Leçon 923 : Analyse lexicale et analyse syntaxique. Applications.

Julie Parreaux

2018 - 2019

Références pour la leçon

[1] Carton, *Langages formels, calculabilité et complexité.*
[2] Legendre et Schwarzentruber, *Compilation : Analyse lexicale et syntaxique du texte à sa structure en informatique.*

Développements de la leçon

$\mathcal{L}(G_{post})$ est $LL(1)$ — Construction des premiers

Plan de la leçon

Motivation

Défense

La compilation permet de transformer un programme écrit dans un langage source en un programme sémantiquement équivalent écrit dans un langage cible. Généralement, on compile un langage de programmation comme C vers l'assembleur (le langage machine). Cependant, ce n'est pas la seule compilation que nous pouvons effectuer : on transforme du code latex en

fichier en format pdf, ou en code html. De plus, pour compiler un langage de programmation comme OCaml ou Python, on utilise des langages intermédiaire comme C.

L'analyse lexicale et l'analyse syntaxique sont les premières étapes de la compilations. Dans un compilateur actuel, elles sont réalisées en étroite collaboration : l'analyse lexicale donnant à l'analyse syntaxique les mots dont elle a besoin pour continuer. Elles permettent de transformer un texte en un arbre de syntaxe abstraite qui sera la structure sur laquelle nous pourrons continuer la compilation. Cet arbre nous permet ensuite de réaliser une analyse sémantique (vérification de type, levée de certaines exceptions, ...) avant de produire un code intermédiaire au langage que nous souhaitons atteindre. Sur ce code intermédiaire, nous effectuons des opérations d'optimisation qui vise à rendre l'exécution plus rapide que nous l'avons écrite. On finit alors par traduire le bout qui manque.

Ces deux analyses reposent sur des outils théoriques simples et dont l'expressivité nous permet d'obtenir des calcul efficaces : les expressions rationnelles et les grammaires algébriques. En étudiant ces deux étapes de la compilation, on remarque que les automates finis sont pratiques pour couper un texte en mots mais qu'ils se révèle insuffisant pour ordonnancer les éléments en fonction de leurs opérandes. Pour cela (et afin de construire l'arbre de syntaxe abstraite), nous sommes obligé d'utiliser la puissance d'expression des grammaires algébriques.

Ce qu'en dit le jury

Cette leçon ne doit pas être confondue avec la 909, qui s'intéresse aux seuls langages rationnels, ni avec la 907, sur l'algorithmique du texte.

Si les notions d'automates finis et de langages rationnels et de grammaires algébriques sont au cœur de cette leçon, l'accent doit être mis sur leur utilisation comme outils pour les analyses lexicale et syntaxique. Il s'agit donc d'insister sur la différence entre langages rationnels et algébriques, sans perdre de vue l'aspect applicatif : on pensera bien sûr à la compilation. Le programme permet également des développements pour cette leçon avec une ouverture sur des aspects élémentaires d'analyse sémantique.

Introduction

Motivation : Chaîne de compilation + importance de la compilation

1 Analyse lexicale

On commence par la première étape de la compilation : l'analyse lexicale qui s'avère être la plus facile.

— *Problème* : Transformer une suite de lettre en lexème

— *Valeur ajoutée* :

- Erreur lexicale (motif interdit dans le langage)
- Filtrer les programmes sur les blancs et les commentaires (équivalence de programme)
- Décoration des lexèmes produit (construire les futurs messages d'erreurs)

— *Méthode* : Pattern-matching sur les expressions régulières

1.1 Expressions rationnelles et automates finis [1, p.38]

On rappelle succinctement ces notions et leur équivalence. Celle-ci donnent les outils efficaces au cœur de l'analyse lexicale : les automates finis reconnaissants les expressions régulières caractérisant les mots autorisé dans le langage.

— *Définition* : Expression régulière
— *Définition* : Automate fini
— *Théorème* : Théorème de Kleene
— *Preuve* : Construction de Thomson (des expressions régulières aux automates)
— *Exemple* : Automate des motifs

1.2 Analyseur lexical

On décrit maintenant les différentes méthodes d'analyse lexicale ainsi que leur complexité.

— *Méthode* : Via les automates finis munis d'une priorité
— *Remarque* : Détection d'erreurs
— *Proposition* : Complexité au pire cas : $O(n^2)$
— *Remarque* : Dans le cas d'un langage de programmation, on est généralement en $O(n)$.
— *Méthode* : Via la programmation dynamique

2 Analyse syntaxique

L'analyse syntaxique est une étape plus compliquée à mettre en place qui demande l'utilisation d'outils plus conséquents : les grammaires algébriques.

— *Problème* : Transformer une suite de lexème en arbre de syntaxe abstraite
— *Remarque* : Problème qui est plus difficile
— *Valeur ajoutée* : Erreur syntaxique
— *Méthode* : ascendante ou descendante sur les grammaires

2.1 Grammaire algébrique

On rappelle succinctement les notions autours des grammaires algébriques. Ces notions vont nous être utile pour réaliser l'analyse syntaxique à partir de la grammaire de notre langage source.

— *Définition* : Grammaire algébrique
— *Définition* : Dérivation (gauche / droite)
— *Définition* : Arbre de dérivation
— *Remarque* : Différence entre une dérivation et un arbre de dérivation
— *Définition* : Ambiguïté
— *Remarque* : Impact lors l'analyse syntaxique
— *Définition* : Langage engendré
— *Remarque* : Les rationnels sont algébriques.

2.2 Analyse générique

Regardons les méthodes génériques : elles peuvent être utilisées pour toutes les grammaires algébriques.

— *Remarque* : La méthode "naïve" qui utilise du backtracking est exponentielle (oups)

— *Définition* : Grammaire de Chomsky

— *Théorème* : Toute grammaire peut être mise sous forme de Chomsky

— *Remarque* : De plus cette transformation n'est pas coûteuse

— *Algorithme* : CYK et sa complexité (programmation dynamique)

— *Proposition* : Le problème du mot est dans P

— *Application* : Analyse syntaxique

2.3 Méthode descendante

L'algorithme générique nous donne une analyse cubique. En réfléchissant sur la structure des grammaires des langages de programmation (qui sont pour la plus part agréable), nous pouvons grâce à des méthodes gloutonnes obtenir une analyse linéaire. Une première analyse est par méthode descendante dans la grammaire : on part de l'axiome pour obtenir le mot.

— *Principe* :

— *Définition* : Grammaire LL(1)

— *Proposition* : Caractérisation du premier DEV

— *Algorithme*

— *Remarque* : Expressivité : langages de programmations concernés

— *Limite*

2.4 Méthode ascendante

Une première analyse est par méthode ascendante : on part des terminaux pour obtenir l'axiome.

— *Principe* :

— *Définition* : Grammaire LR(0)

— *Définition* : Grammaire LR(1)

— *Algorithme* :

— *Remarque* : Expressivité : langages de programmations concernés

— *Limite*

Conclusion : Récapitulation des différents types de grammaires et de leurs méthodes d'analyse.

Ouverture

Analyse sémantique : λ-calcul simplement typé et correspondance de Curry-Howard.

Références

[1] O. Carton. *Langages formels, calculabilité et complexité*. Vuibert, 2008.

[2] R. Legendre and F. Schwarzentruber. *Compilation : Analyse lexicale et syntaxique du texte à sa structure en informatique*. Reference Sciences. Ellipses, 2015.