

# Quelques aspects des automates en informatique

Applications à la vérification & aspects théoriques

Marc Zeitoun

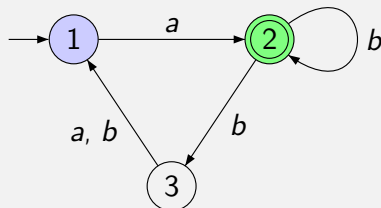
Conférence ENS Cachan, 2/9/2008

## Plan

- A. Rappels et exemples.
- B. Quelques applications des automates (l'utile).
- C. Des problèmes théoriques et des challenges (l'agréable).

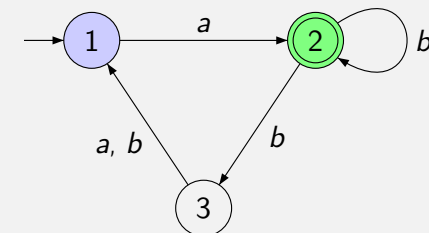
## Automates finis

- ▶ 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.

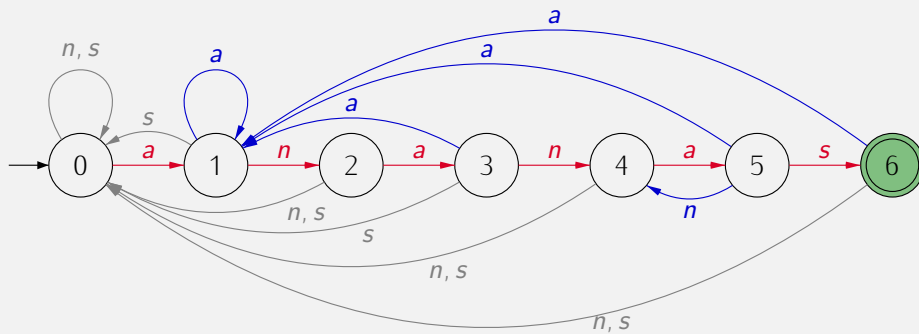
## Runs et langage accepté



- ▶ **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

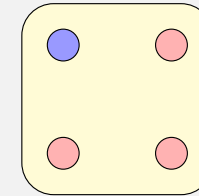
- Pour rechercher le mot **ananas** dans un texte sur  $\{a, n, s\}^*$ , il suffit de lire le texte dans l'automate **déterministe** :



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

## Exemple 2 : barman aveugle

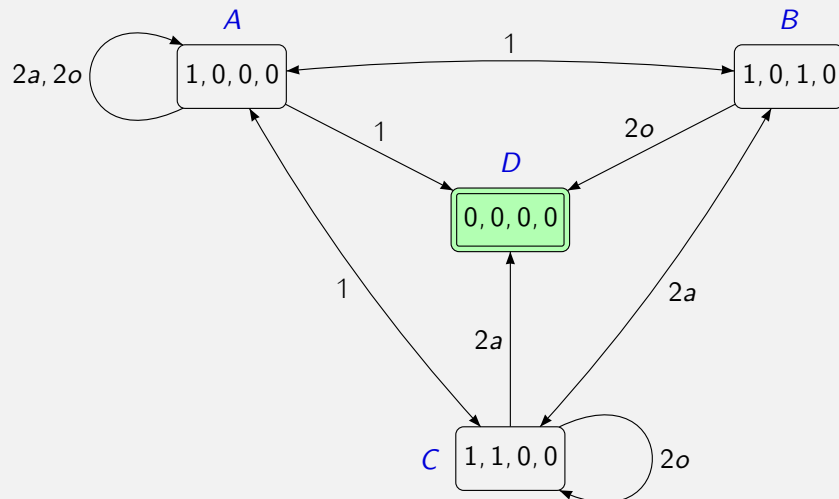
- Un barman aveugle joue au jeu suivant avec un client.
- Il a devant lui un plateau avec 4 verres.



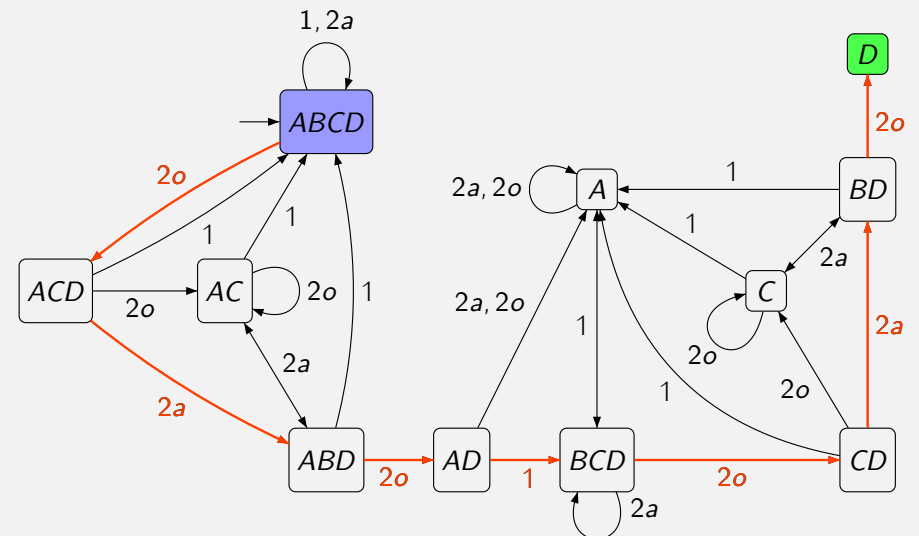
- À chaque tour le barman fait retourner au client 1 ou 2 verres.
- S'il s'agit de 2 verres, il précise s'ils sont **adjacents** ou **opposés**.
- Si tous les verres sont dans le même sens, le barman a gagné.

## 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



- 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.

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

- ▶ 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  $\cap, \cup, A^*$ .
- ⇒ 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  $L \subseteq \Sigma^*$  :

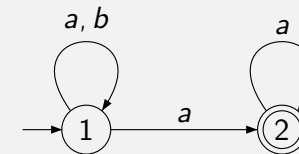
- ▶  $L$  est **accepté** par un **automate fini** sur l'alphabet  $\Sigma$ .
- ▶  $L$  est **reconnu** par un morphisme de  $\Sigma^*$  dans un **monoïde fini**.
- ▶  $L$  est **décrit** par une **expression rationnelle** sur  $\Sigma$ .

## 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, ...,  $\infty$  occurrences de  $b$ .

### Bonnes propriétés

- ▶ Test du vide en temps linéaire.
- ▶ Intersection, union faciles.
- ▶ Complémentables, mais constructions non triviales.  
1<sup>re</sup> 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. Quelques applications des automates

Différents domaines d'application

La vérification automatique

Model-checking : algorithmes pour logiques LTL, FO( $\prec$ ), 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<sup>er</sup> 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)

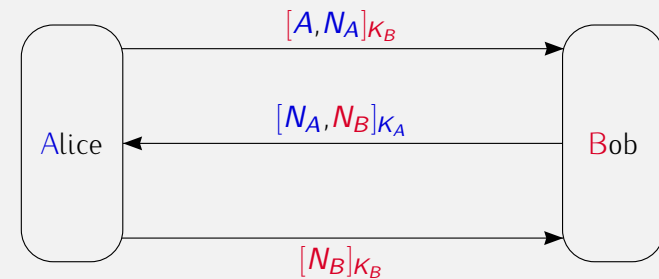
### Informatique

- 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.

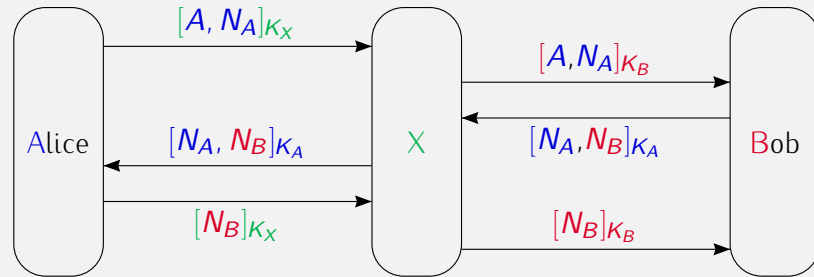
## Protocole de Needham-Schroeder



- ▶ 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**.
- ▶ Raisonement 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.



- ▶ 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 certains 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



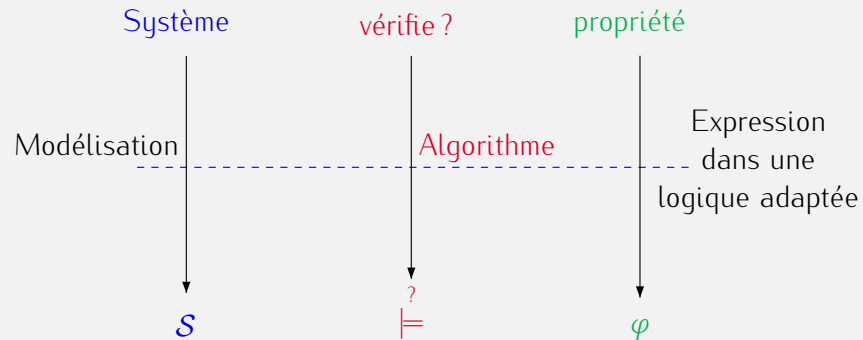
E.A. Emerson



J. Sifakis

Prix Turing 2007.

## Principe du model-checking



## 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

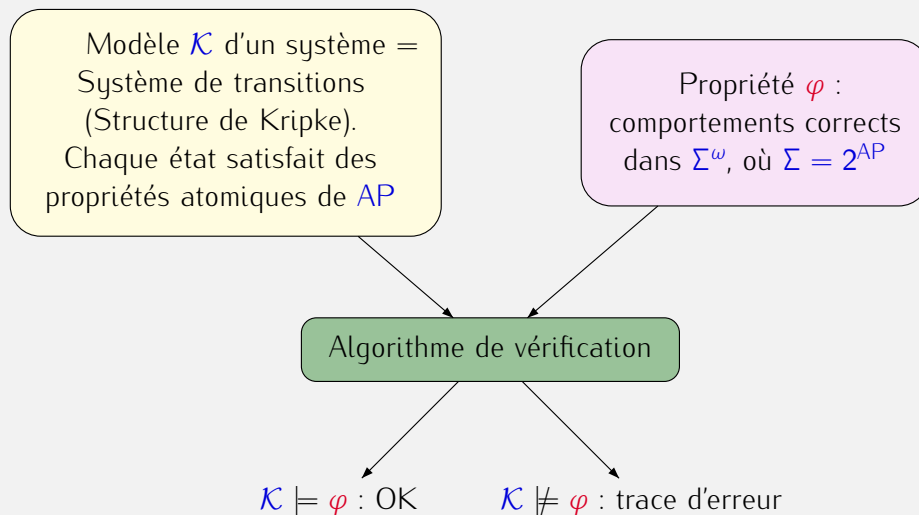
- ▶ Travaille à partir d'un modèle, pas directement du programme.
- ▶ **Explosion** du nombre de configurations à parcourir pour vérifier.
  - ↪ techniques symboliques, ou réduction de l'espace d'états.
    - ▶ Techniques symboliques : ne pas représenter explicitement chaque état, mais calculer sur des 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  $\Sigma = 2^{AP}$ 
  - ↪  $L(\mathcal{K}) \subseteq \Sigma^\omega$ .
- ▶ On utilise ensuite un langage logique pour spécifier.
- ▶ Une formule  $\varphi$  définit un langage  $L(\varphi)$  sur  $\Sigma = 2^{AP}$ .
- ▶ On veut vérifier si tout comportement du système  $\mathcal{K}$  satisfait  $\varphi$  :

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

## Le model-checking : schéma



## Exemple : algorithme de Peterson

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

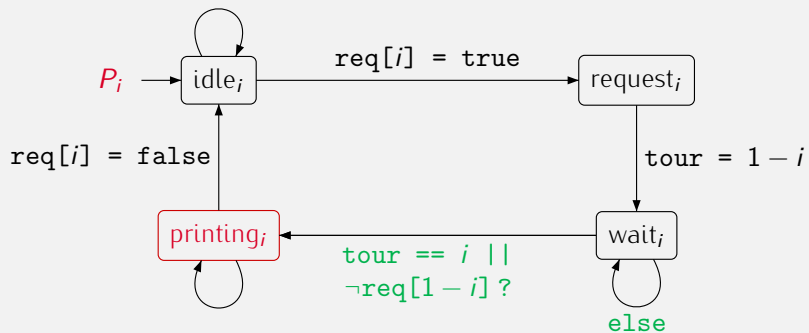
Code pour  $P_i$  :

```
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  
 $\Rightarrow 4^2 \times 2^3 = 128$  états au plus pour la structure de Kripke.
- ▶ Seuls  $\sim 30$  sont accessibles mais protocole non immédiat à prouver.



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

## Le model-checking en pratique

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

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

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

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

## 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  $\mathbb{M}$  :  $\mathcal{C}(MU) = \mathbb{C}$ .

$$\varphi \text{ satisfaisable} \iff MU \not\models \neg\varphi$$

### Réduction MC $\leq$ SAT

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

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

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

Réduction polynomiale si  $\varphi_M$  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.
- ▶ 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  $\Sigma$ , où  $\Sigma = 2^{AP}$ .
- ▶ Les formules sont **interprétées** aux positions du mot.

$t, x \models \varphi$  (mot  $t$ , position  $x$ )

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

## LTL : syntaxe et sémantique

Syntaxe :  $LTL(AP, X, U)$

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

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



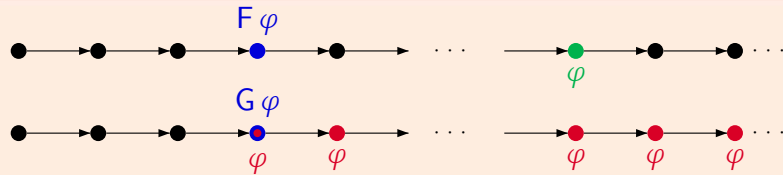
$\varphi \in LTL(AP, X, U)$  définit le langage  $L(\varphi) = \{t \mid t, 0 \models \varphi\}$ .

# 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 :  $G ok$
- ▶ MutEx :  $\neg F(crit_1 \wedge crit_2)$
- ▶ Vivacité :  $G F actif$
- ▶ Réponse :  $G(requ\hat{e}te \Rightarrow F r\acute{e}ponse)$
- ▶ Réponse' :  $G(requ\hat{e}te \Rightarrow X(\neg requ\hat{e}te U r\acute{e}ponse))$
- ▶ Release :  $off R alarme = (alarme U (alarme \wedge off)) \vee (G alarme)$

## Release :

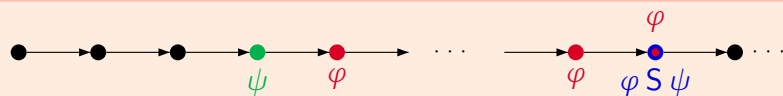
- ▶  $\varphi R \psi = (\psi U (\varphi \wedge \psi)) \vee (G \psi) = \neg(\neg\varphi U \neg\psi)$

# LTL avec passé

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

- $t, x \models Y\varphi$  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

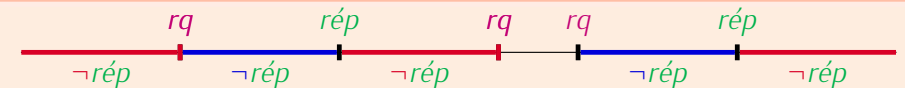


# LTL avec passé

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

- $t, x \models Y\varphi$  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 versus PLTL

$$G(rep \Rightarrow Y(\neg rep S rq)) = (rq R \neg rep) \wedge G(rep \Rightarrow (rq \vee X(rq R \neg rep)))$$

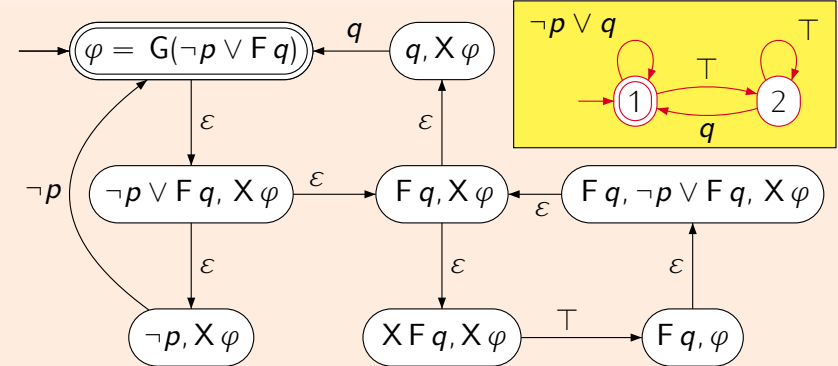
## LTL avec passé

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



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  $\psi$ .

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  $\varphi$  de LTL?

## Logique du premier ordre FO(<)

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

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

**Sémantique**

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

$t, \sigma \models P_q(x)$  si  $q \in \lambda(\sigma(x))$

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

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

$t, \sigma \models x < y$  si  $\sigma(x) < \sigma(y)$

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

# 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  $\varphi \in FO(<)$  définit le langage

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

de  $\Sigma^\omega$  ou  $\Sigma^+$ , selon que l'on travaille sur mots infinis ou finis.

# Logique du premier ordre FO(<) — Exemples

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

$$\begin{aligned} bababababa \dots &\models \varphi \\ abaabaabaaba \dots &\not\models \varphi \end{aligned}$$

- ▶  $\leq = < \setminus <^2$  est exprimable en  $FO_\Sigma(\leq)$  :

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

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

# Spécifications FO

## Spécifications :

- ▶ Sûreté     $\neg \exists x \lambda(x) \in \text{Bad}$
- ▶ 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     $\forall x [\text{alarme} \in \lambda(x) \Rightarrow \text{off} \in \lambda(x) \vee \forall y (x < y \Rightarrow \text{alarme} \in \lambda(y))]$

# FO(<) : satisfaisabilité et model-checking

## Théorème (Büchi)

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

**Idée de preuve.** Induction sur  $\varphi$ . Problème : variables libres.

**Idée de Büchi :** encoder la valuation  $\sigma$  dans le mot.

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

$$\begin{array}{|c|} \hline abaaba \\ \hline \sigma(x_1) = 2 \\ \hline \sigma(x_2) = 5 \\ \hline \sigma(x_3) = 1 \\ \hline \end{array} \text{ codé par } \begin{array}{|c|} \hline abaaba \\ \hline 010000 \\ \hline 000010 \\ \hline 100000 \\ \hline \end{array} = \begin{array}{|c|} \hline t_0 \\ \hline t_1 \\ \hline t_2 \\ \hline t_3 \\ \hline \end{array} = \begin{array}{|c|c|c|c|c|c|} \hline a & b & a & a & b & a \\ \hline 0 & 1 & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 1 & 0 \\ \hline 1 & 0 & 0 & 0 & 0 & 0 \\ \hline \end{array} = t$$

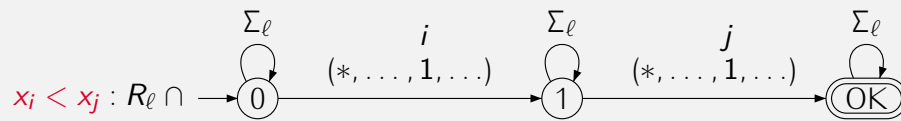
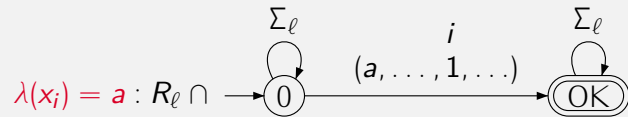
Chaque  $t_i$  est dans  $0^*10^\omega \rightsquigarrow t \in R_\ell$  reconnu par automate de Büchi.

## 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  $\mathcal{A}_\varphi$  sur  $\Sigma_\ell^\omega$  tel que

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



## FO(<), 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 : détermination nécessaire)

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

( $\rightsquigarrow$  produit un automate non-déterministe)

$\Rightarrow$  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**.

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

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

## Logique monadique du second ordre

### Syntaxe : 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  $X, Y \in \text{Var}_2$ .
- ▶  $\sigma$  : interprétation des variables libres tq.  $\sigma(x) \in \mathbb{N}$  et  $\sigma(X) \in 2^{\mathbb{N}}$ .

$$t, \sigma \models x \in X \quad \text{si} \quad \sigma(x) \in \sigma(X)$$

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

## Logique monadique du second ordre — Exemples

### MSO strictement plus expressive que FO

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

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

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

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

## 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  $\text{MSO}(\leq)$  sont les langages rationnels.

Traduction automates vers logique  $\text{MSO}(\leq)$  : 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**.

## C. Des problèmes théoriques et des challenges

Hiérarchies de concaténation

Minimisation d'automates

Coloriage de routes et conjecture de Černý

## Langages sans étoile et langages apériodiques

- ▶ Langages **sans étoile** : langages construits à partir des lettres (et de l'ensemble vide) en utilisant un nombre fini de fois
  - ▶ Union, intersection et **complément**.
  - ▶ Concaténation.

▶ Exemple :  $(ab)^+$  mais pas  $(aa)^+$ . (☹)

- ▶ Langages **apériodiques** : reconnus par monoïde fini apériodique. Un monoïde fini  $M$  est apériodique si

$$\exists k \geq 1 \forall m \in M : m^{k+1} = m^k.$$

**Fait.** On peut effectivement tester si un rationnel  $L$  est **apériodique**.

## Expressivité de FO(<) et LTL

### Théorème de Kamp-Schützenberger-McNaughton-Papert

Les propriétés suivantes pour un langage  $L$  sont équivalentes :

1.  $L$  est exprimable en FO(<).
2.  $L$  est exprimable en LTL.
3.  $L$  est sans étoile.
4.  $L$  est apériodique.
5.  $L$  est reconnaissable et son automate minimal est sans compteur : pas de boucle  $q \xrightarrow{u} p \xrightarrow{u} \dots \xrightarrow{u} q$  avec  $u \in \Sigma^+$  et  $p \neq q$ .

- ▶ C'est un ensemble de résultats difficiles.
- ▶ Preuve élégante de Th. Wilke pour  $2 \iff 4$ .
- ▶  $4, 5 \implies$  **décidable** ! Complexité de 5 : PSPACE-complet (J. Stern).

## Hiérarchies de concaténation

- ▶ On peut former une hiérarchie de classes de langages FO :  $(\mathcal{V}_n)_n$ .
- ▶  $\Sigma_n$  : formules  $\underbrace{\exists^+ \forall^+ \dots \exists^*}_{n \text{ blocs}} \varphi$ ,  $\varphi$  sans quantificateur (forme prénex).

$\mathcal{V}_n = \{\text{combinaisons Booléennes de langages définis par formules } \Sigma_n\}$ .

- ▶ La hiérarchie est stricte.
- ▶ Correspond à l'alternance opérations de concaténation/Booléennes pour les langages sans étoile (W. Thomas 82/J.É. Pin-D. Perrin 86).

**Challenge** : décider l'alternance **minimale** de quantificateurs permettant d'exprimer un langage sans étoile. (i.e., décider l'appartenance à  $\mathcal{V}_n$ ).

## Complexité de la minimisation

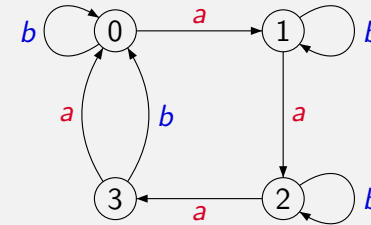
- ▶ J.E. Hopcroft a, en 1974, trouvé un algorithme de minimisation d'automates en  $O(n \log(n))$  ( $n$  : nombre d'états, alphabet fixé).
- ▶ On sait depuis peu que l'algorithme est  $\Omega(n \log(n))$  quels que soient les choix qu'il effectue (J. Berstel, L. Boasson, O. Carton).
- ▶ **Challenge** : La borne  $n \log(n)$  est-elle optimale ?

## Mot synchronisant, automate synchronisant

- ▶ Un mot  $t$  est **synchronisant** pour un automate déterministe s'il envoie tout état sur un même état.

$$\forall p, q \in Q : p.t = q.t.$$

- ▶ Un automate déterministe est **synchronisant** s'il admet un mot synchronisant.



Le mot  $ba^3ba^3b$  est synchronisant.

## Coloriage de routes

- ▶ Problème du **coloriage de routes** : étant donné un graphe orienté de degré sortant constant  $|\Sigma|$ , et dont les longueurs des cycles sont premières entre elles, peut-on l'étiqueter sur  $\Sigma$  pour obtenir un automate synchronisant ?
- ▶ La condition de primalité est nécessaire...
- ▶ Résolu positivement en 2007 par A. Trahtman (preuve courte en cours de vérification. Ouvert pendant 38 ans auparavant).






## Conjecture de Černý

- ▶ **Conjecture de Černý**. Si un automate à  $n$  états est synchronisant, il existe un mot synchronisant de longueur  $\leq (n-1)^2$ .
- ▶ Toujours ouverte depuis 1978, résolue pour des classes particulières d'automates.
- ▶ <http://www.liafa.jussieu.fr/~jep/Problemes/Cerny.html>.

## Un exercice pour terminer

- ▶ Sous-mot de  $t$  : obtenu en prenant une sous-suite de lettres de  $t$ .
- ▶ Soit  $L$  un langage. Montrer que l'ensemble des sous-mots de mots de  $L$  est rationnel
  - ▶ si  $L$  est rationnel (facile).
  - ▶ sans hypothèse sur  $L$ .
- ▶ **Indication.** Lemme de Higman.
- ▶ Ce résultat a des applications en vérification. Utilisé implicitement pour vérifier des systèmes dits **bien structurés** (par exemple, automates communicants avec pertes).

## Pour en savoir plus...

-  **Systems and Software Verification. Model-Checking Techniques and Tools.** B. Bérard, M. Bidoit, A. Finkel, F. Laroussinie, A. Petit, L. Petrucci, P. Schnoebelen, P. McKenzie. Springer, 2001.
-  **Principles of Model Checking.** Ch. Baier, J.-P. Katoen. MIT Press, 2008.
-  **Model Checking.** E.M. Clarke, O. Grumberg, D.A. Peled. MIT Press, 2000.
-  **25 Years of Model Checking : History, Achievements, Perspectives.** O. Grumberg, H. Veith (Eds). Springer, 2008.
-  **Handbook of Formal Languages, vol. 3, chap. 7 (W. Thomas).** G. Rozenberg, A. Salomaa (Eds). Springer, 1997.
- ▶ Quelques problèmes ouverts, dont la conjecture de Černý :  
<http://www.liafa.jussieu.fr/~jep/Problemes/problems.html>.