Rapport d'APS

Kelly Seng
Pierre Gomez

13 mai 2019

Ce projet a été réalisé en Ocaml.

Mode d'emploi

Dans chaque sous-dossier du projet correspondant à chaque niveau d'implémentation du langage APS (0,1,2,3), il faut suivre la procédure suivante pour tester le typage et l'évaluation du langage APS. Nous disposons d'un script `aps.sh` dont la fonction est d'appliquer ce test sur tous les fichiers de test qui se trouvent dans le sous-dossier "exemple". Ces fichiers de tests présentent délibérément des problèmes de typage : prog018.aps, prog114.aps,prog118.aps, prog304.aps,prog305.aps Le reste des exemples sont syntaxiquement corrects et bien typés.

- Créer les exécutables : make
- Lancer le typeur et l'évaluateur sur tous les fichiers d'exemple : ./aps.sh
- Pour supprimer les exécutables : make clean

Il est également possible de tester un seul fichier avec les instructions suivantes.

- Créer les exécutables : make
- Génération de code en Prolog : ./toProlog exemple/prog000.aps (modifier le nom du fichier .aps)
- Typage :

  - swipl

  -[typer]

  -typeProg([],(inserer le résultat de la génération de code en Prolog),void).
- Evaluateur : ./eval exemple/prog000.aps

Réalisation

Nous avons pu réaliser l'intégralité de APS0, APS1, APS2 et APS3, ce qui inclue le typage et l'évaluation pour tous les niveaux d'APS.

Tout fonctionne dans APS0 et APS1.

Dans APS2, tout fonctionne sauf la gestion des tableaux à plusieurs dimensions comme (vec (vec bool)) au niveau de l'évaluateur.

Dans APS3, nous avons rencontré un problème lorsque nous avons un RETURN dans un block puis un autre RETURN en dehors de ce block. Ce cas se présente sur les fichiers de test "exemple/prog302.aps" et "exemple/prog303.aps" Dans le cas de l'exemple "exemple/prog302.aps", le programme retourne 41 au lieu de 42. Hormis ce problème, tout fonctionne.

Chaque implémentation est cohérente avec celles qui la précédent, c'est-à-dire que les exemples de APS0,APS1 et APS2 fonctionnement avec le typeur et l'évaluateur de APS3.

Choix d'implémentation

Typage

Un programme mal typé renvoie **false**, un programme correct renvoie **true**.

Nous considérons une suite d'arguments, de commandes, d'expression ou de types comme une liste en Prolog. Le typeur comprend plusieurs règles :

- typeExpr(Env,Expression,Type) : vérifie le type d'une expression
- typeDec(Env,Declaration,Type) : vérifie le type d'une déclaration
- typeStat(Env,Statement,Type) : vérifie le type d'une instruction
- typeCmds(Env,Commands,Type) : : vérifie le type d'une suite de commandes
- typeProg(Env,Prog,void) : vérifie que le type du programme est toujours void, il s'agit du point d'entrée du typeur

L'environnement correspond à une liste de couples (x,y) tel que x est un identificateur de type String et d'un type t.

Pour le typeur d'APS3, nous avons décidé de représenter les types mixtes sous la forme d'un couple de type.Par exemple, pour représenter t+void, on aura (T,void).

Évaluation

L'environnement est représentée par une liste de couples (String id,Valeur v) et la mémoire par une liste de couple (Valeur v1,Valeur v2) avec v1 qui correspond à une adresse. Le flux de sortie correspond à un String.

Nous avons décidé d'initialiser une variable ou un tableau avec la valeur entière -42 donc il n'est pas possible de se retrouver avec une exception lorsque la valeur n'a pas été définie. Le désavantage de ce choix est la difficulté à distinguer un résultat qui vaut -42 ou une valeur non initialisée.

Pour APS2, il ne nous a pas été possible de renvoyer une valeur de bloc mémoire car cela engendrait une longue attente et pouvait potentiellement un blocage de la machine qui exécute ce programme. Ainsi, nous avons renvoyé à la place l'adresse qui correspond à ce bloc et stocké une variable la taille de bloc.