Automates à état finis

M1 INFO

Marie Tahon
MCF, dpt. Informatique

janvier 2018

Généralités

Objectifs du cours

- Introduction aux modèles utilisés pour décrire, représenter et effectuer des calculs sur des séquences finies de symboles
- Introduction aux automates finis déterministes ou non
- Chaînes de Markov et mélanges gaussiens.

Organisation du cours

- 8h CM
- 8h TD
- 12 TP
- 2h de DS le 3/04/18
- + contrôle continu...

PLAN DE LA SECTION ACTUELLE

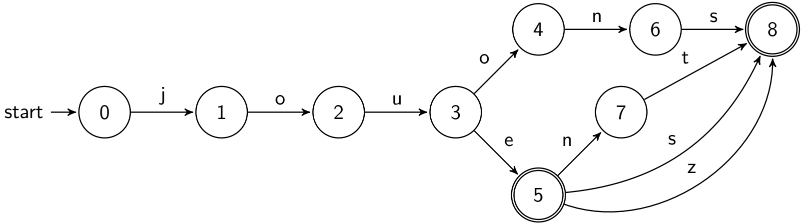
Introduction

A quoi servent les automates finis à états ?

- Compilation:
  - Traduire un ensemble de commandes issues d'un langage vers un autre langage
  - ex: C $\rightarrow$ code machine

  Bio-informatique:
  - Un chromosome est formé de 2 brins d'ADN.
  - A 1 brin d'ADN correspond 1 séquence de nucléotides
  - Modélisation comme des séquences linéaires de symboles dans un alphabet fini
- Traitement des langues naturelles:
  - Suite de sons articulés $\rightarrow$ séquence 1D de symboles dans un inventaire fini (alphabet phonétique)
  - Suite de graphèmes $\rightarrow$ séquence de mots $\rightarrow$ phrases

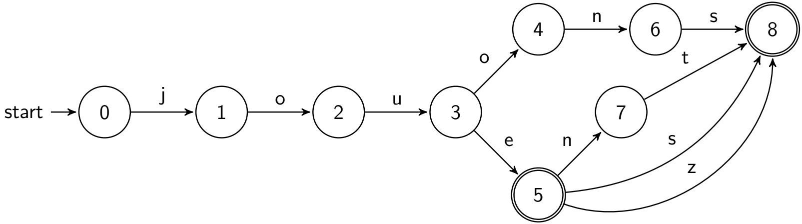
Exemple



Un automate fini est un graphe orienté dont les arcs sont étiquetés par des symboles (ici des lettres). Il comporte:

- des états: les nœuds
- des transitions: les arcs
- un état initial: flèche entrante venant de nulle part et sans étiquette (ici état 0)
- des états finaux, éventuellement $\emptyset$ (ici états 5 et 8)

Exemple



On s'intéresse aux chemins qui vont de l'état initial aux états finaux.

états parcourus	chaîne d'étiquettes associée
$0 \to 1 \to 2 \to 3 \to 5$	joue
$0 \to 1 \to 2 \to 3 \to 4 \to 6 \to 8$	jouons
$0 \to 1 \to 2 \to 3 \to 5 \to 7 \to 8$	jouent
$0 \to 1 \to 2 \to 3 \to 5 \to 8$	joues
$0 \to 1 \to 2 \to 3 \to 5 \to 8$	jouez

L'ensemble de ces chaînes forme le langage de l'automate.

Ici le langage est le présent de l'indicatif du verbe "jouer".

Définitions générales

- Un alphabet est un ensemble Σ fini de symboles.
- Un mot (ou chaîne) est une suite finie d'éléments de Σ.
  le mot vide se note ϵ
- L'ensemble des mots formés à partir de l'alphabet Σ (resp. de tous les mots non vides) est noté Σ^* (resp. Σ^+).
- Un langage L sur Σ est un sous-ensemble de Σ^*.

Exemples avec $\Sigma = \{a, b, c\}$

- $\Sigma^* = \{\epsilon, a, b, c, aa, ab, ac, bb, \ldots, aaa, abc, \ldots\}$
- $L_1 = \{\epsilon, a, b, c\}$
- $L_2 = \{\epsilon, ab, aabb, aaabbb, \ldots\}$, noté $\{a^n b^n\}$ avec $n > 0$
- $L_3 = \{\epsilon, abc, aabbcc, aaabbbccc\}$, noté $\{a^n b^n c^n\}$ avec $n > 0$

Longueur d'un mot

La longueur d'un mot u se note $|u|$ et correspond au nombre total de symboles de u.

- $|\epsilon| = 0$
- $|u|_a$ comptabilise le nombre d'occurences du symbole a dans le mot u

$$|u| = \sum_{a \in \Sigma} |u|_a$$

Exemples avec $\Sigma = \{a, b, c\}$

- $L_2 = \{\epsilon, ab, aabb, aaabbb, \ldots\}$, noté $\{a^n b^n\}$ avec $n > 0$
- $|aabb| = 4$
- $|a^n b^n| = |u|_a + |u|_b = 2n$

Longueur d'un mot

La longueur d'un mot u se note $|u|$ et correspond au nombre total de symboles de u.

- $|\epsilon| = 0$
- $|u|_a$ comptabilise le nombre d'occurences du symbole a dans le mot u

$$|u| = \sum_{a \in \Sigma} |u|_a$$

Exemples avec $\Sigma = \{a, b, c\}$

- $L_2 = \{\epsilon, ab, aabb, aaabbb, \ldots\}$, noté $\{a^n b^n\}$ avec $n > 0$
- $|aabb| = 4$
- $|a^n b^n| = |u|_a + |u|_b = 2n$

Opérations ensemblistes sur les langages

Soit L, M des langages de Σ^*, on définit:

- L'union: $L \cup M = \{x \in \Sigma^\star | x \in L \text{ ou } x \in M\}$
- L'intersection: $L \cap M = \{x \in \Sigma^\star | x \in L \text{ et } x \in M\}$
- La différence: $L \setminus M = \{x \in \Sigma^\star | x \in L \text{ et } x \notin M\}$
- Le complémentaire: $\overline{L} = \{x \in \Sigma^\star | x \notin L\}$

Les opérations $\cup$ et $\cap$ sont associatives et commutatives.

Concaténation: définitions et propriétés

Concaténation de mots:

- La concatenation de 2 mots u et v de Σ^* donne un nouveau mot uv constitué par la juxtaposition des lettres de u puis celles de v
- $|uv| = |u| + |v|$ et $|uv|_a = |u|_a + |v|_a$
- $uv \neq vu$ et $u(vw) = (uv)w = uvw$
- ϵ est l'élément neutre de la concaténation: $u\epsilon = \epsilon u = u$
- u^n est la concaténation de n fois le mot u et $u^0 = \epsilon$

Concaténation de langages ou produit de langages:

- $L \times M = \{xy | x \in L \text{ et } y \in M\}$
- $L_1L_2 \neq L_2L_1$ et $L_1(L_2L_3) = (L_1L_2)L_3 = L_1L_2L_3$
- L^n est la concaténation de n copies du langage L et $L^0 = \epsilon$

Concaténation: fermeture de Kleene

La fermeture de Kleene (ou étoile de Kleene) d'un langage L se définit par:

$$L^\star = \bigcup_{i \geq 0} L^i$$

- $L^\star$ contient tous les mots qu'il est possible de construire en concaténant un nombre fini d'éléments de langage L
- ex: $L = \{a, b\}$ alors $L^\star = \{\epsilon, a, b, aa, ab, bb, aaa, aab, \dots\}$
- $\emptyset^\star \neq \emptyset$: il contient ϵ

On définit aussi la fermeture stricte:

$$L^+ = \bigcup_{i > 0} L^i = LL^\star$$

- Attention, contrairement à $L^\star$, L^+ ne contient pas forcément ϵ

Langages réguliers: définition

Soit Σ un alphabet, un langage régulier (LangReg) de Σ est un langage défini de la façon suivante:

- $\emptyset$ est un langage régulier
- $\{\epsilon\}$ est un langage régulier
- $\forall a \in \Sigma$, a $\{a\}$) est un langage régulier
- Si L et M sont des langages réguliers, alors les langages suivants sont aussi réguliers:
  - $L \cup M$
  - LM
  - $L^\star$
- rien d'autre n'est un langage régulier

Langages réguliers: exemples

Quelques exemples de langages réguliers

- Pour tout mot $u \in \Sigma^*$, $\{u\}$ est un langage régulier

  soit $u = a_1 a_2 \dots a_n$, preuve par concaténation
- Tout langage fini est régulier

  un langage fini est un ensemble fini de mots
- L'ensemble de tous les mots Σ^* est régulier

  Σ^* est la fermeture de Kleene de Σ
- Soit $\Sigma = \{a, b\}$, l'ensemble $\{a^n b^p | n, p \in \mathbb{N}\}$ est régulier

  $\{a^n\}$ est la fermeture de Kleene de $\{a\}$ puis par concaténation

Expressions régulières: définition

Soit Σ un alphabet, une expression régulière est une chaîne de caractères (ou notation) qui dénote un langage régulier sur Σ:

- $\emptyset$ est une expression régulière qui dénote le LangReg $\emptyset$
- ϵ est une expression régulière qui dénote le LangReg $\{\epsilon\}$
- $\forall a \in \Sigma$, a est une expression régulière qui dénote le LangReg $\{a\}$
- Si e_1 et e_2 sont des expressions régulières dénotant respectivement les LangReg L_1 et L_2, alors:
  - $(e_1 + e_2)$ est une expression régulière qui dénote le LangReg $L_1 \cup L_2$,
  - $(e_1 e_2)$ est une expression régulière qui dénote le LangReg $L_1 L_2$,
  - et $(e_1^\star)$ est une expression régulière qui dénote le LangReg $L_1^\star$.
- rien d'autre n'est une expression régulière.

Expressions régulières: définition

Soit Σ un alphabet, une expression régulière est une chaîne de caractères (ou notation) qui dénote un langage régulier sur Σ:

- $\emptyset$ est une expression régulière qui dénote le LangReg $\emptyset$
- ϵ est une expression régulière qui dénote le LangReg $\{\epsilon\}$
- $\forall a \in \Sigma$, a est une expression régulière qui dénote le LangReg $\{a\}$
- Si e_1 et e_2 sont des expressions régulières dénotant respectivement les LangReg L_1 et L_2, alors:
  - $(e_1 + e_2)$ est une expression régulière qui dénote le LangReg $L_1 \cup L_2$,
  - $(e_1 e_2)$ est une expression régulière qui dénote le LangReg $L_1 L_2$,
  - et $(e_1^\star)$ est une expression régulière qui dénote le LangReg $L_1^\star$.
- rien d'autre n'est une expression régulière.

Expressions régulières: définition

Soit Σ un alphabet, une expression régulière est une chaîne de caractères (ou notation) qui dénote un langage régulier sur Σ:

- $\emptyset$ est une expression régulière qui dénote le LangReg $\emptyset$
- ϵ est une expression régulière qui dénote le LangReg $\{\epsilon\}$
- $\forall a \in \Sigma$, a est une expression régulière qui dénote le LangReg $\{a\}$
- Si e_1 et e_2 sont des expressions régulières dénotant respectivement les LangReg L_1 et L_2, alors:
  - $(e_1 + e_2)$ est une expression régulière qui dénote le LangReg $L_1 \cup L_2$,
  - $(e_1 e_2)$ est une expression régulière qui dénote le LangReg $L_1 L_2$,
  - et $(e_1^\star)$ est une expression régulière qui dénote le LangReg $L_1^\star$.
- rien d'autre n'est une expression régulière.

Expressions régulières: définition

Soit Σ un alphabet, une expression régulière est une chaîne de caractères (ou notation) qui dénote un langage régulier sur Σ:

- $\emptyset$ est une expression régulière qui dénote le LangReg $\emptyset$
- ϵ est une expression régulière qui dénote le LangReg $\{\epsilon\}$
- $\forall a \in \Sigma$, a est une expression régulière qui dénote le LangReg $\{a\}$
- Si e_1 et e_2 sont des expressions régulières dénotant respectivement les LangReg L_1 et L_2, alors:
    - $(e_1 + e_2)$ est une expression régulière qui dénote le LangReg $L_1 \cup L_2$,
    - $(e_1 e_2)$ est une expression régulière qui dénote le LangReg $L_1 L_2$,
    - et $(e_1^\star)$ est une expression régulière qui dénote le LangReg $L_1^\star$.
- rien d'autre n'est une expression régulière.

Expressions régulières: définition

Soit Σ un alphabet, une expression régulière est une chaîne de caractères (ou notation) qui dénote un langage régulier sur Σ:

- $\emptyset$ est une expression régulière qui dénote le LangReg $\emptyset$
- ϵ est une expression régulière qui dénote le LangReg $\{\epsilon\}$
- $\forall a \in \Sigma$, a est une expression régulière qui dénote le LangReg $\{a\}$
- Si e_1 et e_2 sont des expressions régulières dénotant respectivement les LangReg L_1 et L_2, alors:
    - $(e_1 + e_2)$ est une expression régulière qui dénote le LangReg $L_1 \cup L_2$,
    - $(e_1 e_2)$ est une expression régulière qui dénote le LangReg $L_1 L_2$,
    - et $(e_1^\star)$ est une expression régulière qui dénote le LangReg $L_1^\star$.
- rien d'autre n'est une expression régulière.

Expressions régulières: définition

Soit Σ un alphabet, une expression régulière est une chaîne de caractères (ou notation) qui dénote un langage régulier sur Σ:

- $\emptyset$ est une expression régulière qui dénote le LangReg $\emptyset$
- ϵ est une expression régulière qui dénote le LangReg $\{\epsilon\}$
- $\forall a \in \Sigma$, a est une expression régulière qui dénote le LangReg $\{a\}$
- Si e_1 et e_2 sont des expressions régulières dénotant respectivement les LangReg L_1 et L_2, alors:
  - $(e_1 + e_2)$ est une expression régulière qui dénote le LangReg $L_1 \cup L_2$,
  - $(e_1 e_2)$ est une expression régulière qui dénote le LangReg $L_1 L_2$,
  - et $(e_1^\star)$ est une expression régulière qui dénote le LangReg $L_1^\star$.
- rien d'autre n'est une expression régulière.

Règles de priorité: fermeture de Kleene $>$ concaténation $>$ union
ex: $0 + 10^\star \equiv (0 + (1(0)^\star))$

Expressions régulières: exemples

Les expressions suivantes sont-elles régulières ? et quel langage dénotent-elles ?

- $toto^\star$?
- $ab(a+b)^\star$
- $r, e, d,$ é
- $(((fa)i)r)e$
- $(a^\star)(\epsilon + b)$

Expressions régulières: exemples

Les expressions suivantes sont-elles régulières ? et quel langage dénotent-elles ?

- $toto^\star$? oui, $\Sigma = \{tot, toto, totoo, \ldots\}$
- $ab(a+b)^\star$
- $r, e, d,$ é
- $(((fa)i)r)e$
- $(a^\star)(\epsilon + b)$

Expressions régulières: exemples

Les expressions suivantes sont-elles régulières ? et quel langage dénotent-elles ?

- $toto^\star$? oui, $\Sigma = \{tot, toto, totoo, \ldots\}$
- $ab(a+b)^\star$ oui, $\Sigma = \{ab, aba, abb, abaa, abbb, \ldots\}$
- $r, e, d,$ é
- $(((fa)i)r)e$
- $(a^\star)(\epsilon + b)$

Expressions régulières: exemples

Les expressions suivantes sont-elles régulières ? et quel langage dénotent-elles ?

- $toto^\star$? oui, $\Sigma = \{tot, toto, totoo, \ldots\}$
- $ab(a+b)^\star$ oui, $\Sigma = \{ab, aba, abb, abaa, abbb, \ldots\}$
- $r, e, d,$ é non
- $(((fa)i)r)e$
- $(a^\star)(\epsilon + b)$

Expressions régulières: exemples

Les expressions suivantes sont-elles régulières ? et quel langage dénotent-elles ?

- $toto^\star$? oui, $\Sigma = \{tot, toto, totoo, \ldots\}$
- $ab(a+b)^\star$ oui, $\Sigma = \{ab, aba, abb, abaa, abbb, \ldots\}$
- $r, e, d,$ é non
- $(((fa)i)r)e$ oui, $\Sigma = \{faire\}$
- $(a^\star)(\epsilon + b)$

Expressions régulières: exemples

Les expressions suivantes sont-elles régulières ? et quel langage dénotent-elles ?

- $toto^\star$? oui, $\Sigma = \{tot, toto, totoo, \ldots\}$
- $ab(a+b)^\star$ oui, $\Sigma = \{ab, aba, abb, abaa, abbb, \ldots\}$
- $r, e, d,$ é non
- $(((fa)i)r)e$ oui, $\Sigma = \{faire\}$
- $(a^\star)(\epsilon + b)$ oui, $\Sigma = \{b, ab, aab, aaab, aa, aaa, \ldots\}$

PLAN DE LA SECTION ACTUELLE

1. Introduction et définitions générales
2. Automate fini à états
   - Définitions
   - Exemples
3. Langage reconnaissable et automates
4. Opérations sur les langages reconnaissables

Automate fini à état

Un automate fini à états est défini par un quintuplet $A = (\Sigma, Q, q_0, F, \delta)$, où:

- Σ est un alphabet (ensemble fini de symboles)
- Q est un ensemble fini d'états ($Q \neq \emptyset$)
- $q_0 \in Q$ est l'état initial
- $F \subset Q$ est l'ensemble des états finaux (éventuellement $F = \emptyset$)
- δ est la fonction de transition définie sur $Q' \times \Sigma' \subseteq Q \times \Sigma$

$$\delta : Q' \times \Sigma' \to Q$$

.

Automate fini à état

Un automate fini à états est défini par un quintuplet $A = (\Sigma, Q, q_0, F, \delta)$, où:

- Σ est un alphabet (ensemble fini de symboles)
- Q est un ensemble fini d'états ($Q \neq \emptyset$)
- $q_0 \in Q$ est l'état initial
- $F \subset Q$ est l'ensemble des états finaux (éventuellement $F = \emptyset$)
- δ est la fonction de transition définie sur $Q' \times \Sigma' \subseteq Q \times \Sigma$

$$\delta : Q' \times \Sigma' \to Q$$

.

L'ensemble des transitions δ est une relation, c'est-à-dire un ensemble de triplets

$$(q, a, r) \in (Q' \times \Sigma' \times Q)$$

- q l'origine de la transition
- a l'étiquette de la transition
- $\delta(q, a) = r$ la destination de la transition

Automate fini à états: définitions

Un automate est dit:

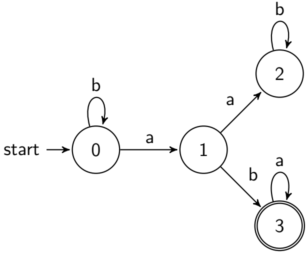
- fini lorsque l'ensemble de ses composantes sont finies (Σ, Q, F).
- déterministe (AFD) si d'un état donné, il part au plus un seul arc étiqueté par une lettre donnée.
  - $\delta : Q' \times \Sigma \rightarrow Q$
  - en informatique, le déterminisme est le fait de ne pas avoir le choix entre plusieurs exécutions.
- non-déterministe (AFN) si d'un état donné, il part plusieurs arc étiquetés par la même lettre.
  - $\delta : Q' \times \Sigma \rightarrow Q^n$ avec n le nombre d'états

Automate fini à états: définitions

Un automate est dit:

- fini lorsque l'ensemble de ses composantes sont finies (Σ, Q, F).
- déterministe (AFD) si d'un état donné, il part au plus un seul arc étiqueté par une lettre donnée.
  - $\delta : Q' \times \Sigma \rightarrow Q$
  - en informatique, le déterminisme est le fait de ne pas avoir le choix entre plusieurs exécutions.
- non-déterministe (AFN) si d'un état donné, il part plusieurs arc étiquetés par la même lettre.
  - $\delta : Q' \times \Sigma \rightarrow Q^n$ avec n le nombre d'états

AFD:

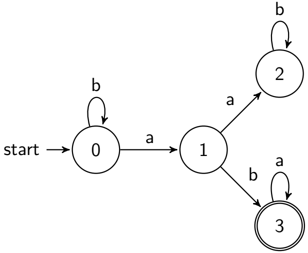


Automate fini à états: définitions

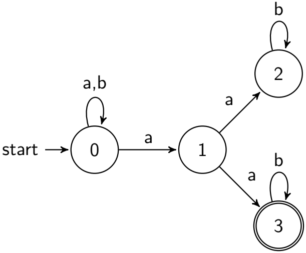
Un automate est dit:

- fini lorsque l'ensemble de ses composantes sont finies (Σ, Q, F).
- déterministe (AFD) si d'un état donné, il part au plus un seul arc étiqueté par une lettre donnée.
  - $\delta : Q' \times \Sigma \to Q$
  - en informatique, le déterminisme est le fait de ne pas avoir le choix entre plusieurs exécutions.
- non-déterministe (AFN) si d'un état donné, il part plusieurs arc étiquetés par la même lettre.
  - $\delta : Q' \times \Sigma \to Q^n$ avec n le nombre d'états

AFD:



AFN:

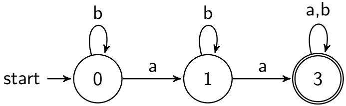


Automate fini à états: définitions

Un automate est dit:

- complet si et seulement si δ est totale: $\delta : Q \times \Sigma \rightarrow Q$
  - de chaque état part alors exactement un arc étiqueté par chacune des lettres de l'alphabet
- partiel si et seulement si δ est partielle $\delta : Q' \times \Sigma' \rightarrow Q$ avec $(Q' \times \Sigma' \subset Q \times \Sigma)$
  - une fonction partielle est définie sur une partie de l'ensemble de départ.
  - en ce cas, l'automate fini peut se trouver bloqué (cf algo de reconnaissance d'un mot)
  - un automate fini partiel peut être complété par des états superflus: états poubelle ou puits afin qu'il devienne complet (cf état puit)

AFD complet

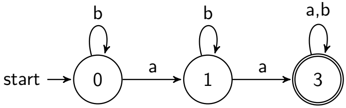


Automate fini à états: définitions

Un automate est dit:

- complet si et seulement si δ est totale: $\delta : Q \times \Sigma \rightarrow Q$
    - de chaque état part alors exactement un arc étiqueté par chacune des lettres de l'alphabet
- partiel si et seulement si δ est partielle $\delta : Q' \times \Sigma' \rightarrow Q$ avec $(Q' \times \Sigma' \subset Q \times \Sigma)$
    - une fonction partielle est définie sur une partie de l'ensemble de départ.
    - en ce cas, l'automate fini peut se trouver bloqué (cf algo de reconnaissance d'un mot)
    - un automate fini partiel peut être complété par des états superflus: états poubelle ou puits afin qu'il devienne complet (cf état puit)

AFD complet

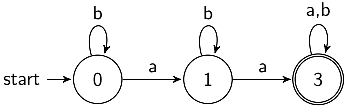


Automate fini à états: définitions

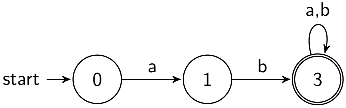
Un automate est dit:

- complet si et seulement si δ est totale: $\delta : Q \times \Sigma \to Q$
  - de chaque état part alors exactement un arc étiqueté par chacune des lettres de l'alphabet
- partiel si et seulement si δ est partielle $\delta : Q' \times \Sigma' \to Q$ avec $(Q' \times \Sigma' \subset Q \times \Sigma)$
  - une fonction partielle est définie sur une partie de l'ensemble de départ.
  - en ce cas, l'automate fini peut se trouver bloqué (cf algo de reconnaissance d'un mot)
  - un automate fini partiel peut être complété par des états superflus: états poubelle ou puits afin qu'il devienne complet (cf état puit)

AFD complet



AFD partiellement spécifié

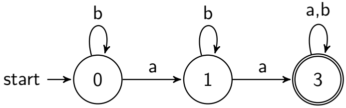


Automate fini à états: définitions

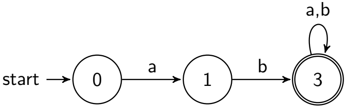
Un automate est dit:

- complet si et seulement si δ est totale: $\delta : Q \times \Sigma \to Q$
  - de chaque état part alors exactement un arc étiqueté par chacune des lettres de l'alphabet
- partiel si et seulement si δ est partielle $\delta : Q' \times \Sigma' \to Q$ avec $(Q' \times \Sigma' \subset Q \times \Sigma)$
  - une fonction partielle est définie sur une partie de l'ensemble de départ.
  - en ce cas, l'automate fini peut se trouver bloqué (cf algo de reconnaissance d'un mot)
  - un automate fini partiel peut être complété par des états superflus: états poubelle ou puits afin qu'il devienne complet (cf état puit)

AFD complet



AFD partiellement spécifié

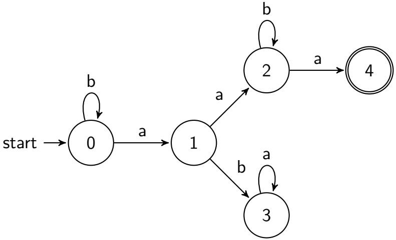


Automate fini à états: définitions

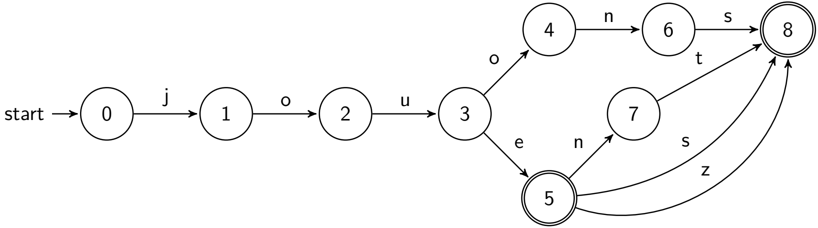
Un état q d'un automate A, est dit:

- accessible si il est possible d'accéder à q à partir de l'état intial
- co-accessible si il est possible d'accéder à état final depuis q.
- utile si il est à la fois accessible et co-accessible. Un automate dont tous les états sont utiles est émondé.

Exemple:



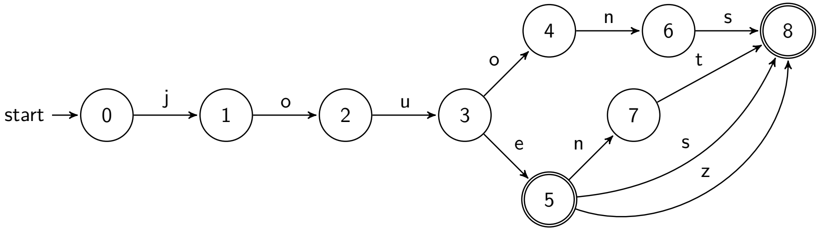
Exemple 1



- Alphabet: $\Sigma = \{e, j, n, o, s, t, u, z\}$
- Ensemble des états: $Q = \{0, 1, 2, 3, 4, 5, 6, 7, 8\}$
- Etat initial: $q_0 = 0$
- Etats finaux: $F = \{5, 8\}$
- Ensemble des transitions δ

$$\delta = \{(0, j, 1), (1, o, 2), (2, u, 3), (3, o, 4), (3, o, 5), (4, n, 6), (6, s, 8), (5, z, 8), \\ (5, n, 7), (7, t, 8), (5, z, 8)\} \in Q \times \Sigma \times Q$$

Exemple 1



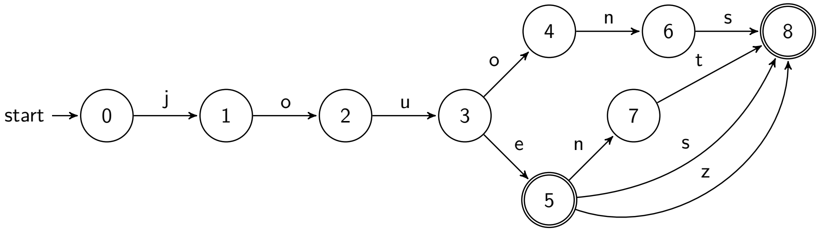
Fonction de transition:

δ	j	o	u	o	e	n	s	t	z
0	1								
1		2							
2			3						
3				4	5				
4						6			
5						7	8		8
6							8		
7								8	

$\delta(q, a) = r$

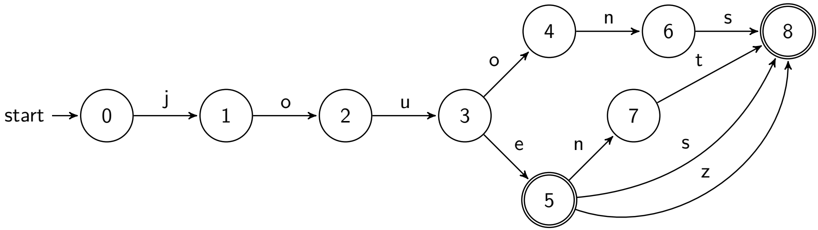
- a étiquette de la transition
- q origine de la transition
- r destination de la transition
- ex: $\delta(0, j) = 1$

Exemple 1



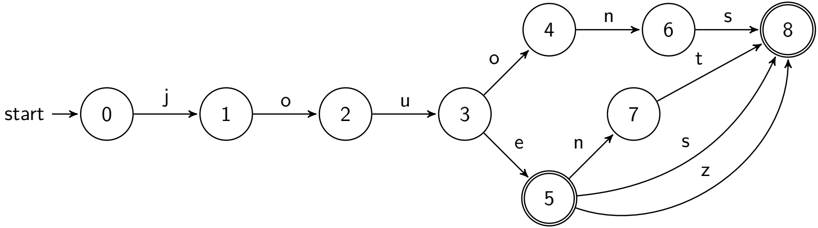
- Automate fini ?
- Automate déterministe ?
- Automate complet ?

Exemple 1



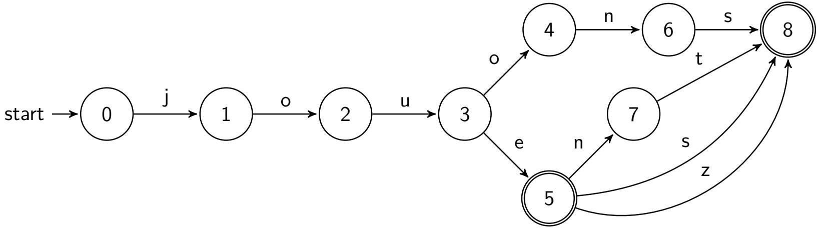
- Automate fini ? Σ, Q, F sont finis $\rightarrow$ oui
- Automate déterministe ?
- Automate complet ?

Exemple 1



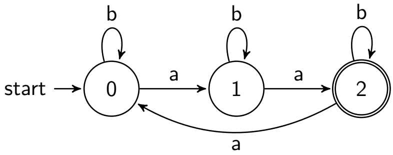
- Automate fini ? Σ, Q, F sont finis $\rightarrow$ oui
- Automate déterministe ? jamais 2 possibilités pour la même étiquette à partir d'un même état $\rightarrow$ oui
- Automate complet ?

Exemple 1



- Automate fini ? Σ, Q, F sont finis $\rightarrow$ oui
- Automate déterministe ? jamais 2 possibilités pour la même étiquette à partir d'un même état $\rightarrow$ oui
- Automate complet ? matrice de transitions incomplète $\rightarrow$ non

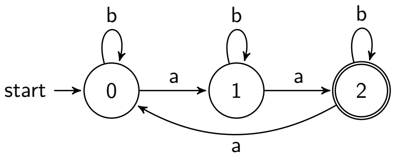
Exemple 2



- Alphabet: $\Sigma = \{a, b\}$
- Ensemble des états: $Q = \{0, 1, 2\}$
- Etat initial: $q_0 = 0$
- Etats finaux: $F = \{2\}$
- Ensemble des transitions δ

$$\delta = \{(0, a, 1), (0, b, 0), (1, a, 2), (1, b, 1), (2, a, 0), (2, b, 2)\} \in Q \times \Sigma \times Q$$

Exemple 2



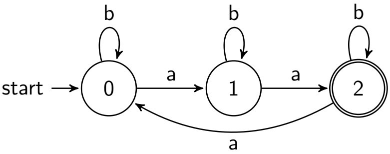
Fonction de transition:

δ	a	b
0	1	0
1	2	1
2	0	2

$\delta(q, a) = r$

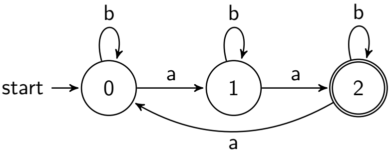
- a étiquette de la transition
- q origine de la transition
- r destination de la transition
- ex: $\delta(0, a) = 1$

Exemple 2



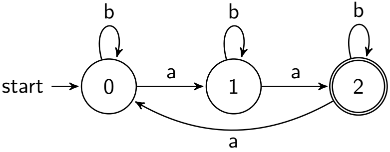
- Automate fini ?
- Automate déterministe ?
- Automate complet ?

Exemple 2



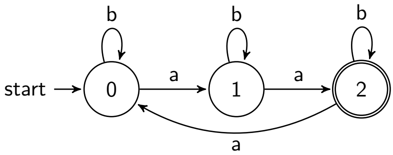
- Automate fini ? Σ, Q, F sont finis $\rightarrow$ oui
- Automate déterministe ?
- Automate complet ?

Exemple 2



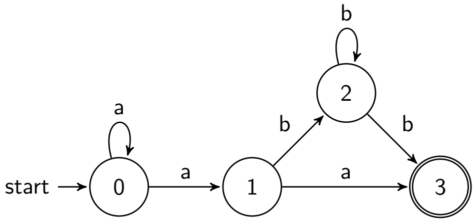
- Automate fini ? Σ, Q, F sont finis $\rightarrow$ oui
- Automate déterministe ? jamais 2 possibilités pour la même étiquette à partir d'un même état $\rightarrow$ oui
- Automate complet ?

Exemple 2



- Automate fini ? Σ, Q, F sont finis $\rightarrow$ oui
- Automate déterministe ? jamais 2 possibilités pour la même étiquette à partir d'un même état $\rightarrow$ oui
- Automate complet ? matrice de transitions complète $\rightarrow$ oui

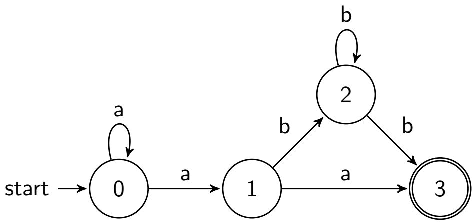
Exemple 3



- Alphabet: $\Sigma = \{a, b\}$
- Ensemble des états: $Q = \{0, 1, 2, 3\}$
- Etat initial: $q_0 = 0$
- Etats finaux: $F = \{3\}$
- Ensemble des transitions δ

$$\delta = \{(0, a, (0, 1)), (1, a, 3), (1, b, 2), (2, b, (2, 3))\} \in Q \times \Sigma \times 2^Q$$

Exemple 3



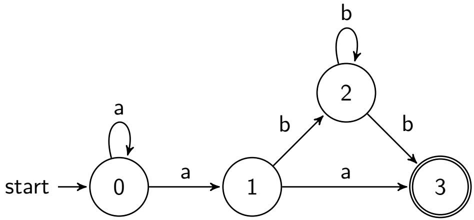
Fonction de transition:

δ	a	b
0	(0,1)	
1	3	2
2		(2,3)

$\delta(q, a) = r$

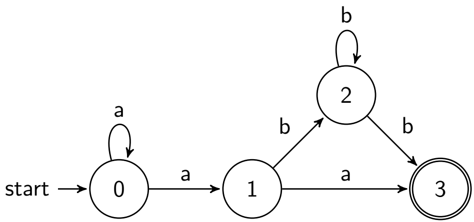
- a étiquette de la transition
- q origine de la transition
- r destination de la transition
- ex: $\delta(0, a) = (0, 1)$

Exemple 3



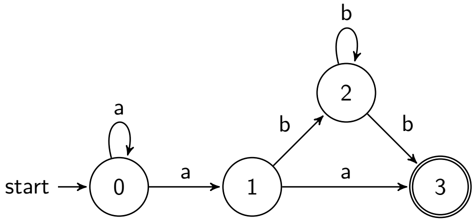
- Automate fini ?
- Automate déterministe ?
- Automate complet ?

Exemple 3



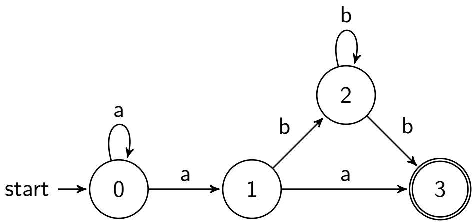
- Automate fini ? Σ, Q, F sont finis $\rightarrow$ oui
- Automate déterministe ?
- Automate complet ?

Exemple 3



- Automate fini ? Σ, Q, F sont finis $\rightarrow$ oui
- Automate déterministe ? à l'état 0 il y a 2 possibilités pour a $\rightarrow$ non
- Automate complet ?

Exemple 3



- Automate fini ? Σ, Q, F sont finis $\rightarrow$ oui
- Automate déterministe ? à l'état 0 il y a 2 possibilités pour a $\rightarrow$ non
- Automate complet ? matrice de transitions partielle $\rightarrow$ non

Calcul: définitions

- Un calcul dans l'automate A est une séquence de transitions $e_1 \ldots e_n$ de A telle que pour tout couple de transitions successives (e_i, e_{i+1}), l'état destination de e_i est l'état d'origine de e_{i+1}.
- L'étiquette d'un calcul est le mot construit par concaténation des étiquettes de chacune des transitions.
- Un calcul dans A est réussi si la 1ère transition a pour origine l'état initial et la dernière transition a pour destination un des états finaux.
- Le langage reconnu par A, noté $L(A)$ est l'ensemble des étiquettes des calculs réussis.

Calcul: définitions

- Un calcul dans l'automate A est une séquence de transitions $e_1 \dots e_n$ de A telle que pour tout couple de transitions successives (e_i, e_{i+1}), l'état destination de e_i est l'état d'origine de e_{i+1}.
- L'étiquette d'un calcul est le mot construit par concaténation des étiquettes de chacune des transitions.
- Un calcul dans A est réussi si la 1ère transition a pour origine l'état initial et la dernière transition a pour destination un des états finaux.
- Le langage reconnu par A, noté $L(A)$ est l'ensemble des étiquettes des calculs réussis.

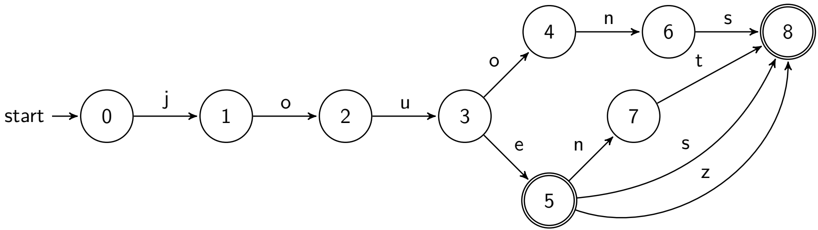
Calcul: définitions

- Un calcul dans l'automate A est une séquence de transitions $e_1 \ldots e_n$ de A telle que pour tout couple de transitions successives (e_i, e_{i+1}), l'état destination de e_i est l'état d'origine de e_{i+1}.
- L'étiquette d'un calcul est le mot construit par concaténation des étiquettes de chacune des transitions.
- Un calcul dans A est réussi si la 1ère transition a pour origine l'état initial et la dernière transition a pour destination un des états finaux.
- Le langage reconnu par A, noté $L(A)$ est l'ensemble des étiquettes des calculs réussis.

Calcul: définitions

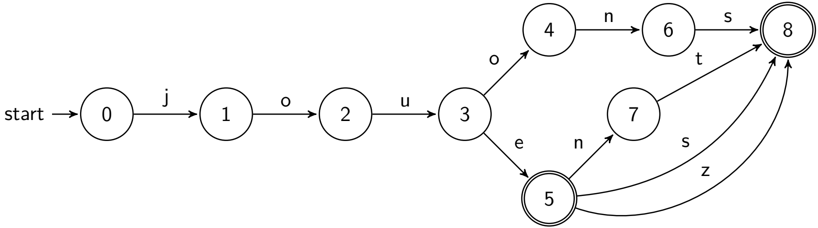
- Un calcul dans l'automate A est une séquence de transitions $e_1 \dots e_n$ de A telle que pour tout couple de transitions successives (e_i, e_{i+1}), l'état destination de e_i est l'état d'origine de e_{i+1}.
- L'étiquette d'un calcul est le mot construit par concaténation des étiquettes de chacune des transitions.
- Un calcul dans A est réussi si la 1ère transition a pour origine l'état initial et la dernière transition a pour destination un des états finaux.
- Le langage reconnu par A, noté $L(A)$ est l'ensemble des étiquettes des calculs réussis.

Calcul: exemple 1



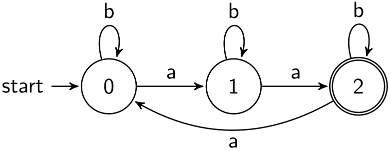
- Calcul 1: $(0, j, 1), (1, o, 2), (2, u, 3), (3, e, 5)$, étiquette *joue*
- Calcul 2: $(0, j, 1), (1, o, 2), (2, u, 3), (3, o, 4), (4, n, 6)$, étiquette *jouon*
- Calcul 1 réussi car 4 état final
- Donc *joue* appartient au langage reconnu
- Le langage défini par l'automate est donc : $L(A) = \{joue, jouons, jouent, joues, jouez\}$

Calcul: exemple 1



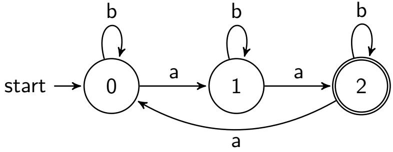
- Calcul 1: $(0, j, 1), (1, o, 2), (2, u, 3), (3, e, 5)$, étiquette *joue*
- Calcul 2: $(0, j, 1), (1, o, 2), (2, u, 3), (3, o, 4), (4, n, 6)$, étiquette *jouon*
- Calcul 1 réussi car 4 état final
- Donc *joue* appartient au langage reconnu
- Le langage défini par l'automate est donc :
  $L(A) = \{joue, jouons, jouent, joues, jouez\}$

Calcul: exemple 2



- Calcul 1: $(0, b, 0)(0, a, 1)$, étiquette *ba*
- Calcul 2: $(0, b, 0)(0, a, 1)(1, a, 2)(2, b, 2)$, étiquette *baab*
- Le calcul 2 est réussi car 2 est un état final
- Donc *baab* appartient au langage reconnu
- Le langage défini par l'automate est l'ensemble des mots contenant au moins $2 + 3^n$ a: $L(A) = \{u \text{ tel que } |u|_a = 2 + 3^n, \}_{n \in \mathbb{N}}$

Calcul: exemple 2



- Calcul 1: $(0, b, 0)(0, a, 1)$, étiquette *ba*
- Calcul 2: $(0, b, 0)(0, a, 1)(1, a, 2)(2, b, 2)$, étiquette *baab*
- Le calcul 2 est réussi car 2 est un état final
- Donc *baab* appartient au langage reconnu
- Le langage défini par l'automate est l'ensemble des mots contenant au moins $2 + 3^n$ a: $L(A) = \{u \text{ tel que } |u|_a = 2 + 3^n, \}_{n \in \mathbb{N}}$

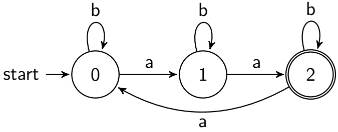
PLAN DE LA SECTION ACTUELLE

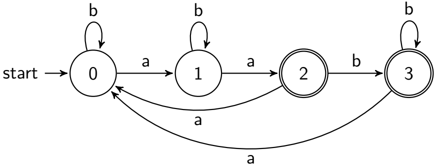
1. Introduction et définitions générales
2. Automate fini à états
3. Langage reconnaissable et automates
   - Définitions
   - Langage reconnaissable par un AFD
   - Langage reconnaissable par un AFN
   - Déterminisation d'un AFN
   - $\epsilon - AFN$
4. Opérations sur les langages reconnaissables

Langage reconnaissable

Définitions:

- Un langage est reconnaissable ou régulier si il existe un automate fini qui le reconnaît
- Deux automates finis A_1 et A_2 sont équivalents si et seulement si ils reconnaissent le même langage.





$L_1(A) = L_2(A) = \{u \text{ tel que } |u|_a = 2 + 3^n\}_{n\in\mathbb{N}}$

Algorithme de reconnaissance d'un mot avec un AFD

Les automates finis sont des machines abstraites qui savent reconnaître l'appartenance ou la non-appartenance d'un mot à un langage régulier donné.

- L'automate "lit" un mot écrit sur un ruban d'entrée
- Il part de l'état initial et à chaque lettre lue, il change d'état.
- Si à la fin du mot, il est dans un état final, le mot à été reconnu par l'automate.

Algorithme de reconnaissance d'un mot avec un AFD

Les automates finis sont des machines abstraites qui savent reconnaître l'appartenance ou la non-appartenance d'un mot à un langage régulier donné.

- L'automate "lit" un mot écrit sur un ruban d'entrée
- Il part de l'état initial et à chaque lettre lue, il change d'état.
- Si à la fin du mot, il est dans un état final, le mot à été reconnu par l'automate.

Ces machines abstraites constituent un modèle théorique de référence.

Algorithme de reconnaissance d'un mot avec un AFD

Les automates finis sont des machines abstraites qui savent reconnaître l'appartenance ou la non-appartenance d'un mot à un langage régulier donné.

- L'automate "lit" un mot écrit sur un ruban d'entrée
- Il part de l'état initial et à chaque lettre lue, il change d'état.
- Si à la fin du mot, il est dans un état final, le mot à été reconnu par l'automate.

Ces machines abstraites constituent un modèle théorique de référence.

Dans la pratique, de nombreuses applications implémentent la notion d'automates finis ou ses variantes (compilateur, machine à café,...).

Algorithme de reconnaissance d'un mot avec un AFD

Soit un automate fini déterministe $A = (\Sigma, Q, q_0, \delta, F)$, la reconnaissance d'un mot $u = u_1 \dots u_n$ du langage $L(A)$ s'effectue avec l'algorithme suivant:

Data: u mot à reconnaître, A AFD
Result: true si le mot est connu, false sinon

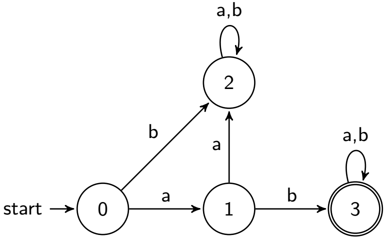
$q \leftarrow q_0$;
$i \leftarrow 1$;
while $i \leqslant n$ **do**
    $q \leftarrow \delta(q, u_i)$;
    $i \leftarrow i + 1$;
end
if $q \in F$ **then**
    return true;
else
    return false;
end

Algorithme de reconnaissance d'un mot avec un AFD

Data: u, A
Result: true, false
$q \leftarrow q_0$;
$i \leftarrow 1$;
while $i \leqslant n$ **do**
    $q \leftarrow \delta(q, u_i)$;
    $i \leftarrow i + 1$;
end
if $q \in F$ **then**
    return true;
else
    return false;
end

Cet algorithme demande exactement $|u|$ étapes de calcul (boucle For).

Problème dans le cas de l'automate (fini, complet, déterministe) suivant qui reconnaît le langage dénoté par $ab(a+b)^\star$



Pas de transition étiquetée par a ou b pour sortir de l'état 2 (état puit).
Automate bloqué !

Etat puits

Il y a stricte équivalence entre les définitions suivantes:

AFD partiel $\Leftrightarrow$ AFD complet avec état puit $\{q_p\}$

Etat puits

Il y a stricte équivalence entre les définitions suivantes:

AFD partiel $\Leftrightarrow$ AFD complet avec état puit $\{q_p\}$

Soit A un AFD partiellement spécifié, on définit A' tel que:

- $Q' = Q \bigcup \{q_p\}$
- $q'_0 = q_0$
- $F' = F$
- $\forall q \in Q, a \in \Sigma, \delta'(q, a) = \begin{cases} \delta(q, a) \text{ si existe} \\ q_p \text{ sinon} \end{cases}$
- $\forall a \in \Sigma, \delta'(q_p, a) = q_p$

Etat puits

Il y a stricte équivalence entre les définitions suivantes:

AFD partiel $\Leftrightarrow$ AFD complet avec état puit $\{q_p\}$

Soit A un AFD partiellement spécifié, on définit A' tel que:

- $Q' = Q \bigcup \{q_p\}$
- $q'_0 = q_0$
- $F' = F$
- $\forall q \in Q, a \in \Sigma, \delta'(q, a) = \begin{cases} \delta(q, a) \text{ si existe} \\ q_p \text{ sinon} \end{cases}$
- $\forall a \in \Sigma, \delta'(q_p, a) = q_p$

L'état puit est celui dans lequel on aboutit dans A' en cas d'échec dans A.
$\rightarrow$ absorbe les transitions interdites.

Etat puits

Il y a stricte équivalence entre les définitions suivantes:

AFD partiel $\Leftrightarrow$ AFD complet avec état puit $\{q_p\}$

Soit A un AFD partiellement spécifié, on définit A' tel que:

- $Q' = Q \bigcup \{q_p\}$
- $q'_0 = q_0$
- $F' = F$
- $\forall q \in Q, a \in \Sigma, \delta'(q,a) = \begin{cases} \delta(q,a) \text{ si existe} \\ q_p \text{ sinon} \end{cases}$
- $\forall a \in \Sigma, \delta'(q_p, a) = q_p$

L'état puit est celui dans lequel on aboutit dans A' en cas d'échec dans A.

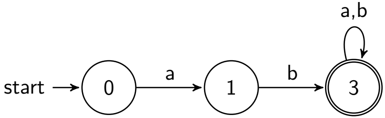
$\rightarrow$ absorbe les transitions interdites.

Une fois dans q_p il est impossible d'atteindre les autres états de A et donc de rejoindre un état final.

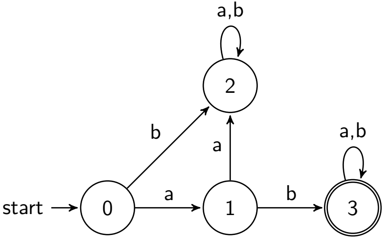
$\rightarrow$ ne possède pas de transitions sortantes.

Etat puits: exemple

Automate partiel A'



Automate complet équivalent A



$L_1(A) = L_2(A) = ab(a+b)^\star \Rightarrow$ les automates sont équivalents.

On montre que: pour tout automate partiel, il existe un automate complet équivalent.
En pratique, il est souvent plus simple de décrire un langage à l'aide d'un automate non-déterministe.

Langage reconnaissable par un AFN

Pour qu'un mot soit reconnu, il suffit qu'un calcul réussisse (même définition que pour les AFD)

- la 1ère transition a pour origine l'état initial et la dernière transition a pour destination un des états finaux.
- plusieurs calculs peuvent mener à un même mot, ces calculs ne sont pas forcément tous réussis.
- pour que le mot soit reconnu, il suffit qu'un des calculs soit réussi.
- La reconnaissance échoue si tous les calculs échouent
- $\rightarrow$ nécessité d'explorer tous les chemins (tant qu'aucun chemin valide n'est été trouvé). Le temps de recherche dépend du nombre de chemins...

Langage reconnaissable par un AFN

Pour qu'un mot soit reconnu, il suffit qu'un calcul réussisse (même définition que pour les AFD)

- la 1ère transition a pour origine l'état initial et la dernière transition a pour destination un des états finaux.
- plusieurs calculs peuvent mener à un même mot, ces calculs ne sont pas forcément tous réussis.
- pour que le mot soit reconnu, il suffit qu'un des calculs soit réussi.
- La reconnaissance échoue si tous les calculs échouent
- $\rightarrow$ nécessité d'explorer tous les chemins (tant qu'aucun chemin valide n'est été trouvé). Le temps de recherche dépend du nombre de chemins...

Langage reconnaissable par un AFN

Pour qu'un mot soit reconnu, il suffit qu'un calcul réussisse (même définition que pour les AFD)

- la 1ère transition a pour origine l'état initial et la dernière transition a pour destination un des états finaux.
- plusieurs calculs peuvent mener à un même mot, ces calculs ne sont pas forcément tous réussis.
- pour que le mot soit reconnu, il suffit qu'un des calculs soit réussi.
- La reconnaissance échoue si tous les calculs échouent
- $\rightarrow$ nécessité d'explorer tous les chemins (tant qu'aucun chemin valide n'est été trouvé). Le temps de recherche dépend du nombre de chemins...

Langage reconnaissable par un AFN

Pour qu'un mot soit reconnu, il suffit qu'un calcul réussisse (même définition que pour les AFD)

- la 1ère transition a pour origine l'état initial et la dernière transition a pour destination un des états finaux.
- plusieurs calculs peuvent mener à un même mot, ces calculs ne sont pas forcément tous réussis.
- pour que le mot soit reconnu, il suffit qu'un des calculs soit réussi.
- La reconnaissance échoue si tous les calculs échouent
- $\rightarrow$ nécessité d'explorer tous les chemins (tant qu'aucun chemin valide n'est été trouvé). Le temps de recherche dépend du nombre de chemins...

Langage reconnaissable par un AFN

Pour qu'un mot soit reconnu, il suffit qu'un calcul réussisse (même définition que pour les AFD)

- la 1ère transition a pour origine l'état initial et la dernière transition a pour destination un des états finaux.
- plusieurs calculs peuvent mener à un même mot, ces calculs ne sont pas forcément tous réussis.
- pour que le mot soit reconnu, il suffit qu'un des calculs soit réussi.
- La reconnaissance échoue si tous les calculs échouent
- $\rightarrow$ nécessité d'explorer tous les chemins (tant qu'aucun chemin valide n'est été trouvé). Le temps de recherche dépend du nombre de chemins...

Langage reconnaissable par un AFN

Pour qu'un mot soit reconnu, il suffit qu'un calcul réussisse (même définition que pour les AFD)

- la 1ère transition a pour origine l'état initial et la dernière transition a pour destination un des états finaux.
- plusieurs calculs peuvent mener à un même mot, ces calculs ne sont pas forcément tous réussis.
- pour que le mot soit reconnu, il suffit qu'un des calculs soit réussi.
- La reconnaissance échoue si tous les calculs échouent
- $\rightarrow$ nécessité d'explorer tous les chemins (tant qu'aucun chemin valide n'est été trouvé). Le temps de recherche dépend du nombre de chemins...

Avec un AFD, il n'y a qu'un seul chemin, l'algorithme de reconnaissance est rapide. C'est pourquoi la déterminisation d'un AFN en AFD est intéressante.

Déterminisation d'un AFN

Théorème:

Pour tout AFN A défini sur Σ, il existe un AFD A' équivalent à A. Si A possède n états, alors A' possède au plus 2^n états.

Soit l'AFN $A = (\Sigma, Q, I, F, \delta)$, on montre que $A' = (\Sigma, 2^Q, \{I\}, F', \delta')$ avec:

- $F' = \{G \subset Q | F \cup G \neq \emptyset\}$
- $\forall G \subset Q, \forall a \in \Sigma, \delta'(G, a) = \bigcup_{q \in G} \delta(q, a)$

Déterminisation d'un AFN

Théorème:

Pour tout AFN A défini sur Σ, il existe un AFD A' équivalent à A. Si A possède n états, alors A' possède au plus 2^n états.

Soit l'AFN $A = (\Sigma, Q, I, F, \delta)$, on montre que $A' = (\Sigma, 2^Q, \{I\}, F', \delta')$ avec:

- $F' = \{G \subset Q | F \cup G \neq \emptyset\}$
- $\forall G \subset Q, \forall a \in \Sigma, \delta'(G, a) = \bigcup_{q \in G} \delta(q, a)$

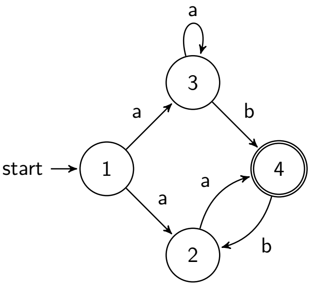
Idée: ne construire que les états accessibles à partir de I (ensemble des états initiaux), de proche en proche. Sinon il faudrait explorer tous les états possibles qui peuvent être nombreux: 2^n avec n le nombre d'états de l'AFN.

Algorithme de déterminisation d'un AFN

Soit un AFN $A = (\Sigma, Q, I, F, \delta)$, et A' tel que définit précédemment, la déterminisation s'effectue avec l'algorithme suivant:

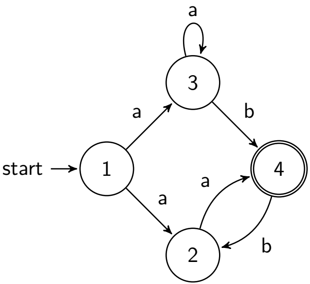
$q'_0 \leftarrow q_0$;
$\delta' \leftarrow \emptyset$;
for $q' \in Q' = 2^Q$ **do**
  for $\sigma \in \Sigma$ **do**
    $p \leftarrow \{y | \exists x \in q', \exists y \in Q | (x, \sigma, y) \in \delta\}$;
    if $p \neq \emptyset$ **then**
      $p \leftarrow p \cup \{y | \exists y \in p, z \in Q | (y, \epsilon, z) \in \delta\}$;
      $\delta' \leftarrow \delta \cup \{(q', \sigma, p)\}$;
      $Q' \leftarrow Q \cup \{p\}$;
    end
  end
end
$F' \leftarrow \{q' | q' \cup F \neq \emptyset\}$;

Algorithme de déterminisation d'un AFN: exemple



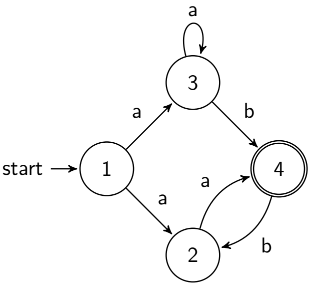
- Automate fini non-déterministe partiel
- $\Sigma = \{a, b\}$
- $I = \{1\}$
- $F = \{4\}$
- $Q = \{1, 2, 3, 4\}$
- δ

Algorithme de déterminisation d'un AFN: exemple



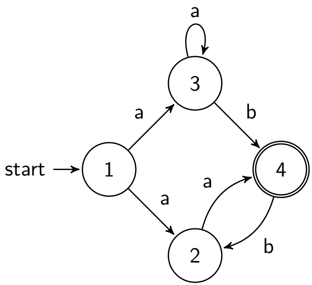
- Automate fini non-déterministe partiel
- $\Sigma = \{a, b\} \Rightarrow \Sigma' = \{a, b\}$
- $I = \{1\}$
- $F = \{4\}$
- $Q = \{1, 2, 3, 4\}$
- δ

Algorithme de déterminisation d'un AFN: exemple



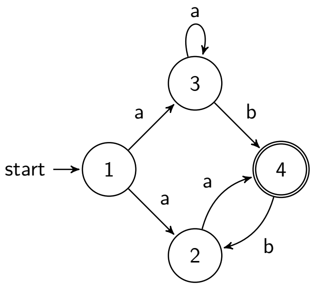
- Automate fini non-déterministe partiel
- $\Sigma = \{a, b\} \Rightarrow \Sigma' = \{a, b\}$
- $I = \{1\} \Rightarrow I' = \{1\}$
- $F = \{4\}$
- $Q = \{1, 2, 3, 4\}$
- δ

Algorithme de déterminisation d'un AFN: exemple



- Automate fini non-déterministe partiel
- $\Sigma = \{a, b\} \Rightarrow \Sigma' = \{a, b\}$
- $I = \{1\} \Rightarrow I' = \{1\}$
- $F = \{4\}$
  $\Rightarrow F' = \{\{4\}, \{1, 4\}, \{2, 4\}, \{3, 4\}, \{1, 2, 4\}, \{1, 3, 4\}, \{2, 3, 4\}\}$
- $Q = \{1, 2, 3, 4\}$
- δ

Algorithme de déterminisation d'un AFN: exemple



- Automate fini non-déterministe partiel
- $\Sigma = \{a, b\} \Rightarrow \Sigma' = \{a, b\}$
- $I = \{1\} \Rightarrow I' = \{1\}$
- $F = \{4\}$
  $\Rightarrow F' = \{\{4\}, \{1, 4\}, \{2, 4\}, \{3, 4\}, \{1, 2, 4\}, \{1, 3, 4\}, \{2, 3, 4\}\}$
- $Q = \{1, 2, 3, 4\} \Rightarrow card(Q') = 2^{card(Q)} = 16$
- δ

Algorithme de déterminisation d'un AFN: exemple

On construit de proche en proche de puis $\{q_0\}$, les états accessibles résultant en général à des automates complets ayant moins que 2^n états.

	δ	a	b
q_0	1	{2,3}	
	2	4	
	3	3	4
q_f	4		2

	δ'	a	b
q_0	$\{1\}$	{2,3}	

Algorithme de déterminisation d'un AFN: exemple

On construit de proche en proche de puis $\{q_0\}$, les états accessibles résultant en général à des automates complets ayant moins que 2^n états.

	δ	a	b
q_0	1	{2,3}	
	2	4	
	3	3	4
q_f	4		2

	δ'	a	b
q_0	{1}	{2,3}	
	{2,3}	{3,4}	{4}

Algorithme de déterminisation d'un AFN: exemple

On construit de proche en proche de puis $\{q_0\}$, les états accessibles résultant en général à des automates complets ayant moins que 2^n états.

	δ	a	b
q_0	1	{2,3}	
	2	4	
	3	3	4
q_f	4		2

	δ'	a	b
q_0	{1}	{2,3}	
	{2,3}	{3,4}	{4}
q_f	{3,4}	{3}	{2,4}

Algorithme de déterminisation d'un AFN: exemple

On construit de proche en proche de puis $\{q_0\}$, les états accessibles résultant en général à des automates complets ayant moins que 2^n états.

	δ	a	b
q_0	1	{2,3}	
	2	4	
	3	3	4
q_f	4		2

	δ'	a	b
q_0	{1}	{2,3}	
	{2,3}	{3,4}	{4}
q_f	{3,4}	{3}	{2,4}
q_f	{4}		{2}

Algorithme de déterminisation d'un AFN: exemple

On construit de proche en proche de puis $\{q_0\}$, les états accessibles résultant en général à des automates complets ayant moins que 2^n états.

	δ	a	b
q_0	1	{2,3}	
	2	4	
	3	3	4
q_f	4		2

	δ'	a	b
q_0	{1}	{2,3}	
	{2,3}	{3,4}	{4}
q_f	{3,4}	{3}	{2,4}
q_f	{4}		{2}
	{3}	{3}	{4}

Algorithme de déterminisation d'un AFN: exemple

On construit de proche en proche de puis $\{q_0\}$, les états accessibles résultant en général à des automates complets ayant moins que 2^n états.

	δ	a	b
q_0	1	{2,3}	
	2	4	
	3	3	4
q_f	4		2

	δ'	a	b
q_0	{1}	{2,3}	
	{2,3}	{3,4}	{4}
q_f	{3,4}	{3}	{2,4}
q_f	{4}		{2}
	{3}	{3}	{4}
q_f	{2,4}	{4}	{2}

Algorithme de déterminisation d'un AFN: exemple

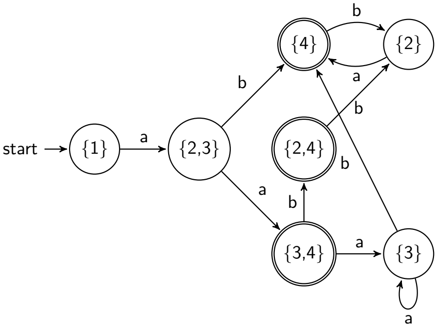
On construit de proche en proche de puis $\{q_0\}$, les états accessibles résultant en général à des automates complets ayant moins que 2^n états.

	δ	a	b
q_0	1	{2,3}	
	2	4	
	3	3	4
q_f	4		2

	δ'	a	b
q_0	{1}	{2,3}	
	{2,3}	{3,4}	{4}
q_f	{3,4}	{3}	{2,4}
q_f	{4}		{2}
	{3}	{3}	{4}
q_f	{2,4}	{4}	{2}
	{2}	{4}	

Algorithme de déterminisation d'un AFN: exemple

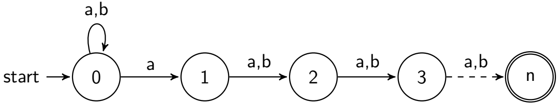
Si on détermine l'ensemble des états possibles (soit 16 états), on obtient la fonction de transition ci dessous. Celle-ce contient plusieurs états non accessibles.



δ'	a	b
{1}	{2,3}	
{2}	{4}	
{3}	{3}	{4}
{4}		{2}
{1,2}	{2,3,4}	
{1,3}	{2,3}	{4}
{1,4}	{2,3}	{2}
{2,3}	{3,4}	{4}
{2,4}	{4}	{4}
{3,4}	{3}	{2,4}
{1,2,3}	{2,3,4}	{4}
{1,2,4}	{2,3,4}	{2}
{1,3,4}	{2,3}	{2,4}
{2,3,4}	{3,4}	{2,4}
{1,2,3,4}	{2,3,4}	{2,4}

On ne représente que les états utiles (ainsi {1,2} n'est pas représenté).

Déterminisation: autre exemple



δ'	a	b
{0}	{0,1}	{0}
{1}	{2}	{2}
...	...	...
{n-1}	{n}	{n}
{0,1}	{0,1}	{0}
{1,2}	{2,3}	{2,3}
{1,4}	{2,3}	{2}
...	...	...
$\{0, 1, 2, \ldots, n-1\}$	$\{0, 1, 2, \ldots, n\}$	$\{0, 1, 2, \ldots, n\}$

Explosion combinatoire: 2^n états possibles. Si $n = 10$, 1024 états possibles !!

Les AFN sont plus faciles à construire et fournissent des machines bien plus "compactes" que les AFD.

AFN à transitions spontanées: définition

Un AFN à transitions spontanées (ϵ-AFN) est défini par un quintuplet $A = (\Sigma, Q, I, F, \delta)$, où:

- Σ est un alphabet (ensemble fini de symboles)
- Q est un ensemble fini d'états ($Q \neq \emptyset$)
- $I \subset Q$ est l'ensemble des états initiaux
- $F \subset Q$ est l'ensemble des états finaux (éventuellement $Q = \emptyset$)
- $\delta \subset Q \times (\Sigma \cup \{\epsilon\}) \times Q$ est la relation de transition

AFN à transitions spontanées: définition

Un AFN à transitions spontanées (ϵ-AFN) est défini par un quintuplet $A = (\Sigma, Q, I, F, \delta)$, où:

- Σ est un alphabet (ensemble fini de symboles)
- Q est un ensemble fini d'états ($Q \neq \emptyset$)
- $I \subset Q$ est l'ensemble des états initiaux
- $F \subset Q$ est l'ensemble des états finaux (éventuellement $Q = \emptyset$)
- $\delta \subset Q \times (\Sigma \cup \{\epsilon\}) \times Q$ est la relation de transition

Transition spontanée: transition étiquetée par le mot vide ϵ. Ce cas de figure est impossible pour des AFD.

Un ϵ-AFN peut changer d'état sans consommer de symbole.

Transitions spontanées

<u>Définition:</u>
Soit A un AFN et $q \in Q$. On appelle ϵ-fermeture de q l'ensemble $\epsilon-$*closure* des états p tels qu'il existe une transition spontanée entre q et p.

<u>Théorème:</u>
Pour tout ϵ-AFN, il existe un AFN équivalent (qui reconnaît le même langage).

Transitions spontanées

Définition:
Soit A un AFN et $q \in Q$. On appelle ϵ-fermeture de q l'ensemble $\epsilon-closure$ des états p tels qu'il existe une transition spontanée entre q et p.

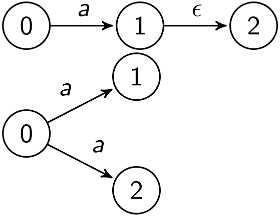
Théorème:
Pour tout ϵ-AFN, il existe un AFN équivalent (qui reconnaît le même langage).

Soit un AFN $A = (\Sigma \cup \{\epsilon\}, Q, q_0, F, \delta)$, on définit $A'(\Sigma, Q, q_0, F', \delta')$ comme suit:

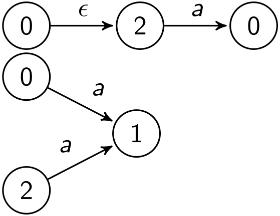
- $F' = \{q \in Q | \epsilon-closure(q) \cap F \neq \emptyset\}$
- $\delta'(q, a) = \bigcup_{p \in \epsilon-closure(q)} \delta(p, a)$

Suppression des transitions spontanées

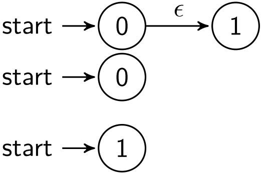
Fermeture avant:



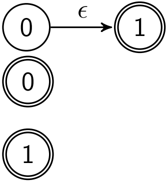
Fermeture arrière:



Fermeture avant avec état initial:

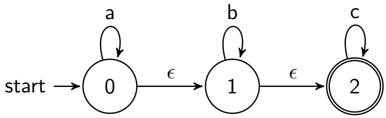


Fermeture arrière avec état final:



Transitions spontanées: exemple

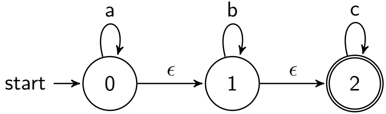
Un ϵ-AFN



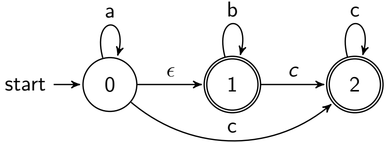
- $\Sigma = \{a, b, c\}$
- $Q = \{q_0, q_1, q_2\}$
- $F = \{q_2\}$

Transitions spontanées: exemple

Un ϵ-AFN



Première étape



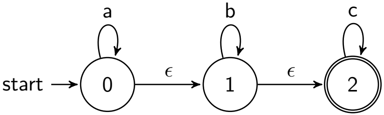
- $\Sigma = \{a, b, c\}$
- $Q = \{q_0, q_1, q_2\}$
- $F = \{q_2\}$

Première étape

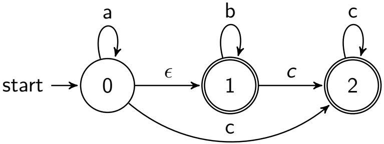
- fermeture arrière avec état final $\rightarrow q_1 \in F'$
- fermeture arrière $(1, \epsilon, 2)(2, c, 2)$, on ajoute $(1, c, 2)$
- fermeture arrière $(0, \epsilon, 1)(1, \epsilon, 2)(2, c, 2)$, on ajoute $(0, c, 2)$

Transitions spontanées: exemple

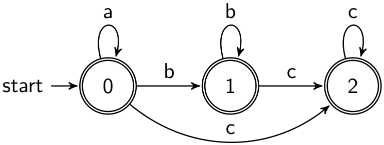
Un ϵ-AFN



Première étape



Un ϵ-AFN sans transitions spontanées



- $\Sigma = \{a, b, c\}$
- $Q = \{q_0, q_1, q_2\}$
- $F = \{q_2\}$

Première étape

- fermeture arrière avec état final $\rightarrow q_1 \in F'$
- fermeture arrière $(1, \epsilon, 2)(2, c, 2)$, on ajoute $(1, c, 2)$
- fermeture arrière $(0, \epsilon, 1)(1, \epsilon, 2)(2, c, 2)$, on ajoute $(0, c, 2)$

Deuxième étape

- fermeture arrière avec état final $\rightarrow q_0 \in F'$
- fermeture arrière $(0, \epsilon, 1)(1, b, 1)$, on ajoute $(0, b, 1)$

PLAN DE LA SECTION ACTUELLE

Théorème de Kleene

Définition (rappel): Un langage est reconnaissable si il existe un automate fini qui le reconnaît.

Théorème de Kleene: Un langage est reconnaissable si et seulement si il est régulier (dénoté par une expression régulière).

Corollaire: Un langage est régulier si et seulement si il est défini par un automate fini.

Théorème de Kleene

Définition (rappel): Un langage est reconnaissable si il existe un automate fini qui le reconnaît.

Théorème de Kleene: Un langage est reconnaissable si et seulement si il est régulier (dénoté par une expression régulière).

Corollaire: Un langage est régulier si et seulement si il est défini par un automate fini.

Des expressions régulières vers les automates

- Algorithme de Thompson (cf plus loin)
- Algorithme de Glushkov (ne contient pas de transitions spontanées)

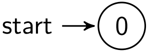
Des automates vers les expressions régulières

- Alogorithme de Mac Naughton & Yamada
- Lemme d'Arden

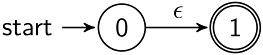
Cas particuliers

Les langages suivants sont réguliers car ils dénotent une expression régulière (cf partie sur les expressions régulières), ils sont aussi reconnaissables car il existe un automate A qui les reconnaît.

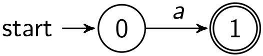
$L(A) = \emptyset$
A n'a aucune chaîne.



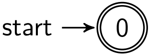
$L(A) = \{\epsilon\}$.
A a une chaîne, la chaîne vide.



$L(A) = \{a\}$.
A a une chaîne, la chaîne a.



Autre représentation pour le langage $\{\epsilon\}$.



Union

Définition (rappel): $L \cup M = \{x \in \Sigma^\star | x \in L \text{ ou } x \in M\}$

Théorème:
L'union de deux langages réguliers est un langage régulier.

$$L(A_1) \cup L(A_2) = L(A_1 + A_2) \Leftrightarrow L(e_1) \cup L(e_2) = L(e_1 + e_2)$$

Union

Définition (rappel): $L \cup M = \{x \in \Sigma^\star | x \in L \text{ ou } x \in M\}$

Théorème:
L'union de deux langages réguliers est un langage régulier.

$$L(A_1) \cup L(A_2) = L(A_1 + A_2) \Leftrightarrow L(e_1) \cup L(e_2) = L(e_1 + e_2)$$

Démonstration:
Soit L_1 et L_2 2 langages réguliers, reconnus respectivement par les AFD complets A_1 et A_2. On construit un automate $A = (\Sigma, Q = Q_1 \times Q_2, q_0, F, \delta)$ pour $L_1 \cup L_2$ tel que:

- $q_0 = (q_0^1, q_0^2)$
- $F = (F_1 \times Q_2) \cup (Q_1 \times F_2)$
- $\delta((q_1, q_2), a) = (\delta_1(q_1, a), \delta_2(q_2, a))$

Union

Définition (rappel): $L \cup M = \{x \in \Sigma^\star | x \in L \text{ ou } x \in M\}$

Théorème:
L'union de deux langages réguliers est un langage régulier.

$$L(A_1) \cup L(A_2) = L(A_1 + A_2) \Leftrightarrow L(e_1) \cup L(e_2) = L(e_1 + e_2)$$

Démonstration:
Soit L_1 et L_2 2 langages réguliers, reconnus respectivement par les AFD complets A_1 et A_2. On construit un automate $A = (\Sigma, Q = Q_1 \times Q_2, q_0, F, \delta)$ pour $L_1 \cup L_2$ tel que:

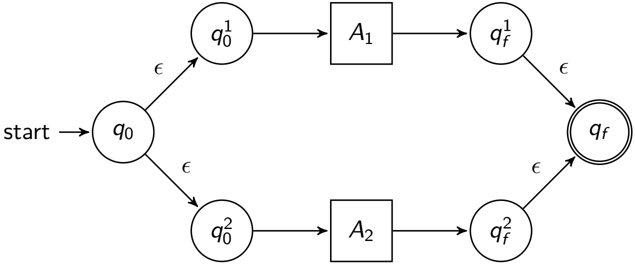
- $q_0 = (q_0^1, q_0^2)$
- $F = (F_1 \times Q_2) \cup (Q_1 \times F_2)$
- $\delta((q_1, q_2), a) = (\delta_1(q_1, a), \delta_2(q_2, a))$

A_1 et A_2 fonctionnent en parallèle. De part la construction de F, un calcul réussi de A le sera soit dans A_1 soit dans A_2.

Union: représentation avec un ϵ-AFN

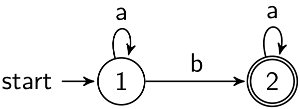
A_1 (resp. A_2) reconnaît l'expression régulière e_1 (resp. e_2)

Union: $e_1 + e_2 = L(A_1) + L(A_2) = L(A_1 \cup A_2)$

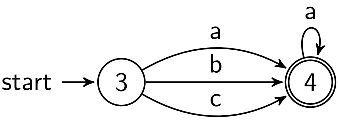


$\rightarrow$ Mise en parallèle !

Union: exemple



δ_1	a	b	c
1	1	2	x
2	2	x	x



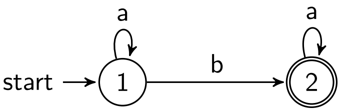
δ_2	a	b	c
3	4	4	4
4	4	x	x

$F = (F_1 \times Q_2) \cup (Q_1 \times F_2) = \{(2,3),(2,4),(2,0)\} \cup \{(1,4),(2,4),(0,4)\}$

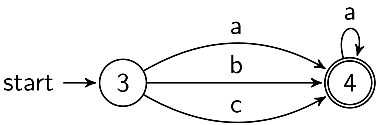
$q_0 = (1,3)$

$\delta((q_1,q_2),a) = (\delta_1(q_1,a),\delta_2(q_2,a))$

Union: exemple



δ_1	a	b	c
1	1	2	x
2	2	x	x

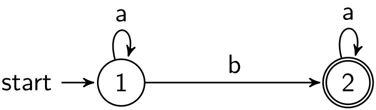


δ_2	a	b	c
3	4	4	4
4	4	x	x

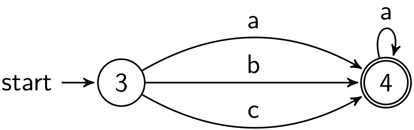
$F = (F_1 \times Q_2) \cup (Q_1 \times F_2)$
$= \{(2,3),(2,4),(2,0)\} \cup \{(1,4),(2,4),(0,4)\}$
$q_0 = (1,3)$
$\delta((q_1,q_2),a) = (\delta_1(q_1,a),\delta_2(q_2,a))$

δ_2	a	b	c
$(1,3) \in I$	(1,4)	(2,4)	(4)
(1)	(1)	(2)	
$(2) \in F$	(2)		
(3)	(4)	(4)	(4)
$(4) \in F$	(4)		
$(1,4) \in F$	(1,4)	(2)	
$(2,3) \in F$	(2,4)	(4)	(4)
$(2,4) \in F$	(2,4)		

Union: exemple



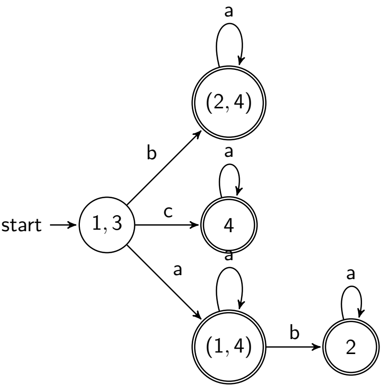
δ_1	a	b	c
1	1	2	x
2	2	x	x



δ_2	a	b	c
3	4	4	4
4	4	x	x

$F = (F_1 \times Q_2) \cup (Q_1 \times F_2)$
$= \{(2,3),(2,4),(2,0)\} \cup \{(1,4),(2,4),(0,4)\}$
$q_0 = (1,3)$
$\delta((q_1,q_2),a) = (\delta_1(q_1,a),\delta_2(q_2,a))$

δ_2	a	b	c
$(1,3) \in I$	(1,4)	(2,4)	(4)
(1)	(1)	(2)	
$(2) \in F$	(2)		
(3)	(4)	(4)	(4)
$(4) \in F$	(4)		
$(1,4) \in F$	(1,4)	(2)	
$(2,3) \in F$	(2,4)	(4)	(4)
$(2,4) \in F$	(2,4)		



Intersection

Définition: (rappel) $L \cap M = \{x \in \Sigma^\star | x \in L \text{ et } x \in M\}$

Théorème:
L'intersection de deux langages réguliers est un langage régulier.

$$L(A_1) \cap L(A_2) = L(A_1 \times A_2) \Leftrightarrow L(e_1) \cap L(e_2) = L(e_1 e_2)$$

Intersection

Définition: (rappel) $L \cap M = \{x \in \Sigma^\star | x \in L \text{ et } x \in M\}$

Théorème:
L'intersection de deux langages réguliers est un langage régulier.

$$L(A_1) \cap L(A_2) = L(A_1 \times A_2) \Leftrightarrow L(e_1) \cap L(e_2) = L(e_1 e_2)$$

Démonstration:
Soit L_1 et L_2 2 langages réguliers, reconnus respectivement par les AFD complets A_1 et A_2. On construit un automate $A = (\Sigma, Q = Q_1 \times Q_2, q_0, F, \delta)$ pour $L_1 \cap L_2$ tel que:

- $q_0 = (q_0^1, q_0^2)$
- $F = (F_1, F_2)$
- $\delta((q_1, q_2), a) = (\delta_1(q_1, a), \delta_2(q_2, a))$

Intersection

Définition: (rappel) $L \cap M = \{x \in \Sigma^\star | x \in L \text{ et } x \in M\}$

Théorème:
L'intersection de deux langages réguliers est un langage régulier.

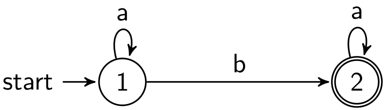
$$L(A_1) \cap L(A_2) = L(A_1 \times A_2) \Leftrightarrow L(e_1) \cap L(e_2) = L(e_1 e_2)$$

Démonstration:
Soit L_1 et L_2 2 langages réguliers, reconnus respectivement par les AFD complets A_1 et A_2. On construit un automate $A = (\Sigma, Q = Q_1 \times Q_2, q_0, F, \delta)$ pour $L_1 \cap L_2$ tel que:

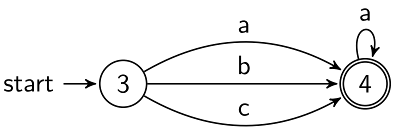
- $q_0 = (q_0^1, q_0^2)$
- $F = (F_1, F_2)$
- $\delta((q_1, q_2), a) = (\delta_1(q_1, a), \delta_2(q_2, a))$

De part la construction de F, un calcul réussi de A le sera dans A_1 et dans A_2.

Intersection: exemple



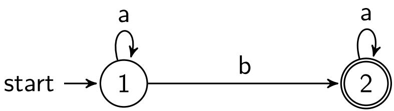
δ_1	a	b	c
1	1	2	x
2	2	x	x



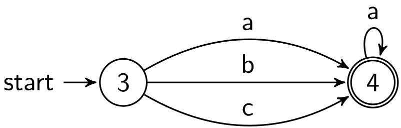
δ_2	a	b	c
3	4	4	4
4	4	x	x

$F = \{(2,4)\}$ et $q_0 = (1,3)$
$\delta((q_1,q_2),a) = (\delta_1(q_1,a), \delta_2(q_2,a))$

Intersection: exemple



δ_1	a	b	c
1	1	2	x
2	2	x	x



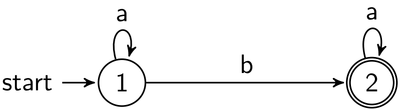
δ_2	a	b	c
3	4	4	4
4	4	x	x

$F = \{(2,4)\}$ et $q_0 = (1,3)$

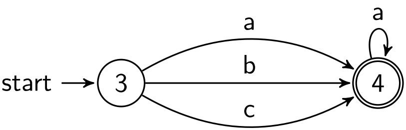
$\delta((q_1, q_2), a) = (\delta_1(q_1, a), \delta_2(q_2, a))$

δ_2	a	b	c
$(1,3) \in I$	(1,4)	(2,4)	(4)
(1)	(1)	(2)	
(2)	(2)		
(3)	(4)	(4)	(4)
(4)	(4)		
(1,4)	(1,4)	(2)	
(2,3)	(2,4)	(4)	(4)
$(2,4) \in F$	(2,4)		

Intersection: exemple



δ_1	a	b	c
1	1	2	x
2	2	x	x

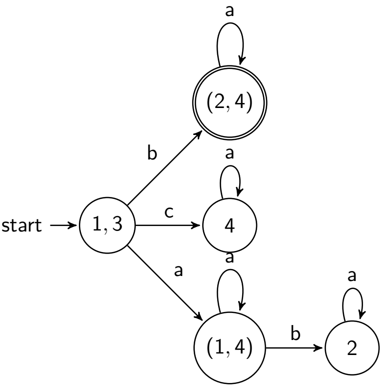


δ_2	a	b	c
3	4	4	4
4	4	x	x

$F = \{(2,4)\}$ et $q_0 = (1,3)$

$\delta((q_1, q_2), a) = (\delta_1(q_1, a), \delta_2(q_2, a))$

δ_2	a	b	c
$(1,3) \in I$	(1,4)	(2,4)	(4)
(1)	(1)	(2)	
(2)	(2)		
(3)	(4)	(4)	(4)
(4)	(4)		
(1,4)	(1,4)	(2)	
(2,3)	(2,4)	(4)	(4)
$(2,4) \in F$	(2,4)		



Complémentaire

Définition (rappel): $\overline{L} = \{x \in \Sigma^\star | x \notin L\}$

Théorème:
Le complémentaire d'un langage régulier est un langage régulier.

Complémentaire

Définition (rappel): $\overline{L} = \{x \in \Sigma^\star | x \notin L\}$

Théorème:
Le complémentaire d'un langage régulier est un langage régulier.

Démonstration:
Soit L un langage régulier reconnu respectivement par l'AFD complet A.
On construit un automate $A = (\Sigma, Q, q_0, Q \setminus F, \delta)$

Complémentaire

Définition (rappel): $\overline{L} = \{x \in \Sigma^\star | x \notin L\}$

Théorème:
Le complémentaire d'un langage régulier est un langage régulier.

Démonstration:
Soit L un langage régulier reconnu respectivement par l'AFD complet A.
On construit un automate $A = (\Sigma, Q, q_0, Q \setminus F, \delta)$

De part sa construction, un calcul réussi de A échouera dans $\overline{A}$ et réciproquement.

Concaténation

Théorème:
La concaténation de deux langages réguliers est un langage régulier.

$$L(A_1) \cup L(A_2) = L(A_1 + A_2)$$

Concaténation

Théorème:

La concaténation de deux langages réguliers est un langage régulier.

$$L(A_1) \cup L(A_2) = L(A_1 + A_2)$$

Démonstration:

Soit L_1 et L_2 2 langages réguliers, reconnus respectivement par les AFD complets A_1 et A_2. On suppose que Q_1 et Q_2 sont disjoints ($Q_1 \cap Q_2 = \emptyset$) et que A_1 a un unique état final ($F = \{q_f\}$). On construit un automate A tel que l'état final de A_1 s'identifie avec l'état initial de A_2.

- $Q = Q_1 \cup Q_2 \setminus \{q_0^2\}$
- $q_0 = q_0^1$
- $F = F_2$
- $\delta(q, a) = \begin{cases} \delta_1(q, a) \text{ si } q \in Q_1, q \neq q_f \\ \delta_2(q, a) \text{ si } q \in Q_2 \\ \delta_1(q_f, a) \cup \delta_2(q_0^2, a) \text{ si } q = q_f \end{cases}$

Concaténation

Théorème:

La concaténation de deux langages réguliers est un langage régulier.

$$L(A_1) \cup L(A_2) = L(A_1 + A_2)$$

Démonstration:

Soit L_1 et L_2 2 langages réguliers, reconnus respectivement par les AFD complets A_1 et A_2. On suppose que Q_1 et Q_2 sont disjoints ($Q_1 \cap Q_2 = \emptyset$) et que A_1 a un unique état final ($F = \{q_f\}$). On construit un automate A tel que l'état final de A_1 s'identifie avec l'état initial de A_2.

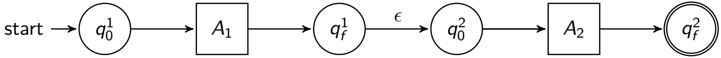
- $Q = Q_1 \cup Q_2 \setminus \{q_0^2\}$
- $q_0 = q_0^1$
- $F = F_2$
- $\delta(q, a) = \begin{cases} \delta_1(q, a) \text{ si } q \in Q_1, q \neq q_f \\ \delta_2(q, a) \text{ si } q \in Q_2 \\ \delta_1(q_f, a) \cup \delta_2(q_0^2, a) \text{ si } q = q_f \end{cases}$

De part la construction de F, un calcul réussi de A atteint nécessairement un état final de A_1 puis un état final de A_2.

Concaténation: représentation avec un ϵ-AFN

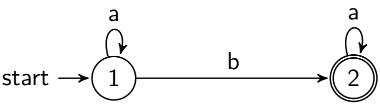
A_1 (resp. A_2) reconnaît l'expression régulière e_1 (resp. e_2)

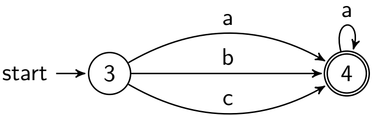
Concaténation: $e_1 e_2 = L(A_1)L(A_2) = L(A_1 + A_2)$



$\rightarrow$ Mise en série !

Concaténation: exemple



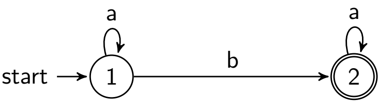


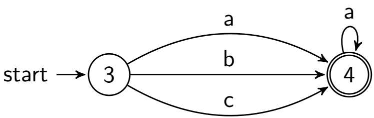
δ_1	a	b	c
1	1	2	x
2	2	x	x

δ_2	a	b	c
3	4	4	4
4	4	x	x

$q_f = 4$ et $q_0 = 1$
$Q = \{1, 2\} \cup \{3, 4\} \setminus \{3\}$

Concaténation: exemple





δ_1	a	b	c
1	1	2	x
2	2	x	x

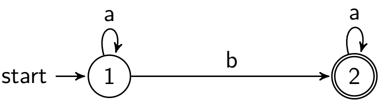
δ_2	a	b	c
3	4	4	4
4	4	x	x

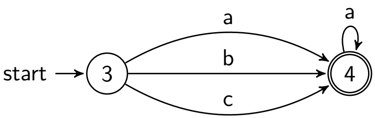
$q_f = 4$ et $q_0 = 1$
$Q = \{1, 2\} \cup \{3, 4\} \setminus \{3\}$

δ_2	a	b	c
$1 \in I$	1	2	x
$(2, 3)$	2,4	4	4
$4 \in F$	x	4	4

NB: si $3 \in F$ alors l'état (2,3) aurait aussi été final.

Concaténation: exemple





δ_1	a	b	c
1	1	2	x
2	2	x	x

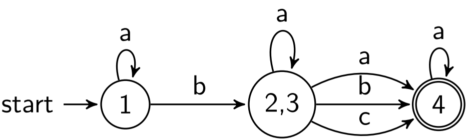
δ_2	a	b	c
3	4	4	4
4	4	x	x

$q_f = 4$ et $q_0 = 1$

$Q = \{1, 2\} \cup \{3, 4\} \setminus \{3\}$

δ_2	a	b	c
$1 \in I$	1	2	x
$(2, 3)$	2,4	4	4
$4 \in F$	x	4	4

NB: si $3 \in F$ alors l'état (2,3) aurait aussi été final.



Fermeture de Kleene

La fermeture de Kleene (ou étoile de Kleene) d'un langage L se définit par:

$$L^\star = \bigcup_{i \geq 0} L^i$$

- $L^\star$ contient tous les mots qu'il est possible de construire en concaténant un nombre fini d'éléments de langage L
- ex: $L = \{a, b\}$ alors $L^\star = \{\epsilon, a, b, aa, ab, bb, aaa, aab, \ldots\}$
- $\emptyset^\star \neq \emptyset$: il contient ϵ

On définit aussi la fermeture stricte:

$$L^+ = \bigcup_{i > 0} L^i = LL^\star$$

- Attention, contrairement à $L^\star$, L^+ ne contient pas forcément ϵ

Fermeture de Kleene

Théorème:
La mise à l'étoile d'un langage régulier est un langage régulier.

Démonstration:
Soit L un langage régulier, reconnu par l'AFD complet A.

- On suppose que A n'a qu'un état initial q_0.
- Pour obtenir l'automate correspondant à $L^\star$, A', il suffit de rajouter une transition spontanée depuis tout état final de A vers l'état initial q_0.

$$q_f^A \xrightarrow{\epsilon} q_0^A$$

- Cette nouvelle transition permet l'itération dans A' de mots de L.
- On vérifie que $\epsilon \in L(A)$ si ce n'est pas le cas, l'état initial de A sera état final de A', sinon il faut ajouter une relation:

$$q_0^{A'} \xrightarrow{\epsilon} q_f^{A'}$$

Fermeture de Kleene

Théorème:
La mise à l'étoile d'un langage régulier est un langage régulier.

Démonstration:
Soit L un langage régulier, reconnu par l'AFD complet A.

- On suppose que A n'a qu'un état initial q_0.
- Pour obtenir l'automate correspondant à $L^\star$, A', il suffit de rajouter une transition spontanée depuis tout état final de A vers l'état initial q_0.

$$q_f^A \xrightarrow{\epsilon} q_0^A$$

- Cette nouvelle transition permet l'itération dans A' de mots de L.
- On vérifie que $\epsilon \in L(A)$ si ce n'est pas le cas, l'état initial de A sera état final de A', sinon il faut ajouter une relation:

$$q_0^{A'} \xrightarrow{\epsilon} q_f^{A'}$$

Fermeture de Kleene

Théorème:
La mise à l'étoile d'un langage régulier est un langage régulier.

Démonstration:
Soit L un langage régulier, reconnu par l'AFD complet A.

- On suppose que A n'a qu'un état initial q_0.
- Pour obtenir l'automate correspondant à $L^\star$, A', il suffit de rajouter une transition spontanée depuis tout état final de A vers l'état initial q_0.

$$q_f^A \xrightarrow{\epsilon} q_0^A$$

- Cette nouvelle transition permet l'itération dans A' de mots de L.
- On vérifie que $\epsilon \in L(A)$ si ce n'est pas le cas, l'état initial de A sera état final de A', sinon il faut ajouter une relation:

$$q_0^{A'} \xrightarrow{\epsilon} q_f^{A'}$$

Fermeture de Kleene

Théorème:
La mise à l'étoile d'un langage régulier est un langage régulier.

Démonstration:
Soit L un langage régulier, reconnu par l'AFD complet A.

- On suppose que A n'a qu'un état initial q_0.
- Pour obtenir l'automate correspondant à $L^\star$, A', il suffit de rajouter une transition spontanée depuis tout état final de A vers l'état initial q_0.

$$q_f^A \xrightarrow{\epsilon} q_0^A$$

- Cette nouvelle transition permet l'itération dans A' de mots de L.
- On vérifie que $\epsilon \in L(A)$ si ce n'est pas le cas, l'état initial de A sera état final de A', sinon il faut ajouter une relation:

$$q_0^{A'} \xrightarrow{\epsilon} q_f^{A'}$$

Fermeture de Kleene

Théorème:
La mise à l'étoile d'un langage régulier est un langage régulier.

Démonstration:
Soit L un langage régulier, reconnu par l'AFD complet A.

- On suppose que A n'a qu'un état initial q_0.
- Pour obtenir l'automate correspondant à $L^\star$, A', il suffit de rajouter une transition spontanée depuis tout état final de A vers l'état initial q_0.

$$q_f^A \xrightarrow{\epsilon} q_0^A$$

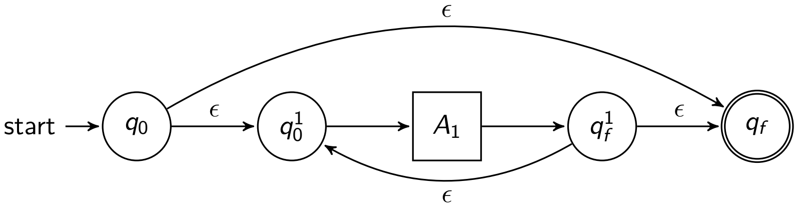
- Cette nouvelle transition permet l'itération dans A' de mots de L.
- On vérifie que $\epsilon \in L(A)$ si ce n'est pas le cas, l'état initial de A sera état final de A', sinon il faut ajouter une relation:

$$q_0^{A'} \xrightarrow{\epsilon} q_f^{A'}$$

Représentation des opérations avec les ϵ transitions

A_1 (resp. A_2) reconnaît l'expression régulière e_1 (resp. e_2)

Etoile de Kleene: $e_1^\star = L(A_1)^\star$



$\rightarrow$ Fermeture de l'état final sur l'état initial + un arc reconnaissant ϵ

Combinaisons d'automates et expressions régulières

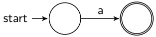
A partir des constructions simples vu précédement, il est possible de dériver un algorithme permettant de construire un automate reconnaissant le langage dénoté par une expression régulière quelconque : il suffit de décomposer l'expression en ses composants élémentaires, puis d'appliquer les constructions précédentes pour construire l'automate correspondant.
Cet algorithme est connu sous le nom d'algorithme de Thompson.

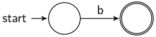
Par exemple, cherchons à construire l'automate reconnaissant le langage dénoté par l'expression régulière définie sur $\Sigma = \{a, b\}$:

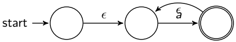
$$e = a^* b + b^* a$$

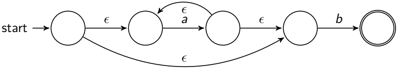
Combinaisons d'automates et expressions régulières

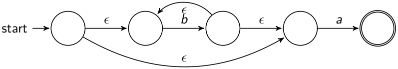
$$e = a^*b + b^*a$$

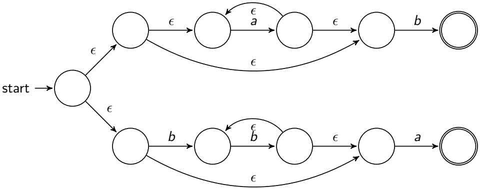
- AF pour a: 

- AF pour b: 

- AF pour a^*: 

- AF pour a^*b: 

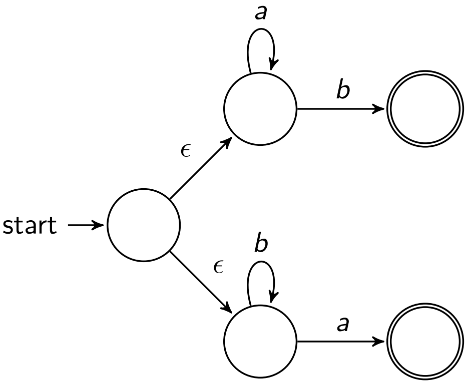
- AF pour b^*a: 

- AF pour $a^*b + b^*a$: 

Combinaisons d'automates et expressions régulières

$$e = a^* b + b^* a$$

équivalent à:



Conclusion

Les automates finis à états et les expressions régulières sont utilisés dans différents contextes:

- description de traitement textuels, ex: recherche de sous-chaîne (traitement de textes, langages de script, awk, perl, python, ...)
- description des lexèmes d'un langage dans un compilateur ou un interpréteur, ex: langage de programmation, de script, d'interrogation d'une base de donnée
- description de l'étage morpho-lexical des langues naturelles, ex: dictionnaires, lexiques
- description d'un flot de contrôle d'un programme séquentiel
- etc.

Conclusion

Les propriétés des automates finis sont intéressantes.

- Grâce à la déterminisation et la minimisation, toute description sous forme d'automate fini peut être exécutée efficacement.
- Les opérations ensemblistes, la concaténation et l'étoile ouvrent la voie à la modularité et au traitement d'exceptions.
- Des boîtes à outil existent et permettent de décrire et d'exécuter de très gros automates (plusieurs millions d'états et de transitions).

Conclusion

Les propriétés des automates finis sont intéressantes.

- Grâce à la déterminisation et la minimisation, toute description sous forme d'automate fini peut être exécutée efficacement.
- Les opérations ensemblistes, la concaténation et l'étoile ouvrent la voie à la modularité et au traitement d'exceptions.
- Des boîtes à outil existent et permettent de décrire et d'exécuter de très gros automates (plusieurs millions d'états et de transitions).

Les automates finis à états ont une puissance limitée:

- leur mémoire est limitée par les états qui sont en nombre fini
- ils n'ont pas la puissance des machines de Turing (abordée en L3 ?)
- par exemple ils ne peuvent pas décrire les chaînes qui ont le même nombre de a que de b.
- etc.

Conclusion

Il existe plusieurs formalismes qui augmentent le pouvoir descriptif des automates finis. Cela permet de représenter des langages non réguliers. Cela augmente la puissance, en contrepartie, certaines bonnes propriétés sont perdues.

- Les transducteurs finis sont utilisés pour décrire des relations régulières ou des fonctions de traduction. Sur chaque transition il y a deux symboles: un en lecture, l'autre en écriture.
- Les automates à pile: une pile de taille illimitée est ajoutée à un automate fini. Chaque transition lit un symbole de chaîne et réalise une opération de pile (empilement, dépilement, lecture du sommet). Ce dispositif décrit des langages non contextuels qui sont utilisés pour décrire la syntaxe des langages informatiques et parfois aussi des langes naturelles.
- Les réseaux de Petri: sur la base d'un graphe orienté, on ajoute la notion de jetons qui peuvent se déplacer dans le graphe, être créés ou éliminés. Des contraintes spécifiques limitent le nombre de jetons dans certains nœuds du graphe.

Conclusion

Il existe plusieurs formalismes qui augmentent le pouvoir descriptif des automates finis. Cela permet de représenter des langages non réguliers. Cela augmente la puissance, en contrepartie, certaines bonnes propriétés sont perdues.

- Les transducteurs finis sont utilisés pour décrire des relations régulières ou des fonctions de traduction. Sur chaque transition il y a deux symboles: un en lecture, l'autre en écriture.
- Les automates à pile: une pile de taille illimitée est ajoutée à un automate fini. Chaque transition lit un symbole de chaîne et réalise une opération de pile (empilement, dépilement, lecture du sommet). Ce dispositif décrit des langages non contextuels qui sont utilisés pour décrire la syntaxe des langages informatiques et parfois aussi des langes naturelles.
- Les réseaux de Petri: sur la base d'un graphe orienté, on ajoute la notion de jetons qui peuvent se déplacer dans le graphe, être créés ou éliminés. Des contraintes spécifiques limitent le nombre de jetons dans certains nœuds du graphe.

Conclusion

Il existe plusieurs formalismes qui augmentent le pouvoir descriptif des automates finis. Cela permet de représenter des langages non réguliers. Cela augmente la puissance, en contrepartie, certaines bonnes propriétés sont perdues.

- Les transducteurs finis sont utilisés pour décrire des relations régulières ou des fonctions de traduction. Sur chaque transition il y a deux symboles: un en lecture, l'autre en écriture.
- Les automates à pile: une pile de taille illimitée est ajoutée à un automate fini. Chaque transition lit un symbole de chaîne et réalise une opération de pile (empilement, dépilement, lecture du sommet). Ce dispositif décrit des langages non contextuels qui sont utilisés pour décrire la syntaxe des langages informatiques et parfois aussi des langes naturelles.
- Les réseaux de Petri: sur la base d'un graphe orienté, on ajoute la notion de jetons qui peuvent se déplacer dans le graphe, être créés ou éliminés. Des contraintes spécifiques limitent le nombre de jetons dans certains nœuds du graphe.

Conclusion

Il existe plusieurs formalismes qui augmentent le pouvoir descriptif des automates finis. Cela permet de représenter des langages non réguliers. Cela augmente la puissance, en contrepartie, certaines bonnes propriétés sont perdues.

- Les transducteurs finis sont utilisés pour décrire des relations régulières ou des fonctions de traduction. Sur chaque transition il y a deux symboles: un en lecture, l'autre en écriture.
- Les automates à pile: une pile de taille illimitée est ajoutée à un automate fini. Chaque transition lit un symbole de chaîne et réalise une opération de pile (empilement, dépilement, lecture du sommet). Ce dispositif décrit des langages non contextuels qui sont utilisés pour décrire la syntaxe des langages informatiques et parfois aussi des langes naturelles.
- Les réseaux de Petri: sur la base d'un graphe orienté, on ajoute la notion de jetons qui peuvent se déplacer dans le graphe, être créés ou éliminés. Des contraintes spécifiques limitent le nombre de jetons dans certains nœuds du graphe.