

Especificação da Etapa 7 do Projeto de Compilador

Otimização

A sétima etapa do trabalho de implementação de um compilador para a **Linguagem IKS** consiste na implementação de técnicas de otimização de código intermediário. Para tal, utilizaremos como representação intermediária a **Linguagem ILOC**, descrita em detalhes no apêndice A de *Engineering a Compiler* [1], mas com o essencial descrito na definição da quinta etapa do projeto de compiladores.

A solução desta etapa deve estar organizada de forma modular. Ela deve ser aplicada sistematicamente sobre a saída gerada do compilador e da sequência de ativação, resultado das etapas 5 e 6. Além disso, ela deve poder funcionar de forma independente, dando como entrada um arquivo com código ILOC e gerando um outro arquivo com código ILOC otimizado. O professor utilizará esta segunda forma de funcionamento para avaliar o trabalho.

1 Funcionalidades Necessárias

Os resultados compreendem *a correção dos problemas encontrados na etapa anterior* e as funcionalidades seguintes:

1.1 Arquivos de entrada ILOC

A solução *deve vir acompanhada por 10 arquivos em formato ILOC* (arquivos com código ILOC devem terminar por *.i*). Os testes automáticos utilizarão estes programas durante o processo de avaliação, além dos códigos especificados pelo professor.

1.2 Módulo para entrada de arquivo ILOC

Crie um módulo para ler código na linguagem ILOC a partir de um arquivo. O módulo deve ser capaz de preencher a estrutura de dados para instruções ILOC implementada na etapa 5 e já utilizada na etapa 6. Por questões de simplicidade, assuma que o arquivo em ILOC, fornecido como argumento para o módulo, está sintaticamente e semanticamente correto.

1.3 Parâmetro opcional

Altere a função principal do compilador para que ela trate um arquivo de entrada com linguagem ILOC, utilizando o módulo do item 1.2 desta especificação para carregá-lo em memória. Sendo assim, o compilador terá dois comportamentos:

- Leitura da *entrada padrão* (*stdin*) de um programa em Linguagem **IKS**.
- Leitura de arquivo em linguagem ILOC mediante passagem do nome do arquivo como argumento.

O compilador deve sempre gerar como saída um arquivo em linguagem ILOC na *saída padrão* (*stdout*). Caso a entrada seja feita pela entrada padrão em Linguagem **IKS**, o compilador deve fazer a análise léxica, sintática e semântica, gerando código ILOC otimizado no final. Caso a entrada seja um nome de arquivo em linguagem ILOC, o compilador deve somente realizar as otimizações sobre esta representação intermediária, imprimindo na saída padrão o código ILOC otimizado que é equivalente semanticamente ao código ILOC da entrada.

1.4 Módulo de Otimização

Crie um módulo de otimização dentro do compilador capaz de receber uma estrutura de dados com código ILOC como entrada e retornar uma outra estrutura com código ILOC otimizado. O retorno pode ser feito somente através de uma alteração do código de entrada, a critério do grupo. Note que o módulo de otimização pode ser invocado várias vezes para que várias passagens de otimização possam ser feitas no código.

1.5 Técnicas de Otimização

Este é o primeiro tópico principal desta etapa. Implemente uma série de técnicas de otimização de janela (*peephole optimization*)¹. O tamanho da janela deve ser parametrizável através de um argumento para o programa, sendo que o tamanho padrão de janela é de duas instruções. O passo de avanço da janela é sempre de uma instrução. As seguintes técnicas devem ser implementadas:

- Eliminação de instrução redundante
- Otimizações de fluxo de controle
- Simplificações algébricas
- Uso de idiomas de máquina²
- Avaliação de constantes
- Propagação de cópias

1.6 Definição da Heurística

Este é o segundo tópico principal desta etapa. O grupo deve estabelecer uma heurística de otimização utilizando as técnicas implementadas. Esta heurística deve ter as seguintes definições:

- Quantidade de passagens de otimização
- Ordem da aplicação das técnicas do item 1.5
- Tamanho da janela em cada passagem

Fica a critério do grupo definir diferentes níveis de otimização, seguindo o estilo dos compiladores mais comuns (-O0, -O1, -O2 e -O3, por exemplo). *Pelo menos um nível de otimização deve ser implementado.*

2 Entrada e Saída Padrão

Organize a sua solução para que o compilador leia o programa em **IKS** da entrada padrão e gere o programa em ILOC na saída padrão. Dessa forma, pode-se realizar o seguinte comando (considerando que **main** é o binário do compilador):

```
./main < entrada.k > saida.i
```

Onde **entrada.k** contém um programa em **IKS**, e **saida.i** contém o programa em ILOC correspondente.

Adicionalmente, e como detalhado nas funcionalidades necessárias, organize a sua solução para que o compilador seja capaz de ter um comportamento alternativo quando somente as técnicas de otimização deve ser aplicadas. Sendo assim, pode-se realizar o seguinte comando:

```
./main entrada.i > saida.i
```

Note que o nome de arquivo **entrada.i** é passado como argumento para o programa, e não o seu conteúdo.

3 Controle e Organização da Solução

A função **main** deve estar em um arquivo chamado **main.c**. Outros arquivos fontes são encorajados de forma a manter a modularidade do código fonte. A entrada para o *bison* deve estar em um arquivo com o nome **parser.y**. A entrada para o *flex* deve estar em um arquivo com o nome **scanner.l**.

¹Ver Aula 25.

²Existem duas instruções ILOC adicionais, **inc** e **dec**, que recebem um registrador, e incrementam e decrementam seu valor de 1.

3.1 Git e Cmake

A solução desta etapa do projeto de compiladores deve ser feita sobre a etapa anterior. Cada ação de commit deve vir com mensagens significativas explicando a mudança feita. Todos os membros do grupo devem ter feito ações de commit, pelo fato deste trabalho ser colaborativo. Estas duas ações – mensagens de commit e quem fez o commit – serão obtidas pelo professor através do comando `git log` na raiz do repositório solução do grupo.

A solução do grupo deve partir do código da etapa anterior do projeto de compilador do mesmo grupo. Novos arquivos de código fonte podem ser adicionados, modificando o arquivo `CMakeLists.txt`, para que ele seja incluído no processo de compilação do analisador sintático.

3.2 Documentação do Código e Testes

Todas as funções devem estar documentadas. A escolha do sistema de documentação fica a critério do grupo e será igualmente avaliada. Uma opção é utilizar `doxygen`.

4 Atualizações e Dicas

Verifique regularmente o Moodle da disciplina e o final deste documento para informar-se de alguma eventual atualização que se faça necessária ou dicas sobre estratégias que o ajudem a resolver problemas particulares. Em caso de dúvida, não hesite em consultar o professor.

Referências

- [1] Keith D. Cooper and Linda Torczon. *Engineering a Compiler*. Morgan Kaufmann, 2nd edition, 2012.