SORBONNE
UNIVERSITY

Faculté des Sciences et Ingénierie

Parcours Science et Technologie du Logiciel (STL)

Analyse des Programmes et Sémantique (APS) : Conception d'un Langage de Programmation Interprété avec Analyseur, Vérificateur et Évaluateur Intégrés

Étudiants :
Abdelkader Boumessaoud, N°21218306, abdelkader.boumessaoud@etu.sorbonne-universite.fr
Hichem Bouzourine, N°21319982, hichem.bouzourine@etu.sorbonne-universite.fr

Encadrants :
Romain Demangeon, Romain.Demangeon@lip6.fr
Loïc Sylvestre, Loic.Sylvestre@lip6.fr

24 janvier 2024 - 7 mai 2024

Table des matières

1 Procédure d'installation et d'utilisation

1.1 Prérequis

— Le langage OCaml.
— L'environnement SWI-Prolog.

1.2 Installation

```
git clone https://stl.algo-prog.info/21319982/APS-tme
cd APS-tme/
cd APSx/
make
```

1.3 Scripts et Exécution

Les commandes suivantes illustrent l'utilisation des scripts pour diverses tâches :
— Pour exécuter le script `test_samples.sh` :

```
bash test_samples.sh
```

les résultats seront sauvegardés dans 2 fichiers CSV, *output_ typage.csv* et *output_ evaluation.csv*

— Pour afficher la requête Prolog avec `prologTerm` :

```
./prologTerm Samples/Sample.aps
```

— Pour vérifier le type avec `typer (swipl)` :

```
swipl typer.pl
?- trace.
[trace]  ?- g0(L),type_prog(L, ---sortie prolog--- , G).
```

— Pour évaluer avec `evaluateur` :

```
./evaluateur Samples/Sample.aps
```

1.4 Débogage

1.4.1 Lexer

Une règle spécifique de débogage est utilisée pour identifier et signaler les caractères inattendus avec leur position exacte (ligne et colonne). Cette méthode améliore le diagnostic et la correction des erreurs lexicales, renforçant la fiabilité de l'analyse syntaxique.

```
| _ as char        {
    let pos = lexbuf.lex_curr_p in
    let msg = Printf.sprintf "Lexing error: unexpected character '%c' at line %d, position %d"
                             char pos.pos_lnum (pos.pos_cnum - pos.pos_bol + 1) in
    raise (Failure msg)
  }
```

1.4.2 Programme .APS

Pour renforcer l'analyse syntaxique de notre projet, nous avons implémenté un mécanisme de gestion des erreurs avec l'aide de notre chargé de TME [6][7]. Ce système identifie et signale précisément les erreurs de syntaxe, indiquant leur emplacement exact (ligne et colonne), ce qui facilite grandement leur correction rapide.

```
with _ ->
      let open Lexing in
      let curr = lexbuf.lex_curr_p in
      let line = curr.pos_lnum in
      let cnum = curr.pos_cnum - curr.pos_bol in
      Printf.printf "Erreur de syntaxe à la ligne %d, colonne %d\n" line cnum;
      exit 1
```

1.4.3 Typeur

Notre démarche de débogage des exemples d'APS était simple, comme on a vu dans la salle TME, en utilisant la commande de *trace* de Prolog.

1.4.4 Évaluateur

Dans le développement de l'évaluateur d'APS, le débogage a été réalisé principalement par l'utilisation de la fonction failwith pour signaler des erreurs critiques et des impressions via Printf.printf pour suivre l'exécution et les états internes. Ces méthodes ont aidé à identifier rapidement les problèmes de logique et de gestion de la mémoire, facilitant les corrections nécessaires.

2 Portée globale

2.1 État d'avancement

- APS0 : DONE
- APS1 : DONE
- APS1a : DONE
- APS2 : DONE
- APS3 : Typage seulement – nous avons essayé de commencer le typage mais il y avait beaucoup de notion avancé qu'on a pas vu en cours, nous avons alors fait de notre mieux pour faire les règles de typage ainsi que quelques exemples.

2.2 Tests et Résultats

Tous les résultats de test sont listés pour chaque version d'APS dans leurs répertoire respective :

- pour le typage dans : *output_typage.csv*
- pour l'evaluation dans : *output_evaluation.csv*
- expected_evaluation_results.csv sert a liste les résultats d'evaluation attendus pour chaque sample.aps

Tous les tests exécutés sur diverses versions d'APS passent, tant du côté du typage que de l'évaluation.

2.2.1 APS1 vs APS1a

Dans le répertoire Samples de APS1 nous avons inclus des programmes créés specifiquement pour APS1a pour constater qu'ils **ne passent pas** au typage et ne s'évaluent pas, preuve que

les evolutions APS1 -> APS1a ont une réelle plus-value

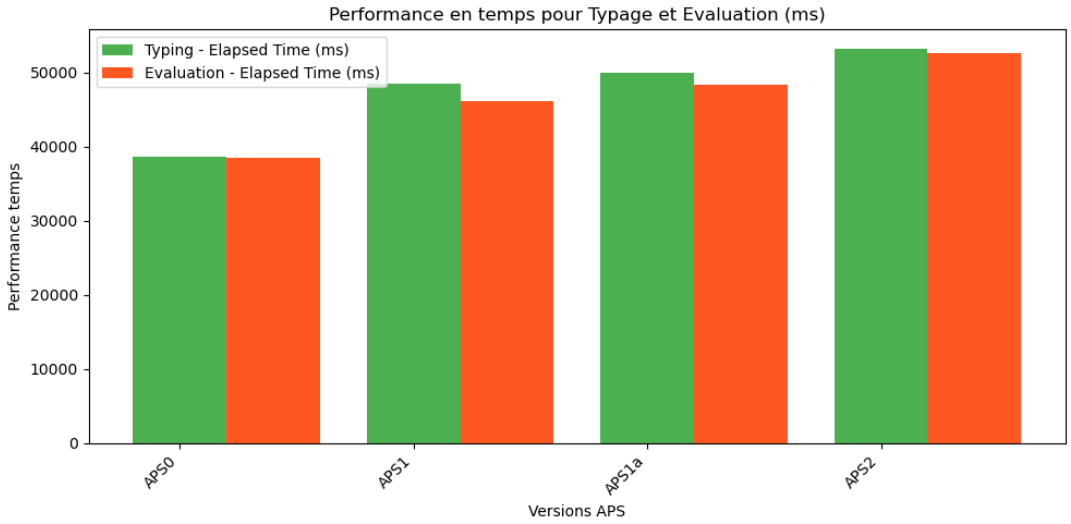
2.2.2 Multiple ECHOs

La capacité d'évaluer un programme APS contenant plusieurs instructions ECHOs a été intégrée dès APS1, mais l'affichage des résultats de plusieurs ECHOs en sortie n'a été implémenté que jusqu'à APS2. Après consultation avec notre chargé de TME et suite à ses conseils, il a été décidé de ne pas revenir aux versions antérieures à APS2 pour ajouter l'affichage des multiples ECHOs.

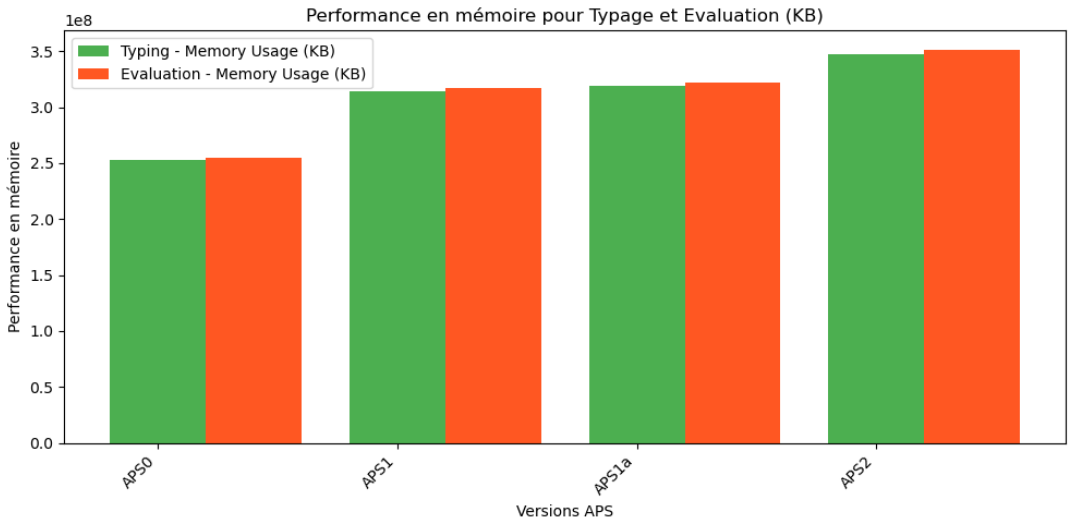
2.3 Fonctionnalités vérifiées

À travers nos tests diversifiés, nous avons tenté de couvrir le plus de fonctionnalités possibles de différentes versions d'APS, mais il est possible que certains cas aient été omis.

2.4 Performances



(a) `Performances en temps (ms)`



(b) `Performances en mémoire (KB)`

FIGURE 1 – Performance pour le Typage et l'Evaluation

3 Choix d'implémentation

3.1 Évaluation

3.1.1 APS0

— **Structures de données principales :** Utilisation de `Map.Make(String)` pour créer un environnement (`Env`) qui mappe les identifiants aux valeurs. Ceci est crucial pour la gestion des scopes locaux dans les expressions lambda et permet une évaluation efficace sans état global mutable.

— **Traitement des expressions et commandes :**
- Les opérations primitives (`eval_prim`) et booléennes (`eval_bool`) sont traitées par des fonctions spécifiques qui simplifient l'analyse et l'évaluation.
- Le support pour les expressions lambda et l'appel de fonctions utilise des environnements fermés (`InF` et `InFR` pour les fonctions récursives), ce qui aide à capturer les liaisons de variables lors de la définition de la fonction.
- Les structures de contrôle comme `if`, `and`, et `or` sont directement gérées dans `eval_expr`, permettant un contrôle de flux basé sur les valeurs évaluées des expressions conditionnelles.

3.1.2 APS1

— **Gestion de la mémoire :** Introduction d'un système de gestion de mémoire (`memory`) qui mappe des adresses à des valeurs, permettant le stockage et la manipulation de variables impératives.

— **Extensions par rapport à APS0 :**
- Implémentation de commandes impératives (`eval_stat`) qui modifient l'état de la mémoire, comme l'affectation et les structures de contrôle basées sur des conditions mutables.
- Les fonctions et procédures sont enrichies pour inclure des environnements à leur fermeture, supportant des fonctions de haut niveau avec des liaisons de portée correctes.

3.1.3 APS1a

— **Optimisation de la gestion de mémoire :** Même structure de mémoire que APS1, mais avec une attention accrue sur l'efficacité de l'allocation mémoire et la manipulation des références pour optimiser les performances.

— **Passage par référence :** Ajout du support pour le passage par référence dans les procédures, ce qui permet des modifications directes sur les arguments passés, réduisant ainsi le besoin de copier de grandes structures de données.

3.1.4 APS2

— **Support pour les structures de données complexes :** Introduction de types complexes tels que les blocs de mémoire et les tableaux. Les opérations spécifiques aux tableaux (`allocn`, `nth_vector_element`) permettent une gestion dynamique des collections de données.

— **Évaluation avec effets secondaires :** L'évaluation des expressions peut avoir des effets secondaires sur la mémoire (`sigma`), ce qui nécessite une gestion soigneuse des états entre les évaluations d'expressions et les exécutions de commandes dans des blocs de programme.

3.2 Difficultés rencontrées

Au cours du développement de ce projet, nous avons été confrontés à plusieurs défis, principalement dus à notre manque de familiarité initiale avec les langages OCaml [1][2] et Prolog.

Toutefois, grâce à une pratique régulière tout au long du projet, notre maîtrise de ces technologies s'est nettement améliorée.

3.2.1 Correction de conflit de variable

Une difficulté technique particulière a émergé lors de l'utilisation de notre simulateur, où l'affectation de valeurs aux variables interférait avec les vecteurs, conduisant à des comportements inattendus et incorrects. Un exemple clair de ce problème a été observé lors des tests suivants :

```
CONST src (vec int) (alloc 2);
SET (nth src 0) 10;
SET (nth src 1) 20;
VAR i int;
SET i 99;
ECHO (nth src 0);
ECHO (nth src 1);
```

Le résultat incorrect obtenu était '99 20', indiquant que l'instruction `SET i 99;` modifiait par erreur la première entrée du vecteur `src`.

Solution apportée : Pour remédier à ce problème, nous avons dû restructurer notre gestion des adresses mémoire. Nous avons clarifié et séparé les espaces d'adresse pour les variables et les vecteurs, assurant une gestion plus rigoureuse et systématique des allocations mémoire :

```
let alloc mem =
  if !next_address >= max_address then
    failwith "Out of memory"
  else (
    let addr = !next_address in
    incr next_address;
    Hashtbl.add mem addr (InZ 0);
    addr
  )
```

```
let allocn sigma n =
  let rec find_space addr =
    if addr + n > max_address then failwith "Memory full"
    else if List.for_all (fun i -> not (Hashtbl.mem sigma (addr + i))) (range 0 n)
    then addr
    else find_space (addr + 1)
  in
  let start_addr = find_space !next_address in
  for i = 0 to n - 1 do
    Hashtbl.add sigma (start_addr + i) (InZ 0)  (* Initialize each cell in the block to zero *)
  done;
  next_address := start_addr + n;  (* Update next_address to ensure no overlap *)
  (start_addr, sigma)
```

Avec cette correction, le problème a été résolu, comme le démontre le résultat correct après modification : '10 20'.

En s'assurant que les allocations pour les vecteurs et les variables sont strictement séparées, nous prévenons les conflits où une variable pourrait accidentellement écraser une partie d'un vecteur ou vice-versa.

3.2.2 Boucle infini

Dans le cadre de nos ébauches de nombreux exemples APS, nous avons parfois créé, par inadvertance, des programmes comportant des boucles infinies. Ces derniers ont posé des problèmes lors de l'exécution massive de tests avec le script test_samples.sh. L'ajout d'une fonction de timeout qui interrompt l'exécution et renvoie une erreur nous aurait permis de gagner beaucoup de temps durant les phases de test.

3.3 Ressources

— **VSC Live Share :**
L'emploi de l'extension VSC Live Share [5] s'est révélé d'une grande utilité lors de nos séances de travail en TME, grandement facilitant la collaboration et le partage d'informations entre les membres de l'équipe.

— **ChatGPT pour l'Optimisation de Code en OCaml :**
Nous avons utilisé ChatGPT [4] principalement pour optimiser et raffiner notre code OCaml. Cela a entraîné une meilleure compréhension des pratiques de codage efficaces, réduisant la longueur des scripts et améliorant leur lisibilité tout en éliminant les redondances. Ce processus d'apprentissage a été crucial pour améliorer la qualité générale et la maintenabilité de notre code.

— **Git d'un autre projet APS :**
Le dépôt Git d'un projet APS similaire [3] a été consulté pour enrichir notre réflexion et non pour reproduire du code. Nous avons intégré des commentaires spécifiques dans notre code pour indiquer clairement les sections inspirées par cette ressource. Cette démarche a renforcé notre capacité à résoudre des problèmes complexes en tirant des leçons des approches d'autres développeurs.

— **Collaboration avec nos camarades de classe :**
Nos camarades, Yanis et Salim Tabellout, nous ont apporté une aide précieuse lors des séances de TME. Leur soutien a été instrumental pour surmonter divers défis techniques. Des annotations dans notre code marquent les points spécifiques où leur assistance a été bénéfique, facilitant la révision et l'apprentissage collaboratif.

Références

[1] *«Learn-OCaml»*. Érik Martin-Dorel, Yann Régis-Gianas, et Louis Gesbert. `http://ocaml-sf.org/learn-ocaml/`.

[2] *«Interpréter un petit langage impératif»*. Thibault Suzanne. `https://github.com/thizanne/imp/blob/master/tutoriel/article.org`.

[3] *«APS»*. Adel EL AMRAOUI. `https://github.com/AdelTechCrafter/APS`.

[4] *«ChatGPT»*. OpenAI. `https://chat.openai.com`.

[5] *«Use Microsoft Live Share to Collaborate with Visual Studio Code.»*. Microsoft. `https://code.visualstudio.com/learn/collaboration/live-share.`.

[6] *«ocamlyacc parse error : what token ?»*. ygrek. `https://stackoverflow.com/questions/1933101/ocamlyacc-parse-error-what-token`.

[7] *«OCaml library : Lexing»*. `https://v2.ocaml.org/api/Lexing.html`.