TP01 parte 3 - Análise Semântica e Geração de Código Intermediário

Larissa Aline Fernandes Vieira - 2019006868
Lucca Carvalho Augusto - 2019006930

Universidade Federal de Minas Gerais (UFMG)
Belo Horizonte - MG - Brasil

1. Introdução

Esse trabalho consistiu em implementar a última parte do compilador, sendo a análise semântica com geração do código intermediário, para a linguagem TIGER.

2. Implementação

A fim de implementar a parte restante do nosso compilador, nós nos apoiamos, principalmente, nas instruções dadas pelo livro *Modern compiler implementation in C*. Além disso, utilizamos de estruturas e exemplos baseados no livro citado anteriormente disponibilizados pela Universidade de Princeton.

Para a completude desse trabalho, fez-se necessária a criação de vários arquivos, os quais serão brevemente descritos a seguir.

- ➔ “absyn.h” e “absyn.c” foram criados para a implementação de funções sintáticas abstratas;
- ➔ “assem.h” e “assem.c” foram criados para a implementação de funções que fazem a tradução para instruções Assembly;
- ➔ “canon.h” e “canon.c” foram criados para a implementação de funções que convertem IRTrees (*Item Response Tree*) em traços e blocos básicos;
- ➔ “codegen.h” e “codegen.c” foram criados para a construção do código assembly;
- ➔ “color.h” e “color.c” foram criados para a implementação de funções que auxiliem no algoritmo de coloração para a alocação dos registradores;

- ➔ “env.h” e “env.c” foram criados para auxiliar na condução do ambiente durante a compilação;
- ➔ “errormsg.h” e “errormsg.c” foram criados para a implementação de funções que são usadas em todas as fases do compilador para o envio de mensagens de erro;
- ➔ “escape.h” e “escape.c” foram criados para achar as variáveis de escape;
- ➔ “flowgraph.h” e “flowgraph.c” foram criados para a implementação de funções que representam grafos de fluxo;
- ➔ “frame.h” define structs e funções temporárias auxiliares;
- ➔ “graph.h” e “graph.c” foram criados para a implementação de funções que manipulam e criam os grafos necessários;
- ➔ “liveness.h” e “liveness.c” foram criados para achar as variáveis que ainda estão vivas no programa;
- ➔ “main.c” foi criado para a administração geral do código, capturando a entrada e direcionando para os caminhos necessários;
- ➔ “parse.h” e “parse.c” foram criados para a utilização do *parser*;
- ➔ “prabsyn.h” e “prabsyn.c” foram criados para *printar* estruturas de dados da sintaxe abstrata;
- ➔ “printtree.h” e “printtree.c” foram criados para *printar* representações intermediárias das IRTrees;
- ➔ “regalloc.h” e “regalloc.c” foram criados para as funções que fazem a alocação de registros;
- ➔ “semant.h” e “semant.c” foram criados para as structs e funções que cuidam da semântica da linguagem;
- ➔ “symbol.h” e “symbol.c” foram criados para a implementação dos símbolos e tabelas de símbolos;
- ➔ “table.h” e “table.c” foram criadas para a implementação de tabelas *Hash*;
- ➔ “temp.h” e “temp.c” foram criados para a criação e manipulação de variáveis temporárias, as quais são usadas na IRTree antes da determinação de quais variáveis irão para os registradores;

- ➔ “tiger.lex” é o arquivo que antes usamos como “lexico.l” na primeira parte do trabalho com algumas alterações para se adequar ao novo formato;
- ➔ “tiger.y” foi criado a fim de implementar as regras gramaticais da linguagem em questão;
- ➔ “tree.h” e “tree.c” foram criados para auxiliar na representação intermediária de IRTrees;
- ➔ “types.h” e “types.c” foram criados para retornarem os tipos das estruturas utilizadas;
- ➔ “util.h” e “util.c” foram criados para a implementação de diferentes tipos de funções que são utilizadas;

3. Instruções de compilação e execução

A implementação do compilador foi feita em C, sua compilação é simples devido ao *Makefile* basta entrar na pasta *src* e digitar o comando *make*, então serão criados todos os arquivos intermediários **.o* e o executável *a.out*.

Para executar o compilador basta, em ambiente Linux, rodar o comando:

./a.out <EnderecoArquivoTigerDeEntrada>

como:

./a.out testcases/test1.tig

Caso nada seja printado no prompt de comando, significa que a compilação foi feita com sucesso e teremos um arquivo *testcases/test1.tig.s* com as instruções em assembly.

Caso algum erro seja encontrado durante o processo, uma mensagem de erro será retornada.

4. Resultados

Além dos arquivos para o pleno funcionamento do compilador, temos uma pasta *testcases* com três arquivos dentro, sendo o primeiro um teste com um programa correto, e os outros dois com diferentes erros.

No primeiro arquivo, como dito na Seção 3, a compilação é finalizada sem nenhum erro, portanto ao compilar o arquivo *testcases/test1.tig* obtemos o arquivo assembly *testcases/test1.tig.s* e nenhuma informação no prompt de comando.

No segundo arquivo temos a repetição da palavra *in* onde deveria ter apenas uma ocorrência, devido a este erro o compilador retorna a mensagem:

testcases/test2.tig:13.4: syntax error

Já no último arquivo teste temos variável da linha 7 escrita de forma errada, sendo o correto *rec1* e não *rec* como está no arquivo, devido a este erro o compilador retorna a mensagem:

a.out: escape.c:155: traverseVar: Assertion `0' failed.
Aborted

5. Conclusão

O compilador foi criado para identificar a linguagem TIGER e possíveis erros nos códigos feitos, com base nas informações oferecidas na especificação do trabalho.

Com a implementação desse trabalho, fomos capazes de entender melhor o funcionamento de um compilador e as estruturas que o compõem. Apesar disso, o conteúdo disponível sobre essas estruturas ainda é um tanto quanto escasso, o que pode ter sido um dificultador durante a implementação das funcionalidades.

Contudo, acreditamos que conseguimos alcançar resultados satisfatórios e cumprir com as especificações do trabalho.

6. Bibliografia

Exercícios feitos pela Universidade de Princeton
https://www.cs.princeton.edu/~appel/modern/c/

Livros:
Aho, A. V., R. Sethi, and J. D. Ullman. *"Compilers: Principles, Techniques, and Tools."* (1985);
Appel, Andrew W. *Modern compiler implementation in C*. Cambridge university press, 2004