Rapport Projet TDL

EL ALOUT Ismail
LAHMOUZ Zakaria

Département Sciences du Numérique - Deuxième année- Parcours HPCBD
2022-2023

Contents

1 Introduction **3**

2 Les types **3**
2.1 La grammaire du langage RAT 3
2.2 L'évolution de l'AST 4

3 Le jugement de typage **5**

4 Les pointeurs **6**
4.1 Passe de gestion des identifiants 6
4.2 Passe de typage 6
4.3 Passe de placement mémoire 6
4.4 Passe de génération de code 6

5 Loop à la Rust **7**
5.1 Passe de gestion des identifiants 7
5.2 Passe de typage 7
5.3 Passe de placement mémoire 7

6 Le If optionnel **7**
6.1 Passe de gestion des identifiants 7
6.2 Passe de typage 7
6.3 Passe de placement mémoire 7
6.4 Passe de génération du code 8

7 L'opérateur ternaire **8**
7.1 Passe de gestion des identifiants 8
7.2 Passe de typage 8
7.3 Passe de placement mémoire 8

8 Conclusion **8**

List of Figures

1 La grammaire de RAT 3
2 L'AST aprés l'analyse syntaxique 5

1 Introduction

L'objectif du projet est de réaliser un compilateur du langage RAT en respectant les modalités de la programmation fonctionnelle et de la traduction des langages.Le langage devient de plus en plus robuste en ajoutant de nouvelles constructions : les pointeurs, le bloc else optionnel dans la conditionnelle, la conditionnelle sous la forme d'un opérateur ternaire, et les boucles "loop" à la Rust.

2 Les types

2.1 La grammaire du langage RAT

La figure 1 représente la grammaire du langage RAT Les nouvelles règles en bleu correspondent

1. $MAIN \rightarrow PROG$
2. $PROG \rightarrow FUN\ PROG$
3. $FUN \rightarrow TYPE\ id\ (\ DP\ )\ BLOC$
4. $PROG \rightarrow\ id\ \ BLOC$
5. $BLOC \rightarrow\ \{\ IS\ \}$
6. $IS \rightarrow\ I\ \ IS$
7. $IS \rightarrow$
8. $I \rightarrow\ TYPE\ \ id\ =\ E\ ;$
9. $I \rightarrow\ A\ =\ E\ ;$
10. $I \rightarrow\ const\ \ id\ =\ entier\ ;$
11. $I \rightarrow\ print\ \ E;$
12. $I \rightarrow\ if\ \ E\ \ BLOC\ \ else\ \ BLOC$
13. $I \rightarrow\ if\ \ E\ \ BLOC$
14. $I \rightarrow\ while\ \ E\ \ BLOC$
15. $I \rightarrow return\ E;$
16. $I \rightarrow loop\ BLOC$
17. $I \rightarrow id\ :\ loop\ BLOC$
18. $I \rightarrow break;$
19. $I \rightarrow break\ id;$
20. $A \rightarrow id$
21. $A \rightarrow (*\ A)$
22. $DP \rightarrow$
23. $DP \rightarrow TYPE\ id\ DP$
24. $TYPE \rightarrow\ bool$
25. $TYPE \rightarrow\ int$
26. $TYPE \rightarrow\ rat$
27. $TYPE \rightarrow\ TYPE\ *$
28. $E \rightarrow call\ id\ (\ CP\ )$
29. $CP \rightarrow$
30. $CP \rightarrow E\ CP$
31. $E \rightarrow\ [\ E\ /\ E\ ]$
32. $E \rightarrow\ num\ \ E$
33. $E \rightarrow\ denom\ \ E$
34. ~~$E \rightarrow\ id$~~
35. $E \rightarrow\ true$
36. $E \rightarrow\ false$
37. $E \rightarrow\ entier$
38. $E \rightarrow\ (\ E\ +\ \ E\ )$
39. $E \rightarrow\ (\ E\ *\ \ E\ )$
40. $E \rightarrow\ (\ E\ =\ \ E\ )$
41. $E \rightarrow\ (\ E\ <\ \ E\ )$
42. $E \rightarrow\ (\ E\ ?\ E\ :\ E\ )$
43. $E \rightarrow A$
44. $E \rightarrow null$
45. $E \rightarrow (new\ TYPE)$
46. $E \rightarrow\ \&\ id$

Figure 1: La grammaire de RAT

au loop à la rust , les règles en rouges sont les règles de production des pointeurs, la règle en vert est de la conditionnelle sans bloc else et la règle jaune est de l'operateur ternaire.

2.2 L'évolution de l'AST

A partir de la grammaire de RAT, on a pu reconstruire l'AST de l'analyse syntaxique ci dessous:

```
module AstSyntax =
struct

(* Opérateurs unaires de Rat *)
type unaire = Numerateur | Denominateur

(* Opérateurs binaires de Rat *)
type binaire = Fraction | Plus | Mult | Equ | Inf

type affectable =
  | Ident of string
  | Deref of affectable

(* Expressions de Rat *)
type expression =
  (* Appel de fonction représenté par le nom de la fonction et la liste des paramètres réels *)
  | AppelFonction of string * expression list
  (* Accès à un identifiant représenté par son nom *)
  | Ident of affectable
  (* Booléen *)
  | Booleen of bool
  (* Entier *)
  | Entier of int
  (* Opération unaire représentée par l'opérateur et l'opérande *)
  | Unaire of unaire * expression
  (* Opération binaire représentée par l'opérateur, l'opérande gauche et l'opérande droite *)
  | Binaire of binaire * expression * expression
  |Ternaire of expression*expression*expression
  | Adresse of string
  | New of typ
  | Null
```

```
type bloc = instruction list
and instruction =
  (* Déclaration de variable représentée par son type, son nom et l'expression d'initialisation *)
  | Declaration of typ * string * expression
  (* Affectation d'une variable représentée par son nom et la nouvelle valeur affectée *)
  | Affectation of affectable * expression
  (* Déclaration d'une constante représentée par son nom et sa valeur (entier) *)
  | Constante of string * int
  (* Affichage d'une expression *)
  | Affichage of expression
  (* Conditionnelle représentée par la condition, le bloc then et le bloc else *)
  | Conditionnelle of expression * bloc * bloc
  (*Boucle TantQue représentée par la conditin d'arrêt de la boucle et le bloc d'instructions *)
  | TantQue of expression * bloc
  (* return d'une fonction *)
  | Retour of expression
  (*Boucle loop représentée par le bloc d'instruction *)
  | Loop of bloc
  (* Boucle loop représentée par son identifiant et son bloc d'instructions *)
  | LoopId of string * bloc
  (* Break *)
  | Break
  (* Break d'une boucle représentée par son nom *)
  | BreakId of string
  | Conditionnelleopt of expression * bloc
(* Structure des fonctions de Rat *)
(* type de retour - nom - liste des paramètres (association type et nom) - corps de la fonction *)
type fonction = Fonction of typ * string * (typ * string) list * bloc

(* Structure d'un programme Rat *)
(* liste de fonction - programme principal *)
type programme = Programme of fonction list * bloc
```

Figure 2: L'AST aprés l'analyse syntaxique

Aprés l'analyse syntaxique vient la phase de gestion des identifiants. Dans l'AstTds, on remplace le string (c-à-d l'identifiant) par (Tds.info_ast) et on supprime les constantes en les substituant dans la tds par empty. Ensuite,on supprime le type de l'AST aprés typage car on renvoie le type des expressions dans l'analyse des expressions. Enfin,on ajoute de l'AST placement en particulier dans le retour d'une expression la taille des paramètres et la taille du retour.

3 Le jugement de typage

Cette partie représente le jugement de typage complet des nouvelles règles de production. Voici le jugement de typage correspondant aux nouvelles règles des pointeurs :

$$\frac{}{\sigma \vdash null : undefined*}$$

$$\frac{}{\sigma \vdash \text{new } \tau : \tau*}$$

$$\frac{\text{a:}\tau*}{\sigma \vdash *a : \tau}$$

$$\frac{\sigma \vdash x : \tau}{\sigma \vdash \delta x : \tau*}$$

$$\frac{\sigma \vdash e : undefined*}{\sigma \vdash e : \tau*}$$

Les jugements de typage de la règle if optionnel

$$\frac{\sigma \vdash E : bool\ \sigma,\tau_r \vdash Bloc : void}{\sigma,\tau_r \vdash \text{if E Bloc : void,[]}}$$

Les jugements de typage de loop à la rust

$$\frac{\sigma\ ,\tau_r \vdash \text{Bloc:void}}{\sigma\ ,\tau_r \vdash \text{loop Bloc:void,[]}}$$

$$\frac{\sigma\ ,\tau_r \vdash \text{Bloc:void}}{\sigma\ ,\tau_r \vdash \text{id : loop Bloc:void,[id,string]}}$$

Les jugements de typage de l'operateur ternaire

$$\frac{\sigma \vdash \text{E1:Bool}\ \sigma \vdash E2 : \tau\ \sigma \vdash E3 : \tau}{\sigma \vdash E1?E2 : E3 : \tau}$$

4 Les pointeurs

Pour ajouter les pointeurs dans le langage RAT, on a ajouté les nouveaux terminaux suivants : "New", "&", ":".

4.1 Passe de gestion des identifiants

On remplace dans tous les éléments de l'AstTds "string" par des pointeurs vers l'information "info_ast". On ajoute dans le fichier "PasseTdsRat.ml" le traitement d'un affectable dans la fonction récursive analyse_tds_affect qui traite les affectables. Dans cette fonction, on vérifie si l'affectable est de la forme "Ident s" où s un identifiant (string); dans ce cas on garde l'ancien traitement effectué lors des tps. Sinon si l'affectable est de la forme "Deref affectable"; on renvoie analyse_tds_affect aff.

On ajoute dans le traitement des instructions le cas Adresse of id; où on cherche l'identifiant dans la table des symboles, si on trouve une info correspondante on retourne l'adresse de celle-ci.

4.2 Passe de typage

On ajoute le type Pointeur of typ dans la définition de typ. Ensuite, on définit une fonction récursive analyse_type_affec qui traite les affectables. Et on ajoute dans le traitement des expressions les cas Adresse of info, Null et New of typ; pour les expressions de la forme Adresse of info : on vérifie que l'info est associée à une variable et on retourne Adresse(info,Pointeur sur le type de la variable).

4.3 Passe de placement mémoire

On ne modifie pas le fichier "PassePlacementRat.ml" pour introduire les pointeurs. On a choisi la taille d'un pointeur (= 1) dans le fichier "type.ml".

4.4 Passe de génération de code

On ajoute le traitement des nouvelles expressions ainsi que la fonction analyse_code_affec qui traite les affectables.

5 Loop à la Rust

Les nouveaux terminaux nécéssaires pour l'implémentation de Loop à la Rust : "Loop", "Break" et "Continue".

5.1 Passe de gestion des identifiants

On remplace dans tous les éléments de l'AstTds "string" par des pointeurs vers l'information "info_ast".

On ajoute dans le traitement des instructions le cas d'une instruction Loop et le cas d'une instruction Loop avec identifiant. Dans le premier cas, on analyse le bloc associé à Loop. Dans le 2ème cas (Loop avec identifiant), on vérifie si l'identifiant n'est pas associé à une autre boucle et si c'est bien le cas on crée une nouvelle table des symbole associée à cette Loop et on ajoute l'information associée (InfoLoop) à l'identifiant dans cette table, puis on analyse le bloc de la boucle. Ici, on a crée une nouvelle information dans le type info InfoLoop(string) qu'on associe aux identifiants de Loop afin de différer entre les identifiants de variables et les identifiants de boucle et aussi pour permettre la définition d'une variable et une boucle Loop qui ont le même identifiant.

On ajoute également le cas des instructions Break et Break avec identifiant. Pour Break avec identifiant, on cherche globalement dans la table des symboles et on vérifie si l'identifiant est bien déclaré comme un identifiant de boucle (InfoLoop).

5.2 Passe de typage

On ajoute le traitement des nouvelles instructions. Pour Loop et Loop avec identifiant, on analyse le bloc associé.

5.3 Passe de placement mémoire

Aprés analyse du bloc avec la fonction analyse_PlacementBloc, on ajoute la nouvelle sortie de type AstPlacement.Loop avec une taille nulle. Le traitement est le même pour les autres instructions.

6 Le If optionnel

Le principe d'implantation du If optionnel est similaire à celui du If else classique .

6.1 Passe de gestion des identifiants

On analyse la condition de If et le bloc en s'apppuyant sur les fonctions analyse_tds_expression et analysetdsbloc.Ainsi on ajoute à la tds la nouvelle conditionnelle optionnelle qui contient les expressions et les blocs analysés.

6.2 Passe de typage

On compare le type attendu Bool et le type réel de la condition. Si ils sont compatibles alors aprés analyse du bloc on ajoute la conditionnelle dans l'ast type sinon on léve l'exception TypeInattendu.

6.3 Passe de placement mémoire

Aprés analyse du bloc avec la fonction analyserPlacementBloc, on ajoute la nouvelle conditionnelle optionnelle dans l'ast placement avec une taille nulle.

6.4 Passe de génération du code

On a défini deux étiquettes etiqFin l'étiquette du fin du programme et etiqSinon l'etiquette qui marque si la condition n'est pas vérifiée. Si la condition est vraie alors grace a JumpIf et analysercodebloc on peut analyser le bloc de la conditionnelle ,sinon on va sur = EtiqFin

7 L'opérateur ternaire

Le principe d'implantation de l'operateur Ternaire ressemble à celui du l'operateur Binaire. On a besoin d'un nouveau terminal : "?".

7.1 Passe de gestion des identifiants

On analyse les trois expressions en s'apppuyant sur la fonction analyse_tds_expression. Ainsi on ajoute à la tds la nouvelle expression ternaire qui contient les trois expressions analysées.

7.2 Passe de typage

On commence par analyser les trois expressions en renvoyant le type de chaque expression.Ensuite, on compare le type de l'expreesion E2 et le type de l'expression de E3 . On verifie aussi que le type de l'expression E1 est booléen. Si les conditions précedentes sont validées alors on ajoute l'expression ternaire qui contient les nouvelles expressions analysées dans l'ast type . Sinon on renvoie l'exception typeInattendu.

7.3 Passe de placement mémoire

Il n' y a pas de modification dans la passe du placement mémoire puisque il s'agit d'une expression

8 Conclusion

Enfin, le projet était très riche et permet d'approfondir les connaissances aquises dans les cours et les TDs.Cependant,il est long et l'implantation de certaines passes à savoir la passe de géneration du code est délicate . Les séeances des travaux dirigés nous ont permis de résoudre les problèmes complexes et d'avancer dans le projet. Nous remercions chaleureusement toutes les personnes qui ont donné des remarques précieuses pour réussir ce projet.