TRADUCTION DES LANGAGES

Projet

Parcours : HPCBD

Chaimae CHELLAF EL HAMMOUD – Younes ALAHYANE

2021-2022

Sommaire :

1- Introduction :

Lors des séances des TPs de traduction des langages, on a commencé l'implantation du compilateur du langage RAT, en se basant essentiellement sur quatre passes : la gestion des identifiants, le typage, le placement mémoire, et finalement la génération du code. Ceci après l'analyse lexicale et syntaxique effectuées sur le code du langage source. Cependant, plusieurs outils n'ont pas été traités lors des séances de TPs, comme les pointeurs, l'opérateur d'assignation d'addition, les types nommés et les enregistrements. L'objectif de ce projet est donc d'ajouter ces outils dans notre compilateur afin de permettre au programmeur la manipulation de ces outils qu'on juge nécessaires dans un compilateur.

2- Types et AST :

➢ Pointeurs :

Les modifications effectuées sur AST après l'ajout des pointeurs sont les suivantes :

- Ajout d'un nouveau type :
  affectable = | Ident of string
  | Deref of affectable
- Ajout d'une nouvelle instruction :
  Affectation of affectable * expression
- Ajout de nouvelles expressions et mise à jour d'une expression :
  | Affectable of affectable
  | New of typ
  | Adresse of string
  | Null

Et l'ajout d'un nouveau cas de type dans le fichier type.ml et type.mli :
Pointeur of typ

➢ Opérateur d'assignation d'addition :

La seule modification effectuée sur AST après l'ajout de l'opérateur (+=) est la suivante :

- Ajout d'une nouvelle instruction : **AssignAdd of affectable * expression**

➢ Types nommés :

Les modifications effectuées sur AST après l'ajout des types nommés sont les suivantes :

- Mise à jour de prog, on change sa structure pour qu'il prenne en compte les types nommés dans sa définition :
  Programme of (string*typ)list * (fonction list) * bloc
- Ajout d'une nouvelle instruction :
  DeclTypeNomme of affectable * expression

Et l'ajout d'un nouveau cas de type dans le fichier type.ml et type.mli :
TNomme of string

➢ Enregistrements :

Les modifications effectuées sur AST après l'ajout des enregistrements sont les suivantes :

- Ajout d'une nouvelle affectation : **AccesChamp of affectable * string**
- Ajout d'une nouvelle expression : **StructAffect of expression list**

Et l'ajout d'un nouveau cas de type dans le fichier type.ml et type.mli :
Struct of (typ*string) list

3- Jugement de typage :

- **Pointeurs :**

$$\frac{x \in \sigma \quad \sigma(x) = t}{\sigma \vdash x : t}$$

$$\frac{x \in \sigma \quad \sigma(x) = t}{\sigma \vdash \&x : \text{Pointeur } t}$$

$$\frac{}{\sigma \vdash (\text{new } t) : \text{Pointeur } t}$$

$$\frac{\sigma \vdash a : \text{Pointeur } t}{\sigma \vdash *a : t}$$

$$\frac{}{\sigma \vdash \text{null} : \text{Pointeur void}}$$

- **Opérateur d'assignation d'addition :**

$$\frac{\sigma \vdash id : int \quad \sigma \vdash e : int}{\sigma \vdash id+=e : int}$$

$$\frac{\sigma \vdash id : rat \quad \sigma \vdash e : rat}{\sigma \vdash id+=e : rat}$$

$$\frac{\sigma \vdash a : Pointeur\ int \quad \sigma \vdash e : int}{\sigma \vdash *a+=e : int}$$

$$\frac{\sigma \vdash a : Pointeur\ rat \quad \sigma \vdash e : rat}{\sigma \vdash *a+=e : rat}$$

4-Les pointeurs :

Afin que notre compilateur puisse traiter les pointeurs, on a tout d'abord modifié lexer.mll en ajoutant les tokens comme décrit dans la grammaire : **new**, **null**, et l'opérateur **&.** Nous avons ensuite modifié le parser.mly en y ajoutant les règles de production qui permettent la définition et la modification des pointeurs. Puis on a ajouté le type Pointeur of typ dans le fichier type.mli et type.ml, et effectué les changements listés dans le paragraphe précédent dans le fichier ast.ml.

Pour le passe de gestion des identifiants, la première chose à faire était la mise à jour de l'AstTds dans le fichier ast.ml en changeant les string en Tds.info_ast. Ensuite, on a ajouté une méthode analyse_tds_affectable qui se charge d'analyser le type affectable, et on a ajouté la nouvelle instruction Affectation(affectable, expression) dans la méthode analyse_tds_instruction, puis modifié et ajouté analyse_tds_expression pour permettre d'analyser les nouvelles expressions ajoutées au langage (Adresse of string, New of typ, Null, et Affectable of affectable).

Dans le passe du typage, on a suivi la même démarche que l'étape précédente, en ajoutant ainsi une méthode analyse_type_affectable, et en effectuant les traitements sur les nouvelles expressions et instructions ajoutées. Il est à noter qu'on a modifié la méthode est_compatible et est_compatible_list dans le fichier type.ml afin de prendre en compte des pointeurs dans les différentes analyses de typage.

Dans le passe de placement mémoire, on n'a effectué aucun changement, sauf l'ajout de la taille du pointeur (qui est égale à 1) dans la méthode getTaille définie dans type.ml.

Dans le passe génération du code, nous avons défini la méthode analyse_code_affectable permettant l'ajout du code TAM pour ce type, tout en modifiant les autres méthodes d'analyse des expressions et instructions afin de prendre en compte de type pointeur. Les instructions TAM ajoutées sont : SUBR MAlloc, LOADA d[r], LOADI d[r], STORI (t).

5-L'opérateur d'assignation d'addition :

L'ajout de l'opération d'assignation d'addition n'a pas été compliqué puisqu'il suffisait d'ajouter une seule instruction et de l'analyser dans tous les passes. On a donc modifié le parser.mly pour prendre en compte toutes les instructions de la forme I -> A += E. Puis on a passé au passe Gestion des identifiants, où on a modifié analyse_tds_instruction en ajoutant la nouvelle instruction AssignAdd(a,e) sur lequelle on a appliqué (analyse_tds_affectable a), puis (analyse_tds_expression e). On a fait le même traitement dans le passe typage.

Au niveau du placement mémoire, on a rien changé puisque c'est une affectation aussi.

Quant au passe génération de code, au niveau de l'instruction AssignAdd(a,e), on a analysé l'affectable a et l'expression e par les méthodes analyse_code_affectable et analyse_code_expression, en faisant un SUBR IAdd ou CALL (SB) RAdd selon le type de l'expression e (Int ou Rat), puis on stocke cette somme dans le placement d[r] de l'affectable a.

6-Les types nommés :

Pour permettre à notre compilateur de traiter les types nommés, on a commencé la modification du lexer.mll en y ajoutant le token **typedef**, puis les identifiants de type (**tid**) qui commencent par une lettre majuscule. Nous avons ensuite modifié le parser.mly en y ajoutant les règles de production qui permettent la manipulation des types nommés (ajout de td, ajout d'une nouvelle instruction pour la définition de ce type, modification de la structure du programme). En ajoutant le nouveau type TNomme of string dans le fichier type.mli et type.ml, notre compilateur reconnait maintenant les types nommés.

Dans le passe de gestion des identifiants, on a ajouté la nouvelle instruction DeclTypeNomme(s,t) dans analyse_tds_instruction, qui renvoie

DeclTypeNomme(info_ast, t). On a aussi analysé l'argument (string*typ) list de la méthode analyser (pour le programme principal), en appliquant le même principe du remplacement de string par l'info_ast en parcourant toute la liste.

Dans le passe typage, on a fait un travail similaire, sauf qu'on a ajouté une méthode chercherTypeTNomme (Tds,typ) permettant l'obtention du type effectif du type nommé, en s'appuyant sur une Tds qui contient les identifiants des types nommés déjà déclarés. Par exemple si on a défini un type Int2 qui est égal à Int, alors cette fonction renvoie le type Int. Cette méthode nous a permis d'utiliser les méthodes est_compatible et est_compatible_list dans le passe typage, pour faire ainsi une bonne analyse des types dans un programme.

Les passes Placement mémoire et génération du code restent inchangées.

7-Les enregistrements :

Pour inclure le type enregistrement dans le langage RAT, on a ajouté les tokens **struct** et **pt** (.) dans le lexer.mll, puis on a modifié le parser.mly en ajoutant un nouveau affectable AccesChamp of affectable*string, et une nouvelle expression StructAffect of expression list. Le fichier type.mli et type.ml ont été également modifié (nouveau type Struct of (typ*string) list).

Cependant, on n'a pu traiter les enregistrements dans tous les passes, donc on a choisi de s'arrêter au passe de gestion des identifiants, où on a ajouté la nouvelle instruction AccesChamp et la nouvelle expression StructAffect. Les tests des enregistrements réussissent après les modifications apportées dans ce passe.

8-Conclusion :

Pour conclure, ce projet était très intéressant vu qu'on a pu implanter nous même un compilateur du langage RAT assez riche, et qui permet non seulement de traiter des constructions basiques, mais aussi de comprendre les pointeurs, l'opérateur d'assignation d'addition, les types nommés, et finalement les enregistrements. C'était un projet très bien construit et encadré, qu'on a beaucoup appris, tout en ayant une nouvelle vision sur les compilateurs et comment ça se fait la traduction des langages.

La réalisation de ce projet n'était pas assez simple. En effet, on a eu beaucoup de difficultés, la première est qu'il y a beaucoup de fichiers et beaucoup de modules,

interfaces et types. On a donc essayé de bien comprendre ce qui est déjà écrit dans chaque fichier, et dans chaque passe, pour pouvoir avancer aisément. Ensuite, chaque modification apportée sur une partie du code pouvait impacter sur tout le reste du projet, donc on devait être prudents à chaque fois qu'on voulait ajouter une nouvelle fonctionnalité. Cependant, l'implantation du type enregistrement était une étape assez difficile, et par contrainte du temps, et vu qu'il y avait pas mal de modifications à faire, on s'est arrêté après le passe de gestion des identifiants.

Pour conclure, nous sommes satisfaits de ce qu'on a arrivé à faire, et très heureux de pouvoir implanter un compilateur du langage RAT, en apprenant tout au long du projet de nouvelles choses sur la structuration, la construction et le fonctionnement d'un tel programme.