Geração de Código

Juan Felipe da S. Rangel[1]

[1]Universidade Tecnológica Federal do Paraná (UTFPR)
Caixa Postal 135 – 87.301-899 – Campo Mourão – PR – Brasil

juanrangel@alunos.utfpr.edu.br

Abstract. *In this article, the intermediate code generation process from the semantic analysis performed in the previous activity will be detailed. The strategies used will be discussed, as well as the tools deemed necessary to implement the intermediate code generation. Finally, we'll look at input and output code examples, taking a* `'.tpp'` *code as input.*

Resumo. *Neste artigo será detalhado o processo de geração de código intermediário a partir da análise semântica realizada na atividade anterior. Serão discutidas as estratégias utilizadas, assim como as ferramentas julgadas como necessárias para implementação da geração de código intermediário. Por fim, veremos exemplos de código de entrada e saída, tendo como entrada um código* `'.tpp'` *e como saída, um código* `'.ll'`*.*

1. Introdução

A geração de códigos intermediários representa o último passo na construção do nosso compilador para linguagem T++. Nesta fase foram utilizadas as bibliotecas `llvmlite` com o objetivo de auxiliar o processo de geração de código, podendo ser utilizado para criação de compiladores de diversas linguagens, porém, neste caso, o nosso objetivo é a criação de código para a linguagem definida no início da matéria de Compiladores, chamada de T++.

2. Detalhes da Implementação

Abaixo serão vistos alguns detalhes sobre a implementação utilizada para geração do código intermediário, como o tratamento de erros e sobre a biblioteca `'llvmlite'`

2.1. Tratamento de Erros

Como já dito, a geração de código representa o último passo na construção de um compilador, ou seja, neste passo, nenhum tratamento de erros ou avisos, baseados na definição da gramática da linguagem. O motivo para não haver nenhum tratamento de erros nesta fase é justamente porque esses tratamentos já forem realizados durante a análise semântica, ou seja, podemos concluir que caso algum erro seja encontrado durante a análise semântica, dado um código de entrada `'.tpp'`, a geração de código nem será executada.

2.2. Inicialização do módulo

O primeiro passo na geração de código é criar um módulo, já que é nele que as variáveis serão declarados ou manipuladas, e o mesmo acontece para as funções, chamadas de funções, declarações, os parâmetros dessas funções, e tudo que será gerado no código intermediário estará contido neste módulo. Abaixo podemos observar o processo de criação de um módulo.

Código 1. Iniciando e declarando o módulo

```
1
2       llvm.initialize()
3       llvm.initialize_all_targets()
4       llvm.initialize_native_target()
5       llvm.initialize_native_asmprinter()
6
7       # Cria modulo
8       modulo = ir.Module('modulo_geracao_cod.bc')
9       modulo.triple = llvm.get_process_triple()
10      target = llvm.Target.from_triple(modulo.triple)
11      target_machine = target.create_target_machine()
12      modulo.data_layout = target_machine.target_data
```

2.3. Processo de geração de código

Para geração dos códigos foi necessário realizar a utilização da árvore podada gerada na análise semântica ou também chamada de `ASTO`. Ela é utilizada para fazer a terceira passada ou a segunda passada, dependendo da implementação realizada, neste caso estamos realizando a terceira passada, possibilitando então a criação do um código executável.
Durante a terceira, é o momento onde as funções contidas no código `.tpp` são definidas e então relacionadas a um bloco de entrada e saída, assim como todas as operações relacionadas as declarações de variáveis e à declarações de funções.

3. Exemplos

Depois de já ter entendido melhor como foi o processo de desenvolvimento desta atividade, veremos a execução do teste `"gencode-001.tpp"`, o qual foi ofertado pelo próprio professor. Abaixo podemos ver o código `.tpp` que representa o código de entrada (Código 2).

Código 2. Código fonte tpp gencode-001.tpp

```
1  {Declaracao de variaveis}
2  inteiro: a
3
4  inteiro principal()
5    inteiro: b
6
7    a := 10
8
9    b := a
10
11     retorna(b)
12 fim
```

Podemos observar que há uma variável global declarada, e em seguida uma função principal, que como já discutido nas atividades anteriores, é uma função que todo código TPP deverá ter. Dentro desta função, a variável 'b' é declarada, sendo que em seguida, duas atribuições são feitas (para as variáveis 'a' e 'b'), retornando então o valor de 'b' ao final.

Primeiramente então é preciso executar o código `'geracao_cod.py'`. Utilizando o compilador `'clang'` é necesário realizar os comandos abaixo (Figura 3)

Por fim, podemos compilar o código gerado utilizando o `'clang'`, através dos comandos abaixo.

Código 3. Comando para rodar a geração de código

```
python3 geracao_cod.py geracao-codigo-testes/gencode-001.tpp --ntabela
```

Como saída teremos o código de máquina gerado (Figura 1).

```
-------------------------------------
Código gerado
; ModuleID = "modulo_geracao_cod.bc"
target triple = "x86_64-unknown-linux-gnu"
target datalayout = "e-m:e-p270:32:32-p271:32:32-p272:64:64-i64:64-f80:128-n8:16:32:64-S128"

@"a" = common global i32 0, align 4
define i32 @"main"()
{
entry:
  %"b" = alloca i32, align 4
  store i32 10, i32* @"a"
  %".3" = load i32, i32* @"a"
  store i32 %".3", i32* %"b"
  br label %"exit"
exit:
  %".6" = load i32, i32* %"b"
  ret i32 %".6"
}
```

Figura 1. Saída da geração de código.

Após rodarmos a geração de código é necessário compilar o código de máquina gerado, utilizando os comandos abaixo (Figura 2).

```
juan_rangel@LAPTOP-5H6E554J:~/Documentos/Compiladores/8.GeracaoCodigo/BCC__BCC36B__P[4]__JuanRangel__2046385/implementacao (main)$ clang modulo_geracao_cod.ll -o tpp_code_module
warning: overriding the module target triple with x86_64-pc-linux-gnu [-Woverride-module]
1 warning generated.
```

Figura 2. Compilação do código gerado.

Após compilar o módulo gerado é necessário executar o código gerado, e isso será feito através dos comandos a seguir, podendo observar que a saída obtida após executar o código foi '10', que é exatamente o que estávamos esperando (Figura 3.

```
juan_rangel@LAPTOP-5H6E554J:~/Documentos/Compiladores/8.GeracaoCodigo/BCC__B
CC36B__P[4]__JuanRangel__2046385/implementacao (main)$ ./tpp_code_module
juan_rangel@LAPTOP-5H6E554J:~/Documentos/Compiladores/8.GeracaoCodigo/BCC__B
CC36B__P[4]__JuanRangel__2046385/implementacao (main)$ echo $?
10
```

Figura 3. Comandos para execução do código gerado.

Referências

Gonçalves, R. A. Aula 18 – geração de código intermediário. ttps://moodle.utfpr.edu.br/pluginfile.pp/1112014/mod$_resource/content/4/aula-18-geracao-de-codigo-ir-llvm-ir.md.slides.pdf$.

Gonçalves, R. A. Aula 19 – otimizações de código intermediário. ttps://moodle.utfpr.edu.br/pluginfile.pp/280130/mod$_r esource/content/4/aula - 19 - otimizacoes - de - llvm - ir.md.slides.pdf$.

Gonçalves, R. A. (2021). Repositório llvm-gencode-samples. ttps://gitub.com/rogerioag/llvm-gencode-samples.