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 quarto trabalho da disciplina DCC045 – Teoria dos Compiladores. Desenvolvimento de um Analisador Semântico.

Juiz de Fora

Julho de 2022

Sumário

1 Introdução

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

2 Analisador Semântico

Para garantir que o algoritmo criado é coerente e possa ser convertido para linguagem de máquina é necessário realizar a execução de análise semântica, que tem como objetivo validar a existência de inconsistências na utilização dos identificadores, validando se os tipos informados estão seguindo as regras definidas pela linguagem.A análise semântica percorre a árvore sintática relacionando os identificadores com seus dependentes de acordo com a estrutura hierárquica.

Na figura 2 temos a exemplificação da etapa de validação de tipos.Vale lembrar que em alguns casos, o compilador realiza a conversão automática de um tipo para outro que seja adequado à aplicação do operador.

```
var s: String;
s := 2 + ‘2’;
```

Figura 1: Exemplo de validação de tipos.

3 Desenvolvimento

3.1 Estrutura

Para este trabalho foi usado como base a configuração do arquivo “Interpreter”, fornecido pelo professor. Sendo assim foi criado um novo pacote, denominado “langUtil”, onde estão localizados os arquivos responsáveis pelo armazenamento dos tipos que são aceitos pela linguagem lang, além do arquivo “STyErr” sendo usado para os erros semânticos identificados ao longo da leitura do arquivo informado na entrada.Neste mesmo pacote está localizado o mais dois tipos denominados "TyEnv"e "LocalEnv"que são tipos que auxiliam nas validações das variáveis declaradas do analisador sintático.

A classe “TypeCheckVisitor” foi criada com o objeto de realizar a varredura da árvore AST e realizar a análise semântica da mesma. Para realizar a validação do analisador

semântico criado foi adicionado ao pacote “visitors” mais uma classe de teste, denominada “TestType”.

Na classe “TypeCheckVisitor” temos algumas definições, que estão sendo melhor detalhadas na lista "Definição da Estrutura de tipos", relacionadas às novas classes do pacote “langUtil” e aos tipos aceitos pela linguagem lang.

1. Definição da Estrutura de tipos

    - O tipo “Data” foi definido como uma hashmap de strings e STyleData(classe localizada no pacote “langUtil”).
    - ”localEnv” é formado por um TreeMap do tipo LocalEnv, onde é armazenado os tipos e os identificadores das variáveis do programa de entrada.
    - A variável “funcs” é formada pelo hashmap de strings e ArrayList contendo o tipo da função atual.
    - A ”tempFunc” é um objeto do tipo STyFunc que tem como intuito armazenar o tipo da função atual
    - As variáveis “tyint, tyfloat, tybool, tychar, tynull, tyvar” são usadas para representar os tipos básicos aceito pelo sistema.
    - A variável stk é uma pilha de SType contendo todos os tipos que estão sendo avaliados na execução do programa

Na classe do “LangCompiler” foi disponibilizado o comando “-bty” que tem como objetivo realizar a análise semântico nos arquivos de entradas, localizadas na pasta “Testes”.Além do que o analisador sintático está sendo executado ao realizar a chamada da execução do interpretador.

Vale ressaltar que as mensagens de erros ,informada na classe "TypeCheckVisitor", tiveram como base o glossário de erros da linguagem Java(estas informações foram retiradas do link compile time error messages : Java Glossary).

4 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, mas agora a verificação do sistemas de tipos foi acrescentada.
- **make run cmd=-bty** Executa uma bateria de testes no sistemas de tipos, referentes ao trabalho 4.
- **make clean** utilizado para realizar a limpeza dos arquivos gerados pelo ANTLR e dos .class gerados.
- **make all cmd=-bsm** ou **make all cmd=-bty** Compila, executa e faz a limpeza.