Chapitre 03 : Analyse syntaxique

1 Introduction

Tout langage de programmation possède des règles qui indiquent la structure syntaxique d'un programme bien formé. Par exemple, en Pascal, un programme bien formé est composé de blocs, un bloc est formé d'instructions, une instruction de ... . La **syntaxe** d'un langage peut être décrite par une **grammaire**.

L'analyseur syntaxique reçoit une suite d'unités lexicales de la part de l'analyseur lexical et doit vérifier que cette suite peut être engendrée par la grammaire du langage. Le problème est donc :

- Etant donnée une grammaire G.
- Etant donnée un mot *m (un programme)*

➔Est-ce que *m* appartient au langage généré par *G* ?

Le principe est d'essayer de construire un arbre de dérivation. Il existe deux méthodes pour cette construction : méthode (analyse) descendante et méthode (analyse) ascendante.

2 Grammaires hors contexte

Pour décrire la syntaxe d'un langage de programmation, on utilise une grammaire. Une grammaire est un ensemble de règles décrivant comment former des phrases.

Définition 1 Une **grammaire** est la donnée de $G=(V_T,V_N,S,P)$ où

- V_T est un ensemble non vide de symboles ***terminaux*** (alphabet terminal)
- V_N est un ensemble de symboles ***non-terminaux***, avec $\mathbf{V_T} \cap \mathbf{V_N}$
- S est un symbole initial $\in \mathbf{V_N}$ appelé ***axiome***
- P est un ensemble de ***règle de productions***.

Définition 2

Une ***règle de production*** $\alpha \rightarrow \beta$ précise que la séquence de symboles α $(\alpha \in (\mathbf{V_T} \mathbf{U} \mathbf{V_N})^+)$ peut être remplacée par la séquence de symboles β $(\beta \in (\mathbf{V_T}\ \mathbf{U}\ \mathbf{V_N})^*)$. α est appelée ***partie gauche*** de la production, et β ***partie droite*** de la production.

Exemple

Symboles terminaux (alphabet) : $\mathbf{V_T=\{a,b\}}$
Symboles non-terminaux : $\mathbf{V_N=\{S\}}$
Axiome : *S*
Règles de production $\begin{cases} \mathbf{S} \rightarrow \varepsilon \\ \mathbf{S} \rightarrow \mathbf{aSb} \end{cases}$ qui se résument en $\mathbf{S} \rightarrow \varepsilon \mid \mathbf{aSb}$

Considérons la grammaire suivante :
Expr → Expr Op nb| nb
Op → +|-|/|*

3 Arbre de dérivation

On appelle ***arbre de dérivation*** (ou arbre syntaxique) tout arbre tel que

- la racine est l'axiome
- les feuilles sont des unités lexicales ou ε
- les noeuds sont des symboles non-terminaux

- les fils d'un nœud α sont $\beta_0 \ldots \beta_n$ si et seulement si $\alpha \rightarrow \beta_0 \ldots \beta_n$ est une production (Le parcours préfixe de l'arbre donne le mot)

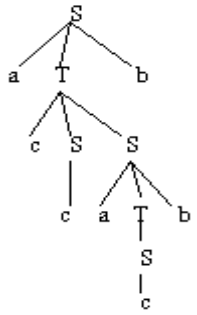
Exemple

Soit la grammaire ayant *S* pour axiome et pour règles de production P:

S→aTb|c

T→cSS|S

Un arbre de dérivation pour le mot ***accacbb*** est :



$S \rightarrow aTb \rightarrow acSSb \rightarrow accSb \rightarrow accaTbb \rightarrow accaSbb \rightarrow accacbb$ (dérivations **gauches**)
ou $S \rightarrow aTb \rightarrow acSSb \rightarrow acSaTbb \rightarrow acSaSbb \rightarrow acSacbb \rightarrow accacbb$ (dérivations **droites**)
Ces deux suites **différentes** de dérivations donnent le **même** arbre de dérivation.

4 Grammaires *ambiguës*

Une grammaire est dite ***ambiguë*** s'il existe un mot de $L(G)$ ayant plusieurs arbres syntaxiques.
Exemple de grammaire ambiguë : Soit *G* donnée par:
instr→ if (expr) instr else instr
inst→ if (expr) instr if (expr) instr
inst→ …
expr→…
Cette grammaire est ambiguë car le mot ***m*= if (*x*>10) if (*y*<0) *a*=1 else *a*=0** possède deux arbres syntaxiques différents :

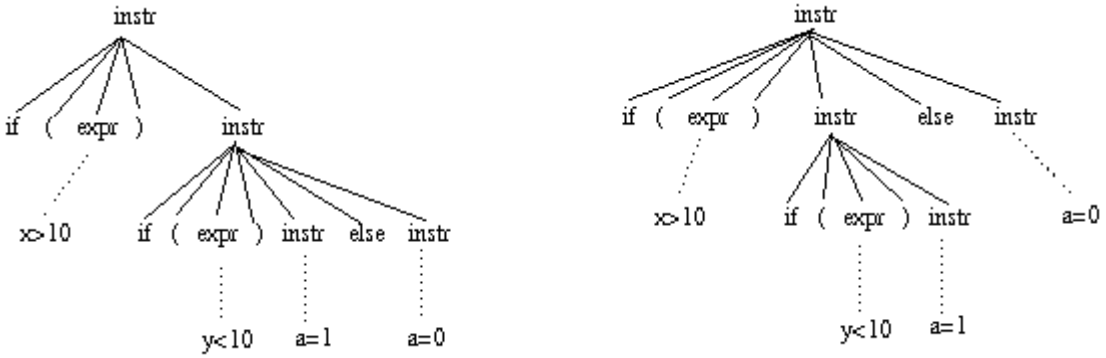


Figure 3.1 : deux arbres différents pour la même chaine.

Car il y a deux interprétations syntaxiques possibles :

```
if (x>10)                  if (x>10)
   if (y<0)                     if (y<0)
      a=1                          a=1
   else                         // finsi
```

```
            a=0                 else
          // finsi                   a=0
        // finsi                 // finsi
```

Solution proposée

stmt → matched-stmt
| open-stmt
matched-stmt→ if expr then matched-stmt else matched-stmt
| other
open-stmt → if expr then stmt
| if expr then matched-stmt else open-stmt

5 Mise en œuvre d'un analyseur syntaxique

Il existe deux approches (deux méthodes) pour construire cet arbre de dérivation : une méthode descendante et une méthode ascendante.

Chapitre 04: Analyse syntaxique descendante