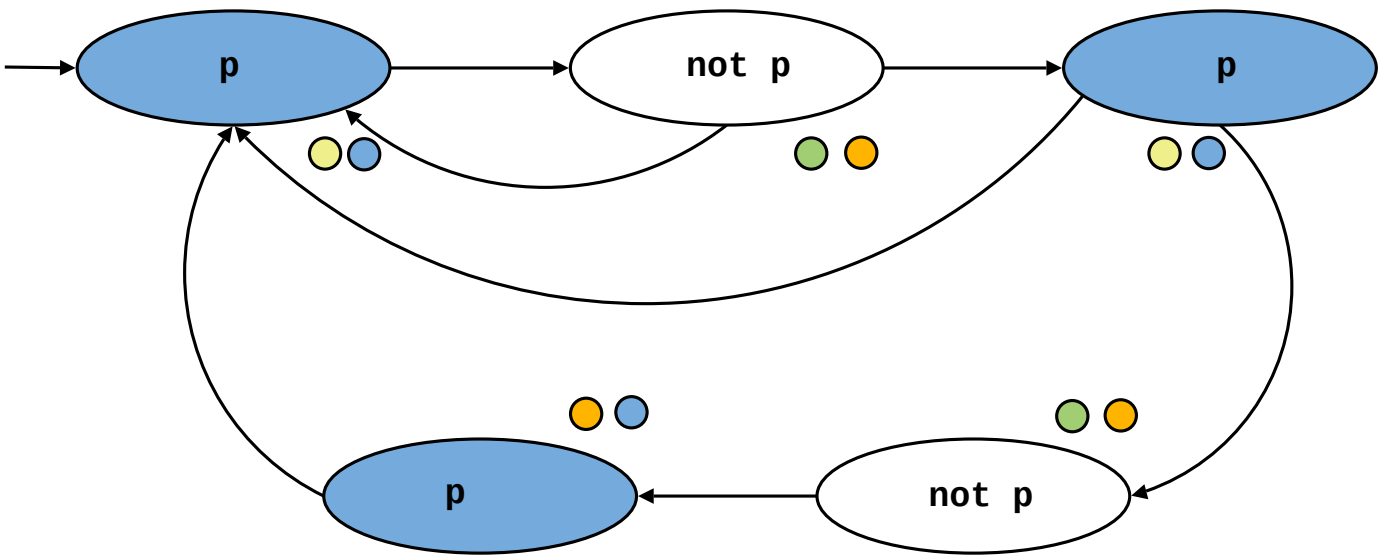
$(\neg \mathbf{EX} \neg p)$



Exemple

On veut vérifier $(\neg \mathbf{EX} \neg p) \wedge p$

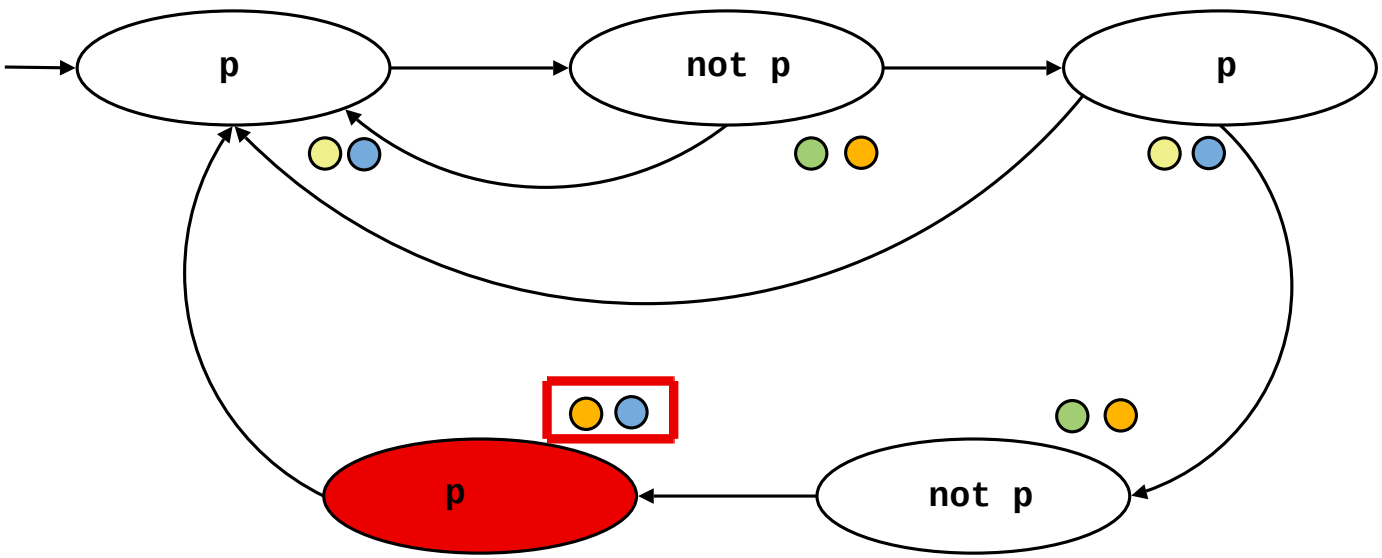
p



Exemple

On veut vérifier $(\neg \mathbf{EX} \neg p) \wedge p$

$(\neg \mathbf{EX} \neg p) \wedge p$



Calcul des marquages

Exemples simples

- $\varphi_1 \wedge \varphi_2$: marquer les états marqués pour φ_1 et pour φ_2
- $\neg\varphi$: marquer les états non marqués pour φ
- **EX**φ : marquer les états qui peuvent atteindre en 1 étape un état marqué par φ
- **EF**φ : marquer les états qui peuvent atteindre en 0, 1 ou $+$ étapes un état marqué par φ
- **AX**φ : marquer les états dont tous les successeurs en une étape sont marqués par φ

Calcul des marquages II

Opérations de base

- q marqué pour p ssi $p \in l(q)$ (donné par $\mathcal{M}$)
- q marqué pour $\varphi \wedge \psi$ ssi q marqué pour φ et q marqué pour ψ

Cas $\mathbf{EX}\varphi$

- q marqué pour $\mathbf{EX}\varphi$

  ssi $q \rightarrow q'$ telque q' marqué pour φ

Cas $\mathbf{EF}\varphi$

- q marqué pour $\mathbf{EF}\varphi$

  ssi $q \rightarrow \ldots \rightarrow q'$ et q' marqué pour Q_φ

Calcul des marquages III

On note Q_φ l'ensemble des états marqués pour φ

Cas $\mathbf{E}\varphi\mathbf{U}\psi$

- $q \in Q_{\mathbf{E}\varphi\mathbf{U}\psi}$

  ssi q peut atteindre Q_ψ par un chemin restant dans Q_φ

Cas $\mathbf{EG}\varphi$

- SCC' ensemble des SCC de Q_φ
- L ensemble des états de SCC'
- $q \in \mathbf{EG}\varphi$

  ssi q peut atteindre L en restant dans Q_φ

Détails de l'algorithme I

Algorithme MARKING

input : formule φ normalisée, $\mathcal{M} = \langle Q, \rightarrow, P, I, s_0 \rangle$

1: Case 1 : $\varphi = p$
2: for all s $\in Q$ do
3: if $p \in I(s)$ then s.φ := true
4: else s.φ := false
5: end for

1: Case 2 : $\varphi = \neg\varphi'$
2: do MARKING(φ',$\mathcal{M}$);
3: for all s $\in Q$ do s.φ := not(s.φ') end for

1: Case 3 : $\varphi = \varphi' \wedge \varphi''$
2: do MARKING(φ',$\mathcal{M}$); MARKING(φ'',$\mathcal{M}$);
3: for all s $\in Q$ do s.φ := and(s.φ',s.φ'') end for

Détails de l'algorithme I

Algorithme MARKING

input : formule φ normalisée, $\mathcal{M} = \langle Q, \rightarrow, P, I, s_0 \rangle$

1: Case 4 : $\varphi = \mathbf{EX}\varphi'$
2: do MARKING(φ',$\mathcal{M}$);
3: for all s $\in Q$ do s.φ := false end for
4: for all $(s, s') \in \rightarrow$ do
5: if s'.φ'=true then s.φ := true
6: end for

Détails de l'algorithme I

Algorithme MARKING

input : formule φ normalisée, $\mathcal{M} = \langle Q, \rightarrow, P, I, s_0 \rangle$

```
 1: Case 5 : φ = E φ' U φ''
 2: do MARKING(φ',M); MARKING(φ'',M);
 3: for all s ∈ Q do
 4:     s.φ := false;
 5:     s.seenbefore := false
 6: end for
 7: L := ∅
 8: for all s ∈ Q do if s.φ''=true then L:=L + {s} end for
 9: while L ≠ ∅ do
10:     choose s ∈ L; L := L - {s};
11:     s.φ := true;
12:     For all (s',s) ∈→ do // s' predecessor of s
13:         if s'.seenbefore = false then
14:             s'.seenbefore := true;
15:             if s'.φ' = true then L := L + {s'};
16:         end if
17:     end for
18: end while
```

Détails de l'algorithme I

Algorithme MARKING

input : formule φ normalisée, $\mathcal{M} = \langle Q, \rightarrow, P, I, s_0 \rangle$

```
 1: Case 6 : φ = A φ' U φ''
 2: do MARKING(φ',M); MARKING(φ'',M);
 3: L := ∅;
 4: for all s ∈ Q do
 5:     s.nb := degree(s); s.φ := false;
 6:     if s.φ'' = true then L := L + {s};
 7: end for
 8: while L ≠ ∅ do
 9:     choose s ∈ L; L := L - {s};
10:     s.φ := true;
11:     for all (s',s) ∈→ do // s' predecessor of s
12:         s'.nb := s'.nb - 1;
13:         if (s'.nb = 0) and (s'.φ' = true) and (s'.φ = false) do
14:             L := L + {s'};
15:         end if
16:     end for
17: end while
```

Détails de l'algorithme I

Algorithme MARKING

input : formule φ normalisée, $\mathcal{M} = \langle Q, \rightarrow, P, I, s_0 \rangle$

```
 1: Case 7 : φ = EGφ'
 2: Q' := {s | φ' ∈ I(s) }; /* computed with MARKING(φ',M) */
 3: SCC := { C | C non trivial SCC of Q' };
 4: L := ⋃_{C∈SCC} { s | s ∈ C };
 5: for all s ∈ L do s.φ := true end for
 6: while L ≠ ∅ do
 7:    choose s ∈ L; L := L - {s};
 8:    for all (s',s) ∈→ such that s' ∈ Q' do
 9:       if (s'.φ = false) then
10:          s'.φ := true;
11:          L := L + {s'};
12:       end if
13:    end for
14: end while
```

CTL Model Checking

Gestion des différents opérateurs

- minimum d'opérateurs : concision (théorie, code)
- ajout de cas spéciaux : efficacité

Complexité

- complexité en $\mathcal{O}(|\mathcal{M}| \cdot |\varphi|)$
- linéaire en chaque entrée

CTL Model Checking

Gestion des différents opérateurs

- minimum d'opérateurs : concision (théorie, code)
- ajout de cas spéciaux : efficacité

Complexité

- complexité en $\mathcal{O}(|\mathcal{M}| \cdot |\varphi|)$
- linéaire en chaque entrée

ATTENTION

- $\mathcal{M}$ supposé déjà calculé
- calculer $\mathcal{M}$ est exponentiel (variables, concurrence)

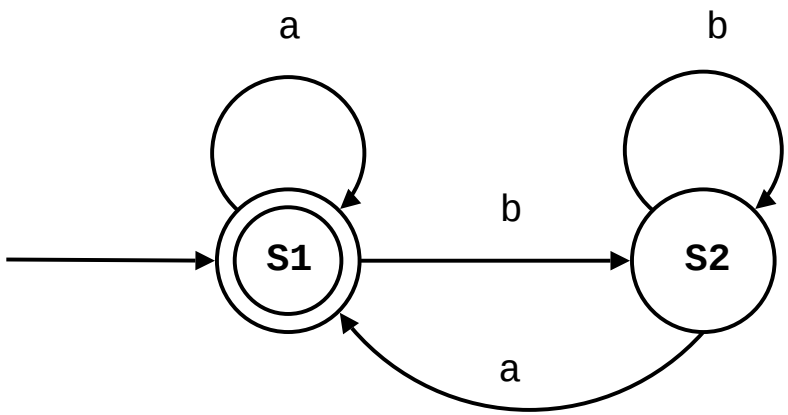
Plan

Automates de Büchi

Extension des automates pour travailler sur des mots infinis

Automate de Büchi $B = \langle \Sigma, Q, \rightarrow, q_0, F \rangle$

- idem que automate sauf pour acceptance
- σ accepté ssi σ visite infiniment F



Aut. fini mots terminant par a
Aut. Büchi mots ayant une infinité de a

Propriétés des automates de Büchi

Permettent de représenter des ensembles infinis de traces (infinies)

- langages dits ω-réguliers
- on note $\mathcal{L}(B)$ le langage (ω-régulier) de B

Permettent de manipuler des ensembles infinis de traces (infinies)

- opérations simples pour $\cup, \cap, \ldots$
- test du vide : on sait tester sur B si $\mathcal{L}(B)$ est vide
- intersection : on sait calculer $B_\otimes$ tq $\mathcal{L}(B_\otimes) = \mathcal{L}(B_1 \cap \mathcal{L}(B_2)$
- ...

Propriétés des automates de Büchi II

Points communs avec les automates finis

- $\cup$ et $\cap$ polynômiaux
- test du vide polynômial

Lien avec les SCC

- soit FSCC les SCC intersectant F
- soit T l'ensemble des états des FSCC
- $\mathcal{L}(B) \neq \emptyset$ ssi T accessible de q_0

Le problème de la complémentation

- très coûteuse en théorie $\mathcal{O}(2^{n^2})$ au mieux
- très coûteuse en pratique et difficile à implanter
- souvent implantée en $\mathcal{O}(2^{2^n})$

Intérêt pour le model checking

Un système de Kripke $\mathcal{M}$ défini naturellement un ensemble ω-régulier de traces infinies représenté par $B_{\mathcal{M}}$

Étant donné un ensemble ω-régulier de traces infinies B_{bad}, il est facile de vérifier que $\mathcal{L}(B_{\mathcal{M}}) \cap \mathcal{L}(B_{bad}) = \emptyset$

Problème : comment passer d'une formule LTL φ à un automate de Büchi $B_{\neg\varphi}$ telle que $[\![\varphi]\!] = \mathcal{L}(B_{\neg\varphi_p})$

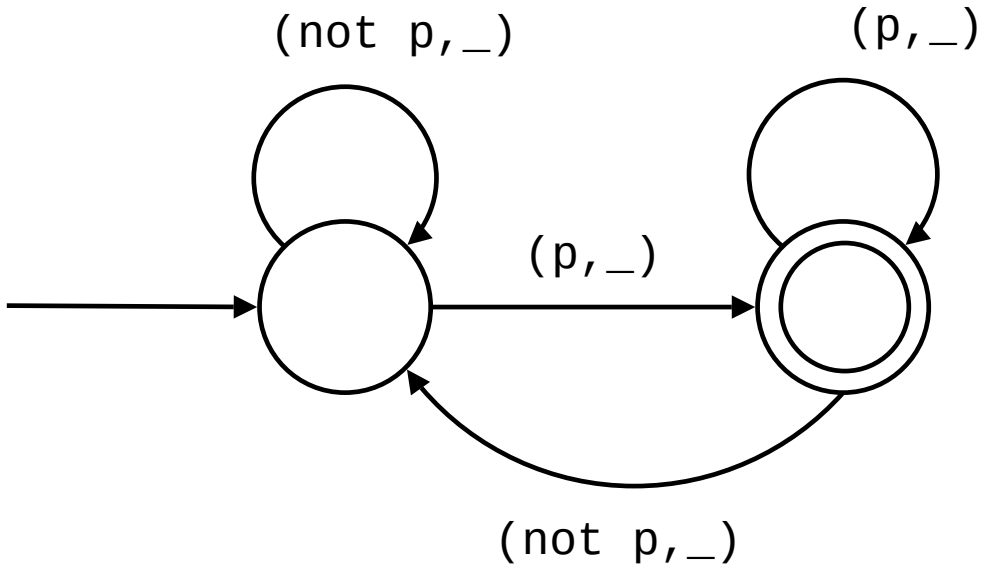
Une fois qu'on a ça, l'algorithme est le suivant

1. Transformer $\mathcal{M}$ en automate $B_{\mathcal{M}}$ (trivial)
2. Transformer φ_p en automate $B_{\neg\varphi_p}$
3. Calculer $B_{\otimes}$ reconnaissant $\mathcal{L}(B_{\mathcal{M}}) \cap \mathcal{L}(B_{\neg\varphi_p})$
4. Tester si $\mathcal{L}(B_{\otimes}) = \emptyset$

Lien formules LTL - automates de Büchi

Théorème

On peut traduire automatiquement toute formule LTL en automate de Büchi définissant le même langage.



Exemple de **AGF***p*

Complexité de la transformation

- exponentiel (attention à complémentation !)

Complexité du Model Checking de LTL

Algorithme

1. Transformer $\mathcal{M}$ en automate de Büchi $B_{\mathcal{M}}$ (trivial)
2. Transformer φ_p en automate de Büchi $B_{\neg\varphi_p}$
3. Calculer $B_{\otimes}$ reconnaissant $\mathcal{L}(B_{\mathcal{M}}) \cap \mathcal{L}(B_{\neg\varphi_p})$
4. Tester si $\mathcal{L}(B_{\otimes}) = \emptyset$

1 est direct, 2 est exponentiel, 3 et 4 poynômiaux

Complexité en temps et en espace en $\mathcal{O}(|\mathcal{M}| \times 2^{|\varphi|})$

- Exponentiel seulement en $|\varphi|$, linéaire en $|\mathcal{M}|$
- LTL-MC est PSPACE-complet, donc algo pas optimal

Plan

Fair CTL

CTL + équité (restreinte) : ajout dans le modèle directement

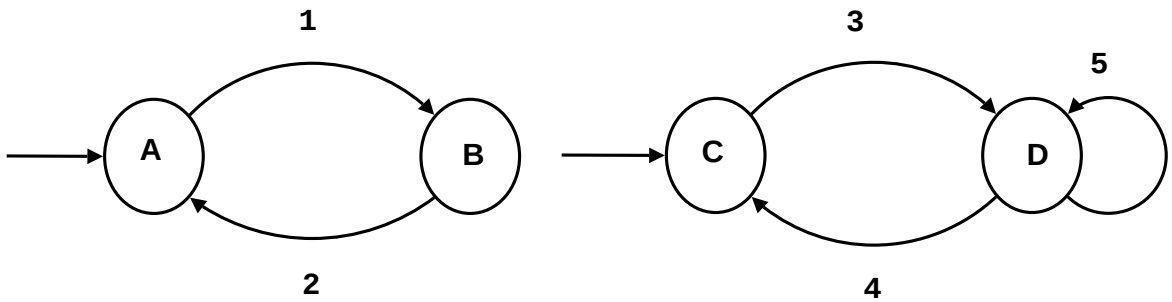
Idée = on modifie la sémantique de S

- contraintes d'équité : ensembles d'ensembles d'états $F_1, \ldots, F_n \subseteq Q$
- chemins fair = passant infiniment souvent par chaque ensemble F_i
- états fair = d'où part un chemin fair

Nouvelle relation de satisfaction $\models_{\text{fair}}$

- $\mathcal{M}, s \models_{\text{fair}} p$ ssi $p \in l(s)$ et s est un état fair
- $\mathcal{M}, s \models_{\text{fair}} \mathbf{A}\varphi$ ssi tous les chemins fair partant de s vérifient φ
- $\mathcal{M}, s \models_{\text{fair}} \mathbf{E}\varphi$ ssi il existe un chemins fair partant de s vérifiant φ

Sémantique "Fair" : exemple



En sémantique normale (asynchrone)

- le chemin $(1.2)^w$ est un chemin légal du système
- chemin certainement irréaliste
- rajoute une sorte de deadlock sur la machine 2

En sémantique *fair* (asynchrone)

- contraintes d'équité : $F_1 = \{A\}, F_2 = \{B\}, F_3 = \{C\}, F_4 = \{D\}$
- le chemin $(1.2)^w$ n'est plus un chemin légal du système

Sémantique "Fair" II

Cette forme d'équité est utile en pratique

- ex : "chaque composante progresse"
- ex : "les messages sont infiniment souvent reçus"

Lien avec CTL*

- (Fair CTL) $\mathbf{A}\varphi_p \approx \mathbf{A}(\mathbf{F}^\infty F_1 \wedge \ldots \wedge \mathbf{F}^\infty F_k \rightarrow \varphi_p)$
- (Fair CTL) $\mathbf{E}\varphi_p \approx \mathbf{E}(\mathbf{F}^\infty F_1 \wedge \ldots \wedge \mathbf{F}^\infty F_k \bigwedge \varphi_p)$

Remarque

- une formule CTL définit un sous-ensemble de Q
- les F_i peuvent être donnés par des formules CTL

Trouver les états fair

- Fair SCC : SCC qui intersecte chaque F_i
- Les états fair sont ceux qui atteignent une Fair SCC

Fair CTL Model Checking

Principe de l'algorithme

1. trouver les états fair par accessibilité de Fair SCC
   on peut adapter algo pour **EG**true
2. marquer ces états avec une nouvelle proposition `fair`
3. se ramener au cas normal en exprimant $\mathcal{M}, s \models_{\text{fair}} \varphi$ en fonction de $\models, \varphi$, `fair`

Complexité en $\mathcal{O}(|\mathcal{M}| \cdot |\varphi| \cdot |F|)$

- complexité semblable à CTL
- et on gagne équité
- en pratique Fair CTL beaucoup plus utile que CTL

Plan

Conclusion

Algorithmes importants

- aspect historique
- la plupart des algos + avancés sont des extensions (cf mc infini)

Limitations importantes

- systèmes finis
- systèmes petits, car $\mathcal{M}$ supposé pré-calculé

La suite

- algorithmes efficaces pour les systèmes finis
- systèmes infinis (software)

Plan

cas $\mathbf{EG}\varphi$, lien avec les SCC

soit une structure de Kripke $\mathcal{M} = \langle Q, \rightarrow, P, l, s_0 \rangle$

Proposition : un état $q \in Q$ vérifie $\mathbf{EG}\varphi$ ssi $q \xrightarrow{\pi} q'$ tq tous les états de π vérifient φ et q' appartient à une composante fortement connexe non triviale (mais pas forcément maximale) dont tous les états vérifient φ

($\Leftarrow$) : ok

($\Rightarrow$) : Soit $q \in Q$ vérifiant $\mathbf{EG}\varphi$. Donc il existe un chemin infini σ partant de q tq φ est vraie dans tous les états de σ. Comme σ est infini et que Q est fini, σ passe forcément une infinité de fois par au moins un état. Soit q' un de ces états. On a donc :

$q \xrightarrow{\pi_1} q' \xrightarrow{\pi_2} q' \xrightarrow{\cdots}$ (remarque : $\pi_2 \neq \epsilon$)

Comme $\pi_1\pi_2$ est un préfixe de σ, tous les états de π_1 vérifient φ.
De plus, q' appartient à la composante fortement connexe non triviale $q' \xrightarrow{\pi_2} q'$ dont tous les états vérifient φ.

cas $\mathbf{EG}\varphi$, lien avec les SCC (2)

Conséquence : calcul du marquage (soit Q_φ les états marqués pour φ)

remarque : "composante fortement connexe non triviale de $\mathcal{M}$ dont tous les états vérifient φ" peut se ramener à "composante fortement connexe non triviale maximale de $\mathcal{M}$ restreint à Q_φ"

- trouver les SCC (maximale) de Q_φ. On note L l'ensemble de tous ces états
- alors : marquer pour $\mathbf{EG}\varphi$ les états qui atteignent L par un chemin dont tous les états vérifient φ
- plus formellement : $Q_{\mathbf{EG}\varphi} = (\lambda X \mapsto pre(X) \cap Q_\varphi)^*(L)$

Rappel : automates de Büchi

Outil pour reconnaître / manipuler des ensembles (infinis) de mots infinis

Automate de Büchi $B = \langle \Sigma, Q, \rightarrow, q_0, F \rangle$

- Σ alphabet (ensemble fini)
- Q ensemble fini d'états
- $\rightarrow \subseteq Q \times \Sigma \times Q$ les transitions
- q_0 l'état initial
- $F \subseteq Q$ l'ensemble des états finaux / acceptants

Un mot infini σ est reconnu si son exécution dans l'automate B, i.e.
$q_0 \xrightarrow{\sigma_0} q_1 \xrightarrow{\sigma_1} \ldots \xrightarrow{\sigma_n} q_n \xrightarrow{\sigma_{n+1}} \ldots$
passe infiniment souvent par l'ensemble F

Test du vide

Proposition : $\mathcal{L}(B) \neq \emptyset$ ssi il existe $q_F \in F$ tel que $q_0 \xrightarrow{*} q_F$ et q_F appartient à une SCC (non triviale, maximale) de B

($\Leftarrow$) : ok car de q_F on peut atteindre q_F, d'où un chemin infini partant de q_0 et passant infiniment par q_F, donc accepté par B

($\Rightarrow$) : Soit un mot infini σ accepté par B. Donc l'exécution infinie de σ dans B passe infiniment souvent par F. Comme F est fini, il existe au moins un $q_F \in F$ tel que l'exécution de σ passe infiniment souvent par q_F : $q_0 \xrightarrow{*} q_F \xrightarrow{+} q_F \rightarrow \ldots$. De plus, q_F appartient à la composante fortement connexe non triviale $q_F \xrightarrow{+} q_F$. Il appartient donc forcément aussi à une SCC non triviale maximale.

Algorithme de test du vide :

- calculer les SCC de B. Ne garder que celles intersectant F (notées FSCC). Soit L l'ensemble des états des FSCC.
- alors $\mathcal{L}(B) \neq \emptyset$ ssi q_0 peut atteindre L
- ou encore $\mathcal{L}(B) \neq \emptyset$ ssi $post^*(q_0) \cap L \neq \emptyset$ ssi $q_0 \in pre^*(L)$

Union d'automates de Büchi

Problème : union d'automates de Büchi

- Entrées : deux automates de Büchi sur le même alphabet Σ
  $B_1 = \langle \Sigma, Q_1, \rightarrow_1, qi_1, F_1 \rangle$ et $B_2 = \langle \Sigma, Q_2, \rightarrow_2, qi_2, F_2 \rangle$
- Question : calculer $B = \langle \Sigma, Q, \rightarrow, qi, F \rangle$ tq
  $\mathcal{L}(B) = \mathcal{L}(B_1) \cup \mathcal{L}(B_2)$

Solution : $B = \langle \Sigma, Q_1 \times Q_2, \rightarrow, (qi_1, qi_2), (F_1 \times Q_2) \cup (Q_1 \times F_2) \rangle$ et $\rightarrow$ est définie par : (soit $a \in \Sigma$)

- $(q_1, q_2) \xrightarrow{a} (q_1', q_2')$ ssi $q_1 \xrightarrow{a}_1 q_1'$ et $q_2 \xrightarrow{a}_2 q_2'$

($\Leftarrow$) : ok car l'automate B déroule l'exécution d'un mot σ en parallèle sur B_1 et sur B_2. Par exemple, si $\sigma \in \mathcal{L}(B_1)$, alors σ dans B passera infiniment souvent par des états de la forme (q_{F_1}, q_2) et sera accepté par B. Idem pour $\mathcal{L}(B_2)$.
($\Rightarrow$) : Le mot est accepté par B ssi il passe infiniment souvent par F_1 ou par F_2. Donc σ passe infiniment souvent par au moins un des deux, donc il est accepté au moins par un des deux automates.

Intersection d'automates de Büchi

Problème : intersection d'automates de Büchi

- Entrées : deux automates de Büchi sur le même alphabet Σ
  $B_1 = \langle \Sigma, Q_1, \rightarrow_1, qi_1, F_1 \rangle$ et $B_2 = \langle \Sigma, Q_2, \rightarrow_2, qi_2, F_2 \rangle$
- Question : calculer $B = \langle \Sigma, Q, \rightarrow, qi, F \rangle$ tq
  $\mathcal{L}(B) = \mathcal{L}(B_1) \cap \mathcal{L}(B_2)$

Solution : $B = \langle \Sigma, Q_1 \times Q_2 \times \{1,2\}, \rightarrow, (qi_1, qi_2, 1), F_1 \times Q_2 \times \{1\} \rangle$ et $\rightarrow$ est définie par : (soit $a \in \Sigma$)

- $(q_1, q_2, 1) \xrightarrow{a} (q_1', q_2', 1)$ ssi $q_1 \xrightarrow{a}_1 q_1'$ et $q_2 \xrightarrow{a}_2 q_2'$ et $q_1' \notin F_1$
- $(q_1, q_2, 1) \xrightarrow{a} (q_1', q_2', 2)$ ssi $q_1 \xrightarrow{a}_1 q_1'$ et $q_2 \xrightarrow{a}_2 q_2'$ et $q_1' \in F_1$
- $(q_1, q_2, 2) \xrightarrow{a} (q_1', q_2', 2)$ ssi $q_1 \xrightarrow{a}_1 q_1'$ et $q_2 \xrightarrow{a}_2 q_2'$ et $q_2' \notin F_2$
- $(q_1, q_2, 2) \xrightarrow{a} (q_1', q_2', 1)$ ssi $q_1 \xrightarrow{a}_1 q_1'$ et $q_2 \xrightarrow{a}_2 q_2'$ et $q_2' \in F_2$

Les deux exécutions sont lancées en parallèle, et le témoin (1 ou 2) indique le prochain type d'état final à atteindre (1 pour F_1, 2 pour F_2). Quand on attend un état dans F_i et qu'on l'atteint, le témoin change de valeur. Cela permet d'assurer que l'exécution a bien une infinité de F_1 et une infinité de F_2.

Structure de Kripke vers automate de Büchi

Problème :

- Entrées : une structure de Kripke $\mathcal{M} = \langle Q, \rightarrow, P, l, s_0 \rangle$
- Question : calculer $B = \langle \Sigma, Q', \rightarrow', q_0, F \rangle$ tq $\mathcal{L}(B) = \mathcal{L}(\mathcal{M})$

Solution : soit $P = \{p_1, \ldots, p_n\}$, on note 2^P l'ensemble des sous-ensembles de P. On définit B par $B = \langle 2^P, Q', \rightarrow', q'_0, Q' \rangle$ et :

- Q' = un état par transition $t \in \rightarrow$ et un état initial nouveau q'_0
- les transitions : $t_1 \xrightarrow{D}{}' t_2$ ssi $t_1 = (\_, s_j)$, $t_2 = (s_j, \_)$ et $D \subseteq P = l(s_j)$, et les transitions $q'_0 \xrightarrow{l(s_0)} (s_0, \_)$

Intuition : si on s'intéressait aux transitions de $\mathcal{M}$: B serait comme $\mathcal{M}$ (l'alphabet serait le "nom" des transitions), tous les états seraient acceptants. Mais on veut des propriétés de suites d'états : on inverse les états / transitions de $\mathcal{M}$, l'alphabet est le sous-ensemble des propriétés vraies (intuitivement : lues dans l'état courant).

Rappel : Fair CTL

CTL + équité (restreinte) : ajout dans le modèle directement

- contraintes d'équité : ensembles d'ensembles d'états $F_1, \ldots, F_n \subseteq Q$
- chemins fair = passant infiniment souvent par chaque ensemble F_i
- états fair = état d'où part un chemin fair

Nouvelle relation de satisfaction $\models_{\text{fair}}$

- $\mathcal{M}, s \models_{\text{fair}} p$ ssi $p \in l(s)$ et s est un état fair
- $\mathcal{M}, s \models_{\text{fair}} \mathbf{A}\varphi$ ssi tous les chemins fair partant de s vérifient φ
- $\mathcal{M}, s \models_{\text{fair}} \mathbf{E}\varphi$ ssi il existe un chemins fair partant de s vérifiant φ

Principe de l'algorithme de MC pour Fair CTL

1. trouver les états fair par accessibilité de Fair SCC
2. marquer ces états avec une nouvelle proposition `fair`
3. se ramener au cas normal en exprimant $\mathcal{M}, s \models_{\text{fair}} \varphi$ en fonction de $\models, \varphi$, `fair`

Trouver les états fair

Problème : marquage des états fair

- Entrées : une structure de Kripke $\mathcal{M} = \langle Q, \rightarrow, P, I, s_0 \rangle$ et des contraintes d'équités $F_1, \ldots, F_n \subseteq Q$
- Question : trouver les états fair de $\mathcal{M}$

On appelle Fair SCC une SCC qui intersecte chacun des F_i

Proposition : un état $q \in Q$ est fair ssi il existe $q' \in Q$ tel que $q \xrightarrow{*} q'$ et q' appartient à une Fair SCC (non triviale, maximale) de $\mathcal{M}$

($\Leftarrow$) : ok (on peut aisément construire un chemin fair partant de q et bouclant dans la Fair SCC)
($\Rightarrow$) : si q est fair, alors il existe un chemin fair σ qui part de q et passe infiniment osuvent par chacun des F_i. Comme les F_i sont finis, il existe au moins pour chaque F_i un q_{F_i} visité infiniment souvent par σ. On peut donc écrire σ comme : $q \xrightarrow{*} q_{F_1} \xrightarrow{*} \ldots q_{F_n} \xrightarrow{*} q_{F_1} \xrightarrow{*} \ldots$. Il vient que les q_{F_i} sont dans la même SCC (maximale, non triviale), cette SCC est donc fair.

Trouver les états fair (2)

Algorithme :

- trouver les SCC (maximale) de $\mathcal{M}$. Trouver celles qui sont fair. On note L l'ensemble de tous les états des Fair SCC.
- alors : les états fair sont ceux qui peuvent atteignent L
- plus formellement : $Q_{fair} = pre^*(L)$

Réutiliser l'algorithme de marquage de CTL

But : exprimer les propriétés CTL sur une structure fair comme des propriétés CTL sur une structure normale avec la proposition atomique `fair`

(attention : ne veut pas dire que Fair CTL = CTL)

Quelques équivalences :

- $\mathcal{M}, s \models_F p$ ssi $\mathcal{M}, s \models p \wedge \text{fair}$
- $\mathcal{M}, s \models_F \neg p$ ssi $\mathcal{M}, s \models \neg p \wedge \text{fair}$
- $\mathcal{M}, s \models_F \mathbf{EX}\varphi$ ssi $\mathcal{M}, s \models \mathbf{EX}(\varphi \wedge \text{fair})$
- $\mathcal{M}, s \models_F \mathbf{AX}\varphi$ ssi $\mathcal{M}, s \models \text{fair} \wedge \mathbf{AX}(\text{fair} \implies \varphi)$
- $\mathcal{M}, s \models_F \mathbf{AG}\varphi$ ssi $\mathcal{M}, s \models \text{fair} \wedge \mathbf{AG}(\text{fair} \implies \varphi)$
- $\mathcal{M}, s \models_F \mathbf{E}\varphi_1\mathbf{U}\varphi_2$ ssi $\mathcal{M}, s \models \mathbf{E}\varphi_1\mathbf{U}(\varphi_2 \wedge \text{fair})$

Remarque : attention à la négation !!! (rentrer les négations)