Langages formels

Paul Gastin

Paul.Gastin@lsv.ens-cachan.fr

http://www.lsv.ens-cachan.fr/~gastin/Langages/

L3 Informatique Cachan
2015-2016

Plan

1. Introduction

Langages reconnaissables

Grammaires

Langages algébriques

Automates à pile

Analyse syntaxique

Automates d'arbres

Fonctions séquentielles

Motivations

Définition :

1. Description et analyse (lexicale et syntaxique) des langages (programmation, naturels, . . .)
2. Modèles de calcul
3. 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.
4. Types de données

Références

[7] Olivier Carton.
Langages formels, calculabilité et complexité.
Vuibert, 2008.

[9] John E. Hopcroft et Jeffrey D. Ullman.
Introduction to automata theory, languages and computation.
Addison-Wesley, 1979.

[10] Dexter C. Kozen.
Automata and Computability.
Springer, 1997.

[13] Jacques Sakarovitch.
Éléments de théorie des automates.
Vuibert informatique, 2003.

[8] Hubert Comon, Max Dauchet, Remi Gilleron, Florent Jacquemard, Denis Lugiez, Sophie Tison, Marc Tommasi.
Tree Automata Techniques and Applications.
http://www.grappa.univ-lille3.fr/tata/

Références

[1] Alfred V. Aho, Ravi Sethi et Jeffrey D. Ullman.
Compilers: principles, techniques and tools.
Addison-Wesley, 1986.

[2] Alfred V. Aho et Jeffrey D. Ullman.
The theory of parsing, translation, and compiling. Volume I: Parsing.
Prentice-Hall, 1972.

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

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

Références

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

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

[11] Jean-Éric Pin.
Automates finis et applications.
Polycopié du cours à l'École Polytechnique, 2004.

[12] Grzegorz Rozenberg et Arto Salomaa, éditeurs.
Handbook of Formal Languages,
Vol. 1, Word, Language, Grammar,
Springer, 1997.

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

Plan

Bibliographie

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

[7] Olivier Carton.
Langages formels, calculabilité et complexité.
Vuibert, 2008.

[9] John E. Hopcroft et Jeffrey D. Ullman.
Introduction to automata theory, languages and computation.
Addison-Wesley, 1979.

[10] Dexter C. Kozen.
Automata and Computability.
Springer, 1997.

[13] Jacques Sakarovitch.
Éléments de théorie des automates.
Vuibert informatique, 2003.

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.

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 $v = v_0 u_1 v_1 u_2 \cdots u_n v_n$ avec $u_i, v_i \in \Sigma^*$ et $u = u_1 u_2 \cdots u_n$

Théorème : 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)

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 par rapport à l'union.

$|K \cdot L| \leq |K| \cdot |L|$

notion de multiplicité, d'ambiguïté

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

Automates déterministes

Définition : 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 $a \in \Sigma$.

Automates déterministes

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

Définition : 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^*.

Automates non déterministes

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

Définition : 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 $0 < i \leq n$.
Langage accepté (reconnu) par $\mathcal{A}$:

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

Automates non déterministes

Théorème : 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*.

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.

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 :

Soit $\mathcal{A}$ un automate.

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

Automates avec ε-transitions

Exemple.

Définition : 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.

Décision

Presque tout est décidable sur les langages reconnaissables donnés par des automates.

Définition :

Problème du vide : étant donné un automate fini $\mathcal{A}$, décider si $\mathcal{L}(\mathcal{A}) = \emptyset$.

Problème du mot : étant donnés un mot $w \in \Sigma^*$ et un automate $\mathcal{A}$, décider si $w \in \mathcal{L}(\mathcal{A})$.

Théorème : vide et mot

Le problème du vide et le problème du mot sont décidables en **NLOGSPACE** pour les langages reconnaissables donnés par automates (déterministe ou non, avec ou sans ε-transitions).

Preuve

C'est de l'accessibilité.

Propriétés de fermeture

Opérations ensemblistes

Proposition :

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 :

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 $L_1 \subseteq L_2$.

Propriétés de fermeture

Opérations liées à la concaténation

Proposition :

$\mathrm{Rec}(\Sigma^*)$ 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.

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 :

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

Preuve

Modification des états initiaux et/ou finaux.

Propriétés de fermeture

Proposition :

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.

Propriétés de fermeture

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 :

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.

Propriétés de fermeture

Définition : 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 $\sigma : A \to \mathrm{Rec}(B^*)$.

Propriétés de fermeture

Proposition :

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.

Langages rationnels

Syntaxe pour représenter des langages.

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

Définition : 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.

Langages rationnels

Définition : 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.

Langages rationnels

Définition :

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.

Langages rationnels

Théorème : 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 :

L'équivalence des expressions rationnelles est décidable.

Preuve

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

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).

Critères de reconnaissabilité

Lemme : itération

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

1. si $|x| \geq N$ alors $\exists u_1, u_2, u_3 \in \Sigma^*$ tels que $x = u_1 u_2 u_3$, $u_2 \neq \varepsilon$ et $u_1 u_2^* u_3 \subseteq L$.
2. si $x = w_1 w_2 w_3$ avec $|w_2| \geq N$ alors $\exists u_1, u_2, u_3 \in \Sigma^*$ tels que $w_2 = u_1 u_2 u_3$, $u_2 \neq \varepsilon$ et $w_1 u_1 u_2^* u_3 w_3 \subseteq L$.
3. si $x = u v_1 v_2 \dots v_N w$ avec $|v_i| \geq 1$ alors il existe $0 \leq j < k \leq N$ tels que $u v_1 \dots v_j (v_{j+1} \dots v_k)^* v_{k+1} \dots v_N w \subseteq L$.

Preuve

Sur l'automate qui reconnaît L.

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

Critères de reconnaissabilité

Exercice : Puissance des lemmes d'itérations

1. Montrer que les langages suivants satisfont (1) mais pas (2) :

$$K_1 = \{w \in \{a,b\}^* \mid |w|_a = |w|_b\}$$

$$K_1' = \{b^p a^n \mid p > 0 \text{ et } n \text{ est premier}\} \cup \{a\}^*$$

2. Montrer que le langage suivant satisfait (2) mais pas (3) :

$$K_2 = \{(ab)^n(cd)^n \mid n \geq 0\} \cup \Sigma^*\{aa, bb, cc, dd, ac\}\Sigma^*$$

3. Montrer que le langage suivant satisfait (3) mais n'est pas reconnaissable :

$$K_3 = \{udv \mid u, v \in \{a,b,c\}^* \text{ et soit } u \neq v \text{ soit } u \text{ ou } v \text{ contient un carré}\}$$

Critères de reconnaissabilité

Théorème : Ehrenfeucht, Parikh, Rozenberg ([13, p. 128])

Soit $L \subseteq \Sigma^*$. Les conditions suivantes sont équivalentes :

1. L est reconnaissable
2. Il existe $N > 0$ tel que pour tout mot $x = uv_1 \dots v_N w \in \Sigma^*$ avec $|v_i| \geq 1$, il existe $0 \leq j < k \leq N$ tels que pour tout $n \geq 0$,

$$x \in L \quad \text{ssi} \quad uv_1 \dots v_j (v_{j+1} \dots v_k)^n v_{k+1} \dots v_N w \in L$$

3. Il existe $N > 0$ tel que pour tout mot $x = uv_1 \dots v_N w \in \Sigma^*$ avec $|v_i| \geq 1$, il existe $0 \leq j < k \leq N$ tels que

$$x \in L \quad \text{ssi} \quad uv_1 \dots v_j v_{k+1} \dots v_N w \in L$$

Remarque : la preuve utilise le théorème de Ramsey.

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 $f^{-1}(L_3) = L_2$.
  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.

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 ?

Automate des résiduels

Définition : Résiduels

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

Exemple : Calculer les résiduels des langages

$L_j = \{u = u_0 u_1 \cdots u_n \in \{0,1\}^* \mid \overline{u}^2 = \sum_{i=0}^{n} u_i 2^i \equiv j\ [3]\}$.

Définition : Automate des résiduels

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

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

Théorème :

Un langage $L \subseteq \Sigma^*$ est reconnaissable ssi L a un nombre fini de résiduels.

Congruences et quotients

Définition : Congruence sur les automates

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 $\delta_\sim([p], a) = [\delta(p, a)]$.

Exemple :

Donner un automate DCA $\mathcal{A}$ à 6 états qui 'calcule' $\overline{u}^2 \mod 3$ et accepte L_1.
Exhiber une congruence non triviale sur $\mathcal{A}$.
Calculer le quotient $\mathcal{A}/{\sim}$.

Proposition :

$\mathcal{A}/{\sim}$ est bien défini et $\mathcal{L}(\mathcal{A}/{\sim}) = \mathcal{L}(\mathcal{A})$.

Équivalence de Nerode

Définition : Équivalence de Nerode

Soit $\mathcal{A} = (Q, \delta, i, F)$ un automate DCA (DC et accessible) reconnaissant L.

Pour $q \in Q$, on note $\mathcal{L}(\mathcal{A}, q) = \{u \in \Sigma^* \mid \delta(q, u) \in F\}$.

L'équivalence de Nerode de $\mathcal{A}$ est définie par $p \sim q$ si $\mathcal{L}(\mathcal{A}, p) = \mathcal{L}(\mathcal{A}, q)$.

Proposition :

L'équivalence de Nerode est une congruence.

L'automate $\mathcal{A}/\!\sim$ est appelé quotient de Nerode de $\mathcal{A}$.

Théorème : $\mathcal{A}/\!\sim\; = \mathcal{R}(L)$

Le quotient de Nerode est isomorphe à l'automate des résiduels.

$\varphi\colon Q/\!\sim\; \to Q_L$ définie par $\varphi([q]) = \mathcal{L}(\mathcal{A}, q)$ est un isomorphisme de $\mathcal{A}/\!\sim$ sur $\mathcal{R}(L)$.

Automate minimal

Théorème :

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

1. Si $\mathcal{A}$ est un automate DCA qui reconnaît L, alors $\mathcal{R}(L)$ est un quotient de $\mathcal{A}$.
2. $\mathcal{R}(L)$ est minimal parmi les automates DCA reconnaissant L.
   (minimal en nombre d'états et minimal pour l'ordre quotient)
3. Soit $\mathcal{A}$ un automate DC reconnaissant L avec un nombre minimal d'états.
   $\mathcal{A}$ est isomorphe à $\mathcal{R}(L)$ (unicité de l'automate minimal)

Corollaire :

1. On obtient l'automate minimal de L en calculant le quotient de Nerode de n'importe quel automate DCA qui reconnaît L.
2. Soient $\mathcal{A}$ et $\mathcal{B}$ deux automates DCA. Pour tester si $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{B})$:
   Calculer les quotients de Nerode puis tester leur égalité : $\mathcal{A}/\sim = \mathcal{B}/\sim$.

Problème : comment calculer le quotient de Nerode *efficacement* ?

Algorithme de Moore

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

$$\begin{array}{lll} p \sim_n q & \text{ssi} & \mathcal{L}(\mathcal{A}, p) \cap \Sigma^{\leq n} = \mathcal{L}(\mathcal{A}, q) \cap \Sigma^{\leq n} \\ & \text{ssi} & \forall w \in \Sigma^{\leq n}, \quad \delta(p, w) \in F \iff \delta(q, w) \in F \end{array}$$

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 : Soit $\mathcal{A}$ automate DC

- $p \sim_{n+1} q$ 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 $\sim = \sim_n$.
- $\sim = \sim_{|Q|-2}$ si $\emptyset \neq F \neq Q$ et $\sim = \sim_0$ sinon.

Permet de calculer l'équivalence de Nerode par raffinements successifs.

Exercice :

Calculer l'automate minimal par l'algorithme d'Hopcroft de raffinement de partitions en $\mathcal{O}(n \log(n))$ (l'algo naïf est en $\mathcal{O}(n^2)$ avec $n = |Q|$).

Logique sur les mots

Définition : Syntaxe de $\mathrm{MSO}(\Sigma, <)$

$$\varphi ::= \bot \mid P_a(x) \mid x < y \mid x \in X \mid \neg\varphi \mid \varphi \vee \varphi \mid \exists x\, \varphi \mid \exists X\, \varphi$$

avec $a \in \Sigma$, $\{x, y, \ldots\}$ variables du premier ordre, {X,Y,...} variables monadiques du second ordre.

Définition : Sémantique de $\mathrm{MSO}(\Sigma, <)$

Soit $w = w_1 w_2 \cdots w_n \in \Sigma^+$ un mot et $\mathrm{pos}(w) = \{1, 2, \ldots, n\}$ les positions du mot.
Soit σ une valuation :
$\sigma(x) \in \mathrm{pos}(w)$ si x est une variable du premier ordre et
$\sigma(X) \subseteq \mathrm{pos}(w)$ si X est une variable monadique du second ordre.

$w, \sigma \models P_a(x)$	si	$w_{\sigma(x)} = a$
$w, \sigma \models x < y$	si	$\sigma(x) < \sigma(y)$
$w, \sigma \models x \in X$	si	$\sigma(x) \in \sigma(X)$
$w, \sigma \models \exists x\, \varphi$	si	$\exists\, i \in \mathrm{pos}(w)$ tel que $w, \sigma[x \mapsto i] \models \varphi$
$w, \sigma \models \exists X\, \varphi$	si	$\exists\, I \subseteq \mathrm{pos}(w)$ tel que $w, \sigma[X \mapsto I] \models \varphi$

Logique sur les mots

Définition : Codage d'une valuation dans l'alphabet

Soit V un ensemble de variables, on note $\Sigma_V = \Sigma \times \{0,1\}^V$.
Un couple (w, σ) où $w = w_1 w_2 \cdots w_n \in \Sigma^+$ est un mot sur l'alphabet Σ et σ est une valuation des variables de V est codé par un mot $W = (w_1, \tau_1) \cdots (w_n, \tau_n) \in \Sigma_V^+$ sur l'alphabet Σ_V avec:

- $\forall i \in \mathrm{pos}(w)$, $\tau_i(x) = 1$ ssi $\sigma(x) = i$
  si $x \in V$ est une variable du premier ordre,
- $\forall i \in \mathrm{pos}(w)$, $\tau_i(X) = 1$ ssi $i \in \sigma(X)$
  si $X \in V$ est une variable monadique du second ordre.

Un mot $W \in \Sigma_V^+$ est valide si il code un couple (w, σ). On identifie W et (w, σ).

Définition : Sémantique de $\mathrm{MSO}(\Sigma, <)$

Soit $\varphi \in \mathrm{MSO}(\Sigma, <)$ et soit V un ensemble de variables contenant les variables libres de φ,

$$\mathcal{L}_V(\varphi) = \{W \in \Sigma_V^+ \mid W = (w, \sigma) \text{ est valide et } (w, \sigma) \models \varphi\}$$

Logique sur les mots

Théorème : Büchi 1960, Elgot 1961, Trakhtenbrot 1961

Un langage $L \subseteq \Sigma^+$ est reconnaissable si et seulement si il est définissable par une formule close $\varphi \in \mathrm{MSO}(\Sigma, <)$.

Preuve

$\Longrightarrow$: Si L est reconnu par un automate $\mathcal{A}$ ayant n états $Q = \{q_1, \ldots, q_n\}$, on écrit une formule de la forme $\varphi = \exists X_1 \cdots \exists X_n \, \psi(X_1, \ldots, X_n)$ qui caractérise l'existence d'un calcul acceptant de $\mathcal{A}$ sur un mot $w \in \Sigma^+$.
X_i est l'ensemble des positions de w pour lesquelles le calcul est dans l'état q_i.
La formule ψ assure que les transitions de l'automate sont respectées.

$\Longleftarrow$: On donne des expressions rationnelles pour les formules atomiques.
On note $\Sigma_V^{x=1} = \{(a, \tau) \in \Sigma_V \mid \tau(x) = 1\}$, de même $\Sigma_V^{x=0}$ ou $\Sigma_V^{X=1}$ou $\Sigma_V^{X=0}$.
$\mathcal{L}_V(P_a(x)) = (\Sigma_V^{x=0})^*(\Sigma_V^{x=1} \cap (\{a\} \times \{0,1\}^V))(\Sigma_V^{x=0})^*$.
$\mathcal{L}_V(x \in X) = (\Sigma_V^{x=0})^*(\Sigma_V^{x=1} \cap \Sigma_V^{X=1})(\Sigma_V^{x=0})^*$.
$\mathcal{L}_V(x < y) = (\Sigma_V^{x=y=0})^*(\Sigma_V^{x=1} \cap \Sigma_V^{y=0})(\Sigma_V^{x=y=0})^*(\Sigma_V^{x=0} \cap \Sigma_V^{y=1})(\Sigma_V^{x=y=0})^*$.
On utilise les propriétés de clôture des langages reconnaissables :
union ($\vee$), complémentaire ($\neg$) et projection ($\exists x$ et $\exists X$).

Morphismes

Définition : Reconnaissance par morphisme

- $\varphi : \Sigma^* \to M$ morphisme dans un monoïde fini M.
  $L \subseteq \Sigma^*$ est *reconnu* ou *saturé* par φ si $L = \varphi^{-1}(\varphi(L))$.
- $L \subseteq \Sigma^*$ est reconnu par un monoïde fini M s'il existe un morphisme $\varphi : \Sigma^* \to M$ qui reconnaît L.
- $L \subseteq \Sigma^*$ est *reconnaissable par morphisme* s'il existe un monoïde fini qui reconnaît L.

Définition : Monoïde de transitions

Soit $\mathcal{A} = (Q, \Sigma, \delta, i, F)$ un automate déterministe complet.
Le monoïde de transitions de $\mathcal{A}$ est le sous monoïde de $(Q^Q, *)$ engendré par les applications $\delta_a : Q \to Q$ $(a \in \Sigma)$ définies par $\delta_a(q) = \delta(q, a)$ et avec la loi de composition interne $f * g = g \circ f$.

Proposition :

Le monoïde de transitions de $\mathcal{A}$ reconnaît $\mathcal{L}(\mathcal{A})$.

Morphismes

Théorème :

Soit $L \subseteq \Sigma^*$. L est reconnaissable par morphisme ssi L est reconnaissable par automate.

Corollaire :

$\mathrm{Rec}(\Sigma^*)$ est fermée par morphisme inverse.

Exemple :

Si L est reconnaissable alors $\sqrt{L} = \{v \in \Sigma^* \mid v^2 \in L\}$ est aussi reconnaissable.

Exercices :

1. Montrer que $\mathrm{Rec}(\Sigma^*)$ est fermée par union, intersection, complémentaire.
2. Montrer que $\mathrm{Rec}(\Sigma^*)$ est fermée par quotients.
   Si $L \in \mathrm{Rec}(\Sigma^*)$ et $K \subseteq \Sigma^*$ alors $K^{-1}L$ et LK^{-1} sont reconnaissables.
3. Montrer que $\mathrm{Rec}(\Sigma^*)$ est fermée par concaténation (plus difficile).

Congruences

Définition :

Soit $L \subseteq \Sigma^*$ et $\equiv$ une congruence sur Σ^*.
Le langage L est *saturé* par $\equiv$ si $\forall u, v \in \Sigma^*$, $u \equiv v$ implique $u \in L \iff v \in L$.

Théorème :

Soit $L \subseteq \Sigma^*$. L est reconnaissable ssi L est saturé par une congruence d'index fini.

Exemple : Automate à double sens (Boustrophédon)

Un automate Boustrophédon est un automate fini non déterministe qui, à chaque transition, peut déplacer sa tête de lecture vers la droite ou vers la gauche.
De façon équivalente, c'est une machine de Turing à une seule bande qui n'écrit pas sur cette bande.

1. Montrer que tout langage accepté par un automate Boustrophédon est en fait rationnel.
2. Montrer qu'à partir d'un automate Boustrophédon ayant n états, on peut effectivement construire un automate déterministe classique équivalent ayant $2^{\mathcal{O}(n^2)}$ états.

Morphismes et Congruences

Exercice :

Soit L un langage reconnaissable. Montrer que le langage

$$L' = \{v \in \Sigma^* \mid v^{|v|} \in L\}$$

est aussi reconnaissable.

Exercice : Machine de Turing et automates

Une machine de Turing qui ne modifie pas sa donnée est une MT à une seule bande qui ne peut pas modifier le mot d'entrée, mais qui peut bien sûr écrire sur sa bande en dehors de la zone occupée par le mot d'entrée. La MT peut être non déterministe et ne s'arrête pas forcément.

1. Montrer qu'une MT qui ne modifie pas sa donnée reconnaît en fait un langage rationnel.
2. Étant donnée une MT qui ne modifie pas sa donnée, montrer que l'on peut effectivement calculer la fonction de transition d'un automate fini déterministe équivalent.
3. Peut-on décider le problème du mot pour une MT qui ne modifie pas sa donnée ?

Congruence et monoide syntaxique

Définition : Congruence syntaxique

Soit $L \subseteq \Sigma^*$. $u \equiv_L v$ si $\forall x, y \in \Sigma^*, xuy \in L \Longleftrightarrow xvy \in L$.

Théorème :

Soit $L \subseteq \Sigma^*$.

- $\equiv_L$ est une congruence et $\equiv_L$ sature L.
- $\equiv_L$ est la plus *grossière* congruence qui sature L.
- L est reconnaissable ssi $\equiv_L$ est d'index fini.

Définition : Monoide syntaxique

Soit $L \subseteq \Sigma^*$. $M_L = \Sigma^* / \equiv_L$.

Théorème :

Soit $L \subseteq \Sigma^*$.

- M_L est le monoïde de transitions de l'automate minimal de L.
- M_L divise (est quotient d'un sous-monoïde) tout monoïde qui reconnaît L.

On peut effectivement calculer le monoïde syntaxique d'un langage reconnaissable.

Congruences

Exercice : Congruence à droite

1. Montrer que $L \subseteq \Sigma^*$ est reconnaissable ssi il est saturé par une congruence à droite d'index fini
2. Soit $u \equiv_L^r v$ si $\forall y \in \Sigma^*$, $uy \in L \iff vy \in L$.
   Montrer que $\equiv_L^r$ est la congruence à droite la plus grossière qui sature L.
3. Faire le lien entre $\equiv_L^r$ et l'automate minimal de L

Apériodiques et sans étoile

Définition : Sans étoile

La famille des langages sans étoile est la plus petite famille qui contient les langages finis et qui est fermée par union, concaténation et complémentaire.

Exemple : Le langage $(ab)^*$ est sans étoile.

Définition : Apériodique

- Un monoïde fini M est apériodique si il existe $n \geq 0$ tel que pour tout $x \in M$ on a $x^n = x^{n+1}$.
- Un langage est apériodique s'il peut être reconnu par un monoïde apériodique.
- Rem: L est apériodique si et seulement si M_L est fini et apériodique.

Théorème : Schützenberger 1965

Un langage est sans étoile si et seulement si son monoïde syntaxique est apériodique.

Exemple : Le langage $(aa)^*$ n'est pas sans étoile.

Exercice :

Montrer que le langage $((a + cb^*a)c^*b)^*$ est sans étoile.

Sans étoile et sans compteur

Définition : Compteur

Soit $\mathcal{A} = (Q, \Sigma, \delta, i, F)$ un automate déterministe complet.
L'automate $\mathcal{A}$ est sans compteur si

$$\forall w \in \Sigma^*,\ \forall m \geq 1,\ \forall p \in Q, \qquad \delta(p, w^m) = p \quad \Rightarrow \quad \delta(p, w) = p\ .$$

Exemple : L'automate minimal de $(aa)^*$ possède un compteur.

Théorème : Mc Naughton, Papert 1971

Un langage est sans étoile si et seulement si son automate minimal est sans compteur.

Exercice :

Montrer que le langage $((a + cb^*a)c^*b)^*$ est sans étoile.

Sans étoile et logique du premier ordre

Théorème : Mc Naughton, Papert 1971

Un langage $L \subseteq \Sigma^+$ est sans étoile si et seulement si il est définissable par une formule de la logique du premier ordre $\mathrm{FO}(\Sigma, <)$.

Exemple : Le langage $(aa)^*$ n'est pas définissable en $\mathrm{FO}(\Sigma, <)$.

Exercice :

Montrer que le langage $((a+cb^*a)c^*b)^*$ est définissable en logique du premier ordre: $\mathrm{FO}(\Sigma, <)$.

Théorème :

Un langage $L \subseteq \Sigma^+$ est sans étoile si et seulement si il est définissable par une formule $\varphi \in \mathrm{FO}_3(\Sigma, <)$ qui utilise au plus 3 noms de variables.

Exercice :

Montrer que $((a+cb^*a)c^*b)^*$ est définissable par une formule de $\mathrm{FO}_3(\Sigma, <)$.

Plan

Introduction

Langages reconnaissables

3 Grammaires

- Type 0 : générale
- Type 1 : contextuelle (context-sensitive)
- Type 2 : hors contexte (context-free, algébrique)
- Hiérarchie de Chomsky

Langages algébriques

Automates à pile

Analyse syntaxique

Automates d'arbres

Grammaires de type 0

Définition : Grammaires générales (type 0)

$G = (\Sigma, V, P, S)$ où

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

Exemple : Une grammaire pour $\{a^{2^n} \mid n > 0\}$

$1: \quad S \to DXaF$	$3: \quad XF \to YF$	$5: \quad DY \to DX$	$7: \quad aZ \to Za$
$2: \quad Xa \to aaX$	$4: \quad aY \to Ya$	$6: \quad XF \to Z$	$8: \quad DZ \to \varepsilon$

Définition : Dérivation

$\alpha \in (\Sigma \cup V)^*$ se dérive en $\beta \in (\Sigma \cup V)^*$, noté $\alpha \to \beta$, s'il existe $(\alpha_2, \beta_2) \in P$ tel que $\alpha = \alpha_1\alpha_2\alpha_3$ et $\beta = \alpha_1\beta_2\alpha_3$.
On note $\xrightarrow{*}$ la clôture réflexive et transitive de $\to$.

Grammaires de type 0

Définition : 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 $L_G(S)$.
Un langage est *de type 0* s'il peut être engendré par une grammaire de type 0.

Théorème : Type 0 [9, Thm 9.3 & 9.4]

Un langage $L \subseteq \Sigma^*$ est de type 0 ssi il est récursivement énumérable.

Grammaires contextuelles

Définition : Grammaire contextuelle (type 1, context-sensitive)

Une grammaire $G = (\Sigma, V, P, S)$ est *contextuelle* si toute règle $(\alpha, \beta) \in P$ vérifie $|\alpha| \leq |\beta|$.
Un langage est de type 1 (ou contextuel) s'il peut être engendré par une grammaire contextuelle.

Exemple : Une grammaire contextuelle pour $\{a^{2^n} \mid n > 0\}$

$1: \ S \to DTF$	$3: \ T \to XT$	$5: \ Xaa \to aaXa$	$7: \ Daaa \to aaDaa$
$2: \ S \to aa$	$4: \ T \to aa$	$6: XaF \to aaF$	$8: DaaF \to aaaa$

Remarque :

Le langage engendré par une grammaire contextuelle est propre.
Si on veut engendrer le mot vide on peut ajouter $\hat{S} \to S + \varepsilon$.

Grammaires contextuelles

Définition : Forme normale (context-sensitive/contextuelle)

Une grammaire $G = (\Sigma, V, P, S)$ contextuelle est en forme normale si toute règle est de la forme $(\alpha_1 X \alpha_2, \alpha_1 \beta \alpha_2)$ avec $X \in V$ et $\beta \neq \varepsilon$.

Théorème : Forme normale [4, Prop. 2, p. 156]

Tout langage de type 1 est engendré par une grammaire contextuelle en forme normale.

Exemple : Une grammaire contextuelle en FN pour $\{a^{2^n} \mid n > 0\}$

$$\begin{array}{llll}
1: \ S \to aTa & 3: \ T \to XT & 5: \ XA \to XY & 8: \ Xa \to AAa \\
2: \ S \to aa & 4: \ T \to AA & 6: \ XY \to ZY & 9: \ ZA \to AAA \\
 & & 7: \ ZY \to ZX & 10: \ aA \to aa
\end{array}$$

Grammaires contextuelles

Théorème : Type 1 [9, Thm 9.5 & 9.6]

Un langage est de type 1 ssi il est accepté par une machine de Turing non déterministe en espace linéaire.
Les langages contextuels sont strictement inclus dans les langages récursifs.

Théorème : Problème du mot

Étant donnés un mot w et une grammaire G, décider si $w \in L_G(S)$.
Le problème du mot est décidable en PSPACE pour les grammaires de type 1.

Théorème : indécidabilité du vide

On ne peut pas décider si une grammaire contextuelle engendre un langage vide.

Exercices :

1. Montrer que $\{a^{n^2} \mid n > 0\}$ est contextuel.
2. Montrer que $\{ww \mid w \in \{a,b\}^+\}$ est contextuel.

Hiérarchie de Chomsky

Théorème : Chomsky

1. Les langages réguliers (type 3) sont strictement contenus dans les langages linéaires.
2. Les langages linéaires sont strictement contenus dans les langages algébriques (type 2).
3. Les langages algébriques propres (type 2) sont strictement contenus dans les langages contextuels (type 1).
4. les langages contextuels (type 1) sont strictement contenus dans les langages récursifs.
5. les langages récursifs sont strictement contenus dans les langages récursivement énumérables (type 0).

Bibliographie

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

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

[7] Olivier Carton.
Langages formels, calculabilité et complexité.
Vuibert, 2008.

[9] John E. Hopcroft et Jeffrey D. Ullman.
Introduction to automata theory, languages and computation.
Addison-Wesley, 1979.

[10] Dexter C. Kozen.
Automata and Computability.
Springer, 1997.

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

Plan

Grammaires algébriques

Définition : Grammaire hors contexte ou algébrique ou de type 2

$G = (\Sigma, V, P, S)$ où

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

Exemples :

1. Le langage $\{a^n b^n \mid n \geq 0\}$ est engendré par $S \to aSb \mid \varepsilon$.
2. Expressions complètement parenthésées.

Définition : Dérivation

$\alpha \in (\Sigma \cup V)^*$ se dérive en $\beta \in (\Sigma \cup V)^*$, noté $\alpha \to \beta$, s'il existe $(\alpha_2, \beta_2) \in P$ tel que $\alpha = \alpha_1 \alpha_2 \alpha_3$ et $\beta = \alpha_1 \beta_2 \alpha_3$.
On note $\xrightarrow{*}$ la clôture réflexive et transitive de $\to$.

Langages algébriques

Définition : 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 $L_G(S)$.
Un langage est algébrique (context-free, de type 2) s'il peut être engendré par une grammaire hors contexte.
On note Alg la famille des langages algébriques.

Lemme : fondamental

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

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

Langages de Dyck

Définition : D_n^*

Soit $\Sigma_n = \{a_1, \ldots, a_n\} \cup \{\bar{a}_1, \ldots, \bar{a}_n\}$ l'alphabet formé de n paires de parenthèses.
Soit $G_n = (\Sigma_n, V, P_n, S)$ la grammaire définie par $S \to a_1 S \bar{a}_1 S + \cdots + a_n S \bar{a}_n S + \varepsilon$.
Le langage $D_n^* = \mathcal{L}_{G_n}(S)$ est appelé langage de Dyck sur n paires de parenthèses

Exercices : Langages de Dyck

1. Montrer que
   $D_1^* = \{w \in \Sigma_1^* \mid |w|_{a_1} = |w|_{\bar{a}_1} \text{ et } |v|_{a_1} \geq |v|_{\bar{a}_1} \text{ pour tous } v \leq w\}$.
2. On considère le système de réécriture (type 0) $R_n = (\Sigma_n, P'_n)$ dont les règles sont $P'_n = \{(a_i \bar{a}_i, \varepsilon) \mid 1 \leq i \leq n\}$.
   Montrer que $D_n^* = \{w \in \Sigma_n^* \mid w \xrightarrow{*} \varepsilon \text{ dans } R_n\}$.
3. Soit Γ un alphabet disjoint de Σ_n, $\Sigma = \Sigma_n \cup \Gamma$ et $L \subseteq \Sigma^*$ un langage.

   On définit la clôture $\mathrm{clot}(L) = \{v \in \Sigma^* \mid \exists w \in L, w \xrightarrow{*} v \text{ dans } R_n\}$.
   Montrer que si L est reconnaissable, alors $\mathrm{clot}(L)$ aussi.

   On définit la réduction $\mathrm{red}(L) = \{v \in \mathrm{clot}(L) \mid v \not\to \text{ dans } R_n\}$.
   Montrer que si L est reconnaissable, alors $\mathrm{red}(L)$ aussi.

Grammaires linéaires

Définition : Grammaire linéaire

La grammaire $G = (\Sigma, V, P, S)$ est

- linéaire si $P \subseteq V \times (\Sigma^* \cup \Sigma^* V \Sigma^*)$,
- linéaire gauche si $P \subseteq V \times (\Sigma^* \cup V \Sigma^*)$,
- linéaire droite si $P \subseteq V \times (\Sigma^* \cup \Sigma^* V)$.

Un langage est linéaire s'il peut être engendré par une grammaire linéaire.

On note Lin la famille des langages linéaires.

Exemples :

- Le langage $\{a^n b^n \mid n \geq 0\}$ est linéaire.
- Le langage $\{a^n b^n c^p \mid n, p \geq 0\}$ est linéaire.

Proposition :

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

Arbres de dérivation

Définition :

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

Un arbre de dérivation pour G est un arbre t étiqueté dans $V \cup \Sigma \cup \{\varepsilon\}$ tel que chaque nœud interne u est étiqueté par une variable $x \in V$ et si les fils de u portent les étiquettes $\alpha_1, \ldots, \alpha_k$ alors $(x, \alpha_1 \cdots \alpha_k) \in P$.

De plus, si $k \neq 1$, on peut supposer $\alpha_1, \ldots, \alpha_k \neq \varepsilon$.

Exemple :

Arbres de dérivation pour les expressions.
Mise en évidence des priorités ou de l'associativité G ou D.

Arbres de dérivation

Lemme : Dérivations et arbres de dérivation

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

1. Si $x \xrightarrow{*} \alpha$ une dérivation de G alors il existe un arbre de dérivation t de G tel que $\mathrm{rac}(t) = x$ et $\mathrm{Fr}(t) = \alpha$.
2. Si t un arbre de dérivation de G alors il existe une dérivation $\mathrm{rac}(t) \xrightarrow{*} \mathrm{Fr}(t)$ dans G.
   Si $\mathrm{Fr}(t) \in \Sigma^*$ alors on peut faire une dérivation gauche $\mathrm{rac}(t) \xrightarrow{*}_g \mathrm{Fr}(t)$.

Une dérivation est *gauche* si on dérive toujours le non terminal le plus à gauche.

Remarques :

- 2 dérivations sont équivalentes si elles sont associées au même arbre de dérivation.
- Il y a bijection entre dérivations gauches terminales et arbres de dérivation ayant une frontière dans Σ^*.
- Si la grammaire est linéaire, il y a bijection entre dérivations et arbres de dérivations.

Grammaires et automates d'arbres

Théorème :

1. Soit L un langage d'arbres reconnaissable.
   Le langage $\mathrm{Fr}(L)$ des frontières des arbres de L est algébrique.
2. Soit L' un langage algébrique *propre* ($\varepsilon \notin L'$).
   Il existe un langage d'arbres reconnaissable L tel que $L' = \mathrm{Fr}(L)$.

Ambiguïté

Définition : Ambiguïté

- 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.

Exemples :

- La grammaire $S \to SS + aSb + \varepsilon$ est ambiguë mais elle engendre un langage non ambigu.
- La grammaire $E \to E + E \mid E \times E \mid a \mid b \mid c$ est ambiguë et engendre un langage rationnel.

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

Exercice : if then else

Montrer que la grammaire suivante est ambiguë.

$$S \to \texttt{if}\ c\ \texttt{then}\ S\ \texttt{else}\ S \mid \texttt{if}\ c\ \texttt{then}\ S \mid a$$

Montrer que le langage engendré n'est pas ambigu.

Lemme d'itération

Théorème : 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 :

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

Lemme d'Ogden

Plus fort que le théorème de Bar-Hillel, Perles, Shamir.

Lemme : 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.

Lemme d'Ogden

Exercice :

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

Corollaire :

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

Exercice :

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 :

Les langages non ambigus ne sont pas fermés par union.

Propriétés de clôture

Proposition :

1. La famille Alg est fermée par concaténation, itération.
2. La famille Alg est fermée par substitution algébrique.
3. Les familles Alg et Lin sont fermées par union et miroir.
4. Les familles Alg et Lin sont fermées par intersection avec un rationnel.
5. Les familles Alg et Lin sont fermées par morphisme.
6. Les familles Alg et Lin sont fermées par projection inverse.
7. Les familles Alg et Lin sont fermées par morphisme inverse.

Définition : Substitutions algébriques

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

Définition : Projection

La projection de A sur $B \subseteq A$ est le morphisme $\pi : A^* \to B^*$ défini par

$$\pi(a) = \begin{cases} a & \text{si } a \in B \\ \varepsilon & \text{sinon.} \end{cases}$$

Transductions rationnelles

Définition : Transduction rationnelle

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 $\tau(w) = \psi(\varphi^{-1}(w) \cap K)$ est une TR.

Proposition :

Les familles Alg, Lin et Rat sont fermées par TR.

Transductions rationnelles

Théorème : 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 $L = \psi(D_n^* \cap K)$.

Corollaire :

Les langages non ambigus ne sont pas fermés par morphisme.

Théorème : Elgot et Mezei, 1965

La composée de deux TR est encore une TR.

Théorème : 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 de $A^* \times B^*$).

Grammaires réduites

La taille d'une grammaire $G = (\Sigma, V, P, S)$ est $|G| = |\Sigma| + |V| + ||P||$
avec $||P|| = \sum_{x \to \alpha \in P} 1 + |\alpha|$.

Définition : 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 : Soit $G = (\Sigma, V, P, S)$ une grammaire.

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

Corollaire : Soit $G = (\Sigma, V, P, S)$ une grammaire

Si $\mathcal{L}_G(S) \neq \emptyset$, on peut construire une grammaire réduite équivalente $\mathcal{O}(|G|)$.

Preuve : Restreindre aux variables productives, puis aux variables accessibles.

Grammaires propres

Définition : Grammaires propres

La grammaire $G = (\Sigma, V, P, S)$ est propre si $P \subseteq V \times ((\Sigma \cup V)^+ \setminus V)$,
i.e., 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 :

Soit $G = (\Sigma, V, P, S)$ une grammaire.
On peut calculer l'ensemble des variables x telles que $\varepsilon \in \mathcal{L}_G(x)$ $\mathcal{O}(|G|)$.
On peut construire une grammaire équivalente sans ε-règle autre que $S \to \varepsilon$ et dans ce cas S n'apparaît dans aucun membre droit $\mathcal{O}(|G|)$.

Proposition :

Soit $G = (\Sigma, V, P, S)$ une grammaire.
On peut construire une grammaire propre G' qui engendre $\mathcal{L}_G(S) \setminus \{\varepsilon\}$ $\mathcal{O}(|G|^2)$.

Remarque : la réduction d'une grammaire propre est une grammaire propre.

Corollaire :

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

Grammaires quadratiques

Définition : Forme normale de Chomsky

Une grammaire $G = (\Sigma, V, P, S)$ est en forme normale

1. quadratique si $P \subseteq V \times (V \cup \Sigma)^{\leq 2}$
2. de Chomsky si $P \subseteq \{(S, \varepsilon)\} \cup (V \times (V^2 \cup \Sigma))$ et si $(S, \varepsilon) \in P$ alors S n'apparaît dans aucun membre droit.

Proposition :

Soit $G = (\Sigma, V, P, S)$ une grammaire.
On peut construire une grammaire équivalente en FN quadratique $\mathcal{O}(|G|)$.
On peut construire une grammaire équivalente en FN de Chomsky $\mathcal{O}(|G|^2)$.

Remarques :

1. La réduction d'une grammaire en FNC est encore en FNC.

Problèmes décidables

Proposition : Problème du mot : Cocke, Younger, Kasami [9, p. 139]

Soit G une grammaire algébrique.

On peut décider si un mot w est engendré par G en temps $\mathcal{O}(|w|^3)$.

Exercice :

Soit G une grammaire algébrique et $\mathcal{A}$ un automate fini.

Montrer que l'on peut décider en temps polynomial si $\mathcal{L}(G) \cap \mathcal{L}(\mathcal{A}) \neq \emptyset$.

Proposition : Vide et finitude

Soit G une grammaire algébrique.

On peut décider si le langage engendré par G est vide, fini ou infini (PTIME).

Problèmes indécidables

Proposition :

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 est-il rationnel ?
- L est-il déterministe ?
- L est-il ambigu ?
- $\overline{L}$ est-il algébrique ?
- $L \cap L'$ est-il algébrique ?

Forme normale de Greibach

Définition :

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(Q) à une FNG(Q).

Théorème :

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.

Forme normale de Greibach

Preuve

Soit $G = (\Sigma, V, P)$ une grammaire avec $V = \{x_1, \ldots, x_n\}$.

Pour $i, j \in \{1, \ldots, n\}$ on pose $\begin{cases} \alpha_{i,j} & = & x_i^{-1}P(x_j) \subseteq (\Sigma \cup V)^* \\ \beta_j & = & P(x_j) \cap (\Sigma \cdot (\Sigma \cup V)^* \cup \{\varepsilon\}) \end{cases}$

de sorte que les règles de G s'écrivent $x_j \to \sum_i x_i \alpha_{i,j} + \beta_j$ pour $1 \leq j \leq n$.

On peut écrire P vectoriellement : $\quad X \to XA + B$

avec $X = (x_1, \ldots, x_n)$, $B = (\beta_1, \ldots, \beta_n)$ et $A = (\alpha_{i,j})_{1 \leq i,j \leq n}$.

On définit $G' = (\Sigma, V', P')$ par $V' = V \uplus \{y_{i,j} \mid 1 \leq i, j \leq n\}$ et

$$P' : \quad \begin{array}{rcl} X & \to & BY + B \\ Y & \to & AY + A \end{array} \quad \text{i.e.} \quad \begin{array}{rcl} x_j & \to & \sum_k \beta_k y_{k,j} + \beta_j \\ y_{i,j} & \to & \sum_k \alpha_{i,k} y_{k,j} + \alpha_{i,j} \end{array}$$

avec $Y = (y_{i,j})_{1 \leq i,j \leq n}$.

Proposition : Equivalence des grammaires

Les grammaires G et G' sont équivalentes, i.e., $\forall x \in V, \quad \mathcal{L}_G(x) = \mathcal{L}_{G'}(x)$.

Forme normale de Greibach

Remarque : Grammaire propre

Si G est propre alors pour $1 \leq i,j \leq n$ on a

$$\alpha_{i,j} \subseteq (\Sigma \cup V)^+ \qquad \text{et} \qquad \beta_j \subseteq \Sigma \cdot (\Sigma \cup V)^*$$

donc les règles $X \to BY + B$ de G' sont en FNPG.

On définit G'' à partir de G' en remplaçant chaque variable x_ℓ en tête d'un mot de $\alpha_{i,j}$ par sa définition $\sum_k \beta_k y_{k,\ell} + \beta_\ell$.

Proposition : FNG et FNGQ

- Les grammaires G et G'' sont équivalentes.
- Si G est une grammaire propre alors G'' est en FNPG.
- Si G est propre et en FN de Chomsky, alors G'' est en FNGQ.

Exemples : Mettre les grammaires suivantes en FNG(Q)

$$G_1 : \begin{cases} x_1 \to x_1 b + a \\ x_2 \to x_1 b + a x_2 \end{cases} \qquad G_2 : \begin{cases} x_1 \to x_1(x_1 + x_2) + (x_2 a + b) \\ x_2 \to x_1 x_2 + x_2 x_1 + a \end{cases}$$

Équations algébriques

Définition : Système d'équations algébriques

Un système d'équations algébriques est un triplet (Σ, V, P) où :

- Σ est l'alphabet terminal,
- $V = \{X_1, \ldots, X_n\}$ est un ensemble fini de variables disjoint de Σ,
- $P = (P_1, \ldots, P_n)$ avec $P_i \subseteq (\Sigma \cup V)^*$ (non nécessairement fini).

On écrit le système d'équations $\overline{X} = P(\overline{X})$ ou $\begin{cases} X_1 = P_1(\overline{X}) \\ \quad\vdots \\ X_n = P_n(\overline{X}) \end{cases}$

Une solution est un tuple $\overline{L} = (L_1, \ldots, L_n)$ de langages sur Σ vérifiant $\overline{L} = P(\overline{L})$.

Exemple :

$\overline{L} = (a^+b^+, ab^*)$ est solution de $\begin{cases} X_1 = aX_1 + X_2b \\ X_2 = X_2b + a \end{cases}$

Équations algébriques

Théorème : Existence de solutions

Tout système (Σ, V, P) d'équations algébriques admet une plus petite solution :

$$\overline{L} = \bigsqcup_{n \geq 0} \overline{L}^n$$

avec $\overline{L}^0 = (\emptyset, \ldots, \emptyset)$ et $\overline{L}^{n+1} = P(\overline{L}^n)$.

Exercice : Grammaire et équations algébriques

Soit $G = (\Sigma, V, Q)$ une grammaire avec $V = \{X_1, \ldots, X_n\}$.
Le système d'équations associé est (Σ, V, P) où $P_i = \{\alpha \in (\Sigma \cup V)^* \mid (X_i, \alpha) \in Q\}$.

Montrer que $(L_G(X_1), \ldots, L_G(X_n))$ est la plus petite solution du système d'équations $\overline{X} = P(\overline{X})$.

Équations algébriques

Définition :

Un système d'équations (Σ, V, P) est

- propre si $P_i \cap (V \cup \{\varepsilon\}) = \emptyset$ pour tout i
- strict si $P_i \subseteq \{\varepsilon\} \cup (\Sigma \cup V)^* \Sigma (\Sigma \cup V)^*$ pour tout i

Le système est faiblement propre (resp. strict) s'il existe $k > 0$ tel que $\overline{X} = P^k(\overline{X})$ est propre (resp. strict).

Théorème : Unicité

Tout système (Σ, V, P) d'équations algébriques faiblement strict ou faiblement propre admet une solution unique.

Exemple :

D_1^* est l'unique solution de $X = aXbX + \varepsilon$.
$Ł$ est l'unique solution de $X = aXX + b$.
On en déduit $Ł = D_1^* b$.

Équations algébriques

Théorème : Résolution par élimination

On considère le système $\begin{cases} \overline{X} = P(\overline{X}, \overline{Y}) \\ \overline{Y} = Q(\overline{X}, \overline{Y}) \end{cases}$
avec $\overline{X} = (X_1, \ldots, X_n)$ et $\overline{Y} = (Y_1, \ldots, Y_m)$.

Soit $\overline{K}$ une solution de $\overline{Y} = Q(\overline{X}, \overline{Y})$ sur $\Sigma \cup \{X_1, \ldots, X_n\}$.
Soit $\overline{L}$ une solution de $\overline{X} = P(\overline{X}, \overline{K})$ sur Σ.

Alors, $(\overline{L}, \overline{K}(\overline{L}))$ est une solution du système $\begin{cases} \overline{X} = P(\overline{X}, \overline{Y}) \\ \overline{Y} = Q(\overline{X}, \overline{Y}) \end{cases}$

Exemple :

Résolution par élimination du système $\begin{cases} X_1 = aX_1 + bX_2 + \varepsilon \\ X_2 = bX_1 + aX_2 \end{cases}$

Exemple :

Résolution par élimination du système $\begin{cases} X = YX + b \\ Y = aX \end{cases}$

Plan

Automates à pile

Définition : $\mathcal{A} = (Q, \Sigma, Z, T, q_0 z_0, F)$ où

- Q ensemble fini d'états
- Σ alphabet d'entrée
- Z alphabet de pile
- $T \subseteq QZ \times (\Sigma \cup \{\varepsilon\}) \times QZ^*$ ensemble fini de transitions
- $q_0 z_0 \in QZ$ configuration initiale
- $F \subseteq Q$ acceptation par état final.

De plus, $\mathcal{A}$ est *temps-réel* s'il n'a pas d'ε-transition.

Définition : Système de transitions (infini) associé

- $\mathcal{T} = (QZ^*, T', q_0 z_0, FZ^*)$
- Une configuration de $\mathcal{A}$ est un état $ph \in QZ^*$ de $\mathcal{T}$
- Transitions de $\mathcal{T}$: $T' = \{pzh \xrightarrow{a} qgh \mid (pz, a, qg) \in T\}$.
- $\mathcal{L}(\mathcal{A}) = \{w \in \Sigma^* \mid \exists\ q_0 z_0 \xrightarrow{w} qh \in FZ^* \text{ dans } \mathcal{T}\}$.

Automates à pile

Exemples :

- $L_1 = \{a^n b^n c^p \mid n, p > 0\}$ et $L_2 = \{a^n b^p c^p \mid n, p > 0\}$
- $L = L_1 \cup L_2$ (non déterministe)

Exercices :

1. Montrer que le langage $\{w\tilde{w} \mid w \in \Sigma^*\}$ et son complémentaire peuvent être acceptés par un automate à pile.
2. Montrer que le complémentaire du langage $\{ww \mid w \in \Sigma^*\}$ peut être accepté par un automate à pile.
3. Soit $\mathcal{A} = (Q, \Sigma, Z, T, q_0 z_0, F)$ un automate à pile. Montrer qu'on peut construire un automate à pile équivalent $\mathcal{A}'$ tel que $T' \subseteq Q'Z \times (\Sigma \cup \{\varepsilon\}) \times Q'Z^{\leq 2}$.
4. Soit $\mathcal{A}$ un automate à pile. Montrer qu'on peut construire un automate à pile équivalent $\mathcal{A}'$ tel que les mouvements de la pile sont uniquement du type *push* ou *pop*.

Propriétés fondamentales

Lemme : fondamental

Soit $\mathcal{A} = (Q, \Sigma, Z, T, q_0 z_0)$ un automate à pile.

$pgh \xrightarrow[n]{w} r$ est un calcul de $\mathcal{A}$ ssi il existe deux calculs de $\mathcal{A}$:
$pg \xrightarrow[n_1]{w_1} q$ et $qh \xrightarrow[n_2]{w_2} r$ avec $w = w_1 w_2$ et $n = n_1 + n_2$.

Preuve

1. Si $pg \xrightarrow[n]{w} p'g'$ est un calcul de $\mathcal{A}$ et $h \in Z^*$ alors
   $pgh \xrightarrow[n]{w} p'g'h$ est aussi un calcul de $\mathcal{A}$.

2. Si $p_0 g_0 h \xrightarrow{a_1} p_1 g_1 h \cdots \xrightarrow{a_n} p_n g_n h$ est un calcul de $\mathcal{A}$ tel que $g_i \neq \varepsilon$ pour
   $0 \leq i < n$ alors $p_0 g_0 \xrightarrow{a_1} p_1 g_1 \cdots \xrightarrow{a_n} p_n g_n$ est un calcul de $\mathcal{A}$.

Acceptation généralisée

Définition :

Soit $\mathcal{A} = (Q, \Sigma, Z, T, q_0 z_0)$ un automate à pile et $K \subseteq QZ^*$ un langage reconnaissable. Le langage reconnu par $\mathcal{A}$ avec *acceptation généralisée* K est

$$\mathcal{L}_K(\mathcal{A}) = \{w \in \Sigma^* \mid \exists\ q_0 z_0 \xrightarrow{w} qh \in K \text{ dans } \mathcal{T}\}$$

Cas particuliers :

- $K = FZ^*$: acceptation classique par état final.
- $K = Q$: acceptation par pile vide.
- $K = F$: acceptation par pile vide et état final.
- $K = QZ'Z^*$ avec $Z' \subseteq Z$: acceptation par sommet de pile.

Exemple :

$L = \{a^n b^n \mid n \geq 0\}$ peut être accepté par pile vide ou par sommet de pile.

Proposition : Acceptation généralisée

Soit $\mathcal{A}$ un automate à pile avec acceptation généralisée K, on peut effectivement construire un automate à pile $\mathcal{A}'$ acceptant par état final tel que $\mathcal{L}_K(\mathcal{A}) = \mathcal{L}(\mathcal{A}')$.

Acceptation généralisée

Preuve : Acceptation généralisée

Soit $\mathcal{A} = (Q, \Sigma, Z, T, q_0 z_0)$ un automate à pile et $K \subseteq QZ^*$ un langage reconnu par l'automate fini déterministe $\mathcal{B} = (P, Z \cup Q, \delta, p_0, F)$ avec $P \cap Q = \emptyset$.
Soit $\mathcal{A}' = (Q', \Sigma, Z \uplus \{\bot\}, T', q_0'\bot, \{f\})$ avec $Q' = Q \uplus P \uplus \{q_0', f\}$, et

1. $q_0' \cdot \bot \xrightarrow{\varepsilon} q_0 \cdot z_0 \bot \in T'$, Initialisation
2. $T \subseteq T'$, Simulation
3. $q \cdot z \xrightarrow{\varepsilon} \delta(p_0, q) \cdot z \in T'$, si $q \in Q$ et $z \in Z \uplus \{\bot\}$, Acceptation
4. $p \cdot z \xrightarrow{\varepsilon} \delta(p, z) \cdot \varepsilon \in T'$, si $p \in P$ et $z \in Z$, Acceptation
5. $p \cdot \bot \xrightarrow{\varepsilon} f \cdot \varepsilon \in T'$, si $p \in F$, Acceptation

On a $\mathcal{L}(\mathcal{A}, K) = \mathcal{L}(\mathcal{A}')$.

Remarque: $\mathcal{A}'$ reconnaît aussi $\mathcal{L}(\mathcal{A}, K)$ par pile vide.

exo: Modifier $\mathcal{A}'$ pour qu'il reconnaisse $\mathcal{L}(\mathcal{A}, K)$ par sommet de pile.

Corollaire :

Tous les modes d'acceptation ci-dessus sont équivalents.

Automates à pile et grammaires

Proposition :

Soit $\mathcal{A} = (Q, \Sigma, Z, T, q_0 z_0)$ un automate à pile reconnaissant par pile vide. On peut construire une grammaire G qui engendre $\mathcal{L}(\mathcal{A})$.

De plus, si $\mathcal{A}$ est *temps-réel* alors G est en FNG.

Proposition :

Soit $G = (\Sigma, V, P, S)$ une grammaire. On peut construire un automate à pile simple (un seul état) $\mathcal{A}$ qui accepte $L_G(S)$ par pile vide.

De plus, si G est en FNPG alors on peut construire un tel $\mathcal{A}$ *temps-réel*.

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

Accessibilité et mots de pile

Proposition : Accessibilité et mots de pile

Soit $\mathcal{A} = (Q, \Sigma, Z, T, q_0 z_0)$ un automate à pile.
Pour $pg \in QZ^*$, on note

$$\mathcal{C}(pg) = \{qh \in QZ^* \mid \exists\ pg \xrightarrow[T]{} qh \text{ dans } \mathcal{T}\}$$

l'ensemble des configurations accessibles à partir de pg.
On peut effectivement construire un automate fini $\mathcal{B}$ qui reconnaît $\mathcal{C}(pg)$.

Corollaire : Décidabilité

Soit $\mathcal{A} = (Q, \Sigma, Z, T, q_0 z_0, F)$ un automate à pile.
On peut décider si $\mathcal{L}(\mathcal{A}) = \emptyset$.

Accessibilité et mots de pile (Preuve)

Soit $\mathcal{A} = (Q, \Sigma, Z, T, q_0 z_0)$ un automate à pile.
On définit $\Gamma = Q \uplus Z \uplus \overline{Q} \uplus \overline{Z}$ et la réduction sur Γ^* par

$$\begin{cases} \overline{q}q \xrightarrow{\text{red}} \varepsilon & \text{pour } q \in Q \\ \overline{z}z \xrightarrow{\text{red}} \varepsilon & \text{pour } z \in Z \end{cases}$$

Pour $L \subseteq \Gamma^*$ on pose $\mathrm{Clot}(L) = \{w \in \Gamma^* \mid \exists v \in L,\ v \xrightarrow[*]{\text{red}} w\}$.

Lemme : Clôture

Si $L \subseteq \Gamma^*$ est un langage rationnel alors $\mathrm{Clot}(L) \subseteq \Gamma^*$ aussi.
De plus, on peut effectivement construire un automate pour $\mathrm{Clot}(L)$ à partir d'un automate pour L.

Soit $K = \{qh\overline{xp} \mid \exists\ px \xrightarrow{a} qh \in T\} \subseteq \Gamma^+$, langage fini donc rationnel.

Lemme :

Soit $n \geq 0$,
il existe un calcul $pg \xrightarrow[n]{} qh$ dans $\mathcal{T}$ ssi il existe $w \in K^n$ tel que $wpg \xrightarrow[2n]{\text{red}} qh$

Corollaire : $\mathcal{C}(pg) = \mathrm{Clot}(K^+ \cdot pg) \cap QZ^*$.

Calculs d'accessibilité

Corollaire :

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

1. $X = \{(p, x, q) \in Q \times Z \times Q \mid \exists\ px \xrightarrow[+]{} q \text{ dans } \mathcal{T}\}$
2. $Y = \{(p, x, q, y) \in Q \times Z \times Q \times Z \mid \exists\ px \xrightarrow[+]{} qyh \text{ dans } \mathcal{T}\}$
3. $W = \{(p, x, q, y) \in Q \times Z \times Q \times Z \mid \exists\ px \xrightarrow[+]{} qy \text{ dans } \mathcal{T}\}$
4. $X' = \{(p, x, q) \in Q \times Z \times Q \mid \exists\ px \xrightarrow[+]{\varepsilon} q \text{ dans } \mathcal{T}\}$
5. $Y' = \{(p, x, q, y) \in Q \times Z \times Q \times Z \mid \exists\ px \xrightarrow[+]{\varepsilon} qyh \text{ dans } \mathcal{T}\}$
6. $W' = \{(p, x, q, y) \in Q \times Z \times Q \times Z \mid \exists\ px \xrightarrow[+]{\varepsilon} qy \text{ dans } \mathcal{T}\}$

Exercice :

Soit $\mathcal{A} = (Q, \Sigma, Z, T, q_0 z_0)$ un automate à pile.
Montrer qu'on peut effectivement calculer les ensembles suivants :

1. $V = \{(p, x) \in Q \times Z \mid \exists\ px \xrightarrow[\omega]{} \text{ dans } \mathcal{T}\}$
2. $V' = \{(p, x) \in Q \times Z \mid \exists\ px \xrightarrow[\omega]{\varepsilon} \text{ dans } \mathcal{T}\}$

Langages déterministes

Définition : Automate à pile déterministe

$\mathcal{A} = (Q, \Sigma, Z, T, q_0 z_0, F)$ est *déterministe* si

- $\forall (pz, a) \in QZ \times (\Sigma \cup \{\varepsilon\}), \quad |T(pz, a)| \leq 1,$
- $\forall\, pz \in QZ, \quad T(pz, \varepsilon) \neq \emptyset \implies \forall\, a \in \Sigma,\ T(pz, a) = \emptyset$

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

Exemples :

1. $\{a^n b a^n \mid n \geq 0\}$ peut être accepté par un automate D+TR mais pas par un automate D+S car il n'est pas fermé par préfixe.
2. 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.

Exercices :

1. Montrer que D_n^* est D+TR mais pas D+S.
2. Montrer que le langage $\{a^n b^n \mid n > 0\} \cup \{a^n b^{2n} \mid n > 0\}$ est non ambigu mais pas déterministe.

Acceptation par pile vide

Exemples :

1. Le langage $\{a^n b a^n \mid n \geq 0\}$ peut être accepté par *pile vide* par un automate D+TR+S.
2. Le langage $\{a^n b^p c a^n \mid n, p > 0\} \cup \{a^n b^p d b^p \mid n, p > 0\}$ peut être accepté par *pile vide* par un automate D.

Exercices :

1. Montrer qu'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. Montrer que pour les automates à pile déterministes, l'acceptation par pile vide est équivalente à l'acceptation par pile vide ET état final.

Exercice :

Montrer que D_n^* peut être accepté par sommet de pile par un automate D+TR+S.

Lemme d'itération pour les déterministes

Lemme : Itération

Soit $L \subseteq \Sigma^*$ un langage déterministe. Il existe un entier $N \in \mathbb{N}$ tel que tout mot $w \in L$ contenant au moins N lettres distinguées se factorise en $w = \alpha u \beta v \gamma$ avec

1. $\forall p \geq 0 : w = \alpha u^p \beta v^p \gamma \in \mathcal{L}(\mathcal{A})$,
2. $u\beta v$ contient moins de N lettres distinguées,
3. soit α, u, β soit β, v, γ contiennent des lettres distiguées,
4. pour tout $\gamma' \in \Sigma^*$,

$$\exists p : \alpha u^p \beta v^p \gamma' \in L \quad \Longrightarrow \quad \forall p : \alpha u^p \beta v^p \gamma' \in L$$

Langages déterministes

Proposition : Décidabilité et indécidabilité

On ne peut pas décider si un langage algébrique est déterministe.

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 ?

Complémentaire

Théorème : Les déterministes sont fermés par complémentaire.

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

Il y a deux difficultés principales :

1. Un automate déterministe peut se bloquer (deadlock) ou entrer dans un ε-calcul infini (livelock). Dans ce cas il y a des mots qui n'admettent aucun calcul dans l'automate.
2. Même avec un automate déterministe, un mot peut avoir plusieurs calculs (ε-transitions à la fin) certains réussis et d'autres non.

Blocage

Définition : Blocage

Un automate à pile $\mathcal{A} = (Q, \Sigma, Z, T, q_0 z_0)$ est sans blocage si pour toute configuration accessible $p\alpha$ et pour toute lettre $a \in \Sigma$ il existe un calcul $p\alpha \xrightarrow[*]{\varepsilon} \xrightarrow{a}$.

Proposition : Critère d'absence de blocage

Un automate *déterministe* est sans blocage si et seulement si pour toute configuration accessible $p\alpha$ on a

1. $\alpha \neq \varepsilon$, et donc on peut écrire $\alpha = x\beta$ avec $x \in Z$,
2. $px \xrightarrow{\varepsilon}$ ou $\forall a \in \Sigma$, $px \xrightarrow{a}$,
3. $px \not\xrightarrow[\omega]{\varepsilon}$.

De plus, ce critère est décidable.

Remarque :

Si $\mathcal{A}$ est sans blocage alors chaque mot $w \in \Sigma^*$ a un unique calcul maximal (et fini) $q_0 z_0 \xrightarrow[*]{w} p\alpha \not\xrightarrow{\varepsilon}$ dans $\mathcal{A}$ (avec $\alpha \neq \varepsilon$).

Blocage

Proposition : Suppression des blocages

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

Preuve

$Q' = Q \uplus \{q'_0, d, f\}$, $F' = F \uplus \{f\}$, $Z' = Z \uplus \{\perp\}$, $z'_0 = \perp$ et
pour $p \in Q$, $a \in \Sigma$ et $x \in Z$

1. $q'_0 \perp \xrightarrow{\varepsilon} q_0 z_0 \perp$,
2. Si $px \xrightarrow{a} q\alpha \in T$ alors $px \xrightarrow{a} q\alpha \in T'$,
3. Si $px \not\xrightarrow{a}$ et $px \not\xrightarrow{\varepsilon}$ dans $\mathcal{A}$ alors $px \xrightarrow{a} dx \in T'$,
4. Si $px \not\xrightarrow[\omega]{\varepsilon}$ dans $\mathcal{A}$ et $px \xrightarrow{\varepsilon} q\alpha \in T$ alors $px \xrightarrow{\varepsilon} q\alpha \in T'$,
5. Si $px \xrightarrow[\omega]{\varepsilon}$ dans $\mathcal{A}$ et $\exists\ px \xrightarrow[*]{\varepsilon} q\alpha$ avec $q \in F$ alors $px \xrightarrow{\varepsilon} fx \in T'$,
6. Si $px \xrightarrow[\omega]{\varepsilon}$ dans $\mathcal{A}$ et $\forall\ px \xrightarrow[*]{\varepsilon} q\alpha$ on a $q \notin F$ alors $px \xrightarrow{\varepsilon} dx \in T'$,
7. $p\perp \xrightarrow{\varepsilon} d\perp$, $d\perp \xrightarrow{a} d\perp$, $dx \xrightarrow{a} dx$ et $fx \xrightarrow{a} dx$.

Cette construction est effective.

Complémentaire

Proposition :

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

Proposition :

Soit $\mathcal{A} = (Q, \Sigma, Z, T, q_0 z_0, F)$ un automate à pile déterministe, 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 $\mathcal{A}'$.

Exercice :

Montrer que tout langage déterministe est non ambigu.

Langages déterministes

Exercice :

Soit $\mathcal{A} = (Q, \Sigma, Z, T, q_0 z_0, K)$ un automate à pile déterministe avec acceptation généralisée par le langage rationnel $K \subseteq QZ^*$.
Montrer qu'on peut effectivement construire un automate à pile déterministe équivalent reconnaissant par état final.

Exercice :

Soit $\mathcal{A}$ un automate à pile déterministe. Montrer qu'on peut effectivement construire un automate à pile déterministe qui reconnaît le même langage et dont les ε-transitions sont uniquement effaçantes : $px \xrightarrow{\varepsilon} q$.

Plan

Bibliographie

[1] Alfred V. Aho, Ravi Sethi et Jeffrey D. Ullman.
Compilers: principles, techniques and tools.
Addison-Wesley, 1986.

[2] Alfred V. Aho et Jeffrey D. Ullman.
The theory of parsing, translation, and compiling. Volume I: Parsing.
Prentice-Hall, 1972.

[9] John E. Hopcroft et Jeffrey D. Ullman.
Introduction to automata theory, languages and computation.
Addison-Wesley, 1979.

Analyse syntaxique

Buts :

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

Rappels :

- 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 $w \in \mathcal{L}_G(S)$?
- L'arbre de dérivation est donné par la suite des règles utilisées lors d'une dérivation gauche (ou droite).

Analyse syntaxique

Rappels : le problème du mot est décidable

- Programmation dynamique : $\mathcal{O}(|w|^3)$.
  Ce n'est pas assez efficace.
- en lisant le mot si on a un automate à pile déterministe complet.
  $\mathcal{O}(|w|)$ si l'automate est temps réel ou si les ε-transitions ne font que dépiler.
  Mais la grammaire qui définit la syntaxe du langage de programmation peut être non déterministe ou ambiguë.

Exercice :

Si la grammaire n'est pas récursive à gauche ($x \overset{+}{\not\Rightarrow} x\alpha$), on peut construire un analyseur récursif avec backtracking. (Cet analyseur n'est pas efficace.)

Analyse descendante (LL)

Définition : Automate LL ou expansion/vérification

Soit $G = (\Sigma, V, P, S)$ une grammaire réduite.
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 : $\{(x, \varepsilon, \alpha) \mid (x, \alpha) \in P\}$ ou
- vérifications : $\{(a, a, \varepsilon) \mid a \in \Sigma\}$.

Remarque : sommet de pile à gauche.

Lemme :

Soient $w \in \Sigma^*$, $\alpha \in (\Sigma \cup V)^*$ et $\beta \in \{\varepsilon\} \cup V(\Sigma \cup V)^*$.

$\exists\ \alpha \xrightarrow{*} w\beta$ *dérivation gauche dans* G *ssi* $\exists\ \alpha \xrightarrow{w} \beta$ *calcul dans* $\mathcal{A}$.

Définition :

Analyse LL : $\left\{ \begin{array}{l} \text{L : le mot est lu de gauche à droite dans } \mathcal{A}. \\ \text{L : on construit une dérivation gauche dans } G. \end{array} \right.$

Analyse descendante (LL)

Problème :

L'automate ainsi obtenu est en général non déterministe.

Solution :

Pour lever le non déterminisme de l'automate on s'autorise à regarder les k prochaines lettres du mot.

Exemple :

1. $G_1 : S \to aSb + ab$.
   On peut lever le non déterminisme de l'automate associé à la grammaire G_1 en regardant les 2 prochaines lettres.

2. $G_2 : \left\{ \begin{array}{lll} E & \to & E + T \mid T \\ T & \to & T \times F \mid F \\ F & \to & (E) \mid a \mid b \mid c \end{array} \right.$
   On ne peut pas lever le non déterminisme de l'automate associé à la grammaire G_2 en regardant les k prochaines lettres.

Analyse LL avec lookahead

Définition : Table d'analyse LL avec lookahead

Soit $G = (\Sigma, V, P, S)$ une grammaire.
Une k-table d'analyse pour G est une application $M : V \times \Sigma^{\leq k} \to 2^P$ telle que pour $x \in V$ et $v \in \Sigma^{\leq k}$ on a $M(x, v) \subseteq P \cap (\{x\} \times (\Sigma \cup V)^*)$.
La table est déterministe si $|M(x, v)| \leq 1$ pour tout $x \in V$ et $v \in \Sigma^{\leq k}$.

Définition : Analyseur LL avec lookahead

Soit $G = (\Sigma, V, P, S)$ une grammaire et soit M une k-table d'analyse pour G.
L'analyseur LL défini par M, noté $\mathcal{A}_M$, est l'automate LL $\mathcal{A}$ associé à G dont les expansions sont pilotées par M:
Si $x \in V$ est au sommet de pile et si $v \in \Sigma^{\leq k}$ est le mot formé des (au plus) k prochaines lettres à lire, alors $\mathcal{A}_M$ choisit une expansion dans $M(x, v)$.
L'analyseur est bloqué (erreur) si $M(x, v) = \emptyset$.

Corollaire : $\mathcal{L}(\mathcal{A}_M) \subseteq \mathcal{L}(\mathcal{A}) = \mathcal{L}_G(S)$

Définition : L'analyseur $\mathcal{A}_M$ est complet si $\mathcal{L}(\mathcal{A}_M) = \mathcal{L}(\mathcal{A}) = \mathcal{L}_G(S)$

On dit aussi que la table M est complète pour G.

Analyse LL avec lookahead

Exemple :

1. Construire une 2-table d'analyse déterministe et complète pour G_1.
2. Construire une 1-table d'analyse déterministe et complète pour la grammaire usuelle du langage de Dyck D_n^* sur n paires de parenthèses:

$$S \to \varepsilon \mid a_1 S b_1 S \mid \cdots \mid a_n S b_n S$$

Exercice :

Transformer l'analyseur LL défini par une table déterministe en un automate à pile déterministe classique (sans lookahead) équivalent.

Objectif de l'analyse $\mathrm{LL}(k)$

Étant donnés une grammaire G et un entier k, construire *automatiquement* une k-table d'analyse déterministe et complète pour la grammaire G.

Analyse descendante First_k

Définition : First

- Pour $w \in \Sigma^*$ et $k \geq 0$, on définit $\mathrm{First}_k(w) = \begin{cases} w & \text{si } |w| \leq k \\ w[k] & \text{sinon.} \end{cases}$
- Pour $L \subseteq \Sigma^*$ et $k \geq 0$, $\mathrm{First}_k(L) = \{\mathrm{First}_k(w) \mid w \in L\}$.
- Soit $G = (\Sigma, V, P, S)$ une grammaire algébrique, $\alpha \in (\Sigma \cup V)^*$ et $k \geq 0$,

$$\mathrm{First}_k(\alpha) = \mathrm{First}_k(\mathcal{L}_G(\alpha)) \subseteq \Sigma^{\leq k}$$

Remarque :

$$\mathrm{First}_k(\alpha\beta) = \mathrm{First}_k(\mathrm{First}_k(\alpha) \cdot \mathrm{First}_k(\beta))$$

Exemple :

Calculer $\mathrm{First}_2(E)$ pour la grammaire G_2.

Remarque :

Pour $\alpha \in (\Sigma \cup V)^*$, $\mathrm{First}_0(\alpha) = \{\varepsilon\}$ ssi toutes les variables de α sont productives.

Calcul de First_k

Définition : Algorithme de calcul pour First_k $(k > 0)$

On définit $X_m(\alpha)$ pour $\alpha \in \Sigma \cup V$ et $m \geq 0$ par :

- si $a \in \Sigma$ alors $X_m(a) = \{a\}$ pour tout $m \geq 0$,
- si $x \in V$ alors $X_0(x) = \emptyset$ et

$$X_{m+1}(x) = \bigcup_{x \to \alpha_1 \cdots \alpha_n \in P} \mathrm{First}_k(X_m(\alpha_1) \cdots X_m(\alpha_n))$$

Proposition : Point fixe $(k > 0)$

1. $X_m(\alpha) \subseteq X_{m+1}(\alpha)$
2. $X_m(\alpha) \subseteq \mathrm{First}_k(\alpha)$
3. Si $\alpha \xrightarrow{m} w \in \Sigma^*$ alors $\mathrm{First}_k(w) \in X_m(\alpha)$.
4. $\mathrm{First}_k(\alpha) = \bigcup_{m \geq 0} X_m(\alpha)$

Ceci fournit un algorithme pour calculer $\mathrm{First}_k(\alpha)$ pour $\alpha \in \Sigma \cup V$.
Pour $\gamma \in (\Sigma \cup V)^*$ on utilise $\mathrm{First}_k(\alpha\beta) = \mathrm{First}_k(\mathrm{First}_k(\alpha) \cdot \mathrm{First}_k(\beta))$.
En particulier, $\mathrm{First}_k(\varepsilon) = \{\varepsilon\}$.

Analyse descendante $\mathrm{LL}(k)$

Définition : $\mathrm{LL}(k)$

Une grammaire $G = (\Sigma, V, P, S)$ est $\mathrm{LL}(k)$ si pour toute dérivation $S \xrightarrow{*} \gamma x \delta$ avec $x \in V$ et pour toutes règles $x \to \alpha$ et $x \to \beta$ avec $\alpha \neq \beta$, on a

$$\mathrm{First}_k(\alpha\delta) \cap \mathrm{First}_k(\beta\delta) = \emptyset.$$

Remarque : on peut se restreindre aux dérivations gauches avec $\gamma \in \Sigma^*$, i.e., aux calculs de l'automate LL.

Exemple :

1. La grammaire G_1 est $\mathrm{LL}(2)$ mais pas $\mathrm{LL}(1)$.
2. La grammaire G_2 n'est pas $\mathrm{LL}(k)$.
3. On peut transformer la grammaire G_2 en une grammaire $\mathrm{LL}(1)$ équivalente. Il suffit de supprimer la récursivité gauche.

$$G_2' = \left\{ \begin{array}{lcllcl} E & \to & TE' & E' & \to & +TE' \mid \varepsilon \\ T & \to & FT' & T' & \to & \times FT' \mid \varepsilon \\ F & \to & (E) \mid a \mid b \mid c & & & \end{array} \right.$$

Analyse descendante $\mathrm{LL}(k)$

Exercices :

1. Construire une k-table d'analyse déterministe et complète pour une grammaire $\mathrm{LL}(k)$.
2. Montrer qu'un langage $\mathrm{LL}(k)$ est déterministe.
3. Montrer que si l'automate expansion/vérification associé à une grammaire est déterministe, alors la grammaire est $\mathrm{LL}(0)$.
4. Montrer qu'une grammaire $\mathrm{LL}(0)$ engendre au plus un mot.
5. Montrer que si G est en FNPG et que pour toutes règles $x \to a\alpha$ et $x \to b\beta$ avec $a, b \in \Sigma$ on a $a \neq b$ ou $\alpha = \beta$, alors G est $\mathrm{LL}(1)$.
6. Montrer que la réciproque est fausse.
7. Montrer qu'un langage rationnel admet une grammaire $\mathrm{LL}(1)$.

Analyse descendante $\mathrm{LL}(k)$

Remarques :

- Étant donnés une grammaire G et un entier k, on peut décider si G est $\mathrm{LL}(k)$.
- Étant données deux grammaires $\mathrm{LL}(k)$, on peut décider si elles engendrent le même langage.
- La hiérarchie des langages $\mathrm{LL}(k)$ est stricte.
- Étant donnée une grammaire G, on ne peut pas décider s'il existe un entier k tel que G soit $\mathrm{LL}(k)$.
- Étant donnée une grammaire G, on ne peut pas décider s'il existe une grammaire équivalente qui soit $\mathrm{LL}(1)$.

Follow

Définition : Follow

Soit $G = (\Sigma, V, P, S)$ une grammaire algébrique, $x \in V$ et $k \geq 0$,

$$\mathrm{Follow}_k(x) = \bigcup_{\delta \,|\, \exists S \xrightarrow{*} \gamma x \delta} \mathrm{First}_k(\delta) = \{w \in \Sigma^* \mid \exists\ S \xrightarrow{*} \gamma x \delta \text{ avec } w \in \mathrm{First}_k(\delta)\}$$

Remarque : on peut se restreindre aux dérivations gauches avec $\gamma \in \Sigma^*$.

Théorème : Caractérisation

Les ensembles $(\mathrm{Follow}_k(x))_{x \in V}$ satisfont le système d'équations :

$$\mathrm{Follow}_k(S) = \{\varepsilon\} \cup \bigcup_{y \to \alpha S \beta} \mathrm{First}_k(\beta \mathrm{Follow}_k(y))$$

$$(x \neq S) \qquad \mathrm{Follow}_k(x) = \bigcup_{y \to \alpha x \beta} \mathrm{First}_k(\beta \mathrm{Follow}_k(y))$$

Exemple :

Calculer $\mathrm{Follow}_1(x)$ pour chaque variable x de la grammaire G_2'.

Calcul de Follow_k

Définition : Algorithme de calcul pour Follow_k

Pour $m \geq 0$ et $x \in V$, on définit $Y_m(x)$ par :

- $Y_0(S) = \{\varepsilon\}$ et $Y_0(x) = \emptyset$ si $x \neq S$
- $$Y_{m+1}(x) = Y_m(x) \cup \bigcup_{y \to \alpha x \beta \in P} \mathrm{First}_k(\beta Y_m(y))$$

Proposition : Point fixe

1. $Y_m(x) \subseteq Y_{m+1}(x)$
2. $Y_m(x) \subseteq \mathrm{Follow}_k(x)$
3. Si $S \xrightarrow{m} \gamma x \delta$ alors $\mathrm{First}_k(\delta) \subseteq Y_m(x)$.
4. $\mathrm{Follow}_k(x) = \bigcup_{m \geq 0} Y_m(x)$

Ceci fournit donc un algorithme pour calculer $\mathrm{Follow}_k(\alpha)$.

Fortement LL

Définition : Fortement $\mathrm{LL}(k)$

Une grammaire $G = (\Sigma, V, P, S)$ est fortement $\mathrm{LL}(k)$ si pour toutes règles $x \to \alpha$ et $x \to \beta$ avec $\alpha \neq \beta$, on a

$$\mathrm{First}_k(\alpha \mathrm{Follow}_k(x)) \cap \mathrm{First}_k(\beta \mathrm{Follow}_k(x)) = \emptyset$$

Proposition :

Si une grammaire G est fortement $\mathrm{LL}(k)$ alors elle est $\mathrm{LL}(k)$.

Exemple :

1. La grammaire G_1 est fortement $\mathrm{LL}(2)$.
2. La grammaire G_2' est fortement $\mathrm{LL}(1)$.
3. La grammaire $G_3 = \left\{ \begin{array}{lcl} S & \to & axaa \mid bxba \\ x & \to & b \mid \varepsilon \end{array} \right.$

   est $\mathrm{LL}(2)$ mais pas fortement $\mathrm{LL}(2)$.

Proposition :

Une grammaire est $\mathrm{LL}(1)$ si et seulement si elle est fortement $\mathrm{LL}(1)$.

Analyseur fortement LL

Définition : Analyseur fortement $\mathrm{LL}(k)$

Soit $G = (\Sigma, V, P, S)$ une grammaire.
La table d'analyse fortement $LL(k)$ de G est définie pour $x \in V$ et $v \in \Sigma^{\leq k}$ par

$$M_k(x, v) = \{x \xrightarrow{\varepsilon} \alpha \mid (x, \alpha) \in P \text{ et } v \in \mathrm{First}_k(\alpha \mathrm{Follow}_k(x))\}$$

L'analyseur fortement $\mathrm{LL}(k)$ associé est $\mathcal{A}_{M_k}$.

Proposition : Correction

Soit $G = (\Sigma, V, P, S)$ une grammaire.
La table d'analyse fortement $\mathrm{LL}(k)$ de G est complète: $\mathcal{L}(\mathcal{A}_{M_k}) = \mathcal{L}_G(S)$.
Si G est fortement $\mathrm{LL}(k)$ alors sa table d'analyse fortement $\mathrm{LL}(k)$ est déterministe.

Exemple :

1. Construire la table d'analyse fortement $\mathrm{LL}(2)$ de la grammaire G_1.
2. Construire la table d'analyse fortement $\mathrm{LL}(1)$ de la grammaire G_2'.
3. Construire la table d'analyse fortement $\mathrm{LL}(1)$ de la grammaire usuelle du langage de Dyck D_n^*.