Compiladores - Projecto

iJava

João Ricardo Lourenço, Nº 2011151194
Joaquim Pedro Bento Gonçalves Pratas Leitão, Nº 2011150072

29 de Maio de 2014

Relatório

Índice

1 Introdução

O presente trabalho pretende-se desenvolver um compilador para a linguagem $iJava$, um pequeno subconjunto da linguagem Java (versão 5.0). Por ser um subconjunto de uma outra linguagem, todos os programas que respeitem as regras impostas em $iJava$ são também, garantidamente, programas válidos em $Java$.

Nesta linguagem todos os programas são constituídos por uma única classe, que possui métodos e atributos estáticos, e públicos. Para além disso, a classe necessita obrigatoriamente de ter um método *main*, onde a execução do programa se inicia.

Podemos utilizar literais dos tipos inteiro e booleano e variáveis inteiras, boleanas e arrays unidimensionais de inteiros e booleanos.

A linguagem implementa também expressões aritméticas e lógicas, operações relacionais simples, instruções de atribuição e controlo ($if - else$ e $while$).

Os métodos definidos, e os respectivos valores de retorno, podem receber como argumentos variáveis de qualquer tipo acima mencionado, com excepção do método *main*, que tal como em $Java$ possui como tipo de retorno o tipo *void*.

É também possível passar parâmetros (literais inteiros) ao nosso programa através da linha de comandos. É o método *main* que vai receber esses parâmetros, armazenando-os num array de objectos do tipo *String*. Embora este tipo de dados não esteja incluído na lista de tipos permitidos em $iJava$, a sua utilização apenas é permitida no método *main*, com a mera finalidade de obter os parâmetros passados ao programa aquando da sua invocação.

O desenvolvimento do compilador foi dividido em três fases distintas.

Numa primeira fase foi realizada a *Análise Lexical* do programa fonte, onde são identificados *tokens*, isto é, cadeias pertencentes à linguagem e que têm significado e relevância para o programa.

Seguiu-se a *Análise Semântica*, composta por quatro etapas principais: FIXME, MUDAR A MANEIRA COMO ESTAMOS A DIZER O QUE SE SEGUE

- **Tradução da gramática-fonte** (fornecida em notação $EBNF$) para o yacc, permitindo assim reconhecer se o programa fornecido é constituído por sequências de *tokens* pertencentes à linguagem, e assim detectar eventuais erros de sintaxe.

- **Construção da árvore de sintaxe abstracta**, etapa que é realizada em simultâneo com a verificação e validação do programa a compilar, de acordo com a gramática da nossa linguagem. A árvore de sintaxe abstracta irá representar o nosso programa a compilar, recorrendo a uma estrutura em árvore para representar as estruturas sintáticas das cadeias que constituem o programa a compilar.

- **Construção da tabela de símbolos**, utilizadas para armazenar informações relevantes sobre a classe (seus atributos e métodos), bem como sobre cada método definido pelo programador (como, por exemplo, o tipo de retorno e os argumentos).

- **Verificação de erros semânticos**, etapa principal da *Análise Semântica*, onde são realizadas verificações de tipos, garantindo que para cada operação a realizar não existem incompatiblidades de tipos entre os operandos nela envolvidos.

A última fase do trabalho consistiu na *Geração de Código Intermédio*, da qual resulta, na representação intermédia de $LLVM$, um programa equivalente ao que pretendemos compilar.

METER IMAGEM BONITINHA, TIPO CACEIRO???

2 Análise Lexical

Tal como referimos anteriormente, na *Análise Lexical* procedemos à identificação dos *tokens* da nossa linguagem. Sempre que é detectada a presença de um comentário no programa a compilar, seja do tipo //... (comentários de apenas uma linha) ou do tipo /∗...∗/ (comentários multi-linha), os caracteres incluídos nesse comentário são ignorados.

Sempre que é detectado um caracter, ou uma sequência de caracteres, que não constitui nenhum *token* é detectado um erro lexical, sendo impressa uma mensagem de erro, indicando a existência de um caracter ilegal, juntamente com a sua posição no programa.

Adicionalmente, caso se verifique a ocorrência de um comentário multilinha que não foi devidamente terminado, o erro lexical é também detectado, sendo impressa uma mensagem de erro que indica a posição no programa onde o comentário foi iniciado.

Em seguida, apresentamos a lista dos *tokens* válidos na linguagem *iJava* e a lista dos *tokens* reservados que, por essa razão, não estão disponíveis na nossa linguagem:

1. **BOOLLIT**
2. **INT**
3. **BOOL**
4. **NEW**
5. **IF**
6. **ELSE**
7. **WHILE**
8. **PRINT**
9. **PARSEINT**
10. **CLASS**
11. **PUBLIC**
12. **STATIC**
13. **VOID**
14. **STRING**

3 Análise Sintática

4 Construção da Árvore de Sintaxe Abstracta

5 Análise Semântica

5.1 Tabelas de Símbolos

5.2 Detecção de Erros Semânticos

6 Geração de Código

7 Apreciação do Trabalho