Compilation

Analyse lexicale et (surtout) syntaxique

Jules Chouquet

SOM2IF15 – 2024

Où on en est :

On sait d'où l'on part : un programme P écrit dans un langage de haut niveau.
On sait où l'on va : traduction vers du code MIPS à terme. Dans un premier temps, vers des arbres syntaxiques.

Objet de ce cours

Un programme qui doit analyser P reçoit dans un premier temps une suite de caractères, et non des données structurées.
L'analyse lexicale va transformer la suite de caractères en une suite de "mots", ou lexèmes (ou *tokens*).
L'analyse syntaxique va structurer les lexèmes en une arborescence représentant la structure du programme lu.

Exemple

Donnée en entrée de l'analyse lexicale :

[v] [o] [i] [d] [␣] [f] [P] [l] [u] [s] [(] [)] [{] [\n] [p] [r] [i] [n] [t] [(] [x] [+] [1] [)] [;] [\n] [}]

En sortie :

[void] [fPlus] [(] [)] [{] [print] [(] [x] [+] [1] [)] [;] [}]

Analyse lexicale

On considère que tous les mots du langage sont définissables par des expressions régulières.
Les expressions régulières sont reconnaissables par des automates déterministes.

Pour les détails

Cours sur les langages et les automates. (ou me contacter pour des références de livres)

Analyse syntaxique : Parlons grammaires

Nous décrivons tous les programmes possibles d'un langage à travers une grammaire hors contexte et des règles de génération.

Définition : Grammaire hors contexte

C'est un quadruplet (NT, T, R, S) :

- NT est un ensemble *fini* de symboles **non terminaux**
- T est un ensemble *fini*[a] de symboles **terminaux**
- R est un ensemble de règles de la forme $V \rightarrow w$ où
  - V est un symbole non terminal
  - $w \in \{T, NT\}^*$
- S est le symbole initial.

a. L'ensemble des entiers N n'est pas correct par exemple.

Les **terminaux** correspondront aux **lexèmes** identifiés par l'analyse lexicale.

Un exemple

$G = (\{X, B, C, N, A, E, I, P\}, \{a, \ldots, z, 0, \ldots, 9, +, -, =, ;\}, R, P)$
Où R est l'ensemble de règles suivant :

1. $C \to 0$
⋮
9. $C \to 9$

10. $N \to C$
11. $N \to NC$

12. $X \to a$
⋮
38. $X \to z$

39. $B \to \texttt{true}$
40. $B \to \texttt{false}$

41. $A \to X$
42. $A \to N$
43. $A \to B$

44. $E \to A$
45. $E \to E + E$
46. $E \to E - E$
47. $E == E$

48. $I \to X = E$
49. $I \to \texttt{if}\ \ E\ \ \texttt{then}\ \ I$

50. $P \to \epsilon$
51. $P \to I; P$

Dérivation et génération de langage

Définition : Dérivation

Soient $G = (NT, T, R, S)$ une grammaire et u, u' deux chaînes de caractères sur $T \cup NT$.

- u' est **immédiatement dérivé** de u si u' est obtenu en remplaçant dans u un symbole non terminal V par une chaîne w avec une règle $V \to W$ de R. On écrit $u \Rightarrow u'$.
- u' est **dérivé** de u s'il existe une suite finie de dérivations immédiates de u vers u'. On écrit $u \Rightarrow^* u'$.

Définition : langage généré

Le langage généré par la grammaire $G = (NT, T, R, S)$ est l'ensemble

$$\mathcal{L}(G) = \{w \in T^* \mid S \Rightarrow^* w\}$$

il contient tous les mots qui peuvent être dérivés depuis le symbole initial.

Exemple de dérivation et exercice

[au tableau]

Structuration arborescente du langage

Définition : Arbre syntaxique

Soit $G = (NT, T, R, S)$ une grammaire.

Un **arbre syntaxique** (*parse tree*) est un arbre (ordonné) dans lequel :

- Les nœuds sont indexés par des symboles de $NT \cup T \cup \{\epsilon\}$
  - Chaque nœud interne ($\neq$ feuille) est un non terminal.
  - Chaque feuille est un terminal ou ϵ
- La racine est indexée par S
- Si un nœud est indexé par $A \in NT$, et ses enfants sont indexés par $X_1, \ldots, X_k$, alors $A \to X_1 \ldots X_k$ est une règle de R.

[exemples d'arbres au tableau]

On dira qu'une chaîne s est reconnue par un arbre syntaxique A si elle est le résultat de la lecture de gauche à droite des feuilles de A.

Une grammaire sera dite **ambiguë** s'il existe une chaîne dans $\mathcal{L}(G)$ qui admet plus d'un arbre syntaxique.

Arbre de syntaxe abstrait

C'est une simplification de l'arbre de syntaxe qui enlève des informations superflues (parenthèses, lexèmes sans contenu) et remplace les non terminaux par le nom de la règle appliquée.

[figures]

Une autre syntaxe pour les grammaires

Forme de Backus-Naur (BNF)

- $\rightarrow$ devient $::=$
- Les non terminaux sont écrits entre chevrons
- $V \rightarrow w1, \ldots, V \rightarrow wn$ sont rassemblées et notées $V ::= w1 \mid \ldots \mid wn$

Il existe des variantes ou des extensions, notamment avec des expressions régulières. Par exemple $\langle P \rangle ::= \{\langle I \rangle ; \}^*$ (un programme est une suite — éventuellement vide) — d'instructions).

Si l'on reprend la grammaire G donnée plus haut, on obtient la présentation suivante :

<digit> → 0|1|2|3|4|5|6|7|8|9

<char> → a|...|z

<nat> → <digit> | <digit><nat>

<bool> → true | false

<const> → <nat> | <bool>

<exp> → <const> | <exp> + <exp> | <exp> - <exp>

<inst> → <char> = <exp> | if <exp> then <inst>

<prog> → ϵ | <inst> ; <prog>

Description et analyse des langages de programmation

Les expressions régulières ne permettent pas en général de décrire les langages. Les grammaires hors contextes, plus expressives, seront utilisées (avec notation BNF le plus souvent).
Pour produire l'analyse syntaxique des langages décrits de cette façon, il existe plusieurs algorithmes que nous n'étudierons pas ce semestre [1].

Nous allons tricher, et utiliser un programme permettant de générer des analyseurs (lexicaux **et** syntaxiques), pour les grammaires que nous lui spécifierons en entrée.

1. Les plus connus : LL(k), LR(k), LALR(1). Écrivez-moi si vous souhaitez des références.

ANTLR : analyse lexicale

Example

```
lexer grammar Example;

ID : [a-zA-Z]+; //identifiants
INT : [0-9]+; //entiers

NEWLINE : '\r'?'\n'; //retour ligne

WS : [ \t] -> skip; //oubli des espaces
```

La priorité suit l'ordre des règles : si je veux ajouter un mot réservé avec `LET : 'let'`, par exemple, il doit être placé avant la règle pour `ID` (sinon `'let'` sera reconnu comme un identifiant).

ANTLR : Analyse syntaxique

Pour la description d'une grammaire :

- On utilise une notation proche de la BNF.
- Les non terminaux sont en minuscule, et les règles suivent la syntaxe suivante : `nt: V1 |...| Vk ;` où les `Vi` sont des expressions comportant des terminaux, des non terminaux, et des caractères propres (comme `')'`).

[demo sur une grammaire pour des arbres binaires d'entiers] [2]

2. Indications pour les tests disponibles sur Celene

Examen d'un analyseur généré par ANTLR

Le code généré [3] permet de représenter les arbres de dérivation.
Les nœuds seront des objets de la classe `Context`.

[demo pour la grammaire des exemples précédents]

3. En java, pour ce qui nous concerne ce semestre