Langages formels

Paul Gastin

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

Magistère 2006

Plan

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

Plan

Bibliographie

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

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

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

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

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

Mots

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

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

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

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

Mots

Ordres partiels :

- u préfixe de v si $\exists u'$, $v = uu'$
- u suffixe de v si $\exists u'$, $v = u'u$
- u facteur de v si $\exists u', u''$, $v = u'uu''$
- u sous-mot de v si $v = v_0 u_1 v_1 u_1 \cdots u_n v_n$ avec $u_i, v_i \in \Sigma^*$ et $u = u_1 u_2 \cdots u_n$

Théorème : Higman

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

Langages

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

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

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

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

Langages

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

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

Automates déterministes

Définition : Automate déterministe

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

Exemples.

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

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

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

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

Automates déterministes

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

Définition : Reconnaissables

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

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

Automates non déterministes

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

Définition : Automate non déterministe

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

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

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

Automates non déterministes

Théorème : Déterminisation

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

Preuve

Automate des parties

Exemple : automate déterministe pour $\Sigma^* \cdot \{aba\}$
On appelle déterminisé de $\mathcal{A}$ l'automate des parties *émondé*.

Exercices :

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

Automates non déterministes

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

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

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

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

Corollaire :

Soit $\mathcal{A}$ un automate.

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

Automates avec ε-transitions

Exemple.

Définition : Automate avec ε-transitions

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

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

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

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

Décision

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

Définition :

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

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

Théorème : vide et mot

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

Preuve

C'est de l'accessibilité.

Propriétés de fermeture

Opérations ensemblistes

Proposition :

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

Preuve

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

Corollaire :

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

Propriétés de fermeture

Opérations liées à la concaténation

Proposition :

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

Concaténation :

Méthode 1 : union disjointe des automates et ajout de transitions.

Méthode 2 : fusion d'états.
On suppose que les automates ont un seul état initial sans transition entrante et un seul état final sans transition sortante.

Itération :

Méthode 1 : ajout de transitions. Ajouter un état pour reconnaître le mot vide.

Méthode 2 : ajout d'ε-transitions.

Propriétés de fermeture

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

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

Proposition :

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

Preuve

Modification des états initiaux et/ou finaux.

Propriétés de fermeture

Proposition :

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

Preuve

Modification des états initiaux et/ou finaux.

Exercice :

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

Propriétés de fermeture

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

Proposition :

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

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

Preuve

Modification des transitions de l'automate.

Propriétés de fermeture

Définition : Substitutions

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

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

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

Propriétés de fermeture

Proposition :

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

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

Preuve

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

Langages rationnels

Syntaxe pour représenter des langages.

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

Définition : Syntaxe

L'ensemble des ER est défini par

B : $\underline{\emptyset}$ et $\underline{a}$ pour $a \in \Sigma$ sont des ER,
I : Si E et F sont des ER alors $(E + F)$, $(E \cdot F)$ et (E^*) aussi.

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

Langages rationnels

Définition : Sémantique

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

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

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

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

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

Langages rationnels

Définition :

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

Exemples : commutativité, associativité, distributivité, ...

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

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

Abus de notation :

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

Langages rationnels

Théorème : Kleene, 1936

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

Preuve

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

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

Corollaire :

L'équivalence des expressions rationnelles est décidable.

Preuve

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

Critères de reconnaissabilité

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

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

Exemples.

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

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

Critères de reconnaissabilité

Lemme : itération

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

1. si $|w| \geq N$ alors $\exists u_1, u_2, u_3 \in \Sigma^*$ tels que $w = u_1 u_2 u_3$, $u_2 \neq \varepsilon$ et $u_1 u_2^* u_3 \subseteq L$.
2. si $w = w_1 w_2 w_3$ 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 $0 \leq i_0 < i_1 < \cdots < i_N \leq |w|$ (positions marquées dans w) alors il existe $0 \leq j < k \leq N$ tels que si on écrit $w = u_1 u_2 u_3$ avec $|u_1| = i_j$ et $|u_1 u_2| = i_k$ alors $u_1 u_2^* u_3 \subseteq L$.

Preuve

Sur l'automate qui reconnaît L.

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

Critères de reconnaissabilité

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

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

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

Critères de reconnaissabilité

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

Exemples : Sachant que L_1 n'est pas reconnaissable.

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

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 ?

Minimisation

Définition : Morphismes d'automates DC

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

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

$$\begin{tikzcd} Q \arrow[r, "\delta_a"] \arrow[d, "\varphi"'] & Q \arrow[d, "\varphi"] \\ Q' \arrow[r, "\delta'_a"] & Q' \end{tikzcd}$$

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

Remarque 1 : Deux automates DC sont isomorphes s'ils ne diffèrent que par le nom des états.

Remarque 2 : Si $\varphi : \mathcal{A} \to \mathcal{A}'$ est un morphisme bijectif, alors $\varphi^{-1} : \mathcal{A}' \to \mathcal{A}$ est aussi un morphisme.

Remarque 3 : Si $\varphi : \mathcal{A} \to \mathcal{A}'$ et $\psi : \mathcal{A}' \to \mathcal{A}''$ sont des morphismes, alors $\psi \circ \varphi : \mathcal{A} \to \mathcal{A}''$ est un morphisme.

Minimisation

Définition : Congruence sur les automates

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

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

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

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

Proposition :

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

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

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

Minimisation

Définition : Équivalence de Nérode

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

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

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

Proposition :

L'équivalence de Nérode est une congruence.
L'automate quotient $\mathcal{A}_\sim$ est appelé automate de Nérode.
On a $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{A}_\sim)$ (Proposition 3)

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

Problème : comment le calculer *efficacement* ?

Minimisation

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

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

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

Proposition : 5

- $p \sim_{n+1} q \quad \text{ssi} \quad p \sim_n q$ 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 $\sim = \sim_1$ sinon.

On utilise la Proposition 5 pour calculer l'équivalence de Nérode par raffinements successifs.

Minimisation

Définition : Résiduels

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

Définition : Automate des résiduels

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

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

Théorème :

L est reconnaissable ssi L a un nombre fini de résiduels.

Minimisation

Théorème :

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

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

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

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

- $\varphi : \Sigma^* \to M$ 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 $\varphi : \Sigma^* \to M$ qui reconnaît L.
- $L \subseteq \Sigma^*$ est *reconnaissable par morphisme* s'il existe un monoïde fini qui reconnaît L.

Définition : Monoïde de transitions

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

Proposition :

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

Morphismes

Théorème :

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

Corollaire :

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

Exemple :

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

Exercices :

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

Congruences

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

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

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.

Monoide syntaxique

Définition : Monoide 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 $\forall y \in \Sigma^*$, $uy \in L \Longleftrightarrow vy \in L$.
   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.

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.

Plan

Bibliographie

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

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

[8] 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,
- $\delta : Q \times A \to Q$ fonction *partielle* de transition,
- $\varphi : Q \times A \to B^*$ fonction *partielle* de sortie avec $\mathrm{dom}(\varphi) = \mathrm{dom}(\delta)$.

Définition : Sémantique : $[\![\mathcal{A}]\!] : A^* \to B^*$

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* $[\![\mathcal{A}]\!] : A^* \to B^*$ définie par

- $[\![\mathcal{A}]\!](u) = \varphi(q_0, u)$.

Noter que $[\![\mathcal{A}]\!](\varepsilon) = \varepsilon$

fonctions séquentielles pures

Définition : fonctions séquentielles pures

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

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 définie par

$$\begin{array}{llll} a \mapsto 0000 & c \mapsto 001 & e \mapsto 011 & g \mapsto 11 \\ b \mapsto 0001 & d \mapsto 010 & f \mapsto 10 & \end{array}$$

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,
- $\rho : Q \to Q$ est la fonction *partielle* finale.

La sémantique de $\mathcal{A}$ est la fonction partielle $[\![\mathcal{A}]\!] : A^* \to B^*$ définie par

- $[\![\mathcal{A}]\!](u) = m\varphi(q_0, u)\rho(\delta(q_0, u))$.

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

Exemples :

1. La fonction $f : A^* \to A^*$ 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 déchiffrage borné.

Ces fonctions sont-elles séquentielles pures ?

fonctions séquentielles

Définition : fonctions séquentielles

Une fonction $f : A^* \to B^*$ est séquentielle s'il existe un automate séquentiel $\mathcal{A}$ qui la réalise : $f = [\![\mathcal{A}]\!]$.

Lemme :

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

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 $m'' = m'\varphi'(q'_0, m)$,
- $\delta''((p, p'), a) = (\delta(p, a), \delta'(p', \varphi(p, a)))$,
- $\varphi''((p, p'), a) = \varphi'(p', \varphi(p, a))$,
- $\rho''((p, p')) = \varphi'(p', \rho(p))\rho'(\delta'(p', \rho(p)))$.

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 $f : A^* \to B^*$ et $g : B^* \to C^*$ définies par $f(n) = (n, 4n)$, i.e., $f(u) = (u, 00u)$ et $g(n, m) = n + m$.

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

Construire les automates séquentiels réalisant f et g et leur produit en couronne.

Composition

Lemme : Extension à A^*

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

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

Théorème : Composition

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

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

Preuve

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 $\mathbf{1}_L : A^* \to \{0, 1\}$ 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 $f : A^* \to B^*$ une fonction séquentielle.

Si $L \subseteq B^*$ est rationnel alors $f^{-1}(L)$ est rationnel.

Théorème : Image directe

Soient $f : A^* \to B^*$ une fonction séquentielle.

Si $L \subseteq A^*$ est rationnel alors $f(L)$ est rationnel.

$B^* \cup \{\mathbf{0}\}$

Définition : $\mathbf{0}$: élément maximal et absorbant.

- Soit $\mathbf{0} \notin B$ un nouvel élément.
- On étend la concaténation de B^* en faisant de $\mathbf{0}$ un élément absorbant : $w \cdot \mathbf{0} = \mathbf{0} \cdot w = \mathbf{0}$ pour tout $w \in B^* \cup \{\mathbf{0}\}$.
- On étend l'ordre préfixe de B^* en faisant de $\mathbf{0}$ un élément maximal : $w \leq \mathbf{0}$ pour tout $w \in B^* \cup \{\mathbf{0}\}$.
- Tout sous ensemble $X \subseteq B^* \cup \{\mathbf{0}\}$ admet un plus grand préfixe commun, i.e., une borne inférieure pour l'ordre préfixe.
  Cette borne inférieure est notée $\bigwedge X$.
  Noter que $\bigwedge \emptyset = \mathbf{0}$.

Fonctions totales dans $B^* \cup \{\mathbf{0}\}$

Définition : Fonction totale

On étend une fonction partielle $f : A^* \to B^*$ en une fonction totale $\hat{f} : A^* \to B^* \cup \{\mathbf{0}\}$ en posant $\hat{f}(u) = f(u)$ si $u \in \mathrm{dom}(f)$ et $\hat{f}(u) = \mathbf{0}$ sinon.
Noter que pour tout $X \subseteq A^*$, on a $\bigwedge f(X) = \bigwedge \hat{f}(X)$.

Proposition : Automate séquentiel total

Si la fonction $f : A^* \to B^*$ est séquentielle (partielle) alors la fonction $\hat{f} : A^* \to B^* \cup \{\mathbf{0}\}$ est séquentielle (totale).

Preuve

Il suffit de compléter l'automate $\mathcal{A} = (Q, A, B, q_0, \delta, \varphi, m, \rho)$ réalisant f en ajoutant au besoin un état puits, et de remplacer ρ par $\hat{\rho}$.

Dans la suite, on confondra f et $\hat{f}$.

Normalisation

Exemple :

Donner un automate séquentiel réalisant la fonction $f : A^* \to A^*$ 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 $p \in Q$ un état de $\mathcal{A}$.
On définit $\mathcal{A}_p = (Q, A, B, p, \delta, \varphi, \varepsilon, \rho)$ et $m_p = \bigwedge [\![\mathcal{A}_p]\!](A^*)$.
L'automate $\mathcal{A}$ est normalisé si pour tout $p \in Q$, $m_p = \varepsilon$.

Proposition : Normalisation

Tout automate séquentiel émondé est équivalent à un automate séquentiel émondé et normalisé ayant les mêmes états et la même fonction de transition.

Proposition : Effectivité

Étant donné un automate séquentiel $\mathcal{A}$, on peut calculer les m_p en temps quadratique.

Séquentielle et séquentielle pure

Définition :

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

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

Proposition :

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

Preuve

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

Résiduels

Définition : Résiduels

Soit $f : A^* \to B^* \cup \{\mathbf{0}\}$ une fonction totale et soit $u \in A^*$.

Le résiduel $f_u : A^* \to B^* \cup \{\mathbf{0}\}$ est défini par $f_u(v) = (\bigwedge f(uA^*))^{-1} f(uv)$ avec la convention $w^{-1}\mathbf{0} = \mathbf{0}$ pour tout $w \in B^* \cup \{\mathbf{0}\}$.

$\bigwedge f(uA^*)$ 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 :

1. Calculer les résiduels de la fonction $f : A^* \to A^* \cup \{\mathbf{0}\}$ définie par $f(w) = w(ab)^{-1}$.
2. Calculer les résiduels de la fonction $f : A^* \to A^*$ définie par $f(w) = ww$.
3. Calculer les résiduels de la fonction *multiplication par 5* où les entiers sont codés en binaire en commençant avec le bit de poids faible.

Résiduels

Théorème : Caractérisation par résiduels

Une fonction $f : A^* \to B^* \cup \{\mathbf{0}\}$ 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 et complet.

Si $u \in A^*$ et $p = \delta(q_0, u)$ alors $f_u = [\![\mathcal{A}_p]\!]$.

On en déduit qu'une fonction séquentielle a un nombre fini de résiduels.

Automate des résiduels

Réciproquement, Supposons $Q = \{f_u \mid u \in A^*\}$ fini.
L'automate des résiduels est $\mathcal{A} = (Q, A, B, q_0, \delta, \varphi, m, \rho)$ où

- $q_0 = f_\varepsilon$ et $m = \bigwedge f(A^*)$,
- $\delta(f_u, a) = f_{ua}$,
- $\varphi(f_u, a) = \bigwedge f_u(aA^*)$,
- $\rho(f_u) = f_u(\varepsilon)$.

Lemme :

1. Soient $u, v, w \in A^*$. On a $f_{uv}(w) = (\bigwedge f_u(vA^*))^{-1} f_u(vw)$.
2. La fonction de transition δ est bien définie et $\delta(f_u, v) = f_{uv}$.
3. Soient $u, v \in A^*$. On a $\varphi(f_u, v) = \bigwedge f_u(vA^*)$.
4. Soit $u \in A^*$. On a $f_u = [\![\mathcal{A}_{f_u}]\!]$.
5. $f = [\![\mathcal{A}]\!]$.
6. L'automate des résiduels est normalisé, accessible et complet.

Minimisation

Théorème : Automate minimal

Soit $f : A^* \to B^* \cup \{0\}$ 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 l'automate
- normaliser l'automate
- quotienter l'automate par l'équivalence définie par $p \sim q$ si $[\![\mathcal{A}_p]\!] = [\![\mathcal{A}_q]\!]$.

  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 $f : A^* \to A^* \cup \{\mathbf{0}\}$ définie par $f(w) = w(ab)^{-1}$.

Plan

Référence

TATA

Tree Automata Techniques and Applications

Hubert Comon, Max Dauchet, Remi Gilleron, Florent Jacquemard, Denis Lugiez, Sophie Tison, Marc Tommasi.

http://www.grappa.univ-lille3.fr/tata/

Arbres

Définition : Arbres

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

Un arbre étiqueté dans Σ et d'arité (au plus) p est une fonction partielle $t : A_p^* \to \Sigma$ dont le *domaine* est un langage $\mathrm{dom}(t) \subseteq A_p^*$
- fermé par préfixe : $u \leq v$ et $v \in \mathrm{dom}(t)$ implique $u \in \mathrm{dom}(t)$,
- fermé par *frère aîné* : $d_i \prec d_j$ et $ud_j \in \mathrm{dom}(t)$ implique $ud_i \in \mathrm{dom}(T)$.

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

Exemples :

1. Arbre représentant l'expression logique

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

2. Arbre représentant le programme

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

Arbres

Définition : Terminologie

La racine de l'arbre est le mot vide $\varepsilon \in \mathrm{dom}(t)$.
Un nœud de l'arbre est un élément $u \in \mathrm{dom}(t)$.
Une feuille de l'arbre est un nœud $u \in \mathrm{dom}(t)$ tel que $ud_1 \notin \mathrm{dom}(t)$.

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

L'arité d'un nœud $u \in \mathrm{dom}(t)$ est le plus grand entier k tel que $ud_k \in \mathrm{dom}(t)$ ($k = 0$ si u est une feuille).
Les fils d'un nœud $u \in \mathrm{dom}(t)$ d'arité k sont les nœuds $ud_1, \ldots, ud_k \in \mathrm{dom}(t)$.

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 $u \in \mathrm{dom}(t)$ 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 dont les étiquettes de toutes les branches sont dans un langage rationnel fixé $L \subseteq \Sigma^*$.
2. L'ensemble des arbres d'arité au plus p dont au moins une branche est étiquetée par un mot d'un langage rationnel fixé $L \subseteq \Sigma^*$.
3. L'ensemble des arbres d'arité au plus p ayant un nombre pair de noeuds internes.
4. 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 $(ab)^*$.

Termes

Définition :

- $\mathcal{F}$ 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 $\mathcal{F}_0$.
- $T(\mathcal{F}, \mathcal{X})$ 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.
- $\mathrm{Free}(t)$ 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 $|f(t_1, \dots, t_n)| = 1 + |t_1| + \cdots + |t_n|$.

Termes

Exemple : Expressions logiques

$\mathcal{F}_2 = \{\wedge, \vee\}$, $\mathcal{F}_1 = \{\neg\}$, $\mathcal{F}_0 = \{\top, \bot\}$, $\mathcal{X} = \{p, q, r\}$

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

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

La hauteur d'un terme est la hauteur de l'arbre qui le représente.
La taille d'un terme 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 $\mathcal{F}_2 = \{\wedge, \vee\}$, $\mathcal{F}_1 = \{\neg\}$, $\mathcal{F}_0 = \{\top, \bot\}$ et $\mathcal{X} = \emptyset$.
   L'ensemble des formules closes du calcul propositionnel qui s'évaluent à *vrai* est reconnaissable.
3. Considérons $\mathcal{F}_2 = \{\wedge, \vee\}$, $\mathcal{F}_1 = \{\neg\}$, $\mathcal{F}_0 = \{\top, \bot\}$ et $\mathcal{X} = \{p_1, \ldots, p_n\}$ 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 $\mathcal{F} = \biguplus_{0 \le i \le p} \Sigma_i$ où Σ_i est une copie de Σ.
Soit t' l'arbre ayant même domaine que t et tel que si $u \in \mathrm{dom}(t)$ 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'.

Substitutions

Définition :

- Une substitution σ est une application d'un sous-ensemble fini de $\mathcal{X}$ dans $T(\mathcal{F}, \mathcal{X})$.
- Si $\sigma = [t_1/x_1, \ldots, t_n/x_n]$ est une substitution et t un terme alors $\sigma(t) = t[t_1/x_1, \ldots, t_n/x_n]$ est défini inductivement par :
  - $\sigma(x_i) = t_i$ pour $1 \le\ i \le n$,
  - $\sigma(f) = f$ pour $f \in \mathcal{F}_0 \cup \mathcal{X} \setminus \{x_1, \ldots, x_n\}$
  - $\sigma(f(s_1, \ldots, s_k)) = f(\sigma(s_1), \ldots, \sigma(s_k))$ pour $f \in \mathcal{F}_k$, $k \ge 1$.

  On dit que $t[t_1/x_1, \ldots, t_n/x_n]$ est une *instance* de t.
- La substitution $\sigma = [t_1/x_1, \ldots, t_n/x_n]$ 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, $t[t_1/x_1, t_2/x_2] \ne t[t_1/x_1][t_2/x_2]$.

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 arbres $t \in T(\mathcal{F})$ 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 $\delta : \bigcup_p Q^p \times \Sigma \to 2^Q$.
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 $\delta : \bigcup_p Q^p \times \Sigma \to Q$ 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 $\delta : Q \times \Sigma \to 2^{\bigcup_p Q^p}$.
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 $\delta : Q \times \Sigma \to \bigcup_p Q^p$ 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 $\delta : \bigcup_p Q_p \times \Sigma \to 2^Q$

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

- 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 : $t : A^* \to \Sigma \cup \{\square\}$, $t^{-1}(\square) = \{u\}$ et u est une feuille.
On note $T_\square(\Sigma)$ l'ensemble des Σ-arbres à trou.

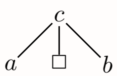
Définition : Concaténation

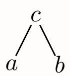
Soit t un Σ-arbre avec un trou en u et soit t' un Σ-arbre (avec ou sans trou). La concaténation $t \cdot t'$ est le Σ-arbre (avec ou sans trou) défini par

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

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 $\mathrm{Fr}(L)$ des mots de feuilles de L est $\{a^n b^n \mid n > 0\}$.

Lemme d'itération

Lemme : itération (pumping) pour les termes

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

- $t_2 \neq \square, \quad t = t_1 \cdot t_2 \cdot t_3, \quad t_1(t_2)^* t_3 \subseteq L$,
- La somme des profondeurs de $\square$ dans t_1 et t_2 est inférieure à 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 $t_1, \ldots, t_n \in T(\Sigma)$. L'arbre $t = a(t_1, \ldots, t_n)$ est défini par

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

Définition : Congruence

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

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

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 $r \cdot s \approx r \cdot t$.

Congruence syntaxique

Définition : Congruence syntaxique

La congruence syntaxique $\equiv_L$ d'un langage $L \subseteq T_p(\Sigma)$ est définie par $r \equiv_L t$ si pour tout $r \in T_{p,\square}(\Sigma)$, $r \cdot s \in L$ ssi $r \cdot t \in L$.

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

Définition :

Soit $L \subseteq T_p(\Sigma)$. L'automate $\mathcal{A}_L = (Q_L, \Sigma, \delta_L, F_L)$ est défini par :

- Q_L est l'ensemble des classes d'équivalences pour $\equiv_L$,
  (on note simplement $[t]$ la classe d'équivalence de t pour $\equiv_L$)
- $\delta_L([t_1], \ldots, [t_n], a) = [a(t_1, \ldots, t_n)]$,
- $F_L = \{[t] \mid\ t \in L\}$.

Proposition : automate minimal

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

1. L'automate $\mathcal{A}_L$ est déterministe, accessible et complet.
2. L'automate $\mathcal{A}_L$ est quotient de tout automate DAC reconnaissant L.
3. $\mathcal{A}_L$ est l'unique (à isomorphisme près) automate minimal reconnaissant L.

Équivalence de Nerode

Définition :

Soit $\mathcal{A} = (Q, \Sigma, \delta, F)$ un automate DAC reconnaissant $L \subseteq T_p(\Sigma)$. On définit les équivalences $\sim$ et $(\sim_n)_{n \geq 0}$ sur Q par :

- $q \sim q'$ si $\mathcal{L}(\mathcal{A}_q) = \mathcal{L}(\mathcal{A}_{q'})$ où $\mathcal{A}_q = (Q, \Sigma, \delta, \{q\})$
- $q \sim_0 q'$ si $q, q' \in F$ ou $q, q' \notin F$
- $q \sim_{n+1} q'$ si $q \sim_n q'$ et $\forall a \in \Sigma$ et $\forall q_1, \ldots, q_{i-1}, q_{i+1}, \ldots, q_m \in Q$ on a $\delta(q_1, \ldots, q_{i-1}, q, q_{i+1}, \ldots, q_m, a) \sim_n \delta(q_1, \ldots, q_{i-1}, q', q_{i+1}, \ldots, q_m, a)$

Proposition :

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

1. $\sim = \bigcap_{n \geq 0} \sim_n = \sim_{|Q|}$.
2. $\sim = \equiv_L$.
3. $\mathcal{A}_L$ est le quotient de $\mathcal{A}$ par $\sim$.

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 $\varphi : T(\mathcal{F}) \to A(\mathcal{F})$ est le morphisme canonique.

Exercice : Problèmes de décision et complexité

Lire la section 7 du chapitre 1 du TATA.

Plan

Bibliographie

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

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

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

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

[13] 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 $\{a^{2^n} \mid n > 0\}$

$$\begin{array}{lll} S \to ACaB & Ca \to aaC & CB \to DB & CB \to E \\ aD \to Da & AD \to AC & aE \to Ea & AE \to \varepsilon \end{array}$$

Définition : Dérivation

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

Grammaires de type 0

Définition : Langage engendré

Soit $G = (\Sigma, V, P, S)$ une grammaire et $\alpha \in (\Sigma \cup V)^*$.
Le langage engendré par α est $\mathcal{L}_G(\alpha) = \{u \in \Sigma^* \mid \alpha \xrightarrow{*} u\}$.
Le langage *élargi* engendré par α est $\widehat{\mathcal{L}}_G(\alpha) = \{\beta \in (\Sigma \cup V)^* \mid \alpha \xrightarrow{*} \beta\}$.
Le langage engendré par G est $L_G(S)$.
Un langage est *de type 0* s'il peut être engendré par une grammaire de type 0.

Théorème : Type 0 [16, 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 $\{a^{2^n} \mid n > 0\}$

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

Remarque :

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

Grammaires contextuelles

Définition : Forme normale

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 [9, Prop. 2, p. 156]

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

Théorème : Type 1 [16, Thm 9.5 & 9.6]

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

Théorème : indécidabilité du vide

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

Grammaires contextuelles

Exercices :

1. Montrer que $\{a^{n^2} \mid n > 0\}$ est contextuel.
2. Montrer que $\{ww \mid w \in \{a,b\}^+\}$ est contextuel.
3. Montrer que le problème du mot est décidable en PSPACE pour les langages contextuels.
   Problème du mot : étant donnés un mot w et une grammaire sous contexte G qui engendre un langage L, décider si $w \in L$.

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.

Grammaires algébriques

Lemme : fondamental

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

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

Exercice : Langage de Dyck

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

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

Plan

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 $\alpha_1, \ldots, \alpha_k$ alors $(x, \alpha_1 \cdots \alpha_k) \in P$.

Exemple :

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

Proposition :

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

Arbres de dérivation

Remarques :

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

Ambiguïté

Définition : Ambiguïté

- Une grammaire est ambiguë s'il existe deux arbres de dérivations (distincts) de même racine et de même frontière.
- Un langage algébrique est non ambigu s'il existe une grammaire non ambiguë qui l'engendre.

Exemples :

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

Proposition :

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

Ambiguïté

Exercice : if then else

Montrer que la grammaire suivante est ambiguë.

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

Montrer que le langage engendré n'est pas ambigu.

Grammaires et automates d'arbres

Théorème : du feuillage

- Soit L un langage d'arbres reconnaissable.
  Le langage $\mathrm{Fr}(L)$ 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 $L' = \mathrm{Fr}(L)$.

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

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

Exemple :

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

Corollaire :

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

Lemme d'Ogden

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

Lemme : Ogden

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

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

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.

Exemple :

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

Corollaire :

Les langages non ambigus ne sont pas fermés par union.

Propriétés de clôture

Proposition :

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

Définition : Substitutions algébriques

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

Définition : Projection

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

Propriétés de clôture

Définition : Transduction rationnelle

Une transduction rationnelle (TR) $\tau : A^* \to \mathcal{P}(B^*)$ est la composée d'un morphisme inverse, d'une intersection avec un rationnel et d'un morphisme.

$$\begin{tikzcd} C^* \arrow[r, "\cap K"] & C^* \arrow[d, "\psi"] \\ A^* \arrow[u, "\varphi^{-1}"] \arrow[r, "\tau"] & B^* \end{tikzcd}$$

Soient A, B, C trois alphabets, $K \in \mathrm{Rat}(C^*)$ et $\varphi : C^* \to A^*$ et $\psi : C^* \to B^*$ deux morphismes. L'application $\tau : A^* \to \mathcal{P}(B^*)$ définie par $\tau(a) = \psi(\varphi^{-1}(a) \cap K)$ est une TR.

Proposition :

Les familles Alg et Lin sont fermées par TR.

Propriétés de clôture

Théorème : Chomsky et Schützenberger

Les propositions suivantes sont équivalentes :

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

Corollaire :

Les langages non ambigus ne sont pas fermés par morphisme.

Théorème : Elgot et Mezei, 1965

La composée de deux TR est encore une TR.

Théorème : Nivat, 1968

Une application $\tau : A^* \to \mathcal{P}(B^*)$ est une TR si et seulement si son graphe

$$\{(u, v) \mid v \in \tau(u)\}$$

est une relation rationnelle (i.e., un langage rationnel de $A^* \times B^*$).

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 $x \to \varepsilon$ ou $x \to y$ avec $x, y \in V$.
Un langage $L \subseteq \Sigma^*$ est propre si $\varepsilon \notin L$.

Lemme :

Soit $G = (\Sigma, V, P, S)$ une grammaire.
On peut calculer l'ensemble des variables x telles que $\varepsilon \in \mathcal{L}_G(x)$ (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.
Naïvement on a un algorithme EXPTIME mais ce problème est dans PTIME (cf. HU, p. 139).

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).

Remarque : La réduction d'une grammaire en FNC est encore en FNC.

Remarque : La mise en FNC d'une grammaire propre est une grammaire propre.

Corollaire :

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

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

Lemme :

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

Preuve

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

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

Exemples :

1. $\begin{cases} x_1 \to x_1 b + a \\ x_2 \to x_1 b + a x_2 \end{cases}$
2. $\begin{cases} x_1 \to x_1(x_1 + x_2) + (x_2 a + b) \\ x_2 \to x_1 x_2 + x_2 x_1 + a \end{cases}$

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 :

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

Plan

Automates à pile

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

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

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

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)

Automates à pile

Exercices :

1. Montrer que le langage $\{w\tilde{w} \mid w \in \Sigma^*\}$ et son complémentaire peuvent être acceptés par un automate à pile.
2. Montrer que le complémentaire du langage $\{ww \mid w \in \Sigma^*\}$ peut être accepté par un automate à pile.
3. Soit $\mathcal{A} = (Q, \Sigma, Z, T, q_0, z_0, F)$ un automate à pile. Montrer qu'on peut construire un automate à pile équivalent $\mathcal{A}'$ tel que $T' \subseteq Q' \times Z' \times (\Sigma \cup \{\varepsilon\}) \times Q' \times Z^{\leq 2}$.
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*.
5. Soit $\mathcal{A} = (Q, \Sigma, Z, T, q_0, z_0, F)$ un automate à pile. Pour $(p, x, q) \in Q \times Z \times Q$, on note $\mathcal{L}(p, x, q) = \{h \in Z^* \mid \exists (p, x) \xrightarrow{*} (q, h)\}$ l'ensemble des mots de pile dans l'état q accessibles à partir de (p, x). Montrer que les langages $\mathcal{L}(p, x, q)$ sont rationnels.

Acceptation par pile vide

Définition :

Soit $\mathcal{A} = (Q, \Sigma, Z, T, q_0, z_0)$ un automate à pile.
Le langage accepté par $\mathcal{A}$ *par pile vide* est

$$\mathcal{L}_e(\mathcal{A}) = \{w \in \Sigma^* \mid \exists (q_0, z_0) \xrightarrow{w} (q, \varepsilon) \text{ dans } \mathcal{T}\}.$$

Exemple :

$L = \{a^n b^n \mid n \geq 0\}$.

Proposition : équivalence pile vide / état final

- Soit $\mathcal{A}$ un automate à pile acceptant par état final, on peut construire un automate à pile $\mathcal{A}'$ acceptant par pile vide tel que $\mathcal{L}(\mathcal{A}) = \mathcal{L}_e(\mathcal{A}')$.
- Soit $\mathcal{A}$ un automate à pile acceptant par pile vide, on peut construire un automate à pile $\mathcal{A}'$ acceptant par état final tel que $\mathcal{L}_e(\mathcal{A}) = \mathcal{L}(\mathcal{A}')$.

Exercice :

Montrer l'équivalence avec l'acceptation par pile vide ET état final.

Acceptation par sommet de pile

Définition :

Soit $\mathcal{A} = (Q, \Sigma, Z, T, q_0, z_0, Z')$ un automate à pile avec $Z' \subseteq Z$.
Le langage accepté par $\mathcal{A}$ *par sommet de pile* est

$$\mathcal{L}_z(\mathcal{A}) = \{w \in \Sigma^* \mid \exists (q_0, z_0) \xrightarrow{w} (q, hz) \text{ dans } \mathcal{T} \text{ avec } z \in Z'\}.$$

Exemple :

$L = \{a^n b^n \mid n \geq 0\}$.

Exercice :

Comparer l'acceptation par sommet de pile avec les autres modes d'acceptation.

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* (pas d'ε-transition) 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* ($T \subseteq Z \times \Sigma \times Z^{\leq 2}$).

Calculs d'accessibilité

Exercice :

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

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

Langages déterministes

Définition : Automate à pile déterministe

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

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

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

Exemples :

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. D_n^* peut être accepté par un automate D+TR mais pas par un automate D+S.
3. 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.
4. Le langage $\{a^n b^n \mid n > 0\} \cup \{a^n b^{2n} \mid n > 0\}$ est non ambigu mais pas déterministe.

Langages déterministes

Exercice :

Montrer que le langage $\{a^n b a^n \mid n \geq 0\}$ peut être accepté par *pile vide* par un automate D+TR+S.

Proposition :

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

Exercice :

Donner un automate à pile déterministe qui accepte par pile vide le langage $\{a^n b^p c a^n \mid n, p > 0\} \cup \{a^n b^p d b^p \mid n, p > 0\}$.

Exercice :

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. $(p, x) \not\xrightarrow[\omega]{\varepsilon}$.

Exercice : Montrer que ce critère est décidable.

Remarque :

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

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

Exercice : Montrer que cette construction est effective.

Langages déterministes

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 $\mathcal{A}'$.

Proposition :

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 : $(p, x) \xrightarrow{\varepsilon} (q, \varepsilon)$.

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

Bibliographie

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

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

[16] 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.

Formalisation :

- Un programme est un mot $w \in \Sigma^*$ (Σ est l'alphabet ASCII).
  L'ensemble des programmes syntaxiquement corrects forme un langage $L \subseteq \Sigma^*$.
  Ce langage est algébrique : la syntaxe du langage de programmation est définie par une grammaire $G = (\Sigma, V, P, S)$.
- Pour tester si un programme w est syntaxiquement correct, il faut résoudre le problème du mot : est-ce que $w \in \mathcal{L}_G(S)$?
- L'arbre de dérivation est donné par la suite des règles utilisées lors d'une dérivation gauche (ou droite).

Application à l'analyse syntaxique

Résultats :

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

- en testant toutes les dérivations de longueur au plus $2|w|$ si la grammaire est propre.
- en lisant le mot si on a un automate à pile déterministe complet.

Ceci se fait en temps linéaire par rapport à $|w|$ si l'automate est temps réel ou si les ε-transitions ne font que dépiler.

Problèmes :

- Efficacité de l'algorithme.
- La grammaire qui définit la syntaxe du langage de programmation peut être non déterministe ou ambiguë.

Analyse descendante (LL)

Définition :

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. $G_1 : S \to aSb + ab$.
2. $G_2 : \begin{cases} E & \to & E + T \mid T \\ T & \to & T * F \mid F \\ F & \to & (E) \mid a \mid b \mid c \end{cases}$

Définition :

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

Analyse descendante (LL)

Problème :

L'automate ainsi obtenu est en général non déterministe.

Solutions :

- Si la grammaire n'est pas récursive à gauche ($x \overset{+}{\not\to} x\alpha$), on peut construire un analyseur récursif avec backtracking. Mais l'analyseur obtenu n'est pas efficace.
- 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 : Trunc et First

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

$$\mathrm{First}_k(\alpha) = \mathrm{Trunc}_k(\mathcal{L}_G(\alpha))$$

Remarque :

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

Exemple :

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

Calcul de First_k

Définition : Algorithme de calcul pour First_k

On suppose $k > 0$ et toutes les variables de la grammaire G productives.
Pour $m \geq 0$ et $\alpha \in (\Sigma \cup V)^*$, on définit $X_m(\alpha)$ par :

- si $a \in \Sigma$ alors $X_m(a) = \{a\}$,
- si $x \in V$ alors $X_0(x) = \emptyset$ et $X_{m+1}(x) = \bigcup_{x \to \alpha \in P} X_m(\alpha)$
- si $\alpha = \alpha_1 \cdots \alpha_n$ avec $\alpha_i \in \Sigma \cup V$ alors $X_m(\alpha) = \mathrm{Trunc}_k(X_m(\alpha_1) \cdots X_m(\alpha_n))$.
  Remarque: on obtient en particulier $X_m(\varepsilon) = \{\varepsilon\}$ pour $m \geq 0$.

Proposition : Point fixe

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

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

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

Définition : $\mathrm{LL}(k)$

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

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

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

Exemple :

1. La grammaire G_1 est $\mathrm{LL}(2)$ mais pas $\mathrm{LL}(1)$.
2. La grammaire G_2 n'est pas $\mathrm{LL}(k)$.

Exercice :

- Montrer que si l'automate expansion/vérification associé à une grammaire est déterministe, alors la grammaire est $\mathrm{LL}(0)$.
- Montrer que si G est en FNPG et que pour toutes règles $x \to a\alpha$ et $x \to b\beta$ avec $a, b \in \Sigma$ on a $a \neq b$ ou $\alpha = \beta$, alors G est $\mathrm{LL}(1)$.
- Montrer que la réciproque est fausse.

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

Remarques :

- Étant donné une grammaire G et un entier k, on peut décider si G est $\mathrm{LL}(k)$.
- Étant donné une grammaire G, on ne peut pas décider s'il existe un entier k tel que G soit $\mathrm{LL}(k)$.
- Étant donné une grammaire G, on ne peut pas décider s'il existe une grammaire équivalente qui soit $\mathrm{LL}(1)$.

Exemple :

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

$$G_2' = \begin{cases} E \to TE' & E' \to +TE' \mid \varepsilon \\ T \to FT' & T' \to *FT' \mid \varepsilon \\ F \to (E) \mid a \mid b \mid c & \end{cases}$$

Follow

Définition : Follow

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

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

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

Exemple :

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

Calcul de Follow_k

Définition : Algorithme de calcul pour Follow_k

On suppose toutes les variables de la grammaire G accessibles.
Pour $m \geq 0$ et $x \in V$, on définit $Y_m(x)$ par :

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

Proposition : Point fixe

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

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

Fortement LL

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

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

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

ou encore

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

Exemple :

1. La grammaire G_1 est fortement $\mathrm{LL}(2)$.
2. La grammaire G_2' est fortement $\mathrm{LL}(1)$.

Fortement LL

Proposition :

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

Exemple :

La grammaire

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

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

Proposition :

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

Table d'analyse LL

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

Soit G une grammaire fortement $\mathrm{LL}(k)$.
On définit $M(x,v)$ pour $x \in \Sigma \cup V$ et $v \in \Sigma^{\leq k}$ par

$$M(x,v) = \begin{cases} x \xrightarrow{x} \varepsilon & \text{si } x \in \Sigma \text{ et } v \in x\Sigma^*, \\ x \xrightarrow{\varepsilon} \tilde{\alpha} & \text{si } x \in V \text{ et } v \in \mathrm{First}_k(\alpha \mathrm{Follow}_k(x)), \\ \texttt{erreur} & \text{sinon.} \end{cases}$$

Exemple :

Construire la table d'analyse $\mathrm{LL}(1)$ de la grammaire G_2'.

Analyseur LL

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

Soit G une grammaire fortement $\mathrm{LL}(k)$.
L'analyseur $\mathrm{LL}(k)$ 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 transitions sont données par la table d'analyse de G.

Proposition : Correction

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

Exercice :

Transformer l'analyseur $\mathrm{LL}(k)$ de G en un automate à pile déterministe classique (sans lookahead).