Rapport Projet PF & TDL

HUANG Julien
AFKER Samy

Département Sciences du Numérique - Deuxième année
2023-2024

Table des matières

Table des figures

1 Introduction

Ce projet consiste à étendre le compilateur du langage RAT pour inclure de nouvelles constructions telles que les pointeurs, les tableaux, les boucles "for" et les goto en plus des fonctionnalités déjà implémenté lors des séances de TP. Le compilateur sera écrit en OCaml en respectant les principes de la programmation fonctionnelle.

Dans ce rapport, nous allons décrire les différentes étapes de notre travail, les choix de conception que nous avons faits, les difficultés rencontrées et les solutions que nous avons trouvées.

Nous allons également expliquer les modification faite, notamment sur la structure des AST.

Introduire de nouvelles notions, comme des fonctions non utilisé en TP (MAlloc par exemple pour les tableaux et les pointeurs).

Nous avons décider de mettre en commentaire les tests qui ne fonctionnent pas (pour une question de lisibilité). Par exemple si testAffectionTableau1.rat est en commentaire cela doit surement dire qu'il y a un problème au niveau des affections.

NB : Nous n'allons pas expliquer les modifications faite dans les fichiers :

— parser.mly

— lexer.mll

On a seulement ajouter dans la syntaxe les nouvelles fonctionnalités (boucle for, goto, pointeurs, tableaux).

Nous n'allons également pas expliquer l'implémentation des fonctionnalités de base faite lors des séances de TP.

2 Types

On a ajouter deux nouveaux type défini de manière récursive :

— *Pointeur of typ* : De cette manière il est possible d'avoir des pointeurs de pointeurs de pointeurs sur des entiers.

— *Tableau of typ* : De même on peut avoir des tableaux de tableaux de pointeurs sur des tableaux d'entiers.

```
(* Interface des arbres abstraits *)
module type Ast =
sig
  type expression
  type instruction
  type fonction
  type programme
  type affectable
end
```

FIGURE 1 – Modification de l'AST

```
(* Affectables de Rat *)
type affectable =
  (* Affectation normale *)
  | Ident of string
  (* Affectation en déréferençant un pointeur *)
  | Deref of affectable
  (* Accès à un tableau en lecture ou en écriture *)
  | AccesTab of affectable * expression
```

FIGURE 2 – Les affectables

En conséquence de l'ajout de pointeurs et de tableaux on a décidé de modifier l'AST, on a ajouter le type affectable. Afin de pouvoir séparer les cas pour une variable "normale", un pointeur

et une valeur d'un tableau. Par conséquent on a également changé la structure des expressions et des instructions.

Ajout avec la grammaire (EBNF) du langage RAT étendu donnée dans le projet :

- Pour les pointeurs :
  - Null
  - New of typ
  - Adresse of string
- Pour les tableaux :
  - CreateTab of typ * expression
  - InitTab of expression list
- Pour les boucles for :
  - For of string * expression * expression * string * expression * bloc
- Pour les GoTo :
  - CallGoto of string (* goto etiq *)
  - DefGoto of string (* etiq : instruction list *)

3 Jugement de typage

Voici ce qu'on a considéré lors de la vérification des types :

- *AccesTab* : Pour accéder à la valeur d'un élément d'un tableau il faut nécessairement que l'expression soit du type entier (Tab [expr]).
- *CreateTab* : lors de la création d'un tableau il faut que l'expression soit un entier (c'est ce qui va déterminer la taille du tableau)
- *InitTab* : Pour initialiser un tableau à partir d'une liste d'expressions il faut que toutes les expressions soient du même type.
- *Affectation* : Lors d'une affectation il faut vérfier que l'expression qu'on veut affecter à une variable soit bien du même type.
- *For* : pour la boucle for, il faut vérifier que la variable de boucle soit de type entier, que la condition d'arrêt soit bien un booléen, et que l'expression d'incrémentation soit un entier.

4 Les pointeurs (totalement traité mais des problèmes lorsque l'on utilise des pointeurs de pointeur)

Dans la passe de gestion des identifiants :

- Construction assez similaire aux constructions déjà faite en TP.
- Le déréférencement se fait de façon récursive à l'aide de la fonction analyse_tds_affectable

Dans la passe de placement mémoire :

- Nous avons rencontré un problème. Quand nous exécutons les tests avec des pointeurs de pointeurs cela "casse" tous les autres tests. Les pointeurs de pointeurs sont mal gérés (peut être dans le lexer ou le parser) , elles ne passent pas les tests qu'on a implémenté (on les as mis en commentaire)

Dans la passe de génération de code :

- Nous avons choisi de représenter null par un 0.
- La création de pointeur est faite grâce à la primitive "MAlloc" de TAM.
- L'obtention de l'adresse d'une variable est réalisé grâce à la fonction loada.

5 Les tableaux (partiellement traité)

Dans la passe de gestion des identifiants (traité) :

— On vérifie lors d'une déclaration si elle a déjà été déclaré dans la TDS.

— Les tests passent.

Dans la passe de placement mémoire :

— On ne change rien

Dans la passe de génération de code (partiellement traité) :

— Code mis en commentaire, mais voici les idées de l'implémentation :

  — Lors de la l'initialisation d'un tableau à partir d'une liste d'expression, on va d'abord analyser chaque expression de cette liste (avec List.map), puis on va déterminer la taille de la liste (avec length), puis il faut déterminer le taille du type d'une expression dans la liste (on peut prendre le premier élément, comme elle a passé la passe de typage) , il faudrait ensuite load_int ces deux valeurs et les multiplier (avec call "SB" RMult) pour obtenir la taille du bloc en mémoire à allouer (avec subr "MAlloc"), enfin store toute les expression à l'adresse empilé.

  — lors de la création d'un tableau : c'est plus simple, on a accès au type directement, on peut donc directement obtenir la taille du type (avec getTaille) puis il faut également analyser l'expression (avec analyse_tam_expression) une fois les deux entiers empilé , on calcul la taille du bloc en mémoire a allouer avec (avec call "SB" RMult et subr "MAlloc")

  — Pour pouvoir accèder à la valeur d'une case d'un tableau, il faudrait utiliser jumpi ou loadl

  Une autre idée serait de stocker les informations du tableau directement les uns a la suite des autres sans utiliser les adresses.

6 Les boucles "for" (totalement traité)

Dans la passe de gestion des identifiants :

— Nous avons ajouté la variable de boucle i à la table des symboles afin que cette dernière puisse être utilisée dans le bloc de la boucle. Nous vérifions également la bonne utilisation de l'identifiant lors de l'incrémentation.
Par exemple, ceci doit être rejeté : for { int i = 0 ; (i<5) ; x = (x+1)}

Dans la passe de placement mémoire :

— Ici, nous modifions uniquement l'adresse de la variable de boucle ainsi que son registre.

Dans la passe de génération de code :

— Le code TAM pour la boucle for est quasiment identique à celui de la boucle while. La seule différence consiste à définir la variable de boucle, et de l'incrémenter à la fin du bloc.

7 Goto (totalement traité)

Cette partie est réalisable mais à cause du manque de temps on a pas pu le finaliser. Mais voici les idées : **Dans la passe de gestion des identifiants :**

— Créer une nouvelle info_ast pour les Goto qui vont stocker leur étiquette.

— Créer une TDS spéciale pour les label seulement.

— Faire un premier scan du code pour mettre tous les labelss (etiq : code) dans la TDS qu'on vient de créer. Comme dans les autres gestions des identifiants, on va chercher gobalement si le label est déjà dans la TDS ou non, si c'est le cas alors on lève une exception de double déclaration, sinon on la met dans la TDS

— Faire un deuxième scan du code, maintenant on regarde les "goto etiq" et on regarde si le label est dans la TDS.

Dans la passe de placement mémoire :

— rien de particulier à dire

Dans la passe de génération de code :

— C'est la partie la plus abordable de Goto à notre avis, il s'agit de mettre les étiquettes aux bons endroits et de faire des jumps vers l'étiquette.

8 Conclusion

Les tests ont vraiment été utile lors de ce projet, ils permettent de mieux cibler les problèmes.
Difficultés rencontrés :

— Implémentation des tableaux, on ne savait pas s'il fallait mettre le tableau dans la pile ou dans un tas avec une adresse.

— La syntaxe Tam : on avait un peu de mal à comprendre ce langage et à l'utiliser sur de nouvelles notions qu'on avait pas vu lors des séance de TP (comme les pointeurs, tableaux).

— Avec les vacances de noël et du nouvel an il est bien sur difficile de travailler en ces temps de fête ! La plus grande difficulté dans ce projet est la gestion du temps, dans une semaine où il y a 2 projets à rendre et 2 exams, il est difficile de tout finir.

Améliorations éventuelles :

— Régler les problèmes lié aux pointeurs de pointeurs

— Gestion du temps

— Ajout de plus de tests élémentaires (même si nous en avons écrit déjà pas mal), pour les Goto avec l'utilisation de fontions et voir si on a bien plusieurs TDS pour les labels.