FACULDADE DE ENGENHARIA DE SOROCABA
CAMPUS ALEXANDRE BELDI NETTO
Reconhecida pela Portaria Ministerial n° 367 de 03/06/1980
MANTIDA PELA
Associação Cultural de Renovação Tecnológica Sorocabana
Declarada de Utilidade Pública Federal - Decreto n° 86.431 de 02/10/1981
Declarada de Utilidade Pública Municipal - Lei n° 1.842 de 04/12/1975

TCC 2015 – Engenharia da Computação

1º CAPÍTULO TEÓRICO

IDENTIFICAÇÃO

Nº	NOME
111693	**Rodrigo Vieira da Silva**

e-mails	Fone / Cel.
FACENS: 111693@li.facens.br	**15 3213-2014**
particular: rodvieirasilva@gmail.com	**15 9 9777-1897**

TÍTULO: Framework para construção de compiladores com conceitos Fuzzy

ORIENTADOR: Marcos Maurício Lombardi Pellini Fernandes

Data da Entrega: / /2015

_____ _____

Visto do Orientador **Profª. Andréa**

(Verificado em ___/___/_____)

SUMÁRIO

2 PRINCÍPIOS DE COMPILADORES

De forma abrangente um compilador converte um código fonte em um código destino. Os princípios e técnicas de construção de compiladores são utilizadas em diversas áreas de aplicação do conhecimento de um profissional envolvido com as disciplinas de computação. Com o aprendizado de técnicas básicas é possível abordar uma grande variedade de problemas quanto a tradutores de linguagens e máquinas. (AHO, LAN E ULLMAN, 1995)

Este capítulo abordará algumas dessa técnicas e conceitos que foram utilizados para o desenvolvimento desse projeto.

2.1 Conceitos

Um compilador é essencial para maior eficiência na construção de um software, por possuir etapas bem definidas é possível a sua modularização e generalização em diversos itens como demonstra a figura 2.1. (AHO, LAN E ULLMAN,1995)

Figura 2.1 – Etapas de um compilador

Fonte: AHO, LAN E ULLMAN (1995)

2.1.1 Análise Léxica

Análise Léxica, análise linear ou ainda scanning (esquadrinhamento) em um compilador é responsável por ler o código fonte e converter em um fluxo de tokens (palavras e tipos que compõe o texto) que será propagado para a próxima etapa, ou seja, recebe uma sequência de caracteres e gera uma lista sequencial de palavras chaves, pontuação e nomes ignorando comentários de código e espaços em brancos. (AHO, LAN E ULLMAN,1995), (RIGO ,2015).

Esse processamento é semelhante ao reconhecimento de cada palavra e sua classe gramatical – verbos, adjetivos e substantivos – de uma linguagem natural. Para realizar esse procedimento normalmente são utilizados expressões regulares e autômatos finitos. (RICARTE, 2008).

2.1.1.1 Linguagens Regulares

Para o correto entendimento de uma linguagem é necessário compreender alguns conceitos.

O primeiro conceito importante é o alfabeto, que se trata de um conjunto finito de símbolos, em segundo, uma cadeia que é uma sequência de símbolos que pertencem a um alfabeto, por exemplo “aoia” é uma cadeia válida sobre o alfabeto das vogais (a, e, i, o, u). Finalmente uma linguagem corresponde a um conjunto de cadeias sobre o alfabeto. Neste trabalho será apenas abordado a linguagem regulares que tem como característica possuir um autômato finito equivalente. (MACIEL, 2006).

Para uma linguagem ser regular deve possuir as seguintes propriedades (MACIEL, 2006):

- Se a linguagem é igual a um conjunto vazio ou a linguagem é igual ao conjunto de qualquer símbolo pertencente ao alfabeto.
- L1 e L2 são linguagens regulares se uma linguagem regular é obtida a partir da união entre as duas linguagens.
- L1 e L2 são linguagens regulares se uma linguagem regular é obtida pela concatenação entre as duas linguagens.
- Se L1 é regular então a linguagem obtida a partir da concatenação de zero ou mais cadeias da linguagem L1 também é regular.

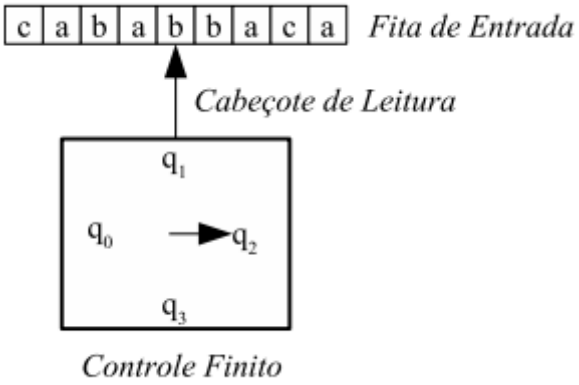
2.1.1.2 Autômatos Finitos Determinísticos

O autômato finito determinístico (AFD) é o modelo computacional existente menos complexo, são capazes de processar informações, recebendo uma entrada e exibindo uma saída, não possuem uma memória auxiliar e apesar disso ele é

totalmente adequado à função de reconhecimento de cadeias de texto. (MARCIEL, 2006).

A Figura 2.2 demonstra a composição de um AFD, fita de entrada, cabeçote de leitura e um controle finito que é um conjunto de estados distintos.

Figura 2.2. – Autômato Finito Determinístico



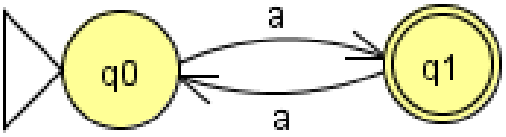
Fonte: MACIEL, 2006.

Um AFD é formado por uma quíntupla com a seguinte definição (AHO, LAN E ULLMAN ,1995):

1) Conjunto finito de estados.
2) Conjunto de símbolos de entrada.
3) Função de transição, que consiste na escolha de um próximo estado a partir de um símbolo e de um estado atual.
4) Estado inicial
5) Conjunto de estados finais

Pode-se representar um AFD através de um grafo de transição, como demonstra a figura 2.3.

Figura 2.3 – Grafo de transição AFD



2.1.1.3 Expressões Regulares

Entendido os conceitos de linguagens regulares e de um AFD é possível aplica-los para definir uma expressão regular e realizar a sua validação computacionalmente.

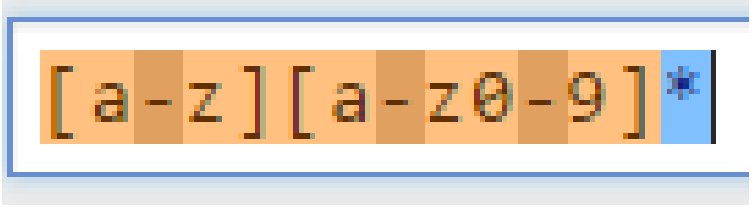
Uma expressão regular representa um padrão de cadeia de caracteres, ela é definida pelo conjunto de cadeias que “valida”. (LOUDEN, 2004).

Existem três principais operações em expressões regulares (LOUDEN, 2004):

1. Operação “OU”, normalmente representada pelo caractere ”|” (barra vertical);
2. Concatenação ou sequência;
3. Repetição, que normalmente é representada pelo caractere “*”;

A seguir um exemplo onde são demonstrados uma expressão regular, sua linguagem regular correspondente e o grafo do autômato determinístico gerado a partir da expressão.

Figura 2.4 – Exemplo de expressão regular



Na figura 2.4 é possível observar uma expressão regular que valida uma cadeia que comece com um caractere que esteja entre o intervalo de “a”-z” e que termine com zero ou mais caracteres entre “a”-z” ou “0-9”.

Figura 2.5 – Definição Exemplo 1

AFD = (S, Q, d, q0, F)
S = {a, b, c, d, ..., z, 0, 1, 2 ,3, ..., 9}
Q = {q0, q1}
D =

	a-z	0-9
q0	q1	-
q1	q1	q1

F= {q1}

A figura 2.5 monta a quíntupla que define um autômato finito determinístico onde:

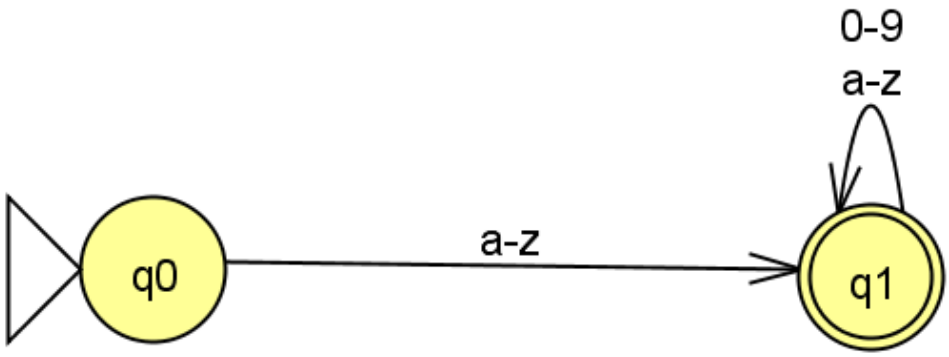
“S” é o alfabeto de símbolos que contempla nos intervalos de ‘a’ a ‘z’ e de ‘0’ a ‘9’.

“Q” é o conjunto de estados que formam o autômato.

“D” é a função de transição, que pode ser definida em formato de tabela, onde as linhas representam os estados de origem que ao consumir um símbolo da coluna da linha superior passa para um novo estado da célula correspondente.

Finalmente ”F” é o conjunto de estados finais representado pelo estado q1.

Figura 2.6 – Grafo Transição Exemplo 1



Na figura 2.6 é possível observar o grafo de transição do AFD correspondente a expressão regular do exemplo 1.

No exemplo 1 podemos validar um tipo de token de identificador (como nomes de variáveis, métodos), muito utilizado na maioria das linguagens de programação. (LOUDEN, 2004).

Existem diversas técnicas e algoritmos para minimização e simplificação de autômatos finitos determinísticos que não serão apresentados nesse trabalho, já no capítulo 4 será descrito as diversas alterações realizadas nos algoritmos de processamento e conversão de uma expressão regular para um autômato.

2.1.2 Análise Sintática

O analisador sintático é o responsável por gerar a árvore sintática a partir do fluxo de tokens que o analisador léxico montou, emitindo qualquer erro sintático encontrado durante o processamento. Para realizar o seu papel o analisador necessita de uma gramática representativa. (AHO, LAN E ULLMAN ,1995).

Existem diversas classificações de gramáticas segundo a classificação de Chomsky, neste trabalho será abordada as gramáticas livres de contexto. (AHO, LAN E ULLMAN ,1995).

2.1.2.1 Gramática Livres de Contexto

Uma gramática livre de contexto é formada por uma quádrupla com a seguinte definição (AHO, LAN E ULLMAN ,1995):

1. Conjuntos de variáveis ou não-terminais
2. Conjunto de símbolos terminais
3. Conjunto de produções, onde uma produção contém um lado esquerdo, composto por uma variável e um lado direito, que pode conter um conjunto de variáveis e terminais.
4. Uma varável inicial

Figura 2.7 – Exemplo gramática

$$
\begin{aligned}
G_2 &= (\{S, A, B, C\}, \{0, 1\}, P_2, S): \\
P_2 &= \{S \rightarrow 0A0 \mid 1B1 \mid BB \\
&\quad A \rightarrow C \\
&\quad B \rightarrow S \mid A \\
&\quad C \rightarrow S \mid \varepsilon\}
\end{aligned}
$$

Fonte: SAKATA, 2015.

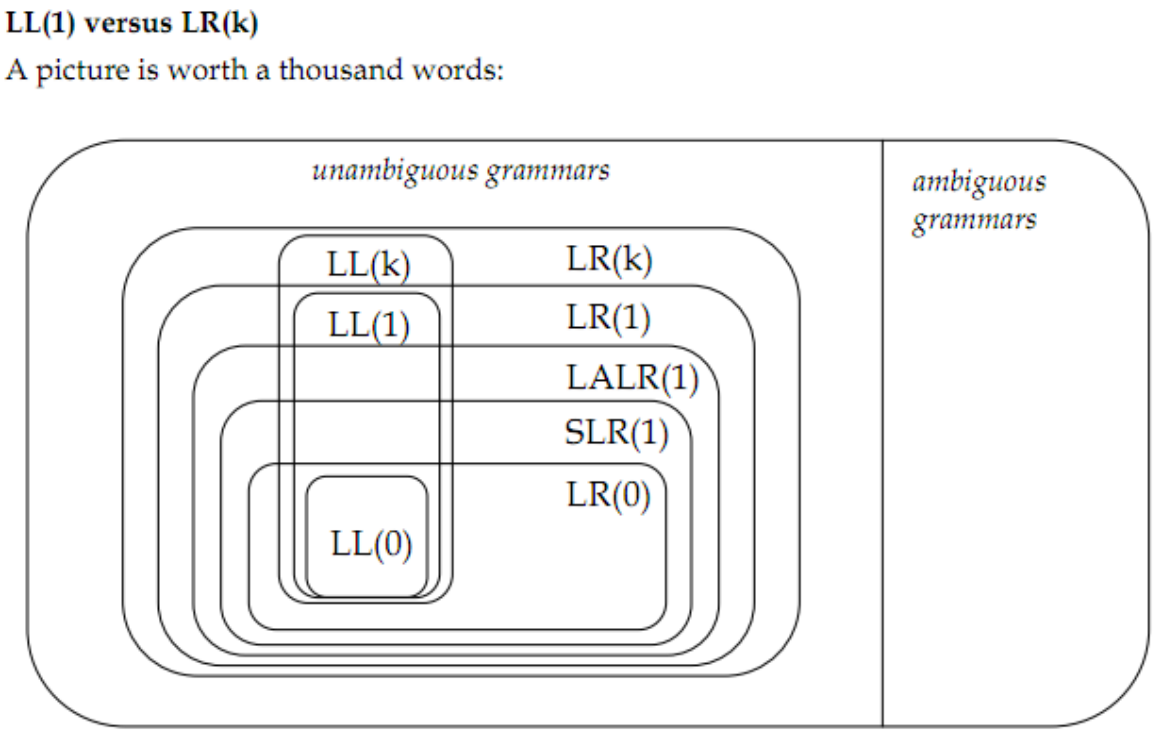
A figura 2.7 demonstra a formalização de uma gramática, onde o conjunto de variáveis é composto por ‘S’, ‘A’, ‘B’, ‘C’, o conjunto de símbolos terminais possui os números 0 e 1, o conjunto P2 representa as diferentes regras de derivações e finalmente o S é a variável inicial.

As GLCs (Gramáticas Livre de Contexto) são capazes de representar a sintaxe de uma linguagem de programação e, assim, nessa etapa é possível a validação e geração da árvore de derivação. (AHO, LAN E ULLMAN 1995).

2.1.2.2 Técnicas

A análise sintática pode ser realizada de forma ascendente ou descendente, a figura 2.8 categoriza os diversos algoritmos para validação sintática através de uma gramática.

Figura 2.8 – Algoritmos Análise Sintática



Fonte: APPEL, GINSBURG, 1998

Ainda na figura 2.8 é possível visualizar a abrangência dos algoritmos quanto as gramáticas suportadas, o algoritmo mais utilizado nos compiladores atuais é a LALR(1) que é uma otimização do algoritmo LR(1). (APPEL, GINSBURG, 1998).

Neste trabalho será abordado apenas a LR(1) devido a sua maior abrangência e mais fácil adaptação, que será demonstrada no capítulo 4.

2.1.3 Outras Análises

Conforme demonstrado na figura 2.1, compiladores possuem diversas etapas bem definidas além dos módulos tratados nesse capítulo, porém sua generalização acaba se tornando muito mais complexa, assim as ferramentas que auxiliam no processo de compilação acabam não abordando essas etapas. (GESSER, 2003).

2.2 Ferramentas

Existem diversas ferramentas que auxiliam nas etapas bem definidas de um compilador, segue alguns exemplos (GESSER, 2003):

LEX*:* Ferramenta de geração de analisadores léxicos, sendo um dos mais tradicionais geradores. Consiste em um programa que a partir de uma especificação léxica de expressões regulares gera uma rotina de analise léxica em diversas linguagens. (GESSER, 2003).

Microlex: Projeto acadêmico que também possui como entrada expressões regulares e a partir delas é gerado um analisador sintático em Object Pascal, C++ ou JAVA. (GESSER, 2003).

Yacc (Yet Another Compiler-Compiler): Programa criador de compiladores que consiste em um gerador de analisador sintático, trabalha em conjunto com a ferramenta LEX. (GESSER, 2003).

REFERÊNCIAS

AHO, A. V.; Lam, M. S.; Sethi R.; Ullman, J. D. **Compiladores, Princípios, técnicas e ferramentas**. 2.ed. São Paulo: Pearson Education do Brasil, 2008. 633 p.

APPEL, A. W.; GINSBURG M. **Modern Compiler Implementation in C.** 1.ed. Cambridge :The Edinburgh Bilding, 1998. 190 p.

BONATO, V. Hierarquia de Chomsky, Exemplos de Gramática. Dispoível em: http://wiki.icmc.usp.br/images/6/6f/Gramatica1_SCC_205.pdf. Acesso em: 10 setembro 2015.

MARCIEL, A. **Aplicação de autômatos finitos nebulosos no reconhecimento aproximado de cadeias**. 2006. 63 f. Dissertação (Mestrado em Sistemas Digitais) – Escola Politécnica da Universidade de São Paulo, São Paulo, 2006.

PESSOA, J. Autômatos Finitos não Determinísticos (AFN) e Determinísticos (AFD). Disponível em: http://www.dsc.ufcg.edu.br/~pet/jornal/junho2014/materias/recapitulando.html. Acesso em: 02 setembro 2015.

RICARTE, I. **Introdução a Compilação.** 1.ed. Rio de Janeiro: Elsevier, 2008. 258 p.

RIGO, S. Análise Léxica. Disponível em: http://www.ic.unicamp.br/~sandro/cursos/mc910/slides/cap2-lex.pdf. Acesso em: 05 setembro 2015.

SAKATA, T. C. Tópicos em Computação - Lista de Exercícios 2 – Linguagem Livre de Contexto. Disponível em: http://www.li.facens.br/~tiemi/Tc1/lista2.pdf. Acesso em: 10 setembro 2015.