université de BORDEAUX

Projet de compilation

Rapport

Pierre Cayrol
Titien Cubilier
Thomas Furelaud
Jon Stark

Mai 2019

Table des matières

1 Plan du programme

1.1 Implémentation du lexer

Après avoir mis en place l'arborescence correcte de l'ensemble des fichiers sources de ce projet de compilation, nous avons commencé par implémenter le lexer permettant de réaliser l'analyse lexicale d'un fichier `.lea`. Pour ce faire, nous avons repris des anciens fichiers JFlex réalisés lors de séances de TP et adapté le programme aux différents tokens à identifier. Nous avons ainsi pu identifier l'ensemble des symboles comme par exemple `[] {} && <=` ainsi que la présence de mots clés comme par exemple `var begin end while` caractéristiques du langage Léa. Nous avons également traité les différents types de commentaires présents `//` ou bien `/*`. Nous avons pu décrire ce qu'étaient des entiers, des identifiants de variables ainsi que des nombres hexadécimaux.

1.2 Implémentation du package "Environment"

Une fois l'implémentation du lexer effectuée, nous avons ensuite implémenté le package `Environment` qui n'était pas fourni de base avec les ressources du projet. Ce package permet de sauvegarder les types nommés, les énumérés, les fonctions et les variables. Pour ce faire, nous avons décidé d'utiliser une table de hachage permettant de stocker l'ensemble des variables utilisées au moyen d'une clé et d'une valeur. La clé ici sera le nom de la variable de type `String` et la valeur sera le type de cette variable `Type`. La classe `Environment` implémente l'interface `EnvironmentInt` contenant les prototypes des deux méthodes réalisées `putVariable` et `getVariableType`. Ces deux méthodes permettront respectivement de stocker une nouvelle variable grâce à son nom et son type et de renvoyer le type d'une variable choisie dans la table de hachage. Enfin, dans la classe `StackEnvironment`, nous avons également stocké l'ensemble des variables locales utilisées dans le programme.

1.3 Implémentation de la grammaire

L'implémentation de la grammaire a représenté la majeure partie du projet. En effet, il a fallu que nous analysions l'intégralité des classes Java des packages `node` et `type` pour que cela corresponde bien aux types attendus et que le typage se réalise correctement. Nous avons implémenté la grammaire en avançant sur chaque programme de test : des programmes `progr1.lea` à `progr9.lea`. Nous avons ainsi pu compléter chaque règle de la grammaire

en relation avec ce qui était dans les fichiers sources du projet.

1.4 Génération des arbres de syntaxe

La génération des arbres de syntaxe a été rendu possible grâce à la fonction `toDot` qui permet de convertir le typage de chaque fichier de test `.lea` en fichier `.dot` permettant ensuite d'avoir un rendu visuel de ce qui a été produit. Nous avons pu ainsi vérifier que le typage du programme d'entrée correspondait bien à l'arbre de syntaxe généré.

Voici un exemple de programme `.lea` de test :

```
1   var
2   a: integer;
3
4   begin
5   a = 100;
6   while (a >= 0) do
7           begin
8           println (a);
9           a = a - 1;
10          end
11  end
```

L'arbre produit correspondant à ce programme est le suivant :

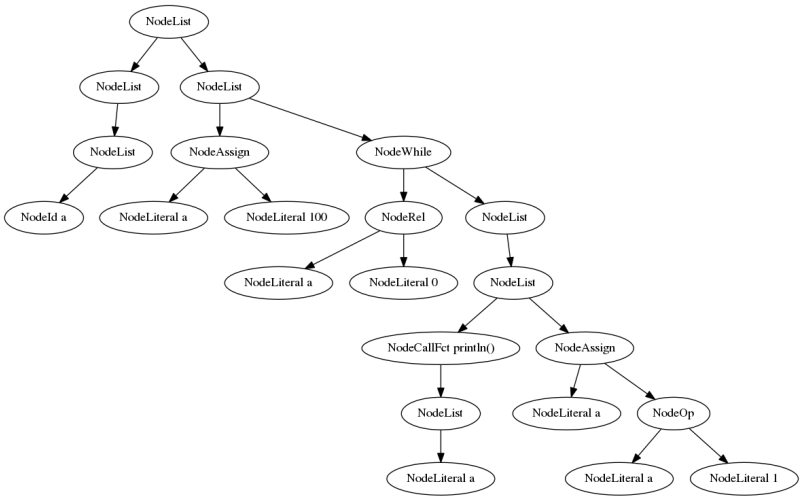

FIGURE 1 – Arbre généré à partir du progr1.lea

2 Difficultés rencontrées

— Nous avons rencontré des difficultés lors de l'implémentation de la règle `procedure_head` avec notamment l'emploi du type `TypeFeature`. En effet, nous ne savions pas quel type appeler lors de la production de la règle de grammaire.

— Nous avons également rencontré des difficultés au niveau du code intermédiaire. La compréhension globale et se lancer dans l'implémentation de celui ci a été difficile.

3 Tests effectués

3.1 Fichiers .lea fournis

Nous avons testé notre grammaire ainsi que l'ensemble de notre programme grâce aux fichiers présents dans les sources du projet. Ces fichiers (`progr1.lea`, ..., `progr9.lea`) représentent une version relativement complète de l'ensemble des règles de grammaires.

3.2 Fichier de test supplémentaire

Nous avons également développé un programme de test supplémentaire `progTest.lea` utile lors du début de l'implémentation de la grammaire :

```
var
a: integer;

begin
a = 100 + 1;
end
```

4 Conclusion

En guise de conclusion, nous pensons avoir bien compris l'aspect de l'analyse sémantique, syntaxique et lexicale. Nous n'avons pas pu avancer davantage dans l'implémentation du code intermédiaire par manque de connaissances et d'exemples concrets.