Cours de Model Checking

Leçon 3 : Algorithmes de Model Checking

Sébastien Bardin

CEA-LIST, Laboratoire de Sûreté Logicielle

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

Rappels

Model Checking

Technique de vérification automatique de systèmes réactifs

Ingrédients

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

Déjà vu

- modélisation
- calcul de l'espace des états
- accessibilité, invariance, sûreté
- vivacité, équité
- logiques temporelles LTL, CTL, CTL*

Avant-propos

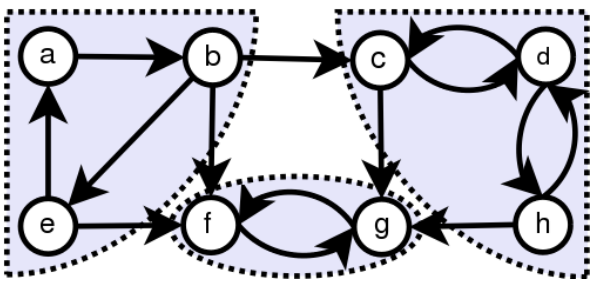
Composante connexe d'un graphe $G = \langle Q, T \rangle$

- 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

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

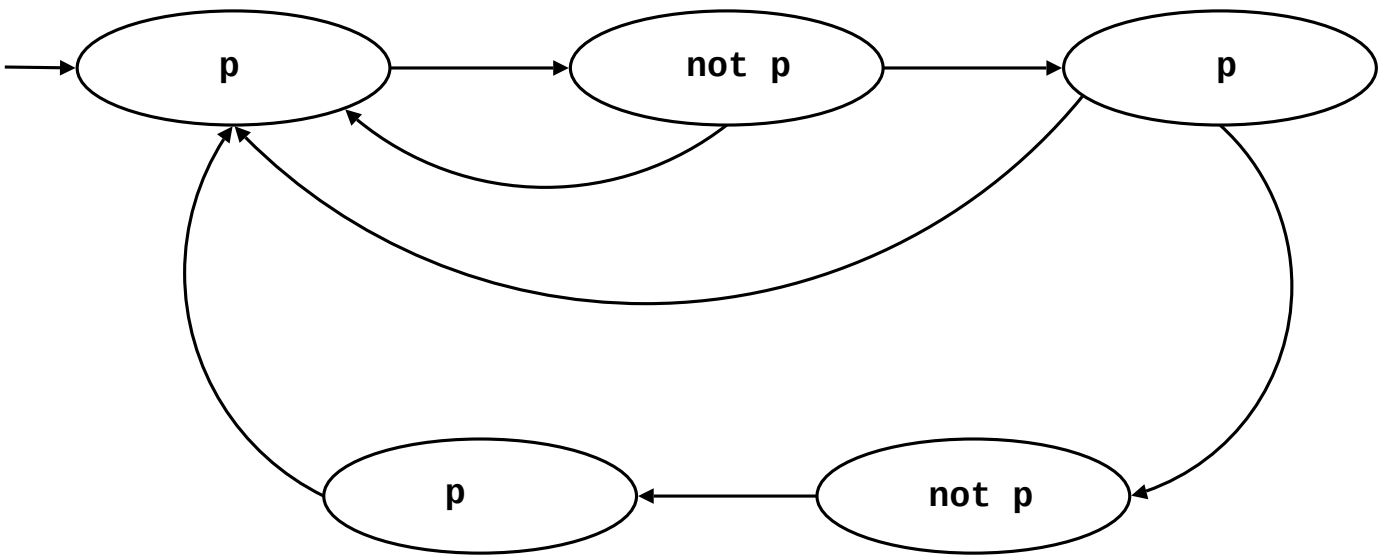
- Marquer les états de $\mathcal{M}$ vérifiant les sous formules de φ
- Marquage récursif, les sous-formules d'abord
- $\mathcal{M} \models \varphi$ ssi s_0 est marqué pour φ

Remarques

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

Exemple

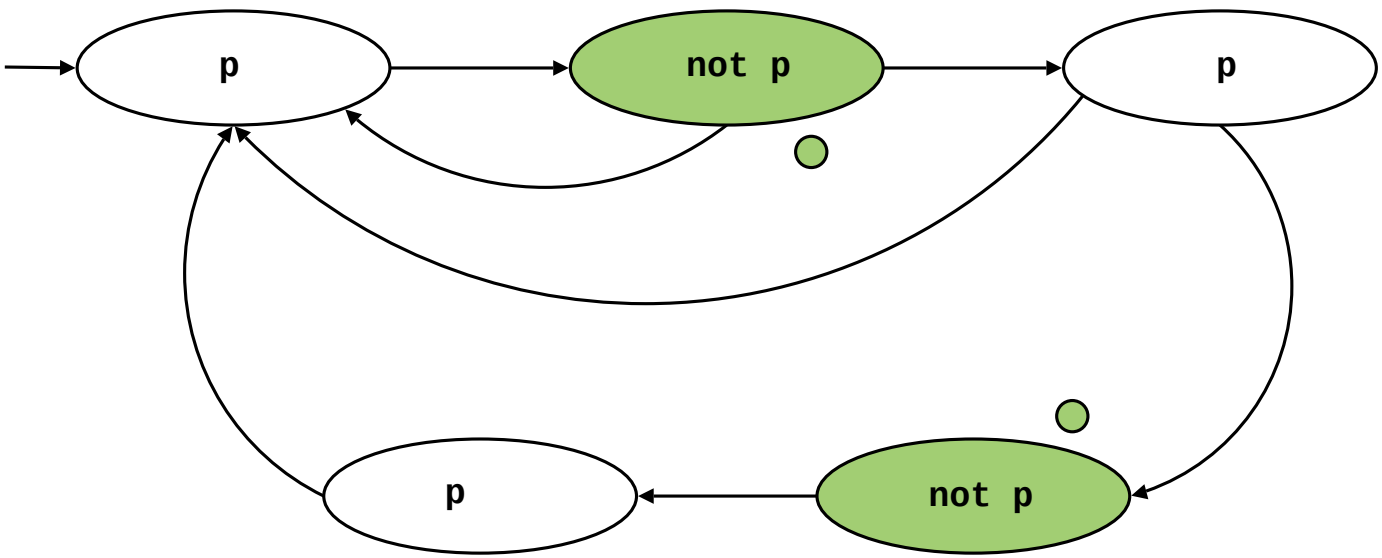
On veut vérifier $(\neg \mathbf{EX} \neg p) \wedge p$



Exemple

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

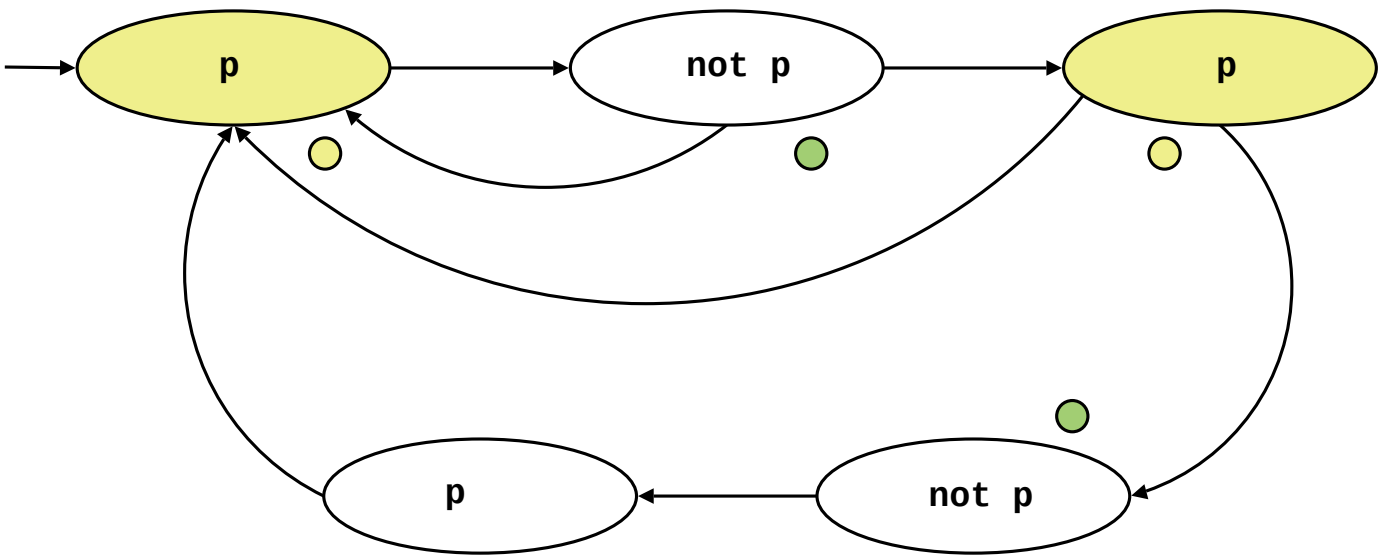
$\neg p$



Exemple

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

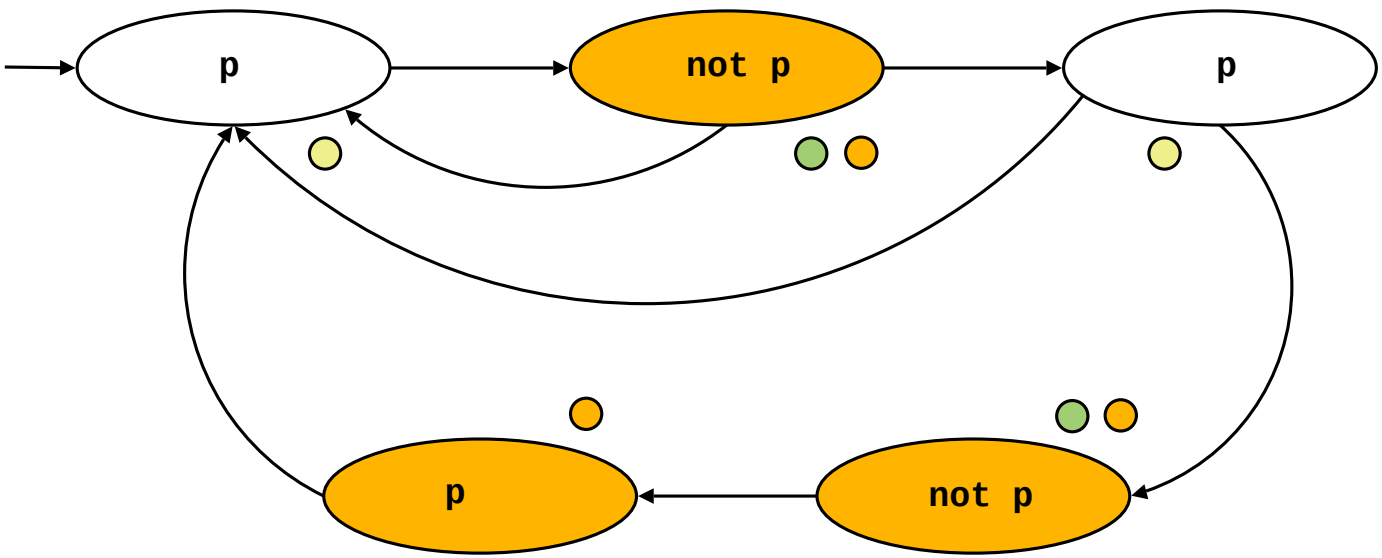
$\mathbf{EX} \neg p$



Exemple

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

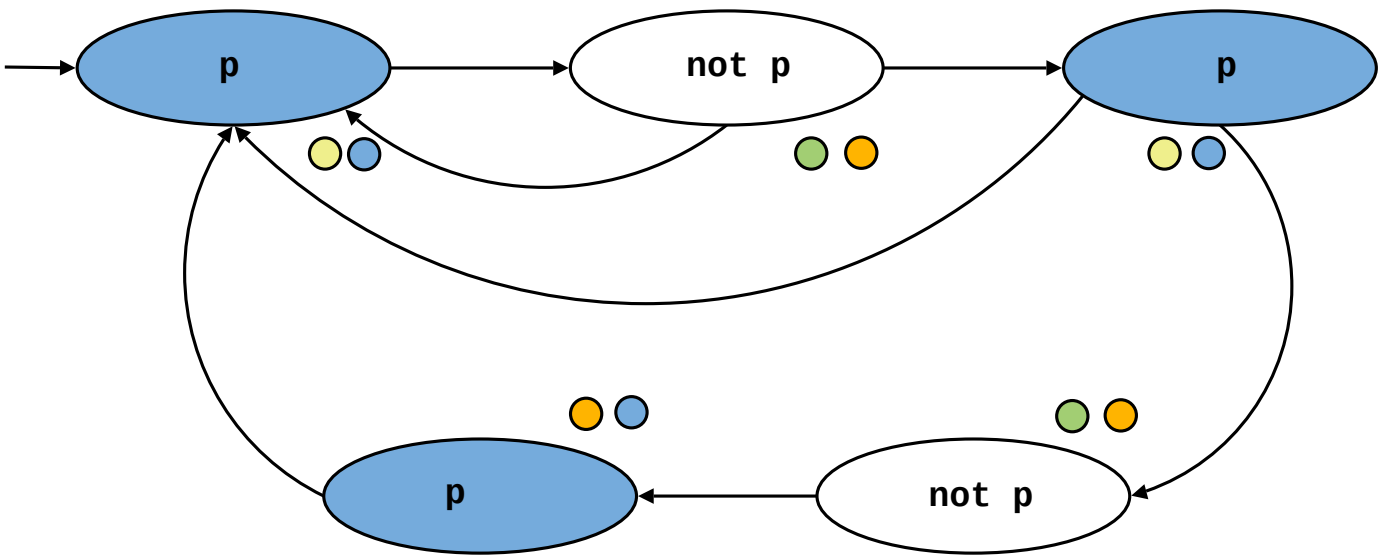
$(\neg \mathbf{EX} \neg p)$



Exemple

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

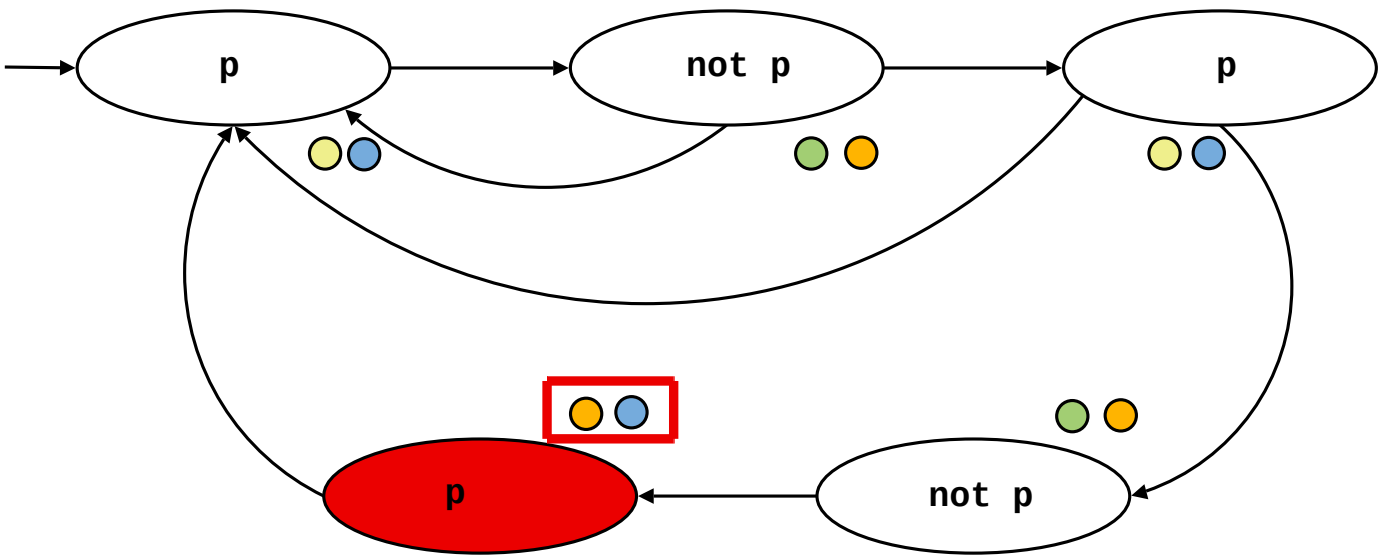
p



Exemple

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

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



Calcul des marquages

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 $q \longrightarrow q'$ telque q' marqué pour φ

Cas $\mathbf{EF}\varphi$

- q marqué pour $\mathbf{EF}\varphi$

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

Calcul des marquages II

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 ens 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

Plan

Fair CTL

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

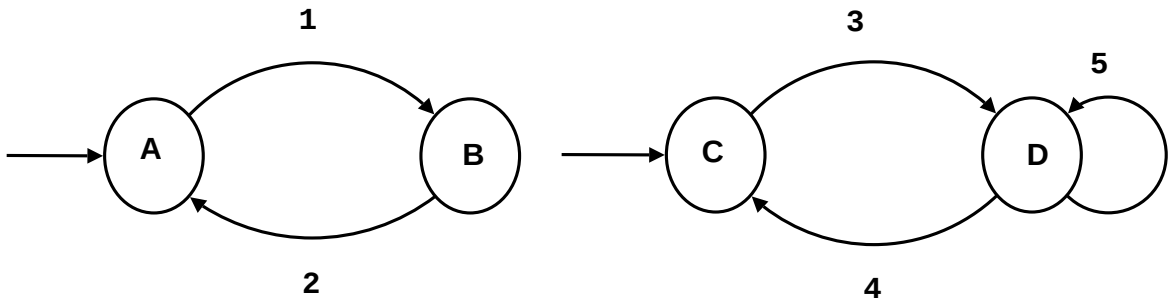
Idée = on modifie la sémantique de S

- 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

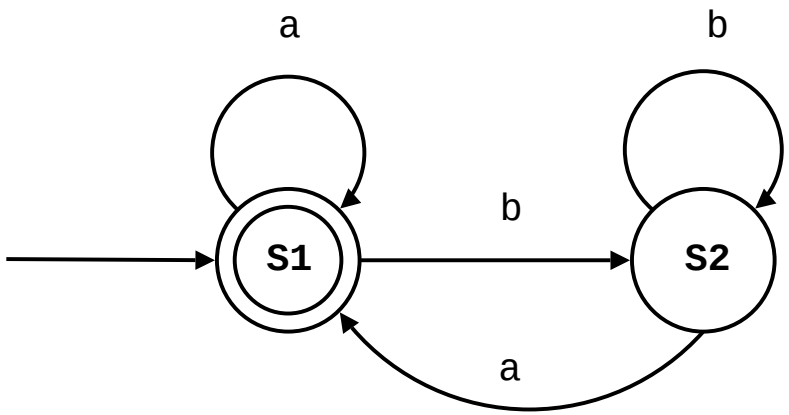
Plan

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 $\mathcal{O}(2^{2n})$

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 $[\![\varphi]\!] = \mathcal{L}(B_{\neg\varphi_p})$

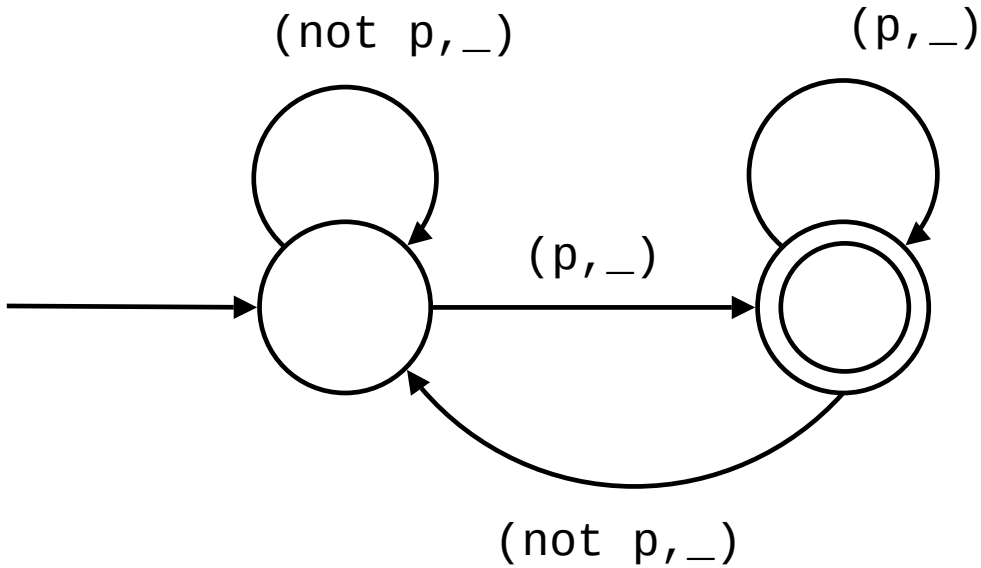
Une fois qu'on a ça, l'algorithme est le suivant

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

- 2-exponentiel si on utilise la complémentation
- exponentiel si on arrive à s'en passer (voir exo)

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

Plan

Model Checking et automates

Initié par Vardi et Wolper (1983)

- Pour LTL optimal : automates de Büchi alternants
- Cas branchant : Automates d'arbres de Büchi (alternants)
- Sûreté et automates finis
- Automates de Büchi comme logique linéaire
  (attention complémentation)

Model Checking Efficace

Model checking symbolique (McMillan 90)

- Ensembles d'états représentés par des équations
- Bien plus concis : $\{x, x \leq 200\}$ plutôt que $1, 2, \ldots, 200$
- Les BDD implantent très efficacement les équations booléennes
- Très adapté à CTL pour circuits synchrones
- Pas parfait : entiers, concurrence, autre logique

Ordres partiels (Valamari, Peled, Godefroid)

- certains états/transitions sont redondants pour certaines propriétés
- pas besoin de construire tout $\mathcal{M}$
- très adapté aux systèmes asynchrones
- pas évident de mixer avec symbolique

Quelques Model Checkers

Outil SMV de CMU (McMillan)

- CTL sur systèmes synchrones avec variables booléennes
- représentation symbolique par BDD
- autres : assume-garantee, abstraction
- 10^{20} états couramment, jusqu'à 10^{800}

Outil SPIN du Bell Labs (Holzmann et Peled)

- LTL sur systèmes asynchrones avec variables non booléennes
- énumération concrète optimisée
- ordres partiels, génération à la volée, memory compression
- 10^{10} états couramment, jusqu'à 10^{30}

Quelques axes de recherche

Model Checking fini

- modularité
- abstraction et raffinement (CEGAR)
- SAT-bounded model checking

Model Checking infini

- décidable : réseaux de Petri, automates à piles, files lossy, ...
- indécidable : compteurs, files, ...

Model Checking temporisé

- time automata : Timed CTL
- systèmes hybrides

Model Checking probabiliste

- cas fini : Probabilistic LTL
- extensions : piles, files lossy, etc.

Software Model Checking

- indécidable
- CEGAR + ordres partiels + theorem proving

Influences académiques I

Diffusion de techniques issues du model checking à d'autres domaines

Liens logiques temporelles et automates (observeurs)

- design by contract, run-time monitoring, model-based testing
- vérification de propriétés de sûreté

Vérification efficace de systèmes concurrents

- model-based testing, test de systèmes multi-tâches

Autres

- Représentation compacte d'ensembles finis
- Analyse statique interprocédurale
- Raffinement d'abstraction par contre-exemple

Influences académiques II

Le model checking est ou a été fortement influencé par

- premiers travaux entre logique et automates
- analyse statique et abstractions
- travaux sur les procédures de décision efficaces (BDD, SAT)