Universidade Federal de Juiz de Fora
Departamento de Ciência da Computação
DCC045 - Teoria dos Compiladores Semestre ERE 2020-1

Interpretador

Edson Lopes da Silva Junior 201635023

Vinicius Alberto Alves da Silva 201665558C

Professor: Leonardo Vieira dos Santos Reis

Relatório do trabalho prático Interpretador, parte integrante da avaliação da disciplina.

Juiz de Fora

Novembro de 2020

1 Introdução

Este relatório é do trabalho prático 3 da disciplina de Teoria dos Compiladores do Ensino Remoto Emergencial (ERE) 2020.1 e tem como objetivo descrever o processo da criação de um Interpretador (AS) para a Linguagem *Lang*.

2 Metodologia Utilizada

De início, foi necessário escolher qual padrão de projeto seria utilizado para o desenvolvimento do interpretador. Devido o código de exemplo ser no padrão Visitor, este padrão foi escolhido.

Para implementação do Visitor o Antlr fornece uma opção de atravessar á árvore de sintaxe, ele automaticamente gera as classes usadas na visita dos nós ao passar o parâmetro -visitor ao gerar os arquivos.

2.1 Organização do Código

Com relação a organização do código, a pasta Parser contém os arquivos gerados pela ferramenta Antlr: LangLexer.tokens, LangLexer.interp, Lang.interp, LangParser.java, LangBaseListener.java, LangLexer.java, LangListener.java. Estes são que estão responsáveis pela análise léxica e sintática. Nesta mesma pasta estão os arquivos LangVisitor.java e LangBaseVisitor.java que são as classes geradas pelo Antlr responsáveis por percorrer a AST. Também na pasta Parser está o arquivo CreateASTFromParser.java que estende LangBaseVisitor<SuperNode> e implementa os métodos de visita de cada nó da AST.

Por último, a classe LangAdaptor implementa a interface ParseAdaptor. A classe LangAdaptor é responsável por chamar o método ParseFile, este faz a leitura do arquivo a partir da classe LangLexer e é feito o parser com a classe LangParser.

Na pasta AST encontram-se as classes que representam cada nó da AST. Todas as classes herdam da superclasse abstrata SuperNode. Além disso todas as classes possuem a possibilidade de utilizar o método *accept* utilizado na execução do Interpretador.

Na pasta Visitor esta a interface Visitable, a classe abstrata Visitor e a classe InterpretVisitor, seguindo modelo visto na aula.

A classe TesteParser fornecida no trabalho anterior foi alterada para rodar apenas um teste, o arquivo para teste chama-se mytest.lan e encontra-se na pasta testes/sintaxe.

2.1.1 Como executar

Para executar basta executar o arquivo bash run.sh, ele contem o seguinte conteúdo:
#!/bin/bash

if [-z "$1"]; then antlr_jar="antlr-4.8-complete.jar"; else antlr_jar=$1; fi

java -jar ./$antlr_jar -o -visitor ./parser/ Lang.g4
javac -cp .:$antlr_jar ast/*.java parser/*.java LangCompiler.java -d .
java -classpath .:$antlr_jar lang.LangCompiler -bs

É possível trocar o endereço do jar do antlr 4.8 por outro de sua preferência, basta fornecer o caminho do .jar na chamada de execução do script, exemplo:
bash run.sh pasta1/pasta2/antlr-4.8-complete.jar

2.1.2 Detalhes de implementação

Na gramática (arquivo Lang.g4) cada regra foi rotulada com um #nome_rotulo. Desta forma o antlr gera um método *visit* para o objeto correto de acordo com o rótulo. As regras que tem apenas uma derivação não precisam de ser rotuladas.

O arquivo gerado automaticamente pelo Antlr *LangBaseVisitor* contém a chamada dos métodos *visit*. Desta forma, foi implementada uma nova classe que estende *LangBaseVisitor* e subscreve seu métodos. Esta nova classe chama *CreateASTFormParser*. A facilidade desta forma de implementação é que visitar um nó da árvore pode ser encarado como apenas chamar as regras definidas na gramática.

Também foi necessário definir como vamos interpretar, a partir do Visitor, cada nó. Isso é feito na classe *InterpretVisitor*, de forma similar ao que foi apresentado na aula.

3 Conclusão

Este relatório apresentou o processo de desenvolvimento de um interpretador para a linguagem lang. Foi escolhido o padrão de projeto Visitor, a ferramenta ANTLR fornece uma opção para atravessar á árvore de sintaxe, bastando apenas definir como cada nó deve ser visitado. Também foi escrita uma classe para interpretar cada nó no momento de visita. O desenvolvimento deste projeto permitiu o contato com o padrão de projeto visitor, e também serviu como complemento do conteúdo exibido nas aulas.

A dificuldade na realização do trabalho foi a necessidade de escrita de muita parte do código para começar a ser possível fazer experimentações que resultassem em algo. O começo da implementação também foi áspero muito pela nossa falta de experiência em lidar com as classes geradas pelo antlr. Na implementação da classe InterpretVisitor a dificuldade foi construir os métodos para interpretar cada nó, principalmente relacionados aos Parâmetros, ao de chamada de função e declaração de arrays. Além disso não foi possível aceitar parametros na função main pelas tomadas de decisão feitas ao longo da implementação do Interpretador.