Compilation : Analyses et Expérimentations

Nestor Laborier Guillaume Kineider
Blaise Bourret-Mathieu

Lycée La Martinière Monplaisir

11 octobre 2017

Chapitre 1

Avancement du TIPE au fil de l'année

1.1 Travail antérieur

L'année précédente, nous avons réalisé un exemple de compilateurs pour Scheme vers Forth à l'aide des outils CamlLex et CamlYacc. Il fonctionne correctement avec les quelques fonctions arithmétiques qu'il est capable de gérer. Cette expérience m'a permis de commencer à comprendre les principes derrière la complation, notamment l'analyse lexicale et l'analyse syntaxique. Cependant il reste beaucoup de zones d'ombres, en particulier le fonctionnement des outils CamlLex et CamlYacc qui semblent, pour l'instant, un peu magiques.

1.2 Avant le 11 octobre 2017

Je me suis demandé si j'allais garder le sujet de compilation. Je me suis renseigné sur les garbages collector, et sur les principes abstrait derrière l'interprétation, la gestion de la mémoire, le fonctionnement basique de l'assembleur.

1.3 11 octobre 2017

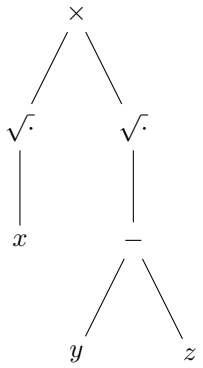
Nous avons donc pris connaissance des nouvelles modalités de l'épreuve des TIPE, d'où la forme de ce présent document.
Je m'intéresse aujourd'hui au code source de Ocaml afin de comprendre le fonctionnement de son compilateur, comparé au nôtre. Je remarque que Xavier Leroy a utilisé, comme nous, les outils CamlLex et CamlYacc

1.4 18 octobre 2017

J'ai revu mes objectifs pour le moment. J'ai décidé de recentrer mon étude sur l'analyse syntaxique à proprement parler. Nous continuerons d'utiliser CamlLex pour l'analyse lexicale, en attendant d'avoir plus de maîtrise concernant les expressions régulières pour la détection des lexèmes.
Notre groupe se concentrera pour l'instant sur la représentation du programme en entrée sous forme d'arbre. Mais cela nous confronte au problème de la curryfication. Notamment face à des fonctions telles que celle-ci :

$$f : x, y, z \longmapsto \sqrt{x}\sqrt{y - z}$$

Cette définition de fonction peut être représentée comme ceci :



La curryfication n'est donc pas évidente ici. Je vais tenter pour l'instant de programmer un compilateur qui rend un arbre syntaxique du programme fourni en entrée.

1.5 8 novembre 2017

Ma difficulté principale, pour le moment, est de trouver un moyen pour récupérer le résultats de l'analyse lexicale avant d'effectuer moi-même l'analyse syntaxique ... Ne sachant pas comment récupérer le résultat de l'analyse lexicale se passant "sous le capot" avec CamlLex, nous décidons collectivement de produire notre propre analyseur lexical, dont nous pourrons exploiter la sortie. Pour se limiter au strict minimum, nous définissons la liste de lexèmes reconnus suivante :

`PLUS +`	`TIMES *`	`LPAR (`	`VAR of string` *les mots*
`MINUS -`	`DIV /`	`RPAR )`	`NUM of int` *les nombres*

1.6 22 novembre 2017

J'ai commencé l'écriture de l'analyse lexicale. On ne dispose pas encore de la maîtrise des outils des expressions régulières, c'est donc avec lourdeur et patience que je commence à écrire l'analyseur lexicale. Blaise et moi travaillons sur une fonction qui sépare la chaine de caractères contenant le code du programme source au niveau des espaces afin d'isoler tous les lexemes naturellement. Nous aboutissons à la fonction `split_string` [1]

.

1.7 29 novembre 2017

Nous avons finalisé l'analyse lexicale ! Elle est contenue dans `lex.ml`[1] et utilise `split_string`. Guillaume, pendant ce temps, a commencé le développement du constructeur d'arbre syntaxique à partir d'une liste de lexèmes. Et Blaise s'est occupé du convertisseur d'arbre syntaxique en Forth.
Mon analyse lexicale est un simple filtrage brut, d'une liste de chaînes issue de la chaîne source qu'on a découpée selon des séparateurs (espaces, tabulations ou autres qu'on pourra définir ultérieurement). J'ai utilisé un petit subterfuge pour parvenir à différencier une chaîne contenant des lettres (donc un mot) d'une chaîne de chiffres (donc un nombre). J'ai ainsi créé une fonction qui renvoie ce qui est supposé être un nombre contenu dans la chaîne, et qui lève une exception si ce n'en est pas un. Cette exception est rattrapée en renvoyant `min_int` (valeur qui ne sera pas utilisée dans notre compilateur).
L'analyse lexicale renvoie alors une liste de lexèmes (type `lexlist`) utilisable dans notre analyse syntaxique.

1.8 6 décembre 2017

La génération d'arbre syntaxique de Guillaume est opérationnelle. Nous avons retravaillé ensemble dessus pour enlever les quelques petits bugs et problèmes d'indices qui restaient qui restaient. Elle traduit correctement des expressions arithmétiques simples en arbres syntaxiques. Nous aboutissons au fichier `synth.ml`[1]. Je décide de mon coté de commencer à construire un programme qui trace les arbres syntaxiques grâce au module `graphics` de Caml.

1.9 13 décembre 2017

A COMPLETER

1. Disponible en ligne sur notre GitHub : github.com/FDCB24DEC9/TIPE-2018