Grammaires formelles

Définitions - grammaires hors contexte

Dérivation

Arbres syntaxiques et ambiguïtés

Forme BNF

Automates à pile

Eléments d'analyse syntaxique

Analyse descendante

Grammaires générales

Langages

Un langage est un ensemble fini ou infini de chaînes de symboles tirés d'un alphabet S.

Exemple:

S = {a, b, X,!}

L1 = {a, bX, bXXa, !!, }

L2 = toutes les chaînes qui commencent par a

L3 = toutes les chaînes qui ont autant de X que de b

L4 = les chaînes formées d'exactement 3 symboles

Comment définir un langage ?

Expressions régulières

- ne marche pas toujours (langages non réguliers)

Idée de Backus, Naur et Chomsky

- utiliser un système formel
  axiomes, règles de dérivation, théorèmes => chaînes grammaticales
- distinguer deux catégories de symboles: variables (« temporaires ») et terminaux

Objectifs (historiques)

Backus et Naur: définir formellement la syntaxe du langage de programmation Algol 58/60

- définir tous les langages de programmation

Chomsky: définir des grammaires génératives pour étudier les langues naturelles

- règles pour générer toutes les phrases correctes

Grammaires hors contexte

Une GHC est formée

d'un ensemble V de **symboles non terminaux** (ou variables)

d'une ensemble T de **symboles terminaux**

d'une ensemble R de **règles de production** de la forme

$$X \to w$$

où $X \in V$ et

$w \in (V \cup T)^*$ (chaîne de symboles de V et T)

d'un **symbole initial** S ∈ V

Dérivation (production)

Application d'une règle sur une chaîne

la règle : **X** → **w**

appliquée à la chaîne : **p X q**

produit : **p w q**

Exemple

A → **a C d C** sur **c A b** donne **c a C d C b**

Choix de la dérivation

P : **X** → **w**

u : **p X q X r**

On peut appliquer P sur n'importe quelle occurrence de X

Plusieurs dérivations possibles.

Exemple

C → **b b d** sur **b C a C**

1er choix : **b b b d a C**

2e choix : **b C a b b d**

Séquences de dérivation

Application successive de différentes règles à partir de u :

$u \to_{P1} u_1$

$u_1 \to_{P2} u_2$

...

$u_{k-1} \to_{Pk-1} u_k$

$u_k \to s$

On note $u \to^* s$.

(avec la règle **C** → **b b d**)

b C a C → **b b b d a C** → **b b b d a b b d**

Existence d'une dérivation

$$u \rightarrow^* s$$

signifie:

il existe des chaînes u_1, u_2, ..., u_k telles que

$$u \rightarrow u_1 \text{ et } u_1 \rightarrow u_2 \text{ et } \ldots \text{ et } u_{k-1} \rightarrow u_k \text{ et } u_k \rightarrow s$$

Dans les systèmes formels : preuve du théorème **s** à partir de l'axiome **u**.

Dérivations multiples

Soit les règles

1. $S \rightarrow XY$;
2. $X \rightarrow aY$;
3. $X \rightarrow a$;
4. $Y \rightarrow c$

On peut appliquer les dérivations dans des ordres différents et produire la même chaîne terminale:

d1: **S** $\rightarrow_1$ **XY** $\rightarrow_2$ **aYY** $\rightarrow_4$ **acY** $\rightarrow_4$ **acc**

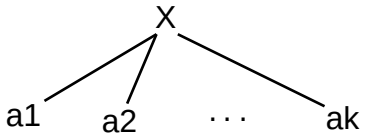
d2: **S** $\rightarrow_1$ **XY** $\rightarrow_4$ **Xc** $\rightarrow_2$ **aYc** $\rightarrow_4$ **acc**

Arbre de dérivation / syntaxique

S'abstraire de l'ordre d'application des règles.

Trouver les dérivations fondamentalement différentes

On représente une dérivation $X \rightarrow a_1\ a_2\ \ldots\ a_k$ par

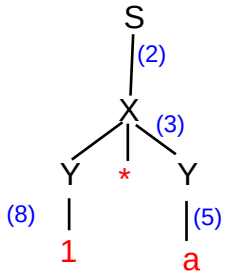


- La racine de l'arbre est le symbole initial
- Les feuilles sont des symboles terminaux
- La chaîne produite correspond à lire les feuilles de gauche à droite

Exemple (1)

(1) S $\rightarrow$ X + X ; (2) S $\rightarrow$ X ; (3) X $\rightarrow$ Y * Y ; (4) X $\rightarrow$ Y ;

(5) Y $\rightarrow$ a ; (6) Y $\rightarrow$ b ; (7) Y $\rightarrow$ 0 ; (8) Y $\rightarrow$ 1 ;

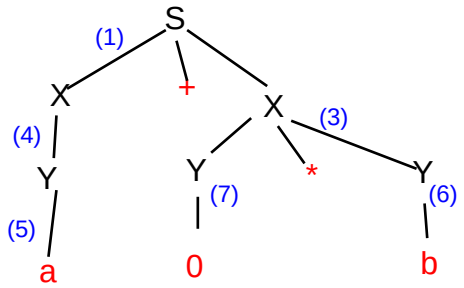


Dérivation de 1*a

Exemple (2)

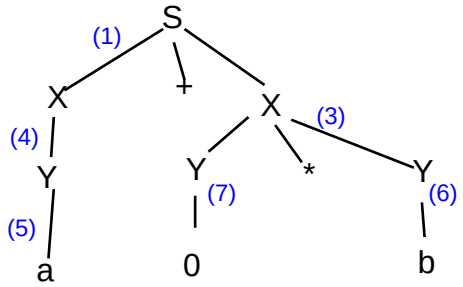
(1) S → X + X ; (2) S → X ; (3) X → Y * Y ; (4) X → Y ;

(5) Y → a ; (6) Y → b ; (7) Y → 0 ; (8) Y → 1 ;



Dérivation de a+0*b

Arbre et dérivations



Représente (entre autres) les dérivations :

$S \rightarrow_1 X+X \rightarrow_4 Y+X \rightarrow_3 Y+Y*Y \rightarrow_5 a+Y*Y \rightarrow_7 a+0*Y \rightarrow_6 a+0*b$

$S \rightarrow_1 X+X \rightarrow_3 X+Y*Y \rightarrow_7 X+0*Y \rightarrow_4 Y+0*Y \rightarrow_6 Y+0*b \rightarrow_5 a+0*b$

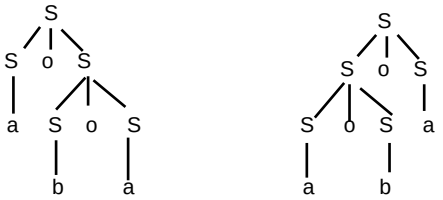
Ambiguïtés

Une grammaire est ambiguë s'il existe plusieurs arbres pour dériver la même chaîne de symboles:

S → S o S;

S → a; S → b;

Il y a deux arbres pour dériver la chaîne a o b o a :



Signification de l'ambiguïté

Chaîne ambiguë

=> deux arbres syntaxiques

=> deux manière d'analyser cette chaîne

Ex. informel en français:

« La belle ferme le voile »

a) Proposition → Sujet Verbe Complément --> ...

(*La belle*) (*ferme*) (*le voile*)

b) Proposition → Sujet Complément Verbe --> ...

(*La belle ferme*) (*le*) (*voile*)

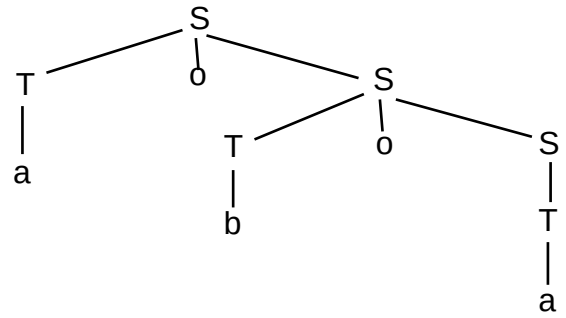
Suppression d'ambiguïté

P.ex. Introduire un nouveau symbole non-terminal T

S → **T** o S; S → **T**;

T → a; **T** → b;

Analyse de a o b o a (seul arbre possible)



Tester l'ambiguité

Mauvaise nouvelle

Il n'y a pas d'algorithme pour tester si une grammaire hors contexte est ambiguë ou non.

Forme de Backus et Naur (BNF)

La forme BNF simplifie l'écriture des grammaire en regroupant plusieurs règles en une seule règle BNF.

BNF de base

- les symboles terminaux sont écrits entre " et "
- la → de dérivation est remplacée par ::=
- le symbole | sert à exprimer des alternatives

T ::= "a" B "c" remplace **T → a B c**

T ::= w_1 | w_2 | ... | w_k remplace

T → w_1, T → w_2, ..., T → w_k

Exemple

(1) S → X + X ; (2) S → X ;

(3) X → Y * Y ; (4) X → Y ;

(5) Y → a ; (6) Y → b ; (7) Y → 0 ; (8) Y → 1 ;

En BNF

S ::= X "+" X | X

X ::= Y "*" Y | Y

Y ::= "a" | "b" | "0" | "1"

BNF: Extensions

Répétitions 0, 1 ou plusieurs fois

T ::= { h } signifie: T → ε; T → hT ;

Option (0 ou 1 fois)

T ::= [h] signifie: T → ε; T → h ;

Répétitions 1 ou plusieurs fois

T ::= (h)$^{+}$ signifie: T → h; T → hT ;

BNF: Extensions (2)

Inclusion de sous-expressions

T ::= w1 *expression BNF* w2

signifie:

T ::= w1 Tb w2

Tb ::= *expression BNF*.

p.ex.

A ::= "top" { "tic" "tac" ["oups"] } ({ "dring" } | "toc" | "boum")

```
top tic tac tic tac tic tac boum
top tic tac tic tac tic tac tic tac dring dring dring
top tic tac tic tac oups tic tac tic tac oups tic tac
```


Ex. BNF d'expressions arithmétiques

Expression ::= Terme { ("+" | "–") Terme }

Terme ::= Facteur { ("*" | "/") Facteur }

Facteur ::= Nombre | Variable | "(" Expression ")"

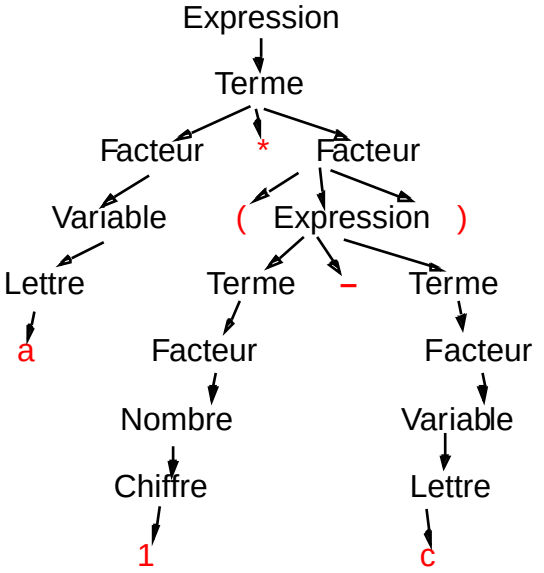
Nombre ::= ["–"] Chiffre { Chiffre }

Chiffre ::= "0" | "1" | …… | "9"

Variable ::= Lettre { Lettre | Chiffre | "_" | "$" }

Lettre ::= "a" | "b" | …… | "Z"

Exemple: a*(1-c)



Exemple : BNF d'un langage algorithmique

```
Instruction ::= Affectation | Condition | Itération |
            "{" {Instruction } "}"
```

Affectation ::= Variable "=" Expression ";"

```
Condition ::= "if" "(" ExpressionLogique ")" Instruction
    ["else" Instruction]
```

Itération ::= "while" "(" ExpressionLogique ")" Instruction

```
ExpressionLogique ::= Comparaison |
    ExpressionLogique ("&&" | "||") ExpressionLogique
```

Comparaison ::= Expression ("==" | ">" | "<" | …) Expression

BNF de la BNF

```
syntax     ::=  { rule }
rule         ::=  identifier  "::="  expression
expression ::=  term { "|" term }
term       ::=  factor { factor }
factor     ::=  identifier | quoted_symbol |
                "("  expression  ")" |
                "["  expression  "]" |
                "{"  expression  "}"
identifier ::=  letter { letter | digit }
quoted_symbol ::= """ { any_character } """
```


Analyse syntaxique

Problème:

étant donné une chaîne de symboles terminaux, cette chaîne appartient-elle au langage engendré par la grammaire G ?

ou bien

reconstruire l'arbre syntaxique qui produit une chaîne donnée.

On cherche à automatiser cette analyse, c'est-à-dire à définir une procédure d'analyse pour chaque langage.

Domaine de l'informatique:

"parsing", *analyseurs syntaxiques, compilateurs*

Automates à pile

Automates avec une mémoire

Capables de reconnaître les langages hors contexte

- plus forts que les automates finis

Principe

- on dispose d'une mémoire à empilement (pile)

à chaque étape

- le symbole au sommet de la pile peut être lu et enlevé
- un nouveau symbole peut être empilé

Définition

Un automate à pile est formé de

- un ensemble fini d'états Q
- un état initial $q_1 \in Q$, un ensemble d'état finals F
- un alphabet d'entrée A
- un alphabet de pile P
- un ensemble de transitions

$$q_i \; a \; u \rightarrow v \; q_j$$

« Dans l'état q_i si on lit le symbole a et que le symbole u est au sommet de la pile, enlève u de la pile, empile v et passe à l'état q_j »

Transitions spéciales

$$q_i \; a \; u \rightarrow v \; q_j$$

si a = 0 : pas de lecture de symbole d'entrée

action indépendante du symbole courant

si u = 0 : le sommet de la pile n'est pas enlevé

si v = 0 : n'empile rien

Acceptation

Configuration d'un AP : (no. symb. entrée, état, pile)

si D = (k, q_i, uα) et le k^e symbole d'entrée est b,

une transition q_i a u $\rightarrow$ v q_j fait passer à

D' = (k+1, q_j, vα) si a = b

D' = (k, q_j, vα) si a = 0

(0α = α)

Un chaîne est acceptée si on atteint un état final et que la pile est vide.

Exemple 1

Alpha. = {a, b}, Alpha. Pile = {u}, Q = {q1, q2}, F {q2}

Transitions

q1 a 0 $\rightarrow$ u q1

q1 b u $\rightarrow$ 0 q2

q2 b u $\rightarrow$ 0 q2

$L = a^n b^n$

Exemple 2

Alpha. = {a, b, c}, Alpha. pile = {A, B}, Q = {q1, q2}, F {q2}

Transitions

q1 a 0 → A q1
q1 b 0 → B q1
q1 c 0 → 0 q2
q2 a A → 0 q2
q2 b B → 0 q2

$L = wcw^R$

Exemple 3 - non déterministe

Alpha. = {a, b}, Alpha. pile = {A, B}, Q = {q1, q2}, F {q2}

Transitions

q1 a 0 → A q1
q1 b 0 → B q1
q1 a A → 0 q2
q1 b B → 0 q2
q2 a A → 0 q2
q2 b B → 0 q2

$L = ww^R$

Langages à parenthèses PAR_3

Terminaux : { a, (,), [,], {, } }

Non terminal, initial : S

Règles

S → a
S → (S)
S → [S]
S → { S }
S → S S
S → 0 (chaîne vide)

Langages à parenthèses PAR_n

Terminaux : { a, $(_1$, $)_1$, …, $(_n$, $)_n$ }

Non terminal, initial : S

Règles

S → a
S → $(_i$ S $)_i$ $\quad i = 1, 2, \ldots, n$
S → S S
S → 0 (chaîne vide)

Automate pour PAR_n

Alpha. = {a, $(_1$, $)_1$, ..., $(_n$, $)_n$ }, Alpha. pile = {J_1, ..., J_n},

Q = {q1}, F {q1}

Transitions

q1 a 0 $\rightarrow$ 0 q1

q1 $(_i$ 0 $\rightarrow$ J_i q1

q1 $)_i$ J_i $\rightarrow$ 0 q1

Analyse de tout langage hors contexte

Forme normale de Chomsky

Une grammaire est en forme normale de Chomsky si toutes les productions sont de la forme

$$X \rightarrow Y\ Z \qquad \text{ou} \qquad X \rightarrow a$$

Théorème

Il y a un algorithme qui transforme toute ghc G (positive) en une grammaire G' en fn Chomsky telle que L(G') = L(G).

(suite)

Séparateur G_s d'une grammaire en fn Chomsky

G_s est obtenue en remplaçant chaque production

$$X_i \rightarrow Y_i\ Z_i$$

par

$$X_i \rightarrow (_i\ Y_i\ )_i\ Z_i$$

Théorème.

G une grammaire en forme normale de Chomsky, T les symboles terminaux

Il existe un lanage régulier R tel que

$$L(G_s) = R \cap PAR_n(T)$$

Automate d'analyse

$$L(G_s) = R \cap PAR_n(T)$$

On peut construire

un automate fini pour R,

un automate à pile pour $PAR_n(T)$

un automate à pile pour l'intersection des deux.

Donc, tout langage hors contexte est analysable par un automate à pile.

Grammaires et SI

Syntaxe des langages

- de programmation, de bases de données
- de modélisation (UML, …)
- de marquage de documents (HTML, LaTeX, etc.)
- de saisie d'information (dates, nombres, etc.)

Définition de structures

- structures de documents
- structures de données

Exemple: XML

Documents typés

Marquer sémantiquement le contenu des documents

- distinguer un prix, une longueur, une adresse, un nom de famille, un prénom, dans un texte

Fournir des "schémas" de documents adaptés au domaine d'application

- liste de prix, formulaire d'inscription, livre, article, mode d'emploi, …

Séparer le contenu de la présentation

- spécifier la présentation indépendamment du document

Marquage d'un document

Ajouter des balises dans le texte, indiquer la nature du texte.

```
<chapitre>
    <titre>Introduction à la signification des choses
    </titre>
    <auteur>H. De Paris</auteur>
    <paragraphe>Longtemps on a considéré … … …
       … … … </paragraphe>
    <paragraphe> … </paragraphe>
    <référence><auteur>W. Floyd.</auteur>
            <titre> "Tout est simple"</titre>
    </référence>
</chapitre>
```


Permet des traitements plus fins

– Cibler les recherches ou l'indexation

  chercher le mot "informatique" dans un titre de chapitre / un titre de référence.

– Créer des documents dérivés

  une liste des auteurs référencés, une table des matières

– Distinguer les différents rôles d'un mot

  Paris en tant que nom d'auteur ou mot d'un titre

– Restructurer les documents

– Réutiliser/intégrer les documents

Définition de types de documents

Le texte entre <balise> et </balise> est un **élément**

Son **type** est donné par le nom de la balise.

Un document est composé d'**éléments imbriqués**

- quels sont les types d'éléments admis dans ce document ?

  p.ex. chapitre, paragraphe, auteur, … dans un livre

- de quoi est composé chaque élément ?

  p.ex. un chapitre est fait d'un auteur, d'un titre, de paragraphes et de références

Grammaire d'un document

Les types d'éléments sont des symboles non terminaux

Une expression de type BNF spécifie la « composition » de chaque type d'élément

```
<!ELEMENT adresse (numéro, rue, ville)>
<!ELEMENT numéro (#PCDATA)>    <!-- des caractères -->
<!ELEMENT rue (#PCDATA)>
<!ELEMENT ville (#PCDATA)>
```

Document

```
<adresse>  <numéro>25</numéro><rue>Lausanne</rue>
           <ville>Genève</ville>
</adresse>
```


Répétitions et options

<!ELEMENT chapitre (titre, auteur, paragraphe*****)>

Un chapitre = un titre, suivi d'un auteur, suivi de 0, 1 ou plusieurs paragraphes.

<!ELEMENT chapitre (titre, auteur, paragraphe**+**)>

Un chapitre doit contenir au moins un paragraphe.

<!ELEMENT chapitre (titre, auteur?, paragraphe**+**)>

Un chapitre peut ne pas avoir d'auteur.

Alternatives

Structure

```
<!ELEMENT les-amis (ami*)>
<!ELEMENT ami (nom,  (poste | mél | fax))>
<!ELEMENT nom (#PCDATA)>
<!ELEMENT poste (#PCDATA)>  ……..   ]>
```

Document

```
<les-amis>
     <ami> <nom> Jeanne </nom> <poste>Genève </poste>
     </ami>
     <ami><nom>Louis</nom><mél>Lou@ici.net</mél>
     </ami>
</les-amis>
```

Documents auto-décrits

```
<?xml version="1.0"?>
<!-- la structure -->
<!DOCTYPE les-amis [
    <!ELEMENT les-amis …>
    …
]>
<!-- le contenu -->
<les-amis>
    …
</les-amis>
```


Représentation de données

Cette technique est très générale

Elle permet de représenter toutes sortes de structures de données

Documents de tous types (livres, articles, formulaires, …)

Mais aussi des données structurées en tableau, relation, graphes, listes, …

=> utilisation pour l'échange de données à l'intérieur d'un système d'information ou entre systèmes d'information.

Exemple: feuille de calcul

```
<!DOCTYPE feuille [
    <!ELEMENT feuille (ligne*)>
    <!ELEMENT ligne (cellule*)>
    <!ELEMENT cellule (#PCDATA)>
]>
<feuille>
    <ligne>
       <cellule>Date</cellule>
       <cellule>Prix</cellule>
    </ligne>
    <ligne>
       . . .
    <ligne>
</feuille>
```


Retour à la théorie: Grammaires générales

Certains langages ne peuvent être exprimés avec des grammaires hors contexte, p.ex. L = $\{a^n b^n c^n\}$

Une **grammaire générale** est formée

de symboles **terminaux** (T) et **non terminaux** (V),

d'un **symbole initial**

d'une ensemble R de **règles de production** de la forme

$$w1 \rightarrow w2$$

où w1 et w2 sont des chaînes de symboles tirés de V et T

Dérivation

règle P = $\alpha \rightarrow \beta$

u = pαq

application de P sur α dans u : u $\rightarrow_P$ pβq

Si α apparaît plusieurs fois dans la chaîne u, on peut appliquer P sur n'importe quelle occurrence de α, ce qui donne plusieurs dérivations différentes.

Hiérarchie de complexité des langages

Complexe signifie qu'il est difficile de répondre à la question

« la chaîne w appartient-elle au langage ? »

c.f. w est elle un théorème du système formel ?

La complexité du langage dépend de la forme des règles

Hiérarchie 1-2

1. Le plus compliqué: règles générales (α, β chaînes quelconques de terminaux et non terminaux)

règles de la forme : $\alpha \rightarrow \beta$

langage **récursivement énumérable**

on ne peut même pas automatiser l'analyse

2. règles de la forme : $\alpha \rightarrow \beta$

mais avec longueur de $\alpha \leq$ longueur de β

langage **dépendant du contexte**

Hiérarchie 3-4

3. règles de la forme A $\rightarrow \beta$

(A: symbole non terminal)

langage **hors contexte**

analysable par un automate à pile

4.règles de la forme a. A $\rightarrow$ a ou A $\rightarrow$ aB

ou de la forme A $\rightarrow$ a ou A $\rightarrow$ Ba (a terminal)

langage **régulier** (linéaire)

analysable avec un **automate à états** (sans mémoire)