X5I0050 - Langages et automates
Hiérarchie de Chomsky

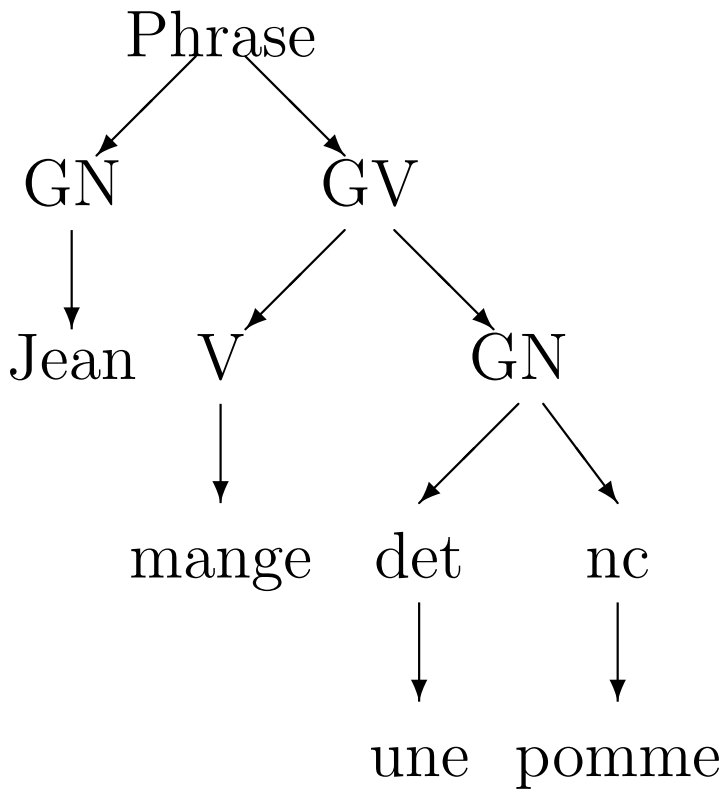
D. Béchet & T. Sadiki

Université de Nantes & Université Internationale de Rabat

12 novembre 2014

Hiérarchie de Chomsky
Introduction

Des grammaires qui décrivent des mots et des langages par **réécriture** :



Grammaires formelles

À un langage L, on associe une **grammaire** G qui est un quadruplet (VT, VN, S, R) avec :
- VT = vocabulaire **terminal** (= alphabet de L)
- VN = vocabulaire **non-terminal** (= symboles intermédiaires)
- S = un symbole de VN appelé **axiome** (symbole départ)
- R = ensemble de **règles** $\alpha \rightarrow \beta$ avec $\beta \in (VT \cup VN)^*$ et $\alpha \in (VT \cup VN)^* VN (VT \cup VN)^*$ (au moins un non-terminal)

$$\text{Ex : } G1 = (\{a, b\}, \{S, A\}, S, \left\{ \begin{array}{l} S \rightarrow bA \\ S \rightarrow a \\ bA \rightarrow a \end{array} \right\})$$

Grammaires formelles

Notations :
- α = partie gauche d'une règle
- β = partie droite d'une règle
- **S** = axiome de toute grammaire

Pour toute grammaire d'un langage non vide :
- $\exists\, \alpha \rightarrow \beta \in R$ tq $\alpha = S$ (règle issue de l'axiome)
- $VT \cap VN = \emptyset$

Autres définitions :
- $\alpha \rightarrow \beta \mid \beta'$ = abréviation des règles $\alpha \rightarrow \beta$ et $\alpha \rightarrow \beta'$
- **L(G)** = langage engendré par G

Grammaires formelles

Notation **Backus-Naur Form** (**BNF**) :
- Tout symbole non-terminal x se note $<\mathbf{x}>$
- Tout mot w sur l'alphabet terminal VT se note $"\mathbf{w}"$
- Toute règle $\alpha \rightarrow \beta$ se note $\alpha' ::= \beta'$ où α' et β' sont les notations BNF de α et β

$$\text{Ex : } G1 = (\{a, b\}, \{S, A\}, S, \left\{ \begin{array}{lcl} <S> & ::= & "b" <A> \mid "a" \\ "b" <A> & ::= & "a" \end{array} \right\})$$

Fonctionnement d'une grammaire : dérivation

Dérivation directe : étant donnée une grammaire $G = (VT, VN, S, R)$,
un mot $u \in (VT \cup VN)^*$ se dérive directement en un mot
$v \in (VT \cup VN)^*$ (noté $\mathbf{u \vdash v}$)
ssi $\exists u_1, u_2 \in (VT \cup VN)^*$ et $r = \alpha \to \beta \in R$ tels que

$$u = u_1 \alpha u_2 \text{ et } v = u_1 \beta u_2$$

Note : on note parfois $\mathbf{u \vdash_r v}$ pour indiquer la règle utilisée

Ex : par $G1 = (\{a, b\}, \{S, A\}, S, \{S \to bA \mid a, bA \to a\})$,
on a $aa\underline{S}ba \vdash_{(S \to a)} aa\underline{a}ba$

Dérivation

Dérivation : étant donné une grammaire $G = (VT, VN, S, R)$, un mot $u \in (VT \cup VN)^*$ se dérive en un mot $v \in (VT \cup VN)^*$ (noté $\mathbf{u \vdash^* v}$) ssi $u = v$ ou $\exists n \geq 0, u_1, \ldots, u_n \in (VT \cup VN)^*$ tels que $u \vdash u_1 \vdash \cdots \vdash u_n \vdash v$

Ex : par $G1 = (\{a, b\}, \{S, A\}, S, \{S \to bA \mid a, bA \to a\})$,
on a $aaS \vdash^* aaa$
car : $aaS \vdash aabA \vdash aaa$ (dérivation indirecte)
mais aussi : $aaS \vdash aaa$ (dérivation directe)

Attention : on a toujours $u \vdash^* u$

Génération

Génération : une dérivation $u \vdash^* v$ est une **génération** ssi $u = S$ (l'axiome) et $v \in VT^*$ (mot terminal)
Ex : pour $G1 = (\{a, b\}, \{S, A\}, S, \{S \to bA \mid a, bA \to a\})$,
$S \vdash^* a$ est une génération

$G = (VT, VN, S, R)$ **engendre L(G)** $= \{w \in VT^* \mid S \vdash^* w\}$
$\Rightarrow$ on retrouve ici une définition ensembliste !

Ex : avec $G1 = (\{a, b\}, \{S, A\}, S, \{S \to bA \mid a, bA \to a\})$
$L(G1) = \{a\}$ car $S \vdash a$ et $S \vdash bA \vdash a$ sont les seules générations

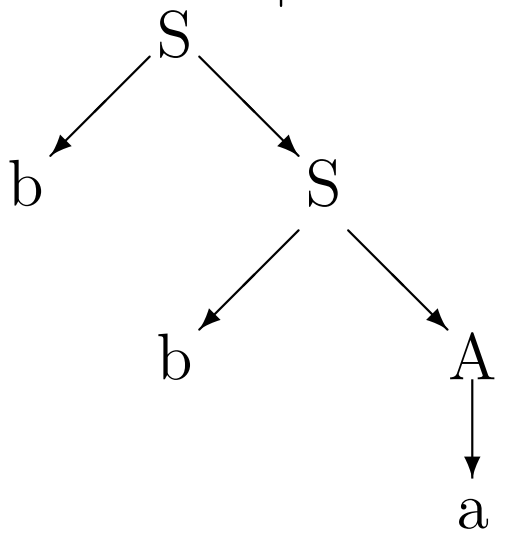
Arbre de dérivation

Génération $\mapsto$ **arbre de dérivation** :

- Axiome $\mapsto$ racine de l'arbre
- Symboles $(VT \cup VN) \mapsto$ noeuds de l'arbre
- Application d'une règle $\alpha \to \beta$ avec $\alpha \in VN$ et $\beta = x_1 x_2 \dots x_n$, $x_i \in VT \cup VN \mapsto$ arcs $(\alpha, x_1), (\alpha, x_2), \dots, (\alpha, x_n)$, de l'arbre

Ex : Soit $G = (\{a, b\}, \{S, A\}, S, \left\{ \begin{array}{lll} S & \to & b \mid bS \mid bA \\ A & \to & a \mid aA \end{array} \right\})$

$S \vdash bS \vdash bbA \vdash bba \mapsto$



Génération VS Reconnaissance

Pourquoi et comment utiliser les grammaires ?

- **Génération** : utiliser mécaniquement les règles pour produire toutes les générations
- **Reconnaissance** : générer jusqu'à trouver le mot à reconnaître

Problème : dérivations infinies possibles !

Ex : $G3 = (\{a, b\}, \{S, A\}, S, \{S \to A, A \to S\})$
donne la dérivation infinie $S \vdash A \vdash S \vdash A \vdash \cdots$

$\Rightarrow$ détecter ces générations = caractériser les grammaires et les langages

Exemples de grammaires et de langages

1. Le langage vide :
   $L((VT, \{S\}, S, \emptyset)) = \emptyset$
2. Le langage contenant uniquement le mot vide :
   $L((VT, \{S\}, S, \{S \rightarrow \epsilon\})) = \{\epsilon\}$
3. Le langage contenant un unique mot (symbole) :
   $L((\{a\}, \{S\}, S, \{S \rightarrow a\})) = \{a\}$
4. Une suite de symboles :
   $L((\{a\}, \{S\}, S, \{S \rightarrow \epsilon \mid Sa\})) = \{a\}^*$
5. Tout langage rationnel (exprimé par une expression rationnelle)

Exemples de grammaires et de langages

1. Parenthèses imbriquées :
   $L((\{ab\},\{S\},S,\{S \to \epsilon \mid aSb\})) = \{a^n b^n, n \geq 0\}$
   $L((\{()\},\{S\},S,\{S \to \epsilon \mid (S)\})) = \{(^n)^n, n \geq 0\}$
2. Phrases/expressions bien parenthésées :
   $L((\{()a\},\{S\},S,\{S \to \epsilon \mid a \mid S(S)\})) =$ *mots sur* $\{()a\}$ *où chaque parenthèse ouvrante correspond à une unique parenthèse fermante*
3. **Langage de Dyck** sur $\{()\}$:
   $L((\{()\},\{S\},S,\{S \to \epsilon \mid S(S)\})) =$ *mots sur* $\{()\}$ *où chaque parenthèse ouvrante correspond à une unique parenthèse fermante*
4. **Langage de Dyck** sur n couples de parenthèses
   $VT = \{(_1)_1 \cdots (_n)_n\}$:
   $L((VT,\{S\},S,\{S \to \epsilon \mid S(_1S)_1 \mid \cdots S(_nS)_n\}))$

Exemples de grammaires et de langages

1. Partie syntaxique des langages de programmation : C, C++, Java...
2. Format de données comme HTML, XML...
3. Spécification formelle d'un programme, formule logique, démonstration formelle...
4. Spécification informelle, langage mathématique ?
5. Français, anglais ?

Hiérarchie de Chomsky

Une grammaire $G = (VT, VN, S, R)$ est de :

- **type 3** ssi $\forall \alpha \rightarrow \beta \in R$, $\alpha \in VN$ et $\beta \in (VT^* \cdot VN) \cup VT^+$
- **type 2** ssi $\forall \alpha \rightarrow \beta \in R$, $\alpha \in VN$ et $\beta \in (VT \cup VN)^+$
- **type 1** ssi $\forall \alpha \rightarrow \beta \in R$ $\alpha\beta \in (VT \cup VN)^+$ et $\mid \alpha \mid \leq \mid \beta \mid$
- **type 0** sinon

Exception : une règle $\alpha \rightarrow \epsilon$ est autorisée dans des grammaires de types 1, 2 et 3 ssi $\alpha = S$ (axiome) et S n'apparaît jamais en partie droite (β) d'aucune règle

- Grammaire de type 3 = **grammaire rationnelle** (ou régulière)
- Grammaire de type 2 = **grammaire algébrique** (ou hors-contexte)
- Grammaire de type 1 = **grammaire contextuelle**

Classification de grammaires

Inclusion : $\forall i > j$, G de type $i \Rightarrow G$ de type j

$$G \text{ de type } 3 \Rightarrow G \text{ de type } 2 \Rightarrow G \text{ de type } 1 \Rightarrow G \text{ de type } 0$$

Classification : déterminer le type d'une grammaire

Inclusion $\mapsto$ essayer le type le plus haut en premier

Ex : $G1 = (\{a, b\}, \{S, A\}, S, R = \{S \rightarrow bA \mid a, bA \rightarrow a\})$

- pas type 3 car $bA \rightarrow a \in R$ et $bA \notin VN$
- pas type 2 (même raison)
- pas type 1 car $bA \rightarrow a \in R$ et $\mid bA \mid > \mid a \mid$
- donc type 0

Ex : $G2 = (\{a, b\}, \{S\}, S, \{S \rightarrow a\})$

De type 3 car $S \in VN$ et $a \in VT^+$

Et pourtant $G1$ et $G2$ engendrent le même langage $\{a\}$

Classification des langages

Un langage est de **type** *i* ssi il peut être engendré par une grammaire de type i et par aucune grammaire de type $j > i$

Dénomination des langages :

- Type 0 = **langages récursivement énumérables** (ou quelconques)
- Type 1 = **langages contextuels**
- Type 2 = **langages algébriques** (ou non contextuels)
- Type 3 = **langages rationnels** (ou réguliers)

Génération VS Reconnaissance

Pour certaines classes de langages, on dispose d'autres formalismes de représentation qui permettent de répondre plus simplement aux problèmes de génération ou de reconnaissance

C'est le cas des langages rationnels qui peuvent être représentés par :

- Grammaires rationnelles (type 3)
- Expressions rationnelles
- Automates finis

Théorème :

langage de type 3 = langage rationnel

Génération VS Reconnaissance

Les langages algébriques (hors contextes) peuvent être représentés par :

- Grammaires algébriques (type 2)
- Automates à pile
- Grammaires en forme normale de Chomsky
- Grammaires en forme normale de Greibach

Les langages contextuels peuvent être représentés par :

- Grammaires contextuelles (type 1)
- Automates bornés linéairement

Les langages récursivement énumérables peuvent être représentés par :

- Grammaires générales (type 0)
- Machine de Turing

Décidabilité de la reconnaissance

Le problème de l'appartenance à un langage de type n est plus ou moins difficile : soit G une grammaire et m un mot, peut-on savoir si $m \in L(G)$?

- Langages rationnels (grammaires de type 3) : le problème est décidable (complexité linéaire avec un automate fini déterministe, compléxité carrée n^2 avec un automate quelconque)
- Langages algébriques (grammaires de type 2) : le problème est décidable (complexité cubique n^3 avec l'algorithme CYK)
- Langages contextuels (grammaires de type 1) : le problème est décidable (algorithmes de reconnaissance exponentiels en temps comme la recherche ascendante par force brute $\Rightarrow$ peu utilisés)
- Langage non-contraint (grammaires de type 0) : le problème est indécidable (langages pas utilisés pour faire de la reconnaissance sauf dans des cas très particuliers)

Langage quelconque - Reconnaissance

Algorithme de recherche ascendante par "force brute"

Entrées

- Une grammaire G (si possible contextuelle)
- Un mot m

Sorties (si le programme termine)

- Une réponse oui/non sur l'appartenance de m au langage $L(G)$
- Certaines (si réponse oui) / toutes (si réponse non) les listes de symboles terminaux/non-terminaux dérivant m

L'algorithme termine toujours si la grammaire est contextuelle

Recherche ascendante par force brute

m en entrée

$M = New = \{m\}$
Tant que *New* est non vide et ne contient pas S faire
  $Old = New$ et $New = \emptyset$
  Pour $l \in Old$ et $\alpha \rightarrow \beta$ une règle de G
    Si $k \vdash_{\alpha \rightarrow \beta} l$ et $k \notin M$ alors
      Ajouter k à M et à New
m appartient au langage ssi $S \in M$

Recherche ascendante par force brute - Exemple

$G1 = (\{a, b\}, \{S, A\}, S, \{S \rightarrow bA \mid a, bA \rightarrow a\})$
$m = a$

1. $M_0 = New_0 = \{a\}$
2. Première itération : $Old_1 = \{a\}$
   $New_1 = \{S, bA\}$ ($S \rightarrow a$ et $bA \rightarrow a$) et $M_1 = \{a, S, bA\}$
3. Deuxième itération : $Old_2 = \{S, bA\}$
   $New_2 = \emptyset$ ($S \rightarrow bA$ ne donne rien de nouveau) et $M_2 = M_1$
4. Fin de la boucle
5. a appartient au langage car $S \in M_2$

Recherche ascendante par force brute - Exemple

$G1 = (\{a, b\}, \{S, A\}, S, \{S \to bA \mid a, bA \to a\})$
$m = ba$

1. $M_0 = New_0 = \{ba\}$
2. Première itération : $Old_1 = \{ba\}$
   $New_1 = \{bS, bbA\}$ ($S \to a$ et $bA \to a$) et $M_1 = \{ba, bS, bbA\}$
3. Deuxième itération : $Old_2 = \{bS, bbA\}$
   $New_2 = \emptyset$ ($S \to bA$ ne donne rien de nouveau) et $M_2 = M_1$
4. Fin de la boucle
5. ba n'appartient pas au langage car $S \notin M_2$

Langage quelconque et force brute

Algorithme donnant une semi-réponse :

- Réponse “oui” si le mot appartient au langage
- Réponse “non” ou boucle infiniment (pas de réponse) si le mot n’appartient pas au langage

Le problème est semi-décidable :

Problème équivalent au problème de l’arrêt d’une machine de Turing

Langage contextuel et force brute

Algorithme donnant une réponse :

- Réponse “oui” si le mot appartient au langage
- Réponse “non” si le mot n’appartient pas au langage

Le problème est décidable :

car l’ensemble M est toujours fini

Forme normale de Chomsky

Une grammaire $G = (VT, VN, S, R)$ est en **forme normale de Chomsky** si les règles de R sont de la forme :

$$\begin{array}{lcll} A & \rightarrow & BC & A \in VN \text{ et } B, C \in VN - \{S\} \\ A & \rightarrow & a & A \in VN \text{ et } a \in VT \\ S & \rightarrow & \epsilon & S = axiome \end{array}$$

Théorème :

Toute grammaire algébrique peut être transformée en une grammaire en forme normale de Chomsky

Utilité :

Simplification des algorithmes de reconnaissance

Exemple de forme normale de Chomsky

Exemple :

$$G = (\{a, b\}, \{S, T\}, S, \left\{ \begin{array}{lcl} S & \rightarrow & aTb \mid ab \mid \epsilon \\ T & \rightarrow & aTb \mid ab \end{array} \right\})$$

On obtient la grammaire suivante :

$$G = (\{a, b\}, \{S, T, A, B, C\}, S, R') \text{ avec}$$

$$R' = \left\{ \begin{array}{lcl} S & \rightarrow & AC \mid AB \mid \epsilon \\ C & \rightarrow & TB \\ T & \rightarrow & AC \mid AB \\ A & \rightarrow & a \\ B & \rightarrow & b \end{array} \right\}$$

Transformation en forme normale de Chomsky

La transformation est un peu technique :

1. Elinimer les règles $X \rightarrow \epsilon$ pour $X \neq S$
2. Eliminer les règles $X \rightarrow Y$
3. Introduire un symbole non-terminal X pour chaque symbole terminal x, ajouter la règle $X \rightarrow x$ et remplacer x par X dans le membre droit des règles comportant plusieurs symboles
4. Introduire des symboles non-terminaux supplémentaires lorsqu'un membre droit comporte plus de 2 symboles

Forme normale de Chomsky et analyse syntaxique

Algorithme Cocke-Younger-Kasami (CYK)

Entrées

- Une grammaire algébrique G sous forme normale de Chomsky
- Un mot m

Sorties

- Une table $T[i, j]$ donnant les non-terminaux associés aux segments de symboles de m de longueur i commençant au symbole j
- Une réponse oui/non sur l'appartenance de m au langage $L(G)$

Complexité : $O(n^3)$ avec $n = taille(G) + taille(m)$

Algorithme Cocke-Younger-Kasami (CYK)

m en entrée

Pour le symbole (terminal) a de m d'occurrence j
  Pour chaque règle $a \rightarrow X$, ajouter X à $T[1, j]$
Pour i de 2 à $longueur(m)$ faire :
  Pour j de 1 à $longueur(m) - i + 1$ faire :
    Pour k de 1 à $i - 1$ faire :
      Pour chaque règle $X \rightarrow YZ$,
        si $Y \in T[k, j]$ et $Z \in T[i - k, j + k]$, ajouter X à $T[i, j]$
m appartient au langage si $S \in T[longueur(m), 1]$

Algorithme Cocke-Younger-Kasami (CYK) - Exemple

$$G = (\{a, b\}, \{S, T, A, B, C\}, S, R) \text{ avec } R = \left\{ \begin{array}{lcl} S & \rightarrow & AC \mid AB \mid \epsilon \\ C & \rightarrow & TB \\ T & \rightarrow & AC \mid AB \\ A & \rightarrow & a \\ B & \rightarrow & b \end{array} \right\}$$

$m = aabb$

	$T[..., 1]$	$T[..., 2]$	$T[..., 3]$	$T[..., 4]$
$T[1, ...]$	$A(A \rightarrow a)$	$A(A \rightarrow a)$	$B(B \rightarrow b)$	$B(B \rightarrow b)$
$T[2, ...]$	$\emptyset$	$S, T(S/T \rightarrow AB)$	$\emptyset$	
$T[3, ...]$	$\emptyset$	$C(C \rightarrow TB)$		
$T[4, ...]$	$S, T(S/T \rightarrow AC)$			

Forme normale de Greibach

Une grammaire $G = (VT, VN, S, R)$ est en **forme normale de Greibach** si les règles de R sont de la forme :

$$\begin{array}{lcll} A & \rightarrow & aB_1 \cdots B_n & A \in VN, a \in VT, n \geq 0, B_1, \ldots, B_n \in VN - \{S\} \\ S & \rightarrow & \epsilon & S = axiome \end{array}$$

Théorème :

Toute grammaire algébrique peut être transformée en une grammaire en forme normale de Greibach

Utilité :

Algorithmes de reconnaissance **lexicalisés**

Exemple de forme normale de Greibach

Exemple :

$$G = (\{a, b\}, \{S, T\}, S, \left\{ \begin{array}{lcl} S & \rightarrow & aTb \mid ab \mid \epsilon \\ T & \rightarrow & aTb \mid ab \end{array} \right\})$$

On obtient la grammaire suivante :

$$G = (\{a, b\}, \{S, T, B\}, S, R') \text{ avec}$$

$$R = \left\{ \begin{array}{lcl} S & \rightarrow & aTB \mid aB \mid \epsilon \\ T & \rightarrow & aTB \mid aB \\ B & \rightarrow & b \end{array} \right\}$$

Transformation en forme normale de Greibach

La transformation est un peu technique :

1. Eliminer les règles $X \rightarrow \epsilon$ pour $X \neq S$
2. Transformer les règles avec récursivité gauche $X \rightarrow X...$ en règles avec récursivité droite $X \rightarrow ...X$
3. Transformer les règles avec un symbole non-terminal à gauche de la partie droite $X \rightarrow Y...$ pour avoir un terminal $X \rightarrow a...$

Les points (2) et (3) doivent être appliqués plusieurs fois.