Rapport Projet APS

Lien GitLab du Rendu :
https://stl.algo-prog.info/28600748/aps

auteurs :
Yanis Alayoud
Hamid Kolli

Sorbonne Université
MU4IN503
Année universitaire 2022-2023

1 installation et execution du projet

Le point d'avancement du projet : ASP3 FINI, le code final est dans la branche main, vous n'avez qu'à le clone car vous êtes ajouté en tant que maintainer du projet

Comment exécuter les différents tests :

- assurez-vous tout d'abord de bien avoir **SWIPL** d'installé sur la machine
- commandes ou script pour **évaluer et vérifier le typage de l'ensemble des tests fournis (APS0 à APS3)**:

  $ chmod u+x ./testaps.sh

  $ chmod u+x ./exeprog.sh

  $ chmod u+x ./typprog.sh

  $ ./testaps.sh

- commandes ou script pour **vérifier le typage de l'ensemble des tests fournis (APS0 à APS3)**:

  $ chmod u+x ./typprog.sh

  $ chmod u+x testtypage.sh

  $ ./testtypage.sh

- commandes ou script pour **obtenir le terme prolog d'un programme en particulier**:

  $ make prologTerm

  $ ./prologTerm $PATH_TEST

- commandes ou script pour **vérifier le typage d'un programme en particulier**:

  $ make prologTerm

  $ chmod u+x ./typprog.sh

  $ ./typprog.sh $PATH_TEST

- commandes ou script pour **évaluer un programme en particulier**:

  $ make eval

  $ chmod u+x ./exeprog.sh

  $ ./exeprog.sh $PATH_TEST

2 Portée du Projet

Il est important de noter qu'il est compliqué de savoir si notre implantation d'APS est 100% fonctionnelle étant donné qu'il faudrait une liste extrêmement exhaustive de tests afin de s'en assurer. Cependant nous avons testé chacune des nouvelles spécificités de chaque version d'APS dans le dossier Sample et tous les tests passent en étant correctement typés et évalués (**53 programmes tests au total répartis à travers APS0,1,2 et 3**).
De plus nous avons donc bien fait attention à ce que notre APS3 passe bien tous les test d'APS0 à APS3, et de même pour toutes nos versions d'APS intermédiaires (APS2 et APS1).

De ce fait, tout semble fonctionner étant donné les tests que nous avons implanté, mais un aspect que nous n'avons pas vérifié en détail, est la possibilité que certains tests puissent passer alors que ce n'est pas le cas (**faux-positifs**).

En effet, lors de nos tests nous avons passé plus de temps à vérifier qu'un test correct passe, plutôt qu'à vérifier qu'un test incorrect ne passe pas, il est donc possible que notre implantation puisse laisser passer quelques programmes incorrects mais il nous est impossible de l'affirmer.

3 Choix d'implantation et difficultés rencontrées

Tout d'abord, il faut savoir que bien que le projet soit réalisé en **Ocaml**, nous avions commencé en **Java** lors des 2 premières semaines.
Le choix de base était lié au fait que nous pouvions utiliser ANTLR de la même manière que dans l'UE DLP du semestre précédent, mais nous avons vite été découragés par la quantité d'interfaces et de classes java nécéssaires pour faire la moindre chose ou créer le moindre **AST**, avant de se rendre compte que le code en Ocaml aurait été beaucoup plus "compacte", notamment les AST qui peuvent simplement se créer en une seule ligne.
Choix qui nous a été extrêmement bénéfique car nous avons pu rattraper notre "retard" (lexer+parser) très rapidement et s'attaquer au **typeur/évaluateur d'APS0** qui correspondaient à l'axe principal du projet.

En parlant du typeur, le typeur d'APS0 est probablement la partie qui nous avait donné le plus de fil à retordre, car bien que développer le code prolog d'APS0 pour les versions suivantes était plus simple, avoir notre premier typeur fonctionnel à pris plus de temps.
Cela s'était notamment ressenti pour l'interprétation de concepts tels que le **type fleche** (t1− >t2− >t3...), où nous avons mis un peu de temps avant de trouver quelque chose de correct.

Pour ce qui est des tests, nous nous sommes assurés d'en faire des très basiques et simples au départ pour en faire des plus complets par la suite. Celà est lié au fait que nous avions eu un peu de mal à **débugger les traces d'executions prolog** avec des codes trop longs et trop complexes, ça nous a donc permis de beaucoup plus rapidement trouver la nature du problème (prédicat jamais appelé avec le bon argument ou type trop générique acceptant des types qu'il n'est pas censé accepter etc...), pour ce fait nous avons également beaucoup utilisé **SWISH** pour pouvoir directement tester en ligne et avoir une trace d'exécution bien plus visuelle et simple à interpréter.

Ainsi, Bien que le début du projet lors des premières semaines était relativement complexe, une fois que nous étions parvenus à finir l'implantation d'APS0, le reste a pu continuer sans encombres.
En suivant au pied de la lettre le formulaire et les notes de cours nous avons pu implanter le reste du projet. (à l'exception d'APS2 où nous avions du faire abstraction des **quelques coquilles** trouvées dans le formulaires, pour ne citer qu'un seul exemple, dans la sémantique il est indiqué que **LEN** ne retourne pas de **inZ**, ce qui est un peu illogique et contradictoire avec ce qui est indiqué dans les notes de cours)