Chapitre 1

Notions fondamentales en théorie des langages

1. Rappels sur la théorie des ensembles

1..1 Définitions

Définition 1.1 : Un ensemble est une collection d'objets sans répétition, chaque objet est appelé un élément. L'ensemble ne contenant aucun élément est dit l'ensemble vide et est noté $\emptyset$.

Un ensemble peut être défini de plusieurs façons, chacune a ses avantages et ses inconvénients. Notons, au passage, le parallèle existant entre la théorie des ensembles et la logique des prédicats, ce qui fait que beaucoup de notions relatives aux ensembles sont définies en termes de la logique des prédicats.

Définition en extension

Définir un ensemble par extension revient simplement à donner la liste de ses éléments. Prenons l'exemple des chiffres entiers de 0 à 9 définis par l'écriture $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$. Cette définition est utile lorsque le nombre d'éléments d'un ensemble est fini et n'est pas très important. Notons que lorsqu'un élément x figure dans un ensemble A, cela est noté par $x \in A$. Par certains artefacts, on arrive à noter des ensembles infinis mais cela n'est pas pratique.

Définition par compréhension

Définir un ensemble A par compréhension revient à définir un prédicat P qui ne peut prendre la valeur "vraie" que si, et seulement si, il est évalué pour un élément de A, on écrira alors : $A = \{x|P(x)\}$. Comme exemple, nous pouvons définir l'ensemble des entiers naturels multiples de 3 par $\{x|x \bmod 3 = 0\}$ ($\bmod$ est l'opération modulo).
La définition par compréhension est plus pratique car elle permet de définir aisément des ensembles ayant un nombre infini d'éléments (comme c'est le cas pour l'exemple donné). Elle peut néanmoins mener à un paradoxe appelé paradoxe de Russel si on autorise P à être n'importe quel prédicat.

Définition par induction

Définir un ensemble A par induction revient à montrer comment les éléments de A sont construits. On parlera de preuve d'appartenance pour tout élément appartenant à A. Concrètement, la définition par induction de A nécessite de définir un ensemble fini d'éléments de A (appelé les triviaux de A et noté $triv(A)$) dont on suppose l'appartenance à A, une ou plusieurs fonctions $f_i : A \longrightarrow A$ et des règles de la forme :

$$x \rightarrow f_i(x)$$

signifiant que si x est un élément de A alors $f_i(x)$ est également un élément de A. On écrira $A = \{triv(A); x \in A \longrightarrow f_i(x) \in A\}$.
Comme exemple, on peut définir les entiers naturels par $\mathbb{N} = \{0; x \in \mathbb{N} \rightarrow x + 1 \in \mathbb{N}\}$. La définition par induction permet, tout comme la définition par compréhension, de définir des ensembles infinis. Cependant, en permettant une meilleure structuration, elle peut simplifier la définition de certains ensembles complexes.

1..2 Démonstration de propriétés sur les ensembles

Souvent, on s'intéresse à une propriété Q censée être satisfaite par tout élément d'un ensemble A. Selon la définition adoptée de l'ensemble, on peut obtenir plusieurs types de démonstration :

Démonstration dans le cas d'une définition en extension

Dans le cas d'une définition en extension, Q est satisfaite par A si elle l'est pour tout élément A. En d'autres termes, il faut prendre chaque élément de A, le placer comme argument de Q puis vérifier si Q s'évalue à vrai. Évidemment, cette démonstration ne peut fonctionner que si l'ensemble A est fini. Elle n'est pas pratique si le nombre d'éléments est important.

Démonstration dans le cas d'une définition par compréhension

Supposons que $A = \{x|P(x)\}$. Dans ce cas, Q est satisfaite par A si on arrive à démontrer que $P(x) \rightarrow Q(x)$ en utilisant les axiomes de l'ensemble considéré. Bien sûr, on sera confronté au problème de complétude et démontrabilité des théorèmes (par exemple, on sait que quel que soit l'ensemble des axiomes définissant les entiers, il existera toujours des propriétés des nombres entiers que l'on ne pourra pas démontrer).

Démonstration dans le cas d'une définition par induction

Supposons que $A = \{triv(A); x \in A \rightarrow f(x) \in A\}$. Montrer qu'une propriété Q est satisfaite par A revient à :

1. Vérifier que Q est satisfaite pour tout élément de $triv(A)$ (faisable car cet ensemble est fini et généralement de petite taille) ;
2. Supposer que x satisfait Q, et démontrer que $f(x)$ satisfait également Q.

Cette méthode de démonstration s'appelle *démonstration par induction*. Elle est très utile pour démontrer des propriétés très complexes. Une de ses applications est la suivante : considérons l'ensemble des entiers naturels $\mathbb{N}$ défini précédemment (par induction), on cherche à vérifier la satisfaction d'une propriété Q sur les nombres entiers. On procède alors comme suit :

1. Vérifier que Q est satisfaite pour 0 ;
2. Supposer que Q est satisfaite pour un entier n et démontrer que Q est satisfaite pour $n + 1$.

On peut facilement s'apercevoir que le dernier raisonnement n'est autre que le raisonnement par récurrence. On vient, en fait, de montrer que ce raisonnement n'est qu'un cas particulier du raisonnement par induction.

1..3 Opérations sur les ensembles

On peut définir plusieurs opérations sur les ensembles. Certaines peuvent être définies pour toutes sortes de définitions d'ensembles, mais ce n'est malheureusement le cas dans tous les cas.

Inclusion et égalité

Soient A et B deux ensembles. On dit que A est inclus ou égal à B (et on note $A \subseteq B$) si $\forall x : x \in A \rightarrow x \in B$ (on dit que A est un sous-ensemble de B). On peut voir tout de suite que pour tout ensemble A, on a : $\emptyset \subseteq A$.

Si on utilise la définition par compréhension, c'est-à-dire $A = \{x|P(x)\}$ et $B = \{x|Q(x)\}$, alors $A \subseteq B$ tient si et seulement si $P \rightarrow Q$.

On parle d'égalité ($A = B$) lorsque $A \subseteq B$ et $B \subseteq A$. En compréhension, cela revient à montrer que : $P \leftrightarrow Q$.

Opérations binaires

Soit Ω un ensemble que l'on appellera l'ensemble référence. Soient A et B deux sous-ensembles quelconques de Ω définis par compréhension comme suit $A = \{x|P(x)\}$ et $B = \{x|Q(x)\}$, et définis par induction comme suit : $A = \{triv(A); x \rightarrow f(x)\}$ et $B = \{triv(B); x \rightarrow g(x)\}$. On définit alors les opérations suivantes :

- **Union :** notée $A \cup B$, elle comporte tout élément appartenant à A *ou* B, en d'autres termes, $A \cup B = \{x|x \in A \vee x \in B\}$. L'union est définie par compréhension comme suit : $A \cup B = \{x|P(x) \vee Q(x)\}$. Elle peut également être définie par induction comme suit : $A \cup B = \{triv(A) \cup triv(B); x \rightarrow f(x), x \rightarrow g(x)\}$.

- **Intersection :** notée $A \cap B$, elle comporte tout élément appartenant à A *et* B ; en d'autres termes, $A \cap B = \{x|x \in A \wedge x \in B\}$. L'intersection est définie par compréhension comme suit : $A \cup B = \{x|P(x) \wedge Q(x)\}$. Cependant, dans le cas général, on ne peut pas donner une définition inductive à l'intersection.

- **Différence :** notée $A - B$, elle comporte tout élément appartenant à A *et* qui n'appartient pas à B ; en d'autres termes, $A - B = \{x|x \in A \wedge x \notin B\}$. La différence est définie par compréhension comme suit : $A - B = \{x|P(x) \wedge \neg Q(x)\}$. Cependant, dans le cas général, on ne peut pas donner une définition inductive à la différence.

- **Complément :** notée $\overline{A} = \Omega - A$; en d'autres termes, $\overline{A} = \{x|x \in \Omega \wedge x \notin A\}$. Le complément est défini par compréhension comme suit : $\overline{A} = \{x|\neg P(x)\}$. Cependant, dans le cas général, on ne peut pas donner une définition inductive au complément.

- **Produit cartésien :** noté $A \times B$, c'est l'ensemble des paires (a, b) telles que $a \in A$ et $b \in B$; en d'autres termes, $A \times B = \{(x, y)|x \in A \wedge x \in B\}$. Le produit cartésien est défini par compréhension comme suit : $A \times B = \{(x, y)|P(x) \wedge Q(y)\}$. La définition par induction se fait comme suit : $A \times B = \{triv(A) \times triv(B); (x, y) \rightarrow (f(x), g(y))\}$.

Autres opérations

- **Cardinalité :** notée $card(A)$, c'est le nombre d'éléments de A. On a : $card(A \times B) = card(A)card(B)$.
- **L'ensemble des parties de** A **:** notée 2^A, c'est l'ensemble de tous les sous-ensembles de A. On a : $card(2^A) = 2^{card(A)}$.

Exemple 1.1 : calculs des opérations ensemblistes

$A = \{a, b\}$,$B = \{a, c\}$,$\Omega = \{a, b, c\}$:

- $A \cap B = \{a\}$;
- $A \cup B = \{a, b, c\}$;
- $A - B = \{b\}$;
- $\overline{A} = \{c\}$;
- $A \times B = \{(a, a), (a, b), (b, a), (b, c)\}$;
- $2^A = \{\emptyset, \{a\}, \{b\}, \{a, b\}\}$

2. Théorie des langages

Le langage a pour vocation d'assurer la communication entre différentes entités. Bien qu'il soit d'usage de voir le langage comme une caractéristique humaine, ou un objet de prédilection de l'intelligence de l'homme, la notion de langage est bien plus universelle.

La notion de langage est étroitement liée à la notion de codage. Lorsque deux entités, ne partageant pas une mémoire commune, souhaitent échanger des informations, il y a besoin de coder ces informations pour les transmettre via un support (l'air, fil électrique, onde électromagnétique, substance chimique, etc). Le scénario de base est alors le suivant :

— L'entité A souhaite envoyer une information à l'entité B.

— L'entité A code son message selon des règles connues par A et B.

— Le message codé est envoyé via un médium.

— L'entité B reçoit le message et procède à son décodage en appliquant des règles connues par A et B.

— L'entité B récupère l'information envoyée par A.

Une condition nécessaire au fonctionnement de ce schéma est l'utilisation de règles de codage et décodage communes et connues par A et B. Imaginons, par exemple, que l'entité A ne parle que l'arabe envoie un message à la personne B ne comprenant que l'anglais : il est impossible d'envoyer l'information dans ce cas.

Étant un domaine de l'informatique théorique, la théorie des langages couvre plutôt les langages formels : les langages construits selon des modèles mathématiques stricts et qui servent généralement à communiquer avec une machine. Il est important ici de faire la différence entre un langage formel et un langage naturel (utilisé par les humains). Ce dernier, bien qu'ayant des bases mathématiques comme celui des langages formels, possède la caractéristique d'être ambigu et d'utiliser la même *construction syntaxique* pour signifier des choses différentes. Son étude relève généralement du domaine de l'intelligence artificielle et ne sera pas couverte par ce cours.

Afin de formaliser les langages étudiés, la théorie des langages introduit un certain nombre de concepts nécessaires à l'étude d'un langage. Ces concepts partent de la notion du symbole pour arriver au langage et grammaire tout en passant par la notion du mot (qui est la brique élémentaire qui construit un langage).

Une question fondamentale nous servira de points de départ pour donner diverses définitions aux langages : comment répondre à la question "est-ce qu'un mot appartient à un langage ?". On définira trois points de vue pour répondre à cette question :

— Définition ensembliste : c'est une définition basique pour un langage. Elle permet de définir des langages et de raisonner dessus (généralement par une définition par compréhension).

— Définition par grammaire : c'est expliquer comment construire les éléments d'un langage. Il s'agit de la notion clé pour développer des analyseurs de programmes notamment les compilateurs.

— Définition par automate : un automate est tout simplement la machine (ou l'algorithme) qui répond à la question posée par oui ou non. Son étude est cruciale car elle permet de dire, d'abord, si l'on peut répondre à la question et, le cas échéant, évaluer le temps nécessaire pour le faire.

Disposer de ces trois points de vue n'est pas un luxe intellectuel, car chacun permet de répondre à la question posée de manière complémentaire à ce que font les autres points. Par exemple, pour les langages de programmation usuels, il est d'usage que l'algorithme d'analyse (compilateur) soit

construit à partir de la grammaire du langage. L'algorithme construit sera par la suite modifié pour faire davantage d'analyses.

Dans ce qui suit, nous allons d'abord passer en revue plusieurs définitions nécessaires à la compréhension du concept de langage.

2..1 Notions sur les mots

Définition 1.2 : Un symbole est une entité abstraite. Les lettres et les chiffres sont des exemples de symboles utilisés fréquemment, mais des symboles graphiques, des sons ou tous types de signaux peuvent également être employés (idéogrammes, signal lumineux, fumée, signal électrique,...).

Définition 1.3 : Un alphabet est un ensemble de symboles. Il est également appelé le vocabulaire.

Définition 1.4 : Un mot (ou bien une chaîne) défini sur un alphabet A est une séquence finie de symboles juxtaposés de A.

Exemple 1.2 : des mots avec leurs alphabets

— Le mot 1011 est défini sur l'alphabet $\{0, 1\}$;

— Le mot 1.23 est défini sur l'alphabet $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, .\}$.

2..2 Longueur d'un mot

Si w est un mot, alors sa longueur est définie comme étant le nombre de symboles contenus dans w, elle est noté par $|w|$. Par exemple, $|abc| = 3, |aabba| = 5$.

En particulier, on note le mot dont la longueur est nulle par ε : $|\varepsilon| = 0$. Pour bien comprendre ε, on peut faire l'analogie entre les mots et chaînes de caractères en langage C. Un mot n'est autre qu'une chaîne de caractères, la chaîne "abc" représente tout simplement le mot abc. La chaîne vide "" (un tableau dont la première case contient $\backslash 0$) représente alors le mot ε.

On définit également la longueur d'un mot w par rapport à un symbole $a \in A$ ($|w|_a$) comme étant le nombre d'occurrences de a dans w. Par exemple, $|abc|_a = 1, |aabba|_b = 2$.

Remarque : Dans ce cours, on suppose que tous les mots considérés sont de taille finie. Il est possible aussi de considérer des mots de longueur infinie mais leur étude dépasse le cadre de ce cours.

2..3 Concaténation des mots

Soient w_1 et w_2 deux mots définis sur l'alphabet A. La concaténation de w_1 et w_2 est un mot w défini sur le même alphabet. w est obtenu en écrivant w_1 suivi de w_2, en d'autres termes, on colle le mot w_2 à la fin du mot w_1 :

$$w_1 = a_1...a_n,\ w_2 = b_1b_2...b_m$$
$$w = a_1...a_nb_1b_2...b_m$$

La concaténation est notée par le point, mais il peut être omis s'il n'y a pas d'ambiguïté. On écrira alors : $w = w_1.w_2 = w_1w_2$.

Propriétés de la concaténation

Soient w, w_1 et w_2 trois mots définis sur l'alphabet A :

— La concaténation n'est généralement pas commutative ;

— $(w_1.w_2).w_3 = w_1.(w_2.w_3)$ (la concaténation est associative) ;

— $w.\varepsilon = \varepsilon.w = w$ (ε est un élément neutre pour la concaténation) ;

— $|w_1.w_2| = |w_1| + |w_2|$;

— $\forall a \in A : |w_1.w_2|_a = |w_1|_a + |w_2|_a$;

Remarque : L'ensemble des mots muni de l'opération concaténation forme ce qu'on appelle *monoïde*. Cette structure algébrique possède des propriétés pouvant être utilisées dans l'analyse des langages (on parle, par exemple, de reconnaissance par morphisme sur un monoïde).

L'exposant

L'opération $w.w$ est notée par w^2. En généralisant, on note $w^n = \underbrace{w...w}_{n\ fois}$. En particulier, l'exposant 0 fait tomber sur ε : $w^0 = \varepsilon$ (le mot w est répété 0 fois).

Le mot miroir

Soit $w = a_1a_2...a_n$ un mot sur A ($a_i \in A$). On appelle mot miroir de w, et on le note par w^R, le mot obtenu en écrivant w à l'envers, c'est-à-dire que $w^R = a_n...a_2a_1$. Il est donc facile de voir que $(w^R)^R = w$ ainsi que $(u.v)^R = v^R.u^R$.

Certains mots, appelés palindromes, sont égaux à leurs miroirs. En d'autres termes, on lit la même chose dans les deux directions. Si l'on considère l'alphabet X, l'ensemble des palindromes sont les mots de la forme ww^R ou waw^R tel que a est un seul symbole.

Préfixe et suffixe

Soit w un mot défini sur un alphabet A. Un mot x (resp. y) formé sur A est un préfixe (resp. suffixe) de w s'il existe un mot u formé sur A (resp. v formé sur A) tel que $w = xu$ (resp. $w = vy$). Si $w = a_1a_2...a_n$ alors tous les mots de l'ensemble $\{\varepsilon, a_1, a_1a_2, a_1a_2a_3, ..., a_1a_2...a_n\}$ sont des préfixes de w. De même, tous les mots de l'ensemble $\{\varepsilon, a_n, a_{n-1}a_n, a_{n-2}a_{n-1}a_n, ...,\ a_1a_2...a_n\}$ sont des suffixes de w.

Entrelacement (mélange)

Soient u et v deux mots définis sur un alphabet A tels que $u = u_1u_2...u_n$ et $v = v_1v_2...v_m$. On appelle entrelacement de u et v (on le note par $u \sqcup v$) l'ensemble des mots w qui mélangent les symboles de u de v tout en gardant l'ordre des symboles de u et de v. Formellement, $w = u \sqcup v = w_1w_2...w_{n+m}$ tel que $w_k \in A$ et si $w_k = u_i$ alors $\nexists i', k' : k' > k \wedge i' < i \wedge w_{k'} = u_{i'}$ (la même contrainte s'applique aussi au mot v).
Par exemple, $ab \sqcup ac = \{abac, aabc, aacb, acab\}$ (notons que cette opération est commutative).

En particulier, lorsque v se réduit un seul symbole, l'opération d'entrelacement se résume à l'insertion dudit symbole quelque part dans le mot u. Si a est un symbole, alors $u \sqcup v = \{u_1.a.u_2 | u_1.u_2 = u\}$. Par exemple, $ab \sqcup c = \{cab, acb, abc\}$.

2..4 Notions sur les langages

Définition 1.5 : Un langage est un ensemble (fini ou infini) de mots définis sur un alphabet donné.

Exemple 1.3 : des langages différents

- Langage des nombre binaires définies sur l'alphabet $\{0, 1\}$ (infini) ;
- Langage des mots de longueur 2 défini sur l'alphabet $\{a, b\}$=$\{aa, ab, ba, bb\}$ (fini) ;
- Langage C (quel est le vocabulaire ?) ;

Opérations sur les langages

Soient L, L_1 et L_2 trois langages dont l'alphabet est X, on définit les opérations suivantes :

- **Union :** notée par $+$ ou $|$ plutôt que $\cup$. $L_1 + L_2 = \{w | w \in L_1 \vee w \in L_2\}$;
- **Intersection :** $L_1 \cap L_2 = \{w | w \in L_1 \wedge w \in L_2\}$;
- **Concaténation :** $L_1.L_2 = \{w | \exists u \in L_1, \exists v \in L_2 : w = uv\}$;
- **Exposant :** $L^n = \underbrace{L.L...L}_{n \text{ fois}} = \{w | \exists u_1, u_2, ...u_n \in L : w = u_1u_2...u_n\}$;
- **Fermeture transitive de Kleene :** notée $L^* = \sum_{i \geqslant 0} L^i$. En particulier, si $L = X$ on obtient X^* c'est-à-dire l'ensemble de tous les mots possibles sur l'alphabet X. On peut ainsi définir un langage comme étant un sous-ensemble quelconque de X^*. Une autre définition possible de L^* est la suivante : $L^* = \{w | \exists n \geqslant 0, \exists u_1 \in L, \exists u_2 \in L, ..., \exists u_n \in L, w = u_1u_2...u_n\}$;
- **Fermeture non transitive :** $L^+ = \sum_{i > 0} L^i$;
- **Le langage miroir :** $L^R = \{w | \exists u \in L : w = u^R\}$;
- **Le mélange ou l'entrelacement de langages :** $L \sqcup L' = \{u \sqcup v | u \in L, v \in L'\}$. En particulier, lorsque le langage L' se réduit à un seul mot composé d'un seul symbole a, on a : $L \sqcup a = \{u.a.v | (u.v) \in L\}$

Propriétés des opérations sur les langages

Soient L, L_1, L_2, L_3 quatre langages définis sur l'alphabet A :

— $L^* = L^+ + \{\varepsilon\}$;
— $L_1.(L_2.L_3) = (L_1.L_2).L_3$;
— $L_1.(L_2 + L_3) = (L_1.L_2) + (L_1.L_3)$;
— $L.L \neq L$;
— $L_1.(L_2 \cap L_3) \subseteq (L_1 \cap L_2).(L_1 \cap L_3)$;
— $L_1.L_2 \neq L_2.L_1$.
— $(L^*)^* = L^*$;
— $L^*.L^* = L^*$;
— $L_1.(L_2.L_1)^* = (L_1.L_2)^*.L_1$;
— $(L_1 + L_2)^* = (L_1^* L_2^*)^*$;
— $L_1^* + L_2^* \neq (L_1 + L_2)^*$

3. Les grammaires (systèmes générateurs de langages)

Afin de bien comprendre l'intérét des différents points de vue de définition des langages, on considère la situation suivante : une personne A veut apprendre à une personne B la langue française. Plusieurs approches sont possibles mais une des plus simples (et des plus naïves aussi) consiste à apprendre toutes les phrases en français. Si une telle prouesse est possible, on peut dire que la personne B connaît le français (même si le contre-exemple de la pièce chinoise nous indique que cela n'est pas forcément le cas). Malheureusement, dans bien des langages (y compris le français), le nombre de phrases (ou mots) et très important à retenir, voire infini. Cette méthode n'est pas alors efficace.

Une autre manière pour faire apprendre le français serait d'expliquer à la personne B comment construire des phrases en français, c'est ce qu'on appelle couramment les règles de grammaire (qui ne sont pas propres au français bien sûr). Au lieu d'apprendre toutes les phrases, il suffit d'expliquer à la personne B qu'une phrase simple en français est composée d'un sujet, d'un verbe et d'un complément d'objet direct (il faut bien entendu expliquer ces notions aussi). L'avantage de la grammaire en français est que le nombre de règles à retenir n'est pas important mais ces règles permettent de construire un grand nombre de phrases, d'où l'intérêt des grammaires pour définir les langages.

Les grammaires possèdent un autre intérêt capital : elles permettent de classer les langages en fonction de leur complexité. Grâce à la classification de Chomsky (voir plus bas), il est possible de classer les langages dans l'une des quatre classes allant de la classe la plus simple (classe 3) à la classe la plus complexe (classe 0).

3..1 Exemple introductif formalisé

Pour analyser une classe de phrases simples en français, on peut supposer qu'une phrase est composée de la manière suivante :

— PHRASE → ARTICLE SUJET VERBE COMPLEMENT

— SUJET → "garçon" ou "fille"

— VERBE → "voit" ou "mange" ou "porte"

— COMPLEMENT → ARTICLE NOM ADJECTIF

— ARTICLE → "un" ou "le"

— NOM → "livre" ou "plat" ou "wagon"

— ADJECTIF → "bleu" ou "rouge" ou "vert"

En effectuant des substitutions (on remplace les parties gauches par les parties droites) on arrive à générer les deux phrases suivantes :

Le garçon voit un livre rouge
Une fille mange le plat vert

On dit ici que PHRASE, ARTICLE, SUJET sont des concepts du langage ou encore des symboles non-terminaux (car ils ne figurent pas dans la phrase auxquelles on s'intéresse). Les symboles "garçon", "fille", "voit", "mange", etc sont des terminaux puisqu'ils figurent dans le langage final (les phrases).

Le processus de génération de la phrase à partir de ces règles est appelé *dérivation*. Voici comment est dérivée la première phrase :

- PHRASE → ARTICLE SUJET VERBE COMPLEMENT
→ le SUJET VERBE COMPLEMENT
→ le garçon VERBE COMPLEMENT
→ le garçon voit COMPLEMENT
→ le garçon voit ARTICLE NOM ADJECTIF
→ le garçon voit un NOM ADJECTIF
→ le garçon voit un livre ADJECTIF
→ le garçon voit un livre rouge

Pour la deuxième phrase, la dérivation est la suivante :

- PHRASE → ARTICLE SUJET VERBE COMPLEMENT
→ une SUJET VERBE COMPLEMENT
→ une fille VERBE COMPLEMENT
→ une fille mange COMPLEMENT
→ une fille mange ARTICLE NOM ADJECTIF
→ une fille mange le NOM ADJECTIF
→ une fille mange le plat ADJECTIF
→ une fille mange le plat vert

3..2 Formalisation

Définition 1.6 : On appelle grammaire formelle le quadruplet (V, N, X, R) tel que :

- V est un ensemble fini de symboles dits terminaux, on l'appelle également vocabulaire terminal ;
- N est un ensemble fini (disjoint de V) de symboles que l'on appelle non-terminaux ou encore concepts ;
- S est un non-terminal particulier appelé axiome (point de départ de la dérivation) ;
- R est un ensemble de règles de production de la forme $g \to d$ tel que $g \in (V + N)^+$ et $d \in (V + N)^*$.

Les règles de la forme $\varepsilon \to \alpha$ sont interdites. Pourquoi ?

Par convention, on utilisera les lettres majuscules pour les non-terminaux, et les lettres minuscules pour représenter les terminaux. Soit une suite de dérivations : $w_1 \to w_2 \to w_3 \to ... \to w_n$, alors on écrira : $w_1 \xrightarrow{*} w_n$. On dit alors qu'il y a une séquence de dérivation qui mène de w_1 vers w_n.

Définition 1.7 : Soit une grammaire $G = (V, N, S, R)$. On dit que le mot u appartenant à V^* est dérivé (ou bien généré) à partir de G s'il existe une suite de dérivations qui, partant de l'axiome S, permettent d'obtenir u : $S \xrightarrow{*} u$. Le langage de tous les mots générés par la grammaire G est noté $L(G)$. La suite des règles appliquées pour obtenir le mot s'appelle chaîne de dérivation.

Exemple 1.4 : grammaire générant le langage $\{a\}^*$

Soit la grammaire $G = (\{a\}, \{S\}, S, \{S \to aS|\varepsilon\})$ qui génère le langage $\{\varepsilon, a, aa, aaa\ , ...\} = \{a\}^*$ selon la figure 1.1. Les boîtes en bleu représentent des mots "non-terminaux" (ce ne sont pas des mots générés par la grammaire car ils comportent au moins un symbole non-terminal), les boîtes en gris représentent des mots générés par la grammaire (car ils ne comportent aucun symbole non-terminal).
La chaîne de dérivation du mot $aaaa$ selon cette grammaire est : $S \to aS \to aaS \to aaaS \to aaaaS \to aaaa$. On peut construire l'ensemble des mots possibles selon l'arbre de la figure 1.1.

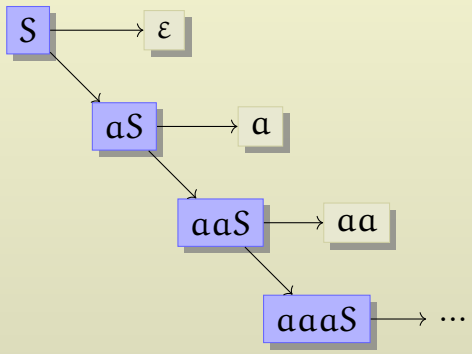

FIGURE 1.1 – Les dérivations possibles (attention, ceci n'est un pas arbre de dérivation)

Définition 1.8 : Étant donnée une grammaire $G = (V, N, S, R)$, les arbres de syntaxe de G sont des arbres où les nœuds internes sont étiquetés par des symboles de N, et les feuilles étiquetées par des symboles de V, tels que si le nœud p apparaît dans l'arbre et si la règle $p \to a_1...a_n$ (a_i terminal

ou non-terminal) est utilisée dans la dérivation, alors le nœud p possède n fils correspondant aux symboles a_i.

Si l'arbre syntaxique a comme racine l'axiome de la grammaire, alors il est dit arbre de dérivation du mot u tel que u est le mot obtenu en prenant les feuilles de l'arbre dans le sens gauche$\rightarrow$ droite et bas $\rightarrow$ haut.

Remarque : Attention : la figure 1.1 n'est un pas un arbre de dérivation !

3..3 Classification de Chomsky

Chomsky a établi une classification hiérarchique qui permet de classer les grammaires en quatre catégories. La classe d'une grammaire est déterminée grâce à la forme de ses règles de production. À chacune des catégories, on associe un type de machine minimale permettant d'analyser les langages générés par ces grammaires. Une grammaire de type i génère un langage de type j tel que $j \geqslant i$.

Soit $G = (V, N, S, R)$ une grammaire, les classes de grammaires de Chomsky sont :

— Type 3 ou grammaire régulière (à *droite*) : toutes les règles de production sont de la forme $g \rightarrow d$ où $g \in N$ et $d = aB$ tel que a appartient à V^* et B appartient à $N + \{\varepsilon\}$;

— Type 2 ou grammaire hors-contexte : toutes les règles de production sont de la forme $g \rightarrow d$ où $g \in N$ et $d \in (V + N)^*$;

— Type 1 ou grammaire contextuelle : toutes les règles sont de la forme $g \rightarrow d$ tel que $g \in (N + V)^+$, $d \in (V + N)^*$ et $|g| \leqslant |d|$. De plus, si ε apparaît à droite alors la partie gauche doit seulement contenir S (l'axiome). On peut aussi trouver la définition suivante des grammaires de type 1 : toutes les règles sont de la forme $\alpha B \beta \rightarrow \alpha \omega \beta$ tel que $\alpha, \beta \in (V + N)^*$, $B \in N$ et $\omega \in (V + N)^*$. **Dans ce cours, on utilise la première définition du type 1**.

— Type 0 : aucune restriction. Toutes les règles sont de la forme : $d \rightarrow g$, $g \in (V+N)^+$, $d \in (V+N)^*$

Le type retenu pour une grammaire est le plus petit qui satisfait les conditions. Il existe une relation d'inclusion entre les types de grammaires selon la figure 1.2.

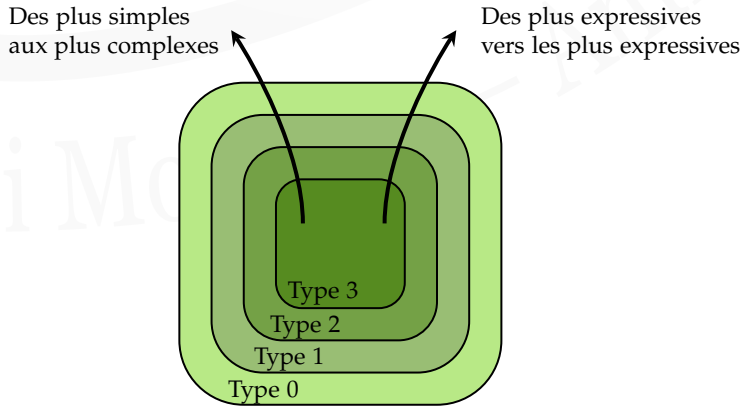

FIGURE 1.2 – La classification hiérarchique de Chomsky pour les grammaires

Exemple 1.5 : des grammaires de quelques langages

— La grammaire $(\{a,b,c\},\{S,T\},S,\{S \to abS|acT, T \to S|\varepsilon\})$ est une grammaire de type 3. On dit également qu'elle est régulière à droite. Le type du langage généré est forcément de type régulier (attention, on ne dit pas langage régulier à droite !). Un exemple d'un mot généré : $S \to abS \to abacT \to abac$ (le langage généré a la forme suivante : $\{((ab)^{p_i}ac)^q | p_i \geqslant 0, q > 0\}$.

— La grammaire $(\{a,b\},\{S,T\},S,\{S \to aSbc|\varepsilon\})$ est une grammaire de type 2 ou encore une grammaire hors-contexte. Pour trouver le type du langage généré, on doit montrer que l'on ne peut pas trouver une grammaire régulière qui génère le même le langage. C'est le cas ici (on l'accepte ici sans démonstration). Le type du langage généré est alors algébrique. Un exemple d'un mot généré est : $S \to aSbc \to aa \to bcbc \to aabcbc$ (le langage généré a la forme suivante : $\{a^n(bc)^n | n \geqslant 0\}$).

— La grammaire $(\{a,"\},\{S,B\},S,\{S \to "a", "a \to "B, Ba \to aaB, B" \to aa"\})$ est une grammaire de type 1 ou une grammaire contextuelle. Elle génère un langage contextuel car on peut montrer que le langage généré ne peut pas être généré par une grammaire hors-contexte. Un exemple d'un mot généré est : $S \to "a" \to "B" \to "aa" \to "Ba" \to "aaB" \to "aaaa"$ (le langage généré a la forme suivante : $\{"a^{2^p}" | p \geqslant 0\}$).

— La grammaire $(\{a\},\{S,T,U\},S,\{S \to UaT|a, T \to TT, aT \to Taa, UT \to U, U \to \varepsilon\})$ est une grammaire de type 0 (sans restriction). Elle génère un langage de type 0 ici. Un exemple de génération de mot est : $S \to UaT \to UaTT \to UaTTT \to UTaaTT \to UTaTaaT \to UTTaaaaT \to UTTaaaTaa \to UTTaaTaaaa \to UTTaTaaaaaa \to UTTTaaaaaaaa \to UTTaaaaaaaa \to UTaaaaaaaa \to Uaaaaaaaa \to aaaaaaaa$ ou encore a^8 (le langage généré a la forme suivante : $\{a^{2^p} | p \geqslant 0\}$).

Remarque : Pour la classe 3 des grammaires, il existe une autre forme possible appelée *grammaire régulière à gauche* (voir le TD). En réalité, tout langage régulier possède une grammaire régulière à droite et aussi une grammaire régulière à gauche (les deux formes sont alors équivalentes et l'on peut passer aisément d'une forme à l'autre).
Notez, cependant, qu'une grammaire régulière est soit à droite soit à gauche, le mélange est interdit. Une grammaire présentant à la fois les formes régulières à gauche et les formes régulières à droite est en réalité une grammaire hors-contexte.

4. Les automates

Les grammaires représentent un moyen qui permet de *décrire* un langage d'une manière que l'on peut qualifier d'inductive. Elles montrent comment les mots du langage sont dérivés.

Pour un langage donné L, on se propose de répondre à la question $w \in L$. On peut répondre à cette question de plusieurs façons. D'abord, on peut vérifier l'existence de w dans la liste des mots de L (impossible à réaliser si le langage est infini).

On peut également chercher une grammaire générant L puis vérifier si cette grammaire génère

w. On cherchera alors une chaîne de dérivation partant de l'axiome de la grammaire vers le mot. En fonction de la classe de grammaire du langage, la recherche de la chaîne peut être efficace ou non ou carrément impossible. Pour les langages de programmation usuels, c'est cette méthode qui est utilisée pour répondre à la question, les grammaires des langages sont alors minutieusement choisies pour que la recherche soit efficace.

Il existe en réalité un troisième moyen permettant de répondre è cette question : les automates. Un automate est une machine qui, après avoir exécuté un certain nombre d'opérations sur le mot, peut répondre à cette question par oui ou non.

Définition 1.9 : Un automate (ou une machine de Turing) est une machine abstraite qui permet de lire un mot et w de répondre à la question : "w appartient-il au langage ?" par oui ou non. Aucune garantie n'est cependant apportée concernant le temps d'analyse ou même la possibilité de le faire.

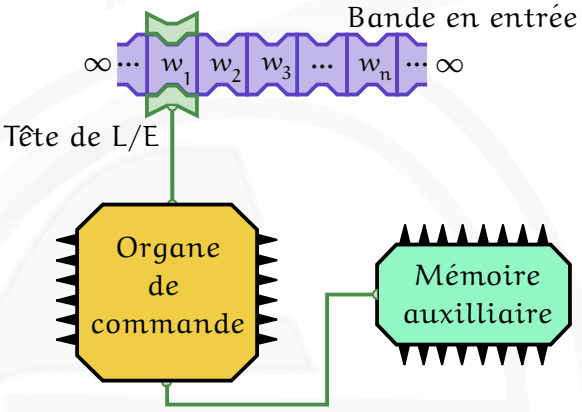

FIGURE 1.3 – Architecture d'une machine de Turing

Une machine de Turing (voir la figure 1.3) est composée de :

— Une bande ou un ruban en entrée finie ou infinie (des deux côtés) sur lequel sera inscrit le mot à lire. Le ruban est composé de cases, chacune stockant un seul symbole à la fois.

— Une tête de lecture/écriture qui peut lire ou écrire dans une seule case du ruban à la fois. Elle peut également se déplacer à droite ou à gauche sur le ruban.

— Un organe de commande qui permet de gérer un ensemble fini de pas d'exécution.

— Éventuellement, une mémoire auxiliaire de stockage.

Formellement, un automate contient au minimum :

— Un alphabet pour les mots en entrée noté X ;

— Un ensemble non vide d'états noté Q ;

— Un état initial noté $q_0 \in Q$;

— Un ensemble non vide d'états finaux $q_f \in Q$;

— Une fonction de transition (permettant de changer d'état) notée δ. Le changement d'état se fait en fonction de l'état en cours de l'automate, du symbole en cours de lecture par la tête et éventuellement du contenu de la mémoire auxiliaire.

4..1 Configuration d'un automate

L'analyse d'un mot par un automate se fait à travers une séquence de configurations.

Définition 1.10 : On appelle configuration d'un automate les valeurs de ses différents composants, à savoir la position de la tête L/E, l'état de l'automate et éventuellement le contenu de la mémoire auxiliaire (lorsqu'elle existe). On peut représenter sommairement la configuration par un triplet (q, a, mem) avec q : l'état de l'automate, a : la position de la tête L/E et mem : le contenu d'une éventuelle mémoire auxiliaire.

Il existe deux configurations spéciales appelées configuration initiale et configuration finale.

Définition 1.11 : La configuration initiale est celle qui correspond à l'état initial q_0 et où la tête de L/E est positionnée sur le premier symbole du mot à lire.

Définition 1.12 : Une configuration finale est celle qui correspond à un des états finaux q_f et où le mot a été entièrement lu.

On dit qu'un mot est **accepté** par un automate si, à partir d'une **configuration initiale**, on arrive à une **configuration finale** à travers une succession de configurations intermédiaires. En d'autres termes, un mot est accepté par une machine de Turing en suivant les étapes suivantes :

1. Inscrire le mot entier sur le ruban (le mot est de longueur finie) ;
2. Positionner la tête de L/E sur le premier symbole du mot ;
3. Mettre la machine à son état intial ;
4. Laisser tourner la fonction de transition jusqu'à arrêt de la machine ;
5. Le mot est accepté si (les deux conditions sont nécessaires) :
   (a) Le mot est entièrement lu (au moins une fois) ;
   (b) La machine se trouve dans un état final.

On dit aussi qu'un langage est accepté par un automate lorsque tous les mots de ce langage (et uniquement les mots de ce langage) sont acceptés par l'automate.

4..2 Classification des automates

Comme les grammaires, les automates peuvent être classés en 4 classes selon la hiérarchie de Chomsky :

— Type 3 ou automate à états finis (AEF) : il accepte les langages de type 3. Sa structure est la suivante :
   — bande en entrée finie ;
   — sens de lecture de gauche à droite ;
   — Pas d'écriture sur la bande et pas de mémoire auxiliaire.

   Un AEF est complètement dépourvue de mémoire.

— Type 2 ou automate à pile : il accepte les langages de type 2. Sa structure est similaire à l'AEF mais dispose en plus d'une mémoire organisée sous forme d'une **seule** pile infinie ;

— Type 1 ou automate à bornes linéaires (ABL) : il accepte les langages de type 1. Sa structure est la suivante :
  — Bande en entrée **finie** accessible en lecture/écriture ;
  — Lecture dans les deux sens ;
  — Pas de mémoire auxiliaire.

— Type 0 ou machine de Turing : il accepte les langages de type 0. Sa structure est la même que l'ABL mais la bande en entrée est infinie.

Le tableau 1.1 résume les différentes classes de grammaires, les langages générés et les types d'automates qui les acceptent :

Grammaire	**Langage**	**Automate**
Type 0	Récursivement énumérable	Machine de Turing
Type 1 ou contextuelle	Contextuel	Automate à borne linéaire
Type 2 ou hors-contexte	Algébrique	Automate à pile
Type 3 ou régulière	Régulier ou rationnel	Automate à états finis

TABLE 1.1 – Les types de grammaires, langages et automates (voir plus bas pour les langages récursivement énumérables)

5. De la décidabilité et de la complexité

5..1 Introduction à la décidabilité

Tous les langages ne se ressemblent pas. Par cela, on entend que le comportement des machines de Turing n'est pas le même pour tous les langages. Certains sont dits plus *complexes* que d'autres car la construction des machines de Turing correspondantes est plus laborieuse ou le temps d'analyse peut être très important. Certains langages ne possèdent même pas de machines de Turing qui les acceptent, on introduit alors la définition d'un langage "récursivement énumérable".

Définition 1.13 : Un langage L défini sur un alphabet X est dit *récursivement énumérable* si l'on peut trouver une machine de Turing qui peut décider de l'appartenance (c'est-à-dire lire tout le mot et s'arrêter avec un état final) de tout mot de L moyennant un nombre fini de configurations (la machine donne la réponse oui).

Les langages qui ne sont pas récursivement énumérables existent en réalité. Ce sont des langages dont l'analyse de certains mots ne peut jamais aboutir (la machine tourne à l'infini). Un exemple

simple de ces langages est l'ensemble des programmes en langages C qui se terminent. En effet, on peut démontrer qu'il est impossible de construire une machine de Turing qui peut décider, en un temps fini, si un programme quelconque écrit en langage C peut se terminer pour une entrée donnée (problème appelé *problème d'arrêt*).

Un langage récursivement énumérable ne garantit donc que la procédure d'analyse d'un mot qui ne lui appartient pas puisse se terminer. Ceci nous amène alors à une nouvelle définition.

Définition 1.14 : Un langage L défini sur un alphabet X est dit *récursif* si l'on peut trouver une machine de Turing qui peut décider de l'appartenance ou non d'un mot de X^* moyennant un nombre fini de configurations.

Pour un langage récursif, on dit que la procédure de son analyse est *décidable*. L'analyse d'un langage récursif se termine alors toujours quel que soit son résultat final. Étant donné que l'objectif final de la construction des automates est la construction d'un programme équivalent qui analyse des mots, il est tout à fait évident de s'intéresser uniquement à la catégorie des langages récursifs ici.

On doit noter ici que la définition utilise l'expression "un nombre fini de configurations". Cette définition est plus intéressante par rapport à une autre qui se base sur le temps d'exécution ou tout autre critère. Ceci permet en effet d'annoncer les résultats suivants :

— Le temps d'analyse des mots d'un langage récursif est fini.

— La taille de la mémoire utilisée pour analyser un mot d'un langage récursif est finie.

— La taille du ruban utilisé pour analyser un mot d'un langage récursif est finie.

5..2 Notions sur la complexité

Lorsqu'un langage est récursif, cela ne signifie pas forcément qu'il soit intéressant. En effet, si l'on s'intéresse au temps par exemple, l'analyse d'un mot, bien qu'elle dure un temps fini, peut prendre un temps considérable pouvant aller à des milliards de siècles sur le plus rapide des ordinateurs. De même, l'analyse d'un mot peut consommer un espace mémoire (ruban+mémoire auxiliaire) qui, bien que fini, peut aller à des bornes défiant toute technologie existante.

On voudrait dire par cette introduction que l'analyse d'un mot par une machine requiert généralement des ressources (temps, espace, énergie, bande passante d'un réseau, etc). Les langages récursifs utilisent un nombre fini de ressources, mais cela ne signifie pas qu'ils consomment peu de ressources.

Lors de l'analyse d'un mot d'une certaine taille, on appelle **complexité** le nombre de ressources mobilisées par l'analyse. Évidemment, vu la limitation des ressources (qui coûtent de l'argent en fin de compte), il est tout à fait justifié de penser à construire les machines qui en consomment le moins.

Traditionnellement, on définit la complexité comme étant la relation qui existe entre la taille du mot à analyser (n) et le nombre de ressources requises. Il est également d'usage de s'intéresser à deux types de ressources plus que d'autres :

— Temps : c'est-à-dire le temps nécessaire pour analyser le mot. On appelle cela la complexité en temps.

— Espace : c'est-à-dire la taille du ruban et la mémoire auxiliaire nécessaires pour analyser le mot. On appelle cela la complexité en espace.

On peut se baser sur un critère qui permet de représenter plusieurs types de complexité. En effet, si l'on s'intéresse au nombre de configurations nécessaires pour analyser un mot (en acceptation ou

en rejet), il est possible de quantifier :

— La complexité en temps : le passage d'une configuration à une autre se fait généralement en un temps fixe a. Si le nombre de configurations est $f(n)$ alors le temps d'analyse sera $a.(f(n)-1)$.

— La complexité en espace : en considérant la suite des configurations, il est possible, non sans difficulté, de quantifier le nombre de cases utilisées sur le ruban ainsi que la taille de la mémoire utilisée. Dans tous les cas, ce nombre est fonction de $f(n)$.

Dans ce qui suit, on s'intéressera alors au nombre de configurations possibles pour un mot donné comme étant le critère de complexité. Cette dernière peut aller de la plus simple des quantités (une seule configuration : $f(n) = 1$) à une fonction exponentielle ($f(n) = a^n$) ce qui rend la machine tout à fait inexploitable.

Il est également à noter que vu la possibilité de construire plusieurs machines de Turing acceptant le même langage, on peut alors associer plusieurs complexités à un langage donné. Parmi toutes les machines existantes, on dira que la machine la moins complexe (celle avec le plus faible $f(n)$ en moyenne) représente la **complexité du langage**.

Une nouvelle fois, la classification de Chomsky nous aide à définir la complexité de certains types de langages. Par exemple, tous les langages réguliers ont une complexité dite linéaire, c'est-à-dire que $f(n)$ est donné par $a \times n$. Ce n'est pas le cas pour les autres classes. En effet, seule une sous-partie des langages algébriques possède une complexité linéaire. Par la suite, on appellera cette classe : les langages algébriques linéaires, c'est d'ailleurs la classe à laquelle appartiennent les langages de programmation usuels. Enfin, en général, la catégorisation des langages contextuels et des langages récursifs par rapport à la complexité est beaucoup plus difficile.