Universidade Federal de Juiz de Fora
Departamento de Ciência da Computação
DCC045 - Teoria dos compiladores

Interpretador

Grupo

André Dias Nunes – MAT: 201665570C

Guilherme Barbosa – MAT: 201435031

Leonardo Reis

Relatório do segundo trabalho da disciplina DCC045 – Teoria dos Compiladores. Desenvolvimento de um Interpretador.

Juiz de Fora

Julho de 2022

Sumário

1 Introdução

Neste relatório será apresentado a estratégia e a ferramenta utilizada para a criação do analisador interpretador para a linguagem `lang`, o qual foi construído com o objetivo de contemplar a execução do terceiro trabalhado de Teoria dos Compiladores.

2 Árvores sintáticas abstratas(AST)

As ASTs representam a estrutura sintática em árvores, sendo que, não existe uma demonstração das derivações realizadas por meio dos não terminais, pois os nós são símbolos terminais da gramática (Figura 2).Vale ressaltar que os nós da árvore representam um construtor no código fonte.

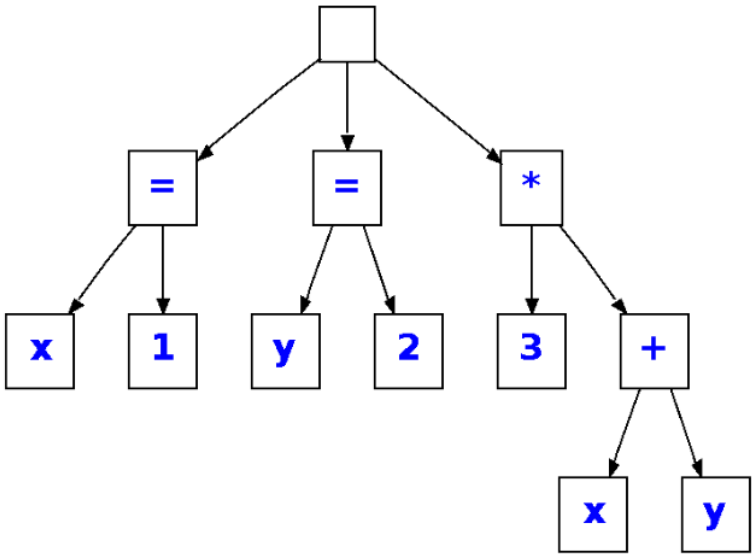

Figura 1: Exemplo de Árvores sintáticas abstratas.

3 Interpretadores e Compiladores

Para que um programa, escrito em uma linguagem de alto nível, possa ser executado em um computador é necessário converter o mesmo em linguagem de máquina. Esta conversão é feita por um tipo especial de programa que recebe instruções em linguagem de alto nível e dá como saída outro programa constituído de instruções binárias.

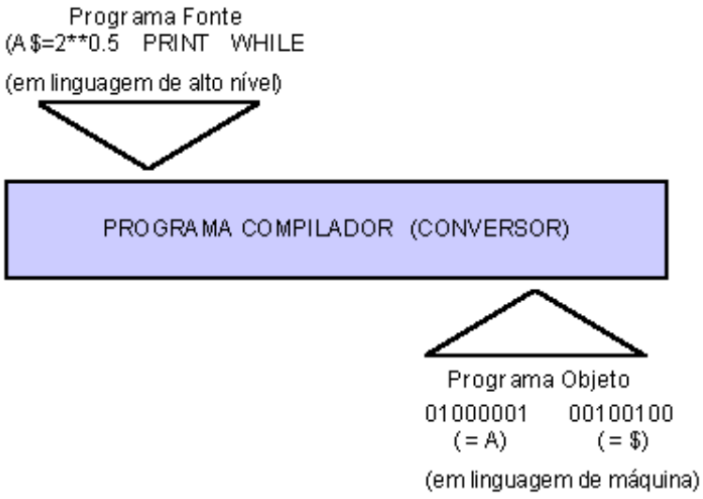

Figura 2: Programa compilador.

Nos tópicos seguintes vamos demonstrar dois métodos básicos de abordagem na tradução de linguagem de alto nível para linguagem de máquina.

3.1 Compiladores

Os compiladores são usados para traduzir uma linguagem de alto nível, como c++, para uma linguagem de mais baixo nível.Para isso o compilador utiliza duas etapas:análise e síntese.

Na análise o compilador verifica se existem erros de digitação no código, como por exemplo a falta de um ";", e se a linguagem de alto nível segue uma estrutura que pode ser reconhecida.

Não havendo inconsistência o compilador poderá executar a síntese.Nesta etapa o código de alto nível será traduzido para um linguagem de baixo nível, como por exemplo assembly. Ao final deste processo um arquivo, contendo o código compilado em linguagem de máquina, será gerado.

3.2 Interpretador

Diferente do compilador, o Interpretador não realiza a "tradução" do programa inteiro, sendo que, é realizada uma conversão do código-fonte por partes. Ao final da verificação de cada instrução, caso não seja encontrado nenhum erro, o Interpretador realiza a tradução e a execução da mesma. Este processo é repetido sucessivamente até a última instrução do

programa fonte. Vale lembrar que quando a segunda instrução é trabalhada, a primeira é perdida.

O processo mencionado acima é executado toda vez que um programa fonte Interpretado é executado. Este comportamento se dá pelo fato do Interpretador não gerar nenhum arquivo, contendo comandos em linguagem de máquina.

4 Desenvolvimento

4.1 AST

Para o presente trabalho foi desenvolvido uma AST a partir da CST gerada pelo ANTLR, com o intuito de construir um interpretador para a linguagem lang.

O parser já havia sido criado no trabalho anterior, mas foi necessário fazer algumas alterações no 'LangAdapter' para aceitar os testes do parse e do interpretador. Após a verificação de sintaxe, a ast é construída e validada. O makefile tb foi alterado, para que se possa gerar o arquivo 'langBaseVisitor' que será usado para realizar a conversão da CST em AST.

Os nós da AST foram gerados na pasta 'ast', sendo que os arquivos foram gerados com base na gramática desenvolvida. Para facilitar o entendimento do código os nomes dos arquivos seguem os padrões dos não terminais definidos nas gramáticas, e nos casos das expressões, como por exemplo 'NOT sexp' o nome do arquivo segue o padrão do terminal relevante, que no exemplo acima é o 'Not'.

Para que a AST seja gerada, determinamos contextos na gramática 'lang.g4', demarcados pelo #, os métodos de cada contexto são definidos em 'LangVisitors',localizado na pasta 'visitors', que faz uso da classe 'langBaseVisitor' gerada pelo próprio ANTLR. Nele, os nós da árvore são criados e validados. Para realizar a visita aos nós da AST foi criada uma classe base abstrata denominada 'Visitor'. Para sua criação, utilizamos como modelo a classe de mesmo nome fornecida nos materiais de aula.

4.2 Interpretador

Para iniciarmos o interpretador, criamos uma classe similar ao 'TestParser'. Esta nova classe foi definida como 'TestVisitor' e contém duas mudanças:

- A primeira é que incluímos o output emitido pelo 'InterpretVisitor', este que, por sua vez, foi criado utilizando como exemplo a classe 'InterpretVisitor' do interpretador fornecido pelo professor nos materiais de aula.

- A segunda mudança está após a validação da análise sintática (Figura 3), onde é inicializado o 'InterpretVisitor' que implementa os métodos da classe abstrata 'Visitor'. Neste momento é percorrido a árvore gerada pelo 'LangVisitors'. Em seguida, é feita a validação.

```
SuperNode node = adp.parseFile(s.getPath());
if(node != null){
    System.out.println(x: "Output: ");
    Visitor v = new InterpretVisitor();
    node.accept(v);
    flips++;
}
```

Figura 3: TestVisitor.java

Vale lembrar que, diferente do analisador sintático, caso seja encontrado um erro ao rodar o interpretador o processo será encerrado e um erro será apresentado na tela (Figura 4).

```
Testando: ../testes/semantica/errado/function1.lan
Output:
java.lang.RuntimeException:  (7, 2) Index 0 out of bounds for length 0
        at lang.visitors.InterpretVisitor.visit(InterpretVisitor.java:129)
        at lang.ast.CallCmd.accept(CallCmd.java:52)
        at lang.visitors.InterpretVisitor.visit(InterpretVisitor.java:197)
        at lang.ast.Func.accept(Func.java:59)
        at lang.visitors.InterpretVisitor.visit(InterpretVisitor.java:58)
        at lang.ast.Program.accept(Program.java:46)
        at lang.visitors.TestVisitor.runOkTests(TestVisitor.java:55)
        at lang.visitors.TestVisitor.<init>(TestVisitor.java:30)
        at lang.LangCompiler.main(LangCompiler.java:54)
```

Figura 4: Exemplo erro interpretador

O Data da gramática foi definido como sendo um objeto de chave dupla, contendo a variável e o tipo da mesma. Ao fazer com que uma variável recebe um identificador do tipo Data temos a realização de uma varredura no mesmo, recuperando as chaves duplas contidas no objeto em questão. As chaves duplas recuperadas serão salvas em uma lista que poderá ser usada para a alteração dos valores das variáveis da Data.

5 Execução

Para a execução do programa foi criado um arquivo makefile, certifique-se de estar dentro do diretório lang para chama-lo. Use os seguinte comandos:

- **make:** utilizado para realizar a criação das classes do ANTLR, a partir da gramática definida no arquivo lang.g4 (localizada no diretório parser), além da compilação das classes do projeto.
- **make run cmd=-bs** Executa uma bateria de testes sintáticos, referentes ao trabalho 2.
- **make run cmd=-bsm** Executa uma bateria de testes no interpretador, referentes ao trabalho 3.
- **make clean** utilizado para realizar a limpeza dos arquivos gerados pelo ANTLR e dos .class gerados.
- **make all cmd=-bsm** Compila, executa e faz a limpeza.