Sorbonne Université
Faculté de Science et d'ingénierie
Département Informatique

Rapport du APS

Informatique

Spécialité :
Science et Technologie Logiciel

TME

Analyse de programme et sémantique

Encadré par

- R.Demangeon
- L.SYLVESTRE

Réalisé par

- Tabellout Yanis
- Tabellout Salim

le : **24/04/2024**

TABLE DES MATIÈRES

TABLE DES FIGURES

CHAPITRE 1

APS ET STRUCTURE DES FICHIERS

Le langage APS est un langage basique dont on essaye de faire évoluer au fur et à mesure, chaque version d'APS représente un noyau pour les APS qui le suivent, initialement *APS0* peut être désignés comme étant un langage d'expression, puis qui sera de plus en plus expressive en rajoutant certaines définition, manipulation de la mémoire ainsi que des *pseudo pointeur*. Il est important de savoir que chaque modification dans un langage ne doit pas casser ce qui a été déjà fait, ce qui représente un peu très important dans les choix qu'il faut prendre à la conception d'APS0 car certains choix, parfois engendrent plus de dégats et rendre l'extension de langage faisable ou carrément impossible.

1. Structure Génrale du code

Chaque Partie ou version d'APS a été séparé dans son propre répertoire, chaque répertoire est divisé comme suit :

- **archive** : Contient les scripts archivés.
- **expected** : Contient les résultats attendues pour chaque Samples pour le typage et l'évaluation.
- **results** : Contient les résultats obtenues lors du typage et de l'évaluation des programmes ainsi qu'un fichier ***csv*** qui nous permettra de faire certains statistiques.
- **Samples** : Contient les différents examples à traiter.
- **Checker.pl** :Le typeur en prolog.
- **eval.ml** :évaluateur.
- **test.sh** : Script pour lancer les tests de l'évaluateur et le typeur (vaut mieux utiliser ipynb).
- **type.sh** : Script pour lancer le typeur sur un fichier cible.

- **Setup.ipynb** : **Notebook** qui nous sera utile pour : *lancer le makefile, générer le output du typeur et l'évaluateur, lancer des tests automatisés pour vérifier la conformité du output obtenues par rapport à celui attendu, affichier la matrice de confusion*

Pour la suite, nous vous conseillons d'utiliser le notebook proposés à la place du fichier shell car il permet d'avoir des résultats un peu plus avancé. En cas de problème veuillez installer les librairies qui sont en import [1].

2. Choix généraux et conceptuelles

Comme dit, nous avons fait certains choix notamment :

- **L'évaluateur** : nous avons choisi **ocaml** pour plusieurs raisons notamment sa simplicité, code très minimaliste notamment pour les noeuds AST, mais aussi grâce à l'efficacité du parcours d'ocaml pour les arbres.
- **Mémoire et environnement** : nous avons opté d'utiliser les lists d'ocaml pour simuler le comportement de la mémoire et de l'environnement. Pour la mémoire nous avons choisi de représenter les addresses par des entiers et de permettre de stocker en mémoire pas que les entiers et les blocs mais toutes les valeurs.
- **Les types** : Nous avons pris le choix de se rapprocher au maximum à la syntaxe du cours pour nous permettre de bien appliquer les notions appliquer en cours.
- **Variables** : Nous avons considéré que dans l'évaluateur, nous nous limitons pas qu'aux entier et aux booléen, mais plutôt les types en entier permettant à un tableau d'être stocké dans une variable par exemple.

3. Ressource

Certaines parties de ce projet a été inspiré par un projet [1] open source qu'on a trouvé sur github, que certains parties notamment pour l'analyse syntaxique d'APS0 (*types et typ*) et certains jugements de typage (parcours de liste et récupérer le type des arguments d'une fonction) . Cependant le repo contient pas mal d'erreurs donc on s'est basé sur notre propre comprehension du cours mais aussi de l'aide de Louic qu'on salue.

4. Portée projet

Dans ce projet d'APS, nous avons traité **tout ce qui a été mentionné** dans le cours ainsi que dans le formulaire tout en respectant (dans la mesure du possible) toutes les règles indiqué (typage et sémantique)

1. en lançant la commande : *pip install* le nom du paquet

CHAPITRE 2

VERSIONNAGE D'APS

1. APS0

APS0 introduit le noyau de base de notre langage, nous pouvons dire que c'est un langage d'expression ou nous avons définit certaines opérations de base (addition, ...)

1.1. Typage

Nous avons définit des jugements de typage de base qui seront utilisé par la suite. Nous avons aussi définit l'environnement de typage de base noté G_0. Les jugements de typage sont justes une application des règles de typage vu en cours. Nous nous sommes inspiré de [1] sur certains jugements plus précisement sur le parcour des listes et récupérer les types des arguments des fonctions.

1.2. Évaluateur

L'évaluateur représente la partie la plus intéressante de ce projet car on a beaucoup plus de liberté. Nous avons définit certains type primitif telle que le Add, Sub car ça nous a permit d'avoir d'étendre les possibilités d'APS [1]. Nous avons comme dit précédement pris le choix de définir l'environnement comme étant une liste, et un compteur et un **Binding** qui est juste un couple clé valeur. Nous avons définit le type de base value qui peut être soit un : *InZ, InF, InFR et InPrim* [2]. Nous avons implémenté certaines fonctionnalités de base tel que l'accès à l'environenment, vérification de type (valeur booléenne), définition des primitives, mais aussi les jugements d'évaluation pour les expressions, les instructions et les définitions.

1. e.g : *(if true add sub)*
2. InPrim : représente les primitives à deux ou trois arguments

1.3. Tests et résultats

Nous avons rajouté des fichiers result (eval et type), les résultats obtenues sont conformes aux résultats attendues. Cependant dans les exemples nous avons rajouté des exemples pour pouvoir étudier un peu les possibilités qu'on pouvait avoir avec APS, notamment pour tester si un programme non typable mais évaluable (et vice versa). Nous avons rajouté un ou deux exemples exprès qui ne sont ni typable ni évaluable. La figure suivante présente la matrice de confusion d'APS0.

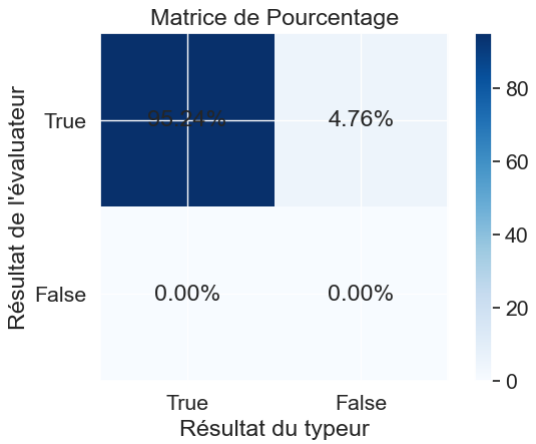

FIGURE 2.1 – Matrice de confusion APS0

2. APS1

APS1 rajoute des notions un peu plus avancé ou on rajoute plus d'instructions et de définition ainsi qu'un type de base qui est le void.

2.1. Typage

Nous avons utilisé les jugements d'APS0 auquelles nous avons rajouté les règles de jugements d'APS1 comme les instructions, des variables ainsi que celles des procédures.

2.2. Évaluateur

Le gros changement dans APS1 est la défintion de mémoire dans l'évaluateur, nous avons pris le choix d'aussi considérer la mémoire comme étant une liste de **Memory** qui est un couple **Addresse et valeur** [3], et on a considérer que les addresses sont des **des entiers** car on essaye de s'inspirer du monde réelle ou la mémoire est représenté par des valeurs en hexa. L'allocation mémoire est maintenue grâce à un compteur de référence (inspiré par [1]) qu'on met à jour à chaque allocation mémoire. Nous avons aussi définit

3. contrairement au cours ou les valeurs en mémoire peuvent être que des entiers, nous avons pris le choix de considérer qu'on peut sauvegarder n'importe quelle valeur en mémoire :
($ID\,1$) Si $x \in \text{ident}$ et si $\rho(x) = \text{inA}(a)$, alors $\rho, \sigma \vdash_{\text{Expr}} e \rightsquigarrow \sigma(a)$.

les fonctionnalités de base pour l'accès et la mis à jour de mémoire. Nous avons rajouté aussi certains type de valeur de notamment *InAddress, InP, InPR* mais aussi une valeur de **None** qui représente une sorte de *undefined*. Certaines règles ont été mis à jours d'autres justes amendés pour permettre l'utilisation de la mémoire.

2.3. Tests et résultats

On rappelle qu'on rajoute exprès des faux exemples pour voir oui ou non nos programmes peuvent être typable mais pas non évaluable.

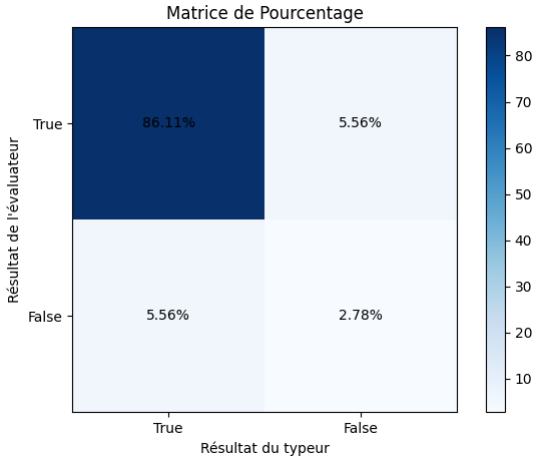

FIGURE 2.2 – Matrice de confusion APS1

3. APS1A

APS1A permet de corriger certaines erreurs délaissés par APS1, notament les références et les valeurs mais aussi le passage des paramètres par valeurs et par référence. Contrairement au passage d'APS0 à APS1, le passage d'APS1 vers APS1A nécessite beaucoup moins de modification car c'est juste une complémentarité à APS1.¨Pour le typage, nous avons rajouté aussi certains jugements pour les paramètres des procédures qui ressemblent à celles de fonctions mais juste en utilisant les règles **expar (ref et val)** définit en cours à la place des expressions. Pour l'évaluateur on a rajouté les fonctions de base pour accéder aux arguments des procédures et de les manipuler ainsi que la règle d'expar. On remarque sur la matrice de confusion par exemple que le taux de programme qui sont ni typable ni évaluable ont augemnté et que les programme qui sont typable mais pas évaluable a baissé, une des raison que dans nos exemples nous avons décidé de faire des exemples qui passent le typage dans APS1 mais pas dans APS1A notamment avec les références.

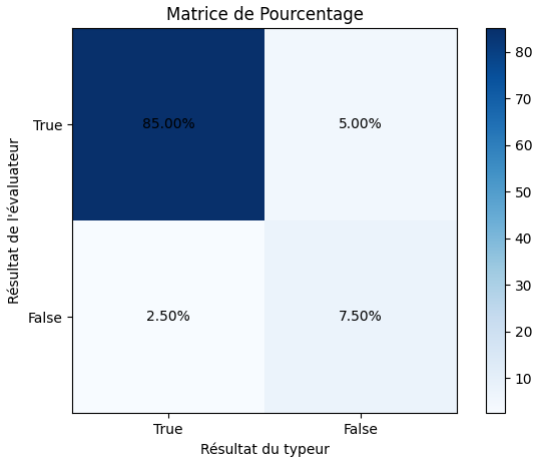

FIGURE 2.3 – Matrice de confusion APS1A

4. APS2

APS2 introduit le principe de liste et l'allocation d'un bloc au lieu de l'allocation d'un élément. Pour le typage nous avons juste rajouté les règles concernées (alloc, nth , len $\cdots$). Pour l'évaluateur, nous avons définit le type **bloc** qui est juste un **couple addresse et un entier** représentant la taille du bloc, le bloc est aussi donc une valeur (). Nous avons définit certaines fonctions qui nous seront utiles par la suite comme la fonction **allocn** qui se charge d'allouer un bloc. La plus grosse modification pour APS2 c'est de permettre au expression de modifier la mémoire c'est pour celà que les règles d'évaluation ont été amendés pour permettre celà, mais nous avons rajouté des nouveaux traits d'APS2 surtout pour l'accès aux listes, l'allocation, mais aussi l'évaluation de la liste qui sera utile pour le set.

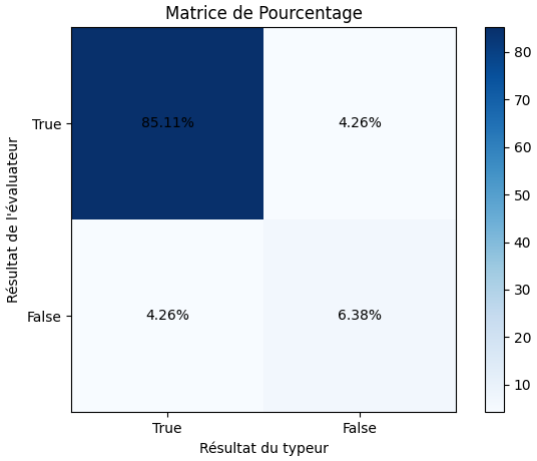

FIGURE 2.4 – Matrice de confusion APS2

Difficulté

La plus grosse difficulté était l'utilisation de prolog et même ocaml

Conclusion et perspective

Nous avons implémenté un modèle d'un langage minimaliste durant ces dix dernières semaines tout en respectant les règles formelle d'APS. Une des choses qu'on souhaiterait faire c'est de terminer APS3 et peut être ajouter certains traits des langages hauts niveaux.

BIBLIOGRAPHIE

[1] valeeraz. https://github.com/valeeraZ/Sorbonne_APS/blob/master.