TELECOM Nancy

Projet Compilation

Thibault Boisseau
Nathan Cornélie
Noé-Laurent Laurent
Maxime Wirth

Responsable de module :
Suzanne Collin
Sébastien Da-Silva

Table des matières

Chapitre 1

Introduction

1.1 Contexte

Le projet présenté dans ce rapport a été réalisé dans le cadre du module PCL1 (Projet de Compilation des Langages 1) afin d'appliquer dans un cadre concret les connaissances acquises en début d'année au travers du cours de traduction des langages. Ce projet est un module à part entière de la seconde année du cycle ingénieur sous statut étudiant du cycle ingénieur de TELECOM Nancy.

Notre groupe étant constitué d'élèves poursuivant leurs études dans les approfondissement ISS et IL, nous poursuiverons donc ce projet au second semestre.

Pour le premier semestre, l'objectif est de réaliser une grammaire du langage Tiger légèrement modifié, puis d'en faire l'analyse afin de construire la table des symboles et de rédiger les contrôles sémantiques permettant dans un second temps, au deuxième semestre, de construire un compilateur associé à ce langage.

1.2 Langage Tiger

Le compilateur que nous devons créer est celui du langage Tiger défini dans le livre d'Andrew Apple, *Modern Compiler Implementation in Java*. Notre grammaire a pris forme grâce au manuel de références fourni par nos encadrants, et toutes les spécificités du langage y sont répertoriées.

1.3 Organisation du document

Le rapport est composé d'une introduction (chapitre actuel), de 5 chapitres ainsi que d'une conclusion suivie d'une annexe.
Le chapitre 2 présentera notre table des symboles.
Le chapitre 3 présentera les contrôles sémantiques.
Le chaitre 4 présentera quelques exemples utilisés pour nos tests.
Le chapitre 5 présentera notre gestion de projet.
La conclusion viendra apporter un bilan tant sur le point personnel que global.

1.4 Précisions légales

L'ensemble des données utilisées ont été inventées pour la réalisation de notre grammaire et son analyse, toute ressemblance avec des personnes existantes ou ayant existées est fortuite.
Ce projet n'est pas destiné à un usage commercial et est à caractère strictement scolaire, ainsi, la réutilisation de code qui n'est pas libre de droit nous est possible en accord avec :

- Code civil : articles 7 à 15, article 9 : respect de la vie privée
- Code pénal : articles 226-1 à 226-7 : atteinte à la vie privée
- Code de procédure civil : articles 484 à 492-1 : procédure de référé
- Loi n78-17 du 6 janvier 1978 : Informatique et libertés, Article 38

Toutes les réutilisations de code sont listées ci dessous :

- https ://github.com/antlr/symtab.

Chapitre 2

Table des symboles

2.1 Introduction

La construction de la table des symboles est une partie importante d'un compilateur. Elle permet l'identification de variables, et c'est grâce à elle que beaucoup de règles du langage peuvent être appliquées. Cette partie détaille comment nous l'avons construite.

2.2 Idée de base

Nous avons, pour implémenter notre table des syboles, utilisé le ParseTreeWalker de ANTLR. Le ParseTreeWalker est un objet qui va parcourir un arbre et effectuer des actions à chaque entrée et sortie de noeuds intérieurs, donc de règles de notre grammaire. Nous demandons aussi à ANTLR de générer un Listener, qui va réagir aux appels du ParseTreeWalker.

La table des symboles sera donc contruite au fur et à mesure du parcours de l'arbre de dérivation. Cette structure était avantageuse pour, par exemple, gérer les entrées et sorties de blocs que l'on peut faire à l'entrée et à la sortie d'une règle. Nous avons aussi géré les contrôles sémantiques parallèlement à la création de la table de symboles.

2.3 Implémentation

L'objet qui va stocker toutes les données de la tables des symboles est un objet de classe tableDeSymboles. Nous avons créé une classe BasicBloc contenant la logique nécéssaire à la gestion des blocs, et nous avons d'autres classes pour gérer les différents types de blocs présents (les blocs des boucle for, while, les fonctions...), qui héritent de cette classe. Nous avons aussi une interface Symbol qui sera implémentée par les classes définissant les différents symboles. Et finalement nous avons une classe BasicType qui va être héritée par toutes les classes définissant les types du langage.

Le Listener d'ANTLR contient une référence à un objet de type tableDeSymboles, qui est généré à sa construction. La table des symboles contient une référence à un bloc qui contient les fonctions predéfinies, une table de hashage avec les types créés et une liste de String contenant le nom des types acceptés. Aussi, elle contient une référence au bloc global, au bloc courant et un ArrayList<String> contenant les erreurs qui seront lancées lors du parcours.

2.4 Gestion des symboles

Ce sont les objets hériant de la classe BasicBloc qui vont faire office de tables de symboles pour les différents blocs. Ils contiennent en outre une référence à une Map<String, Symbol> contenant les noms ainsi que les symboles. Ils ont aussi une référence au bloc qui les englobe, ce qui permet la navigation entre les blocs et la recherche de symboles.

Chapitre 3

Contrôles sémantiques

3.1 Introduction

Les contrôles sémantiques sont des contrôles effectués pendant la compilation et l'exécution du code, vérifiant que les règles de construction du langage sont bien respectées. Il nous est demandé ici d'implémenter les contrôles sémantiques statiques pertinents pour le langage Tiger, c'est-à-dire ceux faits lors de la compilation.

Nous les avons donc implémenté pour qu'ils se produisent pendant le parcours de l'arbre par le ParseTreeWalker. En même temps que nous créons les symboles dans la table des symboles, on vérifie que le code respecte bien les règles du langage Tiger. Quand une erreur est rencontrée, on ajoute un message dans un ArrayList<String> qui va ensuite être écrit dans le terminal à la fin du parcours, avec la table des symboles.

3.2 Liste des contrôles implémentés

Voici la liste des contrôles sémantiques que nous avons implémenté :
— Division par 0
— Les arguments des opérateurs +, -, * et / doivent être de type entier
— Les arguments des opérations de combinaison doivent êtres compatibles et être des entiers ou des chaînes de caractères.
— A le déclaration d'un array, la taille spécifiée doit être un entier
— A l'affectation d'une valeur à un élément d'un array, l'indice doit être entier
— Un symbole utilisé dans une expression doit avoir été défini avant
— A l'affectation d'une valeur à une variable, les types doivent correspondre
— A la déclaration d'une variable, aucune fonction doit avoir le même nom, et inversement.
— Les indice de début et de fin d'une boucle for doivent être des entiers
— Une fonction utilisé doit être définie auparavant
— Le type de retour doit correspondre à l'utilisation
— Une fonction ne peut pas avoir 2 champs de même nom
— Les paramètre d'entrée doivent être définis auparavant
— Pour un appel de fonction, les arguments doivent correspondre à la définition
— A la création d'un record, le type doit être un type de record
— Un record ne doit pas avoir 2 champs avec le même nom
— Le record doit être appelé correctement en définissant les champs

Chapitre 4

Fichiers de test

4.1 sujet.exp

Nous avons repris le code inclus dans le sujet pour tester nos fonctions. Nous modifions quelques morceaux de code un par un pour tester nos contrôles sémantiques.

```
    let
var N := 8
type intArray = array of int
var row := intArray [ N ] of 0
var col := intArray [ N ] of 0
var diag1 := intArray [ N+N-1 ] of 0
var diag2 := intArray [ N+N-1 ] of 0

function printboard()  =
(
        for i := 0 to N-1
            do
                (for j := 0 to N-1
                    do
                        print(if col[i]:=j then "O" else ".");
                        print("\n")
                );
print("\n")
)

function try(c:int) =
if c=N
    then printboard()
    else
    for r := 0 to N-1
        do
        if row[r]=0 & diag1[r+c]=0 & diag2[r+7-c]=0
            then (
                row[r] := 1; diag1[r+c] := 1;
                diag2[r+7-c] := 1; col[c] := r;
                try(c+1);
                row[r] := 0; diag1[r+c] := 0;
                diag2[r+7-c] := 0
            )
in
    try(0) end
```

4.2 declaration.exp

Nous utilisions ce fichier de test pour tester plus précisément nos contrôles sémantiques basés sur les arrays.

```
    let
  type rec = { val : int }
  type rec_arr = array of rec
  var  table := rec_arr[2] of rec { val = 42 }
in
  table[0].val := 51
end
```

4.3 bad.exp

Exemple d'un fichier de test lançant des erreurs. Il y a ici plusieurs erreurs de définitions, ou d'appel de fonction, par exemple.

```
a[c] := 2;
b := 0;

if (a) {
    b=1;
}
else {
    b=2;
}

print b ;
```

4.4 somme.exp

Cet exemple permet de tester toutes les erreurs sémantiques concernant les opérateurs binaires.

```
let
  var N := 2
  var  s := "table"
  type intArray = array of int
  var row := intArray [ N ] of 0
  type stringArray = array of string
  var rowp := stringArray [ N ] of "table"
in
   1 = row[N] & row[2];
   N + 1 = row[2];
    N + 1 / row[2];
    N + 1 / row[2] + 1;
    N + 1 / row[2] + 1 * 2;
    N + 1 / row[2] + 1 * 2 - 3;
    N + 1 / row + 1;
    s <= "table";
    s < "table";
    s >= "table";
    s > "table";
    s = "table";
    s <> "table";
    s < "table" & s < "table";
    s < "table" & 1 + row;
    s >= "table" & 1 + row[2];
    s < "table" | 1 + row[2] * 2;
    s < "table" | 1 + row[N] * 2 - 3;
    s = s;
    s <= s;
    s < row | 1 + row[N];
    1 > s;
    1 > s | 1;
    s * 1;
    "rerere" > "a" | 2;
    row[2];
    row[2] * row;
    row[2] * row[N];
    row[2] * row[N] | 1;
    rowp[2] < "table";
    rowp[2] <= "table";
    rowp[2] > "table";
    rowp[2] >= "table";
    rowp[2] = "table";
    rowp[2] <> "table";
    rowp[2] < s;
    s <= rowp[2];
    row = row;
    row <> row;
    5 | row = row;
    row[4] + 1;
    row[4] + 1 * 2;
```

```
    row[4] < rowp[0];
    row[4] > rowp[0];
    row[4] <= rowp[0];
    row[4] >= rowp[0];
    row[4] = rowp[0];
    row[4] <> rowp[0];
    rowp[1] < row[0];
    rowp < "pois";
    rowp[1] > "pois";
    row / 2
end
```

Chapitre 5

Gestion de projet

5.1 Introduction

Voici les différents outils de gestion de projet que nous avons utilisé pour ce projet.

5.2 Explications

Nous avons beaucoup travaillé en groupe pendant ce projet. Nous travaillions tous les 4 sur des points différents du projet, souvent en équipes de 2, mais nous pouvions échanger et collaborer pour rendre le tout cohérent. Nous faisions périodiquement de petites réunions, souvent par Discord, pour voir où en était le projet et quelle était la marche à suivre. Nous les faisions surtout lorsque que des membres du groupe avaient travaillé de leur côté.

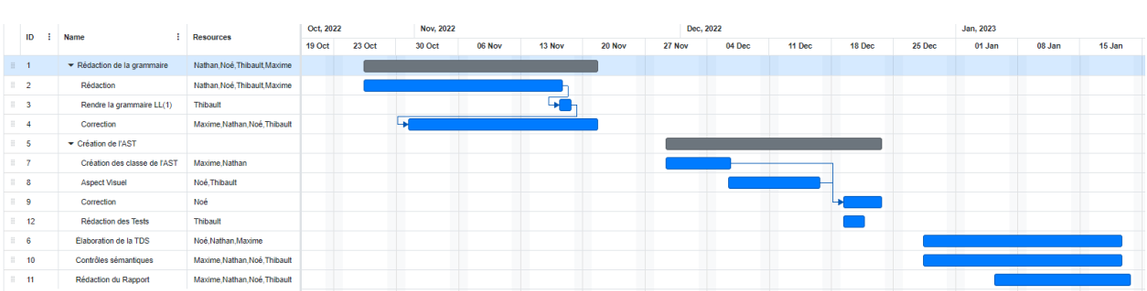
5.3 Matrice SWOT

FIGURE 5.1 – Diagramme de séquence

	Positif	**Négatif**
Interne	Bonne entente globale de l'équipe Langage Java déjà bien connu	Chacun avait des façons différentes de travailler, il a fallu mettre en commun
Externe	Tirer parti d'une correction pour l'AST et de documentation pour la TdS	Vacances scolaires, plus difficile de se réunir

5.4 Diagramme de Gantt

FIGURE 5.2 – Diagramme de séquence



Chapitre 6

Bilan

6.1 Bilan du projet par membre

6.1.1 Thibault Boisseau

Points positifs	- Le groupe de projet s'est jusque-là très bien entendu. Grâce aux outils utilisés notamment live Share, nous avons pu mener plusieurs séances efficacement tous ensemble, en nous entraidant, et, en répartissant le travail entre chacun des membres.
Difficultés rencontrées	- Les parties les plus difficiles étaient sans aucun doute la création de la table des symboles et la création de l'arbre abstrait. La principale raison était qu'il était nécessaire de bien comprendre, sans ambiguïté, les tds associés.
Expérience personnelle	- Bonne expérience de mise en application des connaissances obtenues du premier semestre.
Axes d'amélioration	- Pour la seconde partie du projet, il faudrait être plus rigoureux vis-à-vis de la gestion de projet. Les séances se déroulaient toujours ensemble, donc nous faisions rapidement le bilan du travail fait et à faire puis nous nous adaptions en fonction des besoins.

6.1.2 Nathan Cornélie

Points positifs	-Bonne entente dans le groupes qui menait à de bonnes et efficaces sessions de travail. La réalisation de la grammaire et de l'AST s'est bien passée car nous étions aidés de la correction des Td .
Difficultés rencontrées	-Ce projet était particulièrement compliqué compte tenu de la quantité de travail qui était demandée et l'organisation que l'on devait avoir mêlé en période de partiels.
Expérience personnelle	-Occasion de mettre en application tout ce que l'on a vu en cours de compilation.
Axes d'amélioration	- En points d'amélioration je dirais qu'il faudrait que l'on soit plus rigoureux sur la gestion de projets dans la suite du projet et que je sois plus régulier sur mes temps de travail.

6.1.3 Noé-Laurent Laurent

Points positifs	- Bonne communication dans le groupe. Chacun à su travailler quand il fallait.
Difficultés rencontrées	- Découverte totale du processus de création d'un AST et d'une table des symboles. Le fait que chaque personne ait travaillé de son coté à certains moments rend la reprise du travail plus difficile.
Expérience personnelle	- Cette première partie du projet m'a permis de découvrir de nouvelles façons de coder et de travailler en groupe. J'ai hâte de continuer le projet en PCL2.
Axes d'amélioration	- La gestion de projet aurait dû être suivi plus rigoureusement. Pour ma part, un travail plus régulier aurait sûrement été plus efficace que des grosses sessions de travail ponctuelles.

6.1.4 Maxime Wirth

Points positifs	- Il y avait une bonne entente au sein du groupe, comme on se connaissait déjà. On n'avait pas de problèmes à communiquer, et les séances de travail se passaient bien, on arrivait à faire pas mal de choses. On arrivait souvent à nos objectifs à la fin de la session.
Difficultés rencontrées	- L'implémentation de la grammaire a eu des débuts compliqués, on a dû tout refaire à un moment donné. Ensuite l'autre problème a été de trouver comment implémenter la table des symboles, il y avait une bonne documentation mais elle était unique, je n'ai pas réussi à trouver d'autres exemples pour nous aider.
Expérience personnelle	- C'est une bonne expérience de travail de groupe, avec des personnes avec qui je n'avais jamais travaillé. J'ai aussi consolidé mes compétences en Java, et appris/réappris un bon nombre de notions sur ce langage.
Axes d'amélioration	- Il faut être plus rigoureux au niveau des réunions. Elles étaient pertinentes et on remettait à plat ce qui avait été fait, mais comme elles étaient au final assez informelles nous ne faisions pas de compte-rendus. Je pense aussi que j'aurais pu travailler de manière plus régulière, et plus.

6.1.5 Bilan du projet par le groupe

Dans l'ensemble, la première partie du projet compilation c'est bien déroulé pour l'ensemble des membres du groupe, cependant nous nous accordons sur le fait que nous aurions pu mieux affiner notre gestion de projet.

6.2 Tableaux des temps

Domaines	Maxime	Nathan	Noé-Laurent	Thibault
Grammaire	10	8	8	12
Table des symboles	15	13	12	7
Arbre abstrait	12	10	16	10
GDP	1	2	1	1
Contrôles sémantiques	7	9	6	11
Réunion et Comptes-rendus	3	3	3	3
Rapport final	4	3	1	4
Total	52	48	45	48