COMP : Compte rendu TP2

Aurèle Barrière & Antonin Garret

9 décembre 2016

1 Introduction

Ce projet vise à implémenter la face avant d'un compilateur de VSL `[ref needed]` vers du code 3 adresses. Le but est d'arriver à traiter un langage avec des structures de contrôles, des boucles, des fonctions avec leur prototypes, des appels à des fonctions et des tableaux.

Un parser était déjà fourni et nous a permis de nous concentrer uniquement sur la génération de code 3 adresses depuis un arbre de syntaxe abstrait.

2 Description de la méthodologie

Nous avons implémenté notre compilateur progressivement, de sorte à identifier plus simplement d'éventuelles ereurs. Dans un premier temps, nous ne nous sommes occupés que de la génération de code associé aux expressions arithmétiques. Ensuite, celui du code des expressions et des blocs.

Enfin, il nous a fallu implémenter la génération de code des fonctions et prototypes, puis celle des tableaux.

À chaque étape, nous avons créé des programmes (ou des expressions, ou des instructions) pour tester notre génération. Dans un premier temps, ce code était comparé à celui des exemples dont nous disposions dans le sujet. Ensuite (quand les programmes et les instructions PRINT étaient traités), nous avons pu compiler le code 3 adresses en un code exécutable qui permettait de vérifier le comportement.

3 Bilan du travail réalisé

3.1 Génération du code d'expressions

3.2 Génération du code d'instructions

3.3 Génération du code de programmes

3.4 Vérification de type

Enfin, nous avons implémenté un système de vérification de types pour éviter certaines erreurs. En effet, on ne veut pas qu'un programme puisse assigner un tableau à une valeur entière par exemple. Nous avons donc créé une fonction récursive de typage d'expression. Nous avons également dû augmenter le type de la table de symboles, pour qu'un enregistrement retienne aussi son type.

4 Programme d'exemple

5 Tests effectués

Nous avons commencé par créer un script en Bash qui affichait les programmes de tests puis les compilait, puis les exécutait pour automatiser le lancement des tests.

Une fois que notre compilateur générait du code pour les programmes, nous avons pu lancer les tests fournis avec le sujet. Ceux-ci nous ont permis de vérifier que notre gestion des blocs, fonctions, prototypes, tableaux... était correcte.

Ils nous ont également permis de soulever un problème de notre implémentation : les instructions Return dans les fonctions de type Void. Dans une première version, nous avions décidé que les programmes qui essayaient de retourner une valeur dans un programme de type Void n'étaient pas corrects et ne devaient pas être compilés. Cependant, de nombreux programmes de tests le faisaient, puisqu'il n'y pas d'instruction Return sans argument en VSL. Nous avons donc modifié notre compilateur pour qu'un tel appel crée une intruction `ret void` dans le code 3 adresses généré.

6 Conclusion