Construção de Compiladores

Amanda Faria e Victor Augustus Lopes Costa

Departamento de Ciência da Computação - Universidade Tecnológica Federal do Paraná - Medianeira - PR - Brasil

amandafaria@alunos.utfpr.edu.br e victorc.2019@alunos.utfpr.edu.br

Resumo. *Este meta-artigo descreve o estilo a ser usado na confecção de artigos e resumos de artigos para publicação nos anais das conferências organizadas pela SBC. É solicitada a escrita de resumo e abstract apenas para os artigos escritos em português.*

1. Introdução

Os compiladores desempenham um papel inestimável no mundo da programação, agindo como tradutores que transformam códigos escritos por humanos em linguagens que as máquinas podem compreender. Esses poderosos programas, enquanto traduzem a linguagem de programação de alto nível para a linguagem de máquina, passam por várias etapas complexas de processamento para garantir que o código seja interpretado corretamente. Este estudo se aprofundará nas intricadas etapas de compilação, com ênfase especial nas análises léxica, sintática e semântica, ilustrando a vitalidade dessas fases e como elas influenciam o resultado final: um programa executável.

2. Definição de Compiladores

Um compilador consiste em um programa em que ele irá ler uma linguagem de programação de um código escrito (código fonte) e traduzir esse código em linguagem de máquina, ou como podemos dizer, uma linguagem de código intermediário. Sua principal função é transformar o código fonte em um código de

máquina ou de bytes, que será executado por um sistema operacional diretamente. Ele é fundamental para o desenvolvimento do programa, pois os programadores podem escrever um código em linguagem de alto nível e legível.

Existem diversas etapas de processamento que são realizadas pelos compiladores que tem o intuito de converter o código fonte em código executável. Essas etapas consistem em:

Análise Léxica, onde há uma quebra do código fonte em tokens, que são unidades menores.

Análise Sintática, ela organiza em uma estrutura hierárquica todos os tokens, onde chamamos de árvore sintática, que representa a estrutura gramatical do código.

Análise Semântica, que realiza a verificação do programa, para confirmar se ele é semanticamente correto.
Geração de Código Intermediário, onde se produz uma representação intermediária do código fonte.

Otimização de Código, é quando se melhora a eficiência do código intermediário.
Geração de Código, nessa etapa é realizada a tradução do código intermediário na linguagem alvo.

Ligação (linking), se o programa contiver múltiplas partes ou usar bibliotecas, essas diferentes partes são ligadas juntas para criar um único executável.

Mais a frente iremos falar um pouco mais sobre as análises léxica, sintática e semântica.

3. Descrição das Etapas

3.1. Análise Léxica

É a primeira etapa do processo de compilação, sua principal função é ler o código-fonte, no qual está em uma linguagem de alto nível, e dividi-la em unidades básicas chamadas "tokens". Esses tokens são representações de elementos individuais da linguagem, isto é, são palavras-chave, identificadores, números, operadores e símbolos especiais. É a partir dele que o programa consegue identificar em que linha está, facilitando o computador a entender o que o programa deseja fazer e como fazer.

Por consequência é possível identificar algumas das principais tarefas realizadas durante a análise léxica:

- Leitura do código-fonte: O analisador léxico lê caractere por caractere, da esquerda para a direita, enquanto ignora espaços em branco e comentários. Isso ocorre para que o computador entenda o que fazer, resumidamente ele diz ao programa aonde ir.
- Reconhecimento de palavras-chave: Durante a leitura do programa, o analisador léxico consegue identificar palavras-chave reservadas, como "if", "for", "while", "int", entre outras. Essas palavras têm significados especiais na linguagem e são tratadas de maneira distinta.
- Identificação de identificadores: Os identificadores são nomes que são dados pelos próprios programadores para poder identificar variáveis e funções. O analisador vai coletar essas informações e gerar tokens para que eles possam ser identificados no programa.
- Análise de números: Nessa fase, são reconhecidas as variáveis como os números inteiros, os pontos flutuantes, valores binários, entre outras formas de valores numéricos no código-fonte, convertendo-os em tokens apropriados.
- Tratamento de operadores e símbolos: Operadores, símbolos especiais e pontuação são identificados e transformados em tokens correspondentes. Ou seja, "+" é um token para o operador de adição, "=" para o operador de atribuição, "{" para o início de um bloco de código, etc.
- Manejo de Literais: Literais, como strings e caracteres, são reconhecidos e representados como tokens separados. Isto ocorre para que todos consigam entender o que está escrito no programa.
- Contagem de linhas e colunas: Durante a análise léxica, o analisador também pode manter o controle das informações de localização no código-fonte, incluindo o número de linha e coluna em que cada token é encontrado. Isso ajuda a relatar erros com precisão.

Em conclusão, a análise léxica desempenha um papel fundamental no processo de compilação de programas de computador. Sendo assim, ele vai ler o código-fonte, identificar palavras-chave, números, identificadores e caracteres especiais, e organizá-los em pequenos pedaços chamados de tokens. Isso ajuda o computador a entender as linhas de códigos criadas pelo programador. A análise léxica acaba sendo o primeiro passo para que um programa de computador possa ser transformado em uma linguagem que possa ser compreendida pelas pessoas, ou seja, uma linguagem de alto-nível.

3.2. Análise Semântica

Essa etapa do processo de compilação geralmente ocorre depois da análise léxica e da análise sintática. Ela consiste na interpretação do significado e na

verificação da lógica do código-fonte, indo além da estrutura gramatical e sintaxe da linguagem.

Ou seja, a análise semântica vai ficar responsável por analisar o código e verificar se ele faz sentido. Ela vai verificar se as variáveis e números utilizados no programa não irão fazer o que não é permitido, como por exemplo, somar palavras com números. Além disso, ela garante que o código esteja seguindo todas as regras da linguagem de programação, para que ele funcione corretamente quando for executado. Para isso, existem algumas tarefas que são realizadas durante a análise semântica, como:

- Verificação de tipos: Faz a verificação se os tipos de dados usados no código são consistentes e fazem sentido. Por exemplo, ela verifica se você não está tentando somar uma string com um número, o que poderia causar um erro.
- Escopo de variáveis: Ela acompanha como as variáveis são definidas e usadas no programa, garantindo que as variáveis sejam acessadas apenas onde estão definidas e dentro das regras de escopo da linguagem.
- Resolução de nomes: Garante que nomes de variáveis, funções e outros identificadores sejam resolvidos corretamente, para que o compilador saiba a que eles se referem no contexto do programa.
- Verificação de semântica específica da linguagem: Cada linguagem de programação pode ter regras semânticas específicas. A análise semântica verifica se o código segue essas regras. Por exemplo, em algumas linguagens, pode haver restrições sobre como os loops devem ser usados.
- Otimização de código: Ao aplicar a otimização nos códigos é possível buscar maneiras mais eficientes de realizar as operações específicas.
- Gestão de memória: Em linguagens que gerenciam manualmente a memória, a análise semântica pode garantir que as alocações e as desalocações de memória sejam feitas corretamente.

Desta forma, a análise semântica é uma parte crucial do processo de desenvolvimento de software e compilação de programas. Ela desempenha o papel de um "inspetor de regras" que verifica se o código de programação faz sentido, seguindo as regras da linguagem e evitando erros lógicos. Ela garante que o código seja coerente e seguro para ser executado em um computador. Por isso, é uma etapa essencial para garantir a qualidade e o desempenho dos programas de software.

3.3. Análise Sintática

A Análise Sintática, ou como podemos chamá-la, “parsing”, é uma etapa do processo de compilação que é fundamental. Ela é responsável pela verificação de sequência de tokens produzida pela análise léxica, validando se ela segue a

gramática que foi definida para a linguagem de programação. Seu principal objetivo é construir uma representação estrutural do código fonte, onde geralmente está representado na forma de uma árvore de análise sintática.

A árvore de análise sintática representa a estrutura gramatical do código fonte. Os nodos dessa árvore são correspondentes a construção da linguagem – como declarações, expressões e comandos – enquanto as folhas correspondem aos tokens.

Dentro da análise sintática, existem algumas técnicas para realizá-la. As mais comuns são as “top-down” e “bottom-up”.
A “top-down” começa pelo símbolo inicial da gramática e tenta realizar a derivação da sentença. Um método comum nele é o LL, onde é lido a entrada da esquerda para a direita e produz uma derivação mais à esquerda.
O “bottom-up” começa pelos tokens e tenta construir a árvore de análise sintática para que chegue até o símbolo inicial da gramática. O método que é mais utilizado nele é o LR, onde é lido a entrada da esquerda para a direita e produz uma derivação mais à direita.

Na fase de análise sintática, quando erros são detectados, eles são categorizados como erros sintáticos. Um exemplo clássico é quando um programador não fecha um parêntese, gerando assim uma falha sintática. É essencial que compiladores eficientes ofereçam feedbacks claros e úteis para que os programadores possam identificar e resolver esses problemas.

Por outro lado, em linguagens de programação, certas estruturas podem ter mais de uma interpretação, criando ambiguidades. Um exemplo disso é a expressão "a - b - c", que em algumas linguagens pode ser lida tanto como "(a - b) - c" quanto "a - (b - c)". Para lidar com tais incertezas, o analisador de código (parser) recorre frequentemente a regras específicas, como a ordem de precedência ou a associatividade das operações.

Na etapa de análise sintática, mesmo que a principal intenção seja criar a estrutura hierárquica do código, chamada de árvore de análise sintática, há outros processos em jogo. Durante essa análise, outras tarefas relacionadas ao significado e à lógica do código, como a formação de tabelas de símbolos ou a verificação dos tipos de variáveis, são simultaneamente realizadas.

Desta forma, a análise sintática é uma área complexa e vital para entender como os compiladores interpretam programas. Essa fase apresenta diversas especificidades e particularidades, e os estudos neste domínio seguem avançando para se adaptar a linguagens e regras gramaticais crescentemente intrincadas.

4. **Backus-Naur**

<COMECA> ::= CHAMADA <IDENTIFICADOR> ; <BLOCO> <TERMINA> ::= VIRADADEDOIS

<IDENTIFICADOR> ::= <LETRA> { <LETRA> | <NUMERICO> }
<NUMERO> ::= <NUMERICO> { <NUMERICO> }
<NUMERICO> ::= 0|1|2|3|4|5|6|7|8|9
<LETRA> ::= a|b|c|d|e|f|g|h|i|j|k|l|m|n|o|p|q|r|s|t|u|v|w|x|y|z|A|B|C|D|E|
F|G|H|I|J|K|L|M|N|O|P|Q|R|S|T|U|V|W|X|Y|Z

<bloco> ::= <declarações> >> <comando> << | >> <comando> <<
<declarações> ::= <tipo> <lista_identificadores> ;
<lista_identificadores> ::= <IDENTIFICADOR> ;| <IDENTIFICADOR>, <lista_identificadores> ;
<tipo> ::= TAMBORIM | CAIXA | CHOCALHO
<comando> ::= <variável> <- <expressão> ; <comando>
| MESTRE <expressão> TOQUE <comando> ESPERE <comando>
| CONTAGEM <identificador> <- <expressão> PARADA <expressão><comando>
| BOSSA <EXPRESSAO> TOQUEBOSSA <COMANDO>
| >> <comando> <<
<expressão> ::= <variável>
| <identificador>
| <expressão> <operador> <expressão>
<variável> ::= <identificador>
<operador> ::= SUBIDA| DESCIDA | DOBRA | CONTRA | LIMPO | ACELERA | CAI | SUJO | CAILIMPO | ACELERALIMPO

Palavras Reservadas
CHAMADA: Começa o programa e, imediatamente após essa instrução, deve existir um identificador referente ao nome do seu software.
MESTRE: início declaração de desvio condicional;
TOQUE: a partir dessa declaração, ocorre o desvio caso a expressão condicional seja verdadeira;
ESPERE: a para partir dessa declaração, ocorre o desvio caso a expressão condicional não seja verdadeira;
CONTAGEM: indica a declaração de um laço de repetição contado;
PARADA: anterior a essa declaração está uma expressão, e esta deve ser acrescida ou decrementada até ficar igual a expressão posterior a esta declaração;
BOSSA: indica a declaração de um laço de repetição com base na expressão;
TOQUEBOSSA: indica o comando a ser executado caso a expressão seja verdadeira;
>>: delimitador de bloco inicial;

<<: final de delimitador de bloco;
OPERADORES:

- SUBIDA : soma;
- DESCIDA : subtração;
- DOBRA : multiplicação;
- CONTRA : divisão;
- LIMPO : comparação de igualdade;
- ACELERA : comparação maior;
- CAI : comparação menor;
- SUJO : diferente;
- CAILIMPO : comparação menor ou igual;
- ACELERALIMPO : comparação maior ou igual;
- <- : atribuição;
- ;: final de linha de instrução.

TIPAGEM:

- TAMBORIM: Tipo inteiro;
- CAIXA: Tipo float;
- CHOCALHO: varchar;

5. Conclusão:

A complexidade e a sofisticação dos compiladores tornam-se evidentes ao se compreender as diversas fases envolvidas na transformação do código fonte em um programa executável. As etapas de análise léxica, sintática e semântica, em particular, são fundamentais para garantir que o código seja interpretado conforme a intenção do programador e sem erros lógicos. Essas etapas asseguram que, desde a identificação dos menores elementos do código até a construção de sua estrutura hierárquica, tudo esteja alinhado à gramática e às regras semânticas da linguagem de programação. Ao entender essas fases e sua importância, ganhamos uma apreciação mais profunda pelo trabalho meticuloso que os compiladores realizam e pela arte e ciência da programação de computadores.