

Automates et Langages de Spécification

Marc Zeitoun

Master informatique 2, Septembre 2011

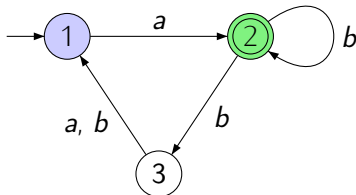
Plan

A. Rappels et exemples.

B. Langages de spécification et automates

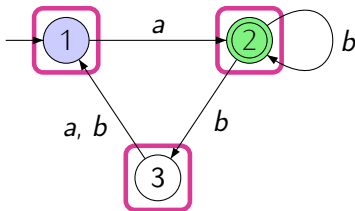
Automates finis

- ▶ Machine lisant un mot en entrée et qui accepte ou rejette ce mot.
- ▶ Capacités de calcul très limitées : chaque lettre lue ne peut qu'influencer une mémoire interne **finie**, les états.



Automates finis

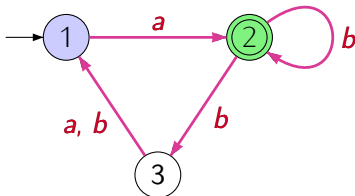
- ▶ Machine lisant un mot en entrée et qui accepte ou rejette ce mot.
- ▶ Capacités de calcul très limitées : chaque lettre lue ne peut qu'influencer une mémoire interne **finie**, les états.



- ▶ États,

Automates finis

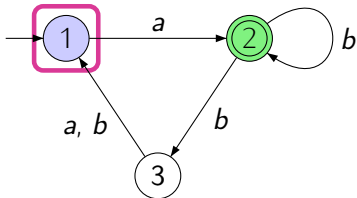
- ▶ Machine lisant un mot en entrée et qui accepte ou rejette ce mot.
- ▶ Capacités de calcul très limitées : chaque lettre lue ne peut qu'influencer une mémoire interne **finie**, les états.



- ▶ États,
- ▶ Transitions, étiquetées sur un alphabet (ici 5, sur $\Sigma = \{a, b\}$),

Automates finis

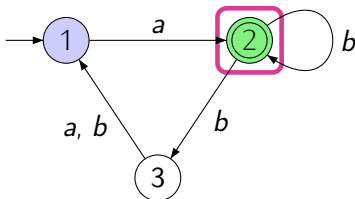
- ▶ Machine lisant un mot en entrée et qui accepte ou rejette ce mot.
- ▶ Capacités de calcul très limitées : chaque lettre lue ne peut qu'influencer une mémoire interne **finie**, les états.



- ▶ États,
- ▶ Transitions, étiquetées sur un alphabet (ici 5, sur $\Sigma = \{a, b\}$),
- ▶ États initiaux,

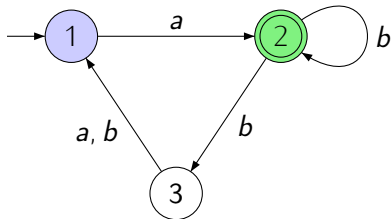
Automates finis

- ▶ Machine lisant un mot en entrée et qui accepte ou rejette ce mot.
- ▶ Capacités de calcul très limitées : chaque lettre lue ne peut qu'influencer une mémoire interne **finie**, les états.



- ▶ États,
- ▶ Transitions, étiquetées sur un alphabet (ici 5, sur $\Sigma = \{a, b\}$),
- ▶ États initiaux,
- ▶ États acceptants.

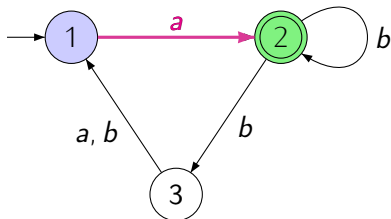
Runs et langage accepté



$$t = a b b a$$

- ▶ Run sur un mot t : chemin étiqueté par t depuis un état initial.

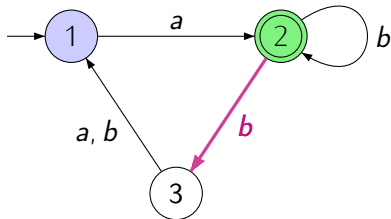
Runs et langage accepté



$t = \boxed{a} b b a$

- ▶ Run sur un mot t : chemin étiqueté par t depuis un état initial.

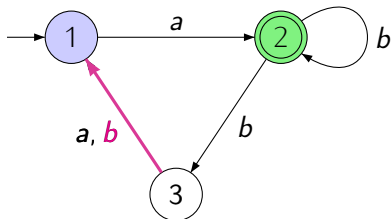
Runs et langage accepté



$$t = a \text{ (b) } b a$$

- Run sur un mot t : chemin étiqueté par t depuis un état initial.

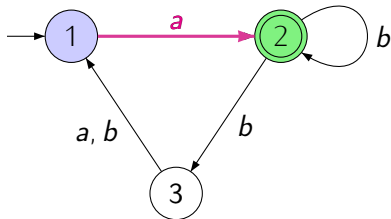
Runs et langage accepté



$$t = a b \boxed{b} a$$

- Run sur un mot t : chemin étiqueté par t depuis un état initial.

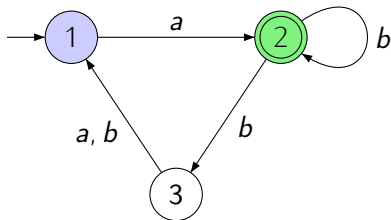
Runs et langage accepté



$t = a b b \boxed{a}$

- ▶ Run sur un mot t : chemin étiqueté par t depuis un état initial.

Runs et langage accepté

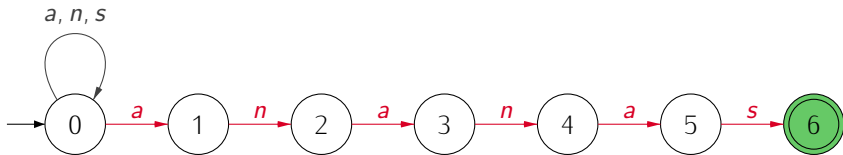


$$t = a b b a$$

- ▶ **Run** sur un mot t : chemin étiqueté par t depuis un état initial.
- ▶ Mot t **accepté** si **au moins** un run sur t va à un état acceptant.
- ▶ **Langage** de l'automate : ensemble des mots acceptés.
Ici : $a[b + b(a + b)a]^*$.

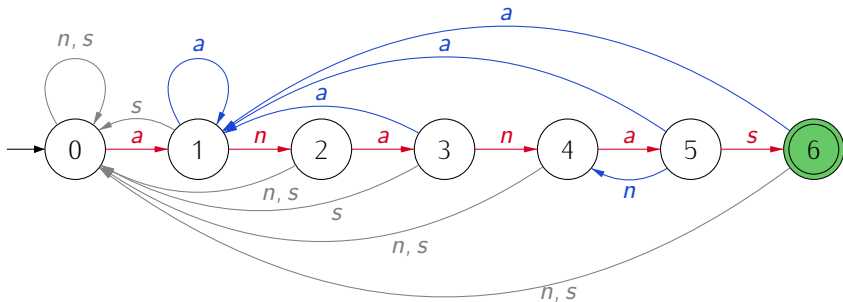
Exemple d'utilisation d'automate : recherche de motif

- Pour rechercher le mot *ananas* dans un texte sur $\{a, n, s\}^*$, il suffit de lire le texte dans l'automate :



Exemple d'utilisation d'automate : recherche de motif

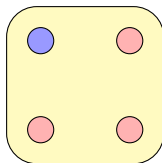
- Pour rechercher le mot *ananas* dans un texte sur $\{a, n, s\}^*$, il suffit de lire le texte dans l'automate **déterministe** :



- Calcul efficace de cet automate : algorithme Knuth-Morris-Pratt.
- Représentation compacte, peu de transitions utiles (I. Simon).

Exemple 2 : barman aveugle

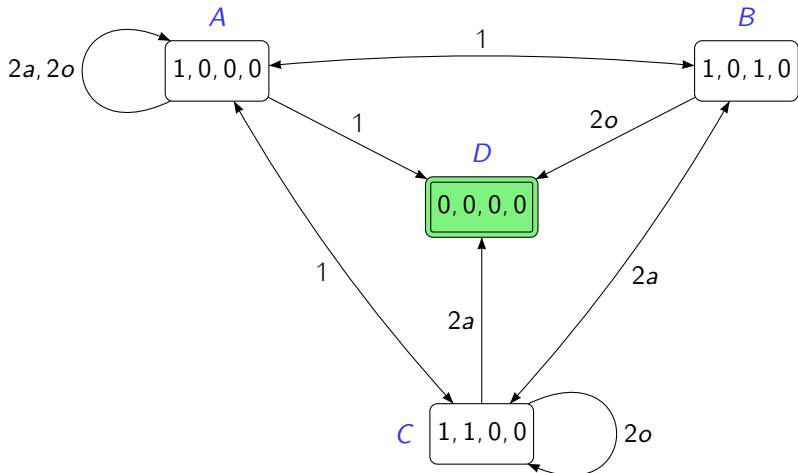
- ▶ Un barman aveugle joue au jeu suivant avec un client.
- ▶ Il a devant lui un plateau avec 4 verres.



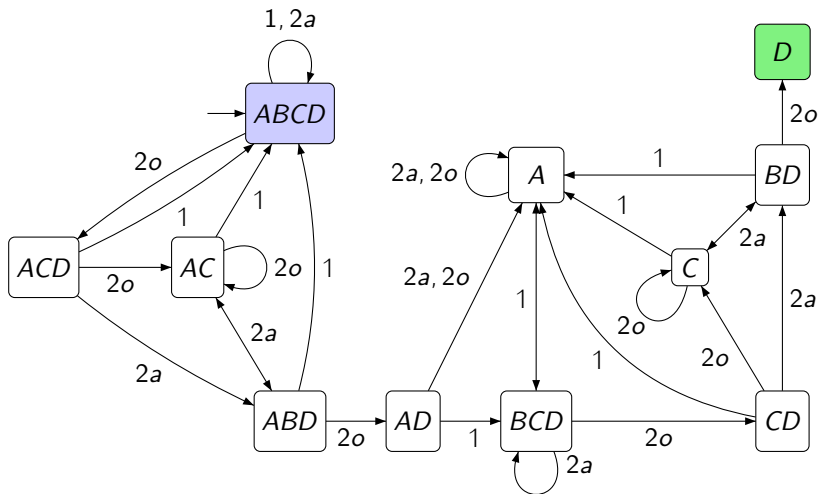
- ▶ À chaque tour le barman fait retourner au client 1 ou 2 verres.
- ▶ S'il s'agit de 2 verres, il précise s'ils sont **adjacents** ou **opposés**.
- ▶ Si tous les verres sont dans le même sens, le barman a gagné.

Barman aveugle : modélisation des situations

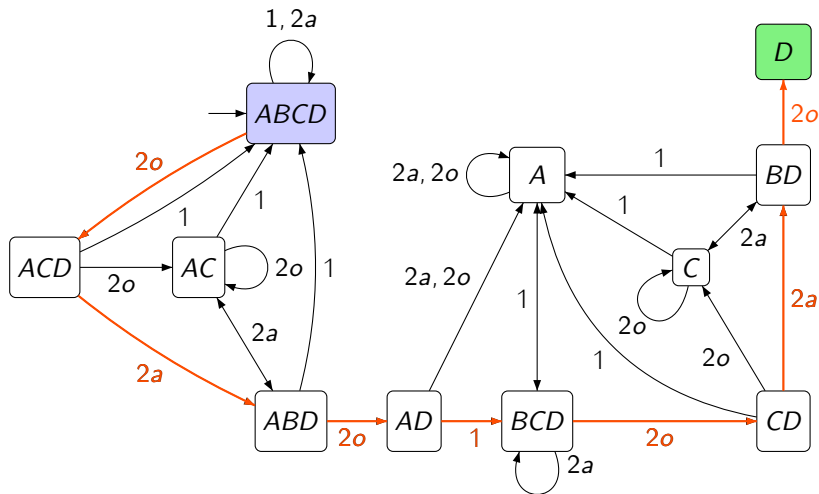
- ▶ $1, 0, 0, 0$ représente 1 verre à l'envers, 3 verres à l'endroit, ou l'inverse.



Barman aveugle : détermination



Barman aveugle : détermination



► La séquence $2o, 2a, 2o, 1, 2o, 2a, 2o$ mène à l'état $0, 0, 0, 0$.

Les automates finis : de nombreuses bonnes propriétés

- ▶ Le pouvoir expressif des **automates non déterministes** est le même que celui des **automates déterministes**...
Mais explosion du nombre d'états potentiellement exponentielle.

$$(a + b)^* a(a + b)^n.$$

Les automates finis : de nombreuses bonnes propriétés

- ▶ Le pouvoir expressif des **automates non déterministes** est le même que celui des **automates déterministes**...
Mais explosion du nombre d'états potentiellement exponentielle.

$$(a + b)^* a(a + b)^n.$$

- ▶ Objet **canonique** pour le langage engendré : automate minimal.
- ⇒ Propriétés **décidables** : $L(\mathcal{A}) = \emptyset$? $L(\mathcal{A}) = L(\mathcal{B})$?
 $O(n \log(n))$ si \mathcal{A}, \mathcal{B} déterministes, PSPACE-complet sinon.
- ⇒ **Clôture** par opérations Booléennes $\cap, \cup, A^* \setminus$.
- ⇒ Propriétés **décidables** : $L(\mathcal{A}) \cap L(\mathcal{B}) = \emptyset$? $L(\mathcal{A}) \subseteq L(\mathcal{B})$?

Les automates finis : une notion robuste

Un langage $L \subseteq \Sigma^*$ est **reconnu** par un morphisme de monoïde $\varphi : \Sigma^* \rightarrow M$ si

$$L = \varphi^{-1}(\varphi(L)).$$

Théorème (Kleene)

Les propriétés suivantes sont équivalentes pour un langage $L \subseteq \Sigma^*$:

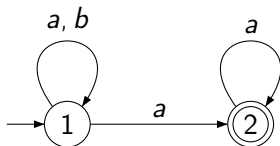
- ▶ L est **accepté par** un **automate fini** sur l'alphabet Σ .
- ▶ L est **reconnu** par un morphisme de Σ^* dans un **monoïde fini**.
- ▶ L est **décrit** par une **expression rationnelle** sur Σ .

Les automates de Büchi

- ▶ On s'intéresse parfois aux comportements infinis de systèmes.
- ▶ Un tel comportement est modélisé par un mot **infini**.
- ▶ Automates de Büchi : automates pour décrire des langages « réguliers » de mots infinis.
- ▶ Formellement, c'est un automate usuel.
- ▶ Critère d'acceptation
 - ▶ **run** infini accepté s'il visite **une infinité de fois** un état acceptant.
 - ▶ **mot** infini accepté s'il **existe** un run accepté sur ce mot.

Propriétés des automates de Büchi

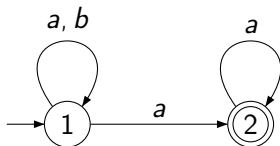
- ▶ Certains automates de Büchi ne peuvent pas être déterminisés.
- ▶ Exemple : $L = \{t \in \{a, b\}^\omega \mid |t|_b < \infty\}$.



Un déterministe aurait des runs acceptés sur mots à 1, 2, ..., ∞ occurrences de b .

Propriétés des automates de Büchi

- ▶ Certains automates de Büchi ne peuvent pas être déterminisés.
- ▶ Exemple : $L = \{t \in \{a, b\}^\omega \mid |t|_b < \infty\}$.



Un déterministe aurait des runs acceptés sur mots à 1, 2, ..., ∞ occurrences de b .

Bonnes propriétés

- ▶ Test du vide en temps linéaire.
- ▶ Intersection, union faciles.
- ▶ Complémentables, mais constructions non triviales.
1^{re} construction : Safra 88, $2^{O(n \log(n))}$ (même borne inf., Michel 88).

Intérêt des automates en informatique

Les automates constituent des objets très simples. Trop ?

- ▶ Ont-ils une réelle **utilité en informatique**? En vérification :
 - ▶ modélisation,
 - ▶ outil technique.
- ▶ Fournissent-ils des **challenges mathématiques**?

B. Langages de spécification et automates

Différents domaines d'application

La vérification automatique

Model-checking : algorithmes pour logiques LTL, FO($<$), MSO

Différents domaines d'application des automates

- ▶ Traitement du texte (recherche de motifs, compression), génome,
- ▶ Compilation,
- ▶ Codage et décodage,
- ▶ Manipulation de corpus de langues naturelles,
- ▶ Théorie du contrôle,
- ▶ Théorie combinatoire des groupes,
- ▶ Modélisation et **vérification de protocoles**, de circuits...
- ▶ ...

Des bugs logiciels aux conséquences désastreuses (1)

Aéronautique

- 1962 Perte d'itinéraire de la sonde Mariner 1 (NASA) au lancement.
Cause. Erreur de transcription de copie papier vers code Fortran.
- 1996 Auto-destruction d'Ariane 5 (1^{er} vol), 37 secondes après décollage.
Cause. Conversion flottant 64 bits trop grand, vers entier 16 bits.
- 2004 Blocage du robot Mars Rover.
Cause. Trop de fichiers ouverts en mémoire flash.

Médecine

- 85–87 5 morts par irradiations massives dues à la machine Therac-25.
Cause. Conflit d'accès aux ressources entre 2 parties logicielles.

Des bugs logiciels aux conséquences désastreuses (2)

Télécoms

1990 Crash à grande échelle du réseau AT&T, effet domino.

Cause. Toute unité défaillante alertait ses voisines, mais la réception du message d'alerte causait une panne du récepteur !

Énergie

2003 Panne d'électricité aux USA & Canada, General Electric.

Cause. À nouveau : mauvaise gestion d'**accès concurrents** aux ressources dans un programme de surveillance.

Des bugs logiciels aux conséquences désastreuses (3)

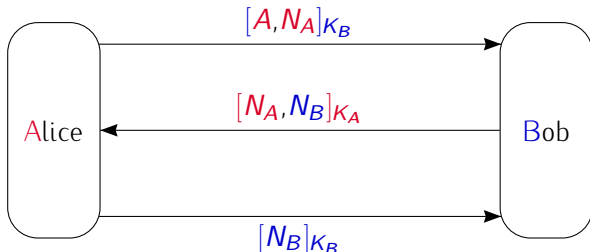
Informatique

- 1994 Bug du Pentium FDIV sur opérations en nombres flottants.
Cause. Algorithme de division erroné (découvert par Th. Nicely).
- 06–08 Clés générées par OpenSSL et données cryptées non sûres, impactant les applications l'utilisant (comme ssh).
Cause. Générateur de nombres aléatoires d'OpenSSL cassé.
- 78–95 Faille dans le protocole d'authentification de Needham-Schroeder.
Cause. Attaque **man in the middle** détectée par G. Lowe.

Un exemple concret : le protocole de Needham-Schroeder

- ▶ **But** du protocole : authentification sur un réseau.
- ▶ **Moyens** : chaque agent A a une paire de « clés » :
 - ▶ K_A , clé publique, connue de tous. Joue le rôle de **cadenas**.
On l'utilise pour coder les messages envoyés à A .
 $[m]_{K_A}$ désigne le message m chiffré par K_A .
 - ▶ K_A^{-1} , clé privée, connue seulement de A . Joue le rôle de **clé**.
- ▶ Chaque partie doit s'assurer de l'identité de l'autre partie,
 - ▶ en transmettant un nombre aléatoire chiffré, appelé **nonce**,
 - ▶ en demandant que le nonce soit décodé et renvoyé.

Protocole de Needham-Schroeder



- ▶ Hypothèse : un message peut être intercepté, mais pas décrypté sans clé privée correspondante. Clés privées sûres.
- ▶ Pour Alice : la seule personne à pouvoir connaître N_A est celui qui possède la clé privée correspondant à K_B , c'est donc Bob.
- ▶ Raisonnement similaire pour Bob.

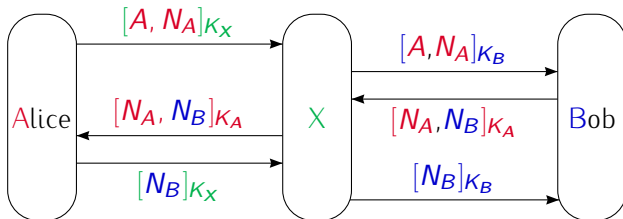
Protocole de Needham-Schroeder : attaque

Mais si Alice utilise le protocole pour parler à X, malhonnête...
... X peut se faire passer pour Alice auprès de Bob.

Protocole de Needham-Schroeder : attaque

Mais si Alice utilise le protocole pour parler à X, malhonnête...

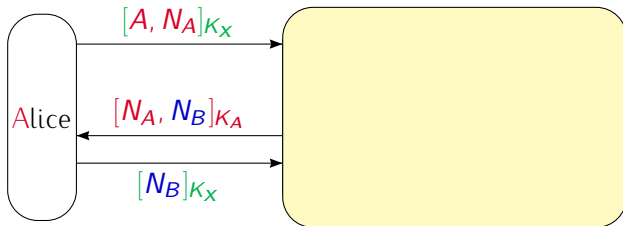
... X peut se faire passer pour Alice auprès de Bob.



Protocole de Needham-Schroeder : attaque

Mais si Alice utilise le protocole pour parler à X, malhonnête...

... X peut se faire passer pour Alice auprès de Bob.

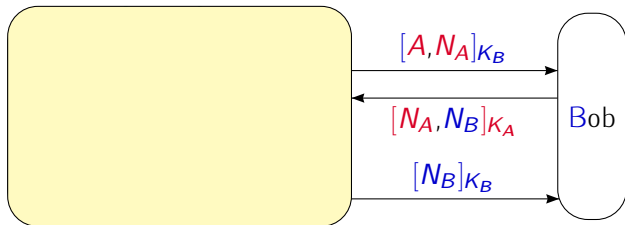


- ▶ Alice et Bob suivent honnêtement le protocole (pas X).
- ▶ Alice parle à X : ok.

Protocole de Needham-Schroeder : attaque

Mais si Alice utilise le protocole pour parler à X, malhonnête...

... X peut se faire passer pour Alice auprès de Bob.

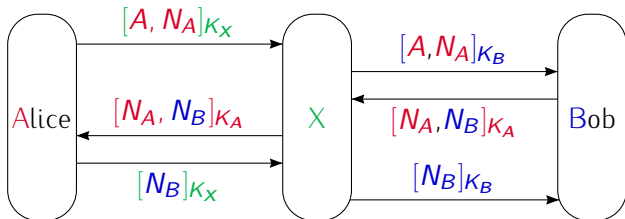


- ▶ Alice et Bob suivent honnêtement le protocole (pas X).
- ▶ Alice parle à X : ok.

Protocole de Needham-Schroeder : attaque

Mais si Alice utilise le protocole pour parler à X, malhonnête...

... X peut se faire passer pour Alice auprès de Bob.



- ▶ Alice et Bob suivent honnêtement le protocole (pas X).
- ▶ Alice parle à X : ok.
- ▶ Mais Bob parle à X croyant parler à Alice (N_B a été révélé à X).
- ▶ Comment corriger simplement ?

Historique du protocole

- 1978 Publié par Needham et Schroeder.
- 1989 « Prouvé » correct par Burrows, Abadi, et Needham.
- 1995 Prouvé erroné par Lowe (17 ans d'utilisation!).
- 1996 Prouvé erroné par Lowe de façon **automatique**, en le modélisant en CSP et en utilisant le logiciel de **model-checking** FDR.

La vérification de logiciel

- ▶ **Constat.** Problèmes logiciels très coûteux, dus à des bugs.
- ▶ Nécessité d'éviter ces erreurs. Approches complémentaires :
 - Simulation/test :
 - ▶ exploration partielle du système.
 - ▶ problème de complétion de la procédure (exploration de **certain**s comportements du système, basée sur des heuristiques),
 - ▶ peut trouver des bugs, mais **pas garantir leur absence**.
 - Preuve de théorème :
 - ▶ pas entièrement automatique,
 - ▶ demande du temps et de l'expertise.
 - Vérification de modèle, ou **model-checking**.
 - Interprétation abstraite.
 - ...

Le model-checking.

Clarke/Emerson & Queille/Sifakis, 1981

- ▶ Objectif : détecter de façon **automatique** les bugs dans les circuits, dans les protocoles de communication,...
- ▶ S'applique bien en phase de conception, ou après modélisation.
- ▶ Travaille sur un modèle de système pour en vérifier des propriétés.



E.M. Clarke



E.A. Emerson



J. Sifakis

Prix Turing 2007.

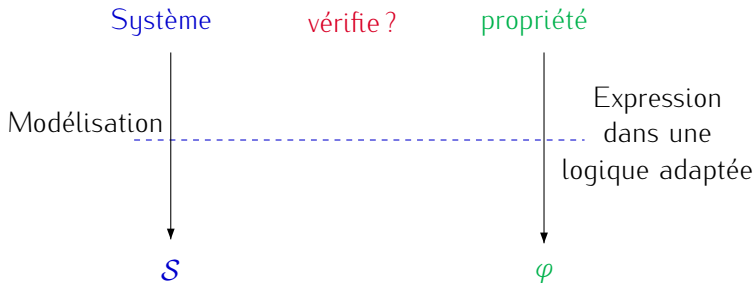
Principe du model-checking

Système

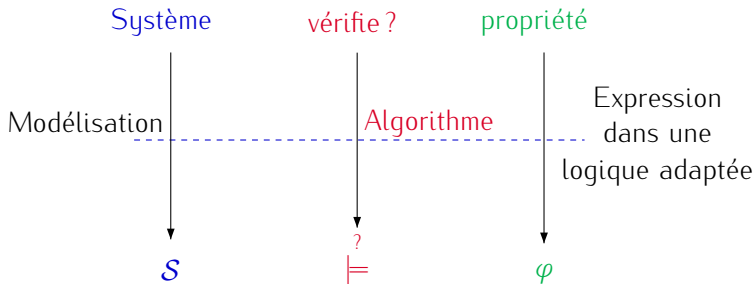
vérifie ?

propriété

Principe du model-checking



Principe du model-checking



L'obstacle du théorème de Rice

- ▶ Problème
 - ▶ Donnée : une machine de Turing M (ou un programme).
 - ▶ Question : la machine M s'arrête-t-elle sur l'entrée vide ?
 - ▶ Ce problème, dit de l'arrêt, est **indécidable**.
- ⇒ On ne peut même pas décider l'**accessibilité** d'un état de contrôle !

Théorème de Rice

Toute propriété non triviale des langages récursivement énumérables est indécidable.

Comment dépasser ces cas d'indécidabilité ?

- ▶ Vérifier des modèles **moins réalistes** que les machines de Turing, en se concentrant sur certains aspects. Aujourd'hui : systèmes finis.
- ▶ Compromis réalisme des modèles / **expressivité des logiques**.
- ▶ Vérifier de façon **approchée**.
 - ▶ Vérifier à nombre de pas de calcul borné.
 - ▶ Semi-algorithmes, plus de garantie de terminaison.
 - ▶ ...

Le model-checking : Avantages

- ▶ Complètement **automatique**.
- ▶ **Assure la correction** d'un modèle vis-à-vis d'une propriété.
Pas de comportement « oublié ».
- ▶ Détection d'**exécution fautive** en cas de propriété non satisfaite.
- ▶ Les propriétés sont exprimées dans des logiques temporelles, permettant d'exprimer **facilement** de nombreuses propriétés.
- ▶ La méthode se généralise à **plusieurs types de systèmes**.

Le model-checking : Inconvénients

- ▶ Travaille à partir d'un modèle, pas directement du programme.
- ▶ **Explosion** du nombre de configurations à parcourir pour vérifier.
 - ↪ techniques symboliques, ou réduction de l'espace d'états.
 - ▶ Techniques symboliques : ne pas représenter explicitement chaque état, mais calculer sur des représentations compactes.
 - ▶ Techniques de réductions : identifier des chemins qu'il suffit d'explorer pour garantir la vérification complète de la propriété.

Vérifier des systèmes finis : Structures de Kripke

- ▶ Structure de Kripke \mathcal{K} : automate fini dont les états sont étiquetés par des propriétés d'un alphabet AP (et sans état final).
- ▶ Permet de modéliser des systèmes finis.
- ▶ Chaque run infini engendre un mot infini sur l'alphabet $\Sigma = 2^{AP}$
 $\rightsquigarrow L(\mathcal{K}) \subseteq \Sigma^\omega$.

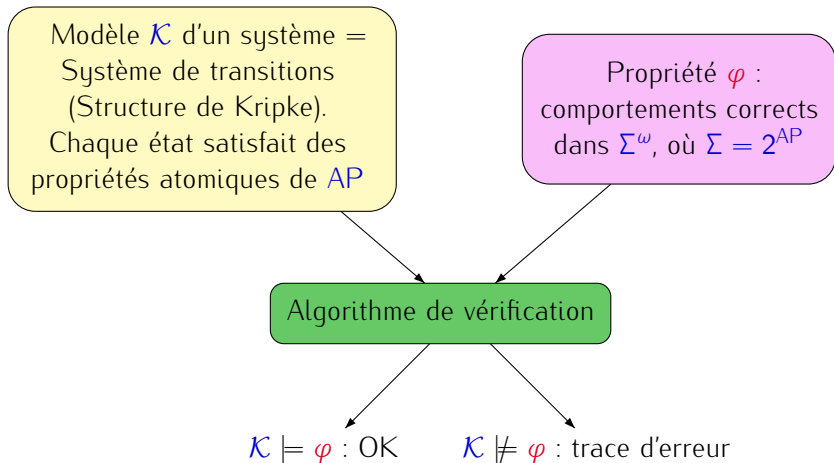
Vérifier des systèmes finis : Structures de Kripke

- ▶ **Structure de Kripke** \mathcal{K} : automate fini dont les **états** sont étiquetés par des propriétés d'un alphabet AP (et sans état final).
- ▶ Permet de modéliser des systèmes **finis**.
- ▶ Chaque run infini engendre un mot infini sur l'alphabet $\Sigma = 2^{AP}$
 $\rightsquigarrow L(\mathcal{K}) \subseteq \Sigma^\omega$.

- ▶ On utilise ensuite un **langage logique** pour spécifier.
- ▶ Une formule φ définit un langage $L(\varphi)$ sur $\Sigma = 2^{AP}$.
- ▶ On veut vérifier si **tout** comportement du système \mathcal{K} satisfait φ :

$$L(\mathcal{K}) \subseteq L(\varphi) ?$$

Le model-checking : schéma



Exemple : algorithme de Peterson

Deux processus P_0, P_1 . Variables $req[0], req[1]$ et $tour$ partagées.

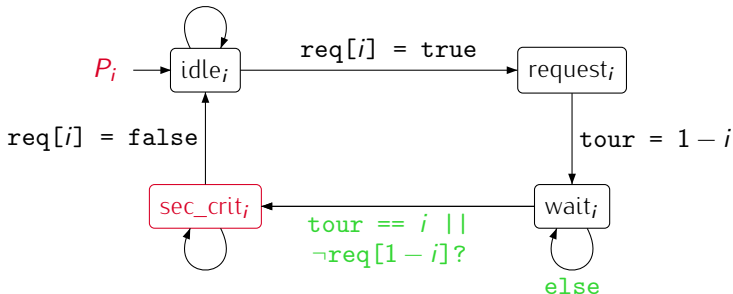
Code pour P_i :

```
req[i] = true
tour = 1-i
while (req[1-i] && tour == 1-i)
    ; // attente
section_critique()
req[i] = false
```

- ▶ Garantit plusieurs propriétés, en particulier **l'exclusion mutuelle**.

Algorithme de Peterson : modélisation

- ▶ 4 états de contrôle par processus et 3 variables Booléennes
⇒ $4^2 \times 2^3 = 128$ états au plus pour la structure de Kripke.
- ▶ Seuls ~ 30 sont accessibles mais protocole non immédiat à prouver.



- ▶ Autres algorithmes MutEx : <http://en.wikipedia.org/wiki/Mutex>
- ▶ Exemple : Dekker, ~ 10 lignes de code, ~ 140 états accessibles.

Le model-checking en pratique

- ▶ Algorithme construisant à partir d'une formule φ un automate \mathcal{A}_φ reconnaissant les comportements (mots infinis) satisfaisant φ :

$$\forall t \in \Sigma^\omega : t \models \varphi \iff t \in L(\mathcal{A}_\varphi).$$

- ▶ Vérifier que tous les comportements de \mathcal{K} satisfont φ :

$$L(\mathcal{K}) \cap L(\mathcal{A}_{\neg\varphi}) = \emptyset \iff \mathcal{K} \models \varphi.$$

Problèmes centraux : Satisfaisabilité et model checking

Étant donné

- ▶ un univers de comportements \mathbb{C} ,
- ▶ une classe de modèles \mathbb{M} ,
- ▶ un langage de spécification \mathbb{L} .

SAT

Donnée Une formule $\varphi \in \mathbb{L}$.

Problème Existe-t-il $t \in \mathbb{C}$ tel que $t \models \varphi$?

MC

Donnée Une formule $\varphi \in \mathbb{L}$,
Un modèle $M \in \mathbb{M}$ avec des comportements $\mathcal{C}(M)$.

Problème Est-il vrai que $t \models \varphi$ pour tout $t \in \mathcal{C}(M)$?

Satisfaisabilité & model checking : réductions

Réduction $SAT \leq MC$

S'il existe un modèle universel MU dans \mathbb{M} : $\mathcal{C}(MU) = \mathbb{C}$.

$$\varphi \text{ satisfaisable} \iff MU \models \neg\varphi$$

Satisfaisabilité & model checking : réductions

Réduction $SAT \leq MC$

S'il existe un modèle universel MU dans \mathbb{M} : $\mathcal{C}(MU) = \mathbb{C}$.

$$\varphi \text{ satisfaisable} \iff MU \models \neg\varphi$$

Réduction $MC \leq SAT$

Si tout modèle peut être effectivement encodé par une formule, i.e. :

$$M \rightsquigarrow \varphi_M : \forall t \in \mathbb{C}, t \models \varphi_M \iff t \in \mathcal{C}(M)$$

$$\begin{aligned} M \models \varphi &\iff (\varphi_M \Rightarrow \varphi) \text{ est une tautologie} \\ &\iff (\varphi_M \wedge \neg\varphi) \text{ n'est pas satisfaisable.} \end{aligned}$$

Réduction polynomiale si φ_M calculée en temps polynomial en $|M|$.

Spécifications : les propriétés typiques

Spécifications : toutes les exécutions du système satisfont...

- ▶ Sûreté : Toutes les exécutions restent dans de bons états.

Spécifications : les propriétés typiques

Spécifications : toutes les exécutions du système satisfont...

- ▶ Sûreté : Toutes les exécutions restent dans de bons états.
- ▶ MutEx : Sûreté d'un état global.

Spécifications : les propriétés typiques

Spécifications : toutes les exécutions du système satisfont...

- ▶ Sûreté : Toutes les exécutions restent dans de bons états.
- ▶ MutEx : Sûreté d'un état global.
- ▶ Vivacité : Le processus p est activé infiniment souvent.

Spécifications : les propriétés typiques

Spécifications : toutes les exécutions du système satisfont...

- ▶ Sûreté : Toutes les exécutions restent dans de bons états.
- ▶ MutEx : Sûreté d'un état global.
- ▶ Vivacité : Le processus p est activé infiniment souvent.
- ▶ Réponse : Chaque requête recevra une réponse dans le futur.

Spécifications : les propriétés typiques

Spécifications : toutes les exécutions du système satisfont...

- ▶ Sûreté : Toutes les exécutions restent dans de bons états.
- ▶ MutEx : Sûreté d'un état global.
- ▶ Vivacité : Le processus p est activé infiniment souvent.
- ▶ Réponse : Chaque requête recevra une réponse dans le futur.
- ▶ Réponse' : Idem + entre 2 requêtes, il y a une réponse.

Spécifications : les propriétés typiques

Spécifications : toutes les exécutions du système satisfont...

- ▶ Sûreté : Toutes les exécutions restent dans de bons états.
- ▶ MutEx : Sûreté d'un état global.
- ▶ Vivacité : Le processus p est activé infiniment souvent.
- ▶ Réponse : Chaque requête recevra une réponse dans le futur.
- ▶ Réponse' : Idem + entre 2 requêtes, il y a une réponse.
- ▶ Release : L'alarme reste active tant qu'elle n'est pas éteinte.

Langages de spécifications et critères

Formalismes logiques pour spécifier des propriétés de systèmes.

Critères pratiques

- ▶ Est-il facile d'écrire une spécification ?
- ▶ Est-il facile de lire une spécification ?

Critères théoriques

- ▶ Les problèmes SAT et MC sont-ils décidables ?
- ▶ Complexité.
- ▶ Pouvoir d'expression.
- ▶ Concision.

La Logique Temporelle Linéaire LTL (Pnueli 1977)

- ▶ Formules construites à l'aide de propositions atomiques ($p \in AP$), connecteurs Booléens, et **modalités temporelles**.
- ▶ Les modèles des formules sont les mots infinis sur Σ , où $\Sigma = 2^{AP}$.
- ▶ Les formules sont **interprétées** aux positions du mot.

$$t, x \models \varphi \text{ (mot } t, \text{ position } x)$$

- ▶ Une formule définit un langage : $L(\varphi) = \{t \in \Sigma^\omega \mid t, 0 \models \varphi\}$.

LTL : syntaxe et sémantique

Syntaxe : LTL(AP, X, U)

$$\varphi ::= \perp \mid p \ (p \in AP) \mid \neg\varphi \mid \varphi \vee \varphi \mid X\varphi \mid \varphi U \varphi$$

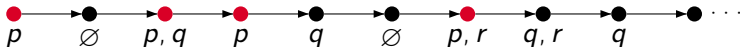
LTL : syntaxe et sémantique

Syntaxe : LTL(AP, X, U)

$$\varphi ::= \perp \mid p \ (p \in AP) \mid \neg\varphi \mid \varphi \vee \varphi \mid X\varphi \mid \varphi U \varphi$$

Sémantique : $t = [\mathbb{N}, \leq, \lambda]$ avec $\lambda : \mathbb{N} \rightarrow \Sigma = 2^{AP}$ et $x \in \mathbb{N}$.

$t, x \models p$ si $p \in \lambda(x)$



LTL : syntaxe et sémantique

Syntaxe : LTL(AP, X, U)

$$\varphi ::= \perp \mid p \ (p \in \text{AP}) \mid \neg\varphi \mid \varphi \vee \psi \mid X\varphi \mid \varphi \text{ U } \psi$$

Sémantique : $t = [\mathbb{N}, \leq, \lambda]$ avec $\lambda : \mathbb{N} \rightarrow \Sigma = 2^{\text{AP}}$ et $x \in \mathbb{N}$.

$t, x \models p$ si $p \in \lambda(x)$

$t, x \models \neg\varphi$ si $t, x \not\models \varphi$

$t, x \models \varphi \vee \psi$ si $t, x \models \varphi \vee t, x \models \psi$

LTL : syntaxe et sémantique

Syntaxe : LTL(AP, X, U)

$\varphi ::= \perp \mid p \ (p \in AP) \mid \neg\varphi \mid \varphi \vee \psi \mid X\varphi \mid \varphi U \psi$

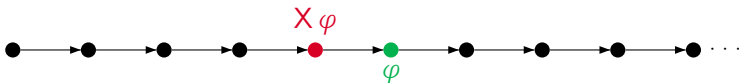
Sémantique : $t = [\mathbb{N}, \leq, \lambda]$ avec $\lambda : \mathbb{N} \rightarrow \Sigma = 2^{AP}$ et $x \in \mathbb{N}$.

$t, x \models p$ si $p \in \lambda(x)$

$t, x \models \neg\varphi$ si $t, x \not\models \varphi$

$t, x \models \varphi \vee \psi$ si $t, x \models \varphi \vee t, x \models \psi$

$t, x \models X\varphi$ si $\exists y. x < y \wedge t, y \models \varphi$



LTL : syntaxe et sémantique

Syntaxe : LTL(AP, X, U)

$$\varphi ::= \perp \mid p \ (p \in AP) \mid \neg\varphi \mid \varphi \vee \psi \mid X\varphi \mid \varphi U \psi$$

Sémantique : $t = [\mathbb{N}, \leq, \lambda]$ avec $\lambda : \mathbb{N} \rightarrow \Sigma = 2^{AP}$ et $x \in \mathbb{N}$.

$t, x \models p$ si $p \in \lambda(x)$

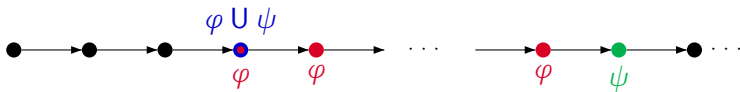
$t, x \models \neg\varphi$ si $t, x \not\models \varphi$

$t, x \models \varphi \vee \psi$ si $t, x \models \varphi \vee t, x \models \psi$

$t, x \models X\varphi$ si $\exists y. x < y \wedge t, y \models \varphi$

$t, x \models \varphi U \psi$ si $\exists z. x \leq z \wedge t, z \models \psi \wedge$

$\forall y. (x \leq y < z) \Rightarrow t, y \models \varphi$



LTL : syntaxe et sémantique

Syntaxe : LTL(AP, X, U)

$$\varphi ::= \perp \mid p \ (p \in AP) \mid \neg\varphi \mid \varphi \vee \psi \mid X\varphi \mid \varphi U \psi$$

Sémantique : $t = [\mathbb{N}, \leq, \lambda]$ avec $\lambda : \mathbb{N} \rightarrow \Sigma = 2^{AP}$ et $x \in \mathbb{N}$.

$$t, x \models p \quad \text{si} \quad p \in \lambda(x)$$

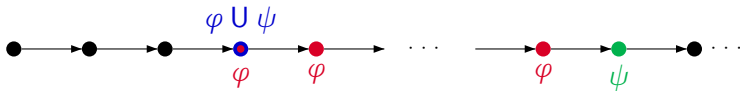
$$t, x \models \neg\varphi \quad \text{si} \quad t, x \not\models \varphi$$

$$t, x \models \varphi \vee \psi \quad \text{si} \quad t, x \models \varphi \vee t, x \models \psi$$

$$t, x \models X\varphi \quad \text{si} \quad \exists y. x < y \wedge t, y \models \varphi$$

$$t, x \models \varphi U \psi \quad \text{si} \quad \exists z. x \leq z \wedge t, z \models \psi \wedge$$

$$\forall y. (x \leq y < z) \Rightarrow t, y \models \varphi$$



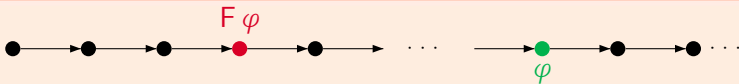
$\varphi \in \text{LTL}(AP, X, U)$ définit le langage $L(\varphi) = \{t \mid t, 0 \models \varphi\}$.

Exprimer des propriétés en LTL

Macros

- ▶ $F\varphi = T U \varphi$ (un jour dans le futur).

Exemple

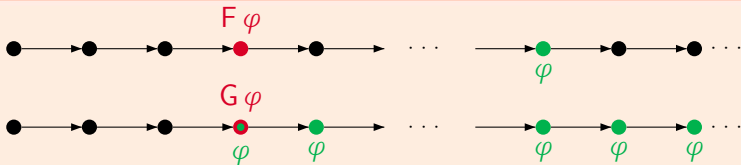


Exprimer des propriétés en LTL

Macros

- ▶ $F\varphi = \top U \varphi$ (un jour dans le futur).
- ▶ $G\varphi = \neg F\neg\varphi$ (toujours dans le futur).

Exemple



Expression de propriétés en LTL

Specifications :

- ▶ Sûreté : Gok

Expression de propriétés en LTL

Specifications :

- ▶ Sûreté : $G\ ok$
- ▶ MutEx : $\neg F(\text{crit}_1 \wedge \text{crit}_2)$

Expression de propriétés en LTL

Specifications :

- ▶ Sûreté : $G ok$
- ▶ MutEx : $\neg F(crit_1 \wedge crit_2)$
- ▶ Vivacité : $GF \text{ actif}$

Expression de propriétés en LTL

Specifications :

- ▶ Sûreté : $G\ ok$
- ▶ MutEx : $\neg F(\text{crit}_1 \wedge \text{crit}_2)$
- ▶ Vivacité : $G F\ \text{actif}$
- ▶ Réponse : $G(\text{requête} \Rightarrow F\ \text{réponse})$

Expression de propriétés en LTL

Specifications :

- ▶ Sûreté : $G ok$
- ▶ MutEx : $\neg F(crit_1 \wedge crit_2)$
- ▶ Vivacité : $G F \text{ actif}$
- ▶ Réponse : $G(\text{requête} \Rightarrow F \text{ réponse})$
- ▶ Réponse' : $G(\text{requête} \Rightarrow X(\neg \text{requête} U \text{ réponse}))$

Expression de propriétés en LTL

Specifications :

- ▶ Sûreté : $G ok$
- ▶ MutEx : $\neg F(crit_1 \wedge crit_2)$
- ▶ Vivacité : $G F \text{ actif}$
- ▶ Réponse : $G(\text{requête} \Rightarrow F \text{ réponse})$
- ▶ Réponse' : $G(\text{requête} \Rightarrow X(\neg \text{requête} U \text{ réponse}))$
- ▶ Release : $\text{off } R \text{ alarme}$

Expression de propriétés en LTL

Specifications :

- ▶ Sûreté : $G \text{ ok}$
- ▶ MutEx : $\neg F(\text{crit}_1 \wedge \text{crit}_2)$
- ▶ Vivacité : $G F \text{ actif}$
- ▶ Réponse : $G(\text{requête} \Rightarrow F \text{ réponse})$
- ▶ Réponse' : $G(\text{requête} \Rightarrow X(\neg \text{requête} U \text{ réponse}))$
- ▶ Release : $\text{off R alarme} = (\text{alarme} U (\text{alarme} \wedge \text{off})) \vee (G \text{ alarme})$

Expression de propriétés en LTL

Specifications :

- ▶ Sûreté : $G ok$
- ▶ MutEx : $\neg F(crit_1 \wedge crit_2)$
- ▶ Vivacité : $GF \text{ actif}$
- ▶ Réponse : $G(\text{requête} \Rightarrow F \text{ réponse})$
- ▶ Réponse' : $G(\text{requête} \Rightarrow X(\neg \text{requête} U \text{ réponse}))$
- ▶ Release : $\text{off R alarme} = (\text{alarme} U (\text{alarme} \wedge \text{off})) \vee (G \text{ alarme})$

Release :

- ▶ $\varphi R \psi = (\psi U (\varphi \wedge \psi)) \vee (G \psi) = \neg(\neg\varphi U \neg\psi)$

LTL avec passé

Sémantique : $t = [\mathbb{N}, \leq, \lambda]$ avec $\lambda : \mathbb{N} \rightarrow \Sigma = 2^{\text{AP}}$ et $x \in \mathbb{N}$

$t, x \models Y\varphi$ si $\exists y. y < x \wedge t, y \models \varphi$

Exemple



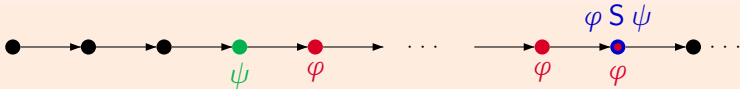
LTL avec passé

Sémantique : $t = [\mathbb{N}, \leq, \lambda]$ avec $\lambda : \mathbb{N} \rightarrow \Sigma = 2^{\text{AP}}$ et $x \in \mathbb{N}$

$t, x \models \Upsilon \varphi$ si $\exists y. y \leq x \wedge t, y \models \varphi$

$t, x \models \varphi S \psi$ si $\exists z. z \leq x \wedge t, z \models \psi \wedge$
 $\forall y. (z < y \leq x) \Rightarrow t, y \models \varphi$

Exemple



LTL avec passé

Sémantique : $t = [\mathbb{N}, \leq, \lambda]$ avec $\lambda : \mathbb{N} \rightarrow \Sigma = 2^{\text{AP}}$ et $x \in \mathbb{N}$

$t, x \models Y \varphi$ si $\exists y. y < x \wedge t, y \models \varphi$

$t, x \models \varphi S \psi$ si $\exists z. z \leq x \wedge t, z \models \psi \wedge$
 $\forall y. (z < y \leq x) \Rightarrow t, y \models \varphi$

Exemple



LTL versus PLTL

$G(\text{rép} \Rightarrow Y(\neg \text{rép} S \text{rq}))$

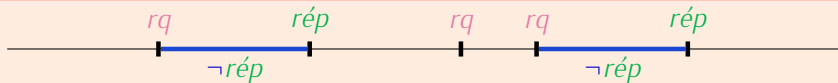
LTL avec passé

Sémantique : $t = [\mathbb{N}, \leq, \lambda]$ avec $\lambda : \mathbb{N} \rightarrow \Sigma = 2^{\text{AP}}$ et $x \in \mathbb{N}$

$t, x \models \mathbf{Y} \varphi$ si $\exists y. y < x \wedge t, y \models \varphi$

$t, x \models \varphi \mathbf{S} \psi$ si $\exists z. z \leq x \wedge t, z \models \psi \wedge$
 $\forall y. (z < y \leq x) \Rightarrow t, y \models \varphi$

Exemple



LTL versus PLTL

$G(\text{rép} \Rightarrow \mathbf{Y}(\neg \text{rép} \mathbf{S} \text{rq}))$

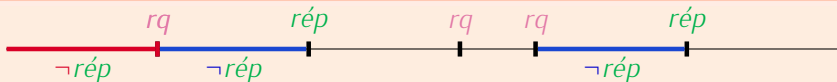
LTL avec passé

Sémantique : $t = [\mathbb{N}, \leq, \lambda]$ avec $\lambda : \mathbb{N} \rightarrow \Sigma = 2^{\text{AP}}$ et $x \in \mathbb{N}$

$t, x \models \mathbf{Y} \varphi$ si $\exists y. y < x \wedge t, y \models \varphi$

$t, x \models \varphi \mathbf{S} \psi$ si $\exists z. z \leq x \wedge t, z \models \psi \wedge$
 $\forall y. (z < y \leq x) \Rightarrow t, y \models \varphi$

Exemple



LTL versus PLTL

$G(\text{rép} \Rightarrow \mathbf{Y}(\neg \text{rép} \mathbf{S} \text{rq})) = (\text{rq} \mathbf{R} \neg \text{rép}) \wedge$

LTL avec passé

Sémantique : $t = [\mathbb{N}, \leq, \lambda]$ avec $\lambda : \mathbb{N} \rightarrow \Sigma = 2^{\text{AP}}$ et $x \in \mathbb{N}$

$t, x \models \mathbf{Y} \varphi$ si $\exists y. y < x \wedge t, y \models \varphi$

$t, x \models \varphi \mathbf{S} \psi$ si $\exists z. z \leq x \wedge t, z \models \psi \wedge$
 $\forall y. (z < y \leq x) \Rightarrow t, y \models \varphi$

Exemple



LTL versus PLTL

$G(\text{rép} \Rightarrow \mathbf{Y}(\neg\text{rép} \mathbf{S} \text{rq})) = (\text{rq} \mathbf{R} \neg\text{rép}) \wedge G(\text{rép} \Rightarrow (\text{rq} \vee \mathbf{X}(\text{rq} \mathbf{R} \neg\text{rép})))$

LTL avec passé

Théorème (Gabbay et al. 1980 / Laroussinie & Schnoebelen 2002)

- ▶ PLTL ne permet pas d'exprimer plus de propriétés que LTL.
- ▶ PLTL peut être exponentiellement plus succincte que LTL.

Compilation LTL vers automates

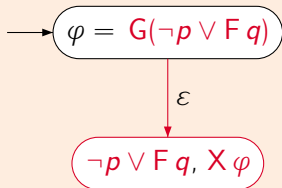
Exemple : $\varphi = G(p \Rightarrow F q)$

→ $\varphi = G(\neg p \vee F q)$

État = ensemble d'obligations.

Compilation LTL vers automates

Exemple : $\varphi = G(p \Rightarrow F q)$

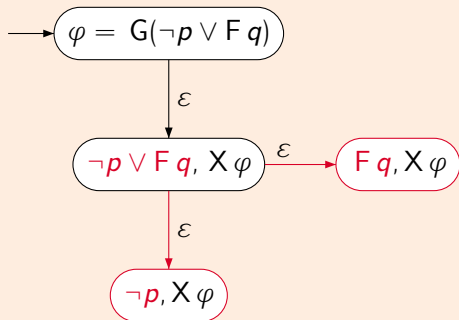


État = ensemble d'obligations.

Saturation = réduire les obligations à des littéraux ou des $X \psi$.

Compilation LTL vers automates

Exemple : $\varphi = G(p \Rightarrow F q)$

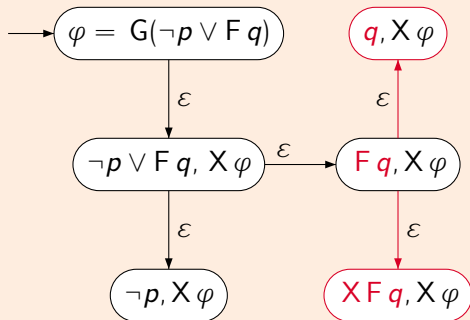


État = ensemble d'obligations.

Saturation = réduire les obligations à des littéraux ou des $X \psi$.

Compilation LTL vers automates

Exemple : $\varphi = G(p \Rightarrow F q)$

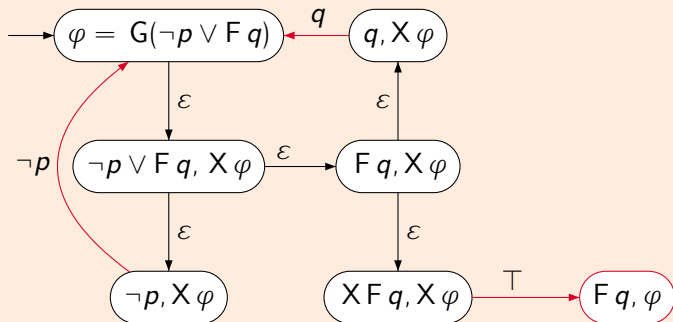


État = ensemble d'obligations.

Saturation = réduire les obligations à des littéraux ou des $X \psi$.

Compilation LTL vers automates

Exemple : $\varphi = G(p \Rightarrow F q)$



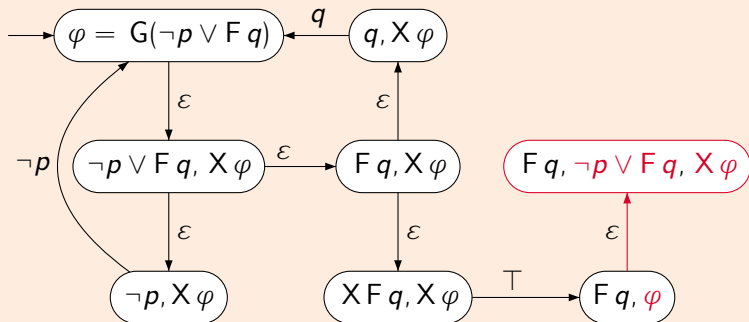
État = ensemble d'obligations.

Saturation = réduire les obligations à des littéraux ou des $X \psi$.

Transition = vérifier les littéraux et propager les $X \psi$ en ψ .

Compilation LTL vers automates

Exemple : $\varphi = G(p \Rightarrow F q)$



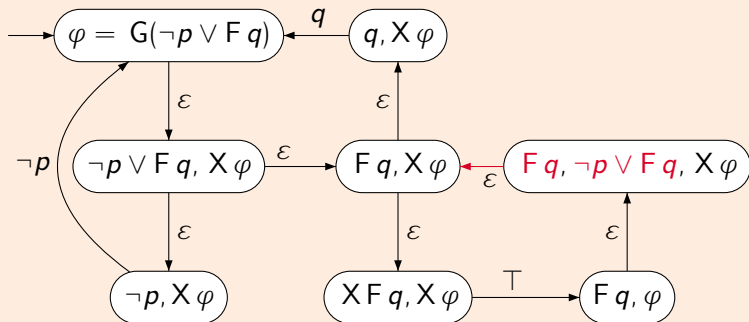
État = ensemble d'obligations.

Saturation = réduire les obligations à des littéraux ou des $X \psi$.

Transition = vérifier les littéraux et propager les $X \psi$ en ψ .

Compilation LTL vers automates

Exemple : $\varphi = G(p \Rightarrow F q)$



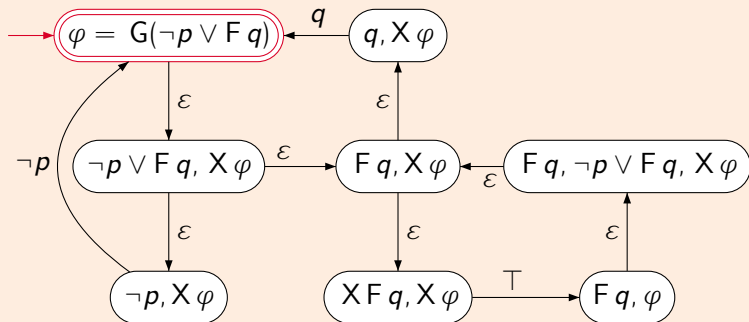
État = ensemble d'obligations.

Saturation = réduire les obligations à des littéraux ou des $X \psi$.

Transition = vérifier les littéraux et propager les $X \psi$ en ψ .

Compilation LTL vers automates

Exemple : $\varphi = G(p \Rightarrow F q)$



État = ensemble d'obligations.

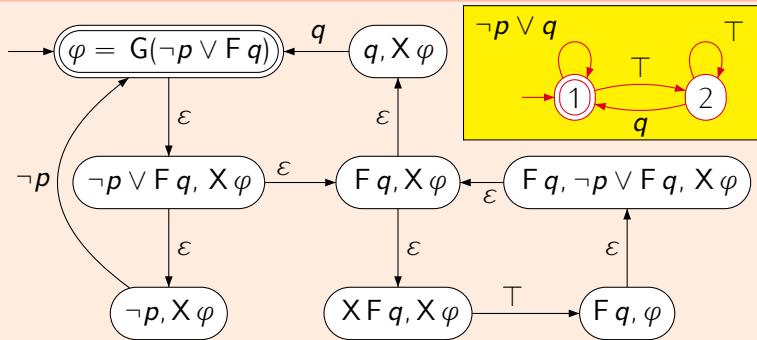
Saturation = réduire les obligations à des littéraux ou des $X \psi$.

Transition = vérifier les littéraux et propager les $X \psi$ en ψ .

Condition d'acceptation adaptée.

Compilation LTL vers automates

Exemple : $\varphi = G(p \Rightarrow F q)$



État = ensemble d'obligations.

Saturation = réduire les obligations à des littéraux ou des $X \psi$.

Transition = vérifier les littéraux et propager les $X \psi$ en ψ .

Condition d'acceptation adaptée.

LTL — Résultats

Décidabilité et complexité (Vardi & Wolper, Sistla 1985)

- ▶ Le problème de satisfaisabilité pour LTL est **décidable**.
 - ▶ Ce problème est **PSPACE**-complet.
-
- ▶ On peut montrer qu'on ne peut pas exprimer le langage $(aa)^+$.
 - ▶ Quelle est l'**expressivité** exacte?
Quels sont les langages du type $L(\varphi)$ pour une formule φ de LTL?

Logique du premier ordre FO(\langle)

Syntaxe : $\text{FO}_\Sigma(\langle)$ ($\Sigma = 2^{\text{AP}}$)

$\varphi ::= \perp \mid P_q(x) \mid \neg\varphi \mid \varphi \vee \varphi \mid x < y \mid \exists x\varphi$ ($q \in \text{AP}, x, y \in \text{Var}$)

Sémantique

- ▶ Une formule est évaluée sur un mot, vu comme un ordre étiqueté (ex. $t = [\mathbb{N}, \leq, \lambda]$ avec $\lambda : \mathbb{N} \rightarrow \Sigma = 2^{\text{AP}}$ pour un mot **infini**).
- ▶ $\sigma : \text{Var} \rightarrow \mathbb{N}$ est une interprétation des variables libres.

Logique du premier ordre FO(\langle)

Syntaxe : $\text{FO}_\Sigma(\langle)$ ($\Sigma = 2^{\text{AP}}$)

$\varphi ::= \perp \mid P_q(x) \mid \neg\varphi \mid \varphi \vee \varphi \mid x < y \mid \exists x\varphi$ ($q \in \text{AP}, x, y \in \text{Var}$)

Sémantique

- ▶ Une formule est évaluée sur un mot, vu comme un ordre étiqueté (ex. $t = [\mathbb{N}, \leq, \lambda]$ avec $\lambda : \mathbb{N} \rightarrow \Sigma = 2^{\text{AP}}$ pour un mot *infini*).
- ▶ $\sigma : \text{Var} \rightarrow \mathbb{N}$ est une interprétation des variables libres.

$$t, \sigma \models P_q(x) \quad \text{si} \quad q \in \lambda(\sigma(x))$$

Logique du premier ordre FO(\langle)

Syntaxe : FO $_{\Sigma}(\langle$) ($\Sigma = 2^{\text{AP}}$)

$\varphi ::= \perp \mid P_q(x) \mid \neg\varphi \mid \varphi \vee \psi \mid x < y \mid \exists x\varphi$ ($q \in \text{AP}, x, y \in \text{Var}$)

Sémantique

- ▶ Une formule est évaluée sur un mot, vu comme un ordre étiqueté (ex. $t = [\mathbb{N}, \leq, \lambda]$ avec $\lambda : \mathbb{N} \rightarrow \Sigma = 2^{\text{AP}}$ pour un mot **infini**).
- ▶ $\sigma : \text{Var} \rightarrow \mathbb{N}$ est une interprétation des variables libres.

$$t, \sigma \models P_q(x) \quad \text{si} \quad q \in \lambda(\sigma(x))$$

$$t, \sigma \models \neg\varphi \quad \text{si} \quad t, \sigma \not\models \varphi$$

$$t, \sigma \models \varphi \vee \psi \quad \text{si} \quad t, \sigma \models \varphi \vee t, \sigma \models \psi$$

Logique du premier ordre FO(\lt)

Syntaxe : FO $_{\Sigma}(\lt)$ ($\Sigma = 2^{\text{AP}}$)

$\varphi ::= \perp \mid P_q(x) \mid \neg\varphi \mid \varphi \vee \varphi \mid x < y \mid \exists x\varphi$ ($q \in \text{AP}, x, y \in \text{Var}$)

Sémantique

- ▶ Une formule est évaluée sur un mot, vu comme un ordre étiqueté (ex. $t = [\mathbb{N}, \leq, \lambda]$ avec $\lambda : \mathbb{N} \rightarrow \Sigma = 2^{\text{AP}}$ pour un mot **infini**).
- ▶ $\sigma : \text{Var} \rightarrow \mathbb{N}$ est une interprétation des variables libres.

$$t, \sigma \models P_q(x) \quad \text{si} \quad q \in \lambda(\sigma(x))$$

$$t, \sigma \models \neg\varphi \quad \text{si} \quad t, \sigma \not\models \varphi$$

$$t, \sigma \models \varphi \vee \psi \quad \text{si} \quad t, \sigma \models \varphi \vee t, \sigma \models \psi$$

$$t, \sigma \models x < y \quad \text{si} \quad \sigma(x) < \sigma(y)$$

Logique du premier ordre FO(<)

Syntaxe : $FO_{\Sigma}(<)$ ($\Sigma = 2^{AP}$)

$\varphi ::= \perp \mid P_q(x) \mid \neg\varphi \mid \varphi \vee \psi \mid x < y \mid \exists x\varphi$ ($q \in AP, x, y \in Var$)

Sémantique

- ▶ Une formule est évaluée sur un mot, vu comme un ordre étiqueté (ex. $t = [\mathbb{N}, \leq, \lambda]$ avec $\lambda : \mathbb{N} \rightarrow \Sigma = 2^{AP}$ pour un mot **infini**).
- ▶ $\sigma : Var \rightarrow \mathbb{N}$ est une interprétation des variables libres.

$$t, \sigma \models P_q(x) \quad \text{si} \quad q \in \lambda(\sigma(x))$$

$$t, \sigma \models \neg\varphi \quad \text{si} \quad t, \sigma \not\models \varphi$$

$$t, \sigma \models \varphi \vee \psi \quad \text{si} \quad t, \sigma \models \varphi \vee t, \sigma \models \psi$$

$$t, \sigma \models x < y \quad \text{si} \quad \sigma(x) < \sigma(y)$$

$$t, \sigma \models \exists x\varphi \quad \text{si} \quad \exists v \in \mathbb{N} : t, \{\sigma \cup [x \mapsto v]\} \models \varphi$$

Logique du 1er ordre FO

Macros

- ▶ $\forall x \varphi : \neg \exists x \neg \varphi$ $\varphi \wedge \psi : \neg(\neg \varphi \vee \neg \psi)$ $\varphi \Rightarrow \psi : \psi \vee \neg \varphi \dots$
- ▶ $\lambda(x) = a$ où $a \in \Sigma = 2^{AP}$: $\bigwedge_{q \in a} P_q(x) \wedge \bigwedge_{q \notin a} \neg P_q(x)$.
- ▶ Réciproquement, $P_q(x) : \bigvee_{a \in \Sigma} \lambda(x) = a$.

Une formule close $\varphi \in \text{FO}(<)$ définit le langage

$$L(\varphi) = \{t \mid t \models \varphi\}$$

de Σ^ω ou Σ^+ , selon que l'on travaille sur mots infinis ou finis.

Logique du premier ordre FO(\lt) — Exemples

► $\varphi = \forall x (\lambda(x) = a \vee \lambda(x) = b) \wedge \forall y \forall z (y \lt z \Rightarrow [\lambda(y) \neq \lambda(z)])$

bababababa ... $\models \varphi$

abaabaabaaba ... $\not\models \varphi$

Logique du premier ordre FO(\prec) — Exemples

- ▶ $\varphi = \forall x (\lambda(x) = a \vee \lambda(x) = b) \wedge \forall y \forall z (y \prec z) \Rightarrow [\lambda(y) \neq \lambda(z)]$

bababababa ... $\models \varphi$

abaabaabaaba ... $\not\models \varphi$

- ▶ $\prec = < \setminus <^2$ est exprimable en FO $_{\Sigma}(\leq)$:

$$x \prec y : (x < y) \wedge (\neg \exists z, x < z < y)$$

Logique du premier ordre FO(\prec) — Exemples

- ▶ $\varphi = \forall x (\lambda(x) = a \vee \lambda(x) = b) \wedge \forall y \forall z (y \prec z) \Rightarrow [\lambda(y) \neq \lambda(z)]$

bababababa ... $\models \varphi$

abaabaabaaba ... $\not\models \varphi$

- ▶ $\prec = < \setminus <^2$ est exprimable en $\text{FO}_\Sigma(\leq)$:

$$x \prec y : (x < y) \wedge (\neg \exists z, x < z < y)$$

- ▶ Réciproquement, $< = \prec^+$ n'est pas exprimable en $\text{FO}_\Sigma(\prec)$.

Logique du premier ordre FO(\prec) — Exemples

- ▶ $\varphi = \forall x (\lambda(x) = a \vee \lambda(x) = b) \wedge \forall y \forall z (y \prec z) \Rightarrow [\lambda(y) \neq \lambda(z)]$

bababababa ... $\models \varphi$

abaabaabaaba ... $\not\models \varphi$

- ▶ $\prec = < \setminus <^2$ est exprimable en $\text{FO}_\Sigma(\leq)$:

$$x \prec y : (x < y) \wedge (\neg \exists z, x < z < y)$$

- ▶ Réciproquement, $< = \prec^+$ n'est pas exprimable en $\text{FO}_\Sigma(\prec)$.
- ▶ $\min(x) : \neg \exists z, z < x$.

Logique du premier ordre FO(\prec) — Exemples

- ▶ $\varphi = \forall x(\lambda(x) = a \vee \lambda(x) = b) \wedge \forall y \forall z (y \prec z) \Rightarrow [\lambda(y) \neq \lambda(z)]$

bababababa ... $\models \varphi$

abaabaabaaba ... $\not\models \varphi$

- ▶ $\prec = < \setminus <^2$ est exprimable en $\text{FO}_\Sigma(\leq)$:

$$x \prec y : (x < y) \wedge (\neg \exists z, x < z < y)$$

- ▶ Réciproquement, $< = \prec^+$ n'est pas exprimable en $\text{FO}_\Sigma(\prec)$.
- ▶ $\text{min}(x) : \neg \exists z, z < x$.
- ▶ $(\text{min} = a) : \exists x[\lambda(x) = a \wedge \text{min}(x)]$.

Logique du premier ordre FO(\prec) — Exemples

- ▶ $\varphi = \forall x(\lambda(x) = a \vee \lambda(x) = b) \wedge \forall y \forall z (y \prec z) \Rightarrow [\lambda(y) \neq \lambda(z)]$

bababababa ... $\models \varphi$

abaabaabaaba ... $\not\models \varphi$

- ▶ $\prec = < \setminus <^2$ est exprimable en $\text{FO}_\Sigma(\leq)$:

$$x \prec y : (x < y) \wedge (\neg \exists z, x < z < y)$$

- ▶ Réciproquement, $< = \prec^+$ n'est pas exprimable en $\text{FO}_\Sigma(\prec)$.
- ▶ $\min(x) : \neg \exists z, z < x$.
- ▶ $(\min = a) : \exists x[\lambda(x) = a \wedge \min(x)]$.
- ▶ $L(\varphi \wedge \min = \{a\} \wedge \max = \{b\}) = (ab)^+$.

Logique du premier ordre FO(\prec) — Exemples

- ▶ $\varphi = \forall x(\lambda(x) = a \vee \lambda(x) = b) \wedge \forall y \forall z (y \prec z) \Rightarrow [\lambda(y) \neq \lambda(z)]$

bababababa ... $\models \varphi$

abaabaabaaba ... $\not\models \varphi$

- ▶ $\prec = < \setminus <^2$ est exprimable en $\text{FO}_\Sigma(\leq)$:

$$x \prec y : (x < y) \wedge (\neg \exists z, x < z < y)$$

- ▶ Réciproquement, $< = \prec^+$ n'est pas exprimable en $\text{FO}_\Sigma(\prec)$.
- ▶ $\min(x) : \neg \exists z, z < x$.
- ▶ $(\min = a) : \exists x[\lambda(x) = a \wedge \min(x)]$.
- ▶ $L(\varphi \wedge \min = \{a\} \wedge \max = \{b\}) = (ab)^+$.
- ▶ Le langage $(aa)^+$ ne peut pas être exprimé !

Spécifications FO

Spécifications :

- ▶ Sûreté $\neg \exists x \lambda(x) \in \text{Bad}$

Spécifications FO

Spécifications :

- ▶ Sûreté $\neg \exists x \lambda(x) \in \text{Bad}$
- ▶ Vivacité $\exists x \lambda(x) = \text{act} \wedge \forall y[(\lambda(y) = \text{act}) \Rightarrow \exists z(z > y) \wedge (\lambda(z) = \text{act})]$

Spécifications FO

Spécifications :

- ▶ Sûreté $\neg \exists x \lambda(x) \in \text{Bad}$
- ▶ Vivacité $\exists x \lambda(x) = \text{act} \wedge \forall y[(\lambda(y) = \text{act}) \Rightarrow \exists z(z > y) \wedge (\lambda(z) = \text{act})]$
- ▶ Réponse $\forall x \lambda(x) = \text{requête} \Rightarrow \exists y(y > x) \wedge \lambda(y) = \text{réponse}$

Spécifications FO

Spécifications :

- ▶ Sûreté $\neg \exists x \lambda(x) \in \text{Bad}$
- ▶ Vivacité $\exists x \lambda(x) = \text{act} \wedge \forall y[(\lambda(y) = \text{act}) \Rightarrow \exists z(z > y) \wedge (\lambda(z) = \text{act})]$
- ▶ Réponse $\forall x \lambda(x) = \text{requête} \Rightarrow \exists y(y > x) \wedge \lambda(y) = \text{réponse}$
- ▶ Release $\forall x [\text{alarm} \in \lambda(x) \Rightarrow \text{off} \in \lambda(x) \vee \forall y(x < y \Rightarrow \text{alarm} \in \lambda(y))]$

FO(\langle) : satisfaisabilité et model-checking

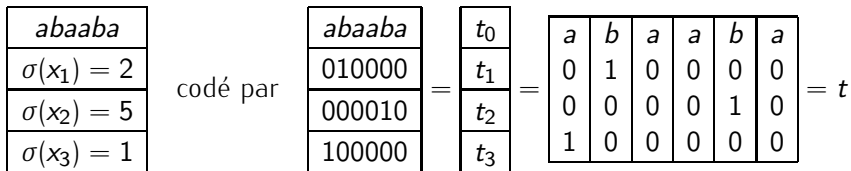
Théorème (Büchi)

Pour toute formule φ de FO(\langle), on peut construire un automate \mathcal{A}_φ tel que $L(\mathcal{A}_\varphi) = \{t \in \Sigma^\omega \mid t \models \varphi\}$.

Idée de preuve. Induction sur φ . Problème : variables libres.

Idée de Büchi : encoder la valuation σ dans le mot.

Soit $\Sigma_\ell = \Sigma \times \{0, 1\}^\ell$. On a $\Sigma_\ell^* \hookrightarrow \Sigma^* \times (\{0, 1\}^*)^\ell = \Sigma^* \times \mathcal{P}(\mathbb{N})^\ell$.



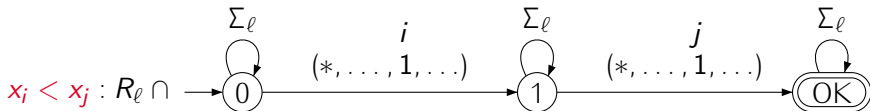
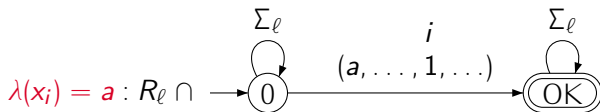
Chaque t_i est dans $0^*10^\omega \rightsquigarrow t \in R_\ell$ reconnu par automate de Büchi.

FO(<), SAT et MC : formules atomiques

Rappel de l'objectif

Soit $\varphi \in \text{FO}(<)$ avec comme variables libres $\text{Var} = \{x_1, \dots, x_\ell\}$. Construire \mathcal{A}_φ sur Σ_ℓ^ω tel que

$$L(\mathcal{A}_\varphi) = \{t \in R_\ell \mid t_0, \sigma \models \varphi\}.$$



FO(\langle), SAT et MC : induction

$$\mathcal{A}_{\varphi \vee \psi} = \mathcal{A}_{\varphi} \cup \mathcal{A}_{\psi}$$

$$\mathcal{A}_{\neg \varphi} = \text{Complément}(\mathcal{A}_{\varphi})$$

(\rightarrow pour mots finis : déterminisation nécessaire)

$$\mathcal{A}_{\exists x_j \varphi} = \text{Projeter la composante } j + 1 \text{ de } \mathcal{A}_{\varphi}$$

(\rightsquigarrow produit un automate non-déterministe)

\implies Tour d'exponentielle de hauteur le nombre d'alternances \neg/\exists .

Logique du premier ordre — Résultats

Décidabilité (Büchi 1960) – Complexité (Stockmeyer 1974)

- ▶ Le problème de satisfaisabilité pour FO(<) est **décidable**.
- ▶ Sa complexité est **non élémentaire**.

$$\text{Élémentaire} = \bigcup_n \text{DTIME} \left(2^{2^{\dots^{2^2}}} \right) \text{ } \left. \vphantom{2^{2^{\dots^{2^2}}}} \right\} n \text{ fois}$$

- ▶ On ne peut pas exprimer le langage $(aa)^+$.
- ▶ Quelle est l'**expressivité** exacte?
Quels sont les langages du type $L(\varphi)$ pour φ formule de FO(<) ?

Logique monadique du second ordre

Syntaxe : $\text{MSO}(<)$ ($\Sigma = 2^{\text{AP}}$)

$\varphi ::= \perp \mid P_q(x) \mid \neg\varphi \mid \varphi \vee \varphi \mid x < y \mid \exists x\varphi \mid x \in X \mid \exists X\varphi$

Sémantique

- ▶ Une formule est évaluée sur un mot vu comme un ordre étiqueté (ex. $t = [\mathbb{N}, <, \lambda]$ pour les mots infinis).
- ▶ Variables du premier ordre $x, y \in \text{Var}_1$.
- ▶ Variables du second ordre $X, Y \in \text{Var}_2$.
- ▶ σ : interprétation des variables libres tq. $\sigma(x) \in \mathbb{N}$ et $\sigma(X) \in 2^{\mathbb{N}}$.

$t, \sigma \models x \in X$ si $\sigma(x) \in \sigma(X)$

$t, \sigma \models \exists X\varphi$ si $t, \sigma \cup [X \mapsto U] \models \varphi$ pour un $U \subseteq V$

Logique monadique du second ordre — Exemples

MSO strictement plus expressive que FO

- ▶ $\leq = \leq^+$ est exprimable en $\text{MSO}(\leq)$.

$$x \leq y : \exists X [(x \in X) \wedge (y \in X) \\ \wedge \forall z \in X (z = x) \vee \exists u (u \in X \wedge u \leq z)]$$

Logique monadique du second ordre — Exemples

MSO strictement plus expressive que FO

- ▶ $\leq = \prec^+$ est exprimable en $\text{MSO}(\prec)$.

$$x \leq y : \exists X [(x \in X) \wedge (y \in X) \\ \wedge \forall z \in X (z = x) \vee \exists u (u \in X \wedge u \prec z)]$$

- ▶ Le langage $(aa)^+$ peut être exprimé par la formule : « toujours a » \wedge

$$\varphi_{\text{Even}} = \exists X \exists Y \left[\forall z ((z \in X) \iff \neg(z \in Y)) \wedge \right. \\ \left. \forall z \forall t (z \prec t) \implies (z \in X \iff t \in Y) \wedge \right. \\ \left. \exists x \exists y x \in X \wedge y \in Y \wedge \neg \exists z [(z < x) \vee (z > y)] \right]$$

Logique monadique du second ordre — Résultats

Décidabilité (Büchi 1960) & complexité

Le problème SAT pour MSO est **décidable** (mais non élémentaire).

Expressivité (Büchi 1960)

Les langages exprimables en $\text{MSO}(<)$ sont les langages rationnels.

Traduction automates vers logique $\text{MSO}(<)$: état $q \rightsquigarrow$ variable X_q codant les positions d'un run où q est atteint.

En résumé

- ▶ Pour les systèmes finis, on peut vérifier des propriétés exprimées en LTL, FO, MSO... grâce aux automates.
- ▶ Des logiciels existent pour traduire des formules en automate, modéliser, et appliquer l'algorithme de model-checking.
 - ▶ **SPIN** (G. Holzmann) <http://www.spinroot.com>. Gère LTL. Permet par exemple de modéliser très simplement les protocoles de Peterson, Dekker, Needham-Schroeder évoqués ici.
 - ▶ **Mona** <http://www.brics.dk/mona/> pour logiques monadiques.
 - ▶ ...
- ▶ Beaucoup de variations **modèles/logiques/algorithmes**.

Extensions

- ▶ Des logiques plus **expressives** ou plus **générales** :
 - ▶ logiques arborescentes.
 - ▶ sur structures plus complexes que les mots (arbres, graphes,...).
- ▶ Des automates plus **réalistes** ou plus **généraux** :
 - ▶ À pile.
 - ▶ D'arbres.
 - ▶ Alternants et/ou boustrophédons.
 - ▶ Probabilistes.
 - ▶ À compteur (deux compteurs : indécidable).
 - ▶ Communicants, avec pertes ou non.
 - ▶ Temporisés : permettent de modéliser le temps (**spécialités LSV**).
 - ▶ ...
- ▶ Ces modèles sont souvent à espace de configurations **infini**.