APS Rapport

PHAM Laetitia RIDEY Guillaume

April 2020

État d'avancement : Nous avons réalisé l'analyseur syntaxique, le typage et la sémantique pour le noyau fonctionnel (*APS0*) et le noyau impératif (*APS1*). Nous avons passé les bases de données de tests, situé sur le site 2018-2019 de l'UE, avec succès pour les deux implémentations. De plus ce projet a été réalisé en *Ocaml*.

1 Description de la structure

Notre projet se situe dans le répertoire *APS* qui est séparé en deux sous-répertoires, *APS0* et *APS1* qui correspondent respectivement à l'implémentations du noyau fonctionnel et du noyau impératif. Chacun de ces répertoires comporte:

- Au niveau de la syntaxe: la définition du lexique définit dans le fichier *lexer.mll* et la grammaire définit par un les fichiers *parser.mly* et *ast.ml* qui regroupe l'ensemble des structures de notre langage.

- Au niveau du typage: nous y trouvons l'affichage de la syntaxe d'un fichier *.aps* en des termes exploitable en Prolog, cela est défini dans le fichier *prologTerm.ml*, et l'analyse de type qui est définit dans le fichier *typrog.pl* (qui analyse le résultat donné par prologTerm).

- Au niveau de la sémantique: un fichier *eval.ml* qui définit quelle est la valeur ou l'effet attendu de l'exécution des programmes de ce langage.

De plus, nous pouvons trouvé un *Makefile* permettant soit la compilation du *PrologTerm* soit celle de l'évacuateur (selon lequel des deux est écrit en premier).

Finalement, pour la réalisation des tests de données celle-ci est réalisable dans dans les répertoires *APS0/Samples/APS0_test* et *APS1/Samples/APS1_test*, où se situe les scripts *testProlog.sh* et *testEval.sh*. Ainsi il suffit d'exécuter ces scripts pour analyser le type ou évaluer tout les tests.

2 APS 0 : Noyau Fonctionnel

Lors des deux premières semaines nous avons hésité sur le choix de langage qui implémentera notre projet. En effet, nous avions tout d'abord commencé le projet en choisissant Java comme langage de développement, car nous avions tout deux suivi l'UE de DLP au premier semestre. Cela nous semblait donc être le choix le plus judicieux. Cependant dés la deuxième séance nous nous sommes rendu compte que contrairement à DLP nous partions avec un squelette de départ beaucoup plus pauvre. L'utilisation de Java nous aurait donc demandé de créer beaucoup de fichiers différents (par exemple une classe java pour chaque Ast), ce qui nous aurait fait perdre du temps inutilement. Nous avons donc pris la décision d'utiliser un langage fonctionnel, Ocaml, qui nous permet ainsi de regrouper toute les Ast dans un seul et même fichier par exemple.

L'une des divergences avec la spécification de la grammaire a été de séparer la primitive *not* des autres primitives. En effet, contrairement aux autres primitives *not* ne prend qu'un seul argument, sa signature dans l'Ast est donc différente.

De plus un autre choix d'implémentation a été de considérer les "string" comme des expressions au niveau de l'Ast. En effet, d'après la grammaire les identifiants (appelé *AstId* dans l'Ast) sont considérés comme des expressions ainsi dans le *parser* pour garantir que ceux-ci soient bien du type "string" le parser ne doit récupérer que des valeurs sous le format *IDENT*.

3 APS 1 : Noyau Impératif

Nous avions commencer la partie syntaxe du Noyau Impératif avant le confinement en utilisant nos notes de cours, or celles-ci ne comprenaient pas dans la grammaire l'extension *Block* nous ne l'avons donc pas implémenté tout au long du projet, contrairement à ce qu'il y a marqué dans les documents de l'année dernière.

Au niveau de la sémantique d'évaluation, les booléens correspondent aux valeurs *InN(1)* et *InN(1)*, respectivement vrai et faux. Cependant cela créer des résultats inattendus, en effet dans l'exemple *APS1_test/prog14.aps* ci-dessous:

```
[
  PROC printBool [b:bool]
    [
      IF b [ ECHO 1 ] [ ECHO 0 ]
    ];
  CALL printBool (lt false true)
]
```

Dans l'analyse de type de ce programme, celui-ci n'est pas correctement typé. En effet, l'opérateur *lt* doit prendre en entré des entiers et non pas des booléens. Cependant au niveau de l'évaluation cela affiche bien un résultat étant donné que les booléens sont évaluer comme étant des la domaine des entiers ce qui permet à la condition d'être évaluer sans erreur. Nous avons donc pu constater les limites de l'évaluation d'un programme et la complémentarité qu'offre l'analyse de type.