Rapport Projet TDL

Bilal Azdad

2A - Département Sciences du numérique - Filière : Image et Multimédia
2023-2024

Contents

List of Figures

1 Introduction

Le but de ce projet était de réaliser un compilateur du langage RAT(simplifié) en se concentrant sur les phases essentielles de la compilation soit les 4 passes, ce projet complète les tps qui ont introduit les méthodes à partir des quelles on réalise un compilateur fonctionnel, grâce à cela nous proposons 4 nouvelles fonctionnalités prises en compte dans le compilateur : **goto (ou saut)**, **les tableaux**, **la boucles "for"** et **les pointeurs(introduits en tp)**. Le compilateur sera construit en Ocaml. Nous détaillerons la manière dont nous avons implanté ces fonctionnalité Passe par Passe pour les 4 fonctionnalités.

2 Ast

Le lexer et le parser ont été complété conformément à la grammaire.

2.1 Pointeur

Les pointeurs n'étaient pas forcément simple à implanter mais ayant vu le principe en TD nous sommes parti sur de bonnes bases :
Nous introduisons un nouveau type :

```
Pointeur of type
```

En même temps nous avons complété les fonctions du fichier **"type.ml"** en incluant le pointeur.

Conformément à la grammaire, notre modèle Ast a été complété par l'ajout de :

```
type affectable =
            | Ident of string
            | Deref of affectable
```

Deref est la valeur pointée.

```
expression =
            | Affectable of affectable
            | New of typ
            | Null
            | Adresse of string

instruction =
            | Affectation of affectable * expression
```

Ajout des règles dans le parser et des tokens associés dans le lexer.
On remarquera qu'un identifiant n'est plus une expression mais un affectable.

2.2 Boucle For

Nous rappelons la règle de grammaire :

```
|for(int id = E; E ; id = E) BLOC
```

Après révision il était possible de considérer **int id** = **E** comme une instruction, cependant nous avons opté pour la soltution suivante simple et fonctionnelle (très littérale):

```
| BoucleFor of typ * string * expression * expression * string * expression * bloc
```

Les identifiants sont des string, les expressions reste des expressions, et 'int' est bien sûr un type.

2.3 Tableau

Conformément à la grammaire nous avons complété l'Ast et notamment les non terminaux **affectable** et **expression** :

```
type affectable =
              | Ident of string
              | Deref of affectable
              | AccesTableau of affectable * expression


type expression =
              | CreationTableau of typ * expression
              | InitialisationTableau of expression list
```

Nous avons également défini un nouveau type dans le fichier **type.ml**, nous avons complété les fonctions de ce module comme est-compatible ou getTaille.

Le parser et le lexer ont été complété conformément à la grammaire.

2.4 Goto

Conformément à la grammaire nous avons complété le non terminal instruction pour **astSyntax** :

```
 instruction =
                | Goto of string
                | Label of string
```

String est remplacé par Tds.info_ast pour les autre Ast, comment partout ailleurs lorsqu'il y a string

3 Gestion des identifiants

Dans les Ast suivantes les **string** sont remplacés par les **info_ast**

3.1 Pointeur

L'ajout du type **affectable** nous a mené à analyser les affectables de manière récursive :

```
    let rec analyse_tds_affectable tds tds_Label modif aff  =
    .......
    | AstSyntax.Ident id ->
    | AstSyntax.Deref aff ->
    .......
```

Et l'ajout des expressions associées aux pointeurs :

```
    .......
    | AstSyntax.Affectable a
    | AstSyntax.New t
    | AstSyntax.Null
    | AstSyntax.Adresse id
    .......
```

New et Null ne sont pas traités ici, on les passe tels quels dans l'ast suivante, l'affectable sera analysé grâce à la méthode précédemment implanté, pour l'adresse on cherchera dans la TDS afin d'obtenir l'info_ast associée, une exception sera levée si elle n'est pas trouvée.

```
    .......
    | AstSyntax.Affectable a
    | AstSyntax.New t
    | AstSyntax.Null
    | AstSyntax.Adresse id
    .......
```

3.2 Boucle For

Pour la boucle for, il s'agit de vérifier que l'identifiant de l'iterateur est unique cas d'exemple incorrecte :

```
for (int x=0; (y<5); y=(y+1)) {}
```

Cela lévera l'exception IterateurForNonDeclare.

De la même manière que pour une fonction, la boucle for dispose d'une tds locale, où l'identifiant est ajouté, une modification de la variable localement n'a pas d'incidence en dehors du bloc.

Les expressions de la boucle for sont classiquement analysées, ainsi que les instructions par l'analyse du bloc .

3.3 Tableau

Pour les tableaux on analyse simlplement les expressions associées:

```
    .......
  | AstSyntax.CreationTableau (t,e) ->
  | AstSyntax.InitialisationTableau el ->
    .......
```

3.4 Goto

Le comportement de la fonctionnalité Goto est très erratique et permet littéralement de sauter n'importe où un label est défini. Le point clé était de ne pas se murer à travailler avec une seule TDS, les identifiants des labels pouvait être des identifiants déjà déclaré pour autant nous devions pas lever l'exception **DoubleDeclaration** d'où l'idée de créer une TDS_Label mère dans la fonction analyser pour ne pas confondre les identifiants. Nous avons également créer un nouveau type d'information pour Goto **InfoGoto** pour plus de clarté.

```
let analyser (AstSyntax.Programme (fonctions,prog)) =
            .........
            let tdsgoto = creerTDSMere() in
            .........
```

4 Jugement de Typage

4.1 Pointeur

Les analyse d'expressions, d'instructions et d'affectable de la foont été faite selon les jugements de typage suivants :

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

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

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

$$\frac{\sigma \vdash \text{A} : t, \sigma \vdash \text{E} : t}{\sigma, t_r \vdash \text{A=E} : \text{Void}, []}$$

$$\frac{}{\sigma \vdash \text{null} : \text{Pointeur}(Undefined)}$$

4.2 Boucle For

L'analyse de l'intruction BoucleFor s'est basée sur les jugements de typages suivants:

$$\frac{\sigma \vdash \text{BLOC} : \text{Void} \ , \ \sigma \vdash TYPE : \text{Int} \ , \ \sigma \vdash E_1 : \text{Int} \ , \ \sigma :: (id, Int) \vdash E_2 : Bool \ , \ \sigma :: (id, Int) \vdash E_3 : Int}{\sigma \vdash \text{for (TYPE } id = E_1 \ ; \ E_2 \ ; \ id = E_3) \text{ BLOC} : \text{Void}, []}$$

4.3 Tableau

L'analyse des expression des tableaux s'est basée sur les jugements de typages suivants:

$$\frac{\sigma \vdash \ TYPE \ : \tau \ , \ \sigma \vdash E : Int}{\sigma \vdash \ new \ TYPE \ [E] \ : \ Tableau \ (\tau)}$$

$$\frac{\sigma \vdash A\text{: } Tableau \ (\tau) \ , \ \sigma \vdash E : Int}{\sigma \vdash (\ A \ [E] \ ) : \tau}$$

4.4 Goto

L'analyse du typage de goto est conforme aux jugements de typages suivants :

$$\frac{\sigma \vdash \text{label} : \ \text{Undefined}}{\sigma \vdash (\text{label} :) : \ \text{Void} \ , []}$$

$$\frac{\sigma \vdash \text{label} : \ \text{Undefined}}{\sigma \vdash \text{goto label} : \ \text{Void} \ , []}$$

5 Placement

5.1 Pointeur

Lors de la passe de placement on récupère seulement la taille du Pointeur, avec une fonction **getTailleTDS** qui récupère la taille d'un type depuis l'**info_ast**.

5.2 Boucle for

Pour la boucle For, l'itérateur a été positionné au préalable avant d'entamer l'analyse du bloc, en considérant le contexte augmenté de deplacement + 1 (qui correspond à la taille d'un entier)..

5.3 Tableau

Pour les tableaux, nous avons simplement attribuer la taille d'un tableau à 1 dans **getTaille** du fichier **type.ml**

5.4 Goto

Pour la fonctionnalité goto, le label est considérer comme un **infoVar** bien que nommé **InfoGoto** ils ont les mêmes paramètres, on change son placement en conséquence.

6 Tam

6.1 Pointeur

Le traitement des **pointeurs** comprend le déréférencement pour accéder ou modifier la valeur pointée. La création de nouveaux pointeurs utilise Malloc pour l'allocation de mémoire. Les pointeurs nulls sont représentés par la valeur -1, et l'obtention de l'adresse d'une variable est effectuée en utilisant loada.

6.2 Boucle for

La **boucle for** est implémentée en générant des étiquettes pour le début et la fin de la boucle, avec une logique de saut conditionnel basée sur la condition de la boucle. Le code inclut l'initialisation, la condition, l'incrémentation, et le corps de la boucle, avec des sauts pour contrôler le flux d'exécution, le traitement est basé sur celui de **TantQue**.

6.3 Tableau

La gestion des **tableaux** en TAM implique des calculs d'adresses pour l'accès et la modification des éléments. Pour accéder à un élément, l'indice est multiplié par la taille de l'élément et ajouté à l'adresse de base du tableau. La création de tableaux utilise Malloc pour allouer de la mémoire, et l'initialisation de tableaux suit une logique similaire avec un stockage en mémoire supplémentaire.

6.4 GOTO

Le **goto** est géré en générant un saut inconditionnel vers une étiquette spécifiée, tandis que les étiquettes (Label) sont utilisées pour marquer des positions dans le code. La correspondance entre les noms des étiquettes et les positions du code est assurée, permettant des sauts directs à ces points.

7 Tests

Les tests écrits permettent de s'assurer d'un fonctionnement nominal du langage, nous ne prétendons pas que notre jeu de test est complet mais qu'il fait l'affaire pour une utilisation normale, simple. Les tests du projet fonctionnent, pour le reste nos test se focalisent principalement sur les tests n'incluant pas de fonctions les tests se trouvent dans les fichier **sans_fonction**, environ 3-4 tests par fonctionnalités pour les passes **gestion identifiant**, **type** et **tam**. Bien sûr nous avons écrit le test complet en fin de sujet qui réunit toutes les fonctionnalités avec fonction.

8 Lisibilité du rendu

Lors des différents tests, beaucoup de pépins sont arrivés et trouver les erreurs était parfois laborieux lorsque le code n'est pas clair, ce processus de test a permi de rédiger un code clair sans failwith"", ceux ci ont été remplacé par des exceptions.

9 Conclusion

Le projet était intéressant, la compilation est moins complexe que ce que je pensais en pratique. Il m'a permis de saisir la structure et le processus de compilation en détail. Je me doute que les compilateurs d'autres langage sont bien plus complexes et aborde des notions très technique mais ce projet reste une belle ébauche. Je suis maintenant capable de coder un compilateur (en partie).