Notes de cours

GIF 340 : ÉLÉMENTS DE COMPILATION

Chapitre 3

Grammaires et langages

Ahmed KHOUMSI

Quelques définitions

Grammaire est un moyen pour spécifier la syntaxe d 'un langage, c-à-d.
l 'ensemble des phrases acceptées par le langage
exemple : langage naturel, langage de programmation

Langage de programmation est un moyen pour communiquer avec l 'ordinateur
exemple : pascal, C, Ada, Java

Spécification d 'une grammaire
une grammaire G d 'un langage est un quadruplet (Vt, Vn, S, P) où :
Vt = Vocabulaire de terminaux
Vn = Vocabulaire de non terminaux
S = Symbole de départ
P = Règles de production

Spécification d 'une grammaire (suite)

Vt est un ensemble fini de symboles terminaux, appelé alphabet terminal
Un symbole terminal est une unité lexicale, c-à-d. un mot qui peut faire partie d 'une phrase acceptée par le langage

Vn est un ensemble fini de symboles non-terminaux
Symbole non-terminal peut être : - un morceau de phrase
- une phrase complète

S est le symbole initial :
il est non-terminal et sert comme symbole de départ

P est un ensemble fini de règles de production (ou de réécriture)
de la forme $\alpha \Longrightarrow \beta$ qui signifient : α peut être remplacé par β
on dit aussi que : β peut être dérivé à partir de α
- α et β sont des séquences de symboles terminaux ou non-terminaux
- α contient au moins un symbole non-terminal

Quelques notations

L 'ensemble de tous les symboles est noté V, c-à-d. $V = Vt \cup Vn$

ε représente la chaîne vide

Si E est un ensemble, alors
- E+ est l 'ensemble des chaînes constituée par des éléments de E.
- E* contient ε et tous les éléments de E+

exemples :
- V+ constitué des chaînes de symboles terminaux et non-terminaux
- V* contient ε et tous les éléments de V+
- Vt+ constitué des chaînes de symboles terminaux

On peut alors définir P plus formellement par l 'ensemble des règles de la forme
$\alpha \Longrightarrow \beta$ telles que :
α appartient à V+-Vt+ (qui est aussi égal à V*-Vt*)
β appartient V*

Définition de langage par une grammaire

Une phrase ϕ est une chaîne de symboles, c-à-d. ϕ appartient à V^* où $V = Vt \cup Vn$

Un langage L est un ensemble de phrases constituées uniquement de symboles terminaux, et qui peut aussi contenir la chaîne vide ε, c-à-d. L est inclus dans Vt^*

Une grammaire G permet de définir un langage, noté L(G), de la manière suivante :
- on part du symbole (non-terminal) initial S
- on effectue des dérivations en appliquant des règles de productions de P jusqu 'à ce qu 'on obtienne une phrase constituée uniquement de symboles terminaux
- l 'ensemble des phrases (terminales) que l 'on peut ainsi obtenir constitue le langage L(G)

Premier exemple : définition de langage par une grammaire

Vt = {il, elle, mange, est, grand, vite}
Vn = {pronom, verbe, adjectif, adverbe, phrase}
S = phrase
P = {

phrase	⟹	pronom verbe adjectif	*(règle 1)*
phrase	⟹	pronom verbe adverbe	*(règle 2)*
pronom	⟹	il	*(règle 3)*
pronom	⟹	elle	*(règle 4)*
verbe	⟹	mange	*(règle 5)*
verbe	⟹	est	*(règle 6)*
adjectif	⟹	grand	*(règle 7)*
adverbe	⟹	vite }	*(règle 8)*

Cette grammaire permet de générer un certain nombre de phrases, à partir de S=phrase

Premier exemple : définition de langage par une grammaire (suite)

Initialement, seules les règles 1 et 2 peuvent être appliquée (car S = phrase)
on obtient alors les phrases :

pronom verbe adjectif (avec la règle 1)
pronom verbe adverbe (avec la règle 2)

Ensuite, on peut effectuer des dérivations en appliquant :

- Les règles 3 et 4 pour remplacer le symbole non-terminal pronom
- Les règles 5 et 6 pour remplacer le symbole non-terminal verbe
- La règle 7 pour remplacer le symbole non-terminal adjectif
  cette règle ne pourra être appliquée que si la règle 1 a été appliquée initialement
- La règle 8 pour remplacer le symbole non-terminal adverbe
  cette règle ne pourra être appliquée que si la règle 2 a été appliquée initialement

Premier exemple : définition de langage par une grammaire (suite)

Les phrases terminales (c-à-d. appartenant à Vt*) qu ’on peut construire sont :

il mange grand
il mange vite
il est grand
il est vite
elle mange grand
elle mange vite
elle est grand
elle est vite

Les 8 phrases constituent le langage reconnu (accepté) par la grammaire

Seules les seconde, troisième et sixième phrases sont sémantiquement correctes
(c-à-d. elles ont un sens)
les autres phrases respectent bien la syntaxe, mais elles n ’ont pas de sens

Lors de la réalisation d ’un compilateur, c ’est la phase d ’analyse sémantique
qui permettra de détecter des constructions grammaticalement (syntaxiquement)
correctes mais qui n ’ont aucun sens

Second exemple : Définition d'expression arithmétique par une grammaire

Vt = { id , + , - , * , / , (,) }
Vn = {E, T, F}
S = E
P = {

E	⟶	T		*(règle 1)*
E	⟶	T+T		*(règle 2)*
E	⟶	T-T		*(règle 3)*
T	⟶	F		*(règle 4)*
T	⟶	F*F		*(règle 5)*
T	⟶	F/F		*(règle 6)*
F	⟶	(E)		*(règle 7)*
F	⟶	id	}	*(règle 8)*

Intuitivement, E, T et F désignent, respectivement, une expression, un terme et un facteur.

Le langage reconnu par cette grammaire est constitué de toutes les phrases terminales qu 'on peut construire : en partant de E et en effectuant des dérivations en appliquant les règles de production de P.

Vérification si une phrase est correcte

Le problème souvent rencontré est de vérifier si une phrase terminale donnée p est correcte (c-à-d. respecte la syntaxe définie par la grammaire)

Lorsque le langage est fini (exemple page 6), la vérification peut être effectuée en comparant p avec tous les éléments du langage

Lorsque le langage est infini (exemple page 9), on peut vérifier si on peut générer p en :
- partant du symbole (non-terminal) de départ (E pour l ' exemple page 9)
- en effectuant des dérivations à l 'aide des règles de production de P

Vérification si une phrase est correcte : exemple

Considérons l 'exemple page 9 et la phrase : id - (id + id)

Cette phrase est correcte car elle peut être obtenue en appliquant les règles représentées dans le tableau ci-dessous

Chaîne courante	règle appliquée
E	E → T - T
T - T	T → F
F - T	F → id
id - T	T → F
id - F	F → (E)
id - (E)	E → T+T
id - (T + T)	T → F
id - (F + T)	T → F
id - (F + F)	F → id
id - (id + F)	F → id
id - (id + id)	

Sur colonne de gauche : chaîne de symboles courante
Sur colonne de droite : règle appliquée

Vérification si une phrase est correcte : exemple (suite)

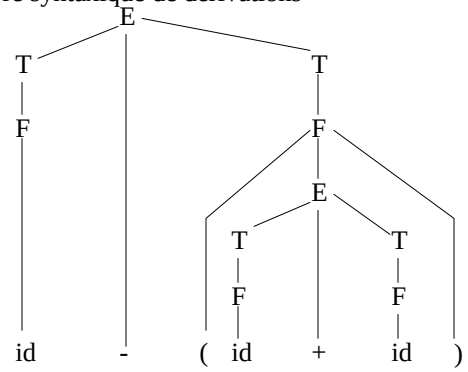
La séquence des dérivations peut aussi être représentée des 2 manières suivantes :

Manière 1.

E $\xrightarrow{3}$ T - T $\xrightarrow{4}$ F - T $\xrightarrow{8}$ id - T $\xrightarrow{4}$ id - F $\xrightarrow{7}$ id - (E) $\xrightarrow{2}$ id - (T + T)
$\xrightarrow{4}$ id - (F + T) $\xrightarrow{4}$ id - (F + F) $\xrightarrow{8}$ id - (id + F) $\xrightarrow{8}$ id - (id + id)

où les numéros représentent les règles appliquées

Manière 2. Arbre syntaxique de dérivations



Méthodes de dérivation

Il y a plusieurs manières de dériver une phrase à partir du symbole de départ. Les deux manières les plus utilisées sont les suivantes

Dérivation à gauche :

À chaque étape, on applique une règle sur le symbole non-terminal le plus à gauche. Pour l 'exemple page 9, on obtient :

$E \xrightarrow{3} T - T \xrightarrow{4} F - T \xrightarrow{8} id - T \xrightarrow{4} id - F \xrightarrow{7} id - (E) \xrightarrow{2} id - (T + T)$
$\xrightarrow{4} id - (F + T) \xrightarrow{8} id - (id + T) \xrightarrow{4} id - (id + F) \xrightarrow{8} id - (id + id)$

Dérivation à droite :

À chaque étape, on applique une règle sur le symbole non-terminal le plus à droite. Pour l 'exemple page 9, on obtient :

$E \Rightarrow T - T \Rightarrow T - F \Rightarrow T - (E) \Rightarrow T - (T + T) \Rightarrow T - (T + F)$
$\Rightarrow T - (T + id) \Rightarrow T - (F + id) \Rightarrow T - (id + id) \Rightarrow F - (id + id)$
$\Rightarrow id - (id + id)$

Typologie des grammaires

On distingue 4 types de grammaires

Type 0 : il n 'y a aucune restriction sur les règles de production

Type 1 : **grammaires sensibles au contexte**

Dans une règle $\alpha \Rightarrow \beta$, la partie droite doit contenir au moins autant de symboles que la partie gauche. Formellement : $|\alpha| \leq |\beta|$

Ces grammaires sont dites sensibles au contexte car elles permettent d 'avoir des règles du genre : $aAb \Rightarrow a\beta b$ où :

a, b sont des terminaux, A est un non-terminal, et β est une chaîne (non vide) dont les éléments peuvent être terminaux ou non-terminaux

Avec une telle règle, A peut être remplacé par β seulement dans le contexte où il est entouré de a et b

La règle $S \Rightarrow \varepsilon$ (où S est le symbole de départ) est permise si aucune des autres règles ne contient S sur sa partie droite

Typologie des grammaires : Type 2

Grammaires hors-contexte (non contextuelles)

Les règles sont de la forme $A \Longrightarrow \beta$ où :

A est un non-terminal et β est une chaîne (possiblement vide) dont les éléments peuvent être terminaux ou non-terminaux

Avantage :

Les grammaires non contextuelles permettent de développer des analyseurs syntaxiques performants

Inconvénient :

Langages de programmation ne peuvent pas être entièrement décrits par ce type de grammaire, car ils possèdent des parties sensibles au contexte, telles qu :

- règles de compatibilité des types
- correspondance entre les paramètres formels et effectifs d 'une procédure
- une variable ne peut pas être déclarée plus d 'une fois dans une même portée

…

Grammaires hors-contexte (suite)

Pour résoudre l 'inconvénient :

- on décrit formellement un langage par une grammaire non contextuelle
- les parties sensibles au contexte sont décrites d 'une manière informelle

Analyse syntaxique :

- basée uniquement sur la spécification formelle
- ne permet donc pas de déterminer avec certitude si un programme est correct

Exemple : on considère l 'exemple page 6

Sur les 8 phrases *syntaxiquement* correctes, seules 3 sont *vraiment* correctes

Par exemple, pour interdire (elle est grand) il faudrait une règle telle que :

il verbe adjectif $\Longrightarrow$ il verbe grand

Avec cette règle, le symbole non-terminal adjectif n 'est remplacé par le terminal grand que si la phrase commence par le terminal il

Analyse sémantique : permet de vérifier les parties sensibles au contexte

Typologie des grammaires : Type 3 : Grammaires régulières

Les règles sont de la forme $A \Rightarrow wB$ ou $A \Rightarrow w$ où :
- A est un non-terminal
- w est une chaîne de terminaux
- B est un non-terminal

Un langage généré par une grammaire régulière est appelé langage régulier et peut être décrit par une expression régulière

Une grammaire régulière peut donc être représentée par un automate à états fini (AEF)

Grammaires régulières : exemple

Ensemble des identificateurs : - commençant par une lettre et
- constitués d 'une séquence de chiffres et de lettres

peut être décrit par la grammaire suivante

Vt = {A, B, …, Z, a, b, …, z, 0, 1, …, 9}
Vn = {lettre, chiffre, id}
S = id
P = {
lettre ⇒ (A | B | … | Z | a | b | … | z)
chiffre ⇒ (0 | 1 | … | 9)
id ⇒ lettre(lettre|chiffre)*
}

La grammaire n 'est pas régulière (à cause de la dernière règle)

Grammaires régulières : exemple (suite)

La grammaire précédente est équivalente à la suivante :

Vt = {A, B, …, Z, a, b, …, z, 0, 1, …, 9}
Vn = {chaîne, id}
S = id
P = { chaîne ⟹ (A | B | … | Z | a | b | … | z | 0 | 1 | … | 9)
chaîne ⟹ (A chaîne | B chaîne | … | Z chaîne)
chaîne ⟹ (a chaîne | b chaîne | … | z chaîne)
chaîne ⟹ (0 chaîne | 1 chaîne | … | 9 chaîne)
id ⟹ (A | B | … | Z | a | b | … | z | 0 | 1 | … | 9)
id ⟹ (A chaîne | B chaîne | … | Z chaîne)
id ⟹ (a chaîne | b chaîne | … | z chaîne) }

La grammaire obtenue est bien régulière puisque les parties droites des règles sont de la forme w ou wB où : w est un non-terminal et B est un terminal

On obtient une autre grammaire équivalente mais qui n 'est pas régulière en prenant :

P = {chaîne ⟹ ε
chaîne ⟹ (A | B | … | Z | a | a | b | … | z | 0 | 1 | … | 9) chaîne
id ⟹ (A | B | … | Z | a | a | b | … | z) chaîne }

Grammaires ambiguës

Une grammaire G est ambiguë s 'il existe une phrase du langage L(G) pour laquelle il y a plusieurs arbres syntaxiques de dérivation

Les techniques pour réaliser des analyseurs syntaxiques de langages hors-contexte supposent que les grammaires à la base de ces langages ne sont pas ambiguës

Exemple d 'ambiguïté :

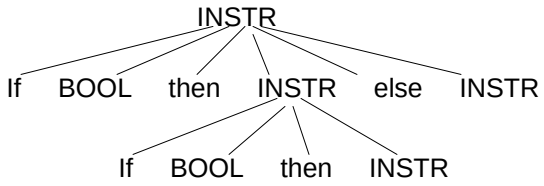
Vt = {if, then, else}
Vn = {BOOL, INSTR}
S = INSTR
P = {
INSTR ⟹ if BOOL then INSTR else INSTR
INSTR ⟹ if BOOL then INSTR
}

Nous considérons la phrase :

if BOOL then if BOOL then INSTR else INSTR

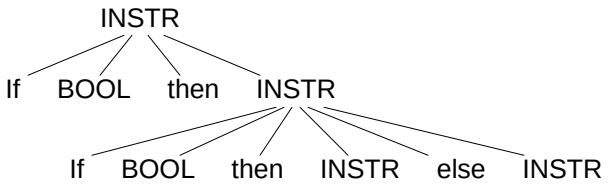
Grammaires ambiguës (suite)

Premier arbre syntaxique de dérivation :



Intuitivement, cela revient à considérer que le else se rapporte au premier if

Deuxième arbre syntaxique de dérivation :



Intuitivement, cela revient à considérer que le else se rapporte au second if

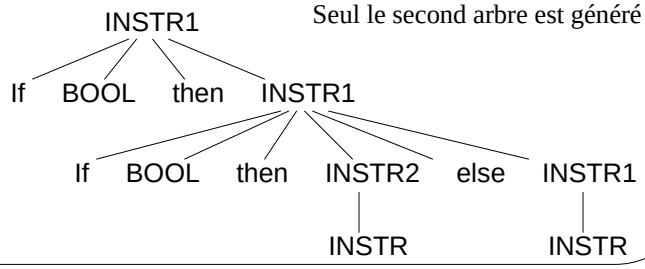
Grammaires ambiguës (suite)

L 'ambiguïté peut être supprimée en modifiant les règles de production

Pour notre exemple, la grammaire suivante suivante n 'est pas ambiguë

Vt = {if, then, else}
Vn = {BOOL, INSTR1, INSTR2, INSTR}
S = INSTR1
P = { INSTR1 ⟹ if BOOL then INSTR1
INSTR1 ⟹ if BOOL then INSTR2 else INSTR1
INSTR1 ⟹ INSTR
INSTR2 ⟹ if BOOL then INSTR2 else INSTR2
INSTR2 ⟹ INSTR }

Seul le second arbre est généré



Comparaison des quatre types de grammaires

Une grammaire de type i est une grammaire de type i-1, pour i = 1, 2 et 3

Grammaires régulières :

- sont donc les plus contraignantes
  exemple : impossible de spécifier imbrication illimitée d 'expressions
- ont néanmoins plusieurs applications, telles que :
  - modélisations de systèmes à événements discrets
  - analyse lexicales

Grammaires hors-contexte :

- moins contraignantes que grammaires régulières
- plus contraignantes que grammaires sensibles au contexte
- ont plusieurs applications : celle qui nous intéresse ici est l 'analyse syntaxique d 'un langage de programmation