TopScript

Vinicios Henrique Wentz[1], Rafael Henrique Tibola[1], Felipe Divensi[1]

[1]Universidade Tecnologica Federal do Paraná (UTFPR)
Av. Brasil, 4232 - Independência, Medianeira - PR, 85884-000

{vinicioswentz, tibola, felipee}@alunos.utfpr.edu.br

Abstract. *aaaa*

Resumo. *aaa*

1. Compilador

Linguagens de programação são notações para a descrição de computações para pessoas e maquinas. Hoje, no mundo em que vivemos as pessoas são totalmente dependente das linguagens de programação, pois, todos os software os quais os computadores rodam são escritos em alguma linguagem de programação. Mas para que isso seja possível, precisa-se de alguma forma que os computadores entendam essas linguagens de programação, software que faz está tradução é chamado de compilador [Alfred V. Aho 2007].

Pode-se definir compiladores como programas que fazem a tradução de outros programas os quais foram escritos em alguma linguagem e os preparam para serem executados. Para fazer está transformação o compilador deve entender [K. Cooper 2012].

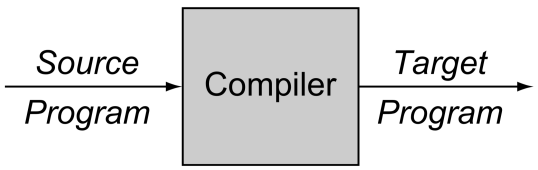

Figure 1. Compilador.

A estrutura de um compilador é compostas por duas partes a parte de analise e a de síntese. A fase de analise quebra o programa fonte em peças constituintes é impões uma estrutura gramatical. Essa estrutura é utilizada para a criação da representação intermediária do programa fonte. Caso a fase de analise detecte que o programa está sintaticamente mal formado ou não está semanticamente claro, essa fase deve prover informações informativas para que o usuário tome as devidas providencias. A analise também coleta algumas informações importante e as armazenas em uma estrutura chamada tabela de símbolo, a qual é passada com a representação intermediária para a analise sintática. A analise sintática é responsável pela construção do programa alvo desejado. Ela usa a representação intermediária e a tabela se símbolo para tal processo [Alfred V. Aho 2007]. Essas fases são chamadas de *front end* e *back end* respectivamente. A fase será abordada com maior detalhes nas Seções 2, 4 e 3 e a síntese na Seção.

2. Analise Léxica

A tarefa principal do analisador léxico na primeira fase da compilação é de ler o programa fonte, agrupa-los em lexemes e produzir uma sequencia de *tokens* para cada lexeme no programa fonte. Muitas vezes o analisador léxico interage com a tabela de símbolos. Quando o analisador léxico processa um lexeme que pertence a um identificador ele precisa adiciona-lo na tabela de símbolos. Em alguns casos informações sobre algum identificador deve ser lido da tabela de símbolos pelo analisador lexico para determinar o *token* apropriado que deverá ser passado para o *parser*. [Alfred V. Aho 2007]. Essa iteração é demonstrada na Figura 2.

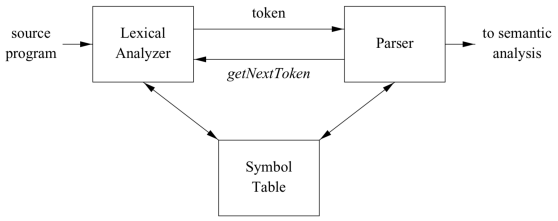

Figure 2. Iterações entre o analisador léxico e o *parser*.

3. Analise Sintática

O analisador sintático conhecido também como *parser* obtêm *strings* dos tokens que do analisador léxico, ele verifica se os nomes dos tokens podem ser gerados pela gramatica da linguagem fonte. É esperado que o *parser* informe qualquer erro de sintaxe. Para programas bem formados, o *parser* faz a construção da arvore sintática e passa para as próximas fases do compilador [Alfred V. Aho 2007].

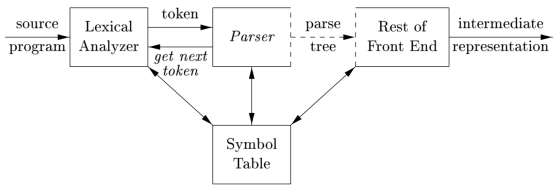

Figure 3. Iterações entre o analisador léxico e o *parser*.

Existem três tipos de analisadores de gramatica, são eles, universal, *top-down* e *bottom-up*. O analisador universal pode fazer a analisa de qualquer gramatica, porem esse método é ineficiente em produção.

Os métodos mais comuns para fazer a analise sintática são os analisadores *top-down* e *bottom-up*. Os seus nomes são bem sugestivos, o analisador top-down faz a construção da arvore sintática do topo para baixo, ou seja, começa a construção a partir da raiz e termina nas folhas, o analisador bottom-up começa essa construção nas folhas indo em direção até a raiz. Em ambos os casos as analises são feitas da esquerda pra direita um simbolo por vez.

Os analisadores top-down e bottom-up trabalham somente com uma sub-classe de linguagens, porem as sub-classes LL e LR são expressivas o bastante para a descrição

da grande parte das construções sintáticas em linguagens de programação modernas [Alfred V. Aho 2007].

4. Analise Semântica

Basicamente a fase de analise semântica é quando o compilador conecta as definições de variáveis a seus usos, faz a checagem de tipo de cada expressão e faz a tradução da arvore sintática em uma representação mais simples para a geração de código de maquina [K. Cooper 2012].

No processo desta tradução do programa o compilador pode construir uma sequencia de representações intermediarias como visto na Figura 4. Uma representação de alto-nivel é proximo a linguagem utilizada no programa fonte. As arvores sintaticas são uma representação de alto-nivel, elas retratam uma estrutura hierarquica natural da estrutura do programa fonte assim como as tarefas de checagem de tipo

Figure 4. Sequencia de representações intermediarias.

A escolha do design do analisador semântico varia de compilador pra compilador. A fase de representação intermediaria pode ser uma linguagem real ou pode consistir em uma estrutura de dados internas que são compartilhadas por fases do compilador. Por exemplo. a linguagem de programação C ainda é frequentemente utilizada como forma intermediaria, por ser flexível e compila de forma eficiente para código de maquiná e ainda ter um compilador disponível para todos os sistemas operacionais [Alfred V. Aho 2007].

5. Documentação

References

Alfred V. Aho, Monica S. Lam, R. S. J. D. U. (2007). *Compilers: principles, techniques, and tools*. Pearson/Addison Wesley, 2 edition.

K. Cooper, L. T. (2012). *Engineering a Compiler*. Morgan Kaufman, 2 edition.