TP2 - Compilateur VSL+ ANTLR/Java

Matthias Gradaive - Romain Chiquet

3 décembre 2014

Sommaire

Introduction

L'objectif de ce TP est de créer un compilateur pour la langage VSL+ en utilisant Java et le framework ANTLR. VSL+ est un langage basique comportant des instructions relativement courantes :

- Expressions mathématiques simples (+,-,*,/)
- Déclaration et affectation de variables entières
- Gestion des bloques d'instructions et de la portée des variables
- Instruction de contrôle if/else et while
- Appel des fonctions systèmes PRINT et READ
- Définition et appel de fonctions (et de leurs prototypes)
- Gestion des tableaux de taille fixe

Le lexer et le parser sont fournis. Par conséquent, notre travail est d'écrire un arpenteur d'arbre (TreeParser) pour convertir un arbre étant le résultat d'un programme VSL+ en une suite d'instruction 3 adresses.

1 Méthodologies de travail

1.1 Méthode de développement

L'exemple de départ (les expressions simples) étant déjà compilable et utilisable, nous sommes avons commencé avec un simple fichier de code test en VSL+ contenant seulement une expression. Nous avons ensuite implémenté les fonctionnalité du langage une à une. Dans le même temps nous complétons notre fichier VSL+ de test pour tester les fonctionnalités qui venaient d'être rajoutées. A chaque mise à jour de notre code VSL+ de test, nous gardons les anciens tests créés pour des fonctionnalités rajoutées auparavant. Cela nous permet de réaliser un test de non-régression (pas infaillible mais utile) pour s'assurer que l'ajout de nouvelles fonctionnalités ne provoquait pas de bogues dans le code déjà écrit.

1.2 Organisation du code

Le TreeParser à globalement le même format que le Parser. La classe Code3aGenerator comporte des méthodes static pour la création du code 3 adresses (Code3a) correspondant aux éléments de l'enum TAC. Si le code Java présent dans le TreeParser est assez long, nous le déplaçons également dans le classe Code3agenerator afin d'améliorer la lisibilité du programme.

Nous n'avons pas utilisé la classe Error. Lorsque le programme rencontre une erreur, nous utilisons la sortie *System.out.err* de Java pour afficher un message permettant d'identifier la source de l'erreur.
Nous n'avons également pas utilisé la classe TypeCheck. Nous effectuons tous les tests de type dans le corps des méthodes présentes dans Code3agenerator.

1.3 Gestion de versions

Nous utilisons Git pour la gestion du code. Cela nous permet entre autres de retrouver une ancienne version d'un code (pour par exemple retrouver un code avant l'apparition d'un bug) et de pour pouvoir travailler à plusieurs sans désagréments.

2 Travail réalisé

TODO

Conclusion

TODO