Relatório - Geração de Código

09 de Julho de 2023

Nome Completo	**Matrícula**
Lívia Pereira Ozório	201835011
Patrick Canto de Carvalho	201935026

Sumário

1 Introdução **3**

2 Geração de Código **3**

2.1 Informações gerais 3

2.2 Estruturas de dados e ferramentas 3

2.3 Geração Código Java 3

2.4 Geração Código Jasmin 4

3 Como compilar **4**

1 Introdução

Este relatório tem como objetivo explicitar os pontos mais relevantes para a compreensão do processo de desenvolvimento da geração de código para a linguagem *lang*, incluindo as decisões de projeto como as estratégias adotadas e estruturas de dados auxiliares.

2 Geração de Código

2.1 Informações gerais

Foi escolhido para esta parte da implementação do trabalho o padrão *visitor*. Para tal, foi utilizada uma classe abstrata Visitor, extendida na classe JavaGenVisitor e JasminGenVisitor.

2.2 Estruturas de dados e ferramentas

Algumas estruturas auxiliares foram criadas para realizar a geração de código:

- codeStack: é uma pilha de objetos do tipo ST (String Template), que armazena o resultado do template gerado por cada nó verificado por meio da função accept. Cada vez que esta função é chamada para um nó, o template gerado é armazendo nesta pilha e quando deseja-se consultar, o último elemento da pilha é removido.
- scopeVisitor: é a estrutura utilizada para a analise semântica. Com ele é possível verificar os tipos das expressões e variaveis existentes no programa.
- unique_id: não é uma estrutura de dados e sim um número inteiro. É útil na geração de identificadores únicos para variáveis ou rótulos. Sendo incrementado a cada vez que é utilizado.

Além das estruturas foi utilizado a ferramenta StringTemplate para ajudar na formatação dos códigos. Sendo gerado 2 templates na ferramenta, um para a geração em Java e outro para a geração em Jasmin.

2.3 Geração Código Java

A seguir estão as principais decisões tomadas para que fosse gerado código Java correto:

- Leitura do teclado: ao inicio do código Java, é criada uma varável input do tipo Scanner, bastando ser chamada sempre que for necessário efetuar uma leitura. Também foi criada uma função auxiliar responsável pela conversão da String digitada para o tipo da variável alvo.
- Declaração de variáveis: as declarações são realizadas ao inicio de cada bloco, sendo declaradas as variáveis criadas no respectivo bloco. Para consultar o seu tipo, é feito um mapeamento entre os tipos do Java e os tipos existentes no Lang utilizando o escopo já existente proveniente do analisador semântico.
- Tipos de dados: são representados como classes. Os campos do tipo de dado original se tornam atributos públicos nesta classe.
- Retorno das funções: toda função que possui retorno retorna uma lista de Object. Dessa forma, tanto funções multivaloradas quanto funções que possuem só um valor funcionam da mesma forma podendo o código ser agnóstico a isto.

As demais estruturas da linguagem Lang foram convertidas para Java de forma trivial, devido à similaridade.

2.4 Geração Código Jasmin

Importante: Houve muita dificuldade na implementação do código na linguagem Jasmin devido a: 1) Pouca familiaridade com a linguagem e, de forma geral, o funcionamento da JVM; 2) Falta ou incompletude de documentação e outras infomações (como ajuda com mensagens de erro); 3) Mensagens de erro extremamente genéricas e difíceis de entender geradas pela JVM; 4) Impossibilidade de depuração do código gerado em Jasmin; 5) Pouco tempo para a execução do trabalho.

A seguir estão as principais decisões tomadas para que fosse gerado código Jasmin correto:

- Tamanho da pilha: Para saber o tamanho da pilha durante a analise semântica é contado o maior número de elementos que a pilha poderá receber durante a execução com base nos comandos contidos no escopo da função. Dessa forma é possível saber o tamanho que ela deve ter.
- Tamanho da área de varáveis locais: Para saber o numero de variáveis locais é conferido qual o numero de variáveis existe em uma função e soma-sem mais 3 para uso especifico das operações e procedimentos existentes no escopo.
- Inicialização das variáveis: Para fazer a inicialização das variáveis foi realizado, na analise semântica, um mapeamento das variáveis para um valor numérico único que representará o endereço de cada variável na área de variáveis locais da JVM. A inicialização desses espaços é realizada ao início de cada função, de forma análoga ao que foi feito no código Java.
- Saltos condicionais: quando se deseja saltar para outra parte do código, gera-se um rótulo único para cada local utilizando a variável auxiliar unique_id mencionada na subseção 2.2.
- Chamada de função: quando se tem uma chamada de função, são resgatados os valores dos parâmetros, que são armazenados na pilha. Logo após, é chamada a função e, por fim, os valores retornados são armazenados nas devidas posições na área de variáveis locais.
- Retorno das funções: toda função que possui retorno retorna uma lista de Object. Dessa forma, tanto funções multivaloradas quanto funções que possuem só um valor funcionam da mesma forma podendo o código ser agnóstico a isto.

3 Como compilar

Para compilar e executar (no modo interpretador) usando o programa Make, basta executar o seguinte comando no diretório do projeto:

```
make run <nome do arquivo lang>
```

Alternativamente, executar os seguintes comandos para compilar :

```
make compile
```

E o seguinte para executar:

```
java -cp .:antlr-4.8-complete.jar LangCompiler <nome do arquivo lang> <h>
```

O parâmetro h pode receber 3 valores, sendo:

- -i Roda o programa de forma interpretada
- -s Faz a compilação para código Java
- -j Faz a compilação para código Jasmin

OBS.: Os aquivos Java serão escritos na pasta JavaCodes e os Jasmin na pasta JasminCodes.

Para executar o código Jasmin, utilizar:

```
java -jar jasmin.jar -d JasminCodes/ JasminCodes/<nome do arquivo jasmin>
```

```
java -cp ./JasminCodes <nome do arquivo>
```

Para realizar todos os testes presentes na pasta testes/semantica com interpretador, executar:

```
./teste_interpretador.sh
```

Para realizar todos os testes presentes na pasta testes/semantica com compilação Java, executar:

```
./teste_java.sh
```

Para realizar todos os testes presentes na pasta testes/semantica com compilação Jasmin, executar:

```
./teste_jasmin.sh
```

O programa deverá ter as suas saídas apresentadas no terminal. A chamada do compilador no arquivo teste_interpretador.sh, teste_java.sh, teste_jasmin.sh possui uma *flag* -v, que faz com que seja impresso o nome do arquivo na saída, antes do resultado da execução, para facilitar a verifiação.