Langages formels, calculabilité, complexité et analyse d'algorithmes

Paul Gastin

Paul.Gastin@liafa.jussieu.fr

LIAFA, Université Paris 7, UMR CNRS 7089

Introduction

1. Langages formels

Description et analyse (lexicale et syntaxique) des langages (programmation, naturels, ...)
Abstractions mathématiques simples de phénomènes complexes dans le but de

- Prouver des propriétés.
- Concevoir des algorithmes permettant de tester des propriétés ou de résoudre des problèmes.

2. Calculabilité

Définition mathématique de ce qui est *algorithmiquement calculable*.
Doit être suffisamment simple pour permettre de faire des preuves de décidabilité/indécidabilité.
Ne doit pas dépendre d'une technologie particulière.

- Machines de Turing (1936),
- Machines RAM (Stepherdson et Sturgis 1963)
- Fonctions récursives (Gödel 1931, Kleene 1936)

Introduction

3. Complexité

Combien de temps/d'espace faut-il pour résoudre un problème ?
La réponse ne dépend que du problème et pas d'un algorithme particulier qui résout ce problème.

4. Analyse d'algorithmes

Analyser les ressources (temps/espace) utilisées par un algorithme particulier.
Pire cas, cas moyen, coût amorti, coût asymptotique.

Bibliographie

[1] Luc Albert, Paul Gastin, Bruno Petazzoni, Antoine Petit, Nicolas Puech et Pascal Weil. *Cours et exercices d'informatique*. Vuibert, 1998.

[2] Jean-Michel Autebert. *Calculabilité et décidabilité*. Masson, 1992.

[3] Jean-Michel Autebert. *Théorie des langages et des automates*. Masson, 1994.

[4] Jean-Michel Autebert, Jean Berstel et Luc Boasson. Context-Free Languages and Pushdown Automata. *Handbook of Formal Languages*, Vol. 1, Springer, 1997.

[5] Jean Berstel. *Transduction and context free languages*. Teubner, 1979.

Bibliographie

[6] Harry R. Lewis and Christos H. Papadimitriou. *Elements of the theory of computation*. Prentice-Hall, 1981.

[7] Christos H. Papadimitriou. *Computational complexity*. Addison Wesley, 1994.

[8] Robert Sedgewick et Philippe Flajolet. *Introduction à l'analyse des algorithmes*. International Thomson Publishing, 1996.

[9] Jacques Stern. *Fondements mathématiques de l'informatique*. Mc Graw Hill, 1990.

[10] Pierre Wolper. *Introduction à la calculabilité*. InterEditions, 1991.

1 Automates et reconnaissables

1. Mots
2. Langages
3. Automates déterministes
4. Automates non déterministes
5. Automates avec ε-transitions
6. Propriétés de fermeture
7. Langages rationnels
8. Critères de reconnaissabilité
9. Minimisation

1.1 Mots

A ou Σ : alphabet (ensemble fini).
$u \in \Sigma^*$: mot $=$ suite finie de lettres.

$\cdot$: concaténation associative.
ε ou 1 : mot vide, neutre pour la concaténation.
$(\Sigma^*, \cdot)$: monoïde libre engendré par Σ.

$|u|$: longueur du mot u.
$|\cdot| : \Sigma^* \to \mathbb{N}$ est le morphisme défini par $|a| = 1$ pour $a \in \Sigma$.
$|u|_a$: nombre de a dans le mot u.

$\tilde{u}$: miroir du mot u.

1.1 Mots

Ordres partiels :

- u préfixe de v si $\exists u'$, $v = uu'$
- u suffixe de v si $\exists u'$, $v = u'u$
- u facteur de v si $\exists u', u''$, $v = u'uu''$
- u sous-mot de v si $v = v_0u_1v_1u_1 \cdots u_nv_n$ avec $u_i, v_i \in \Sigma^*$ et $u = u_1u_2 \cdots u_n$

Théorème 1.1 (Higman) *L'ordre sous-mot est un* bon *ordre, i.e. (de toute suite infinie on peut extraire une sous-suite infinie croissante) (ou tout ensemble de mots a un nombre fini d'éléments minimaux)*

1.2 Langages

Langage = sous-ensemble de Σ^*.
Exemples.

Opérations sur les langages : soient $K, L \subseteq \Sigma^*$

Ensemblistes : union, intersection, complément, différence, ...

Concaténation : $K \cdot L = \{u \cdot v \mid u \in K \text{ et } v \in L\}$
La concaténation est associative et distributive sur l'union.
$|K \cdot L| \leq |K| \cdot |L|$
notion de multiplicité, d'ambiguïté

1.2 Langages

Itération : $L^0 = \{\varepsilon\}$, $L^{n+1} = L^n \cdot L = L \cdot L^n$,
$L^* = \bigcup_{n \geq 0} L^n$, $L^+ = \bigcup_{n > 0} L^n$.
Exemples : Σ^n, Σ^*, $(\Sigma^2)^*$.

Quotients : $K^{-1} \cdot L = \{v \in \Sigma^* \mid \exists u \in K, u \cdot v \in L\}$
$L \cdot K^{-1} = \{u \in \Sigma^* \mid \exists v \in K, u \cdot v \in L\}$

1.3 Automates déterministes

Définition 1.2 (Automate déterministe)
$\mathcal{A} = (Q, \delta, i, F)$
Q *ensemble* fini *d'états,* $i \in Q$ *état initial,* $F \subseteq Q$ *états finaux,*
$\delta : Q \times \Sigma \to Q$ *fonction de transition (totale ou partielle).*

Exemples.

Calcul de $\mathcal{A}$ sur un mot $u = a_1 \cdots a_n$: $q_0 \xrightarrow{u} q_n$

$$q_0 \xrightarrow{a_1} q_1 \cdots q_{n-1} \xrightarrow{a_n} q_n$$

avec $q_i = \delta(q_{i-1}, a_i)$ pour tout $0 < i \leq n$.

Généralisation de δ à $Q \times \Sigma^*$:
$\delta(q, \varepsilon) = q$,
$\delta(q, u \cdot a) = \delta(\delta(q, u), a)$ si $u \in \Sigma^*$ et $a \in \Sigma$.

1.3 Automates déterministes

Langage accepté (reconnu) par $\mathcal{A}$: $\mathcal{L}(\mathcal{A}) = \{u \in \Sigma^* \mid \delta(i,u) \in F\}$.
Exemples.

Définition 1.3 (Reconnaissables) *Un langage $L \subseteq \Sigma^*$ est* reconnaissable, *s'il existe un automate fini $\mathcal{A}$ tel que $L = \mathcal{L}(\mathcal{A})$.*

On note $\mathrm{Rec}(\Sigma^*)$ la famille des langages reconnaissables sur Σ^*.

1.4 Automates non déterministes

Exemple : automate non déterministe pour $\Sigma^* \cdot \{aba\}$

Définition 1.4 (Automate non déterministe)
$\mathcal{A} = (Q, T, I, F)$
Q *ensemble* fini *d'états,* $I \subseteq Q$ *états initiaux,* $F \subseteq Q$ *états finaux,*
$T \subseteq Q \times \Sigma \times Q$ *ensemble des transitions.*
On utilise aussi $\delta : Q \times \Sigma \to 2^Q$.

Calcul de $\mathcal{A}$ sur un mot $u = a_1 \cdots a_n$: $q_0 \xrightarrow{a_1} q_1 \cdots q_{n-1} \xrightarrow{a_n} q_n$ avec $(q_{i-1}, a_i, q_i) \in T$ pour tout $0 < i \leq n$.

1.4 Automates non déterministes

Langage accepté (reconnu) par $\mathcal{A}$:

$$\mathcal{L}(\mathcal{A}) = \{u \in \Sigma^* \mid \exists\ i \xrightarrow{u} f \text{ calcul de } \mathcal{A} \text{ avec } i \in I \text{ et } f \in F\}.$$

Théorème 1.5 (Déterminisation) *Soit $\mathcal{A}$ un automate non déterministe. On peut construire un automate déterministe $\mathcal{B}$ qui reconnaît le même langage ($\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{B})$).*

Preuve: Automate des parties □

Exemple : automate déterministe pour $\Sigma^* \cdot \{aba\}$

On appelle déterminisé de $\mathcal{A}$ l'automate des parties *émondé*.

1.4 Automates non déterministes

Exercices

1. Donner un automate non déterministe avec n états pour $L = \Sigma^* a \Sigma^{n-2}$.
2. Montrer que tout automate déterministe reconnaissant ce langage L a au moins 2^{n-1} états.
3. Donner un automate non déterministe à n états tel que tout automate déterministe reconnaissant le même langage a au moins $2^n - 1$ états.

1.4 Automates non déterministes

Un automate (D ou ND) est *complet* si $\forall p \in Q$, $\forall a \in \Sigma$, $\delta(p, a) \neq \emptyset$.
On peut toujours compléter un automate.

Un automate (D ou ND) est émondé si tout état $q \in Q$ est

- accessible d'un état initial : $\exists i \in I$, $\exists u \in \Sigma^*$ tels que $i \xrightarrow{u} q$,
- co-accessible d'un état final : $\exists f \in F$, $\exists u \in \Sigma^*$ tels que $q \xrightarrow{u} f$

On peut calculer l'ensemble $\mathrm{Acc}(I)$ des états accessibles à partir de I et l'ensemble $\mathrm{coAcc}(F)$ des états co-accessibles des états finaux.

Corollaire 1.6 *Soit $\mathcal{A}$ un automate.*

1. *On peut construire $\mathcal{B}$ émondé qui reconnaît le même langage.*
2. *On peut décider si $\mathcal{L}(\mathcal{A}) = \emptyset$.*

1.5 Automates avec ε-transitions

Exemple.

Définition 1.7 (Automate avec ε-transitions)
$\mathcal{A} = (Q, T, I, F)$
Q *ensemble* fini *d'états,* $I \subseteq Q$ *états initiaux,* $F \subseteq Q$ *états finaux,*
$T \subseteq Q \times (\Sigma \cup \{\varepsilon\}) \times Q$ *ensemble des transitions.*

Un calcul de $\mathcal{A}$ est une suite $q_0 \xrightarrow{a_1} q_1 \cdots q_{n-1} \xrightarrow{a_n} q_n$ avec $(q_{i-1}, a_i, q_i) \in T$ pour tout $0 < i \leq n$.

Ce calcul reconnaît le mot $u = a_1 \cdots a_n$ (les ε disparaissent).

Remarque : Soit $\mathcal{A}$ un automate. On peut construire un automate sans ε-transition $\mathcal{B}$ qui reconnaît le même langage.

1.6 Propriétés de fermeture

1.6.1 Opérations ensemblistes

Proposition 1.8 *La famille* $\mathrm{Rec}(\Sigma^*)$ *est fermée par les opérations ensemblistes (union, complément, . . .).*

Preuve: Union : construction non déterministe.
Intersection : produit d'automates (préserve le déterminisme).
Complément : utilise la déterminisation. □

Corollaire 1.9 *On peut décider de l'égalité ou de l'inclusion de langages reconnaissables.*
Plus précisément, soient $L_1, L_2 \in \mathrm{Rec}(\Sigma^*)$ *donnés par deux automates* $\mathcal{A}_1$ *et* $\mathcal{A}_2$*. On peut décider si* $L_1 \subseteq L_2$*.*

1.6.2 Opérations liées à la concaténation

Proposition 1.10 $\mathrm{Rec}(\Sigma^*)$ *est fermée par concaténation et itération.*

Concaténation :
Méthode 1 : union disjointe des automates et ajout de transitions.
Méthode 2 : fusion d'états.
On suppose que les automates ont un seul état initial sans transition entrante et un seul état final sans transition sortante.

Itération :
Méthode 1 : ajout de transitions. Ajouter un état pour reconnaître le mot vide.
Méthode 2 : ajout d'ε-transitions.

1.6 Propriétés de fermeture

Si $L \subseteq \Sigma^*$, on note

- $\mathrm{Pref}(L) = \{u \in \Sigma^* \mid \exists v \in \Sigma^*,\ uv \in L\}$,
- $\mathrm{Suff}(L) = \{v \in \Sigma^* \mid \exists u \in \Sigma^*,\ uv \in L\}$,
- $\mathrm{Fact}(L) = \{v \in \Sigma^* \mid \exists u, w \in \Sigma^*,\ uvw \in L\}$.

Proposition 1.11 $\mathrm{Rec}(\Sigma^*)$ *est fermée par préfixe, suffixe, facteur.*

Preuve: Modification des états initiaux et/ou finaux. □

1.6 Propriétés de fermeture

Proposition 1.12 *La famille* $\mathrm{Rec}(\Sigma^*)$ *est fermée par quotients gauches et droits :*
Soit $L \in \mathrm{Rec}(\Sigma^*)$ *et* $K \subseteq \Sigma^*$ *arbitraire.*
Les langages $K^{-1} \cdot L$ *et* $L \cdot K^{-1}$ *sont reconnaissables.*

Preuve: Modification des états initiaux et/ou finaux. □

Exercice Montrer que si de plus K est reconnaissable, alors on peut effectivement calculer les nouveaux états initiaux/finaux.

1.6.3 Morphismes

Soient A et B deux alphabets et $f : A^* \to B^*$ un *morphisme*.
Pour $L \subseteq A^*$, on note $f(L) = \{f(u) \in B^* \mid u \in L\}$.
Pour $L \subseteq B^*$, on note $f^{-1}(L) = \{u \in A^* \mid f(u) \in L\}$.

Proposition 1.13 *La famille des langages reconnaissables est fermée par morphisme et morphisme inverse.*

1. *Si* $L \in \mathrm{Rec}(A^*)$ *et* $f : A^* \to B^*$ *est un morphisme alors* $f(L) \in \mathrm{Rec}(B^*)$.
2. *Si* $L \in \mathrm{Rec}(B^*)$ *et* $f : A^* \to B^*$ *est un morphisme alors* $f^{-1}(L) \in \mathrm{Rec}(A^*)$.

Preuve: Modification des transitions de l'automate. □

1.6.4 Substitutions

Une *substitution* est définie par une application $\sigma : A \to \mathcal{P}(B^*)$.
Elle s'étend en un morphisme $\sigma : A^* \to \mathcal{P}(B^*)$ défini par
$\sigma(\varepsilon) = \{\varepsilon\}$ et
$\sigma(a_1 \cdots a_n) = \sigma(a_1) \cdots \sigma(a_n)$.

Pour $L \subseteq A^*$, on note $\sigma(L) = \bigcup_{u \in L} \sigma(u)$.
Pour $L \subseteq B^*$, on note $\sigma^{-1}(L) = \{u \in A^* \mid \sigma(u) \cap L \neq \emptyset\}$.

Une substitution est *rationnelle* (ou *reconnaissable*) si elle est définie par une application $\sigma : A \to \mathrm{Rec}(B^*)$.

1.6 Propriétés de fermeture

Proposition 1.14 *La famille des langages reconnaissables est fermée par substitution rationnelle et substitution rationnelle inverse.:*

1. *Si* $L \in \mathrm{Rec}(A^*)$ *et* $\sigma : A \to \mathrm{Rec}(B^*)$ *est une substitution rationnelle alors* $\sigma(L) \in \mathrm{Rec}(B^*)$.
2. *Si* $L \in \mathrm{Rec}(B^*)$ *et* $\sigma : A \to \mathrm{Rec}(B^*)$ *est une substitution rationnelle alors* $\sigma^{-1}(L) \in \mathrm{Rec}(A^*)$.

Preuve:

1. On remplace des transitions par des automates.
2. Plus difficile.

□

1.7 Langages rationnels

Syntaxe pour représenter des langages.

Soit Σ un alphabet et $\underline{\Sigma}$ une copie de Σ.
Une ER est un mot sur l'alphabet $\underline{\Sigma} \cup \{(,), +, \cdot, *, \underline{\emptyset}\}$

Définition 1.15 (Syntaxe) *L'ensemble des ER est défini par*

B : $\underline{\emptyset}$ *et* $\underline{a}$ *pour* $a \in \Sigma$ *sont des ER,*

I : *Si* E *et* F *sont des ER alors* $(E + F)$, $(E \cdot F)$ *et* (E^*) *aussi.*

On note $\mathcal{E}$ *l'ensemble des expressions rationnelles.*

1.7 Langages rationnels

Définition 1.16 (Sémantique) *On définit* $\mathcal{L} : \mathcal{E} \to \mathcal{P}(\Sigma^*)$ *par*

B : $\mathcal{L}(\underline{\emptyset}) = \emptyset$ *et* $\mathcal{L}(\underline{a}) = \{a\}$ *pour* $a \in \Sigma$,

I : $\mathcal{L}((E + F)) = \mathcal{L}(E) \cup \mathcal{L}(F)$, $\mathcal{L}((E \cdot F)) = \mathcal{L}(E) \cdot \mathcal{L}(F)$ *et* $\mathcal{L}((E^*)) = \mathcal{L}(E)^*$.

Un langage $L \subseteq \Sigma^*$ *est rationnel s'il existe une ER* E *telle que* $L = \mathcal{L}(E)$.
On note $\mathrm{Rat}(\Sigma^*)$ *l'ensemble des langages rationnels sur l'alphabet* Σ.

Remarque : $\mathrm{Rat}(\Sigma^*)$ est la plus petite famille de langages de Σ^* contenant $\emptyset$ et $\{a\}$ pour $a \in \Sigma$ et fermée par union, concaténation, itération.

1.7 Langages rationnels

Définition 1.17 *Deux ER E et F sont équivalentes (noté $E \equiv F$) si $\mathcal{L}(E) = \mathcal{L}(F)$.*

Exemples : commutativité, associativité, distributivité, ...

Peut-on trouver un système de règles de réécriture caractérisant l'équivalence des ER ?
Oui, mais il n'existe pas de système fini.

Comment décider de l'équivalence de deux ER ?
On va utiliser le théorème de Kleene.

Abus de notation :

- On ne souligne pas les lettres de Σ : $((a+b)^*)$.
- On enlève les parenthèses inutiles : $(aa+bb)^* + (aab)^*$.
- On confond langage rationnel et expression rationnelle.

1.7 Langages rationnels

Théorème 1.18 (Kleene, 1936) $\mathrm{Rec}(\Sigma^*) = \mathrm{Rat}(\Sigma^*)$

Preuve:

$\boxed{\supseteq}$: les langages $\emptyset$ et $\{a\}$ pour $a \in \Sigma$ sont reconnaissables et la famille $\mathrm{Rec}(\Sigma^*)$ est fermée par union, concaténation, itération.

$\boxed{\subseteq}$: Algorithme de McNaughton-Yamada.

□

Corollaire 1.19 *L'équivalence des expressions rationnelles est décidable.*

Preuve: Il suffit de l'inclusion $\mathrm{Rat}(\Sigma^*) \subseteq \mathrm{Rec}(\Sigma^*)$. □

1.8 Critères de reconnaissabilité

Y a-t-il des langages non reconnaissables ?
Oui, par un argument de cardinalité.

Comment montrer qu'un langage n'est pas reconnaissable ?

Exemples.

1. $L_1 = \{a^n b^n \mid n \geq 0\}$,
2. $L_2 = \{u \in \Sigma^* \mid |u|_a = |u|_b\}$,
3. $L_3 = L_2 \setminus (\Sigma^*(a^3 + b^3)\Sigma^*)$

Preuves : *à la main* (par l'absurde).

1.8 Critères de reconnaissabilité

Lemme 1.20 (itération) *Soit $L \in \mathrm{Rec}(\Sigma^*)$. Il existe $N \geq 0$ tel que pour tout $w \in L$,*

1. *si $|w| \geq N$ alors $\exists u_1, u_2, u_3 \in \Sigma^*$ tels que $w = u_1u_2u_3$, $u_2 \neq \varepsilon$ et $u_1u_2^*u_3 \subseteq L$.*
2. *si $w = w_1w_2w_3$ avec $|w_2| \geq N$ alors $\exists u_1, u_2, u_3 \in \Sigma^*$ tels que $w_2 = u_1u_2u_3$, $u_2 \neq \varepsilon$ et $w_1u_1u_2^*u_3w_3 \subseteq L$.*
3. *Si $0 \leq i_0 < i_1 < \cdots < i_N \leq |w|$ (positions marquées dans w) alors il existe $0 \leq j < k \leq N$ tels que si on écrit $w = u_1u_2u_3$ avec $|u_1| = i_j$ et $|u_1u_2| = i_k$ alors $u_1u_2^*u_3 \subseteq L$.*

Preuve: Sur l'automate qui reconnaît L. □

Application à L_1, L_2, L_3 et aux palindromes $L_4 = \{u \in \Sigma^* \mid u = \tilde{u}\}$.

1.8 Critères de reconnaissabilité

Le critère (2) est strictement plus fort que le critère (1) :
$K_1 = \{b^p a^n \mid p > 0$ et n est premier$\} \cup \{a\}^*$
satisfait (1) mais pas (2).

Le critère (3) est strictement plus fort que le critère (2) :
$K_2 = \{(ab)^n (cd)^n \mid n \geq 0\} \cup \Sigma^* \{aa, bb, cc, dd, ac\} \Sigma^*$
satisfait (2) mais pas (3).

Le critère (3) n'est pas suffisant :
$K_3 = \{udv \mid u, v \in \{a, b, c\}^*$ et soit $u \neq v$ soit u ou v contient un carré$\}$
satisfait (3) mais n'est pas reconnaissable.

1.8 Critères de reconnaissabilité

Pour montrer qu'un langage n'est pas reconnaissable, on peut aussi utiliser les propriétés de clôture.

Exemples : Sachant que L_1 n'est pas reconnaissable.

- $L_2 \cap a^*b^* = L_1$.
  Donc L_2 n'est pas reconnaissable.
- Soit $f : \Sigma^* \to \Sigma^*$ défini par $f(a) = aab$ et $f(b) = abb$.
  On a $f^{-1}(L_3) = L2$.
  Donc L_3 n'est pas reconnaissable.
- $L_5 = \{u \in \Sigma^* \mid |u|_a \neq |u|_b\} = \overline{L_2}$.
  Donc L_5 n'est pas reconnaissable.

1.9 Minimisation

Il y a une infinité d'automates pour un langage donné.

Exemple : automates D ou ND pour a^*.

Questions :

- Y a-t-il un automate canonique ?
- Y a-t-il unicité d'un automate minimal en nombre d'états ?
- Y a-t-il un lien structurel entre deux automates qui reconnaissent le même langage ?

1.9 Minimisation

1.9.1 Morphismes d'automates DC

Définition 1.21 *Soient* $\mathcal{A} = (Q, \delta, i, F)$ *et* $\mathcal{A}' = (Q', \delta', i', F')$ *deux automates déterministes complets. Une application* $\varphi : Q \to Q'$ *est un* morphisme *si*

- $\forall q \in Q, \forall a \in \Sigma, \varphi(\delta(q, a)) = \delta'(\varphi(q), a)$,
- $\varphi(i) = i'$,
- $\varphi^{-1}(F') = F$, *i.e.*, $q \in F \iff \varphi(q) \in F'$.

$$\begin{array}{ccc} Q & \xrightarrow{\delta_a} & Q \\ {\scriptstyle\varphi}\downarrow & & \downarrow{\scriptstyle\varphi} \\ Q' & \xrightarrow{\delta'_a} & Q' \end{array}$$

$\mathcal{A}$ et $\mathcal{A}'$ sont isomorphes s'il existe un morphisme bijectif de $\mathcal{A}$ vers $\mathcal{A}'$.

Remarque 1 : Deux automates DC sont isomorphes s'ils ne diffèrent que par le nom des états.
Remarque 2 : Si $\varphi : \mathcal{A} \to \mathcal{A}'$ est un morphisme bijectif, alors $\varphi^{-1} : \mathcal{A}' \to \mathcal{A}$ est aussi un morphisme.

1.9 Minimisation

Définition 1.22 *Soit $\mathcal{A}$ un automate DC. Une relation d'équivalence $\sim$ sur Q est une congruence si*

- $\forall p, q \in Q$, $\forall a \in \Sigma$, $p \sim q$ *implique* $\delta(p, a) \sim \delta(q, a)$,
- F *est saturé par* $\sim$, *i.e.*, $\forall p \in F$, $[p] = \{q \in Q \mid p \sim q\} \subseteq F$.

Le quotient de $\mathcal{A}$ par $\sim$ est $\mathcal{A}_\sim = (Q/\sim, \delta_\sim, [i], F/\sim)$ où $\delta_\sim$ est définie par $\delta_\sim([p], a) = [\delta(p, a)]$.

Proposition 1.23 *Soient $\mathcal{A}$ et $\mathcal{A}'$ deux automates DC. Il existe un morphisme surjectif $\varphi : \mathcal{A} \to \mathcal{A}'$ si et seulement si $\mathcal{A}'$ est isomorphe à un quotient de $\mathcal{A}$. Dans ce cas, on note $\mathcal{A}' \preceq \mathcal{A}$ et on a $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{A}')$.*

Remarque : $\preceq$ est un ordre partiel sur les automates DC.

But : Soit $L \in \mathrm{Rec}\Sigma^*$. Montrer qu'il existe un unique automate minimal pour $\preceq$ parmi les automates DC reconnaissant L.

1.9.2 Équivalence de Nérode

Définition 1.24 *Soit $\mathcal{A} = (Q, \delta, i, F)$ un automate DC.*
Pour $p \in D$, on note $\mathcal{L}(\mathcal{A}, p) = \{u \in \Sigma^* \mid \delta(p, u) \in F\}$.
L'équivalence de Nérode $\sim$ sur Q est définie par

$$p \sim q \quad \textit{ssi} \quad \mathcal{L}(\mathcal{A}, p) = \mathcal{L}(\mathcal{A}, q).$$

Remarque : On sait décider si $p \sim q$.

Proposition 1.25 *L'équivalence de Nérode est une congruence.*
L'automate quotient $\mathcal{A}_\sim$ est appelé automate de Nérode.
On a $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{A}_\sim)$ (Proposition 1.23)

On va voir que l'automate de Nérode est minimal (si $Q = \mathrm{Acc}(i)$).

Problème : comment le calculer *efficacement* ?

1.9 Minimisation

Pour $n \geq 0$, on note $\Sigma^{\leq n} = \Sigma^0 \cup \Sigma^1 \cup \cdots \cup \Sigma^n$ et on définit l'équivalence $\sim_n$ sur Q par

$$p \sim_n q \quad \text{ssi} \quad \mathcal{L}(\mathcal{A}, p) \cap \Sigma^{\leq n} = \mathcal{L}(\mathcal{A}, q) \cap \Sigma^{\leq n}.$$

Remarque 1 : $\sim_0$ a pour classes d'équivalence F et $Q \setminus F$.
Remarque 2 : $\sim_{n+1}$ est plus fine que $\sim_n$, i.e., $p \sim_{n+1} q \Longrightarrow p \sim_n q$.
Remarque 3 : $\sim = \bigcap_{n \geq 0} \sim_n$, i.e., $p \sim q$ ssi $\forall n \geq 0$, $p \sim_n q$.

Proposition 1.26

- $p \sim_{n+1} q$ *ssi* $p \sim_n q$ *et* $\forall a \in \Sigma$, $\delta(p, a) \sim_n \delta(q, a)$.
- *Si* $\sim_n = \sim_{n+1}$ *alors* $\sim = \sim_n$.
- $\sim = \sim_{|Q|-1}$, *et même* $\sim = \sim_{|Q|-2}$ *si* $|Q| \geq 2$.

On utilise la Proposition 1.26 pour calculer l'équivalence de Nérode par raffinements successifs.

1.9.3 Automate des résiduels

Définition 1.27 *Soient $u \in \Sigma^*$ et $L \subseteq \Sigma^*$. Le résiduel de L par u est le quotient $u^{-1}L = \{v \in \Sigma^* \mid uv \in L\}$.*

Définition 1.28 *Soit $L \subseteq \Sigma^*$. L'automate des résiduels de L est $\mathcal{R}(L) = (Q_L, \delta_L, i_L, F_L)$ défini par*

- $Q_L = \{u^{-1}L \mid u \in \Sigma^*\}$,
- $\delta_L(u^{-1}L, a) = a^{-1}(u^{-1}L) = (ua)^{-1}L$,
- $i_L = L = \varepsilon^{-1}L$,
- $F_L = \{u^{-1}L \mid \varepsilon \in u^{-1}L\} = \{u^{-1}L \mid u \in L\}$.

Théorème 1.29 *L est reconnaissable ssi L a un nombre fini de résiduels.*

1.9 Minimisation

Théorème 1.30 *Soit $\mathcal{A} = (Q, \delta, i, F)$ un automate DCA (déterministe, complet et accessible, i.e., $Q = \mathrm{Acc}(i)$) reconnaissant $L \subseteq \Sigma^*$.*
L'automate $\mathcal{R}(L)$ est isomorphe à l'automate de Nérode $\mathcal{A}_\sim$ de $\mathcal{A}$.

Corollaire 1.31 *Soit $L \in \mathrm{Rec}(\Sigma^*)$.*

1. *L'automate des résiduels de L est minimal pour l'ordre quotient ($\preceq$) parmi les automates DCA qui reconnaissent L.*
2. *Soit $\mathcal{A}$ un automate DC reconnaissant L avec un nombre minimal d'états. $\mathcal{A}$ est isomorphe à $\mathcal{R}(L)$.*
3. *On calcule l'automate minimal de L avec l'équivalence de Nérode à partir de n'importe quel automate DCA qui reconnaît L.*
4. *On peut décider de l'égalité de langages reconnaissables ($\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{B})$ avec $\mathcal{A}$ et $\mathcal{B}$ automates DCA) en testant* l'égalité *des automates minimaux associés ($\mathcal{A}_\sim = \mathcal{B}_\sim$).*

2 Langages algébriques

1. Grammaires
2. Arbres de dérivation
3. Formes normales et procédures de décision
4. Propriétés de clôture des langages algébriques
5. Lemme d'Ogden
6. Forme normale de Greibach
7. Problèmes indécidables sur les grammaires
8. Automates à pile
9. Langages déterministes
10. Application à l'analyse syntaxique

2.1 Grammaires

Définition 2.1 (Grammaire algébrique ou hors contexte)
$G = (\Sigma, V, P, S)$ *où*

- Σ *est l'alphabet terminal*
- V *est l'alphabet non terminal (variables)*
- $S \in V$ *est l'axiome (variable initiale)*
- $P \subseteq V \times (\Sigma \cup V)^*$ *est un ensemble* **fini** *de règles ou productions.*

Exemple 1 : expressions complètement parenthésées.
Exemple 2 : Expressions avec priorité.

Définition 2.2 (Dérivation) $\alpha \in (\Sigma \cup V)^*$ *se dérive en* $\beta \in (\Sigma \cup V)^*$, *noté* $\alpha \rightarrow \beta$, *s'il existe* $(x, \gamma) \in P$ *tel que* $\alpha = \alpha' x \alpha''$ *et* $\beta = \alpha' \gamma \alpha''$.
On note $\stackrel{*}{\rightarrow}$ *la clôture réflexive et transitive de* $\rightarrow$.

2.1 Grammaires

Définition 2.3 (Langage engendré)
Soit $G = (\Sigma, V, P, S)$ *une grammaire et* $\alpha \in (\Sigma \cup V)^*$.
Le langage engendré par α *est* $\mathcal{L}_G(\alpha) = \{u \in \Sigma^* \mid \alpha \xrightarrow{*} u\}$.
Le langage élargi *engendré par* α *est* $\widehat{\mathcal{L}}_G(\alpha) = \{\beta \in (\Sigma \cup V)^* \mid \alpha \xrightarrow{*} \beta\}$.
Le langage engendré par G *est* $L_G(S)$.
$L \subseteq \Sigma^*$ *est* algébrique *s'il existe une grammaire qui l'engendre.*

On note Alg *la famille des langages algébriques.*

Exemple : Le langage $\{a^n b^n \mid n \geq 0\}$ est algébrique.

2.1 Grammaires

Lemme 2.4 (fondamental)
Soit $G = (\Sigma, V, P, S)$ *une grammaire,* $\alpha_1, \alpha_2, \beta \in (\Sigma \cup V)^*$ *et* $n \geq 0$.

$$\alpha_1\alpha_2 \xrightarrow{n} \beta \quad \Longleftrightarrow \quad \alpha_1 \xrightarrow{n_1} \beta_1, \alpha_2 \xrightarrow{n_2} \beta_2 \text{ avec } \beta = \beta_1\beta_2 \text{ et } n = n_1 + n_2$$

Exercice (Langage de Dyck)
Soit $\Sigma = \{a_1, \ldots, a_n\} \cup \{\bar{a}_1, \ldots, \bar{a}_n\}$ l'alphabet formé de n paires de parenthèses.
Un mot $w \in \Sigma^*$ est *bien parenthésé* s'il est équivalent au mot vide dans la congruence engendrée par $a_i\bar{a}_i \equiv \varepsilon$ pour $1 \leq i \leq n$.
Montrer que le langage de Dyck $D_n^* = \{w \in \Sigma^* \mid w \equiv \varepsilon\}$ est engendré par la grammaire $S \to a_1 S \bar{a}_1 S + \cdots + a_n S \bar{a}_n S + \varepsilon$.

2.1 Grammaires

Définition 2.5 (Grammaire linéaire)
La grammaire $G = (\Sigma, V, P, S)$ *est*

- *linéaire si* $\forall (x, \alpha) \in P$, $\alpha \in \Sigma^* \cup \Sigma^* V \Sigma^*$
- *linéaire gauche si* $\forall (x, \alpha) \in P$, $\alpha \in \Sigma^* \cup V \Sigma^*$
- *linéaire droite si* $\forall (x, \alpha) \in P$, $\alpha \in \Sigma^* \cup \Sigma^* V$

Un langage est linéaire s'il peut être engendré par une grammaire linéaire.
On note Lin *la famille des langages linéaires.*

Exemple : Le langage $\{a^n b^n \mid n \geq 0\}$ est linéaire.

Proposition 2.6 *Un langage est rationnel si et seulement si il peut être engendré par une grammaire linéaire gauche (ou droite).*

2.2 Arbres de dérivation

Soit $A_p = \{a_1, \ldots, a_p\}$ un alphabet ordonné $a_1 \prec \cdots \prec a_p$.

Un arbre étiqueté dans Z et d'arité (au plus) p est une fonction partielle $T : A_p^* \to Z$ dont le domaine est un langage $\mathrm{dom}(T) \subseteq A_p^*$

- fermé par préfixe : $u \leq v$ et $v \in \mathrm{dom}(T)$ implique $u \in \mathrm{dom}(T)$,
- fermé par *frère aîné* : $a_i \prec a_j$ et $ua_j \in \mathrm{dom}(T)$ implique $ua_i \in \mathrm{dom}(T)$.

La racine de l'arbre est le mot vide $\varepsilon \in \mathrm{dom}(T)$.
Un nœud de l'arbre est un élément $u \in \mathrm{dom}(T)$.
L'arité d'un nœud $u \in \mathrm{dom}(T)$ est le plus grand entier k tel que $ua_k \in \mathrm{dom}(T)$ ($k = 0$ si $ua_1 \notin \mathrm{dom}(T)$).
Les fils d'un nœud $u \in \mathrm{dom}(T)$ d'arité k sont les nœuds $ua_1, \ldots, ua_k \in \mathrm{dom}(T)$.
Une feuille de l'arbre est un nœud d'arité 0.

2.2 Arbres de dérivation

Définition 2.7 *Soit $G = (\Sigma, V, P, S)$ une grammaire.*
Un arbre de dérivation pour G est un arbre T étiqueté dans $V \cup \Sigma$ tel que

- *chaque feuille est étiquetée par une variable ou un terminal,*
- *chaque nœud interne n est étiqueté par une variable x et si les fils de n portent les étiquettes $\alpha_1, \ldots, \alpha_k$ alors $(x, \alpha_1 \cdots \alpha_k) \in P$.*

La frontière de T est la concaténation des étiquettes des feuilles de T.

Exemple : arbres de dérivation pour les expressions avec priorité.

Proposition 2.8 *Soit $G = (\Sigma, V, P, S)$ une grammaire et $x \in V$.*
$\widehat{\mathcal{L}}_G(x)$ est l'ensemble des mots $\alpha \in (\Sigma \cup V^*)$ tels qu'il existe un arbre de dérivation de racine x est de frontière α.

2.2 Arbres de dérivation

- À chaque dérivation, on peut associer un arbre de dérivation.
- Si la grammaire est linéaire, il y a bijection entre dérivations et arbres de dérivations.
- 2 dérivations sont équivalentes si elles sont associées au même arbre de dérivation.
- Une dérivation est *gauche* si on dérive toujours le non terminal le plus à gauche.
- Il y a bijection entre dérivations gauches et arbres de dérivation.

2.2 Arbres de dérivation

- Une grammaire est ambiguë s'il existe deux arbres de dérivations (distincts) de même racine et de même frontière.
- Un langage algébrique est non ambigu s'il existe une grammaire non ambiguë qui l'engendre.
- La grammaire $S \longrightarrow SS + aSb + \varepsilon$ est ambiguë mais elle engendre un langage non ambigu.
- La grammaire $S \longrightarrow E + E \mid E \times E \mid a \mid b \mid c$ est ambiguë et engendre un langage rationnel.

Proposition 2.9 *Tout langage rationnel peut être engendré par une grammaire linéaire droite non ambiguë.*

2.3 Formes normales

Définition 2.10 (Grammaires réduites)
La grammaire $G = (\Sigma, V, P, S)$ est réduite si toute variable $x \in V$ est
- productive : $\mathcal{L}_G(x) \neq \emptyset$, i.e., $\exists\ x \xrightarrow{*} u \in \Sigma^*$, et
- accessible : il existe une dérivation $S \xrightarrow{*} \alpha x \beta$ avec $\alpha, \beta \in (\Sigma \cup V)^*$.

Lemme 2.11 *Soit $G = (\Sigma, V, P, S)$ une grammaire.*

1. *On peut calculer l'ensemble des variables productives de G.*
2. *On peut calculer l'ensemble des variables accessibles de G.*
3. *On peut décider si $\mathcal{L}_G(S) = \emptyset$.*

Corollaire 2.12 *Soit $G = (\Sigma, V, P, S)$ une grammaire telle que $\mathcal{L}_G(S) \neq \emptyset$. On peut effectivement restreindre G en une grammaire réduite équivalente $G' = (\Sigma, V', P', S)$ ($\mathcal{L}_G(S) = \mathcal{L}_{G'}(S)$).*

Restreindre aux variables productives, puis aux variables accessibles.

2.3 Formes normales

Définition 2.13 (Grammaires propres)
La grammaire $G = (\Sigma, V, P, S)$ est propre si elle ne contient pas de règle de la forme $x \to \varepsilon$ ou $x \to y$ avec $x, y \in V$.
Un langage $L \subseteq \Sigma^*$ est propre si $\varepsilon \notin L$.

Lemme 2.14 *Soit $G = (\Sigma, V, P, S)$ une grammaire. On peut calculer l'ensemble des variables x qui peuvent engendrer le mot vide.*

Proposition 2.15 *Soit $G = (\Sigma, V, P, S)$ une grammaire.*
On peut effectivement construire une grammaire propre G' qui engendre $\mathcal{L}_G(S) \setminus \{\varepsilon\}$.

La réduction d'une grammaire propre est une grammaire propre.

Corollaire 2.16 *On peut décider si un mot $u \in \Sigma^*$ est engendré par une grammaire G.*

2.3 Formes normales

Définition 2.17 (Forme normale de Chomsky)
Une grammaire $G = (\Sigma, V, P, S)$ est en forme normale de Chomsky

1. *faible si $P \subseteq V \times (V^* \cup \Sigma \cup \{\varepsilon\})$*
2. *forte si $P \subseteq V \times (V^2 \cup \Sigma \cup \{\varepsilon\})$*

Proposition 2.18 *Soit $G = (\Sigma, V, P, S)$ une grammaire.*
On peut effectivement construire une grammaire équivalente G' en forme normale de Chomsky faible ou forte.

La réduction d'une grammaire en FNC est encore en FNC.
La mise en FNC d'une grammaire propre est une grammaire propre.

Corollaire 2.19 *Soit $G = (\Sigma, V, P, S)$ une grammaire.*
On peut décider si $\mathcal{L}_G(S)$ est fini.

2.4 Propriétés de clôture

Proposition 2.20
La famille Alg *est fermée par union, concaténation, itération, miroir.*
La famille Lin *est fermée par union et miroir.*

Remarque : On verra que la famille Lin n'est pas fermée par concaténation et itération.

Proposition 2.21 *Les familles* Alg *et* Lin *sont fermées par morphisme.*

Une substitution $\sigma : A \to \mathcal{P}(B^*)$ est algébrique si $\forall a \in A$, $\sigma(a) \in \mathrm{Alg}$

Proposition 2.22 *La famille* Alg *est fermée par substitution algébrique.*

2.4 Propriétés de clôture

Proposition 2.23
Les familles Alg *et* Lin *sont fermées par intersection avec un rationnel.*

Remarque : On verra que les familles Alg et Lin ne sont pas fermées par intersection ou complémentaire.

Soit $B \subseteq A$ deux alphabets. La projection de A sur B est le morphisme $\pi : A^* \to B^*$ défini par $\pi(a) = \begin{cases} a & \text{si } a \in B \\ \varepsilon & \text{sinon.} \end{cases}$

Proposition 2.24
Les familles Alg *et* Lin *sont fermées par projection inverse.*
Les familles Alg *et* Lin *sont fermées par morphisme inverse.*

2.4 Propriétés de clôture

Définition 2.25 *Une transduction rationnelle (TR)* $\tau : A^* \to \mathcal{P}(B^*)$ *est la composée d'un morphisme inverse, d'une intersection avec un rationnel et d'un morphisme.*
Soient A, B, C *trois alphabets,* $K \in \mathrm{Rat}(C^*)$ *et* $\varphi : C^* \to A^*$ *et* $\psi : C^* \to B^*$ *deux morphismes. L'application* $\tau : A^* \to \mathcal{P}(B^*)$ *définie par* $\tau(a) = \psi(\varphi^{-1}(a) \cap K)$ *est une TR.*

$$\begin{array}{ccc} C^* & \xrightarrow{\cap K} & C^* \\ {\scriptstyle \varphi^{-1}}\uparrow & & \downarrow{\scriptstyle \psi} \\ A^* & \xrightarrow{\tau} & B^* \end{array}$$

Proposition 2.26 *Les familles* Alg *et* Lin *sont fermées par TR.*

2.4 Propriétés de clôture

Théorème 2.27 (Elgot et Mezei, 1965)
La composée de deux TR est encore une TR.

Théorème 2.28 (Nivat, 1968)
Une application $\tau : A^* \to \mathcal{P}(B^*)$ est une TR si et seulement si son graphe $\{(u, v) \mid v \in \tau(u)\}$ est une relation rationnelle (i.e., un langage rationnel sur l'alphabet $A \times B$).

Théorème 2.29 (Chomsky et Schützenberger)
Les propositions suivantes sont équivalentes :

1. *L est algébrique.*
2. *Il existe une TR τ telle que $L = \tau(D_2^*)$.*
3. *Il existe un entier n, un rationnel K et un morphisme alphabétique ψ tels que $L = \psi(D_n^* \cap K)$.*

2.5 Lemme d'Ogden

Comment montrer qu'un langage n'est pas algébrique ? n'est pas linéaire ? est ambigu ?

Lemme 2.30 (Ogden) *Soit $G = (\Sigma, V, P, S)$ une grammaire. Il existe un entier $N \in \mathbb{N}$ tel que pour tout $x \in V$ et $w \in \widehat{L}_G(x)$ contenant au moins N lettres distinguées, il existe $y \in V$ et $\alpha, u, \beta, v, \gamma \in (\Sigma \cup V^*)$ tels que*

- $w = \alpha u \beta v \gamma$,
- $x \xrightarrow{*} \alpha y \gamma$, $y \xrightarrow{*} uyv$, $y \xrightarrow{*} \beta$,
- $u\beta v$ *contient moins de* N *lettres distinguées,*
- *soit* α, u, β *soit* β, v, γ *contiennent des lettres distiguées.*

2.5 Lemme d'Ogden

Théorème 2.31 (Bar-Hillel, Perles, Shamir ou Lemme d'itération)

Soit $L \in \mathrm{Alg}$, *il existe* $N \geq 0$ *tel que pour tout* $w \in L$,
si $|w| \geq N$ *alors on peut trouver une factorisation* $w = \alpha u \beta v \gamma$ *avec* $|uv| > 0$ *et* $|u\beta v| < N$ *et* $\alpha u^n \beta v^n \gamma \in L$ *pour tout* $n \geq 0$.

Exemple : le langage $L_1 = \{a^n b^n c^n \mid n \geq 0\}$ n'est pas algébrique.

Corollaire 2.32 *Les familles* Alg *et* Lin *ne sont pas fermées par intersection ou complémentaire.*

2.5 Lemme d'Ogden

Exemple : le langage $L_2 = \{a^n b^n c^p d^p \mid n, p \geq 0\}$ est algébrique mais pas linéaire.

Corollaire 2.33 *La famille* Lin *n'est pas fermée par concaténation ou itération.*

Exemple : le langage $L_3 = \{a^n b^n c^p \mid n, p > 0\} \cup \{a^n b^p c^p \mid n, p > 0\}$ est linéaire et (inhéremment) ambigu.

Corollaire 2.34

1. *Les langages non ambigus ne sont pas fermés par union.*
2. *Les langages non ambigus ne sont pas fermés par morphisme.*

2.6 Forme normale de Greibach

Définition 2.35 *La grammaire $G = (\Sigma, V, P)$ est en*
FNG *(forme normale de Greibach) si* $P \subseteq V \times \Sigma V^*$
FNPG *(presque Greibach) si* $P \subseteq V \times \Sigma(V \cup \Sigma)^*$
FNGQ *(Greibach quadratique) si* $P \subseteq V \times (\Sigma \cup \Sigma V \cup \Sigma V^2)$

Remarque : on passe trivialement d'une FNPG à une FNG.

Théorème 2.36 *Soit $G = (\Sigma, V, P)$ une grammaire propre. On peut construire $G' = (\Sigma, V', P')$ en FNG équivalente à G, i.e., $V \subseteq V'$ et $\mathcal{L}_G(x) = \mathcal{L}_{G'}(x)$ pour tout $x \in V$.*

La difficulté est d'éliminer la récursivité gauche des règles.

2.6 Forme normale de Greibach

Lemme 2.37 *Soit $x \in V$ et $x \to x\alpha + \beta$ les règles issues de x:*
α ensemble fini de mots de $(V \cup \Sigma)^+$,
β ensemble fini de mots de $\Sigma(V \cup \Sigma)^* \cup (V \setminus \{x\})(V \cup \Sigma)^+$.
si on remplace les règles $x \to x\alpha + \beta$ par $x \to \beta + \beta x'$ et $x' \to \alpha + \alpha x'$, on obtient une grammaire équivalente à G.

On montre par récurrence sur $m \in \mathbb{N}$ que pour tout $y \in V$ et $w \in \Sigma^*$,

$$y \xrightarrow{m} w \text{ dans } G \quad \text{ssi} \quad y \xrightarrow{m} w \text{ dans } G'$$

Exemple :
$$x_1 \to x_1 b + a$$
$$x_2 \to x_1 b + a x_2$$

Exemple :
$$x_1 \to x_1(x_1 + x_2) + (x_2 a + b)$$
$$x_2 \to x_1 x_2 + x_2 x_1 + a$$

2.7 Problèmes indécidables sur les grammaires

Soient L, L' deux langages algébriques et R un langage rationnel.
Les problèmes suivants sont indécidables :

- $L \cap L' = \emptyset$?
- $L = \Sigma^*$?
- $L = L'$?
- $L \subseteq L'$?
- $R \subseteq L$?
- $L \subseteq R$?
- L est-il rationnel ?
- L est-il ambigu ?
- $\overline{L}$ est-il algébrique ?
- $L \cap L'$ est-il algébrique ?

2.8 Automates à pile

Définition 2.38 (Automate à pile)
$\mathcal{A} = (Q, \Sigma, Z, T, q_0, z_0, K)$ *où*

- Q *ensemble fini d'états*
- Σ *alphabet d'entrée*
- Z *alphabet de pile*
- $T \subseteq Q \times Z \times (\Sigma \cup \{\varepsilon\}) \times Q \times Z^*$ *ensemble fini de transitions*
- $(q_0, z_0) \in Q \times Z$ *configuration initiale*
- $K \subseteq Q \times Z^*$ *configurations acceptantes:*
  - $K = Q \times \{\varepsilon\}$ *par pile vide*
  - $K = F \times Z^*$ *par état final*
  - $K = Q \times Z^* Z'$ *par sommet de pile*
  - $K = F \times \{\varepsilon\}$ *par état final et pile vide*

2.8 Automates à pile

La sémantique est donnée par un système de transitions (automate) infini $\mathcal{A}' = (Q \times Z^*, T', (q_0, z_0), K)$ avec

$$T' = \{(p, hz) \xrightarrow{x} (q, hu) \mid (p, z, x, q, u) \in T\}$$

Un état $(p, h) \in Q \times Z^*$ est appelé configuration de $\mathcal{A}$.

Un mot $w \in \Sigma^*$ est accepté par $\mathcal{A}$ s'il existe un calcul acceptant pour w dans A':

$$(q_0, z_0) \xrightarrow{w} (q, h) \in K$$

On note $\mathcal{L}(\mathcal{A})$ le langage reconnu par $\mathcal{A}$.

2.8 Automates à pile

Proposition 2.39 *Soit $\mathcal{A} = (Q, \Sigma, Z, T, q_0, z_0, F \times \{\varepsilon\})$ un automate à pile reconnaissant par pile vide et état final. On peut construire une grammaire G qui engendre $\mathcal{L}(\mathcal{A})$.*

Si $\mathcal{A}$ est *temps-réel* (pas d'ε-transition) alors G est en FNG.

Proposition 2.40 *Soit $G = (\Sigma, V, P, S)$ une grammaire en FNG. On peut construire un automate à pile $\mathcal{A}$ temps-réel et simple (un seul état) qui accepte $L_G(S)$ par pile vide.*

Si G est en FNGQ alors $\mathcal{A}$ est standardisé ($T \subseteq Z \times \Sigma \times Z^{\leq 2}$).

Si G n'est pas en FNG ($P \subseteq V \times (\Sigma \cup \{\varepsilon\})V^*$) l'automate $\mathcal{A}$ obtenu n'est pas nécessairement temps-réel.

2.8 Automates à pile

Définition 2.41

$\mathcal{A} = (Q, \Sigma, Z, T, q_0, z_0, K)$ *est* à fond de pile testable *si*

$Z = Z_1 \uplus Z_2$, $z_0 \in Z_1$ *et*

$\forall (p, z, a, q, h) \in T,\ z \in Z_1 \Rightarrow h \in Z_1 Z_2^* \cup \{\varepsilon\}$ *et* $z \in Z_2 \Rightarrow h \in Z_2^*$

Exercices

1. Soit $\mathcal{A}$ un automate à pile. Montrer qu'on peut construire un automate à pile $\mathcal{B}$ à fond de pile testable et avec même mode d'acceptation tel que $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{B})$.
2. Montrer que l'acceptance par état final a la même puissance d'expression que l'acceptance par pile vide.

2.9 Langages déterministes

Définition 2.42 (Automate à pile déterministe)
$\mathcal{A} = (Q, \Sigma, Z, T, q_0, z_0, K)$ *est* déterministe *si*

- $\forall (p, z, a) \in Q \times Z \times (\Sigma \cup \{\varepsilon\}), \quad |T(p, z, a)| \leq 1,$
- $\forall (p, z, a) \in Q \times Z \times \Sigma, \quad T(p, z, \varepsilon) \neq \emptyset \Longrightarrow T(p, z, a) = \emptyset$

Un langage $L \subseteq \Sigma^*$ *est* déterministe *s'il existe un automate à pile déterministe qui accepte* L *par état final.*

Exemples :

- D_n^* peut être accepté par état final par un automate D+TR mais pas par un automate D+TR+S.
- Un langage accepté par état final par un automate *simple* est fermé par préfixe.
- D_n^* peut être accepté par sommet de pile par un automate D+TR+S.

2.9 Langages déterministes

- $\{a^n b a^n \mid n \geq 0\}$ peut être accepté par état final par un automate D+TR mais pas par un automate D+TR+S.
- $\{a^n b a^n \mid n \geq 0\}$ peut être accepté par pile vide par un automate D+TR+S.
- Le langage $\{a^n b^p c a^n \mid n, p > 0\} \cup \{a^n b^p d b^p \mid n, p > 0\}$ est déterministe mais pas D+TR.

Remarque : Pour les automates à pile déterministes, l'acceptation par pile vide a le même pouvoir d'expression que l'acceptation par pile vide et état final, mais est strictement plus faible que l'acceptation par état final.

Proposition 2.43
Un langage L est déterministe et préfixe ($L \cap L\Sigma^+ = \emptyset$) ssi il existe un automate déterministe qui accepte L par pile vide.

2.9 Langages déterministes

Soit $\mathcal{A} = (Q, \Sigma, Z, T, q_0, z_0, K)$ un automate à pile déterministe, on peut effectivement calculer les ensembles suivants :

1. $X = \{(p, x, q) \in Q \times Z \times Q \mid (p, x) \xrightarrow[*]{\varepsilon} (q, \varepsilon)\}$
2. $Y = \{(p, x) \in Q \times Z \mid (p, x) \xrightarrow[\omega]{\varepsilon}\}$
3. $V = \{(p, x, q) \in Q \times Z \times Q \mid \exists\ (p, x) \xrightarrow[*]{\varepsilon} (q, h)\}$
4. $W = \{(p, x, q, y) \in Q \times Z \times Q \times Z \mid (p, x) \xrightarrow[*]{\varepsilon} (q, y)\}$

2.9 Langages déterministes

Un automate à pile $\mathcal{A} = (Q, \Sigma, Z, T, q_0, z_0, K)$ est complet si

1. aucun calcul à partir de (q_0, z_0) ne vide la pile,
2. pour tout $(p, x) \in Q \times Z$, il n'y a pas de calcul infini $(p, x) \xrightarrow[\omega]{\varepsilon}$,
3. pour tout $(p, x) \in Q \times Z$, soit $(p, x) \xrightarrow{\varepsilon}$ soit $\forall a \in \Sigma$, $(p, x) \xrightarrow{a}$.

Remarques :

1. Un automate à pile complet est *non bloquant*, i.e., pour toute configuration accessible (p, h) et toute lettre $a \in \Sigma$ il existe un calcul $(p, h) \xrightarrow[*]{\varepsilon} \xrightarrow{a} (p', h')$.
2. Avec un automate à pile déterministe complet, chaque mot a un unique calcul maximal (et fini).
3. On peut (facilement) décider si un mot est accepté par un automate déterministe complet.

2.9 Langages déterministes

Proposition 2.44 *Soit $\mathcal{A} = (Q, \Sigma, Z, T, q_0, z_0, F)$ un automate à pile déterministe qui reconnaît par état final ($F \subseteq Q$), on peut effectivement construire un automate à pile* complet *$\mathcal{A}' = (Q', \Sigma, Z', T', q'_0, z'_0, F')$ qui reconnaît le même langage.*

- $Q' = Q \uplus \{q'_0, d, f\}$, $F' = F \uplus \{f\}$, $Z' = Z \uplus \{\bot\}$, $z'_0 = \bot$ et
- $(q'_0, \bot) \xrightarrow{\varepsilon} (q_0, \bot z_0)$, et $(p, \bot) \xrightarrow{a} (d, \bot)$ pour $p \in Q$ et $a \in \Sigma$,
- $(d, x) \xrightarrow{a} (d, x)$ et $(f, x) \xrightarrow{a} (d, x)$ pour $x \in Z'$ et $a \in \Sigma$,
- Si $(p, x) \xrightarrow{a} (q, \alpha) \in T$ et $a \in \Sigma$ alors $(p, x) \xrightarrow{a} (q, \alpha) \in T'$,
- Si dans $\mathcal{A}$ on a $(p, x) \not\xrightarrow{a}$, $a \in \Sigma$ et $(p, x) \not\xrightarrow{\varepsilon}$ alors $(p, x) \xrightarrow{a} (d, x) \in T'$,
- Si $(p, x) \not\xrightarrow[\omega]{\varepsilon}$ et $(p, x) \xrightarrow{\varepsilon} (q, \alpha) \in T$ alors $(p, x) \xrightarrow{\varepsilon} (q, \alpha) \in T'$,
- Si $(p, x) \xrightarrow[\omega]{\varepsilon}$ et $(p, x) \xrightarrow[*]{\varepsilon} (q, \alpha)$ avec $q \in F$ alors $(p, x) \xrightarrow{\varepsilon} (f, x) \in T'$,
- Si $(p, x) \xrightarrow[\omega]{\varepsilon}$ et $(p, x) \xrightarrow[*]{\varepsilon} (q, \alpha) \Longrightarrow q \notin F$ alors $(p, x) \xrightarrow{\varepsilon} (d, x) \in T'$.

2.9 Langages déterministes

Proposition 2.45 *Soit $\mathcal{A} = (Q, \Sigma, Z, T, q_0, z_0, F)$ un automate à pile déterministe qui reconnaît par état final ($F \subseteq Q$), on peut effectivement construire un automate à pile déterministe $\mathcal{A}'$ qui reconnaît $\Sigma^* \setminus \mathcal{L}(\mathcal{A})$.*

Proposition 2.46 *Soit $\mathcal{A} = (Q, \Sigma, Z, T, q_0, z_0, F)$ un automate à pile déterministe qui reconnaît par état final ($F \subseteq Q$), on peut effectivement construire un automate à pile déterministe équivalent $\mathcal{A}'$ tel qu'on ne puisse pas faire d'ε-transition à partir d'un état final de $\mathcal{A}'$.*

Proposition 2.47 *Tout langage déterministe est non ambigu.*

2.9 Langages déterministes

Lemme 2.48 *Soit $\mathcal{A} = (Q, \Sigma, Z, T, q_0, z_0, K)$ un automate à pile déterministe reconnaissant par sommet de pile et état final (une configuration $(q, \alpha z)$ est acceptante si $(q, z) \in K \subseteq Q \times Z$).*
On peut effectivement construire un automate à pile déterministe équivalent reconnaissant par état final.

Proposition 2.49 *Soit $\mathcal{A}$ un automate à pile déterministe, on peut effectivement construire un automate à pile déterministe qui reconnaît le même langage et dont les ε-transitions sont uniquement effaçantes : $(p, x) \xrightarrow{\varepsilon} (q, \varepsilon)$.*

Corollaire 2.50 *On peut (rapidement) décider si un mot est reconnu par un tel automate à pile déterministe.*

2.9 Langages déterministes

Soient L, L' deux langages déterministes et R un langage rationnel.

Les problèmes suivants sont décidables :

- $L = R$?
- $R \subseteq L$?
- L est-il rationnel ?
- $L = L'$?

Les problèmes suivants sont indécidables :

- $L \cap L' = \emptyset$?
- $L \subseteq L'$?
- $L \cap L'$ est-il algébrique ?
- $L \cap L'$ est-il déterministe ?
- $L \cup L'$ est-il déterministe ?

Enfin, on ne sait pas décider si un langage algébrique est déterministe.

2.10 Application à l'analyse syntaxique

Buts :

- Savoir si un programme est syntaxiquement correct.
- Construire l'arbre de dérivation pour piloter la génération du code.

Formalisation :

Un programme est un mot $w \in \Sigma^*$ (Σ est l'alphabet ASCII).
L'ensemble des programmes syntaxiquement corrects forme un langage $L \subseteq \Sigma^*$.
Ce langage est algébrique : la syntaxe du langage de programmation est définie par une grammaire $G = (\Sigma, V, P, S)$.

Pour tester si un programme w est syntaxiquement correct, il faut résoudre le problème du mot : est-ce que $w \in \mathcal{L}_G(S)$?

2.10 Application à l'analyse syntaxique

On sait décider si $w \in \mathcal{L}_G(S)$

• en testant toutes les dériviations de longueur au plus $2|w|$ si la grammaire est propre.

• en lisant le mot si on a un automate à pile déterministe complet. Ceci se fait en temps linéaire par rapport à $|w|$ si l'automate est temps réel ou si les ε-transitions ne font que dépiler.

Le problème est qu'un langage de programmation peut être non déterministe ou ambigu.

2.10 Application à l'analyse syntaxique

Analyse descendante (LL) : Soit $G = (\Sigma, V, P, S)$ une grammaire. On construit l'automate à pile simple non déterministe qui accepte par pile vide : $\mathcal{A} = (\Sigma, \Sigma \cup V, T, S)$ où les transitions de T sont des

- expansions : $\{(x, \varepsilon, \tilde{\alpha}) \mid (x, \alpha) \in P\}$ ou
- vérifications : $\{(a, a, \varepsilon) \mid a \in \Sigma\}$.

Exemple 1 : $S \longrightarrow aSb + ab$.

Exemple 2 :
$$\begin{cases} E & \longrightarrow & E + T \mid T \\ T & \longrightarrow & T * F \mid F \\ F & \longrightarrow & (E) \mid a \mid b \mid c \end{cases}$$

Analyse LL :
$$\begin{cases} \text{L : le mot est lu de gauche à droite.} \\ \text{L : on construit une dérivation gauche.} \end{cases}$$

2.10 Application à l'analyse syntaxique

Si $\mathcal{A}$ est déterministe, la grammaire est LL(1).

Plus généralement, une grammaire est LL(k) si on peut lever le non déterminisme de l'automate en regardant les k prochaines lettres du mot.

La grammaire de l'exemple 1 est LL(2).

La grammaire de l'exemple 2 n'est pas LL(k).

On peut la transformer en une grammaire LL(1) équivalente.

$$\begin{cases} E \to TE' & E' \to +TE' \mid \varepsilon \\ T \to FT' & T' \to *FT' \mid \varepsilon \\ F \to (E) \mid a \mid b \mid c & \end{cases}$$

Cette grammaire admet un automate à pile déterministe simple.

2.10 Application à l'analyse syntaxique

Analyse ascendante (LR) : Soit $G = (\Sigma, V, P, S)$ une grammaire.
On construit un automate à pile généralisé simple non déterministe.
Initialement la pile est vide.
Transitions généralisées : $T \subseteq (\Sigma \cup V)^* \times (\Sigma \cup \{\varepsilon\}) \times (\Sigma \cup V)$

- décalages (shift) : $\{(\varepsilon, a, a) \mid a \in \Sigma\}$ ou
- réductions (reduce) : $\{(\alpha, \varepsilon, x) \mid (x, \alpha) \in P\}$.

L'automate accepte lorsque la pile contient uniquement le symbole S.

Exemple 1 : $S \rightarrow aSb + ab.$

Exemple 2 : $E \rightarrow E + T \mid T \quad , \quad T \rightarrow T * F \mid F \quad , \quad F \rightarrow (E) \mid a \mid b \mid c$

Analyse LR : $\begin{cases} \text{L : le mot est lu de gauche à droite.} \\ \text{R : on construit une dérivation droite.} \end{cases}$

2.10 Application à l'analyse syntaxique

Automate pour la grammaire de l'exemple 2 :

1.	x	$\xrightarrow{\varepsilon}$	F	avec $x \in \{a, b, c\}$	reduce
2.	(E)	$\xrightarrow{\varepsilon}$	F		reduce
3.	$T * F$	$\xrightarrow{\varepsilon}$	T		reduce
4.	F	$\xrightarrow{\varepsilon}$	T		reduce
5.	$E + T$	$\xrightarrow{\varepsilon}$	E		reduce
6.	T	$\xrightarrow{\varepsilon}$	E		reduce
7.	T	$\xrightarrow{*}$	$T*$		shift
8.	E	$\xrightarrow{y}$	Ey	avec $y \in \{+,)\}$	shift
9.	x	$\xrightarrow{y}$	xy	avec $x \in \{\perp, (, +, *\}$ et $y \in \{a, b, c, (\}$	shift

Conflits : (3,4) : on choisit 3. (5,6) : on choisit 5.
(5,7) : on choisit 7. (6,7) : on choisit 7.

2.10 Application à l'analyse syntaxique

Une grammaire est LR(k) si on peut lever les conflits shift/reduce et reduce/reduce en regardant les k prochaines lettres du mot.

La grammaire de l'exemple 1 est LR(0) (il faut réduire dès que possible).

La grammaire de l'exemple 2 est LR(1).

2.10 Application à l'analyse syntaxique

La grammaire de l'exemple 2 est non ambiguë : lors d'un conflit, si on fait le mauvais choix, on ne peut pas prolonger en un calcul acceptant.

Exemple 3 : $E \longrightarrow E + E \mid E * E \mid (E) \mid a \mid b \mid c$.

Cette grammaire est ambiguë mais admet un automate LR(1):

1.	x	$\xrightarrow{\varepsilon}$	E	avec $x \in \{a, b, c\}$	reduce
2.	(E)	$\xrightarrow{\varepsilon}$	E		reduce
3.	$E * E$	$\xrightarrow{\varepsilon}$	E		reduce
4.	$E + E$	$\xrightarrow{\varepsilon}$	E		reduce
5.	E	$\xrightarrow{y}$	Ey	avec $y \in \{+, *,)\}$	shift
6.	x	$\xrightarrow{y}$	xy	avec $x \in \{\perp, (, +, *\}$ et $y \in \{a, b, c, (\}$	shift

La résolution des conflits (3,5) et (4,5) fixe l'associativité et la priorité des opérateurs.

2.10 Application à l'analyse syntaxique

3.	$E * E$	$\xrightarrow{\varepsilon}$	E		reduce
4.	$E + E$	$\xrightarrow{\varepsilon}$	E		reduce
5.	E	$\xrightarrow{y}$	Ey	avec $y \in \{+, *,)\}$	shift

Résolution des conflits.

Sommet de pile	prochain symbole	action	règle
$E + E$	$+$	reduce	associativité gauche de $+$
$E + E$	$*$	shift	priorité de $*$ sur $+$
$E + E$	$)$	reduce	
$E * E$	$+$	reduce	priorité de $*$ sur $+$
$E * E$	$*$	reduce	associativité gauche de $*$
$E * E$	$)$	reduce	

2.10 Application à l'analyse syntaxique

L'instruction if then else présente aussi une ambiguïté classique.

Considérons la grammaire

$$I \rightarrow \text{if } C \text{ then } I \text{ else } I \mid \text{if } C \text{ then } I \mid A$$

Le mot if C then if C then A else A admet deux arbres de dérivation.

L'automate LR présente un conflit shift/reduce.

On choisit le shift.

Règle : un else se rapporte au dernier if qui n'a pas de else.