Compilation : Analyses et Expérimentations

Nestor Laborier Guillaume Kineider
Blaise Bourret-Mathieu

Lycée La Martinière Monplaisir

18 novembre 2017

Mise en cohérence des objectifs du TIPE

Positionnement thematique

Informatique Théorique, Informatique Pratique

Mots-clés

Compilation	Compiling
Analyse lexicale	Lexical Analysis
Analyse syntaxique	Syntaxical Analysis
Programmation fonctionnelle	Functional programming
(Cloture)	(Closure)

Bibliographie commentée

Tout langage doit être interprété par la machine pour être exécuté, et pour cela il doit être compilé, c'est à dire traduit dans un langage compréhensible par la machine mais trop hostile à l'homme pour être utilisé directement. Selon le livre Compilateurs : Principes, Techniques et Outils [1], « un compilateur est un programme qui lit un programme écrit dans un premier langage – le langage source – et le traduit en un programme équivalent écrit dans un autre langage – le langage cible ». Le compilateur ne traduit donc pas nécessairement en langage machine mais vers un autre langage quel qu'il soit. Le compilateur lui-même est écrit dans un langage, qui peut même être celui qu'il traduit ! On parle alors de boostrapping [1]. Nous utiliserons pour notre part le langage Caml pour nos expériences [4].

Le compilateur sert aussi à détecter des erreurs dans le programme source [5] mais nous nous intéresserons uniquement à sa fonction de traducteur. Pour comprendre le programme source, il s'appuie principalement sur deux analyses : lexicale et syntaxique. Pour programmer ces analyses nous utiliserons deux outils fournis par Caml, CamlLex pour l'analyse lexicale, CamlYacc pour la syntaxique [3,4]. Les structures et les syntaxes spécifiques de ces 2 outils, détaillées principalement dans Formation au Langage Caml de Claude Marché [3], nous ont amenés à mieux comprendre les mécanismes de la compilation.

Pour limiter notre étude, nous nous sommes intéressés à deux langages en particulier. Pour le langage source, nous avons choisi Scheme et en langage cible, nous nous sommes portés sur Forth. Le choix de ces langages n'est pas anodin, en effet, leur structure de base (notamment pour les opérations arithmétiques) est assez aisée à comprendre et leurs syntaxes sont assez éloignées pour bien apercevoir la mise en œuvre des mécanismes de la compilation. Nous nous sommes initiés à Scheme grâce à l'ouvrage Structure And Interpretation of Computer Programs de H.Abelson et G.J.Sussman [2]. La documentation Le Langage Caml de Xavier Leroy [6] nous a aidé à comprendre la structure de pile utilisé par le langage Forth. Ces deux ouvrages avec l'aide de celui de Claude Marché [3] nous ont permis de créer, dans un premier temps, deux interpré-

teurs (Scheme et Forth), première étape vers la création de notre compilateur Scheme-Forth. Nous avons donc réalisé la première version de ce compilateur avec CamlLex et CamlYacc, mais leur fonctionnement interne étant très obscur, nous avons ensuite décidé de produire nous-mêmes un nouveau programme entièrement en CamlLight, ainsi qu'un programme représentant le résultat de l'analyse syntaxique sous forme d'arbre très lisible.