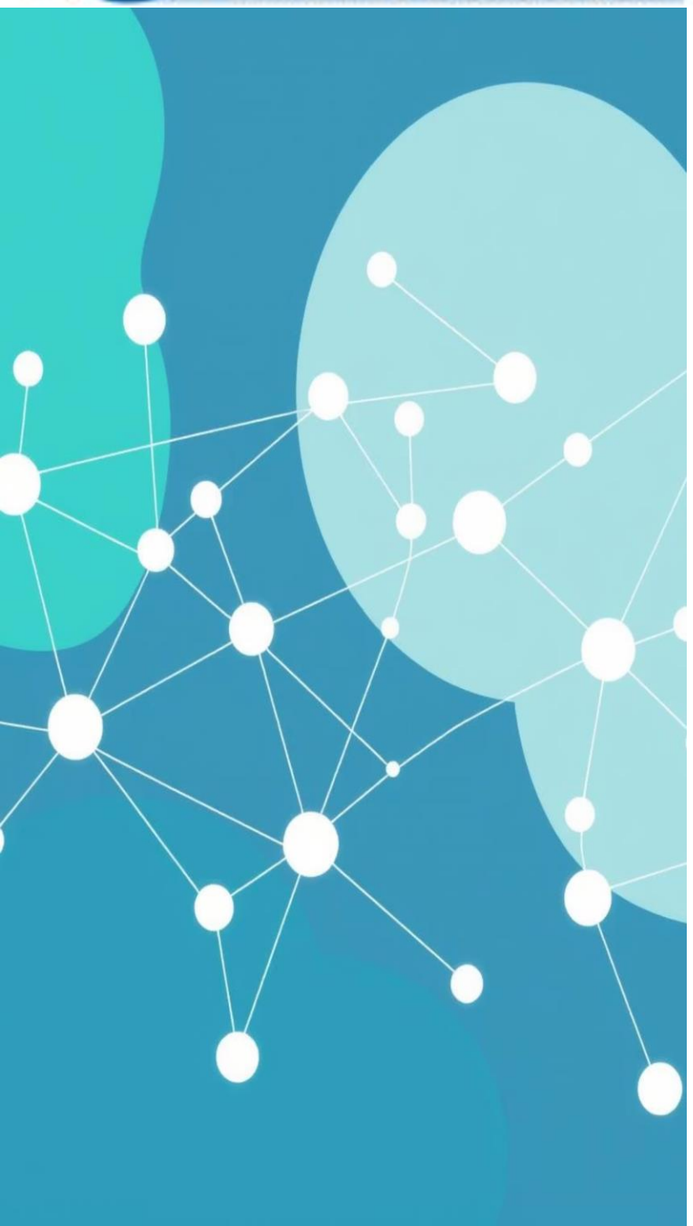


Théorie du langage et Automates: Introduction générale

2L GLSI

H. Merdassi

Introduction

- Les langages sont les supports naturels de communication.
- Ils permettent aux hommes d'échanger des informations et des idées et ils leur permettent également de communiquer avec les machines
- Les langages utilisés dans la vie de tous les jours entre être humain sont dits naturels
- Ils sont généralement informels et ambigus et demandent toute la subtilité d'un cerveau humain pour être interprétés correctement
- Les langages créés par l'homme pour communiquer avec les ordinateurs sont des langages dits artificiels ou formels
- Ils doivent être formalisés et non ambigus pour pouvoir être interprétés par une machine

Définition générale

- La théorie des langages formels est une branche de l'informatique théorique qui étudie les langages, c'est-à-dire des ensembles de mots construits à partir d'un alphabet donné. Elle s'intéresse à trois grandes questions :
  - Comment définir un langage ? (par une grammaire, une expression régulière, un automate)
  - Comment reconnaître si un mot appartient à ce langage ?
  - Quels sont les types de langages et leurs limites de reconnaissance ?

- Un langage peut être simple, comme l'ensemble des mots binaires de longueur paire, ou beaucoup plus complexe, comme le langage de tous les programmes valides écrits en Java ou en C.

Pourquoi étudier cette théorie ?

- **Comprendre la nature du calcul et de la représentation de l'information**

La théorie des langages permet de formaliser la notion de "langage" au sens large : qu'il s'agisse d'un langage de programmation, d'un protocole de communication, d'un langage naturel ou d'un système de codage.

Elle offre un cadre **mathématique** rigoureux pour analyser ce qui peut (ou ne peut pas) être exprimé, reconnu ou calculé.

- **La théorie des langages constitue un pilier de l'informatique.**

Elle permet de comprendre :

- Ce que les ordinateurs peuvent calculer.
- Ce qu'ils ne pourront jamais calculer (problèmes indécidables).
- Comment classer les problèmes selon leur complexité.

C'est une étape essentielle pour relier **mathématiques** et **informatique**.

Pourquoi étudier cette théorie ?

Vérification et sécurité

•Vérification de protocoles de communication (réseaux, télécoms).
•Vérification de systèmes critiques (avionique, ferroviaire, ascenseurs, etc.) pour garantir qu'aucune séquence d'actions dangereuses n'est possible.
•Vérification de contrats intelligents (smart contracts) en blockchain.

Traitement du langage naturel

•Correcteurs orthographiques et grammaticaux.
•Reconnaissance vocale.
•Traduction automatique.
Tous utilisent, à la base, des modèles de langages et des automates.

Bio-informatique et sciences des données

•L'ADN peut être vu comme une suite de symboles sur l'alphabet ${A, C, G, T}$.
•Les chercheurs utilisent des automates pour rechercher des motifs dans les séquences génétiques.
•Les algorithmes de compression et de reconnaissance de motifs en big data reposent aussi sur des concepts similaires

Théorie des langages

- **Fondements de la théorie des langages**

• La théorie des langages étudie les structures syntaxiques et sémantiques des langages. Elle vise à comprendre la manière dont les langages sont formés et interprétés, ainsi que les règles qui les régissent.

- **Langages formels**

• Elle explore les langages formels tels que les grammaires formelles et les automates, qui sont utilisés pour représenter et analyser la syntaxe des langages de programmation.

- **Applications dans la compilation**

• La théorie des langages est essentielle pour la conception de langages de programmation, les analyseurs syntaxiques et sémantiques, ainsi que pour la compréhension des langages naturels et leur traduction en langages formels.

Limites des ordinateurs

- En apprenant à programmer, on a souvent l'impression que toute tâche de calcul est réalisable par un programme suffisamment long et complexe. **Cette intuition est fausse et certaines tâches n'admettent aucun algorithme.**

- **Certains problèmes qui ont des solutions algorithmiques n'ont pas de solution algorithmique efficace.**

Exemple

- **Problème** : vérifier si une suite de parenthèses est bien formée.
  - Exemple correct : (( ) ( ))
  - Exemple incorrect : ( ) (( )

- **Pour résoudre ce problème** :
  - On définit une grammaire qui décrit toutes les suites correctes.
  - On construit un automate à pile capable de lire chaque mot et de dire s'il est correct ou non.

Automates : des machines abstraites pour reconnaître les langages

Les automates sont des modèles mathématiques de machines abstraites conçues pour reconnaître les mots qui appartiennent à un langage formel donné. Ils lisent un mot symbole par symbole et, en fonction de leur état interne et du symbole lu, ils passent à un nouvel état. À la fin du mot, l'automate détermine si le mot est "accepté" (fait partie du langage) ou "rejeté".

Automates finis

Les plus simples, avec un nombre limité d'états, reconnaissent les langages réguliers.

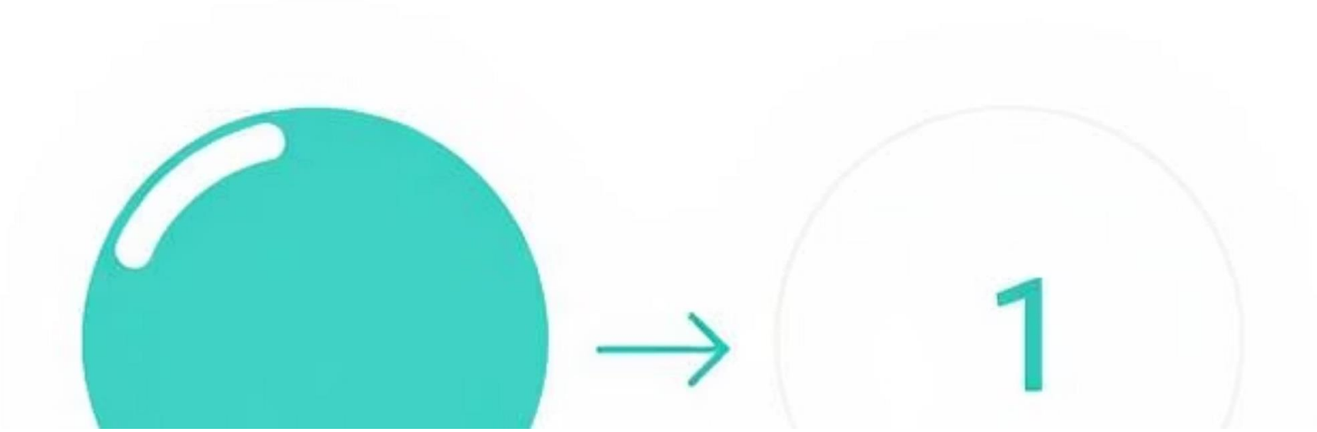
Automates à pile

Disposent d'une mémoire "pile" supplémentaire, reconnaissent les langages hors-contexte.

Machines de Turing

Le modèle de calcul le plus puissant, avec une mémoire infinie, reconnaissent les langages récursivement énumérables.

Chaque type d'automate a une puissance de reconnaissance différente, ce qui nous permet de classer les langages selon leur complexité. Par exemple, un automate peut être conçu pour reconnaître si un nombre binaire est premier.



Automate fini : reconnaître des motifs

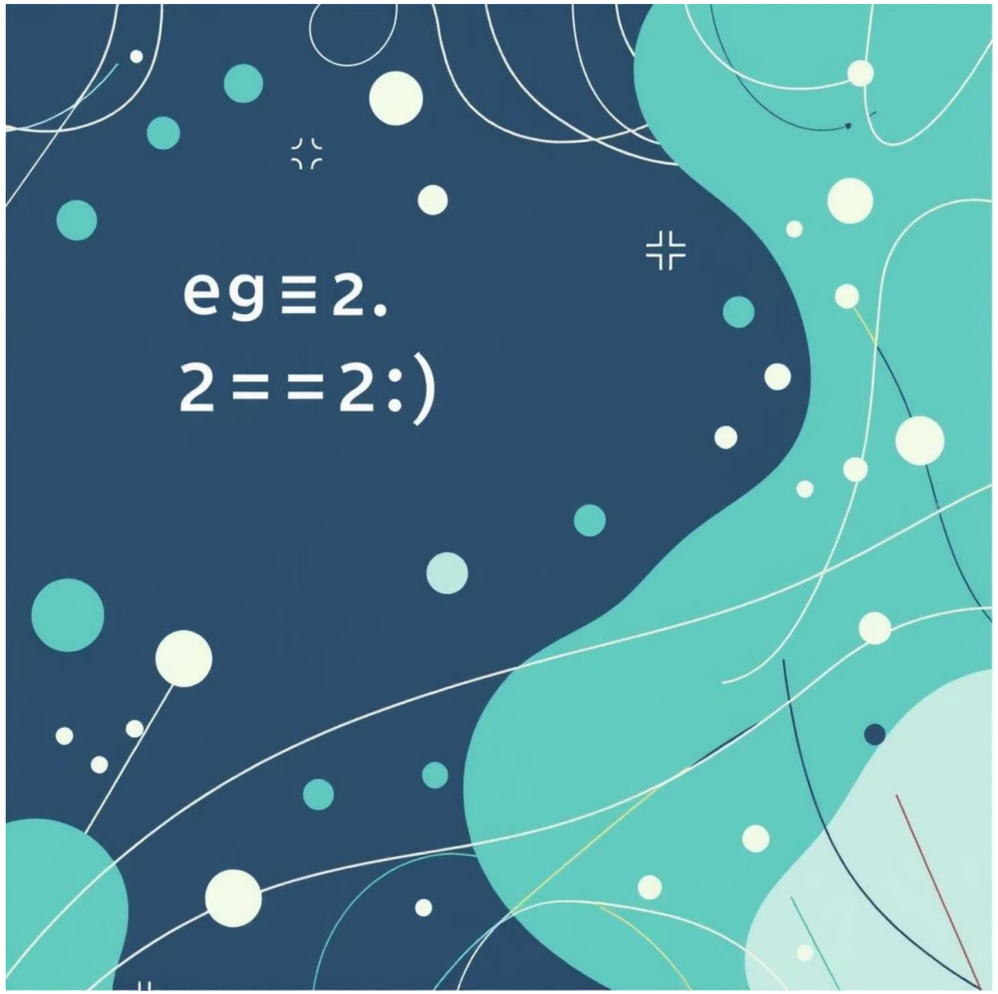
Un automate fini est comme un petit robot qui suit un chemin basé sur ce qu'il lit. Il a un nombre fini d'états et passe de l'un à l'autre selon les symboles qu'il rencontre. Si, après avoir lu tout le mot, il se retrouve dans un "état final", alors le mot est reconnu. Ce mécanisme simple est extrêmement puissant pour identifier des motifs spécifiques.

Automates finis et langages réguliers

La relation entre les automates finis et les langages réguliers est une pierre angulaire de la théorie des langages formels. Le théorème de Kleene établit que la classe des langages acceptés par les automates finis est exactement la classe des langages réguliers.

- **AFD vs AFN :** Les Automates Finis Déterministes (AFD) et les Automates Finis Non-déterministes (AFN) peuvent sembler différents, mais ils sont équivalents en termes de puissance de reconnaissance. Tout langage reconnu par un AFN peut également l'être par un AFD, et vice-versa.
- **Expressions Régulières :** Ces automates sont intimement liés aux expressions régulières, qui sont des séquences de caractères permettant de décrire des motifs de texte. Elles sont utilisées partout, de la recherche de texte aux filtres de données.

La compréhension de ces concepts est essentielle pour quiconque travaille avec l'analyse lexicale, la recherche de motifs ou la validation de données.



Automates à pile et langages hors-contexte

Pour des structures plus complexes que de simples motifs (comme les parenthèses imbriquées ou les balises XML), les automates finis ne suffisent plus. C'est là qu'interviennent les automates à pile.

> Les automates à pile (AP) sont des automates finis augmentés d'une mémoire supplémentaire appelée "pile" (stack). Cette pile leur permet de stocker et de récupérer des informations selon le principe du "dernier entré, premier sorti" (LIFO), ce qui est crucial pour gérer les dépendances structurelles.

Les AP sont capables de reconnaître les langages hors-contexte (LHC), qui sont la base de la modélisation de la syntaxe de la plupart des langages de programmation. Par exemple, lors de la compilation d'un programme en C, l'analyseur syntaxique utilise implicitement un automate à pile pour vérifier la validité de la structure du code (imbrication des accolades, des fonctions, etc.).

Un exemple classique de LHC est le langage des expressions parenthésées bien formées : `(((())))` est valide, mais `()(()` ne l'est pas.

Machines de Turing : le modèle universel de calcul

La machine de Turing, conçue par Alan Turing en 1936, est le modèle abstrait le plus puissant de calcul. Elle est capable de simuler n'importe quel algorithme exécuté par un ordinateur.

- **Mémoire illimitée :** Contrairement aux automates précédents, la machine de Turing dispose d'un ruban de lecture/écriture infini, lui conférant une capacité de mémoire illimitée.
- **Modèle universel :** Toute fonction calculable par un algorithme peut être calculée par une machine de Turing. C'est le fondement de la théorie de la calculabilité.
- **Décidabilité :** La théorie des machines de Turing nous aide à comprendre les limites de ce qui peut être calculé. Certains problèmes sont décidables (la machine peut toujours donner une réponse), d'autres sont indécidables (la machine peut ne jamais s'arrêter).

Un exemple de problème décidable est le test de primalité : une machine de Turing peut toujours déterminer si un nombre donné est premier ou non en un temps fini.

Applications concrètes des automates et langages

La théorie des langages et automates, bien que théorique, est au cœur de nombreuses technologies que nous utilisons quotidiennement.

1

Compilation

Les compilateurs utilisent les automates pour l'analyse lexicale (reconnaissance des mots clés, identificateurs) et syntaxique (vérification de la structure du code source).

2

Vérification formelle

Garantir la fiabilité des systèmes critiques (aéronautique, médical) en vérifiant mathématiquement leur comportement à l'aide de modèles basés sur les automates.

3

IA & NLP

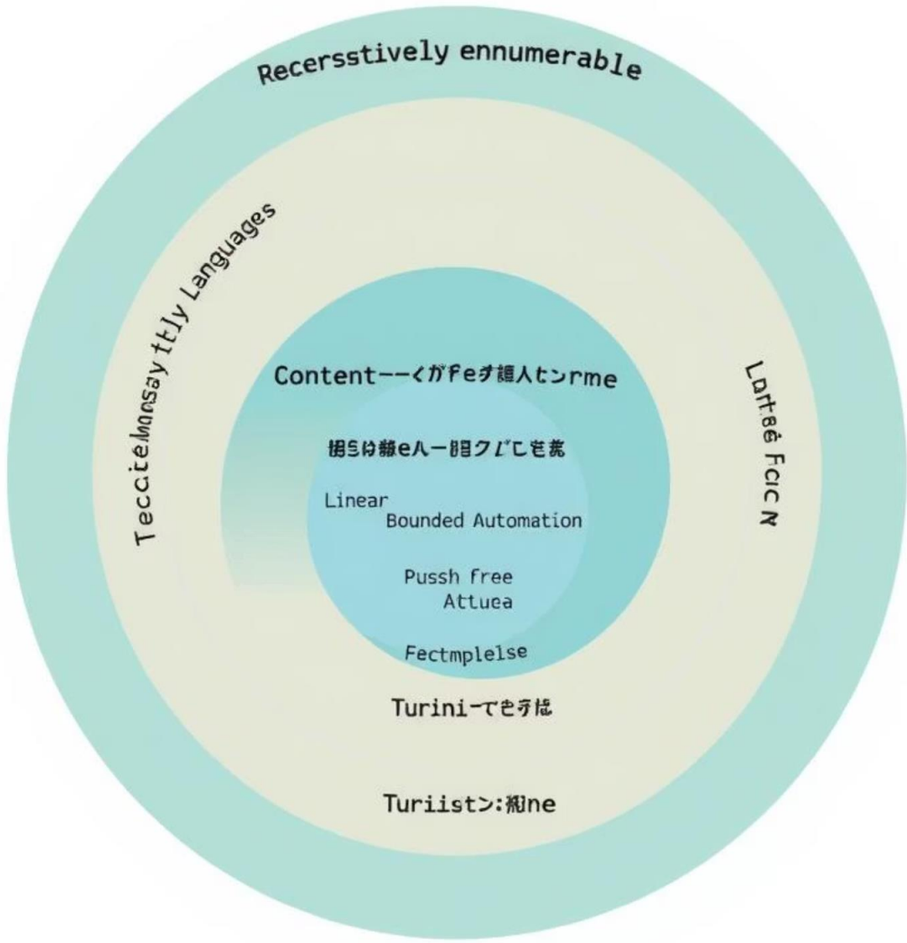
Le Traitement du Langage Naturel (NLP) et l'Intelligence Artificielle s'appuient sur ces théories pour la reconnaissance de la parole, la traduction automatique et la compréhension sémantique.

Ces applications démontrent la pertinence continue de ces fondations théoriques dans le développement de solutions informatiques avancées.

La Hiérarchie de Chomsky

La hiérarchie de Chomsky classe les langages formels en quatre types, du plus simple au plus complexe, chacun étant reconnaissable par un type spécifique d'automate. Cette hiérarchie est fondamentale pour comprendre la puissance expressive des différents formalismes linguistiques.

- **Type 3 : Langages Réguliers** – Reconnu par les automates finis.
- **Type 2 : Langages Hors-Contexte** – Reconnu par les automates à pile.
- **Type 1 : Langages Contextuels** – Reconnu par les automates linéairement bornés.
- **Type 0 : Langages Récursivement Énumérables** – Reconnu par les machines de Turing.



Conclusion : Pourquoi maîtriser la théorie des langages et automates ?

Comprendre les fondements

Acquérir une compréhension profonde des mécanismes sous-jacents aux langages informatiques et aux processus de calcul.

Concevoir des outils fiables

Développer des compilateurs, des analyseurs et des systèmes de vérification robustes et performants.

Innover en technologie

Participer aux avancées dans l'IA, le NLP et la cybersécurité, des domaines en constante évolution.

Cette théorie n'est pas seulement un domaine académique ; elle est le moteur silencieux derrière une grande partie de l'innovation technologique actuelle et future. La maîtriser, c'est se donner les moyens de comprendre et de façonner le monde numérique.