Langages formels

Paul Gastin

Paul.Gastin@lsv.fr

http://www.lsv.fr/~gastin/Langages/

L3 Informatique Cachan
2019-2020

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

Références

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

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

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

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

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

Références

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

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

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

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

Références

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

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

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

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

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

Plan

Plan

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 : Grammaire G_1 pour $\{a^{2^n} \mid n > 0\}$

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

Définition : Dérivation

$\alpha \in (\Sigma \cup V)^*$ se dérive en $\beta \in (\Sigma \cup V)^*$, noté $\alpha \to \beta$, s'il existe $(\alpha_2, \beta_2) \in P$ tel que $\alpha = \alpha_1\alpha_2\alpha_3$ et $\beta = \alpha_1\beta_2\alpha_3$.
On note $\overset{*}{\to}$ 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 : Chomsky 1959, voir [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.

Les règles 6 et 8 de la grammaire G_1 ne sont pas 'non-décroissantes'.

Exemple : Grammaire G_2 contextuelle pour $\{a^{2^n} \mid n > 0\}$

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

Remarque :

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

Grammaires contextuelles

Définition : Forme normale (context-sensitive/contextuelle)

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

Les règles 5, 7 et 8 de la grammaire G_2 ne sont pas en forme normale.

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

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

Grammaires contextuelles

Théorème : Kuroda 1964, voir [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 (Kuroda 1964 pour déterministe)

Étant donnés un mot w et une grammaire G, décider si $w \in L_G(S)$.
Le problème du mot est PSPACE-complet 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 $n \geq 0$.

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

Langages de Dyck

Définition : D_n^*

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

Exercices : Langages de Dyck

1. Montrer que
   $D_1^* = \{w \in \Sigma_1^* \mid |w|_{a_1} = |w|_{\bar{a}_1}$ et $|v|_{a_1} \geq |v|_{\bar{a}_1}$ 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 $D_n^* = \{w \in \Sigma_n^* \mid w \xrightarrow{*} \varepsilon$ dans $R_n\}$.
3. Soit Γ un alphabet disjoint de Σ_n, $\Sigma = \Sigma_n \cup \Gamma$ et $L \subseteq \Sigma^*$ un langage.

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

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

Grammaires linéaires

Définition : Grammaire linéaire

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

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

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

On note Lin la famille des langages linéaires.

Exemples :

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

Proposition :

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

Hiérarchie de Chomsky

Théorème : Chomsky

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

Bibliographie

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

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

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

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

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

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

Plan

Arbres de dérivation

Définition :

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

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

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

Exemple :

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

Définition : Ambigüité

- Une grammaire est ambigüe 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 ambigüe qui l'engendre. Dans le cas contraire, on dit qu'il est *inhéremment ambigu*.

Arbres de dérivation

Lemme : Dérivations et arbres de dérivation

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

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

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

Remarques :

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

Ambigüité

Exemples :

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

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

Exercice : if then else

Montrer que la grammaire suivante est ambigüe.

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

Montrer que le langage engendré n'est pas ambigu.

Grammaires et automates d'arbres

Théorème : (admis)

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

Lemme d'itération

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

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

Exemple :

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

Corollaire :

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

Lemme d'Ogden

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

Lemme : Ogden

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

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

Lemme d'Ogden

Exercice :

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

Corollaire :

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

Exercice :

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

Corollaire :

Les langages non ambigus ne sont pas fermés par union.

Grammaires quadratiques

Définition : Taille d'une grammaire

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

Pour simplifier certains algorithmes ou obtenir une meilleure complexité, il est parfois utile de borner la taille des seconds membres des règles.

Définition : Forme normale quadratique faible (FNQF)

Une grammaire $G = (\Sigma, V, P, S)$ est en FNQF si $P \subseteq V \times (V \cup \Sigma)^{\leq 2}$.

Remarque: Pour une grammaire en FNQF on a $||P|| = \mathcal{O}(|P|)$.

Proposition :

Soit $G = (\Sigma, V, P, S)$ une grammaire.
On peut construire en temps $\mathcal{O}(|G|)$ une grammaire équivalente G' en FNQF.
En particulier, $|G'| = \mathcal{O}(|G|)$.

Grammaires réduites

Définition : Grammaires réduites

La grammaire $G = (\Sigma, V, P, S)$ est réduite si toute variable $x \in V$ est

- productive : $\mathcal{L}_G(x) \neq \emptyset$, i.e., $\exists\ x \xrightarrow{*} u \in \Sigma^*$, et
- accessible : il existe une dérivation $S \xrightarrow{*} \alpha x \beta$ avec $\alpha, \beta \in (\Sigma \cup V)^*$.

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

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

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

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

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

Grammaires propres

Définition : Grammaires propres

La grammaire $G = (\Sigma, V, P, S)$ est propre si $P \subseteq V \times ((\Sigma \cup V)^+ \setminus V)$,
i.e., elle ne contient pas de règle de la forme $x \to \varepsilon$ ou $x \to y$ avec $x, y \in V$.
Un langage $L \subseteq \Sigma^*$ est propre si $\varepsilon \notin L$.

Lemme :

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

Proposition :

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

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

Corollaire :

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

Grammaires quadratiques

Définition : Forme normale de Chomsky (FNC)

Une grammaire $G = (\Sigma, V, P, S)$ est en forme normale de Chomsky si $P \subseteq \{(S, \varepsilon)\} \cup (V \times (V^2 \cup \Sigma))$ et si $(S, \varepsilon) \in P$ alors S n'apparaît dans aucun membre droit.

Proposition :

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

Remarque :

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

Problèmes décidables

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

Soit G une grammaire algébrique.

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

Exercice :

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

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

Proposition : Vide et finitude

Soit G une grammaire algébrique.

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

Problèmes indécidables

Proposition :

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

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

Propriétés de clôture

Proposition :

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

Définition : Substitutions algébriques

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

Définition : Projection

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

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

Transductions rationnelles

Définition : Transduction rationnelle

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

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

Proposition :

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

Transductions rationnelles

Théorème : Chomsky et Schützenberger

Les propositions suivantes sont équivalentes :

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

Corollaire :

Les langages non ambigus ne sont pas fermés par morphisme.

Théorème : Elgot et Mezei, 1965

La composée de deux TR est encore une TR.

Théorème : Nivat, 1968

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

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

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

Forme normale de Greibach

Définition :

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

FNG (forme normale de Greibach)	si $P \subseteq V \times \Sigma V^*$
FNPG (presque Greibach)	si $P \subseteq V \times \Sigma(V \cup \Sigma)^*$
FNGQ (Greibach quadratique)	si $P \subseteq V \times (\Sigma \cup \Sigma V \cup \Sigma V^2)$

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

Théorème :

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

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

Exemple : Mettre la grammaire suivante en FNPG

$$G_1 : \begin{cases} x_1 \to x_1 b + a \\ x_2 \to x_1 b + a x_2 x_1 \end{cases}$$

Forme normale de Greibach

Preuve

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

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

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

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

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

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

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

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

Proposition : Equivalence des grammaires

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

Forme normale de Greibach

Exemple : Mettre la grammaire suivante en FNG(Q)

$$G_2 : \begin{cases} x_1 \to x_1x_1 + x_1x_2 + x_2a + b \\ x_2 \to x_1x_2 + x_2x_1 + a \end{cases}$$

Remarque : Grammaire propre

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

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

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

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

Proposition : FNG et FNGQ

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

Plan

Automates à pile

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

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

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

Définition : Système de transitions (infini) associé

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

Automates à pile

Exemples :

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

Exercices :

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

Propriétés fondamentales

Lemme : fondamental

Soient $\mathcal{A} = (Q, \Sigma, Z, T, q_0 z_0)$ un automate à pile, $p, r \in Q$, $g, h \in Z^*$, $w \in \Sigma^*$ et $n \geq 0$. Les conditions suivantes sont équivalentes:

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

Preuve

$\Longrightarrow$: Dans le calcul $pgh \xrightarrow[n]{w} r$, on considère la première fois que le contenu de la pile est h (possible car initialement gh, finalement ε). Soit qh la configuration correspondante après n_1 étapes. Le calcul s'écrit $pgh \xrightarrow[n_1]{w_1} qh \xrightarrow[n_2]{w_2} r$.

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

$\Longleftarrow$: On utilise la remarque suivante: Si $sf \xrightarrow[k]{v} s'f'$ est un calcul de $\mathcal{A}$ avec $s \in Q$, $f, f', f'' \in Z^*$ alors $sff'' \xrightarrow[k]{v} s'f'f''$ est aussi un calcul de $\mathcal{A}$.

Acceptation généralisée

Définition :

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

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

Cas particuliers :

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

Exemple :

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

Proposition : Acceptation généralisée

Soit $\mathcal{A}$ un automate à pile avec acceptation généralisée K, on peut effectivement construire un automate à pile $\mathcal{A}'$ acceptant par état final tel que $\mathcal{L}_K(\mathcal{A}) = \mathcal{L}(\mathcal{A}')$. Donc, tous les modes d'acceptation ci-dessus sont équivalents.

Automates à pile et grammaires

Proposition : Automate à pile vers grammaire

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 : Grammaire vers automate à pile

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* ($T \subseteq Z \times \Sigma \times Z^{\leq 2}$).

Configurations accessibles (mots de pile)

Proposition : Reconnaissabilité des configurations accessibles

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

$$\mathcal{C}(pg) = \{qh \in QZ^* \mid \exists\ pg \not\Rightarrow qh \text{ dans } \mathcal{T}\}$$

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

Preuve : Voir plus loin.

Corollaire : Décidabilité du vide

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

Preuve

Il suffit de tester si $q_0 \in F$ ou $\mathcal{C}(q_0 z_0) \cap FZ^* \neq \emptyset$.

Calculs d'accessibilité

Corollaire :

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

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

Preuve

1. $(p, x, q) \in X$ ssi $q \in \mathcal{C}(px)$.
2. $(p, x, q, y) \in Y$ ssi $qy \in \mathcal{C}(px)$.
3. $(p, x, q, y) \in W$ ssi $\mathcal{C}(px) \cap qyZ^* \neq \emptyset$.

On applique ce qui précède à l'automate $\mathcal{A}'$ obtenu à partir de $\mathcal{A}$ en ne conservant que les ε-transitions.

Clôture et réduction.

Soit Γ un alphabet, $\overline{\Gamma} = \{\overline{a} \mid a \in \Gamma\}$ une copie de Γ et $\widetilde{\Gamma} = \Gamma \uplus \overline{\Gamma}$.
On définit la réduction sur $\widetilde{\Gamma}^*$ par $\overline{a}a \xrightarrow{\text{red}} \varepsilon$ pour $a \in \Gamma$.

Remarque : Soit $D = \{w \in \widetilde{\Gamma}^* \mid w \xrightarrow[*]{\text{red}} \varepsilon\}$.

On peut montrer que D est engendré par la grammaire $S \to \varepsilon + \sum_{a\in\Gamma} \overline{a}SaS$.
Il s'agit donc du langage de Dyck en considérant $\overline{a}$ comme une parenthèse ouvrante et a comme la parenthèse fermante correspondante.

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

Lemme : Reconnaissabilité de la clôture

Si $L \subseteq \widetilde{\Gamma}^*$ est un langage reconnaissable alors $\mathrm{Clot}(L) \subseteq \widetilde{\Gamma}^*$ aussi.
On peut construire un automate pour $\mathrm{Clot}(L)$ à partir d'un automate pour L (PTime).

Configurations accessibles (Preuve)

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

On définit $\Gamma = Q \uplus Z$ et le langage fini $K = \{qh\overline{x}\,\overline{p} \mid \exists\ (px, a, qh) \in T\} \subseteq \widetilde{\Gamma}^+$.

Lemme : Soit $n \geq 0$

il existe un calcul $pg \overset{}{\underset{n}{\Rightarrow}} qh$ dans $\mathcal{A}$ ssi il existe $w \in K^n$ tel que $wpg \xrightarrow[2n]{\text{red}} qh$

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

Puisque K est un langage fini, le langage K^+pg est reconnaissable et on peut construire (PTIME) un automate $\mathcal{B}$ qui reconnaît $\mathrm{Clot}(K^+pg) \cap QZ^*$.

Calculs d'accessibilité

Exercice : Calculs infinis

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

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

Preuve : Indications

On calcule l'ensemble V comme plus grand point fixe. Pour cela, on définit:

$$
\begin{aligned}
V_0 &= Q \times Z \\
V_{m+1} = \{(p, x) \in Q \times Z \mid\ & \exists (px, a, p_1 y_1 \cdots y_n) \in T, \\
& \exists 1 \leq k \leq n,\ \exists p_2, \ldots, p_k \in Q \text{ tels que} \\
& (p_k, y_k) \in V_m \text{ et } \forall 1 \leq i < k,\ (p_i, y_i, p_{i+1}) \in X\}
\end{aligned}
$$

Montrer alors que $(V_m)_{m \geq 0}$ est une suite décroissante et que

$$V = \bigcap_{m \geq 0} V_m$$

Langages déterministes

Définition : Automate à pile déterministe

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

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

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

Exemples :

1. Le langage $L_4 = \{a^n b^p c^n \mid n, p > 0\} \cup \{a^n b^p d^p \mid n, p > 0\}$ est déterministe mais pas D+TR.
2. $L_5 = \{a^n b a^n \mid n > 0\}$ peut être accepté par un automate D+TR mais pas par un automate D+S car il n'est pas fermé par préfixe.

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 $L_5 = \{a^n b a^n \mid n \geq 0\}$ peut être accepté par *pile vide* par un automate D+TR+S.
2. Le langage $L_4 = \{a^n b^p c^n \mid n, p > 0\} \cup \{a^n b^p d^p \mid n, p > 0\}$ 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 $p\alpha$ et pour toute lettre $a \in \Sigma$ il existe un calcul $p\alpha \xrightarrow[*]{\varepsilon} \xrightarrow{a}$.

Proposition : Critère d'absence de blocage

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

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

De plus, ce critère est décidable.

Remarque :

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

Blocage

Proposition : Suppression des blocages

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

Preuve

$Q' = Q \uplus \{q_0', d, f\}$, $F' = F \uplus \{f\}$, $Z' = Z \uplus \{\bot\}$, $z_0' = \bot$ et pour $p \in Q$, $a \in \Sigma$ et $x \in Z$

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

Cette construction est effective.

Complémentaire

Proposition :

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

Proposition :

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

Exercice :

Montrer que tout langage déterministe est non ambigu.

Langages déterministes

Exercice :

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

Exercice :

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

Lemme d'itération pour les déterministes

Lemme : Itération

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

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

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

Preuve Admis. Voir [4]

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 :

- $R \subseteq L$? $\qquad R \subseteq L \iff R \cap \overline{L} = \emptyset$
- $L = R$? $\qquad R = L \iff R \cap \overline{L} = \emptyset = \overline{R} \cap L$
- L est-il rationnel ?
- $L = L'$? $\qquad$ Géraud Sénizergues, prix Gödel 2002

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

Plan

Plan