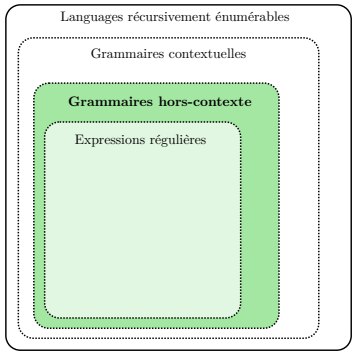
Grammaires hors-contexte. Applications à l'analyse syntaxique.

I. Grammaires hors-contexte [CAR]



Définition 1 $G = (S_0, T, V, P)$ est une grammaire hors-contexte où :

- T (terminaux) et V (variables ou non-terminaux) sont des alphabets, finis et disjoints.
- $S_0 \in V$ est l'axiome, le non-terminal duquel on commence à dériver
- P est un sous-ensemble fini de $V \times (T \cup V)^*$ appelées règles de production. On les note $X \rightsquigarrow w_1 ... w_n$.

Notation 2 Par convention, les mots dont les symboles appartient à :

- T sont denotés par des minuscules (a, b, ...)
- V sont denotés par des majuscules (A, B, ...)
- $T \cup V$ sont denotés par des lettres grecques (α, β, ...)

Définition 3 **Relation de dérivation** On peut dériver $s = \alpha X \beta$ en $s' = \alpha \omega \beta$ si $X \rightsquigarrow \omega$. On note cette dérivation $s \to s'$.

Notation 4 **Clôture** De plus, on note $\to^*$ la clôture réflexive et transitive de $\to$. Si $s_1 \to ... \to s_2$, alors $s_1 \to^* s_2$.

Définition 5 **Mot engendré** Un mot $w \in T^*$ est engendré par la grammaire G si $S \to^* w$.

Définition 6 **Le langage engendré** par une grammaire G à partir du non-terminal X est noté $L_G(X) = \{w \in T^* \mid X \to^* w\}$. Par convention, $L_G = L_G(S_0)$ dénote le langage engendré par G à partir de son axiome.

Exemple 7 **Langage des mots bien parenthésés** La grammaire

$$S \rightsquigarrow ST \qquad S \rightsquigarrow \varepsilon \qquad T \rightsquigarrow (S)$$

reconnaît le langage des mots bien parenthésés, aussi appelé langage de Dyck.

Définition 8 **Dérivations** Soit un mot $s = wX\alpha$ (resp $\alpha X w$) avec $w \in T^*$. Appliquer une règle $X \rightsquigarrow x$ tels que $s \to wx\alpha$ (resp $s \to \alpha x w$) est une dérivation à **gauche** (resp **droite**).

Lemme 9 **Fondamental** Pour $\alpha = \alpha_1 \alpha_2$ et β quelconque,

$$\alpha \overset{k_1+k_2}{\to} \beta \Leftrightarrow \exists \beta_1, \beta_2, k_1, k_2 \text{ tq } \beta = \beta_1 \beta_2 \text{ et } \alpha_1 \overset{k_1}{\to} \beta_1, \alpha_2 \overset{k_2}{\to} \beta_2$$

Cela signifie que l'on peut dériver un mot à gauche ou à droite de manière indépendante et équivalente.

Remarque 10 Les grammaires hors-contexte sont aussi appelées « algébriques », car on peut montrer que les langages engendrés par les grammaires hors-contextes sont des solutions de systèmes d'équations polynomiales.

A. Transformations

Définition 11 **Forme normale de Chomsky** Une grammaire $G = (S_0, T, V, P)$ est en **forme normale de Chomsky** si toutes ses règles dont d'une des formes suivantes :

$$A \rightsquigarrow BC \quad \text{où } B, C \in V$$
$$A \rightsquigarrow a \quad \text{où } a \in T$$

Remarque 12 La forme normale de Chomsky est également appelé **forme normale quadratique**

B. Arbre de dérivation [TIGER]

Définition 13 **Arbre de dérivation** L'application d'une règle de dérivation $X \to \alpha_1 \alpha_2 ... \alpha_n$ peut-être vu comme un **arbre** ayant pour racine X et pour sous-arbres des arbres de dérivation des α_i produits.

On peut ainsi représenter les applications successives de règles de dérivation par un **arbre de dérivation** ayant pour racine S.

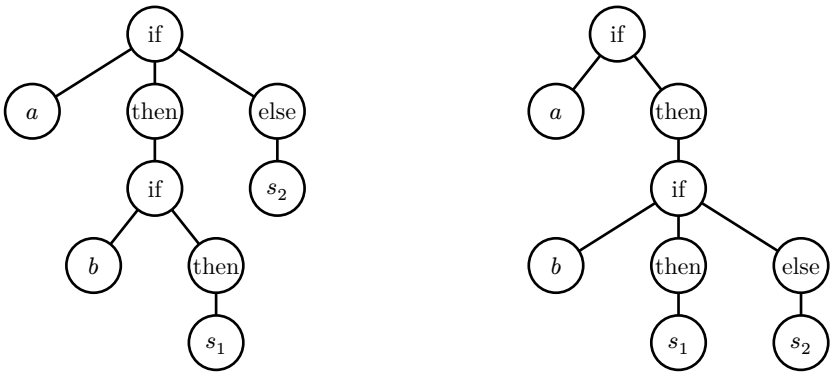
Définition 14 **Ambiguïté d'une grammaire** [TIGER] Une grammaire G est ambiguë lorsqu'il existe au moins un mot appartenant

au langage engendré L_G pour lequel il existe deux arbres de dérivation distincts.

Exemple 15 **Sinon en suspens** Avec la grammaire

$$S \rightsquigarrow \text{if } E \text{ then } S \text{ else } S \qquad S \rightsquigarrow \text{if } E \text{ then } S$$
$$S \rightsquigarrow s_1 \mid s_2 \qquad E \rightsquigarrow a \mid b$$

le mot « if a then if b then s_1 else s_2 » possède deux arbres de dérivation :



Définition 16 **Langage inhéremment ambigu** On parle bien de **grammaire** ambiguë et non de langage. Il est possible de construire deux grammaire distinctes ayant le même langage engendré, avec l'une ambiguë et l'autre non.

Cependant, il existe de langages **inhéremment ambiguës**, pour lesquels toute grammaire engendrant ce langage est ambiguë.

Exemple 17 La grammaire

$$S \rightsquigarrow M \qquad S \rightsquigarrow U$$
$$M \rightsquigarrow \text{if } E \text{ then } M \text{ else } M \qquad M \rightsquigarrow s_1 \mid s_2$$
$$U \rightsquigarrow \text{if } E \text{ then } S \qquad E \rightsquigarrow a \mid b$$

reconnaît le même langage que précédemment, mais n'est pas ambiguë.

Exemple 18 Le langage $L = \{a^m b^m c^n \mid m, n \geq 0\} \cup \{a^m b^n c^n \mid m, n \geq 0\}$ est inhéremment ambigu.

Théorème 19 **Bar-Hillel, Perles et Shamir** Pour tout langage algébrique L, il existe $N > 0$ tel que pour tout mot $f \in L$, si $|f| \geq N$ alors on peut trouver une factorisation $f = \alpha u \beta v \gamma$ telle que $|uv > 0, |u\beta v| < N$ et $\alpha u^n \beta v^n \gamma \in L$ pour tout $n \geq 0$.

Remarque 20 Ce théorème est analogue au lemme de l'étoile dans le contexte des grammaires hors-contexte.

C. Automates à pile

Ouverture 21 **Automate à pile** On peut étendre la définition d'un automate pour y ajouter une pile : les transitions dépendent alors de l'état, de la lettre lue, et du sommet de la pile. On peut également choisir d'empiler ou de dépiler des éléments.

Propriété 22 Un langage $L \subseteq A^*$ est algébrique si et seulement s'il existe un automate à pile qui accepte L.

II. Analyse syntaxique [TIGER]

DEV

Algorithme 23 **Cocke-Younger-Kasami (CYK)** L'algorithme CYK est un algorithme d'analyse syntaxique sur les grammaires sous forme normale de Chomsky utilisant la programmation dynamique. Sa complexité en $O(|w|^3 + |P|)$ et sa flexibilité le rendent utile pour certaines tâches.

Remarque 24 En pratique, l'algorithme CYK est trop lent pour être utilisable dans un compilateur. On préférera des algorithmes moins généraux, mais traitant un sous-ensemble choisi de languages en un temps de traitement linéaire.

A. Analyse syntaxique LL(1)

Définition 25 **Ensembles premiers, suivants et annulable** Soit une grammaire $G = (S_0, T, V, P)$, on définit :

- Annulable(X), pour $X \in V$ est vrai si $X \to^* \varepsilon$
- Premiers(γ), pour $\gamma \in (T \times V)^*$ est l'ensemble des terminaux pouvant commencer les mots dérivants de γ
- Suivants(X), pour $X \in V$ est l'ensemble des terminaux pouvant immédiatement suivre X, c'est à dire $t \in \text{Suivants}(X)$ s'il existe une dérivation de la forme $Y \xrightarrow{*} \omega X t$, avec $Y \in V$ et $\omega \in (T \times V)^*$.

DEV

Algorithme 26 Calcul des ensembles premiers, suivants et annulables. On peut calculer ces ensembles à l'aide d'un algorithme de recherche de point fixe. On initialise premiers et suivants aux ensembles vides, et annulables à tout faux.

```
pour tout terminal Z
  premiers[Z] ← {Z}
répéter
  pour chaque règle X → Y1Y2 · · · Y k
    si Y1 . . . Y k are tous annulable (ou si k = 0)
      alors annulable[X] ← vrai
    pour i de 1 à k, ji de i + 1 à k
      si Y1 · · · Y i-1 sont tous annulable (ou si i = 1)
        alors premiers[X] ← premiers[X] ∪ premiers[Y i ]
      si Y i+1 · · · Y k sont tous annulable (ou si i = k)
        alors suivants[Y i ] ← suivants[Y i ] ∪ suivants[X]
      si Y i+1 · · · Y j -1 sont tous annulable (ou si i + 1 = j )
        alors suivants[Y i ] ← suivants[Y i ] ∪ premiers[Y j ]
tant que premiers, suivants, et annulables n'ont pas changé
```

Définition 27 Méthode LL(1) La méthode d'analyse syntaxique LL(1) (pour **L**eft to right, **L**eftmost derivation, 1, de gauche à droite, dérivation gauche, en regardant 1 token) consiste à déterminer quelle règle de la grammaire appliquer en lisant 1 token, de gauche à droite.

On dit alors qu'un langage est LL(1) si il est reconnu par un analyseur LL(1).

Propriété 28 Grammaire LL(1) [COMPI] Une grammaire G est dite LL(1) si et seulement si, pour tout non-terminal N, avec les règles de productions $N \rightsquigarrow \alpha_1, ..., N \rightsquigarrow \alpha_n$

- Les ensembles Premiers(α_1), ..., Premiers(α_n) sont disjoints deux à deux
- Si de plus on a Annulable(N), alors Suivants(N) est disjoint de chaque Premiers(α_i)

Exemple 29 La grammaire

$$S \rightsquigarrow +SS \qquad S \rightsquigarrow T \qquad T \rightsquigarrow \text{id} \qquad T \rightsquigarrow \text{num}$$

est LL1. On peut construire la table suivante :

	$+$	id	num
S	$S \rightsquigarrow +SS$	$S \rightsquigarrow T$	$S \rightsquigarrow T$
T		$T \rightsquigarrow$ id	$T \rightsquigarrow$ num

qui nous donne, pour chaque état, pour chaque token lu, quelle règle doit être appliquée.

Définition 30 Récursivité à gauche Une grammaire est récursive à gauche si elle contient une règle de production de la forme : $A \rightsquigarrow A\omega$.

Proposition 31 Certains langages ne sont pas analysables par un algorithme LL(1). C'est notamment le cas des grammaires récursives à gauche.

DEV

Transformation 32 Élimination de la récursion à gauche [TIGER 3.2] Certains langages sont LL(k), mais leur grammaire peut-être donnée récursive à gauche. On peut transformer ces grammaires de sorte à pourvoir les donner en entrée d'un analyseur LL(k).

Proposition 33 Complexité de l'analyse LL(1) L'algorithme est de complexité linéaire en la taille du mot d'entrée.

Exemple 34 LR(k) Il existe d'autres méthodes d'analyse syntaxique, par exemple en faisant des dérivations droite au lieu de dérivations gauche, donnant les algorithmes LR(k).

Pratique 35 Générateur d'analyseur syntaxique En pratique, on utilise des programmes comme `Yacc`, `Bison`, `Menhir` ou `antlr` pour générer automatiques un analyseur syntaxique à partir d'une grammaire.

Pratique 36 Résolution d'ambiguïté Certaines ambiguïtés inhérentes peuvent se résoudre avec des règles ad-hoc.

Exemple 37 Problème du sinon en suspens Les grammaires acceptant à la fois les formes `if () then ()` et `if () then () else ()` sont souvent ambigus, mais on peut résoudre ce problème en choisissant de toujours lier un `else` au `if` le plus proche.

Grammaires hors-contexte. Applications à l'analyse syntaxique.

I. Grammaires hors-contexte [CAR]

1 Def

2 Not

3 Def Relation de dérivation

4 Not Clôture

5 Def Mot engendré

6 Def Le langage engendré

7 Ex Langage des mots bien parenthésés

8 Def Dérivations

9 Lemme Fondamental

10 Rem

A. Transformations

11 Def Forme normale de Chomsky

12 Rem

B. Arbre de dérivation [TIGER]

13 Def Arbre de dérivation

14 Def Ambiguïté d'une grammaire [TIGER]

15 Ex Sinon en suspens

16 Def Langage inhéremment ambigu

17 Ex

18 Ex

19 Thm Bar-Hillel, Perles et Shamir

20 Rem

C. Automates à pile

21 Ouverture Automate à pile

22 Prop

II. Analyse syntaxique [TIGER]

23 Algo Cocke-Younger-Kasami (CYK)

24 Rem

A. Analyse syntaxique LL(1)

25 Def Ensembles premiers, suivants et annulable

26 Algo Calcul des ensembles premiers, suivants et annulables.

27 Def Méthode LL(1)

28 Prop Grammaire LL(1) [COMPI]

29 Ex

30 Def Récursivité à gauche

31 Prop

32 Transformation Élimination de la récursion à gauche [TIGER 3.2]

33 Prop Complexité de l'analyse LL(1)

34 Ex LR(k)

35 Prat Générateur d'analyseur syntaxique

36 Prat Résolution d'ambiguïté

37 Ex Problème du sinon en suspens

Remarque

Bibliographie

[CAR] O. Carton & G. G. Gagnees, *Langages formels: Calculabilité et complexité.*

[TIGER] A. Appel, *Modern compiler implementation.*

[COMPI] F. Schwarzentruber & R. Legendre, *Compilation : analyse lexicale et syntaxique.*

Benjamin Voisin & Aloïs Rautureau