Construção Das Etapas De Compilação De Uma Linguagem De Programação

Alexsandro Meurer Schneider
Ismael Bortoluzzi

Curso de Ciências da Computação – Universidade Federal da Fronteira Sul (UFFS)
Chapecó-SC, Brasil

RESUMO

Compilação de uma linguagem de programação qualquer se dá através da tradução de um arquivo de texto contendo código-fonte válido da linguagem, para um arquivo de código intermediário, comumente chamado de *bytecode*, ou para um arquivo executável no formato de linguagem de máquina. Esse processo é dividido em várias etapas, como análise léxica, análise sintática, análise semântica, geração de código intermediário, otimização de código, geração de código objeto e geração de código de máquina. O processo de compilação é fundamental para o desenvolvimento de software e é aplicado em todas as linguagens de programação, algumas gerando apenas o código intermediário, outras o código de máquina.

1 INTRODUÇÃO

Compiladores são programas que traduzem códigos-fonte escritos em linguagem de programação para código executável em linguagem de máquina. Eles são essenciais para o desenvolvimento de softwares pois permitem que os programadores escrevam em linguagem de alto nível, mais próxima da linguagem humana, e depois traduzam esses códigos para a linguagem de máquina que o computador entende, ou para um código intermediário capaz de ser interpretado por um interpretador.

O objetivo deste trabalho é projetar e implementar as etapas de compilação para uma linguagem de programação hipotética interpretada, que será definida por um conjunto de regras e produções apresentada na sessão Desenvolvimento. Isso envolverá

a geração do autômato finito da gramática, criação de um analisador léxico, um analisador sintático, um analisador semântico, um gerador de código intermediário e sua otimização, que juntos irão traduzir o código-fonte para código intermediário passível de ser interpretado.

2 REFERENCIAL TEÓRICO

Um compilador é um programa responsável por transformar o código-fonte de uma linguagem de programação em um código executável em uma plataforma específica, ou em um código intermediário capaz de ser interpretado por um software interpretador. Ele é responsável por realizar uma série de etapas, que incluem análise léxica, análise sintática, análise semântica, geração de código intermediário e otimização de código.

A análise léxica é a primeira etapa de um compilador e é responsável por transformar o código-fonte em uma sequência de *tokens,* armazenados numa estrutura chamada “fita de saída” e colocar as variáveis na tabela de símbolos. Os *tokens* são unidades básicas da linguagem de programação, como palavras-chave, identificadores, operadores e símbolos especiais.

A análise sintática é responsável por verificar se a sequência de *tokens* está de acordo com a gramática da linguagem de programação. Ou seja, se conjunto de *tokens* que o *lexer* colocou na fita de saída geram uma cadeia válida no alfabeto da linguagem formal da linguagem de programação.

A análise semântica é responsável por verificar se o programa está correto do ponto de vista semântico, ou seja, se as operações estão sendo realizadas de acordo com as regras da linguagem de programação.

A geração de código intermediário é a etapa em que são gerados códigos de três endereços, que podem ser interpretados por uma máquina virtual, comumente chamada de interpretador.

A otimização de código é a etapa responsável por melhorar o código gerado, de forma a torná-lo mais eficiente e reduzir o tempo de execução do programa. Este objetivo é atingido através da construção de um grafo acíclico dirigido e posterior aplicação de um algoritmo de otimização, que reduz a quantidade de instruções de até três operandos necessárias para executar o código.

Existem diversas técnicas e teoremas fundamentais utilizados no desenvolvimento de compiladores, como a gramática livre de contexto, o algoritmo LR, a árvore de análise

sintática, entre outros. É importante entender esses conceitos para o desenvolvimento de um compilador eficiente e confiável.

3 IMPLEMENTAÇÃO

3.1 Análise Léxica

A função de análise léxica percorre as linhas do arquivo uma por uma, lendo cada caractere escrito, iniciando pela regra inicial e a variável que vai construindo o token da gramática, denominada *string*, vazia.

Caso lemos um operador e a *string* não está vazia, o último caractere não é um operador e a regra do caractere lido é um dos estados finais, adicionamos a regra na fita de saída. Porém se o último caractere é um operador, adiciona em *string* o caractere e continuamos lendo a cadeia.

Caso lemos um separador (chave, ponto e vírgula, etc.) e o estado do caractere é um estado final, adicionamos na tabela de símbolos e na fita de saída.

No caso de lermos um espaçador (espaço, quebra de linha, etc.) apenas continuamos. Se não for um separador, operador e já existe algo na *string,* apenas o adicionamos na *string.* Senão o adicionamos na tabela de símbolos e fita de saída.

3.2 Análise Sintática

A função análise sintática consome a fita de saída que foi gerada durante a análise léxica. Neste trabalho foi utilizada a tabela de *Parsing* gerada pelo programa *GOLD Parser Builder* para a Gramática Livre de Contexto desenvolvida, facilitando a identificação das ações de empilhamento (shift), redução (reduce), salto (goto) e aceite.

O analisador sintático obtém sequencialmente os *tokens* presentes na fita de saída, e partindo do estado inicial da pilha, verifica na tabela de *Parsing* qual ação executar. Caso seja identificado uma ação de empilhamento, o *token* e o próximo estado são empilhados, seguido da obtenção do próximo *token*.

Em uma ação de redução, a tabela de *Parsing* identifica qual o código da produção a ser reduzida. O analisador obtém o número de símbolos da produção e desempilha o dobro desse valor da pilha. Em seguida o analisador verifica qual o estado restante no topo da pilha, e, transacionando através do símbolo que dá nome à produção reduzida, efetua a ação de salto, empilhando esse símbolo e também o próximo estado. É durante a

execução da redução que pode ocorrer uma operação de tradução dirigida por sintaxe, caso haja uma ação associada à produção reduzida.

A ação de aceite ocorre apenas após a redução de todas as produções restantes na pilha. Quando o analisador sintático obtém o token $ (EOF) da fita de saída, o mesmo verifica o estado presente no topo da pilha e, transacionando pelo símbolo de final de sentença, identifica a ação de aceite.

Caso uma transição não possua ação associada, identifica-se um erro sintático no código-fonte.

3.3 Análise Semântica

A função de análise semântica irá varrer a tabela de símbolos procurando por produções *def* para anotar o nome da variável que foi definida, além de verificar chamadas a variáveis ao longo do código. Ao localizar a utilização de uma variável ainda não definida, a função identifica um erro semântico na linha do código-fonte onde a chamada ocorreu.

3.4 Geração de código intermediário

A geração de código intermediário ocorre durante a aplicação da função de tradução dirigida por sintaxe, utilizando um esquema de tradução que associa ações semânticas às produções. Através das regras definidas são gerados atributos para as produções que forem reduzidas, como a identificação de um tipo de variável, ou a anotação de seu nome ou de seu valor. O código intermediário é gerado através da transformação das operações matemáticas em códigos de três endereços, que envolvem variáveis temporárias, o operador e os operandos.

3.5 Otimização de código

A função de otimização de código utiliza os códigos intermediários gerados na etapa anterior. É criado um grafo acíclico dirigido, no formato matriz de adjacência, para as variáveis temporárias, que representam as operações matemáticas. Desse modo é possível identificar outras operações que utilizem essas variáveis. O algoritmo de otimização efetua a verificação de todas as variáveis, buscando na matriz de adjacência se há alguma outra operação que o utilize. Se houver, é executada uma nova verificação

sobre essa operação. Quando não houverem outras operações utilizando a variável, ou seus pais já estiveram na lista, então essa variável pode ser inserida na lista. Ao concluir a verificação de todas as variáveis, é obtida uma sequência otimizada para a execução das operações matemáticas.

4 RESULTADOS

A implementação dos analisadores léxico, sintático e semântico, juntamente com o gerador de código intermediário e o otimizador de código possibilitam a execução de códigos-fonte escritos com base na Gramática Livre de Contexto (GLC) que definimos. No entanto, há limitações na tradução dirigida por sintaxe, pois criamos esquemas de tradução para apenas algumas produções específicas. Além disso, o código é otimizado em cada produção separada devido às limitações na implementação do Grafo Acíclico Dirigido e algoritmo de otimização. Porém, para fim de demonstração das etapas de compilação, os componentes desenvolvidos funcionam de maneira esperada.

CONCLUSÃO

Compiladores são ferramentas fundamentais no desenvolvimento de software, permitindo a conversão de código de linguagens de alto nível para código intermediário e executável de máquina. O processo de compilação envolve diversas etapas, como análise léxica, sintática, semântica, otimização e geração de código, sendo cada uma delas essencial para garantir a correta conversão do código fonte. Apesar de ser um processo complexo, existem técnicas e ferramentas disponíveis para tornar a construção de compiladores mais eficiente e eficaz, incluindo geradores de analisador sintático e linguagens específicas de domínio. Compreender os fundamentos dos compiladores e suas técnicas permite ao desenvolvedor criar software mais seguro e eficiente, melhorando a qualidade do código produzido.

Um ponto que pode ser melhorado neste curso para aprimorar o processo de aprendizagem seria acelerar o conteúdo inicial, para que tenhamos mais tempo para desenvolver as etapas de geração de código intermediário e otimização do mesmo.