

Langages formels

Paul Gastin

LSV (Cachan)
Paul.Gastin@lsv.ens-cachan.fr

L3 2010

Plan

1 Introduction

Langages reconnaissables

Automates d'arbres

Grammaires

Langages algébriques

Automates à pile

Analyse syntaxique

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

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.
- [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.

Bibliographie

- [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.
- [7] Olivier Carton.
Langages formels, calculabilité et complexité.
Vuibert, 2008.
- [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/>
- [9] John E. Hopcroft et Jeffrey D. Ullman.
Introduction to automata theory, languages and computation.
Addison-Wesley, 1979.

Bibliographie

- [10] Dexter C. Kozen.
Automata and Computability.
Springer, 1997.
- [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.
- [13] Jacques Sakarovitch.
Éléments de théorie des automates.
Vuibert informatique, 2003.
- [14] Jacques Stern.
Fondements mathématiques de l'informatique.
Mc Graw Hill, 1990.

Plan

Introduction

- ② Langages reconnaissables
 - Mots
 - Langages
 - Automates déterministes
 - Automates non déterministes
 - Automates avec ε -transitions
 - Propriétés de fermeture
 - Langages rationnels
 - Critères de reconnaissabilité
 - Minimisation
 - Morphismes et congruences

Automates d'arbres

Grammaires

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.
 (Σ^*, \cdot) : monoïde libre engendré par Σ .

$|u|$: longueur du mot u .
 $|\cdot| : \Sigma^* \rightarrow \mathbb{N}$ 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 $v = v_0u_1v_1u_1 \cdots u_nv_n$ avec $u_i, v_i \in \Sigma^*$ et $u = u_1u_2 \cdots u_n$

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,
 $\delta : Q \times \Sigma \rightarrow Q$ 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 $Q \times \Sigma^*$:

$$\delta(q, \varepsilon) = q,$$

$$\delta(q, u \cdot a) = \delta(\delta(q, u), a) \text{ si } u \in \Sigma^* \text{ et } a \in \Sigma.$$

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 $\text{Rec}(\Sigma^*)$ 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 $\delta : Q \times \Sigma \rightarrow 2^Q$.

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 $\text{Acc}(I)$ des états accessibles à partir de I et l'ensemble $\text{coAcc}(F)$ 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 donné 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 $\text{Rec}(\Sigma^*)$ est fermée par les opérations ensemblistes (union, complément, ...).

Preuve

Union : construction non déterministe.

Intersection : produit d'automates (préserve le déterminisme).

Complément : utilise la détermination.

Corollaire :

On peut décider de l'égalité ou de l'inclusion de langages reconnaissables.

Plus précisément, soient $L_1, L_2 \in \text{Rec}(\Sigma^*)$ 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 :

$\text{Rec}(\Sigma^*)$ 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

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

Proposition :

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

Preuve

Modification des états initiaux et/ou finaux.

Propriétés de fermeture

Proposition :

La famille $\text{Rec}(\Sigma^*)$ est fermée par quotients gauches et droits :

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

Preuve

Modification des transitions de l'automate.

Propriétés de fermeture

Définition : Substitutions

Une *substitution* est définie par une application $\sigma : A \rightarrow \mathcal{P}(B^*)$.
Elle s'étend en un morphisme $\sigma : A^* \rightarrow \mathcal{P}(B^*)$ 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 $\sigma : A \rightarrow \text{Rec}(B^*)$.

Propriétés de fermeture

Proposition :

La famille des langages reconnaissables est fermée par substitution rationnelle et substitution rationnelle inverse.

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

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 ER est un mot sur l'alphabet $\underline{\Sigma} \cup \{(\ , \cdot, +, *, \emptyset\}$

Définition : Syntaxe

L'ensemble des ER est défini par

$B : \emptyset$ 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 $\mathcal{L} : \mathcal{E} \rightarrow \mathcal{P}(\Sigma^*)$ par

$B : \mathcal{L}(\emptyset) = \emptyset$ 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 $\text{Rat}(\Sigma^*)$ l'ensemble des langages rationnels sur l'alphabet Σ .

Remarque : $\text{Rat}(\Sigma^*)$ 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

$$\text{Rec}(\Sigma^*) = \text{Rat}(\Sigma^*)$$

Preuve

- \supseteq : les langages \emptyset et $\{a\}$ pour $a \in \Sigma$ sont reconnaissables et la famille $\text{Rec}(\Sigma^*)$ est fermée par union, concaténation, itération.
- \subseteq : Algorithme de McNaughton-Yamada.

Corollaire :

L'équivalence des expressions rationnelles est décidable.

Preuve

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

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 $L \in \text{Rec}(\Sigma^*)$. 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 $x = w v_1 v_2 \dots v_N w$ avec $|v_i| \geq 1$ alors il existe $0 \leq j < k \leq N$ tels que $w v_1 \dots v_j (v_{j+1} \dots v_k)^* v_{k+1} \dots v_N w \subseteq L$.

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 $f : \Sigma^* \rightarrow \Sigma^*$ 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

$L_j = \{u = u_0u_1 \dots u_n \in \{0, 1\}^* \mid \bar{u}^2 = \sum_{i=0}^n u_i 2^i \equiv j[3]\}$.

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.

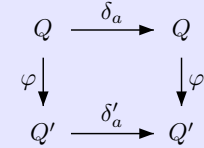
Morphismes d'automates

Définition : Morphismes d'automates DC (déterministes complets)

Soient $\mathcal{A} = (Q, \delta, i, F)$ et $\mathcal{A}' = (Q', \delta', i', F')$ deux automates DC.

Une application $\varphi : Q \rightarrow Q'$ 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., $q \in F \iff \varphi(q) \in F'$.



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

Remarques :

Deux automates DC sont isomorphes s'ils ne diffèrent que par le nom des états.
Si $\varphi : \mathcal{A} \rightarrow \mathcal{A}'$ est un morphisme bijectif, alors $\varphi^{-1} : \mathcal{A}' \rightarrow \mathcal{A}$ est un morphisme.
Si $\varphi : \mathcal{A} \rightarrow \mathcal{A}'$ et $\psi : \mathcal{A}' \rightarrow \mathcal{A}''$ sont des morphismes, alors $\psi \circ \varphi : \mathcal{A} \rightarrow \mathcal{A}''$ est un morphisme.

Morphismes et Résiduels

Proposition : morphisme et résiduels

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'application $\varphi : Q \rightarrow Q_L$ définie par $\varphi(q) = \mathcal{L}(\mathcal{A}, q)$ est un morphisme surjectif (canonique) de \mathcal{A} vers $\mathcal{R}(L)$.

Exemple :

Soit \mathcal{A} un automate DCA qui 'calcule' naturellement (6 états) $\bar{u}^2[3]$ et accepte L_1 (cf. exemple précédent). Calculer le morphisme de \mathcal{A} vers $\mathcal{R}(L_1)$.

Quotients

Définition : Quotients

Soient \mathcal{A} et \mathcal{A}' deux automates DC. On dit que \mathcal{A}' est un quotient de \mathcal{A} , et on note $\mathcal{A}' \preceq \mathcal{A}$, s'il existe un morphisme surjectif $\varphi : \mathcal{A} \rightarrow \mathcal{A}'$.

Proposition :

- ▶ \preceq est un ordre partiel sur les automates DC.
- ▶ Si $\mathcal{A}' \preceq \mathcal{A}$ alors $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{A}')$.

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

Minimalité de l'automate des résiduels

Théorème : Résiduels et quotients

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

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

Problème : comment calculer $\mathcal{R}(L)$?

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 δ_\sim est définie par $\delta_\sim([p], a) = [\delta(p, a)]$.

Remarque : $[-] : \mathcal{A} \rightarrow \mathcal{A}/\sim$ est un morphisme surjectif.

Proposition :

Soient $\mathcal{A} = (Q, \delta, i, F)$ et $\mathcal{A}' = (Q', \delta', i', F')$ deux automates DC.

Soit $\varphi : Q \rightarrow Q'$ une application et \sim_φ l'équivalence associée définie par $p \sim_\varphi q$ si $\varphi(p) = \varphi(q)$.

- ▶ Si φ est un morphisme alors \sim est une congruence.
- ▶ Si de plus φ est surjectif, alors \mathcal{A}' est isomorphe à \mathcal{A}/\sim_φ .

Ceci explique que l'on nomme "quotient" l'image par un morphisme surjectif.

Équivalence de Nerode

Définition : Équivalence de Nerode

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

L'équivalence de Nerode est la congruence associée au morphisme de \mathcal{A} sur $\mathcal{R}(L)$:

$$p \sim q \quad \text{ssi} \quad \mathcal{L}(\mathcal{A}, p) = \mathcal{L}(\mathcal{A}, q) \\ \text{ssi} \quad \forall w \in \Sigma^*, \delta(p, w) \in F \iff \delta(q, w) \in F$$

Donc le quotient \mathcal{A}/\sim (appelé automate de Nerode) est isomorphe à $\mathcal{R}(L)$.

Corollaire : Soit $L \in \text{Rec}(\Sigma^*)$.

1. On calcule l'automate minimal de L avec l'équivalence de Nerode à partir de n'importe quel automate DCA qui reconnaît L .
Remarque : On sait décider si $p \sim q$.
2. 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 ($\mathcal{A}/\sim = \mathcal{B}/\sim$).

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

Algorithme de Moore

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

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

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., $p \sim_{n+1} q \implies p \sim_n q$.

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 $n = |Q|$).

Morphismes

Définition : Reconnaissance par morphisme

- ▶ $\varphi : \Sigma^* \rightarrow M$ morphisme dans un monoïde fini M .
 $L \subseteq \Sigma^*$ est reconnu par φ si $L = \varphi^{-1}(\varphi(L))$.
- ▶ $L \subseteq \Sigma^*$ est reconnu par un monoïde fini M s'il existe un morphisme $\varphi : \Sigma^* \rightarrow M$ 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 $\delta_a : Q \rightarrow Q$ ($a \in \Sigma$) 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 :

$\text{Rec}(\Sigma^*)$ 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 $\text{Rec}(\Sigma^*)$ est fermée par union, intersection, complémentaire.
2. Montrer que $\text{Rec}(\Sigma^*)$ est fermée par quotients.
Si $L \in \text{Rec}(\Sigma^*)$ et $K \subseteq \Sigma^*$ alors $K^{-1}L$ et LK^{-1} sont reconnaissables.
3. Montrer que $\text{Rec}(\Sigma^*)$ 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 $\forall u \in \Sigma^*, \forall v \in L, u \equiv v$ implique $u \in L$.

Théorème :

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

Définition : Congruence syntaxique

Soit $L \subseteq \Sigma^*$.

$$u \equiv_L v \quad \text{si} \quad \forall x, y \in \Sigma^*, xuy \in L \iff xvy \in L.$$

Théorème :

Soit $L \subseteq \Sigma^*$.

- ▶ \equiv_L sature L .
- ▶ \equiv_L est la plus grossière congruence qui sature L .
- ▶ L est reconnaissable ssi \equiv_L est d'index fini.

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 : 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 : 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 ?

Monoïde syntaxique

Définition : Monoïde syntaxique

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

Théorème :

Soit $L \subseteq \Sigma^*$.

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

Corollaire :

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

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

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

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

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.

Plan

Introduction

Langages reconnaissables

3 Automates d'arbres

- Arbres
- Automates d'arbres
- Termes
- Ascendant / Descendant
- Déterminisme
- Lemme d'itération
- Congruences
- Minimalité

Grammaires

Langages algébriques

Bibliographie

- [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/>

Arbres

Définition : Arbres

Soit $A_p = \{d_1, \dots, d_p\}$ un alphabet ordonné $d_1 < \dots < d_p$.

Un arbre étiqueté dans Σ et d'arité (au plus) p est une fonction partielle $t : A_p^* \rightarrow \Sigma$ dont le *domaine* est un langage $\text{dom}(t) \subseteq A_p^*$

- ▶ fermé par préfixe : $u \leq v$ et $v \in \text{dom}(t)$ implique $u \in \text{dom}(t)$,
- ▶ fermé par frère aîné : $d_i < d_j$ et $ud_j \in \text{dom}(t)$ implique $ud_i \in \text{dom}(t)$.

On note $T_p(\Sigma)$ l'ensemble des arbres **finis** d'arité au plus p sur l'alphabet Σ .

Exemples :

1. Arbre représentant l'expression logique

$$((x \rightarrow y) \wedge (\neg y \vee \neg z)) \wedge (z \vee \neg x)$$

2. Arbre représentant le programme

lire a; lire b; q := 0; r := a;
Tant que b ≤ r faire
q := q+1; r := r-b
Fin tant que

Arbres

Définition : Terminologie

La racine de l'arbre est le mot vide $\varepsilon \in \text{dom}(t)$.

Un nœud de l'arbre est un élément $u \in \text{dom}(t)$.

Une feuille de l'arbre est un nœud $u \in \text{dom}(t)$ tel que $ud_1 \notin \text{dom}(t)$.

La frontière $\text{Fr}(t)$ (ou mot des feuilles) de l'arbre t est la concaténation des étiquettes des feuilles de t .

L'arité d'un nœud interne $u \in \text{dom}(t)$ est $\max\{k \mid ud_k \in \text{dom}(t)\}$

L'arité d'une feuille est 0.

Les fils d'un nœud $u \in \text{dom}(t)$ d'arité k sont les nœuds $ud_1, \dots, ud_k \in \text{dom}(t)$.

Définition : $T_p(\Sigma)$ inductivement (notation préfixe)

- ▶ Si $a \in \Sigma$ alors $a() \in T_p(\Sigma)$ (ou simplement $a \in T_p(\Sigma)$)
- ▶ Si $a \in \Sigma$ et $t_1, \dots, t_k \in T_p(\Sigma)$ avec $1 \leq k \leq p$ alors $a(t_1, \dots, t_k) \in T_p(\Sigma)$

Automates d'arbres

Définition : Automate

Un automate d'arbres est un quadruplet $\mathcal{A} = (Q, \Sigma, \delta, F)$ où

- ▶ Q est un ensemble fini d'états
- ▶ Σ est un alphabet fini
- ▶ $\delta \subseteq \bigcup_p Q^p \times \Sigma \times Q$ est l'ensemble **fini** des transitions
- ▶ $F \subseteq Q$ est l'ensemble des états finaux.

Définition : Calcul, langage

- ▶ Un calcul de l'automate \mathcal{A} sur un Σ -arbre t est un Q -arbre ρ ayant même domaine que t et tel que pour tout $u \in \text{dom}(t)$ d'arité n , on a $(\rho(u \cdot d_1), \dots, \rho(u \cdot d_n), t(u), \rho(u)) \in \delta$.
- ▶ Le calcul est acceptant si $\rho(\varepsilon) \in F$.
- ▶ $\mathcal{L}(\mathcal{A})$ est l'ensemble des Σ -arbres acceptés par \mathcal{A} .
- ▶ Un langage d'arbre est reconnaissable s'il existe un automate d'arbres qui l'accepte.

Automates d'arbres

Exemples : Donner des automates pour les langages d'arbres suivants :

1. L'ensemble des arbres d'arité au plus p ayant un nombre pair de noeuds internes.
2. L'ensemble des arbres sur $\Sigma = \{a, b, c\}$ dont les noeuds internes sont d'arités 2 et étiquetés par c et la frontière est dans $(ab)^*$.
Peut-on généraliser aux expressions rationnelles arbitraires ?
3. L'ensemble des arbres d'arité au plus p dont les étiquettes de toutes les branches sont dans un langage rationnel fixé $L \subseteq \Sigma^*$.
4. L'ensemble des arbres d'arité au plus p dont au moins une branche est étiquetée par un mot d'un langage rationnel fixé $L \subseteq \Sigma^*$.

Grammaires et automates d'arbres

Théorème : du feuillage

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

Termes

Définition : Un terme est un arbre avec symboles **typés**

- ▶ \mathcal{F} un ensemble fini de symboles de fonctions avec arités.
 - ▶ On note \mathcal{F}_p les symboles d'arité p .
 - ▶ \mathcal{X} un ensemble de variables (arité 0) disjoint de \mathcal{F}_0 (les constantes).
 - ▶ $T(\mathcal{F}, \mathcal{X})$ ensemble des termes sur \mathcal{F} et \mathcal{X} défini inductivement par :
 - ▶ $\mathcal{F}_0 \cup \mathcal{X} \subseteq T(\mathcal{F}, \mathcal{X})$,
 - ▶ si $f \in \mathcal{F}_n$ ($n \geq 1$) et $t_1, \dots, t_n \in T(\mathcal{F}, \mathcal{X})$ alors $f(t_1, \dots, t_n) \in T(\mathcal{F}, \mathcal{X})$
- Remarque : on peut aussi utiliser une notation suffixe ou infixe parenthésée.
- ▶ $\text{Free}(t)$ est l'ensemble des variables de t .
 - ▶ $T(\mathcal{F})$ l'ensemble des termes qui ne contiennent pas de variable (termes clos).
 - ▶ Un terme t est linéaire s'il contient au plus une occurrence de chaque variable.
 - ▶ Hauteur : $H(x) = 0$ pour $x \in \mathcal{X}$ et $H(f) = 1$ pour $f \in \mathcal{F}_0$ et $H(f(t_1, \dots, t_n)) = 1 + \max(H(t_1), \dots, H(t_n))$.
 - ▶ Taille : $|x| = 0$ pour $x \in \mathcal{X}$ et $|f| = 1$ pour $f \in \mathcal{F}_0$ et $|f(t_1, \dots, t_n)| = 1 + |t_1| + \dots + |t_n|$.

Termes

Exemple : Expressions logiques

$\mathcal{F}_2 = \{\wedge, \vee, \rightarrow, \oplus, \dots\}$, $\mathcal{F}_1 = \{\neg\}$, $\mathcal{F}_0 = \{\top, \perp\}$, $\mathcal{X} = \{p, q, r\}$

$$\wedge(\vee(\neg(p), q), \vee(\neg(q), r)) = (\neg p \vee q) \wedge (\neg q \vee r)$$

Exemple : Expressions arithmétiques

$\mathcal{F}_2 = \{+, -, \times, /, \dots\}$, $\mathcal{F}_1 = \{\sin, \cos, \ln, !, \dots\}$,
 $\mathcal{F}_0 = \{0, \dots, 9\}$ et $\mathcal{X} = \{x, y, \dots\}$.

$$+(3, \times(2, !(x))) = 3 + (2 \times x!)$$

Arbres et termes

Un terme est un arbre

Un terme peut être vu comme un arbre t étiqueté dans $\mathcal{F} \cup \mathcal{X}$ tel que

- ▶ si $u \in \text{dom}(t)$ et $t(u) \in \mathcal{F}_n$ alors u est d'arité n .
- ▶ si $u \in \text{dom}(t)$ et $t(u) \in \mathcal{X}$ alors u est une feuille.

La hauteur d'un terme clos est la hauteur de l'arbre qui le représente.

La taille d'un terme clos est le nombre de noeuds de l'arbre qui le représente.

Exemples :

1. Soit \mathcal{F} un ensemble fini de symboles de fonctions avec arités et \mathcal{X} un ensemble fini de variables. Le langage d'arbres $T(\mathcal{F}, \mathcal{X})$ est reconnaissable.
2. Considérons $\mathcal{F}_2 = \{\wedge, \vee\}$, $\mathcal{F}_1 = \{\neg\}$, $\mathcal{F}_0 = \{\top, \perp\}$ et $\mathcal{X} = \emptyset$. L'ensemble des formules closes du calcul propositionnel qui s'évaluent à *vrai* est reconnaissable.
3. Considérons $\mathcal{F}_2 = \{\wedge, \vee\}$, $\mathcal{F}_1 = \{\neg\}$, $\mathcal{F}_0 = \{\top, \perp\}$ et $\mathcal{X} = \{p_1, \dots, p_n\}$ fini. L'ensemble des formules *satisfaisables* du calcul propositionnel est reconnaissable.

Arbres et termes

Un arbre est la projection d'un terme

Soit $t \in T_p(\Sigma)$ un Σ -arbre d'arité au plus p .

Soit $\mathcal{F} = \bigsqcup_{0 \leq i \leq p} \Sigma_i$ où Σ_i est une copie de Σ .

Soit t' l'arbre ayant même domaine que t et tel que si $u \in \text{dom}(t)$ est d'arité i et $t(u) = f$ alors $t'(u) = f_i$ est la copie de f dans Σ_i .

$t' \in T(\mathcal{F})$ est un terme clos et t est le projeté de t' .

Remarque :

Un arbre de dérivation n'est pas toujours un terme car les règles associées à une variable n'ont pas forcément une longueur fixe.

Exemple : $S \rightarrow aSb + ab$

Substitutions

Définition :

- ▶ Une substitution σ est une application d'un sous-ensemble fini de \mathcal{X} dans $T(\mathcal{F}, \mathcal{X})$.
- ▶ Si $\sigma = [t_1/x_1, \dots, t_n/x_n]$ est une substitution et t un terme alors $\sigma(t) = t[t_1/x_1, \dots, t_n/x_n]$ est défini inductivement par :
 - ▶ $\sigma(x_i) = t_i$ pour $1 \leq i \leq n$,
 - ▶ $\sigma(f) = f$ pour $f \in \mathcal{F}_0 \cup \mathcal{X} \setminus \{x_1, \dots, x_n\}$
 - ▶ $\sigma(f(s_1, \dots, s_k)) = f(\sigma(s_1), \dots, \sigma(s_k))$ pour $f \in \mathcal{F}_k$, $k \geq 1$.On dit que $t[t_1/x_1, \dots, t_n/x_n]$ est une *instance* de t .
- ▶ La substitution $\sigma = [t_1/x_1, \dots, t_n/x_n]$ est *close* si chaque t_i est clos.
- ▶ Si t_1, t_2 sont clos, alors $t[t_1/x_1, t_2/x_2] = t[t_1/x_1][t_2/x_2]$.
En général, $t[t_1/x_1, t_2/x_2] \neq t[t_1/x_1][t_2/x_2]$.

Exemple : Instances d'un terme

Soit \mathcal{F} un ensemble fini de symboles de fonctions avec arités et \mathcal{X} un ensemble fini de variables. Soit $s = f(g(x), f(y, a)) \in T(\mathcal{F}, \mathcal{X})$.

L'ensemble des termes $t \in T(\mathcal{F})$ qui sont instances de s est reconnaissable.

Généraliser à l'ensemble des instances d'un ensemble fini de termes linéaires.

Vision ascendante

Définition : calcul ascendant

Soit $\mathcal{A} = (Q, \Sigma, \delta, F)$ un automate d'arbres.

On voit δ comme une fonction $\delta : \bigcup_p Q^p \times \Sigma \rightarrow 2^Q$.

L'étiquetage d'un calcul est construit à partir des feuilles en remontant vers la racine.

Exemples :

1. Évaluation d'une expression logique close.
2. Instances du terme $s = f(g(x), f(y, a)) \in T(\mathcal{F}, \mathcal{X})$.

Définition : Déterminisme ascendant

Un automate $\mathcal{A} = (Q, \Sigma, \delta, F)$ est *déterministe ascendant* si $\delta : \bigcup_p Q^p \times \Sigma \rightarrow Q$ est une **fonction** (partielle si \mathcal{A} n'est pas complet).

Exercice :

Parmi les langages reconnaissables vus précédemment, quels sont ceux qui sont déterministes ascendants ?

Vision descendante

Définition : calcul descendant

Soit $\mathcal{A} = (Q, \Sigma, \delta, I)$ un automate d'arbres.

On voit δ comme une fonction $\delta : Q \times \Sigma \rightarrow 2^{\bigcup_p Q^p}$.

L'étiquetage d'un calcul est construit à partir de la racine en descendant vers les feuilles.

L'étiquette de la racine doit être dans I .

On dit que I est l'ensemble des états *initiaux*.

Exemples :

1. Instances du terme $s = f(g(x), f(y, a)) \in T(\mathcal{F}, \mathcal{X})$.
2. Évaluation d'une expression logique close.

Définition : Déterminisme descendant

Un automate $\mathcal{A} = (Q, \Sigma, \delta, I)$ est *déterministe descendant* s'il a un seul état initial et si $\delta : Q \times \Sigma \rightarrow \bigcup_p Q^p$ est une **fonction** (partielle si \mathcal{A} n'est pas complet).

Exercice :

Parmi les langages reconnaissables vus précédemment, quels sont ceux qui sont déterministes descendants ?

Automates déterministes

Théorème : Déterminisation

Soit \mathcal{A} un automate d'arbres. On peut effectivement construire un automate *déterministe ascendant* \mathcal{B} tel que $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{B})$.

Théorème : Clôture

La classe des langages d'arbres reconnaissables est effectivement close par union, intersection et complémentaire.

Proposition :

La classe des langages d'arbres reconnaissables par un automate déterministe descendant est strictement incluse dans la classe des langages d'arbres reconnaissables.

Exemple : le langage $\{f(a, b), f(b, a)\}$ n'est pas déterministe descendant.

Automates avec ε -transitions

Définition : ε -transitions

▶ L'automate peut avoir des transitions du type $p \xrightarrow{\varepsilon} q : \delta_\varepsilon \subseteq Q \times Q$.

▶ Il faut changer la définition des calculs.

Vision ascendante avec $\delta : \bigcup_p Q^p \times \Sigma \rightarrow 2^Q$

$$\delta'(q_1, \dots, q_p, a) = \delta_\varepsilon^*(\delta(q_1, \dots, q_p, a))$$

▶ On peut éliminer les ε -transitions

▶ les ε -transitions peuvent être utiles dans les preuves et les constructions sur les automates d'arbres.

Concaténation d'arbres

Définition : Arbre à trou

Un Σ -arbre à trou t est un $(\Sigma \cup \{\square\})$ -arbre ayant un unique noeud étiqueté \square et ce noeud doit être une feuille : $t : A^* \rightarrow \Sigma \cup \{\square\}$, $t^{-1}(\square) = \{u\}$ et u est une feuille. On note $T_{\square}(\Sigma)$ l'ensemble des Σ -arbres à trou.

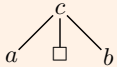

Définition : Concaténation

Soit t un Σ -arbre avec un trou en u et soit t' un Σ -arbre (avec ou sans trou). La concaténation $t \cdot t'$ est le Σ -arbre (avec ou sans trou) défini par

$$v \mapsto \begin{cases} t(v) & \text{si } u \not\leq v \\ t'(u^{-1}v) & \text{si } u \leq v \end{cases}$$

L'ensemble $T_{\square}(\Sigma)$ est un monoïde avec comme élément neutre \square .

Exemple :

Soit t_1 l'arbre  et t_2 l'arbre .

Le langage $L = t_1^* t_2$ est reconnaissable.

Remarque : le langage $\text{Fr}(L)$ des mots de feuilles de L est $\{a^n b^n \mid n > 0\}$.

Lemme d'itération

Lemme : itération (pumping)

Soit L un langage d'arbres reconnaissable.

$\exists n \geq 0, \forall t \in L$, si $H(t) > n$ alors $\exists t_1, t_2 \in T_{\square}(\Sigma)$, $\exists t_3 \in T(\Sigma)$ tels que

- ▶ $t_2 \neq \square$, $t = t_1 \cdot t_2 \cdot t_3$, $t_1(t_2)^* t_3 \subseteq L$,
- ▶ $\text{prof}_{\square}(t_1) + \text{prof}_{\square}(t_2) \leq n$ ou $\text{prof}_{\square}(t_2) + h(t_3) \leq n$.

Exemples :

- ▶ $L = \{f(g^n(a), g^n(a)) \mid n > 0\}$ n'est pas reconnaissable.
- ▶ L'ensemble des instances de $f(x, x)$ n'est pas reconnaissable.

▶ Associativité.

Soit $\mathcal{F}_2 = \{f\}$ et $\mathcal{F}_0 = \{a, b\}$.

Un langage $L \subseteq T(\mathcal{F})$ est associativement clos si il est fermé par la

congruence engendrée par $f(f(x, y), z) = f(x, f(y, z))$.

Soit $t_1 = f(f(a, \square), b)$ et $t_2 = f(a, b)$.

La clôture associative de $t_1^* t_2$ n'est pas reconnaissable.

Congruences

Définition :

Soient $a \in \Sigma$ et $t_1, \dots, t_n \in T(\Sigma)$. L'arbre $t = a(t_1, \dots, t_n)$ est défini par

- ▶ $\text{dom}(t) = \{\varepsilon\} \cup \bigcup_{i=1}^n d_i \text{dom}(t_i)$,
- ▶ $t(\varepsilon) = a$ et la racine de t est d'arité n ,
- ▶ $t(d_i v) = t_i(v)$ pour $1 \leq i \leq n$ et $v \in \text{dom}(t_i)$.

Définition : Congruence (en haut)

Une relation d'équivalence \equiv sur $T_p(\Sigma)$ est une congruence si pour tous $a \in \Sigma$, et $t_1, \dots, t_n, s_1, \dots, s_n \in T_p(\Sigma)$ avec $n \leq p$ on a

$$(\forall 1 \leq i \leq n, s_i \equiv t_i) \implies a(s_1, \dots, s_n) \equiv a(t_1, \dots, t_n)$$

Proposition :

Une relation d'équivalence \equiv sur $T_p(\Sigma)$ est une congruence si et seulement si pour tout $r \in T_{p, \square}(\Sigma)$ et tous $s, t \in T_p(\Sigma)$, on a $s \equiv t$ implique $r \cdot s \equiv r \cdot t$.

Congruence syntaxique

Définition : Résiduels et Congruence syntaxique

Soit $L \subseteq T_p(\Sigma)$ un langage d'arbres et $s \in T_p(\Sigma)$. Le résiduel de L par s est

$$L \setminus s = \{r \in T_{p, \square}(\Sigma) \mid r \cdot s \in L\}.$$

La congruence syntaxique \equiv_L associée à L est définie par $s \equiv_L t$ si $L \setminus s = L \setminus t$.

Remarque :

La relation d'équivalence \equiv_L est bien une congruence et sature le langage L .

Lemme :

Soit $\mathcal{A} = (Q, \Sigma, \delta, F)$ un automate DC (déterministe, complet) reconnaissant L .

Pour $t \in T_p(\Sigma)$, on note $\mathcal{A}(t)$ l'état à la racine du run de \mathcal{A} sur t .

Pour $s, t \in T_p(\Sigma)$, on a $\mathcal{A}(s) = \mathcal{A}(t)$ implique $s \equiv_L t$.

Donc \equiv_L est d'index fini.

Congruence et reconnaissabilité

Définition :

Soit \equiv une congruence d'index fini qui sature $L \subseteq T_p(\Sigma)$.

On note $[t]$ la classe pour \equiv d'un arbre $t \in T_p(\Sigma)$.

On définit l'automate $\mathcal{A}_{\equiv} = (Q, \Sigma, \delta, F)$ par :

- ▶ $Q = T_p(\Sigma)/\equiv$,
- ▶ $\delta([t_1], \dots, [t_n], a) = [a(t_1, \dots, t_n)]$ (bien définie car \equiv congruence),
- ▶ $F = \{[t] \mid t \in L\}$.

Pour la congruence syntaxique, on note simplement $\mathcal{A}_L = (Q_L, \Sigma, \delta_L, F_L) = \mathcal{A}_{\equiv_L}$.

Lemme :

L'automate \mathcal{A}_{\equiv} est DAC (déterministe, accessible, complet) et reconnaît L .

Théorème : Myhill-Nerode

Soit $L \subseteq T_p(\Sigma)$. Les conditions suivantes sont équivalentes :

1. L est reconnaissable,
2. L est saturé par une congruence d'index fini,
3. la congruence syntaxique \equiv_L est d'index fini.

Automate minimal

Lemme : Quotient

Soit $\mathcal{A} = (Q, \Sigma, \delta, F)$ un automate DAC reconnaissant L .

On définit φ de \mathcal{A} dans \mathcal{A}_L par :

$$\begin{aligned} \varphi : Q &\rightarrow Q_L \\ q &\mapsto [t] \quad \text{si } \mathcal{A}(t) = q \end{aligned}$$

L'application φ est bien définie. C'est un morphisme surjectif de \mathcal{A} sur \mathcal{A}_L .

L'équivalence associée à φ est définie par $q \sim q'$ si $\varphi(q) = \varphi(q')$.

Elle est appelée équivalence de Nerode de l'automate \mathcal{A} .

\mathcal{A}_L est isomorphe au quotient de Nerode \mathcal{A}/\sim .

Théorème : automate minimal

Soit $L \subseteq T_p(\Sigma)$ un langage reconnaissable.

1. On obtient \mathcal{A}_L comme quotient de Nerode de tout automate DAC reconnaissant L .
2. \mathcal{A}_L est l'unique à isomorphisme près automate DAC minimal reconnaissant L .

Calcul de l'équivalence de Nerode

Lemme :

Soit $\mathcal{A} = (Q, \Sigma, \delta, F)$ un automate DAC reconnaissant $L \subseteq T_p(\Sigma)$.

Pour $q \in Q$, soit $\mathcal{A}_q = (Q, \Sigma \cup \{\square\}, \delta \cup \{(\square, q)\}, F)$ et $\mathcal{L}_{\square}(\mathcal{A}_q) = \mathcal{L}(\mathcal{A}_q) \cap T_{\square}(\Sigma)$.

Soit $t \in T_p(\Sigma)$ et $q = \mathcal{A}(t) \in Q$. On a $\mathcal{L}_{\square}(\mathcal{A}_q) = L \setminus t$.

Donc $q \sim q'$ ssi $\mathcal{L}_{\square}(\mathcal{A}_q) = \mathcal{L}_{\square}(\mathcal{A}_{q'})$.

Proposition :

Soit $\mathcal{A} = (Q, \Sigma, \delta, F)$ un automate DAC reconnaissant $L \subseteq T_p(\Sigma)$.

On définit \sim_m par : $q \sim_m q'$ ssi $\mathcal{L}_{\square}^{\leq m}(\mathcal{A}_q) = \mathcal{L}_{\square}^{\leq m}(\mathcal{A}_{q'})$

où $\mathcal{L}_{\square}^{\leq m}(\mathcal{A}_q) = \mathcal{L}(\mathcal{A}_q) \cap T_{\square}^{\leq m}(\Sigma)$ et $T_{\square}^{\leq m}(\Sigma) = \{r \in T_{\square}(\Sigma) \mid \text{prof}_{\square}(r) \leq m\}$.

L'équivalence de Nerode se calcule par approximations de la façon suivante :

- ▶ $q \sim_0 q'$ si $q, q' \in F$ ou $q, q' \notin F$
- ▶ $q \sim_{m+1} q'$ si $q \sim_m q'$ et $\forall a \in \Sigma$ et $\forall q_1, \dots, q_{i-1}, q_{i+1}, \dots, q_n \in Q$ on a $\delta(q_1, \dots, q_{i-1}, q, q_{i+1}, \dots, q_n, a) \sim_m \delta(q_1, \dots, q_{i-1}, q', q_{i+1}, \dots, q_n, a)$
- ▶ $\sim = \bigcap_{m \geq 0} \sim_m = \sim_{|Q|}$.

Exercices

Exercice : Morphisme

Montrer que $L \subseteq T(\mathcal{F})$ est reconnaissable ssi il existe une \mathcal{F} -algèbre finie $A(\mathcal{F})$ telle que $L = \varphi^{-1}(\varphi(L))$ où $\varphi : T(\mathcal{F}) \rightarrow A(\mathcal{F})$ est le morphisme canonique.

Exercice : Problèmes de décision et complexité

Lire la section 7 du chapitre 1 du TATA.

Plan

Introduction

Langages reconnaissables

Automates d'arbres

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

Langages algébriques

Automates à pile

Analyse syntaxique

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.
- [6] Jean Berstel.
Transduction and context free languages.
Teubner, 1979.
- [7] Olivier Carton.
Langages formels, calculabilité et complexité.
Vuibert, 2008.

Bibliographie

- [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.

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 $\{a^{2^n} \mid n > 0\}$

$$\begin{array}{llll} S \rightarrow DXaF & Xa \rightarrow aaX & XF \rightarrow YF & XF \rightarrow Z \\ aY \rightarrow Ya & DY \rightarrow DX & aZ \rightarrow Za & DZ \rightarrow \varepsilon \end{array}$$

Définition : Dérivation

$\alpha \in (\Sigma \cup V)^*$ se dérive en $\beta \in (\Sigma \cup V)^*$, noté $\alpha \rightarrow \beta$, 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 \rightarrow^* la clôture réflexive et transitive de \rightarrow .

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 $\hat{\mathcal{L}}_G(\alpha) = \{\beta \in (\Sigma \cup V)^* \mid \alpha \xrightarrow{*} \beta\}$.

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 $\{a^{2^n} \mid n > 0\}$

$$\begin{array}{llll} S \rightarrow DTF & T \rightarrow XT & Xaa \rightarrow aaXa & Daaa \rightarrow aaDaa \\ S \rightarrow aa & T \rightarrow aa & XaF \rightarrow aaF & DaaF \rightarrow aaaa \end{array}$$

Remarque :

Le langage engendré par une grammaire contextuelle est propre.

Si on veut engendrer le mot vide on peut ajouter $\hat{S} \rightarrow S + \varepsilon$.

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 $\{a^{2^n} \mid n > 0\}$

$$\begin{array}{llll} S \rightarrow aTa & T \rightarrow XT & XA \rightarrow XY & Xa \rightarrow AAa \\ S \rightarrow aa & T \rightarrow AA & XY \rightarrow ZY & ZA \rightarrow AAA \\ & & ZY \rightarrow ZX & aA \rightarrow aa \end{array}$$

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 donné 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.

Grammaires algébriques

Définition : Grammaire hors contexte ou algébrique ou de type 2

Une grammaire $G = (\Sigma, V, P, S)$ est *hors contexte* ou *algébrique* si $P \subseteq V \times (\Sigma \cup V)^*$ (sous ensemble *fini*).

Un langage est de type 2 (ou hors contexte ou algébrique) s'il peut être engendré par une grammaire hors contexte.

On note Alg la famille des langages algébriques.

Exemples :

1. Le langage $\{a^n b^n \mid n \geq 0\}$ est algébrique.
2. Expressions complètement parenthésées.

Lemme : fondamental

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

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

Langages de Dyck

Définition : D_n^*

Soit $\Sigma_n = \{a_1, \dots, a_n\} \cup \{\bar{a}_1, \dots, \bar{a}_n\}$ l'alphabet formé de n paires de parenthèses.
Soit $G_n = (\Sigma_n, V, P_n, S)$ la grammaire définie par $S \rightarrow a_1 S \bar{a}_1 S + \dots + a_n S \bar{a}_n S + \varepsilon$.
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 $\text{clot}(L) = \{v \in \Sigma^* \mid \exists w \in L, w \xrightarrow{*} v \text{ dans } R_n\}$.
Montrer que si L est reconnaissable, alors $\text{clot}(L)$ aussi.
On définit la réduction $\text{red}(L) = \{v \in \text{clot}(L) \mid v \not\xrightarrow{*} \varepsilon \text{ dans } R_n\}$.
Montrer que si L est reconnaissable, alors $\text{red}(L)$ 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).

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 récurrents.
5. les langages récurrents sont strictement contenus dans les langages récursivement énumérables (type 0).

Plan

Introduction

Langages reconnaissables

Automates d'arbres

Grammaires

5 Langages algébriques

- Arbres de dérivation
- Propriétés de clôture
- Formes normales
- Problèmes sur les langages algébriques
- Équations algébriques

Automates à pile

Analyse syntaxique

Arbres (rappel)

Définition : Arbres

Soit $A_p = \{d_1, \dots, d_p\}$ un alphabet ordonné $d_1 \prec \dots \prec d_p$.

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

- ▶ fermé par préfixe : $u \leq v$ et $v \in \text{dom}(t)$ implique $u \in \text{dom}(t)$,
- ▶ fermé par frère aîné : $d_i \prec d_j$ et $ud_j \in \text{dom}(t)$ implique $ud_i \in \text{dom}(t)$.

Définition : Terminologie

La racine de l'arbre est le mot vide $\varepsilon \in \text{dom}(t)$.

Un nœud de l'arbre est un élément $u \in \text{dom}(t)$.

Une feuille de l'arbre est un nœud $u \in \text{dom}(t)$ tel que $ud_1 \notin \text{dom}(t)$.

L'arité d'un nœud $u \in \text{dom}(t)$ est le plus grand entier k tel que $ud_k \in \text{dom}(t)$ ($k = 0$ si u est une feuille).

Les fils d'un nœud $u \in \text{dom}(t)$ d'arité k sont les nœuds $ud_1, \dots, ud_k \in \text{dom}(t)$.

La frontière $\text{Fr}(t)$ (ou mot des feuilles) de l'arbre t est la concaténation des étiquettes des feuilles de t .

Arbres de dérivation

Définition :

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

Un arbre de dérivation pour G est un arbre t étiqueté dans $V \cup \Sigma$ 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 $\alpha_1, \dots, \alpha_k$ alors $(x, \alpha_1 \dots \alpha_k) \in P$.

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

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 $\text{rac}(t) = x$ et $\text{Fr}(t) = \alpha$.
2. Si t un arbre de dérivation de G alors il existe une dérivation **gauche** $\text{rac}(t) \xrightarrow{*} \text{Fr}(t)$ dans G .

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 et arbres de dérivation.
- ▶ Si la grammaire est linéaire, il y a bijection entre dérivations et arbres de dérivations.

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

Proposition :

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

Ambiguïté

Exercice : if then else

Montrer que la grammaire suivante est ambiguë.

$$S \rightarrow \text{if } c \text{ then } S \text{ else } S \mid \text{if } c \text{ then } S \mid a$$

Montrer que le langage engendré n'est pas ambigu.

Grammaires et automates d'arbres

Théorème : du feuillage

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

Lemme d'itération

Théorème : Bar-Hillel, Perles, Shamir ou Lemme d'itération

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

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

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 :

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 $\sigma : A \rightarrow \mathcal{P}(B^*)$ est algébrique si $\forall a \in A, \sigma(a) \in \text{Alg}$

Définition : Projection

La projection de A sur $B \subseteq A$ est le morphisme $\pi : A^* \rightarrow B^*$ 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) $\tau : A^* \rightarrow \mathcal{P}(B^*)$ est la composée d'un morphisme inverse, d'une intersection avec un rationnel et d'un morphisme.

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

Soient A, B, C trois alphabets, $K \in \text{Rat}(C^*)$ et $\varphi : C^* \rightarrow A^*$ et $\psi : C^* \rightarrow B^*$ deux morphismes. L'application $\tau : A^* \rightarrow \mathcal{P}(B^*)$ 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 $\tau : A^* \rightarrow \mathcal{P}(B^*)$ 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 $A^* \times B^*$).

Formes normales

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

Corollaire :

Soit $G = (\Sigma, V, P, S)$ une grammaire telle que $\mathcal{L}_G(S) \neq \emptyset$. On peut effectivement calculer une grammaire réduite équivalente $G' = (\Sigma, V', P', S)$ ($\mathcal{L}_G(S) = \mathcal{L}_{G'}(S)$).
Preuve : Restreindre aux variables productives, puis aux variables accessibles.

Formes normales

Définition : Grammaires propres

La grammaire $G = (\Sigma, V, P, S)$ est propre si elle ne contient pas de règle de la forme $x \rightarrow \varepsilon$ ou $x \rightarrow y$ 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 $\varepsilon \in \mathcal{L}_G(x)$ (PTIME).

Proposition :

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

On peut construire une grammaire propre G' qui engendre $\mathcal{L}_G(S) \setminus \{\varepsilon\}$ (PTIME).

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 .

Formes normales

Exercice :

Montrer que le problème du mot pour les grammaires est dans PTIME [9, p. 139].

Exercice :

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

Montrer que l'on peut décider si $\mathcal{L}_G(S)$ est fini.

Donner un algorithme PTIME pour ce problème.

Définition : Forme normale quadratique

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

$$P \subseteq V \times (V \cup \Sigma)^{\leq 2}$$

Proposition :

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

On peut effectivement construire une grammaire équivalente G' en forme normale quadratique (PTIME).

Formes normales

Définition : 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 :

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

On peut effectivement construire une grammaire équivalente G' en forme normale de Chomsky faible ou forte (PTIME).

Remarques :

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

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 à une FNG.

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 $V = \{x_1, \dots, x_n\}$.

Pour $i, j \in \{1, \dots, n\}$ 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 $x_j \rightarrow \sum_i x_i \alpha_{i,j} + \beta_j$ pour $1 \leq j \leq n$.

On peut écrire P vectoriellement : $X \rightarrow XA + B$
avec $X = (x_1, \dots, x_n)$, $B = (\beta_1, \dots, \beta_n)$ 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

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

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

Proposition : Equivalence des grammaires

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

Forme normale de Greibach

Remarque : Grammaire propre

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

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

donc les règles $X \rightarrow BX + B$ de G' sont en FNG.

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 FNG.
- ▶ Si G est propre et en FN quadratique, alors G'' est en FNGQ.

Exemples : Mettre les grammaires suivantes en FNG(Q)

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

Problèmes décidables

Proposition :

Soit G une grammaire algébrique.

- ▶ On peut décider si le langage engendré par G est vide, fini ou infini (PTIME).
- ▶ On peut décider si un mot est engendré par G (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 ?
- ▶ \bar{L} est-il algébrique ?
- ▶ $L \cap L'$ est-il algébrique ?

Équations algébriques

Définition : Système d'équations algébriques

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

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

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

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

Exemple :

$\bar{L} = (a^+b^+, ab^*)$ 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 :

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

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

Exercice : Grammaire et équations algébriques

Soit $G = (\Sigma, V, Q)$ une grammaire avec $V = \{X_1, \dots, X_n\}$.

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

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 $X = aXbX + \varepsilon$.

\mathcal{L} est l'unique solution de $X = aXX + b$.

On en déduit $\mathcal{L} = D_1^*b$.

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 $\overline{X} = (X_1, \dots, X_n)$ et $\overline{Y} = (Y_1, \dots, Y_m)$.

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

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

Introduction

Langages reconnaissables

Automates d'arbres

Grammaires

Langages algébriques

6 Automates à pile

- Définition et exemples
- Modes de reconnaissance
- Lien avec les langages algébriques
- Langages déterministes

Analyse syntaxique

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
- ▶ $T \subseteq Q \times Z \times (\Sigma \cup \{\varepsilon\}) \times Q \times Z^*$ ensemble fini de transitions
- ▶ $(q_0, z_0) \in Q \times Z$ configuration initiale
- ▶ $F \subseteq Q$ 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

- ▶ $\mathcal{T} = (Q \times Z^*, T', (q_0, z_0))$ avec $T' = \{(p, hz) \xrightarrow{x} (q, hu) \mid (p, z, x, q, u) \in T\}$.
- ▶ Une configuration de \mathcal{A} est un état $(p, h) \in Q \times Z^*$ de \mathcal{T} .
- ▶ $\mathcal{L}(\mathcal{A}) = \{w \in \Sigma^* \mid \exists (q_0, z_0) \xrightarrow{w} (q, h) \text{ dans } \mathcal{T} \text{ avec } q \in F\}$.

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 $\{w\bar{w} \mid w \in \Sigma^*\}$ 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 $T' \subseteq Q' \times Z' \times (\Sigma \cup \{\varepsilon\}) \times Q' \times Z'^{\leq 2}$.
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 :

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

1. Si $p, h \xrightarrow{w} p', h'$ est un calcul de \mathcal{A} et $g \in Z^*$ alors $p, gh \xrightarrow{w} p', gh'$ est aussi un calcul de \mathcal{A} .
2. Si $p_0, h_0 \xrightarrow{a_1} p_1, h_1 \cdots \xrightarrow{a_n} p_n, h_n$ est un calcul de \mathcal{A} tel que $|h_i| > k$ pour $0 \leq i < n$ alors il existe $g \in Z^k$ tel que $h_i = gh'_i$ pour $0 \leq i \leq n$ et $p_0, h'_0 \xrightarrow{a_1} p_1, h'_1 \cdots \xrightarrow{a_n} p_n, h'_n$ est un calcul de \mathcal{A} .
3. $p, gh \xrightarrow{w} q, \varepsilon$ est un calcul de \mathcal{A} ssi il existe deux calculs de \mathcal{A} : $p, h \xrightarrow{w_1} r, \varepsilon$ et $r, g \xrightarrow{w_2} q, \varepsilon$ avec $w = w_1 w_2$ et $n = n_1 + n_2$.

Mots de pile

Lemme : Effacement

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

$$X = \{(p, x, q) \in Q \times Z \times Q \mid \exists (p, x) \rightarrow^* (q, \varepsilon) \text{ dans } \mathcal{T}\}$$

Proposition : Mots de pile

Soit $\mathcal{A} = (Q, \Sigma, Z, T, q_0, z_0)$ un automate à pile.
Pour $(p, x, q) \in Q \times Z \times Q$, on note

$$\mathcal{L}(p, x, q) = \{h \in Z^* \mid \exists (p, x) \rightarrow^* (q, h)\}$$

l'ensemble des mots de pile dans l'état q accessibles à partir de (p, x) .
On peut effectivement construire un automate fini \mathcal{B} qui reconnaît $\mathcal{L}(p, x, q)$.

Calculs d'accessibilité

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. $Y = \{(p, x, q, y) \in Q \times Z \times Q \times Z \mid \exists (p, x) \rightarrow^* (q, hy) \text{ dans } \mathcal{T}\}$
2. $V = \{(p, x) \in Q \times Z \mid \exists (p, x) \rightarrow^* \text{ dans } \mathcal{T}\}$
3. $W = \{(p, x, q, y) \in Q \times Z \times Q \times Z \mid \exists (p, x) \rightarrow^* (q, y) \text{ dans } \mathcal{T}\}$
4. $X' = \{(p, x, q) \in Q \times Z \times Q \mid \exists (p, x) \xrightarrow{\varepsilon} (q, \varepsilon) \text{ dans } \mathcal{T}\}$
5. $Y' = \{(p, x, q, y) \in Q \times Z \times Q \times Z \mid \exists (p, x) \xrightarrow{\varepsilon} (q, hy) \text{ dans } \mathcal{T}\}$
6. $V' = \{(p, x) \in Q \times Z \mid \exists (p, x) \xrightarrow{\varepsilon} \text{ dans } \mathcal{T}\}$
7. $W' = \{(p, x, q, y) \in Q \times Z \times Q \times Z \mid \exists (p, x) \xrightarrow{\varepsilon} (q, y) \text{ dans } \mathcal{T}\}$

Acceptation généralisée

Définition :

Soit $\mathcal{A} = (Q, \Sigma, Z, T, q_0, z_0)$ un automate à pile et $K \subseteq Q \times Z^*$ tel que pour tout $q \in Q$, $K_q = \{h \in Z^* \mid (q, h) \in K\}$ est reconnaissable.

Le langage reconnu par \mathcal{A} avec *acceptation généralisée par K* est

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

Cas particuliers :

- ▶ $K = F \times Z^*$: acceptation classique **par état final**.
- ▶ $K = Q \times \{\varepsilon\}$: acceptation **par pile vide**.
- ▶ $K = F \times \{\varepsilon\}$: acceptation **par pile vide et état final**.
- ▶ $K = Q \times Z^*Z'$ 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.

Acceptation généralisée

Proposition : équivalence acceptation

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

Exercice :

Montrer que tous les modes d'acceptation définis 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* ($T \subseteq Z \times \Sigma \times Z^{\leq 2}$).

Langages déterministes

Définition : Automate à pile déterministe

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

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

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

Exemples :

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.

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, α) 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, α) on a

1. $\alpha \neq \varepsilon$, et donc on peut écrire $\alpha = \beta x$ avec $x \in Z$,
2. $(p, x) \xrightarrow{\varepsilon}$ ou $\forall a \in \Sigma, (p, x) \xrightarrow{a}$,
3. $(p, x) \xrightarrow{\varepsilon}^*$.

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) $(q_0, z_0) \xrightarrow{w}^* (p, \alpha) \xrightarrow{\varepsilon}$ 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

1. $(q'_0, \perp) \xrightarrow{\varepsilon} (q_0, \perp z_0)$, et $(p, \perp) \xrightarrow{\varepsilon} (d, \perp)$ pour $p \in Q' \setminus \{q'_0\}$,
2. Si pour $a \in \Sigma$ on a $(p, x) \xrightarrow{a} (q, \alpha) \in T$ alors $(p, x) \xrightarrow{a} (q, \alpha) \in T'$,
3. Si pour $a \in \Sigma$ on a $(p, x) \not\xrightarrow{a}$ et $(p, x) \xrightarrow{\varepsilon}$ dans \mathcal{A} alors $(p, x) \xrightarrow{a} (d, x) \in T'$,
4. Si $(p, x) \xrightarrow{\varepsilon} (q, \alpha) \in T$ et $(p, x) \not\xrightarrow{\varepsilon}$ alors $(p, x) \xrightarrow{\varepsilon} (q, \alpha) \in T'$,
5. Si $(p, x) \xrightarrow{\varepsilon}$ et $\exists (p, x) \xrightarrow{\varepsilon}^* (q, \alpha)$ avec $q \in F$ alors $(p, x) \xrightarrow{\varepsilon} (f, x) \in T'$,
6. Si $(p, x) \xrightarrow{\varepsilon}$ et $\forall (p, x) \xrightarrow{\varepsilon}^* (q, \alpha)$ on a $q \notin F$ alors $(p, x) \xrightarrow{\varepsilon} (d, x) \in T'$.
7. $(d, x) \xrightarrow{a} (d, x)$ et $(f, x) \xrightarrow{a} (d, x)$ pour $x \in Z'$ et $a \in \Sigma$,

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 $\mathcal{A} = (Q, \Sigma, Z, T, q_0, z_0, F, K)$ un automate à pile déterministe reconnaissant par sommet de pile et état final (une configuration $(q, \alpha z)$ est acceptante si $(q, z) \in K \subseteq Q \times Z$). 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 : $(p, x) \xrightarrow{\varepsilon} (q, \varepsilon)$.

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

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 ?

Plan

Introduction

Langages reconnaissables

Automates d'arbres

Grammaires

Langages algébriques

Automates à pile

- 7 Analyse syntaxique
 - Analyse descendante (LL)
 - Analyse ascendante (LR)
 - Analyseur SLR
 - Analyseur LR(1)

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.

Application à l'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).

Application à l'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 ($x \xrightarrow{+} x\alpha$), 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.

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. $G_1 : S \rightarrow aSb + ab$.

2. $G_2 : \begin{cases} E \rightarrow E + T \mid T \\ T \rightarrow T * F \mid F \\ F \rightarrow (E) \mid a \mid b \mid c \end{cases}$

Définition :

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

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. On peut lever le non déterminisme de l'automate associé à la grammaire G_1 en regardant les 2 prochaines lettres.
2. On ne peut pas lever le non déterminisme de l'automate associé à la grammaire G_2 en regardant les k prochaines lettres.

Analyse descendante First_k

Définition : First

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

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

Remarque :

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

Exemple :

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

Remarque :

Pour $\alpha \in (\Sigma \cup V)^*$, $\text{First}_0(\alpha) = \{\varepsilon\}$ 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

$$X_{m+1}(x) = \bigcup_{x \rightarrow \alpha_1 \dots \alpha_n \in P} \text{First}_k(X_m(\alpha_1) \dots X_m(\alpha_n))$$

Proposition : Point fixe ($k > 0$)

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

Ceci fournit un algorithme pour calculer $\text{First}_k(\alpha)$ pour $\alpha \in \Sigma \cup V$.

Pour $\alpha \in (\Sigma \cup V)^*$ on utilise la remarque.

En particulier, $\text{First}_k(\varepsilon) = \{\varepsilon\}$.

Analyse descendante $LL(k)$

Définition : $LL(k)$

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

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

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 $LL(2)$ mais pas $LL(1)$.
2. La grammaire G_2 n'est pas $LL(k)$.

Analyse descendante $LL(k)$

Exercices :

- ▶ Montrer que si l'automate expansion/vérification associé à une grammaire est déterministe, alors la grammaire est $LL(0)$.
- ▶ Montrer qu'une grammaire $LL(0)$ engendre au plus un mot.
- ▶ Montrer que si G est en FNPG et que pour toutes règles $x \rightarrow a\alpha$ et $x \rightarrow b\beta$ avec $a, b \in \Sigma$ on a $a \neq b$ ou $\alpha = \beta$, alors G est $LL(1)$.
- ▶ Montrer que la réciproque est fausse.
- ▶ Montrer qu'un langage rationnel admet une grammaire $LL(1)$.
- ▶ Montrer qu'un langage $LL(k)$ est non-ambigu.
- ▶ Montrer que, étant donné une grammaire G et un entier k , on peut décider si G est $LL(k)$.
- ▶ Montrer que, étant données deux grammaires $LL(k)$, on peut décider si elles engendrent le même langage.

Analyse descendante $LL(k)$

Remarques :

- ▶ La hiérarchie des langages $LL(k)$ est stricte.
- ▶ Étant donnée une grammaire G , on ne peut pas décider s'il existe un entier k tel que G soit $LL(k)$.
- ▶ Étant donnée une grammaire G , on ne peut pas décider s'il existe une grammaire équivalente qui soit $LL(1)$.

Exemple :

On peut transformer la grammaire G_2 en une grammaire $LL(1)$ équivalente. Il suffit de supprimer la récursivité gauche.

$$G'_2 = \begin{cases} E \rightarrow TE' & E' \rightarrow +TE' \mid \varepsilon \\ T \rightarrow FT' & T' \rightarrow *FT' \mid \varepsilon \\ F \rightarrow (E) \mid a \mid b \mid c \end{cases}$$

Follow

Définition : Follow

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

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

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

Exemple :

Calculer $\text{Follow}_1(x)$ 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 $x \neq S$
- ▶ $Y_{m+1}(x) = Y_m(x) \cup \bigcup_{y \rightarrow \alpha x \beta \in P} \text{First}_k(\beta Y_m(y))$

Proposition : Point fixe

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

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

Fortement LL

Définition : Fortement $\text{LL}(k)$

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

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

Exemple :

1. La grammaire G_1 est fortement $\text{LL}(2)$.
2. La grammaire G'_2 est fortement $\text{LL}(1)$.

Fortement LL

Proposition :

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

Exemple :

La grammaire

$$G_3 = \begin{cases} S \rightarrow axaa \mid bxba \\ x \rightarrow b \mid \varepsilon \end{cases}$$

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

Proposition :

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

Table d'analyse fortement LL

Définition : Table d'analyse fortement $\text{LL}(k)$

Soit G une grammaire fortement $\text{LL}(k)$.

On définit $M(x, v)$ pour $x \in V$ et $v \in \Sigma^{\leq k}$ par

$$M(x, v) = \begin{cases} x \xrightarrow{\varepsilon} \tilde{\alpha} & \text{si } x \rightarrow \alpha \in P \text{ et } v \in \text{First}_k(\alpha \text{Follow}_k(x)), \\ \text{erreur} & \text{sinon.} \end{cases}$$

Exemple :

1. Construire la table d'analyse $\text{LL}(2)$ de la grammaire G_1 .
2. Construire la table d'analyse $\text{LL}(1)$ de la grammaire G'_2 .

Exercice : Langage de Dyck

Montrer que la grammaire usuelle pour le langage de Dyck D_n^* sur n paires de parenthèses est $\text{LL}(1)$. Donner sa table d'analyse $\text{LL}(1)$.

$$S \rightarrow \varepsilon \mid a_1 S b_1 S \mid \dots \mid a_n S b_n S$$

Analyseur fortement LL

Définition : Analyseur fortement LL(k)

Soit G une grammaire fortement LL(k).

L'analyseur LL(k) de G est l'automate à pile simple **déterministe** qui accepte par pile vide, qui **regarde k lettres à l'avance (lookahead)** et dont les **expansions sont pilotées** par la table d'analyse de G .

Proposition : Correction

Soit G une grammaire fortement LL(k).

L'analyseur fortement LL(k) de G accepte exactement $\mathcal{L}_G(S)$.

Exercice :

Transformer l'analyseur fortement LL(k) de G en un automate à pile déterministe classique (sans lookahead).

Exercice :

1. Construire un analyseur **déterministe** avec lookahead pour une grammaire LL(k).
2. Montrer qu'un langage LL(k) est déterministe.

Analyse ascendante (LR)

Définition : Automate shift/reduce (LR)

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

On construit un automate à pile **généralisé** simple (non déterministe) \mathcal{B} .

Alphabet de pile : $\Sigma \cup V$. 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 .

Exemples :

1. $G_1 : S \rightarrow aSb \mid ab$
2. $G_2 : E \rightarrow E + T \mid T, T \rightarrow T * F \mid F, F \rightarrow (E) \mid \text{id}$

Définition :

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

Analyse ascendante (LR)

Lemme :

$\forall \alpha, \beta \in (\Sigma \cup V)^*, \forall u \in \Sigma^*$, sont équivalents

1. l'existence d'un calcul $\beta \xrightarrow{u} \alpha$ dans \mathcal{B} .
2. l'existence d'une dérivation droite $\alpha \xrightarrow{*}_r \beta u$ dans G dont la dernière étape éventuelle $\alpha \xrightarrow{*}_r \gamma x v \rightarrow_r \gamma \delta v = \beta u$ vérifie $|v| \leq |u|$.

Remarque: cette condition est en particulier vérifiée si $\beta \notin (\Sigma \cup V)^* \Sigma$.

Corollaire :

L'automate LR reconnaît le langage $\mathcal{L}_G(S)$

Conflits dans un automate LR

Exemple : Automate LR pour la grammaire G_2 :

1	id	$\xrightarrow{\varepsilon}$	F	7	ε	$\xrightarrow{*}$	$*$
2	(E)	$\xrightarrow{\varepsilon}$	F	8	ε	$\xrightarrow{+}$	$+$
3	$T * F$	$\xrightarrow{\varepsilon}$	T	9	ε	$\xrightarrow{(}$	$($
4	F	$\xrightarrow{\varepsilon}$	T	10	ε	$\xrightarrow{)}$	$)$
5	$E + T$	$\xrightarrow{\varepsilon}$	E	11	ε	$\xrightarrow{\text{id}}$	id
6	T	$\xrightarrow{\varepsilon}$	E				

Conflits

reduce/reduce : (3,4) : on choisit 3
(5,6) : on choisit 5

shift/reduce : {1,2,3,4} contre {7,8,9,10,11} : on choisit reduce
{5,6} contre 7 : on choisit shift (**priorité de * sur +**)
{5,6} contre {8,9,10,11} : on choisit reduce

La grammaire G_2 est non ambiguë : lors d'un conflit, si on fait le mauvais choix, on ne peut pas prolonger en un calcul acceptant.

k -conflits et grammaires LR(k)

Définition : k -conflits

- ▶ Un k -conflit **shift/reduce** est un tuple (x, α, w, av) tel qu'il existe $\delta \in (\Sigma \cup V)^*$ et deux calculs dans \mathcal{B} :

$$\delta\alpha \xrightarrow[\text{reduce}]{\varepsilon} \delta x \xrightarrow[*]{w} S \quad \text{et} \quad \delta\alpha \xrightarrow[\text{shift}]{a} \delta\alpha a \xrightarrow[*]{v} S$$

avec $\text{First}_k(w) = \text{First}_k(av)$.

- ▶ Un k -conflit **reduce/reduce** est un tuple $(x, \alpha, w, x', \alpha', w')$ tel qu'il existe $\delta, \delta' \in (\Sigma \cup V)^*$ et deux calculs dans \mathcal{B} :

$$\delta\alpha \xrightarrow[\text{reduce}]{\varepsilon} \delta x \xrightarrow[*]{w} S \quad \text{et} \quad \delta'\alpha' \xrightarrow[\text{reduce}]{\varepsilon} \delta'x' \xrightarrow[*]{w'} S$$

avec $\delta\alpha = \delta'\alpha'$, $\text{First}_k(w) = \text{First}_k(w')$ et $(x, \alpha) \neq (x', \alpha')$.

k -conflits

Exemples :

- ▶ La grammaire G_1 n'a aucun 0-conflit (il faut réduire dès que possible).
- ▶ La grammaire $G_3 : E \rightarrow E + E \mid E * E \mid (E) \mid \text{id}$ a des k -conflits pour tout k .

Exercice :

- ▶ Montrer que la grammaire G_2 n'a aucun 1-conflit.

Grammaires augmentées

Remarque :

Pour une grammaire LR(0) il faut aussi pouvoir décider si on doit s'arrêter sans regarder s'il reste des lettres à lire.

C'est le cas pour la grammaire G_1 : on s'arrête si la pile est exactement S .

Ce n'est pas le cas pour la grammaire $S \rightarrow Sa \mid a$ qui n'a pourtant aucun 0-conflit. Formellement, cette grammaire doit donc être LR(1) et pas LR(0).

Définition : Grammaire augmentée

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

La grammaire **augmentée** de G est $G' = (\Sigma, V \uplus \{S'\}, P \uplus \{S' \rightarrow S\}, S')$.

Définition : Grammaire LR(k)

Une grammaire G est LR(k) si sa grammaire augmentée G' n'a aucun k -conflit.

Remarque :

Soit G une grammaire sans dérivation du type $S \xrightarrow{\pm} S$ et $k > 0$.

La grammaire G n'a aucun k -conflit si et seulement si G' n'a aucun k -conflit.

Grammaires LR(k)

Remarques :

1. Une grammaire est LR(k) si et seulement si

$$\left. \begin{array}{l} S \xrightarrow{*}_r \delta x w \xrightarrow{1}_r \delta \alpha w \\ S \xrightarrow{*}_r \gamma \xrightarrow{1}_r \delta \alpha w' \\ \text{First}_k(w) = \text{First}_k(w') \end{array} \right\} \implies \gamma = \delta x w'$$

2. Toute grammaire LR(k) engendre un langage déterministe.
3. Tout langage déterministe peut être engendré par une grammaire LR(1).
4. La hiérarchie des grammaires LR(k) est stricte :
Pour tout $k > 0$ il existe une grammaire LR(k) qui n'est pas LR($k - 1$).
5. Étant donnée une grammaire G , on ne peut pas décider s'il existe un entier k tel que G soit LR(k).
6. Toute grammaire LL(k) est une grammaire LR(k).
7. On peut décider si une grammaire LR(k) est aussi LL(k).
8. Étant donnée une grammaire LR(k) G , on ne peut pas décider s'il existe n tel que G soit LL(n).

Analyseur LR(k)

Définition :

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

Soit \mathcal{A}_k l'analyseur LR(k). Il est défini par

- ▶ l'automate des contextes : $\mathcal{C}_k = (Q, \Sigma \cup V, q_0, \text{goto})$ automate fini déterministe
- ▶ la table des actions :

action : $Q \times \Sigma^{\leq k} \rightarrow \{\text{accept, error, shift}\} \cup \{\text{reduce}_{A \rightarrow \alpha} \mid A \rightarrow \alpha \in P\}$

Soit (γ, w) une configuration de \mathcal{A}_k où γ est le contexte (contenu de la pile) et w le mot qui reste à lire.

Dans la configuration (γ, w) , \mathcal{A}_k effectue $\text{action}(\text{goto}(q_0, \gamma), \text{First}_k(w))$.

Analyseur SLR

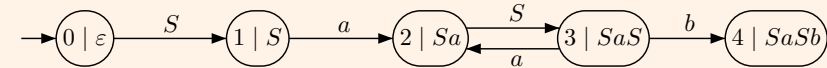
Exemple : Analyseur SLR pour G_4

0 : $S' \rightarrow S$ 1 : $S \rightarrow SaSb$ 2 : $S \rightarrow \varepsilon$

Analyseur SLR	action			goto
	a	b	ε	S
0 : ε	r_2	r_2	r_2	1
1 : S	s_2		accept	
2 : Sa	r_2	r_2	r_2	3
3 : SaS	s_2	s_4		
4 : $SaSb$	r_1	r_1	r_1	

s_i : shift and goto i

r_j : reduce with rule j .



Analyseur LR(k)

Remarques :

- ▶ Pour éviter de calculer $\text{goto}(q_0, \gamma)$ à chaque transition, on mémorise les états intermédiaires sur la pile :
Si $\gamma = \gamma_1 \dots \gamma_k$ alors la pile est en fait le calcul de l'automate :

$$q_0 \ \gamma_1 \ q_1 \ \dots \ \gamma_k \ q_k$$

avec $q_{i+1} = \text{goto}(q_i, \gamma_{i+1})$.

Initialement, la pile est donc q_0 .

- ▶ Lors d'un shift(a) on empile a puis $\text{goto}(q_k, a)$
- ▶ Lors d'un $\text{reduce}_{A \rightarrow \alpha}$ on dépile $2|\alpha|$ symboles et on empile A puis $\text{goto}(q_{k-|\alpha|}, A)$.
- ▶ Lors d'un $\text{reduce}_{A \rightarrow \alpha}$, α sera toujours un suffixe de γ .
- ▶ Les symboles $\gamma_1, \dots, \gamma_k$ sont en fait inutiles, il suffit d'avoir la pile des états q_0, \dots, q_k .

Analyseur SLR (Simple LR)

Définition : 0-item

- ▶ Un 0-item est une règle pointée : $A \rightarrow \alpha_1 \cdot \alpha_2$ avec $A \rightarrow \alpha_1 \alpha_2 \in P$.
- ▶ Le 0-item $A \rightarrow \alpha_1 \cdot \alpha_2$ est valide dans le contexte γ si α_1 est suffixe de γ et s'il existe dans l'automate shift/reduce \mathcal{B} un calcul

$$\gamma \alpha_2 = \delta \alpha_1 \alpha_2 \xrightarrow[\text{reduce}]{\varepsilon} \delta A \xrightarrow[*]{w} S'$$

ou de façon équivalente, s'il existe dans G une dérivation droite :

$$S' \xrightarrow[*]{r} \delta A w \rightarrow_r \delta \alpha_1 \alpha_2 w = \gamma \alpha_2 w$$

- ▶ On note $V_0(\gamma)$ l'ensemble des 0-items valides pour γ .

Remarque :

- ▶ Si $A \rightarrow \alpha_1 \cdot a \alpha_2 \in V_0(\gamma)$ alors l'action shift_a est utile dans le contexte γ .
- ▶ Si $A \rightarrow \alpha \cdot \in V_0(\gamma)$ alors l'action $\text{reduce}_{A \rightarrow \alpha}$ est utile dans le contexte γ .

L'automate des contextes \mathcal{C}_0 calcule les 0-items valides.

Calcul des 0-items valides

Définition : Clôture

Soit W un ensemble de 0-items.

- ▶ Règle de clôture : $\frac{A \rightarrow \alpha_1.B\alpha_2 \in W, B \rightarrow \beta \in P}{B \rightarrow \cdot\beta \in W}$
- ▶ On note $\text{clot}(W)$ la clôture de W .

Lemme : Clôture (G réduite)

Pour tout $\gamma \in (\Sigma \cup V)^*$, l'ensemble $V_0(\gamma)$ est clos.

Définition : goto

Soit W un ensemble de 0-items et $x \in \Sigma \cup V$.

$$\text{goto}(W, x) = \text{clot}(\{A \rightarrow \alpha_1 x \alpha_2 \mid A \rightarrow \alpha_1 \cdot x \alpha_2 \in W\})$$

Lemme : goto (G réduite)

Pour tout $\gamma \in (\Sigma \cup V)^*$, on a $\text{goto}(V_0(\gamma), x) \subseteq V_0(\gamma x)$.

Automate des contextes

Définition : Automate des contextes SLR

L'automate $\mathcal{C}_0 = (Q, \Sigma \cup V, q_0, \text{goto})$ est défini par

- ▶ Q est un sous-ensemble des ensembles de 0-items
- ▶ $q_0 = \text{clot}(\{S' \rightarrow \cdot S\})$
- ▶ goto est déjà défini.

On ne considère que les états accessibles.

Proposition : Automate des contextes (G réduite)

L'automate \mathcal{C}_0 calcule les 0-items valides : pour tout $\gamma \in (\Sigma \cup V)^*$ on a

$$V_0(\gamma) = \text{goto}(q_0, \gamma)$$

Exemple : Automate \mathcal{C}_0 des contextes SLR de G_4

$$0 : S' \rightarrow S \quad 1 : S \rightarrow SaSb \quad 2 : S \rightarrow \varepsilon$$

Table des actions

Définition : Table des actions de l'analyseur SLR \mathcal{A}_0

Soit W un ensemble de 0-items, $a \in \Sigma$ et $u \in \Sigma^{\leq 1}$:

- ▶ $\text{action}(W, a) = \text{shift}$ si W contient un 0-item du type $A \rightarrow \alpha_1 \cdot a \alpha_2$
- ▶ $\text{action}(W, u) = \text{reduce}_{A \rightarrow \alpha}$ si $A \rightarrow \alpha \in W$ et $u \in \text{Follow}_1(A)$ et $A \neq S'$
- ▶ $\text{action}(W, \varepsilon) = \text{accept}$ si $S' \rightarrow S \in W$
- ▶ $\text{action}(W, u) = \text{error}$ sinon

Remarque : les actions ne sont utiles que pour les états accessibles de l'automate des contextes \mathcal{C}_0 .

Définition : Grammaire SLR

Une grammaire G est SLR s'il n'y a pas de conflit dans la table action de son analyseur SLR

Analyseur SLR

Exemple : Analyseur SLR pour G_4

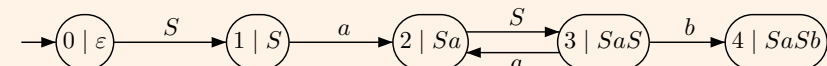
$$0 : S' \rightarrow S \quad 1 : S \rightarrow SaSb \quad 2 : S \rightarrow \varepsilon$$

$$\text{Follow}_1(S') = \{\varepsilon\} \quad \text{Follow}_1(S) = \{\varepsilon, a, b\}$$

Analyseur SLR	action			goto
	a	b	ε	S
0 : ε	r_2	r_2	r_2	1
1 : S	s_2		accept	
2 : Sa	r_2	r_2	r_2	3
3 : SaS	s_2	s_4		
4 : $SaSb$	r_1	r_1	r_1	

s_i : shift and goto i

r_j : reduce with rule j .



Analyse SLR

Exercice :

Calculer l'automate des contextes et la table des actions pour la grammaire G_2 :

0 : $E' \rightarrow E$ 1 : $E \rightarrow E + T$ 2 : $E \rightarrow T$
 3 : $T \rightarrow T * F$ 4 : $T \rightarrow F$
 5 : $F \rightarrow (E)$ 6 : $F \rightarrow \text{id}$

En déduire que G_2 est une grammaire SLR.

Analyse SLR

Proposition : Correction

Soit \mathcal{A}_0 l'analyseur SLR de $G = (\Sigma, V, P, S')$. On a $\mathcal{L}(\mathcal{A}_0) = \mathcal{L}_G(S')$.

Preuve

Soit \mathcal{B} l'analyseur shift/reduce général de la grammaire G .

Tout calcul de \mathcal{A}_0 est un calcul de \mathcal{B} : $\mathcal{L}(\mathcal{A}_0) \subseteq \mathcal{L}(\mathcal{B}) = \mathcal{L}_G(S')$.

Tout calcul **acceptant** $\varepsilon \xrightarrow{*} S'$ de \mathcal{B} est un calcul de \mathcal{A}_0 : $\mathcal{L}(\mathcal{A}_0) \supseteq \mathcal{L}(\mathcal{B}) = \mathcal{L}_G(S')$.

Remarque : non déterminisme

Si G n'est pas SLR, l'automate \mathcal{A}_0 est **non déterministe** : plusieurs actions peuvent être possibles dans une configuration (γ, w) .

On a quand même $\mathcal{L}(\mathcal{A}_0) = \mathcal{L}_G(S')$.

Analyseur SLR

Exemple : Analyseur SLR pour G_5

0 : $S' \rightarrow S$ 1 : $S \rightarrow L := R$ 2 : $S \rightarrow R$
 3 : $L \rightarrow *R$ 4 : $L \rightarrow \text{id}$ 5 : $R \rightarrow L$

$\text{Follow}_1(S') = \text{Follow}_1(S) = \{\varepsilon\}$ $\text{Follow}_1(R) = \text{Follow}_1(L) = \{\varepsilon, :=\}$

Analyseur SLR	action				goto		
	id	*	:=	ε	S	L	R
0 : ε	s_5	s_4			1	2	3
1 : S				accept			
2 : L			s_6 r_5	r_5			
3 : R				r_2			
4 : $*$	s_5	s_4				8	7
5 : id			r_4	r_4			
6 : $L :=$	s_5	s_4				8	9
7 : $*R$			r_3	r_3			
8 : $*L$			r_5	r_5			
9 : $L := R$				r_1			

Un conflit shift/reduce.

Grammaires et génération du code

Remarque : Grammaires équivalentes

La grammaire G_5

0 : $S' \rightarrow S$ 1 : $S \rightarrow L := R$ 2 : $S \rightarrow R$
 3 : $L \rightarrow *R$ 4 : $L \rightarrow \text{id}$ 5 : $R \rightarrow L$

est équivalente à la grammaire G'_5

0 : $S' \rightarrow S$ 1 : $S \rightarrow L := L$ 2 : $S \rightarrow L$
 3 : $L \rightarrow *L$ 4 : $L \rightarrow \text{id}$

et elle engendre même un langage rationnel donc elle est équivalente à une grammaire linéaire (gauche ou droite).

Cependant G'_5 est mieux adaptée à la génération du code.

Elle explicite la différence entre **adresse (L)** et **valeur (R)** et les règles permettent de générer le code correspondant :

- ▶ $L \rightarrow \text{id}$: obtenir l'**adresse** de la variable
- ▶ $R \rightarrow L$: obtenir la **valeur** contenue à une **adresse**
- ▶ $L \rightarrow *R$: convertir **valeur** en **adresse**.

Exemple : arbres syntaxiques pour les instructions $\text{id} := \text{id}$ et $*\text{id} := *\text{id}$.

Grammaire ambiguë

Exemple : Table SLR pour G_3

0 : $E' \rightarrow E$ 1 : $E \rightarrow E + E$ 2 : $E \rightarrow E * E$
 3 : $E \rightarrow (E)$ 4 : $E \rightarrow \text{id}$

$\text{Follow}_1(E) = \{\varepsilon, +, *,)\}$

	id	+	*	()	ε	E
0 : ε	s_3			s_2			1
1 : E		s_4	s_5			accept	
2 : $($	s_3			s_2			6
3 : id		r_4	r_4		r_4	r_4	
4 : $E+$	s_3			s_2			7
5 : $E*$	s_3			s_2			8
6 : $(E$		s_4	s_5		s_9		
7 : $E + E$		$r_1 \mid s_4$	$r_1 \mid s_5$		r_1	r_1	
8 : $E * E$		$r_2 \mid s_4$	$r_2 \mid s_5$		r_2	r_2	
9 : (E)		r_3	r_3		r_3	r_3	

4 conflits shift/reduce que l'on résout grâce aux règles de priorité et d'associativité.

169/194

IF THEN ELSE

Remarque :

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

170/194

Insuffisance de l'analyse SLR

Remarque :

Supposons que $\text{reduce}_{A \rightarrow \alpha} \in \text{action}(V_0(\gamma), a)$, i.e.,

$$A \rightarrow \alpha. \in V_0(\gamma) \quad \text{et} \quad a \in \text{Follow}_1(A)$$

Mais que pour tout $\gamma = \delta\alpha \xrightarrow[\text{reduce}]{\varepsilon} \delta A \xrightarrow[*]{w} S'$ on ait $a \notin \text{First}_1(w)$ alors,

l'action $\text{reduce}_{A \rightarrow \alpha}$ est inutile pour $(V_0(\gamma), a)$.

cf. [preuve de la proposition Correction](#).

Ceci est dû à l'imprecision de $\text{Follow}_1(A)$.

171/194

Analyseur LR(1)

Définition : 1-item

- ▶ 1-item : $[A \rightarrow \alpha_1.\alpha_2, u]$ avec $A \rightarrow \alpha_1\alpha_2 \in P$ et $u \in \Sigma^{\leq 1}$.
- ▶ Le 1-item $[A \rightarrow \alpha_1.\alpha_2, u]$ est **valide** dans le contexte γ si α_1 est suffixe de γ et s'il existe dans l'automate shift/reduce \mathcal{B} un calcul

$$\gamma\alpha_2 = \delta\alpha_1\alpha_2 \xrightarrow[\text{reduce}]{\varepsilon} \delta A \xrightarrow[*]{w} S' \quad \text{avec } u = \text{First}_1(w)$$

ou de façon équivalente, s'il existe dans G une dérivation **droite** :

$$S' \xrightarrow[*]{r} \delta A w \rightarrow_r \delta\alpha_1\alpha_2 w = \gamma\alpha_2 w \quad \text{avec } u = \text{First}_1(w)$$

- ▶ On note $V_1(\gamma)$ l'ensemble des 1-items valides pour γ .

Remarque :

- ▶ Si $[A \rightarrow \alpha_1.a\alpha_2, u] \in V_1(\gamma)$ alors l'action shift_a est utile dans le contexte γ .
- ▶ Si $[A \rightarrow \alpha., u] \in V_1(\gamma)$ alors l'action $\text{reduce}_{A \rightarrow \alpha}$ est utile dans une configuration (γ, w) avec $u = \text{First}_1(w)$.

L'automate des contextes \mathcal{C}_1 calcule les 1-items valides.

172/194

Calcul des 1-items valides

Définition : Clôture

Soit W un ensemble de 1-items.

- ▶ Règle de clôture : $\frac{[A \rightarrow \alpha_1.B\alpha_2, u] \in W, B \rightarrow \beta \in P, v \in \text{First}_1(\alpha_2 u)}{[B \rightarrow \cdot\beta, v] \in W}$
- ▶ On note $\text{clot}(W)$ la clôture de W .

Lemme : Clôture

Pour tout $\gamma \in (\Sigma \cup V)^*$, l'ensemble $V_1(\gamma)$ est clos.

Définition : goto

Soit W un ensemble de 1-items et $x \in \Sigma \cup V$.

$$\text{goto}(W, x) = \text{clot}(\{[A \rightarrow \alpha x.\alpha_2, u] \mid [A \rightarrow \alpha.x\alpha_2, u] \in W\})$$

Lemme : goto

Pour tout $\gamma \in (\Sigma \cup V)^*$, on a $\text{goto}(V_1(\gamma), x) \subseteq V_1(\gamma x)$.

Automate des contextes

Définition : Automate des contextes

L'automate $\mathcal{C}_1 = (Q_1, \Sigma \cup V, q_0, \text{goto})$ est défini par

- ▶ Q_1 est un sous-ensemble des ensembles de 1-items
- ▶ $q_0 = \text{clot}(\{[S' \rightarrow \cdot S, \varepsilon]\})$
- ▶ goto est déjà défini.

On ne considère que les états accessibles.

Proposition : Automate des contextes

L'automate \mathcal{C}_1 calcule les 1-items valides : pour tout $\gamma \in (\Sigma \cup V)^*$ on a

$$V_1(\gamma) = \text{goto}(q_0, \gamma)$$

Exemple :

Calcul de l'automate des contextes \mathcal{C}_1 pour la grammaire G_5 .

Exercices :

1. Calcul de l'automate des contextes \mathcal{C}_1 pour la grammaire G_2 .
2. Calcul de l'automate des contextes \mathcal{C}_1 pour la grammaire G_4 .

Table des actions

Définition : Table des actions

Soit W un ensemble de 1-items, $a \in \Sigma$ et $u \in \Sigma^{\leq 1}$:

- ▶ $\text{action}(W, a) = \text{shift}$ si W contient un 1-item du type $[A \rightarrow \alpha_1.a\alpha_2, u]$
- ▶ $\text{action}(W, u) = \text{reduce}_{A \rightarrow \alpha}$ si $[A \rightarrow \alpha., u] \in W$ et $A \neq S'$
- ▶ $\text{action}(W, \varepsilon) = \text{accept}$ si $[S' \rightarrow S., \varepsilon] \in W$
- ▶ $\text{action}(W, u) = \text{error}$ sinon

Remarque : les actions ne sont utiles que pour les états accessibles de l'automate des contextes.

Exemple :

Tables action et goto pour l'analyseur LR(1) de G_5 .

Exercices :

1. Tables action et goto pour l'analyseur LR(1) de G_2 .
2. Tables action et goto pour l'analyseur LR(1) de G_4 .

Analyseur LR(1)

Exemple : Analyseur LR(1) pour G_5

Analyseur LR(1)	action				goto		
	id	*	:=	ε	S	L	R
0 : ε	s_5	s_4			1	2	3
1 : S				accept			
2 : L			s_6	r_5			
3 : R				r_2			
4 : *	s_5	s_4				8	7
5 : id			r_4	r_4			
6 : L :=	s_{12}	s_{11}				10	9
7 : *R			r_3	r_3			
8 : *L			r_5	r_5			
9 : L := R				r_1			
10 : L := L				r_5			
11 : L := *	s_{12}	s_{11}				10	13
12 : L := id				r_4			
13 : L := *R				r_3			

Analyse LR(1)

Lemme : \mathcal{A}_0 versus \mathcal{A}_1

- ▶ Si $[A \rightarrow \alpha_1.\alpha_2, u] \in V_1(\gamma)$ alors $A \rightarrow \alpha_1.\alpha_2 \in V_0(\gamma)$ et $u \in \text{Follow}_1(A)$.
- ▶ Si $A \rightarrow \alpha_1.\alpha_2 \in V_0(\gamma)$ alors il existe $u \in \Sigma^{\leq 1}$ tel que $[A \rightarrow \alpha_1.\alpha_2, u] \in V_1(\gamma)$.
- ▶ \mathcal{A}_0 et \mathcal{A}_1 ont les mêmes actions **shift**.
- ▶ Les actions **reduce** de \mathcal{A}_1 sont des actions de \mathcal{A}_0 .

Proposition : Correction

Soit \mathcal{A}_1 l'analyseur LR(1) de $G = (\Sigma, V, P, S')$. On a $\mathcal{L}(\mathcal{A}_1) = \mathcal{L}_G(S')$.

Proposition :

Une grammaire G est LR(1) si et seulement si il n'y a pas de conflit dans la table action de son analyseur LR(1)

Plan

Introduction

Langages reconnaissables

Automates d'arbres

Grammaires

Langages algébriques

Automates à pile

Analyse syntaxique

- 8 Fonctions séquentielles
 - Définitions et exemples
 - Composition
 - Normalisation

Bibliographie

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

Automates séquentiels purs

Définition : Automates séquentiels purs (Mealy machine)

$\mathcal{A} = (Q, A, B, q_0, \delta, \varphi)$ où

- ▶ Q ensemble fini d'états et $q_0 \in Q$ état initial,
- ▶ A et B alphabets d'entrée et de sortie,
- ▶ $\delta : Q \times A \rightarrow Q$ fonction *partielle* de transition,
- ▶ $\varphi : Q \times A \rightarrow B^*$ fonction *partielle* de sortie avec $\text{dom}(\varphi) = \text{dom}(\delta)$.

Remarque : L'automate "d'entrée" (Q, A, q_0, δ) est déterministe.

Définition : Sémantique : $\llbracket \mathcal{A} \rrbracket : A^* \rightarrow B^*$

On étend δ et φ à $Q \times A^*$ par

- ▶ $\delta(q, \varepsilon) = q$ et $\varphi(q, \varepsilon) = \varepsilon$
- ▶ $\delta(q, ua) = \delta(\delta(q, u), a)$ et $\varphi(q, ua) = \varphi(q, u)\varphi(\delta(q, u), a)$

et la sémantique de \mathcal{A} est la fonction *partielle* $\llbracket \mathcal{A} \rrbracket : A^* \rightarrow B^*$ définie par

- ▶ $\llbracket \mathcal{A} \rrbracket(u) = \varphi(q_0, u)$.

Noter que $\llbracket \mathcal{A} \rrbracket(\varepsilon) = \varepsilon$

fonctions séquentielles pures

Définition : fonctions séquentielles pures

Une fonction $f : A^* \rightarrow B^*$ est séquentielle pure s'il existe un automate séquentiel pur \mathcal{A} qui la réalise : $f = \llbracket \mathcal{A} \rrbracket$.

Exemples :

1. Transformation d'un texte en majuscules.
2. Remplacement d'une séquence d'espaces ou *tabulations* par un seul espace.
3. Codage et décodage avec le code préfixe défini par

$a \mapsto 0000$	$c \mapsto 001$	$e \mapsto 011$	$g \mapsto 11$
$b \mapsto 0001$	$d \mapsto 010$	$f \mapsto 10$	

4. Division par 3 d'un entier écrit en binaire en commençant par le bit de poids fort. [Qu'en est-il si on commence avec le bit de poids faible ?](#)

Automates séquentiels

Définition : Automates séquentiels

$\mathcal{A} = (Q, A, B, q_0, \delta, \varphi, m, \rho)$ où

- ▶ $\mathcal{A} = (Q, A, B, q_0, \delta, \varphi)$ est un automate séquentiel pur,
- ▶ $m \in B^*$ est le préfixe initial,
- ▶ $\rho : Q \rightarrow B^*$ est la fonction *partielle* finale.

La sémantique de \mathcal{A} est la fonction partielle $\llbracket \mathcal{A} \rrbracket : A^* \rightarrow B^*$ définie par

- ▶ $\llbracket \mathcal{A} \rrbracket(u) = m\varphi(q_0, u)\rho(\delta(q_0, u))$.

On appelle état final un état dans $\text{dom}(\rho)$.

Exemples :

1. La fonction $f : A^* \rightarrow A^*$ définie par $f(u) = u(ab)^{-1}$.
2. Addition de deux entiers écrits en binaire en commençant par le bit de poids faible.
3. La multiplication par 3 d'un entier écrit en binaire en commençant par le bit de poids faible.
4. Le décodage par un code à délai de déchiffrement borné.

[Ces fonctions sont-elles séquentielles pures ?](#)

fonctions séquentielles

Définition : fonctions séquentielles

Une fonction $f : A^* \rightarrow B^*$ est séquentielle s'il existe un automate séquentiel \mathcal{A} qui la réalise : $f = \llbracket \mathcal{A} \rrbracket$.

Lemme :

Une fonction séquentielle peut être réalisée par un automate séquentiel ayant un préfixe initial vide ($m = \varepsilon$).

Proposition :

Une fonction séquentielle peut être réalisée par un automate **émondé**, i.e., tel que $\forall p \in Q, \exists u, v \in A^*$ tels que $\delta(q_0, u) = p$ et $\delta(p, v) \in \text{dom}(\rho)$.

Composition

Théorème : Composition

Soient $f : A^* \rightarrow B^*$ et $g : B^* \rightarrow C^*$ deux fonctions partielles.

1. Si f et g sont séquentielles alors $g \circ f : A^* \rightarrow C^*$ est aussi séquentielle.
2. Si f et g sont séquentielles *pures* alors $g \circ f$ est aussi séquentielle *pure*.

Exemple : Multiplication par 5

Dans cet exemple, $A = C = \{0, 1\}$, $B = \{0, 1\}^2$ et les mots représentent des entiers codés en binaire en commençant par le bit de poids faible.

On considère les fonctions séquentielles $f : A^* \rightarrow B^*$ et $g : B^* \rightarrow C^*$ définies par $f(n) = (n, 4n)$, i.e., $f(u) = (u00, 00u)$ et $g(n, m) = n + m$.

La fonction $g \circ f$ code la multiplication par 5.

Construire les automates séquentiels réalisant f et g .

En déduire un automate séquentiel pour $g \circ f$.

Produit en couronne

Définition : Produit en couronne

Soient $\mathcal{A} = (Q, A, B, q_0, \delta, \varphi, m, \rho)$ et $\mathcal{A}' = (Q', B, C, q'_0, \delta', \varphi', m', \rho')$ deux automates séquentiels.

Le produit en couronne $\mathcal{A}' \circ \mathcal{A} = (Q'', A, C, q''_0, \delta'', \varphi'', m'', \rho'')$ est défini par

- ▶ $Q'' = Q \times Q'$, $q''_0 = (q_0, \delta'(q'_0, m))$ et $m'' = m' \varphi'(q'_0, m)$,
- ▶ $\delta''((p, p'), a) = (\delta(p, a), \delta'(p', \varphi(p, a)))$,
- ▶ $\varphi''((p, p'), a) = \varphi'(p', \varphi(p, a))$,
- ▶ $\rho''((p, p')) = \varphi'(p', \rho(p)) \rho'(\delta'(p', \rho(p)))$.

Lemme : Extension à A^*

Pour tout $u \in A^*$, on a

- ▶ $\delta''((p, p'), u) = (\delta(p, u), \delta'(p', \varphi(p, u)))$,
- ▶ $\varphi''((p, p'), u) = \varphi'(p', \varphi(p, u))$,

Preuve (Composition)

1. Si f et g sont réalisées par \mathcal{A} et \mathcal{A}' alors $g \circ f$ est réalisée par $\mathcal{A}' \circ \mathcal{A}$.
2. Si \mathcal{A} et \mathcal{A}' sont purs alors $\mathcal{A}' \circ \mathcal{A}$ est pur.

Fonct. séquentielles et lang. rationnels

Définition : Fonction caractéristique

Soit $L \subseteq A^*$ un langage. La fonction caractéristique de L est la fonction totale $\mathbf{1}_L : A^* \rightarrow \{0, 1\}$ définie par $\mathbf{1}_L(u) = 1$ si et seulement si $u \in L$.

Théorème :

Un langage $L \subseteq A^*$ est rationnel si et seulement si sa fonction caractéristique $\mathbf{1}_L$ est séquentielle.

Corollaire : Image inverse

Soient $f : A^* \rightarrow B^*$ une fonction séquentielle.
Si $L \subseteq B^*$ est rationnel alors $f^{-1}(L)$ est rationnel.

Théorème : Image directe

Soient $f : A^* \rightarrow B^*$ une fonction séquentielle.
Si $L \subseteq A^*$ est rationnel alors $f(L)$ est rationnel.

Plus grand préfixe commun

Définition :

- ▶ Tout sous ensemble $\emptyset \neq X \subseteq B^*$ admet un plus grand préfixe commun, i.e., une borne inférieure pour l'ordre préfixe. Cette borne inférieure est notée $\bigwedge X$.
- ▶ Noter que \emptyset n'admet pas de plus grand préfixe commun. Donc $\bigwedge \emptyset$ n'est pas défini.

Remarque :

1. Soit $u \in B^*$ et $\emptyset \neq X \subseteq B^*$.
 - ▶ $\bigwedge u \cdot X = u \cdot \bigwedge X$
 - ▶ si $u \leq \bigwedge X$ alors $\bigwedge u^{-1} \cdot X = u^{-1} \cdot \bigwedge X$
2. Soit $f : A^* \rightarrow B^*$ une fonction partielle, on a $f(A^*) = \{f(u) \mid u \in \text{dom}(f)\}$.
Donc $\bigwedge f(A^*)$ est défini si $\text{dom}(f) \neq \emptyset$.

Exemple :

Soit $f : A^* \rightarrow A^*$ la fonction partielle définie par $f(w) = w(ab)^{-1}$.
Pour $u \in A^*$, calculer $\bigwedge f(uA^*)$.

Normalisation

Exemple :

Donner un automate séquentiel réalisant la fonction $f : A^* \rightarrow A^*$ définie par $f(a^{2n}b) = (ab)^n a$.
Cet automate devra sortir les lettres du résultat le plus rapidement possible.

Définition : Automate normalisé

Intuitivement, un automate est normalisé s'il écrit son résultat au plus tôt.

Soit $\mathcal{A} = (Q, A, B, q_0, \delta, \varphi, m, \rho)$ un automate séquentiel et $p \in Q$.

On définit $\mathcal{A}_p = (Q, A, B, p, \delta, \varphi, \varepsilon, \rho)$ et $m_p = \bigwedge \llbracket \mathcal{A}_p \rrbracket(A^*)$ si $\llbracket \mathcal{A}_p \rrbracket(A^*) \neq \emptyset$.

L'automate \mathcal{A} est normalisé si pour tout $p \in Q$, $\llbracket \mathcal{A}_p \rrbracket(A^*) = \emptyset$ ou $m_p = \varepsilon$.

Proposition : Effectivité

Étant donné un automate séquentiel \mathcal{A} , on peut calculer les m_p en temps quadratique (cf. DM1 2006).

Normalisation

Proposition : Normalisation

Tout automate séquentiel est équivalent à un automate séquentiel normalisé, qui peut être choisi émondé ou complet.

Preuve

Soit $\mathcal{A} = (Q, A, B, q_0, \delta, \varphi, m, \rho)$ un automate séquentiel **émondé**.

On définit $\mathcal{A}' = (Q, A, B, q_0, \delta, \varphi', m', \rho')$ par :

- ▶ $m' = mm_{q_0} = \bigwedge \llbracket \mathcal{A} \rrbracket (A^*)$,
- ▶ $\varphi'(p, a) = m_p^{-1}(\varphi(p, a)m_{\delta(p,a)})$ si $(p, a) \in \text{dom}(\varphi) = \text{dom}(\delta)$
- ▶ $\rho'(p) = m_p^{-1}\rho(p)$ si $p \in \text{dom}(\rho)$

On vérifie que \mathcal{A}' est normalisé et $\llbracket \mathcal{A}' \rrbracket = \llbracket \mathcal{A} \rrbracket$.

Pour obtenir un automate complet, il suffit d'ajouter un état puits.

Séquentielle et séquentielle pure

Définition :

Une fonction partielle $f : A^* \rightarrow B^*$ **préserve les préfixes** si

- ▶ son domaine est préfixiel : $u \leq v$ et $v \in \text{dom}(f)$ implique $u \in \text{dom}(f)$,
- ▶ et elle est croissante : $u \leq v$ et $v \in \text{dom}(f)$ implique $f(u) \leq f(v)$.

Proposition :

1. Une fonction séquentielle pure préserve les préfixes.
2. Soit $f : A^* \rightarrow B^*$ une fonction séquentielle. Si $f(\varepsilon) = \varepsilon$ et f préserve les préfixes alors f est séquentielle pure.

Preuve

L'automate normalisé émondé d'une fonction séquentielle f qui préserve les préfixes et telle que $f(\varepsilon) = \varepsilon$ est un automate séquentiel pur.

Résiduels

Définition : Résiduels

Soit $f : A^* \rightarrow B^*$ une fonction partielle et soit $u \in A^*$.

Le résiduel $f_u : A^* \rightarrow B^*$ est défini par

- ▶ $\text{dom}(f_u) = u^{-1}\text{dom}(f)$ et
- ▶ $f_u(v) = (\bigwedge f(uA^*))^{-1}f(uv)$ pour $uv \in \text{dom}(f)$.

$\bigwedge f(uA^*)$ représente tout ce qu'on peut sortir si on sait que la donnée commence par u . Le résiduel $f_u(v)$ est donc ce qui reste à sortir si la donnée est uv .

Exemple :

Calculer les résiduels de la fonction $f : A^* \rightarrow A^*$ définie par $f(w) = w(ab)^{-1}$.

Lemme : Composition

Soient $u, v \in A^*$. On a $f_{uv} = (f_u)_v$, i.e.,
 $\text{dom}(f_{uv}) = v^{-1}\text{dom}(f_u)$ et $f_{uv}(w) = (\bigwedge f_u(vA^*))^{-1}f_u(vw)$.

Résiduels

Théorème : Caractérisation par résiduels

Une fonction $f : A^* \rightarrow B^*$ est séquentielle si et seulement si elle a un nombre fini de résiduels.

Lemme :

Soit $\mathcal{A} = (Q, A, B, q_0, \delta, \varphi, m, \rho)$ un automate normalisé complet.

Soit $u \in A^*$ et $p = \delta(q_0, u)$. Alors $f_u = \llbracket \mathcal{A}_p \rrbracket$.

On en déduit qu'une fonction séquentielle réalisée par \mathcal{A} a au plus $|Q|$ résiduels.

Exemple :

La fonction $f : A^* \rightarrow A^*$ définie par $f(w) = ww$ est-elle séquentielle ?

Automate des résiduels

L'automate des résiduels de f est $\mathcal{R} = (Q, A, B, q_0, \delta, \varphi, m, \rho)$ où

- ▶ $Q = \{f_u \mid u \in A^*\}$ (supposé fini pour la réciproque du théorème),
- ▶ $q_0 = f_\varepsilon$ et $m = \bigwedge f(A^*)$ si $\text{dom}(f) \neq \emptyset$, et $m = \varepsilon$ sinon,
- ▶ $\delta(f_u, a) = (f_u)_a = f_{ua}$,
- ▶ $\varphi(f_u, a) = \bigwedge f_u(aA^*)$ si $\text{dom}(f_{ua}) \neq \emptyset$, et $\varphi(f_u, a) = \varepsilon$ sinon,
- ▶ $\rho(f_u) = f_u(\varepsilon)$ si $\varepsilon \in \text{dom}(f_u)$, et $f_u \notin \text{dom}(\rho)$ sinon.

Lemme :

1. Soient $u, v \in A^*$. On a $\delta(f_u, v) = f_{uv}$.
2. Soient $u, v \in A^*$. On a $\varphi(f_u, v) = \bigwedge f_u(vA^*)$ si $\text{dom}(f_{uv}) \neq \emptyset$.
3. Soit $u \in A^*$. On a $f_u = \llbracket \mathcal{R}_{f_u} \rrbracket$.
4. $f = \llbracket \mathcal{R} \rrbracket$.
5. L'automate des résiduels est normalisé, accessible et complet.

Exemple :

Calculer l'automate des résiduels de la fonction *multiplication par 5* où les entiers sont codés en binaire en commençant avec le bit de poids faible.

Minimisation

Théorème : Automate minimal

Soit $f : A^* \rightarrow B^*$ une fonction séquentielle.

L'automate des résiduels de f , noté \mathcal{R}_f , est minimal parmi les automates normalisés et complets qui réalisent f .

Construction de l'automate minimal

Soit $\mathcal{A} = (Q, A, B, q_0, \delta, \varphi, m, \rho)$ un automate réalisant une fonction f .

- ▶ émonder puis normaliser puis compléter l'automate.
- ▶ quotienter l'automate par l'équivalence définie par $p \sim q$ si $\llbracket \mathcal{A}_p \rrbracket = \llbracket \mathcal{A}_q \rrbracket$.

Cette équivalence se calcule par raffinement :

- ▶ $p \sim_0 q$ si $\rho(p) = \rho(q)$.
- ▶ $p \sim_{n+1} q$ si $p \sim_n q$ et $\forall a \in A, \delta(p, a) \sim_n \delta(q, a)$ et $\varphi(p, a) = \varphi(q, a)$.

Exemple :

Minimiser l'automate *naturel* de $f : A^* \rightarrow A^*$ définie par $f(w) = w(ab)^{-1}$.