LIAGRE Enzo

Analyse et transformation syntaxique au service de la lisibilité d'un code

Positionnement(s) thématique(s) :

- Informatique (informatique pratique et théorique)

Mots-clés :

Mots-clés (français)	Mots-clés (anglais)
Analyse syntaxique	Parsing
Grammaire non contextuelle	Context-free grammar
Langage de programmation	Programming Language
Compilateur	Compiler
Paradigme de programmation	Programming paradigm

Bibliographie commentée :

L'analyse syntaxique d'un code s'inscrit souvent dans une démarche de compilation. On y retrouve un certain nombre d'enjeux de l'informatique pratique et théorique, qu'il s'agisse de créer des algorithmes de compilation raisonnables en temps et en mémoire, ou encore de l'analyse de grammaires non contextuelles afin de créer des liens entre un langage compris par la machine et un langage compris par l'Homme.

Linguistiquement, la compilation d'un code consiste, à partir des règles du langage de programmation, à trouver celles qui définissent ce code. Ce travail se fait dans un contexte formel de grammaire, formalisme introduit par M. Chomsky **[4]**. Pour éviter les définitions mathématiques peu praticables, a été introduite la notation en forme de Backus-Naur (Backus-Naur Form, ou BNF en anglais), expliquée en détail dans le cours de M. Janikom **[6]**, permettant une implémentation simple des grammaires à l'aide de types arithmétiques. On pourrait prendre en exemple la définition de la grammaire d'OCaml **[7]**, donnant les règles d'écriture d'un code en OCaml sous la forme de Backus-Naur.

Ainsi, l'analyse syntaxique est rendue possible par le lien entre la syntaxe d'une grammaire non contextuelle définissant un langage et la sémantique qui en découle. Entre autres, ce travail est possible grâce aux méthodes d'attribution de sémantique à la syntaxe définies par M. Knuth **[5]**. Cependant, cette syntaxe, pour être compréhensible par un humain (tout comme par l'ordinateur), doit suivre quelques principes. M. Hoare **[3]** discute de ces principes, utiles pour la lisibilité d'un langage comme pour celle d'un code.

Au niveau pratique, l'analyse syntaxique se fait avec ce que l'on appelle un parser (de l'anglais « parser »). Son but est de transformer la sémantique en arbre syntaxique. Pour obtenir une complexité polynomiale, au maximum cubique, on peut utiliser la programmation dynamique, construisant pas à pas l'arbre associé à un code **[1]**. Néanmoins, il reste envisageable, au vu du coût en temps que peut provoquer un tel parser lors de la compilation, de vouloir un algorithme plus rapide. Dans ce cas, l'utilisation du retour sur trace (en anglais « backtracking ») et de la mémoïsation fournit un algorithme en temps linéaire, au prix d'une complexité spatiale bien plus importante **[2]**.

Ainsi, l'analyse lexicale est autant un procédé théorique que pratique. Celle-ci requiert un certain nombre de considérations, allant du temps d'exécution à la mémoire sollicitée par l'algorithme, jusqu'à l'utilisabilité et la praticité des résultats.

Problématique retenue :

On cherche ici a appliquer des méthodes d'analyse syntaxique, sur un code OCaml, afin de la transformer pour le rendre plus lisible tout en conservant une compilation cohérente, au sens du paradigme de programmation choisi.

Objectifs du TIPE :

1. Pouvoir analyser un code écrit en OCaml.
2. Savoir transformer une partie itérative d'un code en une partie récursive.
3. Proposer une définition de « lisibilité » de sorte que le problème de lisibilité d'un programme soit calculable.

Références bibliographiques :

[1] EARLEY JAY : An Efficient Context-Free Parsing Algorithm : *https://dl.acm.org/doi/10.1145/362007.362035*

[2] FORD BRYAN : Packrat Parsing: a Practical Linear-Time Algorithm with Backtracking : *https://pdos.csail.mit.edu/~baford/packrat/thesis/*

[3] HOARE CHARLES A. R. : Hints on Programming Language Design. : *https://dl.acm.org/doi/10.5555/892013*

[4] CHOMSKY NOAM : Syntatic Structures : *https://www.ling.upenn.edu/courses/ling5700/Chomsky1957.pdf*

[5] KNUTH DONALD E. : Selected Papers on Computer Languages – Chapitre : Semantics of Context-Free Languages : *https://cs.stanford.edu/~knuth/cl.html*

[6] JANIKOM CEZARY Z. : What is BNF ? : *http://www.cs.umsl.edu/~janikow/cs4280/bnf.pdf*

[7] Institut National de Recherche en Informatique et en Automatique : The OCaml Language : *https://ocaml.org/manuel/5.4/lex.html*