Part-I : Langages

Anne Fouilloux

<mi_isty1@libertysurf.fr>

Plan

Position dans le cours

1 – Introduction

1.1 – Alphabets, mots et langages

1.1.1 – Le type alphabet

Un **alphabet**, noté Σ, est un ensemble non vide de symboles.

1.1.2 – le type mot

Un **mot**, défini sur un alphabet Σ, est une suite finie d'éléments de Σ.

1.1.3 – Définitions

- $|x|$: longueur du mot x, i.e., $|x|$ = nombre de lettres qui composent x ;
- ϵ : le mot vide de longueur $|\epsilon| = 0$;
- $x.y$ (ou xy) : la concaténation des deux mots x et y, i.e., le mot formé en faisant suivre les lettres de x par les lettres de y ;
- x^n : le mot x concaténé n fois ($x^0 = \epsilon$, $x^1 = x$, $x^2 = xx$, $x^3 = xxx$, etc.) ;

- Un mot u est dit **facteur** d'un mot v s'il existe deux mots s et t tels que l'on ait $v = sut$. On parle de facteur gauche, ou **préfixe**, lorsque s est vide et de facteur droit, ou **suffixe**, lorsque t est vide.
- Σ^+ : l'ensemble des mots de longueur supérieure ou égale à 1 que l'on peut construire à partir de l'alphabet Σ ;
- Σ^* : l'ensemble des mots que l'on peut construire à partir de Σ, y compris le mot vide :

$$\Sigma^* = \{\epsilon\} \cup \Sigma$$

1.1.4 – Définition (dépendance multiplicative)

Deux mots u et v sont dits **multiplicativement dépendants** s'ils sont puissances d'un même troisème, c'est à dire s'il existe un mot w et deux entiers m et n tels que $u = w^m$ et $v = w^n$.

1.1.5 – Proposition

Deux mots u et v commutent si et seulement s'ils sont multiplicativement dépendants.

1.1.6 – Le type langage

Un **langage**, défini sur un alphabet Σ, est un ensemble de mots définis sur Σ.

Remarque : Deux langages particuliers sont indépendants de l'alphabet Σ :

- Le langage vide ($L = \emptyset$) ;
- Le langage contenant le seul mot vide ($L = \{\epsilon\}$).

1.1.7 – Propriétés

Soient deux langages L_1 et L_2 respectivement définis sur les alphabets Σ_1 et Σ_2 :

- L'union de L_1 et L_2 est un langage défini sur $\Sigma_1 \cup \Sigma_2$ contenant tous les mots qui sont soit dans L_1, soit dans L_2 :
$$L_1 \cup L_2 = \{x/x \in L_1 \text{ ou } x \in L_2\}$$
- L'intersection de L_1 et L_2 est le langage défini sur $\Sigma_1 \cap \Sigma_2$ contenant tous les mots qui sont à la fois dans L_1 et dans L_2 :
$$L_1 \cap L_2 = \{x/x \in L_1 \text{ et } x \in L_2\}$$
- Le complément de L_1 est le langage défini sur Σ_1 contenant tous les mots qui ne sont pas dans L_1 :
$$\bar{L_1} = \{x/x \notin L_1\}$$
- La différence de L_1 et L_2 est le langage défini sur Σ_1 contenant tous les mots de L_1 qui ne sont pas dans L_2 :
$$L_1 - L_2 = \{x/x \in L_1 \text{ et } x \notin L_2\}$$

☞ Le produit ou concaténation de L_1 et L_2 est le langage défini sur $\Sigma_1 \cup \Sigma_2$ contenant tous les mots formés d'un mot de L_1 suivi d'un mot de L_2 :

$$L_1 L_2 = \{xy / x \in L_1 \text{ et } y \in L_2\}$$

☞ La **fermeture itérative** de L_1 (ou **fermeture de Kleene ou itéré** de L_1) est l'ensemble des mots formés par une concaténation finie de mot de L_1 :

$$L_1^* = \{x / \exists k \geq 0 \text{ et } x_1, \ldots, x_k \in L_1 \text{ tels que } x = x_1 x_2 \ldots x_k\}$$

$$L_1^* = \epsilon \cup L_1 \cup L_1^2 \cup L_1^3 \cup \ldots \cup L_1^n \cup \ldots$$

1.1.8 – Proposition

Pour tous langages K et L sur un alphabet Σ, on a $(K + L)^* = (K^* L)^* K^*$

1.1.9 – Lemme d'Arden (équations aux langages)

Soient A et B deux langages. L'équation $X = AX + B$ (respectivement $X = XA + B$) a pour plus petite solution A^*B (respectivement BA^*), et cette solution est unique lorsque ϵ n'est pas élément de A.

1.2 – Langages réguliers

1.2.1 – Définition (langages)

L'ensemble R des **langages réguliers** sur un alphabet Σ est le plus petit ensemble de langages satisfaisant les conditions suivantes :

1. $\emptyset \in R; \{\epsilon\} \in R$;
2. $\{a\} \in R$ pour tout $a \in \Sigma$;
3. si $A, B \in R$, alors $A \cup B, A \cdot B$ et $A^* \in R$.

1.2.2 – Définition (Expressions)

Les **expressions régulières** pour un alphabet Σ sont les expressions formées par les règles suivantes :

1. $\emptyset, \epsilon$ et les éléments de Σ sont des expressions régulières,
2. si α et β sont des expressions régulières, alors $(\alpha \cup \beta), (\alpha\beta), (\alpha)^*)$ sont des expressions régulières.

Remarque : Dans la pratique, les parenthèses seront omises (sauf ambiguités).

1.2.3 – Définition

Le langage $L(\xi)$ représenté par une expression régulière ξ est défini comme suit :

1. $L(\emptyset) = \emptyset, L(\epsilon) = \{\epsilon\}$,
2. $L(a) = \{a\}$ pour tout $a \in \Sigma$,
3. $L((\alpha \cup \beta)) = L(\alpha) \cup L(\beta)$,
4. $L((\alpha\beta)) = L(alpha) \cdot L(\beta)$,
5. $L((\alpha)^*) = L(\alpha)^*$.

1.2.4 – Théorème

Un langage est régulier si et seulement si il est dénoté par une expression régulière.

1.3 – Langages non réguliers

Nous allons démontrer qu'il existe des langages non réguliers en comparant le nombre de langages et d'expressions régulières. Le problème est qu'il y a un **nombre infini** d'expressions régulières et de langages.

1.3.1 – Définition (Cardinalité)

Si deux ensembles peuvent être mis en bijection, ils ont la même **cardinalité** i.e., le même nombre déléments.

1.3.2 – Définition (Ordinalité)

L'ordinalité n'est définie que pour des ensembles ordonnés (sur lesquels a été défini une relation d'ordre). Deux ensembles ont alors la même **ordinalité** s'il existe entre eux une bijection qui préserve ces relations d'ordre.

1.3.3 – Définition (dénombrable)

Un ensemble infini est **dénombrable** (**énumérable**) s'il existe une bijection entre cet ensemble et l'ensemble des nombres naturels.

Exemples :

❶ L'ensemble des nombres pairs est dénombrable. La bijection avec les entiers naturel est : $\{(0,0),(2,1),(4,2),(6,3),...\}$.

Une généralisation de cette observation est que tout sous-ensemble infini des entiers naturels est dénombrable.

❷ L'ensemble des mots sur l'alphabet $\{a,b\}$ est dénombrable. Pour obtenir une énumération (bijection avec les naturel) des mots sur l'alphabet $\{a,b\}$ on classe les mots par ordre de taille croissante et, pour les mots d'une même taille, par ordre lexicographique (l'ordre du dictionnaire) :

$\{(\epsilon,0),(a,1),(b,2),(aa,3),(ab,4),(ba,5),(bb,6),(aaa,7)...\}$

❸ Les expressions régulières sont dénombrables. L'ensemble de toutes les chaînes de caractères sur un alphabet est dénombrables (exemple-2). De plus, les expressions régulières sont un sous-ensemble infini de ces chaînes de caractères et sont donc aussi dénombrables (exemple-1).

1.3.4 – Théorème

L'ensemble des sous-ensembles d'un ensemble dénombrable n'est pas dénombrable.

Ce thèorème implique aue tous les langages ne peuvent être réguliers :

- L'ensemble des langages est l'ensemble des sous-ensembles d'un ensemble dénombrable (l'ensemble des mots) et n'est donc pas dénombrable.
- L'ensemble des langages réguliers est dénombrable puisque chaque langage régulier est dénoté par un expression régulière et que l'ensemble des expressions régulières est dénombrable ;
- Il y a donc beaucoup plus de langages que de langages réguliers.

Position dans le cours

2 – Automates finis déterministes

2.1 – Notion d'automates

2.1.1 – Objectif

Donner une méthode algorithmique permettant de décider si un mot w quelconque est, ou n'est pas, dans L.

2.1.2 – Méthode

Lire le mot testé w de gauche à droite en mettant en mémoire au fur et à mesure les éléments souhaités, et à conclure à la fin de la lecture en fonction des éléments présents dans la mémoire.

2.1.3 – Hypothèse

On suppose que la mémoire est :

- finie ;
- de taille fixe, indépendante du mot w considéré.

2.1.4 – Définition

Un **automate**, ou **automate fini déterministe**, ou **AFD**, est un quintuplet (Q, Σ, T, q_0, A), où Q et Σ sont deux ensembles finis, appelés l'**ensemble des états** et l'**alphabet** de l'automate, T est une application de $Q \times \Sigma$ dans Q, appelée la **fonction de transition** de l'automate, q_0 est un élément de Q, appelé l'**état initial**, et A est un sous-ensemble de Q, appelé l'**ensemble des états acceptants**.

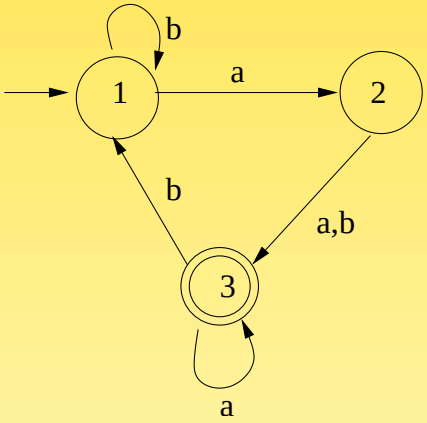
2.1.5 – Représentation d'un automate

- ☞ Table des valeurs de T, comme un tableau à double entrée (états pour les lignes et caractères pour les colonnes)
- ☞ Graphe dont les sommets représentent les états et où les flèches sont étiquetées par les caractères.

Dans les deux cas, l'état inital est indiqué par une flèche "entrante", et les états acceptants en entourant les états concernés.

2.1.6 – Exemple

	a	b
$\rightarrow 1$	2	1
2	3	3
$\boxed{3}$	3	1



2.1.7 – Définition (mot accepté, langage décidé)

Supposons que **M** est un automate, soit $M = (Q, \Sigma, T, q_0, A)$. La **fonction de transition itérée** est l'application $T^* : Q \times \Sigma^* \to Q$ définie par :

$$T^*(q, w) = \begin{cases} q \text{ pour } w = \epsilon \\ T(T^*(q, w_0), x) \text{ pour } w = w_0 x \text{ avec } x \in \Sigma \end{cases}$$

- ☞ Le mot w est **accepté** par l'automate M si et seulement si l'état $T^*(q_0, w)$ est dans A ;
- ☞ w est **refusé** dans le cas contraire.

Si M est un automate d'alphabet Σ, et L un langage sur Σ, on dit que M **décide** L si L est l'ensemble des mots acceptés par M.

2.1.8 – Lemme

Supposons que (Q, Σ, T, q_0, A) est un automate. Pour tout état q et tous mots u, v dans Σ^*, on a l'égalité $T^*(q, uv) = T^*(T^*(q, u), v)$.

Position dans le cours

3 – Langages rationnels

3.1 – Définitions (rationnel)

Supposons que L est un langage. On dit que L est **rationnel** ou **automatique** ou **décidable par automate fini déterministe** s'il existe au moins un automate qui décide L.

3.2 – Proposition (décidabilité)

Tout langage rationnel est décidable.

3.3 – Construction d'automates

3.3.1 – Proposition (singleton)

Tout langage composé d'un mot unique est rationnel.

3.3.2 – Proposition (complément)

Supposons que l'automate (Q, Σ, T, q_0, A) décide le langage L. Alors l'automate $(Q, \Sigma, q_0, Q \backslash A)$ décide le langage complémentaire $\Sigma^* \backslash L$.

3.3.3 – Définition (produit cartésien)

Supposons que T_1 et T_2 sont des fonctions respectivement de $Q_1 \times \Sigma$ dans Q_1 et de $Q_2 \times \Sigma$ dans Q_2. On note $T_1 \times T_2$ la fonction de $(Q_1 \times Q_2) \times \Sigma$ dans $Q_1 \times Q_2$ définie par $T_1 \times T_2((q_1, q_2), x) = (T_1(q_1, x), T_2(q_2, x))$.

3.3.4 – Lemme

Supposons que T_1 et T_2 sont des fonctions respectivement de $Q_1 \times \Sigma$ dans Q_1 et de $Q_2 \times \Sigma$ dans Q_2. Alors, pour tout couple (q_1, q_2) dans $(Q_1 \times Q_2)$ et pour tout mot w dans Σ^*, on :

$$(T_1 \times T_2)^*((q_1, q_2), w) = (T_1^*(q_1, w), T_2^*(q_2, w))$$

3.3.5 – Proposition (intersection)

Supposons que, pour $i = 1, 2$, l'automate $(Q_i, \Sigma, T_i, q_{0,i}, A_i)$ décide le langage L_i. Alors, l'automate $(Q_1 \times Q_2, \Sigma, T_1 \times T_2, (q_1, q_2), A_1 \times A_2)$ décide le langage $L_1 \cap L_2$.

3.3.6 – Proposition (union)

Supposons que, pour $i = 1, 2$, l'automate $(Q_i, \Sigma, T_i, q_{0,i}, A_i)$ décide le langage L_i. Alors l'automate $(Q_1 \times Q_2, \Sigma, T_1 \times T_2, (q_1, q_2), A_1 \times Q_2 \cup Q_1 \times A_2)$ décide le langage $L_1 \cup L_2$.

3.3.7 – Proposition (différence)

Supposons que, pour $i = 1, 2$, l'automate $(Q_i, \Sigma, T_i, q_{0,i}, A_i)$ décide le langage L_i.

Alors l'automate $(Q_1 \times Q_2, \Sigma, T_1 \times T_2, (q_1, q_2), A_1 \times Q_2 \backslash Q_1 \times A_2)$ décide le langage $L1 \backslash L_2$.

3.3.8 – Proposition (fini)

Tout langage fini est rationnel.

3.3.9 – Proposition (clôture)

La classe des langages rationnels est close par union, intersection, complémentaire, produit, étoile.

3.4 – Définition (langage régulier)

Soit L un langage d'alphabet Σ. On dit que L est régulier si L peut se construire à partir l'ensemble vide et des singletons $\{x\}$ pour x dans Σ en appliquant un **nombre fini** de fois les opérations union, produit et étoile.

3.5 – Proposition (rationnel)

Tout langage régulier est rationnel.

3.6 – Proposition (Kleene)

Tout langage rationnel est régulier - et, donc, un langage est rationnel si et seulement si il est régulier.

3.7 – Lemme de l'étoile

3.7.1 – Lemme (de l'étoile ou de la pompe)

Si un langage L est reconnu par un automate fini, alors il existe un entier n tel que pour tout mot u de L de longueur au moins n il existe des mots x, v et y tels que

$u = xvy$ et que :

- ☞ $v \neq \epsilon$;
- ☞ xv est de longueur au plus n ;
- ☞ $\forall i \in N$, $xv^i y$ est élément de L.

3.7.2 – Théorème

Si L est reconnaissable par un automate fini $\exists n$ tel que $\forall u = svt \in L$ avec $|v| \geq n$, $\exists x, y, z$ tels que $u = sxyzt$ et $1 \leq |y| \leq |n|$, vérifiant $\forall i \in N, sxy^i zt \in L$.

Position dans le cours

4 – Automates finis non-déterministes

4.1 – Introduction

On introduit la notion de non déterminisme en admettant :

- ☞ présence de plusieurs états initiaux
- ☞ pour chaque état et chaque caractère, choix de passer dans plusieurs états

Techniquement, Q étant l'ensemble des états et Σ l'alphabet considéré, il suffit de remplacer une fonction de transition allant de $Q \times \Sigma$ dans Q par une fonction de transition allant de $Q \times \Sigma$ dans l'ensemble des parties $B(Q)$: $T(q, x)$ représente maintenant l'ensemble des états où on peut aller en partant de q et en lisant x, le cas d'un automate déterministe correspondant toujours au cas où $T(q, x)$ est un singleton.

4.2 – Description formelle

4.2.1 – Définition (AFN)

Un **automate fini non déterministe**, ou **AFN**, d'ensemble d'états Q et d'alphabet Σ, est un quintuplet (Q, Σ, T, I, A) où Q et Σ sont deux ensembles finis, appelés l'**ensemble des états** et l'**alphabet** de l'AFN, T est une application de $Q \times \Sigma$ dans $B(Q)$, appelée la **fonction de transition** de l'AFN, et I et A sont deux sous-ensembles de Q, appelés respectivement l'**ensemble des états initiaux** et l'**ensemble des états acceptants** de l'AFN.

4.2.2 – Définition (Lecture, Acceptation)

Supposons que M est un AFN, soit $M = (Q, \Sigma, T, I, A)$, et w est un mot sur Σ, soit $w = x_1 \dots x_l$. Une **lecture** de w par M est une suite d'états $(q_0, q_1, \dots, q_l)$ vérifiant :

- ☞ $q_0 \in I$
- ☞ $q_i \in T(q_{i-1}, x_i)$ pour $1 \leq i \leq l$

On dit que le mot w est **accepté** par M s'il existe au moins une lecture de w par M qui se termine par un état acceptant.

4.2.3 – Définition (Transition itérée)

Supposons que (Q, Σ, T, I, A) est un AFN. La **fonction de transition itérée** de cet AFN est la fonction T^* de $B(Q) \times \Sigma$ dans $B(Q)$ définie par :

$$T^*(X, w) = \begin{cases} X \text{ pour } w = \epsilon \\ \bigcup_{q \in T^*(X, w_0)} T(q, x) \text{ pour } w = w_0 x \text{ avec } x \in \Sigma \end{cases}$$

4.2.4 – Lemme

Supposons que (Q, Σ, T, I, A) est un AFN, et que w est un mot sur Σ. Alors $T^*(I, w)$ est l'ensemble des états finaux des lectures de w par (Q, Σ, T, I, A).

4.2.5 – Définition (équivalence)

Supposons que M et M' sont deux automates, AFN, ou plus généralement, deux systèmes quelconques permettant d'accepter et de refuser des mots sur un même alphabet. On dit que M et M' sont **équivalents** s'ils acceptent, et donc refusent, exactement les mêmes mots.

4.3 – Elimination du non-déterminisme

4.3.1 – Théorème

Pour tout automate fini non déterministe, il est possible de construire un automate fini déterministe équivalent.

4.3.2 – Principe de la construction

Pour passer d'un AFN à un AFD, il faut que l'AFD prenne en compte toutes les transitions possibles de l'automate non déterministe.

La solution est de construire un automate déterministe qui, à chaque étape, mémorise tous les états dans lesquels l'automate non déterministe pourrait se trouver. Cela implique que chaque état de l'automate déterministe correspondra à un ensemble d'états de l'automate non déterministe.

4.3.3 – Proposition (déterminisation)

Supposons que M est un QFN, soit $M = (Q, \Sigma, T, I, A)$. Alors M est équivalent à l'automate $\tilde{M}$, défini par $\tilde{M} = (P(Q), \Sigma, \tilde{T}, I, \tilde{A})$, avec

$$\tilde{T}(X, x) = \bigcup_{q \in X} T(q, x), \tilde{A} = \{X \subseteq Q; X \cap A \neq \emptyset\}$$

4.3.4 – Algorithme formel

But : Partant d'un AFN (Q, Σ, T, I, A) construire une AFD équivalent.

Méthode : Construire une suite croissante de fonctions de transitions $\tilde{T}_k$ sur des ensembles d'états $\tilde{Q}_k$ inclus dans $B(Q)$ en partant $\tilde{T}_0 = \emptyset$, $\tilde{Q}_0 = I$, puis en posant :

$$\tilde{T}_{k+1} = \tilde{T}_k \cup \{(X, x, T(X, x)); X \in \tilde{Q}_k, x \in \Sigma\},$$

$$\tilde{Q}_{k+1} = \tilde{Q}_k \cup \bigcup_{X \in \tilde{Q}_k, x \in \Sigma} \{T(X, x)\},$$

où $T(X, x)$ désigne $\cup_{q \in X} T(q, x)$. Arrêter au plus petit indice l tel que $\tilde{Q}_{l+1}$ est égal à $\tilde{Q}_l$. L'automate cherché est $(\tilde{Q}_l, \Sigma, \tilde{T}_{l+1}, I, \{X \in \tilde{Q}_l; X \cap A \neq \emptyset\})$.

4.4 – Epsilon-transitions

4.4.1 – Ajout d'epsilon-transition

Avec les AFN, on peut associer à la lecture d'un même caractère plusieurs transitions, i.e., plusieurs changements d'états, mais chaque transition s'accompagne toujours de la lecture d'un unique caractère.

Il peut être pratique de changer d'état sans lire de nouveau caractère. Un transition où on change d'état sans lire de caractère est traditionnellement appelée une $\epsilon-$transition car tout ce passe comme si on avait lu le mot vide ϵ à la place d'une lettre.

4.4.2 – Définition (AFNϵ)

Pour tout alphabet Σ, on note Σ_+ pour $\Sigma \cup \{\square\}$ où $\square$ est une lettre fixée supposée non dans Σ, et $\pi_\square$ l'homomorphisme de Σ_+^* sur Σ^* qui envoie chaque lettre de Σ sur elle-même, et $\square$ sur le mot vide. Un AFN avec $\epsilon-$**transition**, ou AFNϵ, d'alphabet Σ est un quintuplet (Q, Σ, T, I, A) où T est une application de $Q \times \Sigma_+$ dans $B(Q)$, et où I et A sont des sous-ensembles de Q.

Remarque : Au changement d'alphabet près, un AFNϵ d'alphabet Σ est un AFN d'alphabet Σ_+.

4.4.3 – Définition (mot accepté)

Supposons que M est un AFNϵ d'alphabet Σ, soit $M = (Q, \Sigma, T, I, A)$, et que w est un mot sur Σ. On dit que w est **accepté** par M s'il existe au moins un mot w_+ sur Σ^* qui est accepté par l'AFN (Q, Σ_+, T, I, A) et qui est tel que w est une image de w_+ par la contraction $\pi_{\square}$.

Remarque : par construction, le langage décidé par un AFNϵ M d'alphabet Σ est l'image par l'homomorphisme $\pi_{\square}$ du langage décidé par l'AFN M. Noter que, pour w mot sur Σ, les mots dans $\pi_{\square}^{-1}(w)$ sont les mots obtenus en "rembourrant" w avec des lettres $\square$ en nombre quelconque.

4.4.4 – Définition (lecture)

Supposons que M est un AFNϵ d'alphabet Σ, soit $M = (Q, \Sigma, T, I, A)$, et que w est un mot sur Σ, soit $w = x_1 \dots x_n$. Une **lecture** de w par M est une suite finie d'états $(q_0, q_1, \dots, q_N)$ telle qu'il existe une injection croissante f de $\{1, \dots, n\}$ dans $\{1, \dots, N\}$ telle que :

- q_0 dans I ;
- pour tout i, ou bien il existe j vérifiant $i = f(j)$, et $q_i \in T(q_{i-1}, x_j)$, ou bien i n'est pas dans l'image de f, et on a $q_i \in T(q_{i-1}, \square)$.

4.4.5 – Lemme

Supposons que M est une AFNϵ d'alphabet Σ, et que w est un mot sur Σ. Alors w est accepté par M si et seulement si il existe au moins une lecture de w par M qui finit par un état acceptant.

4.4.6 – Définition (clôture)

Supposons que (Q, Σ, T, I, A) est un AFNϵ. Pour chaque état q, la **clôture** de q par $\epsilon-$transition, ou $\epsilon-$clôture de q, est l'ensemble d'états $\mathrm{cl}_\epsilon(q)$ défini par :

$$\mathrm{cl}_\epsilon(q) = \bigcup_{k \geq 0} \tilde{T}^*(\{q\}, \square^k),$$

où $\tilde{T}$ est l'extension de T aux sous-ensembles de Q.

4.4.7 – Algorithme (clôture)

Les éléments de $\mathrm{cl}_\epsilon(q)$ sont les états auxquels on peut parvenir à partir de q par des $\epsilon-$transitions. La clôture se détermine par un algorithme simple.

But : Partant d'un AFNϵ (Q, Σ, T, I, A), déterminer la *epsilon*$-$clôture d'un état q.

Méthode : Construire une suite d'ensembles d'états $Q_0, Q_1, \ldots$ en partant de $Q_0 = \{q\}$ et en posant $Q_{k+1} = Q_k \cup \bigcup_{q \in Q_k} T(q, \square)$. La clôture est le premier ensemble Q_l vérifiant $Q_{l+1} = Q_l$.

4.4.8 – Proposition (suppression $\epsilon-$transition)

Supposons que M est un AFNϵ, soit $M = (Q, \Sigma, T, I, A)$. Alors M est équivalent à l'AFN M' défini par $M' = (Q, \Sigma, T', I, A')$ avec :

$$T'(q, x) = \bigcup_{q' \in cl_\epsilon(q)} T(q', x),\ A' = \{q; cl_\epsilon(q) \cap A \neq \emptyset\}.$$

4.4.9 – Proposition (produit)

Supposons que les langages L_1 et L_2 sont décidés respectivement par des automates (ou des AFN, ou des AFNϵ) M_1 et M_2, et que les ensembles d'états de M_1 et M_2 sont disjoints. Alors le langage $L_1.L_2$ est décidé par l'AFNϵ obtenu en ajoutant à la réunion de M_1 et M_2 des $\epsilon-$transitions allant de chaque état acceptant de M_1 vers chaque état initial de M_2, et en ne gardant comme états initiaux que ceux de M_1, et comme états acceptants que ceux de M_2.

4.4.10 – Proposition (étoile)

Supposons que le langage L est décidé par un automate M d'état initial q_0. Alors le langage L^* est décidé par l'AFNϵ obtenu en ajoutant à M un nouvel état initial acceptant q_0', et des $\epsilon-$transitions de chaque état acceptant de M vers q_0.

Position dans le cours

5 – Automate minimal d'un langage

5.1 – Introduction

Dans le cas d'un langage L rationnel :

- ☞ il n'y a jamais unicité d'un automate décidant L i.e., partant d'un automate M qui décide L, on peut construire un nouvel automate M' différent de M décidant encore L en ajoutant à M de nouveaux états où n'arrive aucune flèche venant des états de M.
- ☞ il y a unicité à isomorphisme près quand on considère les automates de taille minimale parmi les divers automates qui décident L.
- ☞ l'unique automate minimal associé à L peut être construit de façon effective à partir de tout automate décidant L.

5.2 – Minimalisation d'un automate

5.2.1 – Définition (isomorphes)

Deux automates (Q, Σ, T, q_0, A) et $(Q', \Sigma, T', q'_0, A')$ sont dits isomorphes s'il existe une bijection f de Q sur Q' telle que $T'(f(q), x)$ est égal à $f(T(q, x))$ pour tous q et x, q'_0 est $f(q_0)$, et $f(q) \in A'$ est équivalent à $q \in A$.

Remarque : Dire que deux automates sont isomorphes, c'est dire qu'il s'agit du même automate au nom des états près. Une induction immédiate montre que, si M et M' sont isomorphes, alors on a $f(T^*(q_0, w)) = T'^*(q'_0, w)$ pour tout mot w, et donc M et M' sont équivalents.

5.2.2 – Objectif

Partant d'un automate quelconque M, on veut construire un automate M' équivalent à M et plus petit au sens où M' a moins d'états que M (si c'est encore possible) :

- ☞ Suppression des états inaccessibles ;
- ☞ Utilisation du principe de l'identification des états

5.2.3 – Définition (état accessible)

Supposons que (Q, Σ, T, q_0, A) est un automate. L'état q est dit **accessible** s'il existe au moins un mot w dans Σ^* vérifiant $q = T^*(q_0, w)$.

5.2.4 – Lemme

Supposons que M est un automate, soit $M = (Q, \Sigma, T, q_0, A)$. Soit Q^a l'ensemble des états accessibles de M et M^a l'automate $(Q^a, \Sigma, T|Q^a, q_0, A \cap Q^a)$. Alors, M^a est équivalent à M, et tous les états de M^a sont accessibles.

5.2.5 – Détermination pratique des états accessibles

But : Partant d'un automate M, déterminer l'automate M^a.

Méthode : Supposons $M = (Q, \Sigma, T, q_0, A)$. Construire une suite $Q_0, Q_1, \ldots$ d'ensembles d'états en partant de $Q_0 = \{q_0\}$, et en posant $Q_{k+1} = Q_k \cup \{T(q, x); q \in Q_k, X \in \Sigma\}$. Alors Q^a est le premier ensemble Q_l tel que Q_{l+1} est égal à Q_l.

5.2.6 – Définition (états indiscernables)

Supposons que (Q, Σ, T, q_0, A) est un automate. Deux états q, q' de Q sont dits indiscernables, noté $q \approx q'$, si, quel que soit le mot u dans Σ^*, les états $T^*(q, u)$ et $T^*(q', u)$ sont ou bien tous les deux dans A, ou bien tous les deux hors de A.

5.2.7 – Lemme

Supposons que M est un automate. Alors $\approx$ est une relation d'équivalence sur l'ensemble des états de M, elle est compatible avec la fonction de transition de M au sens où $q \approx q'$ entraîne $T(q,x) \approx T(q',x)$ pour tout x dans Σ et l'ensemble des états acceptants de M est saturé pour $\approx$ (c'est-à-dire que tout état équivalent à un état acceptant est acceptant).

5.2.8 – Lemme

Supposons que M est un automate, soit $M = (Q, \Sigma, T, q_0, A)$, et que $\sim$ est une relation d'équivalence sur Q telle que $q \sim q'$ entraîne $T(q,x) \sim T(q',x)$ pour tout x dans Σ et telle que A est saturé pour $\sim$. Alors la formule $\bar{T}(\bar{q},x) = \overline{T(q,x)}$ définit une fonction de $Q/\sim \times \Sigma$ dans $Q/\sim$, et, notant $\bar{A}$ l'ensemble des classes $\bar{q}$ pour $q \in A$, l'automate $(Q/\sim, \Sigma, \bar{T}, \overline{q_0}, \bar{A})$ est équivalent à l'automate M.

5.2.9 – Définition (automate-quotient)

L'automate $(Q/\sim, \Sigma, \bar{T}, \bar{q_0}, \bar{A})$ est appelé **automate-quotient** de M par $\sim$, et est noté $M/\sim$.

5.2.10 – Définition (minimalisé)

Soit M un automate. On appelle **minimalisé** de M l'automate M^{min} défini par $M^{min} = (M^a)/\approx$.

5.2.11 – Lemme

Soit M un automate. L'automate M^{min} est équivalent à M, tous ses états sont accessibles, et deux états distincts sont discernables.

5.2.12 – Algorithme (calcul de M^{min})

But : Partant d'un automate M, calculer M^{min}.

Méthode :

- Calculer d'abord M^a ;
- Supposons $M^a = (Q, \Sigma, T, q_0, A)$. Construire une suite de partitions de plus en plus fines de Q, en partant de $\Pi_0 = (A, Q \backslash A)$.

  Supposant $\Pi_k = (Q_1, \ldots, Q_m)$, Π_{k+1} s'obtient en partageant chacun des ensembles Q_i en autant de nouveaux ensembles $Q'_1, Q'_2, \ldots$ que nécessaire pour que, pour tous q, q' dans Q'_i et pour toute lettre x dans Σ, les états $T(q, x)$ et $T(q', x)$ appartiennent à la même classe de Π_k. On arrête au plus petit indice l vérifiant $\Pi_{l+1} = \Pi_l$. Alors $q \approx q'$ est vrai si et seulement si q et q' sont dans la même classe de Π_l.
- Déterminer alors $M^{min} = (M^a)/\approx$.

5.3 – Classes à droite suivant un langage

L'automate M^{min} ne dépend que du langage décidé par M, mais pas vraiment de M lui-même : en partant d'un autre automate M' équivalent à M, l'automate $(M')^{min}$ est isomorphe à M^{min}.

5.3.1 – Définition (classes à droite)

Supposons que L est un langage d'alphabet Σ. Pour tout w dans Σ^*, on pose :

$$L/w = \{u \in \Sigma^*; wu \in L\}$$

Pour w et w' dans Σ^*, la relation $L/w = L/w'$ est notée $w \equiv_L w'$, et ses classes d'équivalence sont appelées **classes à droite suivant** L. Leur ensemble est noté Q_L.

Remarque :

- On notera $\bar{w}$ la classe à droite du mot w suivant L ;
- On a toujours $L/\epsilon = L$;
- Même si un langage L est non vide, un langage L/w peut l'être : par exemple, pour $L = \{a\}.\{a, b\}^*$, le langage L/b est vide.

5.3.2 – Lemme

Supposons que L est un langage d'alphabet Σ. La relation $\equiv_L$ est une relation d'équivalence sur Σ^*, et elle est compatible à droite avec le produit.

5.3.3 – Définition (équivalence associée à un automate)

Supposons que M est un automate, soit $M = (Q, \Sigma, T, q_0, A)$. Deux mots w, w' de Σ^* sont dits **équivalents suivant** M, noté $w \equiv_M w'$ si on a $T^*(q_0, w) = T^*(q_0, w')$.

5.3.4 – Lemme

Supposons que M est un automate possédant m états accessibles. La relation $\equiv_M$ est une relation d'équivalence sur Σ^*, elle est compatible à droite avec le produit, et admet m classes d'équivalence.

5.3.5 – Lemme

Supposons que M est un automate décidant le langage L. Alors $w \equiv_M w'$ entraîne $w \equiv_L w'$ et le nombre d'états accessibles de M est au moins égal au nombre de classes à droite suivant L.

5.3.6 – Théorème de Myhill-Nerode

Supposons que L est un langage d'alphabet Σ. Alors il y a équivalence entre :

- ☞ le langage L est rationnel ;
- ☞ le nombre de classes à droite suivant L est fini.

Plus précisément, si L possède m_L classes à droite, alors il existe un automate M_L à m_L états décidant L ; tout automate décidant L a au moins m_L états, et tout automate à m_L états décidant L est isomorphe à M_L.

5.3.7 – Définition (automate minimal)

Supposons que L est un langage rationnel. On appelle automate minimal de L, et on note M_L, l'automate dont les états sont les classes à droite suivant L.

5.3.8 – Lemme

Supposons que M est un automate à m états dont tous les états sont accessibles et deux à deux discernables. Alors le langage décidé par M possède au moins m classes à droite.

5.3.9 – Proposition (minimalisation)

Supposons que L est un langage rationnel. Alors, pour tout automate M décidant L, l'automate M^{min} est isomorphe à l'automate minimal du langage L.

Position dans le cours

6 – Grammaires formelles

Nous considérons une nouvelle approche pour spécifier des langages, à savoir celles des règles de réécriture et des grammaires formelles.

6.1 – Règles de réécriture

- On se donne une relation binaire (fonctionnelle ou non) sur l'ensemble des mots, et on considère l'ensemble des mots déduits d'un mot initial fixé en itérant l'application de la relation.
- En fait, on parle de règles de réécriture plutôt que de relations binaires sur les mots et on note $\vdash$ une telle règle.
- On lit $w \vdash w'$ comme "le mot w peut se réécrire en le mot w'" ou "le mot w' est dérivable à partir du mot w".

- On note $\vdash^p$ la p-ième puissance de $\vdash$, c'est à dire la règle de réécriture telle que $w \vdash^p w'$ est vraie si et seulement si il existe des mots $w_1, \ldots, w_{p-1}$ vérifiant $w \vdash w_1$, $w_1 \vdash w_2$, ..., $w_{p-1} \vdash w'$: on dit alors que w' est dérivable à partir de w en p étapes.
- On note $\vdash^*$ la réunion des règles $\vdash^p$ pour $p \geq 0$, $\vdash^0$ étant l'égalité
- On introduit, pour une règle de réécriture $\vdash$ et un mot initial w_0 donnés, le langage :

$$\{w \in \Sigma^*; w_0 \vdash^* w\}$$

6.2 – Productions

Les langages obtenus en appliquant des règles de réécriture peuvent être très complexes si l'on n'impose aucune restrictions sur ces règles.

6.2.1 – Définition (production)

Une **production** sur l'alphabet Σ est un couple (s, u) où s est une lettre de Σ, et u un mot sur σ. Il est usuel de noter $s \to u$ la production (s, u). Si $s \to u$ est une production sur Σ, on lui associe la règle de réécriture $\vdash_{s \to u}$ sur Σ^* telle que $w \vdash_{s \to u} w'$ est vraie si et seulement si il existe des mots w_1 et w_2 vérifiant $w = w_1 s w_2$ et $w' = w_1 u w_2$.

6.3 – Grammaire

6.3.1 – Définition

Une grammaire G est un quadruplet $(\Sigma_t, \Sigma_n, S, R)$ où :

1. Σ_t est un ensemble fini de symboles dits terminaux, appelé **vocabulaire terminal** i.e., l'alphabet sur lequel est défini le langage ;
2. Σ_n est un ensemble fini (disjoint de Σ_t) de symboles dits non-terminaux, appelé **vocabulaire non terminal** i.e., l'ensemble des symboles qui n'apparaissent pas dans les mots générés, mais qui sont utilisés au cours de la génération ;
3. $\Sigma = \Sigma_n \cup \Sigma_t$ est appelé le **vocabulaire de la grammaire G** ;
4. S est un non-terminal particulier appelé **source**, ou **axiome**, ou encore **symbole d'entrée**. C'est à partir de ce symbole non terminal que commencera la génération de mots au moyen des règles de la grammaire ;
5. R est un ensemble fini de règles de grammaires de la forme $u_1 \longrightarrow u_2$, où $u_1 \in (\Sigma_n \cup \Sigma_t)^-$ et $u_2 \in (\Sigma_n \cup \Sigma_t)^*$.

6.3.2 – Terminologie

- Une suite de symboles terminaux et non terminaux (i.e., un élément de $(\Sigma_n \cup \Sigma_t)^*$) est appelée une **forme** ;
- Une règle $u_1 \rightarrow u$ telle que $u \in \Sigma_t^*$ est appelée une **règle terminale**.

Remarque : Le langage défini, ou généré, par une grammaire est l'ensemble des mots qui peuvent être obtenus à partir du symbole de départ par application des règles de la grammaire.

6.4 – Dérivation

6.4.1 – Dérivation en une étape

Soit une grammaire $G = (\Sigma_t, \Sigma_n, S, R)$, une forme non vide $u \in (\Sigma_n \cup \Sigma_t)^+$ et une forme éventuellement vide $v \in (\Sigma_n \cup \Sigma_t)^*$. La grammaire G permet de dériver v de u en une étape (noté $u \Rightarrow v$) si et seulement si :

- $u = xu'y$ (u peut être décomposé en x, u' et y, avec x et y qui peuvent être vides) ;
- $v = xv'y$ (v peut être décomposé en x, v' et y),
- $u' \rightarrow v'$ est une règle de R.

6.4.2 – Dérivation en plusieurs étapes

Une forme v peut être dérivée d'une forme u en plusieurs étapes :

- $u \Rightarrow^* v$: si v peut être obtenue de u par une succession de 0, 1 ou plusieurs dérivations en une étape ;
- $u \Rightarrow^+ v$: si v peut être obtenue de u par une succession de 1 ou plusieurs dérivations en une étape.

6.5 – Langage généré

6.5.1 – Définition

Le langage généré par une grammaire $G = (\Sigma_t, \Sigma_n, S, R)$ est l'ensemble des mots sur Σ_t qui peuvent être dérivés à partir de S.

$$L(G) = v \in \Sigma_t^* / S \Rightarrow^* v$$

Remarque : une grammaire définit un seul langage. Par contre, un même langage peut être engendré par plusieurs grammaires différentes.

6.5.2 – Définition (langage algébrique)

Un langage est dit **algébrique** s'il existe au moins une grammaire qui engendre L.

6.6 – Type de grammaires

En introduisant des critères plus ou moins restrictifs sur les règles de production, on obtient des classes de grammaires hiérarchisées, ordonnées par inclusion. La classification des grammaires, définie en 1957 par CHOMSKY, distingue les 4 classes suivantes :

- **Type-0** : pas de restriction sur les règles
- **Type-1** : grammaires sensibles au contexte ou contextuelles. Les règles de R sont de la forme :

  $$uAv \rightarrow uwv$$

  avec $A \in \Sigma_n$; $u, v \in (\sigma_n \cup \Sigma_t)^*$ et $w \in (\Sigma_n \cup \Sigma_t)^+$. Autrement dit, le symbole non terminal A est remplacé par la forme w de $(\Sigma_n \cup \Sigma_t)^+$ si on a les contextes u à gauche et v à droite.
- **Type-2** : grammaires hors-contexte, Les règles de R sont de la forme :

  $$A \rightarrow w$$

  avec $A \in \Sigma_n$ et $w \in (\Sigma_n \cup \Sigma_t)^*$. Autrement dit, le membre de gauche de chaque règle est constitué d'un seul symbole non terminal.

☞ **Type-3** : grammaires régulières à droite (resp. à gauche). Les règles de R sont de la forme :

$$A \to aB \text{ (resp. } A \to Ba) \text{ ou } A \to a$$

avec $A, B \in \sigma_n$ et $a \in T$. Autrement dit, le membre de gauche de chaque règle est constitué d'un seul symbole non terminal, et le membre de droite est constitué d'un symbole terminal éventuellement suivi (resp. précédé) d'un seul non terminal.

Remarques :

- Les langages qui ne peuvent pas être générés par une grammaire de type 0 sont dits "indécidables" ;
- Les langages sont ordonnés par inclusion : l'ensemble des langages générés par les grammaires de type n est strictement inclus dans celui des grammaires de type $n-1$ (pour $n \in \{1, 2, 3\}$).
- A chaque type de grammaire est associé un type d'automate qui permet de reconnaître les langages de sa classe :
  - Les langages réguliers sont reconnus par des automates finis ;
  - Les langages hors-contexte sont reconnus par des automates à pile ;
  - les autres langages, décrits par des grammaires de type 1 ou 0, sont reconnus par des machines de Turing.

6.6.1 – Lemme

Supposons que G est une grammaire, et que w' est dérivable en p étapes à partir de w. Alors, pour tous mots w_1, w_2, le mot $w_1 w' w_2$ est dérivable à partir de $w_1 w w_2$ en p étapes.

6.6.2 – Proposition (compatibilité)

Supposons que G est une grammaire.

- ☞ Si w'_1 est dérivable à partir de w_1 en p_1 étapes et w'_2 est dérivable à partir de w_2 en p_2 étapes, alors $w'_1 w'_2$ est dérivable à partir de $w_1 w_2$ en $p_1 + p_2$ étapes.
- ☞ Inversement, si w' est $G-$dérivable en p étapes à partir de $w_1 w_2$, il existe une décomposition (w'_1, w'_2) de w' et deux entiers p_1, p_2 dont la somme est p tels que w'_1 est dérivable à partir de w_1 en p_1 étapes et w'_2 est dérivable à partir de w_2 en p_2 étapes.

6.7 – Construction de grammaires

On cherche à savoir si les langages rationnels sont algébriques i.e., si partant d'un automate, on peut construire une grammaire qui engendre exactement les mots acceptés par l'automate.

6.7.1 – Proposition (automatique)

Supposons que M est un automate, soit $M = (Q, \Sigma, T, q_0, A)$. Alors le langage décidé par M est engendré par la grammaire (Q, Σ, P, q_0), où P est l'ensemble des productions de la forme $q \to xT(q, x)$ pour q dans Q et x dans Σ, augmenté des productions $q \to \epsilon$ pour q dans A.

Remarque : Les grammaires ainsi obtenus, où toutes les productions sont du type $A \to xB$ ou $A \to \epsilon$, sont appelées **régulières**.

6.8 – Opérations sur les langages algébriques

6.8.1 – Proposition (union)

Supposons que G_1 et G_2 sont deux grammaires de même alphabet terminal, soit $G_i = (V_i, \Sigma, P_i, S_i)$. On suppose en outre V_1 et V_2 disjoints. Alors la grammaire $(V_1 \cup V_2 \cup \{S\}, \Sigma, P_1 \cup P_2 \cup \{S \to S_1 | S_2, S)$ engendre $L(G_1) \cup L(G_2)$.

6.8.2 – Proposition (produit)

Supposons que G_1 et G_2 sont deux grammaires de même alphabet terminal, soit $G_i = (V_i, \Sigma, P_i, S_i)$. On suppose en outre V_1 et V_2 disjoints. Alors la grammaire $(V_1 \cup V_2 \cup \{S\}, \Sigma, P_1 \cup P_2 \cup \{S \to S_1.S_2, S)$ engendre $L(G_1).L(G_2)$.

6.8.3 – Proposition (étoile)

Supposons que G_0 est une grammaire, soit $G_0 = (V_0, \Sigma, P_0, S_0)$. Alors la grammaire $(V_0 \cup \{S\}, \Sigma, P_0 \cup \{S \to S_0 S | \epsilon\}, S)$ engendre $L(G_0)^*$.

6.8.4 – Corollaire

La classe des langages algébriques est close par union, produit et opération étoile.

6.8.5 – Définition (grammaires équivalentes)

Supposons que G et G' sont deux grammaires de même alphabet terminal. On dit que G et G' sont **équivalentes** si elles engendrent le même langage.

6.8.6 – Lemme

Supposons que G est une grammaire et que le mot u est $G-$dérivable à partir de X. Alors la grammaire G' obtenue en ajoutant à G la production $X \to u$ est équivalente à G.

6.9 – Suppression des epsilon-transitions

6.9.1 – Définition ($\epsilon-$production)

On appelle $\epsilon-$production toute production du type $X \to \epsilon$.

6.9.2 – Algorithme (antécedents de ϵ)

Avant de supprimer les $\epsilon-$productions, il faut déterminer les variables dont le mot ϵ est dérivable.

But : Soit G une grammaire, soit $G = (V, \Sigma, P, S)$. Déterminer l'ensemble $V_\epsilon = \{X \in V ; X \vdash_G^* \epsilon\}$.

Méthode : Définit inductivement un sous-ensemble V_k de V par :

$$V_k = \begin{cases} \{X \in V ; X \to \epsilon \in P\} & \text{pour } k = 1 ; \\ V_{k-1} \cup \{X \in V ; (\exists \in V_{k-1}^*)(X \to u \in P)\} & \text{pour } k \geq 2. \end{cases}$$

Alors on a $V_\epsilon = V_l$, où l est le plus petit indice vérifiant $V_{l+1} = V_l$.

6.9.3 – Algorithme (suppression $\epsilon-$production)

But : Soit G une grammaire quelconque ; construire une grammaire G' sans $\epsilon-$production qui engendre le langage $L(G)\backslash\{\epsilon\}$.

Méthode : Soit $G = (V, \Sigma, P, S)$; déterminer $V_\epsilon = \{X \in V ; X \vdash_G^* \epsilon\}$ à l'aide de l'algorithme précédent ; former G' à partir de G en supprimant toutes les $\epsilon-$productions et en ajoutant, pour chaque production $X \to u$ de G avec $u \neq \epsilon$, toutes les productions $X \to u'$ où u' est un mot non vide obtenu à partir de u en supprimant des occurrences de variables dans V_ϵ.

6.10 – Suppression des productions-unité

L'étape suivante consiste à supprimer les productions dont le second membre est une variable.

6.10.1 – Définition (production-unité,cycle)

Une **production-unité** est une production du type $X \to Y$, avec X, Y variables. On dit que les variables $X_1, \ldots, X_l$ forment un **cycle** de longueur l pour la grammaire G si G contient les productions $X_1 \to X_2 \to \ldots \to X_l \to X_l$.

6.10.2 – Algorithme (suppression des cycles)

But : Partant de G grammaire sans $\epsilon-$production, construire une grammaire équivalente à G sans cycle de variables.

Méthode : Si $X_1, \ldots, X_l$ forment un cycle pour G, former G' à partir de G en remplaçant partout $X_2, \ldots, X_l$ par X_1. Itérer jusqu'à ce qu'il ne reste plus de cycle (ce qu'on peut effectivement tester en explicitant le graphe des productions unité restantes).

6.10.3 – Algorithme (suppression des productions-unité)

But : Partant de G grammaire sans $\epsilon-$production et sans cycle de variables, construire une grammaire équivalente à G sans production-unité.

Méthode : Supposons $G = (V, \Sigma, P, S)$. Choisir une variable X_0 telle qu'il existe au moins une production-unité du type $X \to X_0$, mais pas de production-unité de type $X_0 \to X$. On définit G' à partir de G en supprimant les productions-unité $X \to X_0$ et en ajoutant, pour chacune d'elles, toutes les productions $X \to u$ telles que $X_0 \to u$ est dans P. Itérer jusqu'à ce qu'il ne reste plus de production-unité.

6.10.4 – Lemme

Supposons que G est une grammaire sans $\epsilon-$production ni production-unité. Alors, pour tout mot w, et toute variable X de G, il y a équivalence entre :

- Le mot w est $G-$dérivable à partir de X ;
- Le mot w est $G-$dérivable à partir de X en au plus $2n-1$ étapes, n étant la longueur de w.

6.10.5 – Proposition (décidabilité)

Supposons que G est une grammaire d'alphabet terminal Σ. Alors il existe un algorithme qui, partant d'un mot w sur Σ, décide si w appartient ou non à $L(G)$.

Il est maintenant facile de montrer que toute grammaire est équivalente, au mot vide près, à une grammaire d'un type normalisé.

6.11 – Forme normale de chomsky

6.11.1 – Définition

Soit G une grammaire. G est **en forme normale de Chomsky** si ses productions sont toutes d'un des 2 types $X \to YZ$ avec Y, Z variables, et $X \to x$ avec x terminal.

6.11.2 – Algorithme (forme normale de Chomsky)

But : Partant de G grammaire sans $\epsilon-$production ni production-unité, construire une grammaire équivalente à G en forme normale de Chomsky.

Méthode :

- Introduire la variable X_x pour chaque terminal, ainsi que la production $X_x \to x$;
- Remplacer x par X_x dans les seconds membres des productions de longueur ≥ 2 ;
- Remplacer chaque production $X \to X_1 X_2 \dots X_l$ avec $l \geq 3$ par $X \to X_1 Y_1$, $Y_1 \to X_2 Y_2, \dots, Y_{l-2} \to X_{l-1} X_l$, où $Y_1, \dots, Y_{l-2}$ sont de nouvelles variables.

6.11.3 – Proposition (forme normale)

Si L est un langage algébrique alors il existe une grammaire en forme normale de Chomsky qui engendre $L \backslash \{\epsilon\}$.

6.12 – Forme normale de Greibach

6.12.1 – Définition

Soit G une grammaire. On dit que G est **en forme normale de Greibach** si ses productions sont toutes du type $X \to xu$, avec x terminal.

6.12.2 – Lemme

Soient G une grammaire en forme normale de Chomsky, X une variable de G, et $X \to u_1|\ldots|u_p$ les productions de G dont le premier membre est X. Alors G est équivalente à la grammaire G' obtenue en supprimant les productions du type $Y \to Xu$ avec $Y \neq X$ et en ajoutant les productions $Y \to u_1|u\ldots|u_pu$.

6.12.3 – Lemme

Soit G une grammaire en forme normale de Chomsky, et X une variable de G. Soient $X \to Xu_1|\ldots|Xu_p$ les productions dont le second membre commence par X, et $X \to v_1|\ldots|v_q$ les autres productions de G commençant par X. Alors G est équivalente à la grammaire G' obtenue en introduisant une nouvelle variable X' et remplaçant les productions précédentes par
$X \to v_1|\ldots|v_q|v_1X'|\ldots|v_qX', X' \to u_1|\ldots|u_p|u_1X'|\ldots|u_pX'$.

6.12.4 – Algorithme (forme normale de Greibach)

But : Partant de G grammaire en forme normale de Chomsky, construire une grammaire équivalente à G en forme normale de Greibach.

Méthode : Supposons $G = (V, \Sigma, P, S)$, avec $V = \{X_1, \ldots, X_m\}$. On introduit m nouvelles variables $X'_1, \ldots, X'_m$, et on construit $2m$ grammaires successives en partant de $G_1 = G$:

- G_{2i} s'obtient à partir de G_{2i-1} en appliquant la méthode du lemme précédent 6.12.3 à la variable X_i et en introduisant X'_i ;
- G_{2i+1} s'obtient à partir de G_{2i} en appliquant la méthode du lemme 6.12.2 à la variable X_i.

La grammaire G_{2m+1} est équivalente à G, et elle est en forme normale de Greibach.

Position dans le cours

7 – Automates à pile

On définit d'abord les automates à pile non-déterministes car ils sont plus puissants que les automates à pile déterministes.

7.1 – Automate à pile non-déterministe

7.1.1 – Description

Un **automate à pile non-déterministe** se compose des mêmes éléments qu'un automate fini, i.e. :

- un ruban d'entrée et une tête de lecture ;
- un ensemble d'états dont un état initial et des états accepteurs ;
- une relation de transition ;

Particularité : Une pile de capacité illimitée initialement vide.

A chaque étape, l'automate à pile consulte une partie du sommet de sa pile et la remplace par une autre suite de symboles.

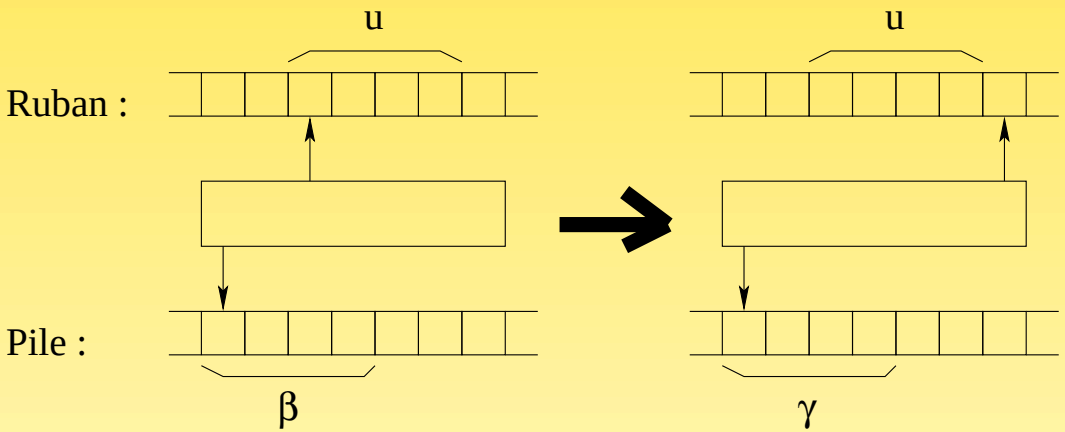
7.1.2 – Formalisation

Un **automate à pile non-déterministe** est défini par un septuplet $M = (Q, \Sigma, \Gamma, \Delta, Z, s, F)$, où :

- Q est un ensemble fini d'états ;
- Σ est un **alphabet d'entrée** ;
- Γ est un **alphabet de pile** ;
- $Z \in \Gamma$ est le **symbole initial de pile** ;
- $s \in Q$ est l'état initial ;
- $F \subseteq Q$ est l'état initial ;
- $\Delta \subset ((Q \times \Sigma^* \times \Gamma^*) \times (Q \times \Gamma^*))$ est la relation de transition (ensemble fini).

7.1.3 – Représentation

On peut représenter un automate à pile à l'aide de 2 rubans et 2 têtes de lecture (une pour le ruban et l'autre pour désigner le sommet de la pile).



7.1.4 – Définition (dérivabilité en une étape)

La configuration (q', w', α') est dérivable en **une étape** de la configuration (q, w, α) par la machine M (notation $(q, w, \alpha) \vdash_M (q', w', \alpha')$) si :

- $w = uw'$ (le mot w commence par le préfixe $u \in \Sigma^*$) ;
- $\alpha = \beta\delta$ (avant la transition le sommet de la pile lu de gauche à droite contient $\beta \in \Gamma^*$) ;
- $\alpha' = \gamma\delta$ (après la transition la partie β de la pile a été remplacée par γ, le premier symbole de γ est maintenant le sommet de la pile) ;
- $((q, u, \beta), (q', \gamma)) \in \Delta$.

7.1.5 – Définition (dérivabilité en plusieurs étape)

Une configuration C' est **dérivable en plusieurs étapes** de la configuration C par la machine M (notation $C \vdash_M^* C'$) s'il existe $k \geq 0$ et des configurations intermédiaires $C_0, C_1, C_2, ..., C_k$ telles que :

- $C = C_0$
- $C' = C_k$
- $C_i \vdash_M C_{i+1}$ pour $0 \leq i < k$.

7.1.6 – Définition (exécution)

Une **exécution d'un automate à pile** sur un mot w est une suite de configurations

$(s, w, Z) \vdash (q_1, w_1, \alpha_1) \vdash ... \vdash (q_n, w_n, \alpha_n) \vdash ...$ où s est l'état initial et Z est le symbole initial de pile qui est maximale, c'est à dire telle que soit :

- ☞ Elle aboutit à une configuration (p, ϵ, γ) où le mot d'entrée a été entièrement consommée et où l'état est accepteur ;
- ☞ Elle aboutit à une configuration à partir de laquelle aucune transition n'est possible
- ☞ Elle est infinie.

7.1.7 – Définition (acceptation)

Un mot w est **accepté par un automate à pile** $M = (Q, \Sigma, \Gamma, \Delta, Z, s, F)$ si :

$(s, w, Z) \vdash_M^* (p, \epsilon, \gamma)$, avec $p \in F$.

7.2 – Les langages hors-contexte

7.2.1 – Définition

Un langage est hors-contexte s'il existe une grammaire hors-contexte qui génère ce langage.

7.2.2 – Théorème

Un langage est hors-contexte si et seulement si il est accepté par un automate à pile.

7.2.3 – Propriétés

Soient L_1 et L_2 deux langages hors-contexte.

- Le langage $L_1 \cup L_2$ est hors-contexte.
- Le langage $L_1 \cdot L_2$ est hors-contexte.
- Le langage L_1^* est hors-contexte.
- Les langages $L_1 \cap L_2$ et $\overline{L_1}$ ne sont pas forcément hors-contexte.
- Si le langage L_R est régulier et le langage L_2 hors-contexte alors le lanage $L_R \cap L_2$ est hors-contexte.

7.3 – Arbre d'analyse (arbre de dérivation)

L'objectif est de donner à la dérivation une structure d'arbre.

7.3.1 – Définition

Un arbre d'analyse pour une grammaire hors-contexte $G = (V, \Sigma, R, S)$ est un arbre dont chaque noeud est étiqueté par un élément de $V \cup \epsilon$ et qui satisfait les conditions suivantes :

- La racine est étiquetée par le symbole de départ S
- Chaque noeud intérieur est étiqueté par un non-terminal (un élément de $V\ \Sigma$)
- Chaque feuille est étiquetée par un symbole terminal (un élément de Σ) ou par ϵ (le mot vide)
- Pour tout noeud intérieur, si son étiquette est le non-terminal A et si ses descendants immédiats sont les noeuds $n_1, n_2, ..., n_k$ ayant respectivement pour étiquettes $X_1, X_2, ..., X_k$, alors $A \rightarrow X_1X_2...X_k$ doit être une production de G
- Si un noeud est étiqueté par ϵ, alors ce noeud est le seul descendant immédiat de son prédécesseur (cette dernière règle évite simplemet l'introduction d'instances inutiles de ϵ dans l'arbre d'analyse).

7.3.2 – Définition

Le mot généré par un arbre d'analyse est celui obtenu par la concaténation des feuilles de l'arbre prises de gauche à droite.

7.3.3 – Théorème

Etant donné une grammaire hors-contexte G, un mot w est généré par $G(S \Rightarrow_G^* w)$ si et seulement si il existe un arbre d'analyse de la grammaire G qui génère w.

7.4 – Théorème du “gonflement”

Le théorème du “gonflement” pour les langages hors-contexte affirme qu'à partir d'un mot suffisamment long, il est possible de construire d'autres mots du langage en répétant une **ou deux** sous-chaînes du mot.

7.4.1 – Théorème

Soit un langage hors-contexte L. Alors, il existe une constante K telle que tout mot $w \in L$ satisfaisant $|w| \geq K$ peut être écrit $w = uvxyz$ avec v ou $y \neq \epsilon$, $|vxy| \leq K$ et $yv^n xy^n z \in L$ pour tout $n > 0$.

7.5 – Automates à pile déterministe

7.5.1 – Intérêt

L'intérêt des automates déterministes est que, contraitement aux automates non-déterministes, ils représentent une procédure de calcul qui peut être directement simulée (donc un algorithme).

7.5.2 – Définition (compatible)

Deux transitions $((p_1, u_1, \beta_1), (q_1, \gamma_1))$ et $((p_2, u_2, \beta_2), (q_2, \gamma_2))$ sont compatibles si :

- $p_1 = p_2$
- u_1 et u_2 compatibles (i.e. u_1 est un préfixe de u_2 ou vice versa)
- β_1 et β_2 compatibles.

7.5.3 – Définition (déterministe)

Un automate à pile est déterministe si pour toute paire de transitions compatibles, ces transitions sont identiques.

Remarque : Tous les langages hors-contexte ne sont pas acceptés par des automates déterministes.

7.5.4 – Langage hors-contexte déterministe

Soit L un langage défini sur un alphabet Σ, le langage L est un langage hors-contexte déterministe si et seulement si il est accepté par un automate à pile déterministe.

7.5.5 – Propriétés

Si L_1 et L_2 sont des langages hors-contexte déterministes, on a les propriétés suivantes :

- ☞ Le langage $\Sigma^* - L_1$ est aussi hors-contexte déterministe.
- ☞ Les langages $L_1 \cup L_2$ et $L_1 \cap L_2$ ne sont pas forcément déterministes hors-contexte.