Langages et Compilation

https://terrier.users.greyc.fr/LangCompil

Langage formel

Expression régulière

Automate fini déterministe

Équivalence entre expression régulière et AF

Des langages non réguliers

Langage formel

Domaine partagé par

les linguistes
: décrire les langues naturelles avec l'idée de les traiter automatiquement

les informaticiens
: élaborer des langages de programmation évolués et les techniques de compilation associées

Thème central : étudier des modèles qui permettent une spécification ou une représentation finie des langages

Cette formalisation permet de développer des outils qui traitent ces langages.

Langage formel

Plusieurs approches :

spécifier les propriétés des mots du langage :
description en français, expression régulière (Kleene)

engendrer les mots du langage :
les grammaires formelles (Chomsky)

reconnaître qu'un mot donné est un mot du langage avec des mécanismes automatiques réalisés par différents types de machines (Mc Culloch et Pitt, automate fini, Turing)

+ caractérisations logique, algébrique ...

Langage formel – Vocabulaire

Un **alphabet** Σ est un ensemble fini non vide de symboles appelés aussi lettres ou caractères.

$\Sigma = \{0, 1\}$ l'alphabet des langages machines.

$\Sigma = \{A, C, G, T\}$ les quatre nucléotides de l'ADN.

$\Sigma = \{$ ⠁, ⠃, ⠉, ⠙, ⠑, $\cdots$, ⠵ $\}$ l'alphabet braille.

Un **mot** est une suite finie de symboles.

La **longueur** d'un mot u notée $|u|$ est son nombre de symboles.

Sur l'alphabet $\{0, 1\}$, le mot $u = 011$ est de longueur $|u| = 3$.

Le **mot vide** noté ε est le seul mot de longueur nulle. $|\varepsilon| = 0$.

Langage formel – Vocabulaire

La **concaténation** de deux mots u et v, notée uv, est la juxtaposition de u et v (dans cet ordre).

pour $u = a_1 a_2 \cdots a_p$ et $v = b_1 b_2 \cdots b_q$, $uv = \underbrace{a_1 a_2 \cdots a_p}_{u} \underbrace{b_1 b_2 \cdots b_q}_{v}$

- opération associative $(uv)w = u(vw) = uvw$
- non commutative a priori $uv \neq vu$
- qui a ε comme élément neutre $u\varepsilon = \varepsilon u = u$

La **puissance** n-ème d'un mot est définie inductivement :

$$\begin{cases} u^0 = \varepsilon \\ u^{n+1} = u^n u \end{cases}$$

$u = aba$, $u^3 = abaabaaba$

Un mot $u \in \Sigma^*$ est **facteur** du mot $w \in \Sigma^*$ s'il existe $v, v' \in \Sigma^*$ tels que $w = vuv'$.

abb est facteur de *babba* , *aa* non.

u est **facteur gauche** ou **préfixe** lorsque $w = uv'$

u est **facteur droit** ou **suffixe** lorsque $w = vu$.

pre est préfixe de *prefixe*, *fixe* est suffixe de *suffixe*

Langage formel – Vocabulaire

Σ^* dénote l'ensemble des mots sur l'alphabet Σ.

Σ^+ dénote l'ensemble des mots sur l'alphabet Σ de longueur ≥ 1.

Un **langage** sur l'alphabet Σ est un ensemble de mots sur cet alphabet, i.e., toute partie de Σ^*

Exemples

- les mots réservés de Java sur l'alphabet à 26 lettres :
  $\{\text{abstract}, \text{ assert}, \text{ boolean}, \cdots\}$
- Le langage de Dyck sur l'alphabet $\{(,)\}$
- L'ensemble vide $\{\}$
- L'ensemble réduit au mot vide $\{\varepsilon\}$
- $L = \{a^n b^n c^n : n \in \mathbb{N}\}$ sur l'alphabet $\{a, b, c\}$

Langage formel

Opérations classiques sur les ensembles

- l'union $\cup$
- l'intersection $\cap$
- la différence $\setminus$
- le complémentaire (différence de Σ^* est de L)

- La **concaténation** ou le **produit** de deux langages :
  $L_1L_2 = \{uv : u \in L_1 \text{ et } v \in L_2\}$
  $L_1 = \{a, ab\},\ L_2 = \{c, bc\} : L_1L_2 = \{ac, abc, abbc\}$.
- La **puissance** n-ème d'un langage L est définie inductivement :
  $$\begin{cases} L^0 = \{\varepsilon\} \\ L^{n+1} = L^nL \end{cases}$$
  $L = \{a, ab\} : L^0 = \{\varepsilon\},\ L^1 = \{a, ab\},\ L^2 = \{aa, aab, aba, abab\}$
  $L^3 = \{aaa, aaab, aaba, aabab, abaa, abaab, ababa, ababab\}$
- L'**étoile** d'un langage L
  $L^* = \bigcup_{n\geq 0} L^n = \{\varepsilon\} \cup L \cup L^2 \cup \cdots L^n \cup \cdots$
  $L = \{a, ab\} : L^* = \{\varepsilon, a, ab, aa, aab, aba, abab, aaa, aaab, aaba,$
  $aabab, abaa, abaab, ababa, ababab, \cdots\}$
  Les mots tels que bb n'est pas facteur, b n'est pas préfixe.
  L^* langage de cardinalité infinie
  $L^+ = \bigcup_{n>0} L^n = L \cup L^2 \cup \cdots L^n \cup \cdots$

Langage formel

Expression régulière

Automate fini déterministe

Équivalence entre expression régulière et AF

Des langages non réguliers

Expression régulière

Une façon élémentaire de spécifier (de manière finie) certains langages.

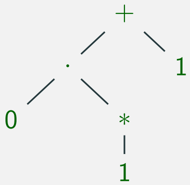
Les **expressions régulières** ou **rationnelles** sur l'alphabet Σ et leurs langages correspondants sont définis récursivement :

1. - $\emptyset$ est une expression régulière qui décrit l'ensemble vide $\{\}$;
   - ε est une expression régulière qui décrit l'ensemble réduit au mot vide $\{\varepsilon\}$;
   - pour toute lettre a de l'alphabet Σ, a est une expression régulière qui décrit l'ensemble $\{a\}$.
2. Si r et s sont des expressions régulières qui décrivent les langages R et S, alors $(r + s)$, (rs) et (r^*) sont des expressions régulières qui décrivent les langages $R \cup S$, RS et R^*.

Expression régulière

Ordre de priorité sur les opérations : l'étoile $>$ la concaténation $>$ l'union.

$((0(1^*))+1)$ s'écrit 01^*+1



Alphabet $\{a, b\}$

description en français	expression régulière	langage associé
se terminent par *a*	$(a+b)^*a$	$\{a,b\}^*\{a\}$
contiennent le facteur *bb*	$(a+b)^*bb(a+b)^*$	$\{a,b\}^*\{bb\}\{a,b\}^*$
ne contiennent pas le facteur *bb*	$(\varepsilon+b)(a+ab)^*$ $=$ $(\varepsilon+b)(a^+b)^*a^*$	$\{\varepsilon,b\}\{a,ab\}^*$

Expression régulière – Utilisation en compilation

Les unités lexicales des langages de programmation (mots clés, identificateurs, nombres ...)
sont définies par des expressions régulières

Les nombres décimaux 110 , 3•45 , 9•00 , 5•

$$[1\text{-}9][0\text{-}9]^*(\varepsilon + \bullet)[0\text{-}9]^*$$

Enjeu

Construire un mécanisme qui reconnaît l'ensemble des mots décrits par une expression rationnelle

Langage formel

Expression régulière

Automate fini déterministe

Équivalence entre expression régulière et AF

Des langages non réguliers

Automate fini déterministe

Un modèle de calcul élémentaire qui reconnaît les expressions régulières

Un **automate fini déterministe** (**AFD**) est un quintuplet $(\Sigma, Q, \delta, q_o, F)$ où :

- Σ est un alphabet
- Q est un ensemble **fini** d'états
- δ est la fonction de transition de $Q \times \Sigma \to Q$
- $q_0 \in Q$ est l'état initial
- $F \subset Q$ est l'ensemble des états d'acceptation

L'automate est **complet** si δ est défini sur tout $Q \times \Sigma$.

$\mathcal{A} = \big(\{a, b\}, \{0, 1, 2\}, \delta, 0, \{0, 2\}\big)$ avec

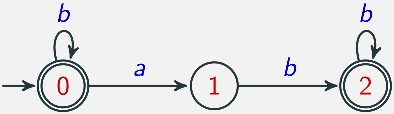
$$\begin{array}{rccc} \delta : & Q \times \Sigma & \to & Q \\ & (0, a) & \to & 1 \\ & (0, b) & \to & 0 \\ & (1, b) & \to & 2 \\ & (2, b) & \to & 2 \end{array}$$

L'automate n'est pas complet, les transitions $\delta(1, a)$ et $\delta(2, a)$ ne sont pas définies.

Deux représentations utiles de l'automate :

	a	b
$\to$ 0	1	0
1		2
2		2

la table de transition



le graphe de transition

On représente une transition $\delta(q, a)$ par :

- $q \xrightarrow{a} p$ si $\delta(q, a) = p$
- $q \xrightarrow{a} \perp$ si $\delta(q, a)$ est non définie

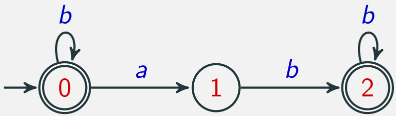
Le **calcul** de l'automate sur le mot d'entrée $w = a_1 a_2 \cdots a_n$
se décrit par la suite de transitions qui part de l'état initial q_0 et
qui est étiquetée par w :

$$q_0 \xrightarrow{a_1} q_1 \xrightarrow{a_1} q_2 \ \cdots \ q_{n-1} \xrightarrow{a_n} q_n$$

Le mot w est **accepté** par l'automate si le calcul se termine dans un état d'acceptation.

Caractéristiques du calcul :

- lecture du mot sans retour
- déterministe : à chaque transition, au plus un choix possible
- l'espace mémoire de la machine est de taille constante



calcul sur *babb* : $0 \xrightarrow{b} 0 \xrightarrow{a} 1 \xrightarrow{b} 2 \xrightarrow{b} 2$

le calcul termine sur l'état 2 qui est un état d'acceptation $\rightsquigarrow$ *babb* est accepté

calcul sur *abbaab* : $0 \xrightarrow{a} 1 \xrightarrow{b} 2 \xrightarrow{b} 2 \xrightarrow{a} \perp$ l'automate se bloque $\rightsquigarrow$ *abbaab* n'est pas accepté

calcul sur *bba* : $0 \xrightarrow{b} 0 \xrightarrow{b} 0 \xrightarrow{a} 1$

le calcul termine sur l'état 1 qui n'est pas état d'acceptation $\rightsquigarrow$ *bba* n'est pas accepté

mot accepté

$\equiv$ mot qui étiquette un chemin du graphe de l'état initial à un état d'acceptation

Automate fini déterministe

Distributeur de café

Le café coûte 40 centimes

Les pièces acceptées sont celles de 10 et 20 centimes

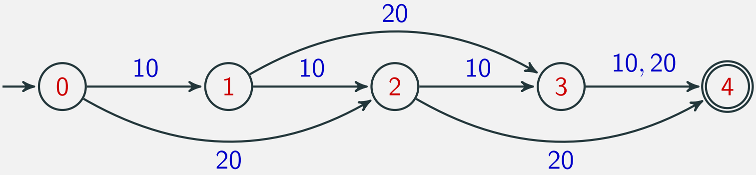
Pas de retour de monnaie

L'alphabet $\Sigma = \{10, 20\}$

L'ensemble des états $Q = \{0, 1, 2, 3, 4\}$

($\sim$ reçu $0, 10, 20, 30$ ou ≥ 40 centimes)

0 : l'état initial, $F = \{4\}$: l'ensemble des états d'acceptation



Automate fini déterministe

Recherche de toutes les occurrences d'un mot dans un texte

Le mot cherché : *abaab*

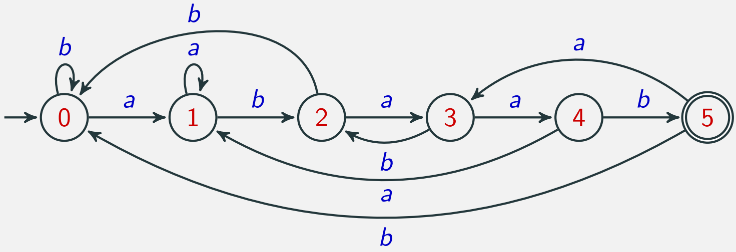
Le texte : une chaîne sur l'alphabet $\Sigma = \{a, b\}$

Principe : Enregistrer dans l'état le nombre maximal de lettres lues consécutivement qui soient préfixe du mot cherché.

Les états $\equiv$ les préfixes du mot

ε	a	ab	aba	$abaa$	$abaab$
0	1	2	3	4	5

L'état initial $0 = |\varepsilon|$, un seul état d'acceptation $5 = |abaab|$



Automate fini déterministe

Prolongement de la fonction de transition δ en une fonction de $Q \times \Sigma^*$ dans Q :

$\delta(q, w)$: l'état atteint après lecture du mot w en partant de q

$$q \overset{w}{\rightsquigarrow} \delta(q, w)$$

En toute généralité, on a pour tout état q et tous mots $u, v \in \Sigma^*$

$$\delta(q, uv) = \delta(\delta(q, u), v)$$

$$q \overset{u}{\rightsquigarrow} \delta(q, u) \overset{v}{\rightsquigarrow} \delta(q, uv)$$

Automate fini déterministe

Langage régulier

Le langage **reconnu** par un AFD $\mathcal{A} = (\Sigma, Q, \delta, q_0, F)$ est l'ensemble des mots acceptés par $\mathcal{A}$:

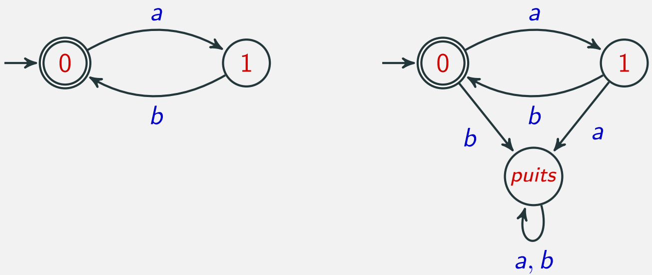
$$L(\mathcal{A}) = \{w \in \Sigma^* : \delta(q_0, w) \in F\}$$

Un langage est dit **régulier** ou **rationnel** si il est reconnu par un AFD.

Automate fini déterministe

Rendre **complet** un AFD :

- ajouter un état **puits**
- toute transition qui était non définie arrive alors dans cet état puits



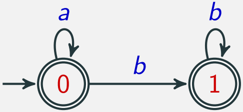
L'automate initial et l'automate complété sont équivalents : ils reconnaissent le même langage.

Automate fini déterministe

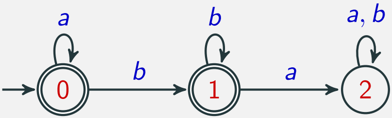
Propriétés de stabilité - COMPLÉMENT

Si $L \subset \Sigma^*$ est régulier alors $\Sigma^* \backslash L$ est régulier

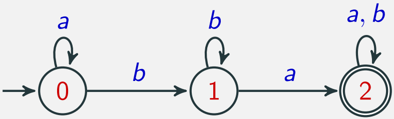
$L = \{a^i b^j : i, j \in \mathbb{N}\}$



l'automate complété

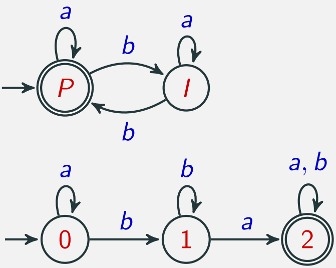


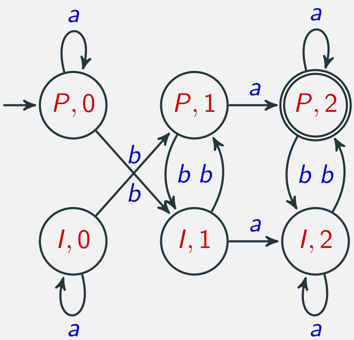
$\Sigma^* \backslash L = \{w \in \Sigma^* : ba$ **est facteur de** $w\}$



Si $\mathcal{A} = (\Sigma, Q, \delta, q_0, F)$ est un automate complet qui reconnaît L alors $\mathcal{A}^C = (\Sigma, Q, \delta, q_0, Q \backslash F)$ reconnaît $\Sigma^* \backslash L$.

Si L_1 et L_2 sont réguliers alors $L_1 \cap L_2$ est régulier





Si $\mathcal{A}_i = (\Sigma, Q_i, \delta_i, q_i^0, F_i)$ reconnaît L_i pour $i = 1, 2$,

alors $\mathcal{A}_\cap = (\Sigma, Q_1 \times Q_2, \delta, (q_1^0, q_2^0), F_1 \times F_2)$

avec $\delta((r, s), a) = (\delta_1(r, a), \delta_2(s, a))$ reconnaît $L_1 \cap L_2$.

Si L_1 et L_1 sont réguliers alors $L_1 \cup L_2$ est régulier

argument ensembliste :

$$L_1 \cup L_2 = \Sigma^* \backslash \left(\Sigma^* \backslash L_1 \cap \Sigma^* \backslash L_2 \right)$$

ou directement avec l'automate produit :

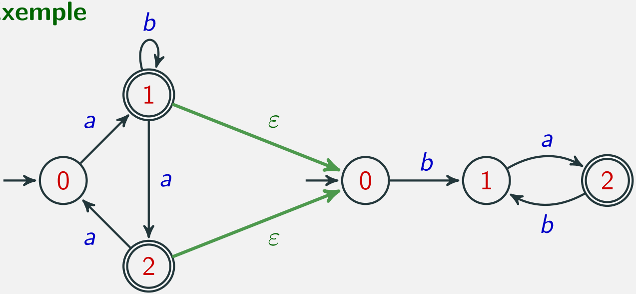
$$\mathcal{A}_\cup = (\Sigma, Q_1 \times Q_2, \delta, (q_1^0, q_2^0), F_1 \times Q_2 \cup Q_1 \times F_2)$$

Automate fini déterministe

Propriétés de stabilité - CONCATÉNATION

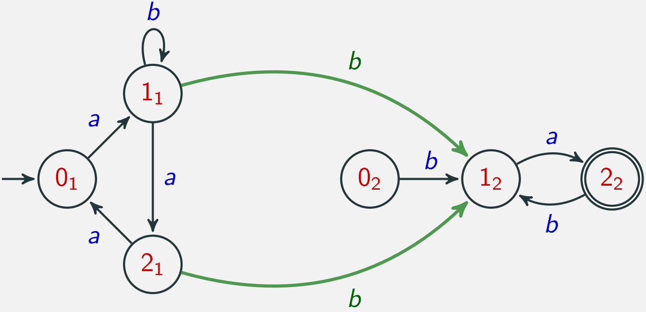
Si L_1 et L_1 sont réguliers alors L_1L_2 est régulier

Exemple



Passer des états d'acceptation de L_1 à l'état initial de L_2 sans lire de symbole : effectuer des ε-transitions

Mais on introduit du non-déterminisme : lorsque la machine est dans l'état 1_1 et lit le symbole b, dans quel état va-t-elle ?



On montrera que pour tout automate fini non-déterministe, il existe un automate fini déterministe qui fait le même travail.

Si L est régulier alors L^* est régulier

Même approche que pour la concaténation

Langage formel

Expression régulière

Automate fini déterministe

Équivalence entre expression régulière et AF

Des langages non réguliers

Équivalence entre expression régulière et AF

la classe des langages décrits par les expressions régulières correspond exactement à celle des langages réguliers

Le théorème de Kleene

Un langage est reconnaissable par automate fini
si et seulement si une expression régulière le représente.

D’expression régulière vers automate fini

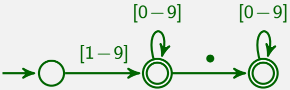
$\Leftarrow$ Tout langage décrit par une expression régulière est reconnaissable par automate fini :

- Les langages $\{\}$, $\{\varepsilon\}$ et $\{a\}$ pour toute lettre a de l’alphabet Σ sont reconnaissables par automate fini.
- De plus la classe des langages reconnaissables par automate fini est close par union, concaténation et étoile.

Enjeu. Définir une construction automatique et efficace qui traduise une expression régulière en automate fini.

Ce qui est réalisé par exemple avec la méthode `compile` du module `re` qui construit un automate à partir d’une expression régulière

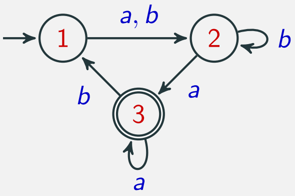
```
decimal.py
import re
regex = re.compile(r"^[1-9][0-9]*\.?[0-9]*$")
for mot in ["123", "23.0", "2b", "001"]:
            if regex.match(mot) is not None:
                      print(mot, "est accepté")
            else:
                      print(mot, "est rejeté")
```



D'automate fini vers expression régulière

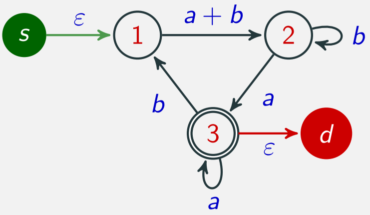
$\Rightarrow$ Un algorithme simple qui trouve une expression régulière à partir d'un automate fini

On part du graphe de transition de l'automate.

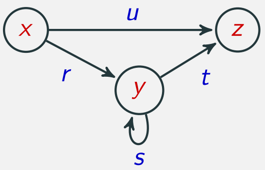


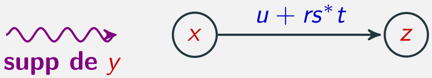
On ajoute

- une source avec une transition étiquetée par ε sur l'état initial.
- une destination avec des transitions étiquetées par ε des états d'acceptation vers cette destination

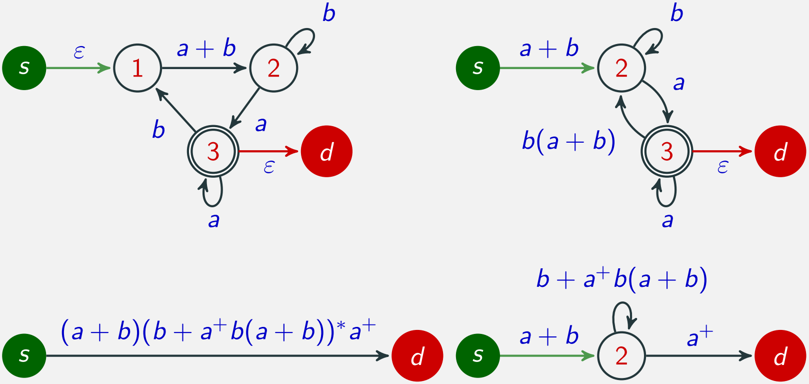


On supprime un à un les états de la façon suivante :





invariant : les transitions sont étiquetées par des expressions régulières



Langage formel

Expression régulière

Automate fini déterministe

Équivalence entre expression régulière et AF

Des langages non réguliers

Des langages non réguliers

Un AFD ne sait pas compter plus loin que le nombre de ces états.

$L = \{a^n b^n \colon n \in \mathbb{N}\}$ **n'est pas régulier.**

Preuve

Supposons le contraire : un AFD $(\Sigma, Q, \delta, q_0, F)$ reconnaît L .

Soit $k = |Q|$ son nombre d'états.

Considérons la suite des $k+1$ états :
$q_0 = \delta(q_0, \varepsilon)\,,\ \delta(q_0, a)\,,\ \delta(q_0, a^2)\,,\ \cdots\,,\ \delta(q_0, a^k)$.

Parmi ces $k+1$ états, deux sont nécessairement identiques :
il existe $i \neq j$ tels que $\delta(q_0, a^i) = \delta(q_0, a^j)$.

On a donc $\delta(q_0, a^i b^i) = \delta(q_0, a^j b^i)$ avec

- $\delta(q_0, a^i b^i) \in F$ $(a^i b^i \in L)$
- $\delta(q_0, a^j b^i) \notin F$ $(i \neq j \rightarrow a^j b^i \notin L)$.

Ce qui est une contradiction

Donc aucun AFD ne reconnaît le langage $\{a^n b^n \colon n \in \mathbb{N}\}$.

Des langages non réguliers

Un outil pour montrer que des langages sont non réguliers :

Le lemme de l'étoile

Pour tout langage régulier L,

il existe un entier n tel que pour tout mot $x \in L$ avec $|x| \geq n$,

il existe u, v, w avec $|v| \geq 1, |uv| \leq n$, $x = uvw$

et pour tout i, on a $uv^i w \in L$.