

IA41

**Concepts fondamentaux en Intelligence Artificielle
et langages dédiés**

CM #6

**Introduction à la
théorie des langages
formels**

Fabrice LAURI

Théorie des langages formels

- Introduction**
- Définitions : alphabet, langage, système de réécriture, grammaire**
- Exemples d'analyseurs PROLOG**

Théorie des langages formels

- **Introduction**

- **Définitions : alphabet, langage, système de réécriture, grammaire**

- **Exemples d'analyseurs PROLOG**

Introduction

- Un des domaines de l'Informatique les plus étudiés
- Applications :
 - compilation
 - études des langues naturelles
 - étude de la calculabilité et de la complexité

Théorie des langages formels

- Introduction
- Définitions : alphabet, langage, système de réécriture, grammaire
- Exemples d'analyseurs PROLOG

Définitions

Alphabet :

Ensemble fini non vide de symboles. On le notera Ω .

Mot sur un alphabet Ω :

Suite de n symboles de Ω , avec $n \geq 0$. λ est le mot vide ($n=0$).

Concaténation de mots :

Consiste à placer deux mots m et m' côte à côte pour former le mot mm' . mm' est constitué de la suite des symboles de m suivie de la suite des symboles de m' .

Langage engendré par un alphabet Ω :

Ensemble de tous les mots sur Ω . Noté Ω^* si $\lambda \in \Omega^*$ et Ω^+ sinon.

Langage formel sur un alphabet Ω :

Sous-ensemble de Ω^* .

Exemples de langages

Exemple #1 :

$\Omega = \{ a, b \}$

Ω^* : ensemble des mots ne contenant que des a ou des b.

$L = \{ ab, b, aba, bba, bbbababa \}$ langage fini sur Ω .

$L' = \{ a^p b^q, p \geq 0, q \geq 0 \}$ langage infini sur Ω .

Exemple #2 :

Ω : instructions d'un langage de programmation.

Ω^* : ensemble des programmes pouvant être écrits à l'aide de Ω
sans respecter une syntaxe précise.

L : ensemble des programmes syntaxiquement corrects en utilisant Ω .

Systemes de réécriture (1/2)

Objectif :

Construire un langage à partir d'un alphabet et de règles de formation (ou règles de production) des mots du langage.

Systeme de réécriture :

Triplet $R = \{ \Omega, \Pi, \rightarrow \}$, où Ω est l'alphabet, Π un ensemble de couple (m, m') avec m et m' des mots sur Ω et \rightarrow est la relation de production directe.

Les couples (m, m') sont appelés *règles de production* et sont notées $m \rightarrow m'$.

La relation de production directe \rightarrow est définie par $x \rightarrow y$ ssi il existe des mots x_1 et x_2 et une règle de production $m \rightarrow m'$ tels que $x_1 m x_2 \rightarrow x_1 m' x_2$.

Systemes de réécriture (2/2)

Relation de dérivation :

Relation notée \rightarrow_* définie par : $x \rightarrow_* y$ ssi il existe des mots

m_1, m_2, \dots, m_p avec $p \geq 0$ telle que $x \rightarrow m_1 \rightarrow m_2 \rightarrow \dots \rightarrow m_p \rightarrow y$.

On dit alors que y dérive de x .

Exemple :

- $\Omega = \{ a, b, c \}$

- $R = \{ \Omega, \Pi, \rightarrow \}$ avec $\Pi = \{ (a,aa), (a,cc), (aa,aba), (cc,aca) \}$

Est-ce que :

- $aaaa \rightarrow ccaa ?$

- $cc \rightarrow_* ccccc ?$

- $aaaa \rightarrow accaa ?$

- $aaaa \rightarrow_* abababaca ?$

- $aaaa \rightarrow acc ?$

Grammaire (1/4)

Grammaire :

Définie par un quadruplet $G = (R, \Omega_N, \Omega_T, D)$, où :

- R : système de réécriture défini sur $\Omega = \Omega_N \cup \Omega_T$
- Ω_N : ensemble des symboles non terminaux (lettres majuscules)
- Ω_T : ensemble des symboles terminaux (lettres minuscules)
- D : symbole de Ω_N représentant le symbole de départ (**axiome**).

Langage généré par une grammaire G :

Défini par $L(G) = \{ m \in \Omega_T^* \mid D \rightarrow_* m \}$

Grammaires équivalentes :

Deux grammaires sont équivalentes si elles génèrent le même langage.

Grammaire (2/4)

Langage récursivement énumérable :

Un langage L est récursivement énumérable ssi il existe une grammaire G tel que $L = L(G)$.

Autrement dit, L est récursivement énumérable ssi il existe une procédure capable de déterminer si un mot M appartient ou non à L .

Exemple de grammaire :

- $\Omega_N = \{ D \}$

- $\Omega_T = \{ a, b \}$

- Règles de production :

- $D \rightarrow aD$

- $D \rightarrow Db$

- $D \rightarrow \lambda$

} peuvent s'écrire plus simplement : $D \rightarrow aD \mid Db \mid \lambda$

Grammaire (3/4)

Grammaire et langage algébriques :

Une grammaire est algébrique (**libre de contexte**) ssi toutes les règles de production sont de la forme :

$$\mathbf{A} \rightarrow \mathbf{m} \text{ avec } \mathbf{A} \in \Omega_N \text{ et } \mathbf{m} \in \Omega^*.$$

Un langage est algébrique s'il est généré par une grammaire algébrique.

Exemple :

- $\Omega_N = \{ D \}$
- $\Omega_T = \{ a, b \}$
- Règles de production : $D \rightarrow aD \mid Db \mid \lambda$

Grammaire (4/4)

Grammaire et langage réguliers :

Une grammaire est régulière ssi toutes les règles de production sont de la forme :

$$\mathbf{A} \rightarrow \mathbf{m} \text{ ou } \mathbf{A} \rightarrow \mathbf{mB}, \text{ avec } \mathbf{A}, \mathbf{B} \in \Omega_N \text{ et } \mathbf{m} \in \Omega_T^*.$$

Un langage est régulier s'il est généré par une grammaire régulière.

Exemple :

- $\Omega_N = \{ D, S_1, S_2 \}$

- $\Omega_T = \{ a, b \}$

- D : axiome

- Règles de production :

- $D \rightarrow aS_1 \mid bS_2$

- $S_1 \rightarrow bS_1 \mid \lambda$

- $S_2 \rightarrow aS_2 \mid \lambda$

Théorie des langages formels

- Introduction
- Définitions : alphabet, langage, système de réécriture, grammaire
- Exemples d'analyseurs PROLOG

Analyseurs PROLOG (1/5)

PROLOG a été conçu pour permettre à un utilisateur de dialoguer dans une langue proche du français avec un système expert.

Systeme capable de dialoguer doit :

- reconnaître des mots particuliers d'une phrase
→ *analyse lexicale*
- reconnaître certains agencements de mots dans une phrase
→ *analyse grammaticale (syntaxique)*
- identifier le sens d'une phrase :
→ *analyse sémantique*
- répondre à une question en fonction du sens de la question et de ses connaissances

Construire un analyseur PROLOG consiste à définir un prédicat dialogue/1 qui retourne, par exemple, Yes s'il a « compris » ou reconnu la phrase, et No sinon.

Analyseurs PROLOG (2/5)

Les mots d'une phrase à reconnaître sont disposés dans une liste.

Par exemple, pour pouvoir reconnaître la phrase :

« PROLOG est un langage à connaître »

PROLOG doit transformer cette phrase en une liste :

['PROLOG', est, un, langage, 'à', 'connaître']

Reconnaître une phrase : vérifier pour chaque mot m reconnu, tel que m appartient au lexique, que sa position dans la phrase respecte une certaine grammaire.

Analyseurs PROLOG (3/5)

Pour construire un analyseur syntaxique en PROLOG :

- 1) Définir la **grammaire** G génératrice du langage à reconnaître
- 2) Constituer le **lexique (symboles terminaux)** en utilisant des prédicats PROLOG possédant éventuellement des propriétés :

type1([<mot1> | R], R, <prop1>, ..., <propN>).

type2([<mot2> | R], R, <prop1>, ..., <propS>).

...

- 3) Transformer les règles de la grammaire G en clauses :
1 règle de G = 1 clause PROLOG

Analyseurs PROLOG (4/5)

Exemple : analyseur de groupes nominaux

Soit la grammaire suivante :

Phrase → **GroupeNominal**
GroupeNominal → **Prénom** | **Prénom et Prénom**
Prénom → **'Paul'** | **'Emile'** | **'Anne'** | **'Marie'**

permettant de reconnaître des phrases telles que :

« Paul »

« Paul et Emile »

« Marie et Anne et Paul »

...

Analyseurs PROLOG (5/5)

Exemple : analyseur de phrases du type *Sujet Verbe*

Soit la grammaire suivante :

Phrase	→ GroupeNominal GroupeVerbal
GroupeVerbal	→ AuxiliaireEtre Participe
GroupeNominal	→ Prénom Prénom et Prénom
AuxiliaireEtre	→ est sont
Participe	→ parti partis partie parties
Prénom	→ 'Paul' 'Jean' 'Anne' 'Marie'

permettant de reconnaître les phrases suivantes :

« Marie est partie »

« Paul est parti »

« Anne est partie »

« Paul et Jean sont partis »

« Marie et Paul sont partis »

« Marie et Anne sont parties »