MINES PARIS | PSL

Natural Language Processing (NLP)

2 — Grammaires formelles

Georges-André Silber
https://mines.paris/nlp

Session 2025/2026 — ES3A_MES-07

École des mines de Paris

Informatique et langages

- Informatique = science du traitement automatique des données.
- Données = suite d'informations représentées dans un certain langage.
- Théorie des langages formels : étude des structures internes formelles des langages (niveau syntaxique).
- Auparavant, les systèmes d'analyse syntaxique utilisaient principalement des grammaires formelles, aidées éventuellement par des statistiques.
- Aujourd'hui, dans ces systèmes, les grammaires formelles sont complétées, voire remplacées, par des techniques d'apprentissage automatique.

Hiérarchie des grammaires formelles de N. Chomsky et M. P. Schützenberger :

- les grammaires de type 3 génèrent la famille des *langages rationnels* ou *langages réguliers*. Langages reconnaissables par les *automates finis* ;
- les grammaires de type 2 génèrent la famille des *langages algébriques*. Langages reconnaissables par les *automates à pile* ;
- les grammaires de type 1 génèrent la famille des *langages contextuels*. Langages reconnaissables par les *automates linéairement bornés* ;
- les grammaires de type 0, dites grammaires générales, génèrent la famille des *langages récursivement énumérables*. Langages reconnaissables par une *machine de Turing*.

Langages rationnels

- A : ensemble fini de symboles (*alphabet*) ;
- A^* : ensemble des mots que l'on peut former avec A ;
- Partie de A^* : un *langage* ;
- Opérations rationnelles, avec X et Y deux parties de A^* :
  - Concaténation : XY

$$\{ab, c\}\{ba, c\} = \{abba, abc, cba, cc\}$$

  - Union : $X \cup Y$

$$\{ab, c\} \cup \{ba, c\} = \{ab, ba, c\}$$

  - Étoile de Kleene (notée X^*) : le plus petit langage contenant ϵ, X et qui est clos pour l'opération de concaténation.

$$\{a, ab\}^* = \{\epsilon, a, aa, ab, aaa, ababaa, ...\}$$

Application pratique : **expressions régulières** par génération d'un automate fini (Ken Thompson). `lex`, analyseur lexical.

Langages algébriques

- Grammaires non-contextuelles / hors-contexte
- Règles de la forme $X \to \alpha$ où α est un terminal ou un non terminal
- Un langage est algébrique si il ∃ une grammaire algébrique le décrivant
- La plupart des langages de programmation ont une grammaire algébrique
- Peuvent être décrits sous la forme Backus-Naur (BNF)[1]
- Exemples de langages algébriques (et non rationnels) :

$$S \to aSb \mid \epsilon$$

$$\{a^n b^n \mid n \geq 0\}$$

$$S \to x \mid y \mid z \mid S + S \mid S - S \mid S * S \mid S/S \mid (S)$$

- Outils : YACC (LALR), ANTLR (LL)

1. https://fr.wikipedia.org/wiki/Forme_de_Backus-Naur

Langages algébriques (2)

- Panini, un grammairien de l'Inde antique a formalisé une grammaire du Sanskrit (IVe siècle avant J.C.), avec près de 4000 dans une notation proche de la BNF.
- Langages de Dyck : ensemble des mots bien parenthésés sur un ensemble fini de parenthèses ouvrantes et fermantes.

Définition (Morphisme alphabétique)
$h : A^* \to B^*$ avec A et B des monoïdes libres et h un morphisme. h est *alphabétique* si l'image d'une lettre de A est une lettre de B ou ϵ.

Théorème (Chomsky–Schützenberger)
Un langage L est algébrique ssi il existe un langage de Dyck D, un langage rationnel K et un morphisme alphabétique h tels que

$$L = h(D \cap K)$$

Langages algébriques (3)

- Soit la grammaire :

$$S \rightarrow S + S \mid a \mid 1$$

- Dérivations de $1 + a + a$

Dérivation gauche	Dérivation droite
S	S
$S + S$	$S + S$
$1 + S$	$S + a$
$1 + S + S$	$S + S + a$
$1 + a + S$	$S + a + a$
$1 + a + a$	$1 + a + a$

⇔ Machine de Turing non déterministe linéairement bornée (avec n la taille de l'entrée, ruban de taille kn où k est une constante indépendante de n).

$$L = \{a^n b^n c^n \mid n \geq 1\}$$

La machine de Turing

Objet mathématique abstrait composé :

- d'une bande infinie découpé en cases pouvant contenir un symbole ;
- d'une tête de lecture pouvant à chaque étape lire un symbole, écrire un symbole, puis se déplacer sur la bande d'une case à gauche ou à droite ;
- un registre fini d'états dans lesquels peut se trouver la machine ;
- une table d'action indiquant pour un état et un symbole l'action à effectuer.

La machine de Turing

Une machine de Turing déterministe est un septuplet $M = \langle Q, \Gamma, b, \Sigma, \delta, q_0, F \rangle$ où :

- Q est l'ensemble fini non vide des *états* ;
- Γ est l'ensemble fini non vide des *symboles de la bande* ;
- $b \in \Gamma$ est le symbole *blanc* ;
- $\Sigma \subseteq \Gamma \setminus \{b\}$ est l'ensemble des *symboles d'entrée*, les seuls symboles autorisés initialement sur la bande ;
- $\delta : (Q \setminus F) \times \Gamma \nrightarrow Q \times \Gamma \times \{\leftarrow, \rightarrow\}$ est la fonction partielle de *transition*. Si δ n'est pas définie sur l'état courant et le symbole courant, la machine s'arrête ;
- $q_0 \in Q$ est l'état initial ;
- $F \subseteq Q$ est l'ensemble des *états acceptants* : le contenu initial de la bande est *accepté* par M si elle s'arrête dans un état de F.

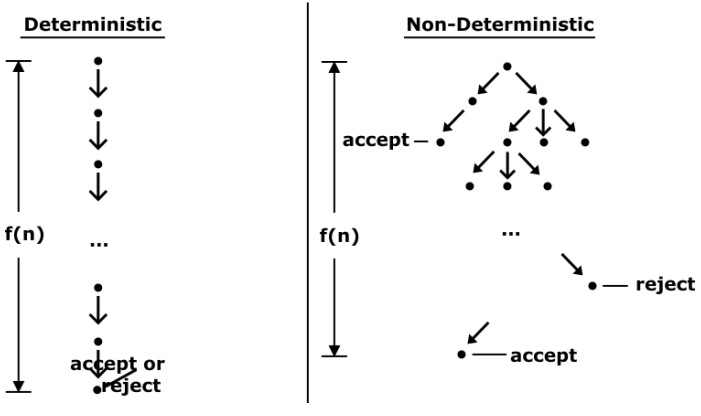
Exemple de partie de δ : $\delta(q_1, x) = (q_2, y, \leftarrow)$ indique que dans l'état q_1 quand x est lu sur la bande, on passe en état q_2, on écrit y et on se déplace à $\leftarrow$.

La machine de Turing non déterministe

Une machine de Turing non déterministe est un septuplet $M = \langle Q, \Gamma, b, \Sigma, \delta, q_0, F \rangle$ où :

- Q est l'ensemble fini non vide des *états* ;
- Γ est l'ensemble fini non vide des *symboles de la bande* ;
- $b \in \Gamma$ est le symbole *blanc* ;
- $\Sigma \subseteq \Gamma \setminus \{b\}$ est l'ensemble des *symboles d'entrée*, les seuls symboles autorisés initialement sur la bande ;
- $\delta \subseteq (Q \setminus F \times \Gamma) \times (Q \times \Gamma \times \{\leftarrow, \rightarrow\})$ est la relation de *transition* ;
- $q_0 \in Q$ est l'état initial ;
- $F \subseteq Q$ est l'ensemble des *états acceptants* : le contenu initial de la bande est *accepté* par M si *une branche* s'arrête dans un état de F.

Machine de Turing déterministe vs non-déterministe



Expressions régulières

Expressions régulières

- Expressions régulières par génération d'un automate fini (Ken Thompson).
- `grep`, `lex`, analyseur lexical
- `https://regexcrossword.com`
- Python : `import re`
- hyperscan

Exemple 1 : utilisation dans un IDE

```
#define MAX_URI_COUNTRY 3
#define MAX_URI_CORPUS 5
#define MAX_URI_NATURE 70
#define MAX_URI_YEAR 5
#define MAX_URI_MONTH 3
#define MAX_URI_DAY 3
#define MAX_URI_NUMBER 30
#define MAX_URI_VERSION 9
#define MAX_URI 256

MAX_(\w+)
$1_MAX
```

Exemple 2 : découpage d'un arrêt de cour d'appel d'Agen

```
intro_re = re.compile(
  r'^(?P<intro>.*?)(?=('
  r'<p>\s*A\s+rendu\s+l.arrêt\s+((réputé\s+)?
  r'contradictoire|par\s+défaut)'
  r'|<p>\s*EXPOS(É|E)\s*DU\s*LITIGE'
  r'|<p>A rendu réputé l.arrêt réputé contradictoire'
  r'))',
  re.UNICODE|re.DOTALL|re.MULTILINE|re.IGNORECASE)
decision_re = re.compile(
  r'(?P<decision>(<p>par\s*ces\s*motifs).*)$',
  re.U|re.DOTALL|re.MULTILINE|re.IGNORECASE)
```

Exemple 3 : numéros d'alinéas (Droit Quotidien)

```
alinea_number = (
  r"("
  r"\w\)(?=\s+)"
  r"|\d{1,2}°(\s+bis)?(?=\.?\s+)"
  r"|\d{1,2}(\s+bis)?(?=\.?\s+)"
  r"|[IVX]+(?=\.\s+)"
  r")"
)
```