Projet de Compilation
Langage Tiger

Rapport de projet 3

Auteurs
Thibaut CAGNON
Salim EL OUAFKI
Baptiste GHEYSENS

Responsable de module
Suzanne COLLIN

Octobre 2022-Janvier 2023

1 Introduction

L'objectif du projet de compilation est, pour les étudiants des filières IL, LE et ISS, d'écrire le compilateur d'un langage dit de "haut niveau", comme le language "Tiger". Pour les élèves des filières IAMD et SIE, l'objectif du projet est d'écrire la grammaire du langage afin d'en faire une analyse syntaxique et sémantique. Ce rapport d'activité rend compte, dans un premier temps, de la conception de la TDS (Table des symboles), puis des contrôles sémantiques du projet, et dans un second temps, de la gestion du projet.

2 Table des Symboles

2.1 Principe

Lors de l'analyse sémantique, on va vérifier que les constructions d'un programme sont sémantiquement cohérentes, c'est-à dire qu'une variable/fonction est bien déclarée, qu'une fonction contient le bon nombre ainsi que le bon type de paramètres, que les opérateurs sont appliqués à des types compatibles etc..

Pour cela on construit ce que l'on appelle une *Table des Symboles*, celle-ci rassemble toutes les informations utiles concernant les variables et les fonctions du programme. Elle est construite lors du parcours de l'arbre abstrait et grandit au fur à mesure des différentes déclarations. Elle est enfin, consultée pendant la compilation des parties exécutables comme un appel de fonction ou une référence à une variable.

2.2 Construction de la TDS

Nous avons construit un structure de table des symboles (TDS) afin de pouvoir effetuer par la suite des contrôles sémantiques sur les différentes classes de notre AST (Abstract Syntaxic Tree / Arbre Syntaxique Abstrait).

Pour cela, nous avons construit une classe TDS et une classe Tableau qui représente la structure d'une table des symboles avec son nom et ses instances.
Nous avons ensuite implémenter les classes nécessaires à la représentation d'une fonction, d'une variable et d'un type dans une TDS. Ces fonctions contiennent toutes leurs méthodes permettant de récupérer leurs paramètres.

Concernant la gestion des types, nous avons choisi de construire une classe abstraite Type qui permettra d'implémenter les classes gérant les types primitifs, Array et Record. Les types primitifs représentent non seulement les types **int** et **String** mais aussi tous les types qui ont été définis à partir de ces types là.

La classe TDS est la classe principale où les méthodes de vérification de déclaration sont implémentées. On y trouve aussi les méthodes permettant l'ajout de nouvelles infomations ainsi que l'extraction de celles-ci.

3 Contrôles sémantiques

3.1 Conception

Les contrôles sémantiques s'effectuent en même temps que la construction de la table des symboles. Pour les réaliser, nous avons utilisé une méthode récursive. Après la création de l'AST, nous parcourons les noeuds à l'aide de la fonction `getTDSandCheck`.

Ainsi, chaque classe hérite de cette fonction définie dans la classe abstraite AST. C'est dans chaque classe que sont effectués les ajouts à la TDS et les contrôles sémantiques lorsqu'ils sont nécessaires.

Nous avons considéré qu'il était nécessaire de créer une nouvelle région seulement lors de l'utilisation d'un `let..in..end` ou lors de la déclaration d'une fonction.

Concernant les contrôles sémantiques, nous avons utilisé l'un des documents à notre disposition (Tiger-Specification) et réalisé la majorité des contrôles sémantiques indiqués dans celui-ci. Ceux-ci étaient directement implémentés dans la fonction `getTDSandCheck` des noeuds concernés. À chaque fois qu'une erreur sémantique est détectée, la ligne de code où l'erreur se trouve est récupérée et affichée dans l'erreur.

```
(let var N := 8
in
end;
let var M := 8
in
if "bonsoir" then print("M is zero") else 0;
let
var N := 8
var N := 5
in
end
end)
```

Figure 1: Exemple de programme

```
région 0 --> imbrication 0
liste des tableaux déclarées :
    1 a un décalage égal à : 0
    2 a un décalage égal à : 0
--------------------------
région 1 --> imbrication 0.1
liste des instances déclarées :
    int N 0
--------------------------
région 2 --> imbrication 0.2
liste des tableaux déclarées :
    3 a un décalage égal à : 0
liste des instances déclarées :
    int M 0
--------------------------
région 3 --> imbrication 0.2.3
liste des instances déclarées :
    int N 0
--------------------------
```

Figure 2: TDS

```
Erreur ligne 9 : la condition doit être de type int
Erreur ligne 9 : le Then et le Else doivent être de même type
Erreur ligne 12 : la variable N est déjà instancié dans le scope
```

Figure 3: Liste des erreurs associées

3.2 Difficultés rencontrées

La partie des contrôles sémantiques a posé quelques difficultés, notamment au début pour mettre en place la récursion de la fonction. Nous avons ensuite pu réaliser la majorité des contrôles sé-

mantiques, même si la partie des déclarations de fonctions, variables et types a demandé un certain temps avant d'être comprise pleinement.

De plus, nous n'avons pas eu la possibilité de régler l'un des problèmes rencontrés concernant les lValue. En effet, dans la grammaire réalisée au début du projet, nous avions conservé une méthode récursive gauche pour la construction des lValue. Cependant, nous avons réalisé tardivement que cette décision, ajoutée à la sélection d'une fonction récursive pour traiter les contrôles sémantiques, rendaient très vite complexe leur gestion lors de ces derniers. Nous n'aurions pas eu le temps de réaliser les modifications nécessaires ou d'utiliser une autre méthode pour remplir la TDS. Nous avons donc décidé de ne traiter que des cas simples concernant les lValue.

4 Gestion de Projet

Les trois étudiants suivants forment l'équipe-projet :

- Thibaut CAGNON
- Salim EL OUAFKI
- Baptiste GHEYSENS

Nous avons réalisé les réunions majoritairement en présentiel principalement lors des créneaux prévus à cet effet, dans les locaux de TELECOM Nancy. Le fait que l'école laisse ses salles libres, ouvertes et à disposition nous a permis de nous regrouper plus facilement. Au début du projet, nous avons aussi créé un serveur Discord pour communiquer facilement entre nous et commencer à organiser les différentes parties du projet. Un Webhook a été mis en place, envoyant une notification sur Discord dès qu'un push était fait sur le GitLab du projet, nous permettant de suivre efficacement l'avancée des autres membres de l'équipe. Par la suite, ce Discord nous a permis de faire des réunions à distance, en vocal. Cela a également servi lorsque les membres ne pouvait pas être disponibles en présentiel.

Pour partager notre code, nous avons seulement utilisé le GitLab du projet créé par les professeurs. Ce rapport a été rédigé sous Overleaf, qui nous a été recommandé pour réaliser des projets en LaTeX.

4.1 Matrice SWOT

Nous avons réalisé une matrice SWOT afin d'évaluer les risques liés au déroulement du projet, mais aussi de mettre en avant les qualités de l'équipe et ses opportunités.

Forces	**Faiblesses**
Compétences en informatique et en gestion de projet; Introduction à Antlr;	Utilisation d'un nouveau logiciel ; Connaissances limitées autour de l'écriture d'un langage ;
Opportunités	**Menaces**
Plusieurs soutenances intermédiaires ; Des enseignants disponibles pour répondre à nos problèmes;	Durée des soutenances intermédiaires courtes;

4.2 Diagramme de Gantt

Nous avons réalisé un diagramme de Gantt afin de jalonner les étapes du projet :

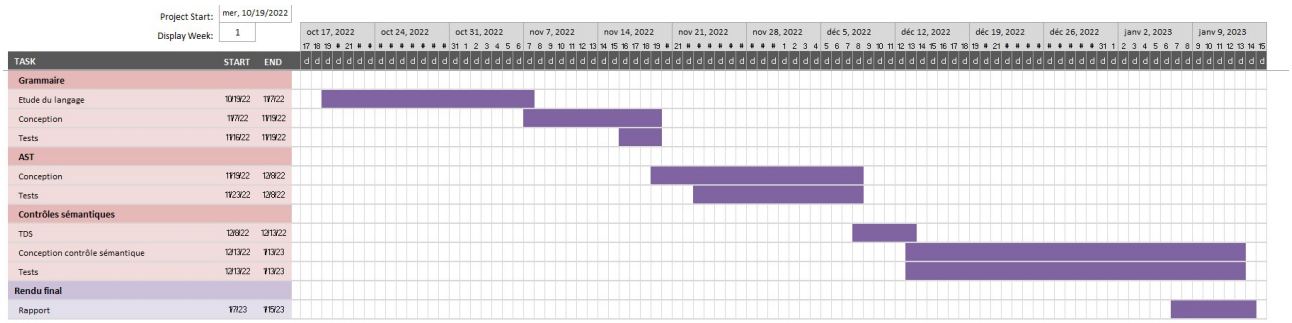

Figure 4: Diagramme de Gantt

4.3 Difficultés rencontrées

La principale difficulté rencontrée lors de ce projet a été le manque de communication au sein du groupe. Cela a eu pour résultat de freiner différentes étapes du projet. Cependant, ce problème a été résolu et a permis de réaliser les dernières étapes du projet plus efficacement.

5 Conclusion

En conclusion, nous avons réussi à implémenter la majeure partie des contrôles sémantiques souhaités dans notre projet de compilation. Cependant, il reste des erreurs au niveau des lValue qui n'ont pas été traitées complètement. Nous avons traité uniquement les cas simples en raison des contraintes de temps. Quelques petits problèmes liés aux types persistent et ne sont pas encore gérés. Malgré ces limitations, l'ensemble du projet de la grammaire aux contrôles sémantiques a été réalisé et fonctionne de manière satisfaisante.

6 Annexes

	Grammaire	AST	Contrôles sémantiques
Insuffisant	Avoir une grammaire qui ne prend pas en charge tout le langage	Avoir un AST qui ne simplifie pas l'arbre syntaxique ou qui n'est pas complet	Avoir moins de 3/4 contrôles sémantiques par personne
Réussite Acceptable	Avoir une grammaire qui prend en charge entièrement le langage	Avoir un AST qui simplifie l'arbre syntaxique à l'essentiel	Avoir 3/4 contrôles sémantiques par personne
Bon Travail	Avoir une grammaire qui produit un arbre syntaxique équilibré	//	Avoir 5/6 contrôles sémantiques par personne
Excellent	Avoir une grammaire LL(1)	Avoir un AST qui permette de pouvoir retrouver le programme facilement	Avoir l'entièreté des contrôles sémantiques

Figure 5: Matrice Objectif

MATRICE RACI	Acteurs		
Étapes	Baptiste	Thibaut	Salim
Grammaire			
Étude langage	AR	AR	AR
Conception	R	AR	R
Tests	AR	R	R
AST			
Conception	AR	AR	R
Tests	R	R	R
Contrôles sémantiques			
TDS	R	R	AR
Conception des contrôles	AR	AR	AR
Tests	R	R	R
Rapport	R	R	R

Figure 6: Matrice RACI

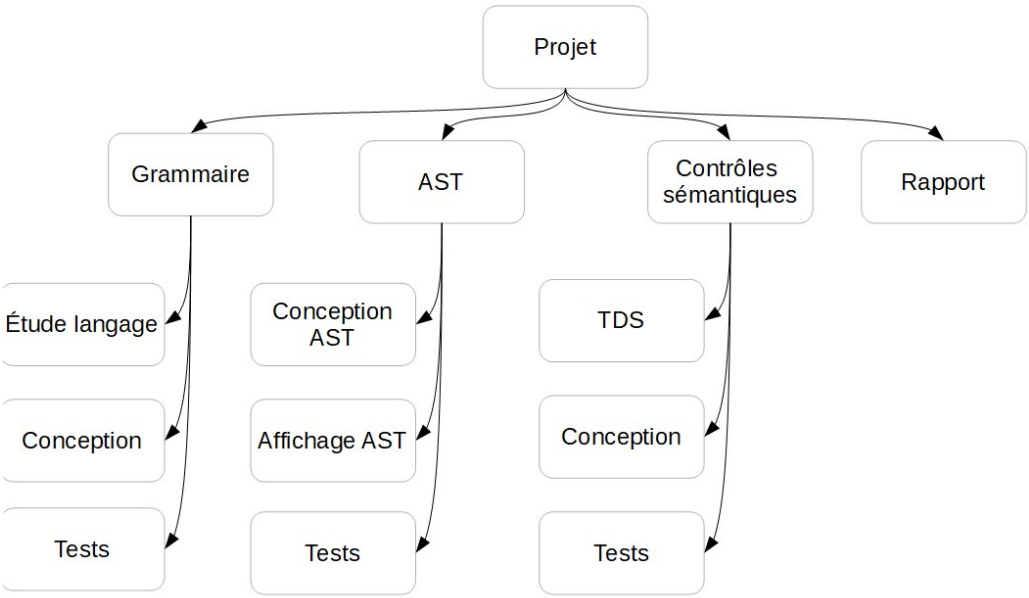
Projet
Grammaire
AST
Contrôles sémantiques
Rapport
Étude langage
Conception
Tests
Conception AST
Affichage AST
Tests
TDS
Conception
Tests

Figure 7: WBS