Projet d'ICO

Maxime Heckel & Philippe Gaultier

Introduction

Compiler, corriger les erreurs de syntaxe, exécuter: tel est le cycle habituel du développeur. Cependant ces outils familiers que sont le compilateur et l'analyseur syntaxique sont plus complexes qu'ils ne semblent, d'autant que chaque langage a ses spécificités et ses règles particulières.

Dès lors se posent les questions suivantes:

- Peut-on valider la syntaxe d'un programme à partir de règles de grammaires prédéfinies?
- Comment faire pour connaître la pile d'appel du programme, bien utile pour débugguer?
- Est-il possible de représenter graphiquement et simplement la structure d'un programme, même complexe?

Ce projet répond a ces questions en prenant pour base le langage `Pseudo-Pascal`.

Le but du projet

La finalité de ce projet de compilation était de créer un analyseur syntaxique d'un programme écrit en `Pseudo-Pascal`, avec les spécifications suivantes:

- Les erreurs de syntaxe sont détectées
- La structure du programme est analysée et le résultat est le graphe d'appel du programme, donné en syntaxe `dot`
- Le programme à analyser consiste en un seul fichier
- L'analyseur est écrit en `Ocaml`
- L'analyseur utilise comme lexer `ocamllex`
- L'analyseur utilise `ocamlyacc` pour spécifier les règles de grammaire

En plus de cela, notre projet final accomplit les choses suivantes:

- A partir du graphe d'appel en syntaxe dot (résultat de l'analyse syntaxique), on obtient le graphe d'appel sous forme graphique, pour plus de lisibilité
- Il est possible d'analyser plusieurs fichiers à la suite, au lieu d'un seul

Les outils utilisés

Ocaml

Le projet a été écrit en `Ocaml`. Pourquoi ce choix au détriment du `C`? Nous avions une expérience appronfondie de ces deux langages, et il est apparu que `Ocaml` avait plusieurs

avantages dans notre situation:

- Facilité d'utilisation (pas de pointeurs, pas d'allocations mémoires)
- Langage objet
- Libraire standard fournie (Hashtable, lecture/écriture de fichiers, etc)

De plus, nous n'avions pas de contrainte de performances, ce qui aurait pu faire pencher la balance en faveur du `C`.

Ocamllex

Nous avons utilisé l'outil `ocamllex` comme lexeur. Ce dernier a pour rôle de lire chaque caractère dans le fichier écrit en `Pseudo-Pascal`, et de reconnaître les motifs, nommés "tokens", spécifiés dans les règles de grammaire. Il est par exemple capable de reconnaître le token `if` en lisant le caractère `i`, puis `f`, puis ` `, à condition que le token `if` soit spécifié dans les règles de grammaire. De même, s'il lit `f` puis `i`, le mot-clé `if` du langage ne sera pas identifié, à raison.

En somme, c'est grâce à lui que les mots-clés du langages sont reconnus, et différenciés des noms de variables, des noms de fonction, etc.

Ocamlyacc

La pierre manquante à l'édifice est `ocamlyacc`, qui permet d'exprimer les règles de grammaire. C'est la que la syntaxe du `Pseudo-Pascal` est exprimée. Ainsi c'est ici que l'on édicte par exemple la forme que prend la structure de contrôle conditionnelle

```
IF expr THEN instructions ELSE instructions
```

C'est grâce à ces règles que l'analyseur syntaxique sait que la structure

```
IF THEN instructions ELSE instructions
```

est invalide.

On spécifie ainsi chaque brique élémentaire du langage: qu'est ce qu'un booléen, quelle forme prend la définition d'une fonction, quelle est la syntaxe d'une boucle, etc.

Conclusion

Ce projet nous a interpellé pour plusieurs raisons. D'abord, c'était la première fois que nous voyions le langage Ocaml dans un contexte semi-professionnel, en tout cas autre que purement académique. De plus, malgré leur apparante âpreté, l'utilité des outils `ocamllex` et `ocamlyacc` nous est rapidement apparue. Enfin, les potentiels prolongements de ce projet nous ont frappés. La création d'un langage complet ne semble plus si inaccessible.

En conclusion, malgré certains moments de découragement, le sentiment qui demeure à la

fin est la satisfaction d'avoir accompli ce qui nous apparaissait au début insurmontable, et d'avoir pu mettre en pratique nos apprentissages des cours de théorie de langage et de compilation.