RAPPORT DE PROJET

Master Science et technologie du logiciel

Analyse Syntaxique de Programmation

Rédigé par

Pierre FRANCK-PAPUCHON 21401699

1 Spécification du projet

Pour ce projet, nous avons programmé un langage. Ce dit langage possédait plusieurs parties (APS 0, APS 1, APS 1.5, APS 2 et APS 3). Or, pour ce projet, je suis allé jusqu'à APS 2 et ai commencé APS 3 sans l'avoir terminé. Ce projet se concentre sur l'implémentation de ce dit langage de programmation. Pour cela, nous avons dû faire différents choix. Voici une petite liste des premiers choix à faire :

— **Langage de programmation :** OCaml

— **Le build system :** Dune

— **Choix du parser et lexeur :** OCamlyacc

1.1 Implémentation

Nous avons ensuite dû construire nos différents éléments, comme l'AST, le parseur et le lexeur. Pour la composition de nos différentes parties, j'ai décidé de faire quelque chose de très littéral, ce qui permet une plus grande liberté dans l'implémentation. Par "littéral", je veux dire que j'ai choisi d'implémenter un nouveau type de nœud pour chaque sous-élément possible, afin de déconstruire au maximum notre langage. Dans le premier APS (APS0), je n'ai pas eu besoin de réfléchir à beaucoup de spécificités de programmation, car la spécification permettait simplement de le programmer

À partir d'APS 1, nous avons dû introduire de la mémoire (représentant le tas). Pour cela, contrairement à la spécification, j'ai décidé de la passer en variable globale, car j'ai trouvé cela plus conforme à la logique globale que le tas doit avoir sur la pile. De plus, j'ai directement choisi de mettre en place un compteur représentant le premier pointeur libre. (Nous pourrions envisager un ramasse-miettes (garbage collector), mais cette méthode poserait des problèmes : un premier pointeur pourrait alors se trouver à une position inférieure à notre compteur. On pourrait donc envisager de créer une liste stockant les différentes positions libres dans le tas.) Cette méthode s'avère finalement plus simple que celle consistant à renvoyer constamment la mémoire, car nous n'avons pas réellement besoin de modifier grand-chose dans le code.

Le reste de l'implémentation de nos différents APS est resté conforme à ce que la spécification préconise, y compris pour l'ébauche d'APS 3.

1.2 Test

Lorsque nous avons implémenté ces différents APS, nous avons dû réfléchir à des tests pour vérifier leur bon fonctionnement. Pour cela, j'ai décidé d'aborder les tests un peu à la manière du TDD (Test Driven Development), où chaque élément implémenté devait être testé de manière indépendante. Je partais du principe que si chaque élément fonctionnait individuellement, alors l'ensemble, plus global, fonctionnerait également.

Or, cette méthode s'est avérée non concluante, car lorsque j'ai essayé un code donné en cours :

```
1 [FUN REC f (int->int) [l:(int->int), n:int]
2     (if (eq n 0)
3     [y:int] y
4     [y:int] (l ((f l (sub n 1)) y)));
```

```
5       ECHO ((f [z : int] (mul 2 z) 3) 1) ]
```

J'ai pu mieux comprendre le langage, car l'évaluation de la fonction récursive posait problème. J'ai alors corrigé mon code pour qu'il fonctionne. Par la suite, j'ai décidé de conserver mes tests unitaires, mais d'y ajouter des programmes plus complexes afin de vérifier le bon fonctionnement de l'ensemble.

1.3 APS 3

Lorsque j'ai essayé d'implémenter APS 3, je me suis retrouvé face à certains problèmes d'implémentation, par exemple la compréhension du code mort ou les multiples définitions dans le typage, qui m'ont demandé un certain temps de réflexion. J'ai alors commencé une implémentation, mais je n'ai eu ni le temps de la terminer, ni celui de la déboguer. Normalement, le typeur devrait être terminé, mais pas l'évaluateur.