REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE
Ministère de l'enseignement supérieur et de la recherche scientifique
Université Larbi Ben M'Hidi Oum El Bouaghi

Cours traitement du langage naturel

# CHAPITRE 4 : Analyse syntaxique

# Contenu du Chapitre :

01

Grammaires formelles et contextuelles

02

Analyse syntaxique basée sur les règles

03

Méthodes d'analyse syntaxique probabiliste

# 01 Grammaires formelles et contextuelles

# Grammaires formelles et contextuelles

**Définition :**

- Une grammaire formelle est un système mathématique permettant de décrire les structures syntaxiques d'un langage, qu'il soit naturel ou artificiel.

- Elle est constituée de règles de production qui définissent comment les phrases sont construites à partir d’un ensemble de symboles.

# Grammaires formelles et contextuelles

## Composants d'une grammaire formelle :

Une grammaire est souvent représentée par un quadruplet **G=(N,T,P,S)** où :

N : Ensemble de symboles **non-terminaux**

**T : Ensemble de symboles terminaux**

**P : Ensemble de règles de production, définissant comment les non-terminaux se transforment.**

**S : Symbole de départ**

# Grammaires formelles et contextuelles

**Types de grammaire :**

La hiérarchie de **Chomsky** classe les grammaires formelles selon leur complexité et leur capacité à décrire les langages.

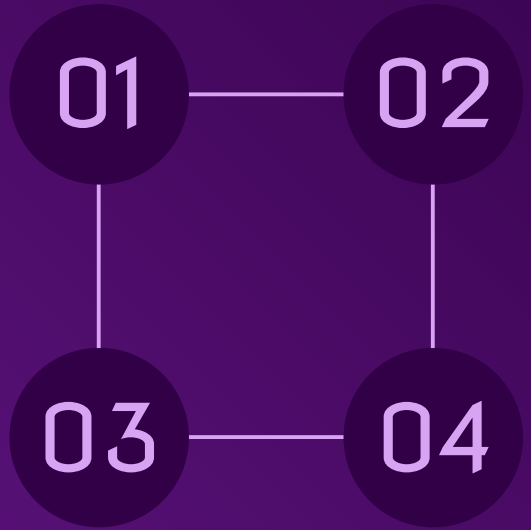
Elle comporte quatre types, du plus restreint (Type 3) au plus général (Type 0)

# Grammaires formelles et contextuelles

## Types de grammaire

**Type 3** : Grammaires régulières

**Type 2** : Grammaires hors-contexte (Context-Free Grammars, CFG)




**Type 1** : Grammaires sensibles au contexte (Context-Sensitive Grammars, CSG)

**Type 0** : Grammaires récursivement énumérables.

# Grammaires formelles et contextuelles

**Types de grammaire :**

## Grammaires régulières (Type 3)

- Ce sont les grammaires les plus simples

- Chaque règle de production suit une forme stricte :

$$A \rightarrow aB \text{ ou } A \rightarrow a$$

Où A et B sont des non-terminaux, et a est un terminal

# Grammaires formelles et contextuelles

**Types de grammaire :**

**Exmple de grammaire de types 3**

Expliqué en python

$$S \rightarrow aS|bS|\epsilon$$

Cette grammaire génère toutes les chaînes composées de 'a' et 'b'  : y compris la chaîne vide

[ab]*

# Grammaires formelles et contextuelles

**Types de grammaire :**

Expliqué en python

**Exmple de grammaire de types 3** $S \rightarrow aS|bS|\epsilon$

`import re` : le module **re** (regular expressions), qui permet de travailler avec les expressions régulières.

```
regex = r'^[ab]*$'
```

- ^ : Début de la chaîne
- [ab]* :Accepte 0 ou plus caractères a ou b (accepte la chaîne vide)
- [ab]+ → Accepte au moins 1 caractère a ou b (rejette la chaîne vide)
- [ ] : Définit un ensemble de caractères autorisés (a et b uniquement)
- * : Répétition de 0 à l'infini des caractères dans [ab]
- $ : Fin de la chaîne

# Grammaires formelles et contextuelles

**Types de grammaire :**

## Caractéristiques et applications des Grammaires régulières (Type 3)

- Génèrent des langages réguliers, souvent représentés par des expressions régulières ou des automates finis

- Ne permettent pas de modéliser des structures imbriquées

Appliqués à :

- la reconnaissance de modèles simples comme les mots-clés dans un texte.
- Analyse lexicale dans les compilateurs.

# Grammaires formelles et contextuelles

**Types de grammaire :**

## Grammaires hors-contexte (Type 2)

- Une grammaire est dite hors-contexte si ses règles ont la forme : $A \rightarrow \alpha$

- Où A est un non-terminal et α est une chaîne contenant des terminaux et/ou non-terminaux.

- Produisent les **langages hors contexte**, reconnus par des **automates à pile**.

# Grammaires formelles et contextuelles

**Types de grammaire :**

**Exemple de Grammaires hors-contexte (Type 2)**

Expliqué en python

**S → aSb | ε** : cette grammaire génère des chaînes comme $a^n b^n$ , par exemple "ab", "aabb", "aaabbb", etc.

abab" : invalide car elle n'est pas de la forme $a^n b^n$ (les 'a' et 'b' sont mélangés).

# Grammaires formelles et contextuelles

**Types de grammaire :**

## Exemple de Grammaires hors-contexte (Type 2) : S → aSb | ε

Expliqué en python

```python
#Grammaires hors-contexte (Type 2)
def is_context_free(string):
    n = len(string) // 2
    if len(string) % 2 != 0:
        return False
    return string[:n] == 'a' * n and string[n:] == 'b' * n
```

- `string[:n] == 'a' * n`
- Vérifie si la première moitié est composée uniquement de a
- `string[n:] == 'b' * n`
- Vérifie si la deuxième moitié est composée uniquement de b

# Grammaires formelles et contextuelles

**Types de grammaire :**

## Grammaires hors-contexte (Type 2)

Ces types de grammaires sont appliqués à la:

- Construction d'arbres syntaxiques pour les phrases
- Modélisation des langages de programmation et de structures de phrases simples.

# Grammaires formelles et contextuelles

**Types de grammaire :**

## Grammaires sensibles au contexte (Type 1)

- Ces grammaires permettent des règles où le contexte autour d'un non-terminal influence son remplacement.

- Une règle typique a la forme : $\alpha A \beta \rightarrow \alpha \gamma \beta$

- Où A est un non-terminal, α, β, et γ sont des chaînes de terminaux et de non-terminaux.
- De plus, $|\gamma| \geq |A|$ , garantissant une expansion ou une conservation de la taille

# Grammaires formelles et contextuelles

**Types de grammaire :**

## Grammaires sensibles au contexte (Type 1)

Appliqués à la :

- Modélisation des règles syntaxiques avancées (accords, morphologie)
- Analyse des langues riches en morphologie comme l’arabe ou le russe

# Grammaires formelles et contextuelles

**Types de grammaire :**

## Grammaires sensibles au contexte (Type 1)

Grammaire :
Variables : S,A,BS, A, BS,A,B
Alphabet terminal : a,ba, ba,b
Production :

- S→aSb
- S→AB
- AB→aB
- B→b

**générer des chaînes de la forme $a^n$ $b^n$ comme aabb**

# Grammaires formelles et contextuelles

**Types de grammaire :**

## Grammaires sensibles au contexte (Type 1)

**Exemple** :
Accord entre sujet et verbe dans une langue :

Sujet (singulier) Verbe (singulier) → Phrase
Sujet (pluriel) Verbe (pluriel) → Phrase

# Grammaires formelles et contextuelles

**Types de grammaire :**

## Grammaires récursivement énumérables (Type 0))

Les grammaires de type 0 permettent des règles de production totalement libres, sauf qu'au moins un non-terminal doit apparaître dans la partie gauche

Une règle a la forme : $\alpha \rightarrow \beta$
Où α et β sont des chaînes de terminaux et/ou non-terminaux.

# Grammaires formelles et contextuelles

**Types de grammaire :**

## Grammaires récursivement énumérables (Type 0))

- Ces grammaires génèrent des **langages récursivement énumérables**, les plus généraux de la hiérarchie.
- Peuvent modéliser tout calcul pouvant être effectué par une machine de Turing.
- Utilisé en théorie des automates et des langages formels
- Pas directement utilisée en TLN en raison de sa complexité

# Grammaires formelles et contextuelles

**Types de grammaire :**

## Grammaires récursivement énumérables (Type 0))

**Soit G = (N, T, P, S) où :**
**N = {S, A, B} est l'ensemble des symboles non-terminaux**
**T = {a, b, c} est l'ensemble des symboles terminaux**
**S est le symbole de départ**
**P est l'ensemble des règles de production suivantes :**
**S → aSBC**
**S → aBC**
**CB → BC**
**aB → ab**
**bB → bb**
**bC → bc**
**cC → cc**
**Cette grammaire génère des chaînes de la forme** $a^n b^n c^n$

# Grammaires formelles et contextuelles

**Types de grammaire :**

## Grammaires sensibles au contexte (Type 1)

```python
class CSGValidator:
    def __init__(self):
        # Définition de la grammaire qui génère a^n b^n c^n
        self.rules = [
            ('S', 'aSBC'),
            ('S', 'aBC'),
            ('CB', 'BC'),
            ('aB', 'ab'),
            ('bB', 'bb'),
            ('bC', 'bc'),
            ('cC', 'cc')
        ]
        self.start_symbol = 'S'
```

# Grammaires formelles et contextuelles

**Types de grammaire :**

## Grammaires sensibles au contexte (Type 1)

```
def is_valid_string(self, input_string):
    if not input_string:
        return False

    # Validation rapide: la chaîne doit contenir uniquement a, b, c
    if not all(char in 'abc' for char in input_string):
        return False
    a_count = 0
    b_count = 0
    c_count = 0
            i = 0
    while i < len(input_string) and input_string[i] == 'a':
        a_count += 1
        i += 1
```

# Grammaires formelles et contextuelles

## Récapitulation sur les types des grammaires

| | Structure des règles | Langages générés | Complexité | Applications |
|---|---|---|---|---|
| Types 3 | $A \rightarrow aB$ ou $A \rightarrow a$ | Langages réguliers | Très faible | Analyse lexicale, expressions régulières. |
| Types 2 | $A \rightarrow \alpha$ | Langages hors-contexte | Modérée | Parsing syntaxique, structures imbriquées. |
| Types 1 | $\alpha A \beta \rightarrow \alpha \gamma \beta$ | Langages sensibles au contexte | Élevée | Morphologie, accords grammaticaux. |
| Types 0 | $\alpha \rightarrow \beta$ | Langages récursivement énumérables | Très élevée | Théorie des langages formels |

02

# Analyse syntaxique basée sur les règles

# Analyse syntaxique basée sur les règles

## Définition

- L'analyse syntaxique basée sur des règles est une approche clé dans la validation des langages formels

- Elle repose sur l'application de règles de production définies dans une grammaire, afin de structurer une chaîne de symboles d'entrée sous la forme d'un arbre syntaxique

- Une règle de production associe un non-terminal à une séquence de symboles terminaux et non-terminaux, permettant ainsi de décomposer ou de générer des chaînes de caractères valides

# Analyse syntaxique basée sur les règles

**Méthodes d'Analyse Syntaxique :**

**Analyse Descendante (Top-Down Parsing)**

- L'analyse commence avec le symbole de départ de la grammaire (généralement un non-terminal) et cherche à réduire cette catégorie vers les terminaux en appliquant les règles de production.

- Le processus suit la structure de la grammaire, en appliquant successivement les règles pour tenter de dériver la chaîne d'entrée

- Une approche courante dans l'analyse descendante est **l'analyse récursive descendante**, où chaque non-terminal est associé à une fonction récursive qui applique les règles de production correspondantes.

# Analyse syntaxique basée sur les règles

## Méthodes d'Analyse Syntaxique :

### Méthodes de l’analyse Descendante (Top-Down Parsing)

- Algorithme LL(k): Left-to-right, Leftmost derivation, avec k symboles d'anticipation

- Analyse récursive descendante: implémentation directe par des fonctions récursives

# Analyse syntaxique basée sur les règles

## Méthodes d'Analyse Syntaxique : Descendante vs Montante

### Exemple d'Analyse Descendante :

Prenons l'exemple de l'expression arithmétique (3 + 4) * 5 avec une grammaire donnée. L'analyse descendante commencerait par le symbole de départ Expression et appliquerait les règles suivantes :

1. Expression → Expression + Terme
2. Puis, Expression → Terme (en raison de la structure de l'expression)
3. Enfin, Terme → Facteur, où Facteur devient ( Expression ), et ainsi de suite jusqu'à obtenir une dérivation complète

# Analyse syntaxique basée sur les règles

## Méthodes d'Analyse Syntaxique : Descendante vs Montante

### Analyse Montante (Bottom-Up Parsing)

Contrairement à l'analyse descendante, l'analyse montante commence avec les terminaux (symboles de la chaîne d'entrée) et essaie de les combiner en appliquant des réductions successives pour obtenir le symbole de départ

# Analyse syntaxique basée sur les règles

## Méthodes d'Analyse Syntaxique : Descendante vs Montante

### Analyse Montante (Bottom-Up Parsing)

- **Algorithme LR(k):** Left-to-right, Rightmost derivation in reverse, avec k symboles d'anticipation
- **Algorithme CYK (Cocke-Younger-Kasami)**: utilise la programmation dynamique pour l'analyse

# Analyse syntaxique basée sur les règles

## Méthodes d'Analyse Syntaxique : Descendante vs Montante

### Analyse Montante (Bottom-Up Parsing) Algorithme CYK

Pour la phrase "Le chat mange":

**Grammaire (en CNF) :**

1. S→NP VP (une phrase est constituée d'un groupe nominal suivi d'un groupe verbal)
2. NP→Det N (un groupe nominal est un déterminant suivi d'un nom)
3. VP→V NP (un groupe verbal est un verbe suivi d'un groupe nominal)
4. Det→Le (le déterminant "Le")
5. N→chat (le nom "chat")
6. V→mange(le verbe "mange")

# Analyse syntaxique basée sur les règles

## Méthodes d'Analyse Syntaxique : Descendante vs Montante

### Analyse Montante (Bottom-Up Parsing) Algorithme CYK

Pour la phrase "Le chat mange":

**Étapes de l'algorithme CYK :**

**1. Initialisation** :

- Nous commençons par préparer une table vide où chaque cellule correspond à un sous-ensemble de la phrase

- La phrase "Le chat mange" a 3 mots, donc nous aurons une table de 3x3 (pour les 3 mots, avec des sous-parties qui sont des combinaisons possibles).

# Analyse syntaxique basée sur les règles

## Méthodes d'Analyse Syntaxique : Descendante vs Montante

### Analyse Montante (Bottom-Up Parsing) Algorithme CYK

Pour la phrase "Le chat mange":

**Étapes de l'algorithme CYK :**

**2. Remplissage de la diagonale** (analyse des mots individuels) :

Le mot "Le" peut être un **Det**, donc on place **Det** dans la cellule (1,1).
Le mot "chat" peut être un **N**, donc on place **N** dans la cellule (2,2).
Le mot "mange" peut être un **V**, donc on place **V** dans la cellule (3,3)

# Analyse syntaxique basée sur les règles

## Méthodes d'Analyse Syntaxique : Descendante vs Montante

### Analyse Montante (Bottom-Up Parsing) Algorithme CYK

Pour la phrase "Le chat mange":

| | 1 | 2 | 3 |
|---|---|---|---|
| 1 | Det | | |
| 2 | | N | |
| 3 | | | V |

# Analyse syntaxique basée sur les règles

## Méthodes d'Analyse Syntaxique : Descendante vs Montante

### Analyse Montante (Bottom-Up Parsing) Algorithme CYK

Pour la phrase "Le chat mange":

**3. Analyse des sous-phrases** : À partir de la diagonale, nous analysons les combinaisons possibles entre les mots pour construire des sous-phrases :

- Dans la cellule (1,2), nous cherchons une combinaison entre **Det** (dans (1,1)) et **N** (dans (2,2)). La règle NP→Det N nous permet de conclure que la cellule (1,2) peut contenir **NP**.
- Dans la cellule (2,3), nous cherchons une combinaison entre **N** (dans (2,2)) et **V** (dans (3,3)). Il n'y a pas de règle directe pour cela, donc la cellule reste vide.

# Analyse syntaxique basée sur les règles

## Méthodes d'Analyse Syntaxique : Descendante vs Montante

### Analyse Montante (Bottom-Up Parsing) Algorithme CYK

Pour la phrase "Le chat mange":

| | 1 | 2 | 3 |
|---|---|---|---|
| 1 | Det | NP | |
| 2 | | N | |
| 3 | | | V |

# Analyse syntaxique basée sur les règles

## Méthodes d'Analyse Syntaxique : Descendante vs Montante

### Analyse Montante (Bottom-Up Parsing) Algorithme CYK

Pour la phrase "Le chat mange":

**4. Combiner les sous-phrases plus grandes** :

Dans la cellule (1,3), nous cherchons une combinaison entre **NP** (dans (1,2)) et **V** (dans (3,3)). La règle VP→V NP nous permet de conclure que la cellule (1,3) peut contenir **VP**.
La table finale :

# Analyse syntaxique basée sur les règles

## Méthodes d'Analyse Syntaxique : Descendante vs Montante

### Analyse Montante (Bottom-Up Parsing) Algorithme CYK

Pour la phrase "Le chat mange":

| | 1 | 2 | 3 |
|---|---|---|---|
| 1 | Det | NP | VP |
| 2 | | N | |
| 3 | | | V |

# Analyse syntaxique basée sur les règles

## Méthodes d'Analyse Syntaxique : Descendante vs Montante

### Analyse Montante (Bottom-Up Parsing) Algorithme CYK

Pour la phrase "Le chat mange":

**5. Vérification de la phrase complète** : Si la cellule en haut à droite (1,3) contient **S** (la règle de départ), alors la phrase est valide selon la grammaire.

Dans cet exemple, la cellule (1,3) contient **VP**, mais pas **S**.

C à d que la phrase "Le chat mange" ne peut pas être générée directement par cette grammaire.

Pour obtenir **S**, il faudrait que le groupe nominal (NP) soit suivi par un groupe verbal (VP), ce qui pourrait être amélioré en réévaluant ou en ajustant la grammaire.

# Analyse syntaxique basée sur les règles

## Méthodes d'Analyse Syntaxique : Descendante vs Montante

### Analyse Montante (Bottom-Up Parsing) Algorithme CYK

Pour la phrase "Le chat mange": Appliquer CYK Avec cette grammaire

S→NP VP *(une phrase est un groupe nominal suivi d'un groupe verbal)*
NP→Det N *(un groupe nominal est un déterminant suivi d'un nom)*
VP→V *(un groupe verbal peut être simplement un verbe)*
Det→Le *(le déterminant "Le")*
N→chat *(le nom "chat")*
V→mange *(le verbe "mange")*

# Analyse syntaxique basée sur les règles

## Méthodes d'Analyse Syntaxique : Descendante vs Montante

### Limites de l'analyse basée sur les règles

- Difficulté à gérer l'ambiguïté syntaxique
- Complexité d'élaboration des règles pour couvrir tous les cas
- Manque de robustesse face aux erreurs et aux constructions non-standard
- Difficulté à s'adapter à différentes langues ou domaines

# Analyse syntaxique basée sur les règles

## Arbres Syntaxiques : Représentation Hiérarchique

- L'objectif principal de l'analyse syntaxique est de générer un arbre syntaxique, une structure hiérarchique qui représente la composition de l'expression en fonction des règles de la grammaire.

- Chaque nœud de l'arbre représente une règle de production appliquée, et les feuilles de l'arbre sont les symboles terminaux.

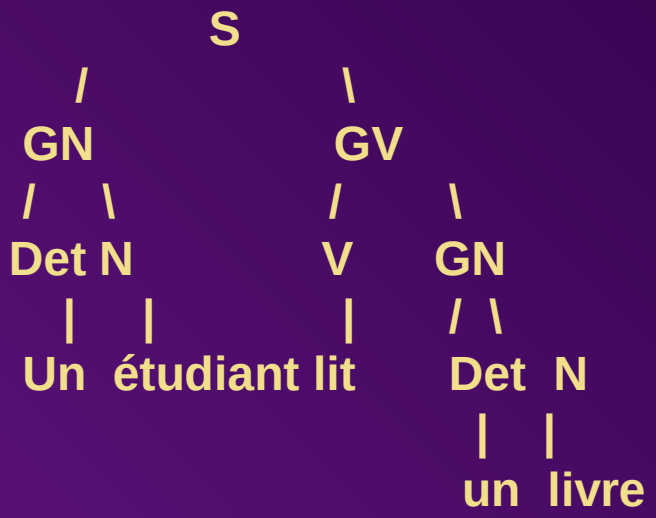
# Analyse syntaxique basée sur les règles

## Types d’arbres syntaxiques

### 1. Arbres de constituants (ou de syntagmes)

Ces arbres divisent la phrase en syntagmes hiérarchiques. Ils montrent comment les mots se regroupent pour former des unités grammaticales plus larges.

Exemple : Un étudiant lit un livre



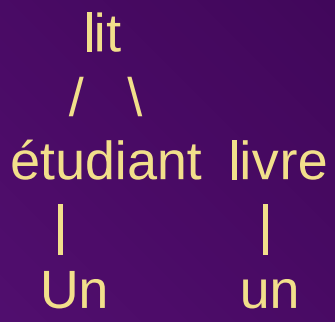
# Analyse syntaxique basée sur les règles

## Types d’arbres syntaxiques

### 1. Arbres de dépendance

Ces arbres représentent les relations de dépendance entre les mots. Chaque mot, sauf la racine, dépend d'un autre mot.

Exemple : Un étudiant lit un livre



# Analyse syntaxique basée sur les règles

## Propriétés de l'Analyse Syntaxique

### Ambiguïté syntaxique

Une phrase peut avoir plusieurs arbres syntaxiques valides, révélant une ambiguïté:

Exemple : "Elle regarde l'homme avec un télescope«

Interprétation 1 : Elle utilise un télescope pour regarder l'homme
Interprétation 2 : Elle regarde un homme qui possède un télescope

Ces deux interprétations donnent deux arbres syntaxiques différents

# 03 Méthodes d'analyse syntaxique probabiliste

# Méthodes d'analyse syntaxique probabiliste

## Introduction

- L’analyse syntaxique probabiliste utilise des techniques statistiques pour résoudre les ambiguïtés syntaxiques qui surviennent dans le traitement automatique des langues.

- En informatique, cette approche repose sur l'idée que la structure syntaxique la plus probable d'une phrase peut être identifiée à l'aide de modèles probabilistes, souvent formés à partir de grandes quantités de données linguistiques

- Ces méthodes sont particulièrement utiles pour traiter des langues naturelles où les structures syntaxiques peuvent être ambiguës.

# Méthodes d'analyse syntaxique probabiliste

## Types d'analyse syntaxique probabiliste

### 1. Grammaires Hors Contexte Probabilistes (PCFG - Probabilistic Context-Free Grammars)

Les **grammaires hors contexte** (CFG) sont couramment utilisées pour décrire la syntaxe des langages formels.

Lorsqu'on les enrichit avec des probabilités, on obtient des **grammaires hors contexte probabilistes** (PCFG).

# Méthodes d'analyse syntaxique probabiliste

## Types d'analyse syntaxique probabiliste

### Grammaires Hors Contexte Probabilistes (PCFG - Probabilistic Context-Free Grammars)

Une PCFG étend les grammaires hors-contexte en associant une probabilité à chaque règle de production:

$A \rightarrow \beta$ [p]

Où:

- A est un symbole non-terminal
- β est une séquence de symboles (terminaux ou non-terminaux)
- p est la probabilité que A se réécrive en β (avec $\Sigma p = 1$ pour toutes les règles ayant A en partie gauche)

# Méthodes d'analyse syntaxique probabiliste

## Types d'analyse syntaxique probabiliste

### Exemple de grammaires Hors Contexte Probabilistes

Une grammaire hors contexte simple pourrait inclure une règle comme :

$$S \rightarrow NP\ V\ P$$

Cette règle peut être enrichie avec une probabilité telle que :

$$P(S \rightarrow NP\ VP) = 0.7$$

CAD qu'il y a 70% de chances que la phrase soit composée d'un syntagme nominal (NP) suivi d'un syntagme verbal (VP).

# Méthodes d'analyse syntaxique probabiliste

## Types d'analyse syntaxique probabiliste

### 2. Grammaires de Dépendance Probabilistes (PDG - Probabilistic Dependency Grammars)

Les **grammaires de dépendance** sont une alternative aux grammaires hors contexte, où l'accent est mis sur les relations de dépendance entre les mots d'une phrase

Les **grammaires de dépendance probabilistes** ajoutent une dimension probabiliste à ces relations, où chaque dépendance entre deux mots est associée à une probabilité.

# Méthodes d'analyse syntaxique probabiliste

## Types d'analyse syntaxique probabiliste

### Exemple de grammaires de Dépendance Probabilistes

Dans une phrase comme "Le chat mange la souris", une grammaire de dépendance probabiliste pourrait spécifier qu'il est 80% probable que "mange" dépende de "chat" et 70% probable que "mange" dépende de "souris".

# Méthodes d'analyse syntaxique probabiliste

## Types d'analyse syntaxique probabiliste

### 4. Modèles de Langage Basés sur les Réseaux Neuronaux (Deep Learning)

- Avec l'avènement de l'apprentissage profond, les **réseaux neuronaux** ont gagné en popularité pour l'analyse syntaxique probabiliste.

- Les modèles modernes, comme **BERT** ou **GPT**, utilisent des architectures de type Transformer pour apprendre des représentations syntaxiques complexes à partir de données massives.

# Méthodes d'analyse syntaxique probabiliste

## Types d'analyse syntaxique probabiliste

### Modèles de Langage Basés sur les Réseaux Neuronaux (Deep Learning)

Ces modèles apprennent non seulement à attribuer des étiquettes grammaticales mais aussi à prédire des structures syntaxiques complexes (par exemple, les relations de dépendance entre les mots), sans nécessiter une grammaire explicite.

# Applications de l'Analyse Syntaxique en TLN