Relatório da Construção de um Analisador Léxico e Sintático

Felipe Chabatura Neto[1], **Henrique José Dalla Corte**[1],
João Paulo K. Castilho[1], **Leonardo Tironi**[1]

[1]Departamento de Ciência da Computação –
Universidade Federal da Fronteira Sul (UFFS)
Chapecó – SC – Brasil

{felipechabat, riquejdc, joao.pkc, leehtironi}@gmail.com

Abstract. *This paper seeks to present the construction of a lexical analyzer that recognizes tokens from a created language with* 8 *reserved words,* 2 *symbols,* 5 *mathematicals operators,* 6 *relational operators and identifiers. Also the grammar construction from the Deterministic Finite Automata (DFA) for the word recognition. Therefore, a symbols table was built to be used in the syntatical recognition.*
In the second part of the work, it'll be presented the construction of a syntatic recognizer, where a Free Context Grammar(FCG) was made, over the work of the lexical analyzer. Therefore, this FCG is used on Gold Parsing System..

Resumo. *O presente trabalho busca apresentar a elaboração de um analisador léxico que reconhece tokens de uma linguagem criada com* 8 *palavras reservadas,* 2 *símbolos,* 5 *operadores matemáticos,* 6 *operadores relacionais e identificadores. Também a construção da gramática para avaliar e validar o conjunto de tokens para a linguagem elaborada, utilizando o trabalho da construção do Autômato Finito Determinístico para o reconhecimento das palavras. Com isso foi construída uma tabela de símbolos para ser utilizada no reconhecimento sintático.*
Na segunda parte do trabalho, será apresentada a construção de um reconhecedor sintático, onde foi feito uma Gramática Livre de Contexto(GLC) sobre o trabalho da Construção do Reconhecedor Léxico. Logo, esta GLC é utilizada no Gold Parsing System.

1. Introdução

O objetivo do trabalho é a implementação de um reconhecedor léxico e um reconhecedor sintático para uma linguagem criada.

O reconhecedor léxico utilizará um autômato finito determinístico, e no final de sua execução terá uma tabela de símbolos pronta para ser usada pela análise sintática. É necessária a definição de *tokens* de linguagem, construir uma Gramática Regular para validar e avaliar o conjunto de *tokens*, construir e ter um autômato finito determinístico, implementar o algoritmo para mapeamento desse autômato para o reconhecedor e por fim criar a tabela de símbolos com a estrutura definida.

A análise léxica é a primeira varredura do código fonte, esta agrupa caracteres em lexemas, produzindo sequências de símbolos léxicos, os *tokens*. O analisador léxico insere informações destes tokens na tabela de símbolos. Este trabalho é feito pelo algoritmo implementado no arquivo *lexica.cpp* e seu cabeçalho *lexica.h*.

O analisador léxico também é responsável por retirar comentários e espaços. Outra função é reconhecer uma sequência de tokens e enviar para a análise sintática que é a segunda parte do processo de compilação.

A construção do reconhecedor sintático se deu inicialmente pela definição das regras sintáticas da linguagem, através da construção de uma gramática livre de contexto(GLC). Depois, esta gramática foi utilizada como entrada no software *Gold Parsing System* a fim de se obter a tabela de *parsing*, através da construção do conjunto de itens válidos, transições, conjunto *follow* e construção da tabela de parsing *Simple Left-Right(SLR)* ou uma *Left-Right com um LookAhead (LALR)*.

Logo em seguida é feita a implementação do algoritmo que faz o mapeamento dos resultados obtidos pelo *parsing*, os quais estão em um arquivo .xml. Este algoritmo faz a leitura e mineração dos dados necessários do *.xml* resultante (uma vez que ele foi ajustado as necessidades dos autores), o qual se encontra no arquivo *sintatica.cpp*, junto com seu cabeçalho *sintatica.h*. Para tratar o erro, é verificada a linha que o *token* se encontra, além de qual seu rótulo. Então, é escrito na saída padrão que houve um erro na linha lida, devido ao *token* não esperado.

2. Referencial Teórico

Para o referencial teórico foi usado o livro *"Compiladores: Princípios, Técnicas e Ferramentas"* [AHO et al.]. Onde ele apresenta o papel do analisador léxico, o qual deve ler os caracteres do código fonte de entrada, agrupando estes em lexemas e dando como saída uma sequência de *tokens* para cada lexema. O analisador léxico altera a tabela de símbolos, como por exemplo quando descobre um identificador e o insere na tabela.

Outra tarefa é eliminar comentários, espaços e correlacionar mensagens de erros gerados pelo compilador ao utilizar o código fonte.

Algumas definições:

Tokens: Considerado um nome e um valor atribuído para ele, sendo opcional. Este nome consiste em um símbolo abstrato que é uma unidade léxica. Podendo ser um identificador ou uma palavra chave [AHO et al.].

Padrão: É a forma que um lexema de *token* pode assumir. A palavra-chave é um *token* que é formado por uma sequência de caracteres. Já para identificadores e outros *tokens*, o padrão é uma estrutura muito maior, que se utiliza de várias sequências de caracteres [AHO et al.].

Lexema: Sequência de caracteres do código fonte que casa com o padrão para um *token* e é identificado pelo analisador léxico como uma instância desse *token* [AHO et al.].

Neste trabalho se estabeleceu padrões com as definições de *tokens* da linguagem e a construção da Gramática Regular para o reconhecimento de lexemas como *tokens*. Estes podendo ser símbolos, identificadores, operadores e números.

3. Implementação

A implementação do trabalho consiste na programação feita após um estudo teórico das funções da análise léxica e sintática, com as Estruturas:

Tabela 1. Gerada a partir do codigo 3

Rotulo	Nome	Linha	Estado Reconhecedor	Novo Reconhecedor
PALAVRA_CHAVE	int	1	77	-
IDENTIFICADOR	a	1	1	-
SIMBOLO	,	1	3	-
IDENTIFICADOR	b	1	1	-
SIMBOLO	,	1	3	-
IDENTIFICADOR	a	2	1	-
OPERADOR	=	2	4	-
NUMERO	4	2	2	-
OPERADOR	*	2	4	-
NUMERO	2	2	2	-
SIMBOLO	;	2	3	-
IDENTIFICADOR	b	3	1	-
OPERADOR	=	3	4	-
NUMERO	7	3	2	-
SIMBOLO	;	3	3	-
PALAVRA_CHAVE	se	4	79	-
IDENTIFICADOR	a	4	1	-
OPERADOR_RELACIONAL	}	4	5	-
IDENTIFICADOR	b	4	1	-
PALAVRA_CHAVE	entao	4	61	-
IDENTIFICADOR	b	5	1	-
OPERADOR	=	5	4	-
IDENTIFICADOR	a	5	1	-
SIMBOLO	;	5	3	-
PALAVRA_CHAVE	fimse	6	74	-

- *linguicona*: *Vector* de *linguicona_t*. *linguicona_t* contém os campos rótulo, nome e linha;
- Identificadores: *Map* de *string* para *int*, contendo o nome da variável e seu valor. Os valores não serão preenchidos ainda;
- *Tabela_simbolos_t*: estrutura de tabela de símbolos, contém um *map* de identificadores e uma *linguicona*.

Usando o código 3, a tabela gerada é a tabela 1.

Código 3: Função que imprime a tabela de símbolos.

```
int a , b ;
a = 4 * 2 ;@Este comentário é ignorado!
b = 7 ;
se a } b entao
 b = a ;
fimse
```

Faz uso das funções: *analiseLexica*: Recebe o arquivo do código como entrada. Lê cada uma das linhas, e processa cada um dos lexemas, terminando o reconhecimento

ou não reconhecimento do lexema atual em espaços. Espaços adicionais são ignorados;

pegaRotulo: Recebe o estado atual e retorna o rótulo correspondente no reconhecimento;

colocaTabelaSimbolos: Coloca um lexema já reconhecido na tabela de símbolos;

imprimeTabelaSimbolos: Imprime o conteúdo da tabela de símbolos.

transcreveRotulo: Dado um inteiro, retorna a *string* do rótulo correspondente.

A análise sintática, conhecida como parser tem sua função de visualizar se a entrada com as sequências de *tokens* tem as sentenças válidas para a linguagem de programação utilizada. Sendo que esta parte deve obter as formas utilizadas no código fonte e validar ou retornar um erro. Para isso pode ser utilizado um algoritmo que faz a derivação das construções da linguagem, esta derivação é feita para determinar se uma sequência de palavras está dentro da sintaxe da linguagem de programação construída. Algumas definições:

Símbolos: Elementos mínimos da linguagem.

Sentença: Conjunto ordenado de Símbolos, sendo uma *string* ou cadeia destes.

Alfabeto: Conjunto de Símbolos.

Linguagem: Conjunto de Sentenças.

Gramática: Forma que representa as regras da formação de uma linguagem.

Uma Gramática Livre de Contexto(GLC) é a categoria onde estão as linguagens de programação, esta é uma base para a construção de analisadores sintáticos, a qual é usada para definir as regras da linguagem de programação. A Gramática Livre de Contexto é reconhecida por um autômato de pilha. Onde na GLC:

N - Conjunto Finito de Símbolos Não Terminais

T - Conjunto Finito de Símbolos Terminais

P - Conjunto de Regras de Produções

S - Simbolo Inicial da Gramática

Terminologias da GLC:

Símbolos Terminais: Conjunto de símbolos das palavras da linguagem, são os *tokens* reconhecidos pelo analisador léxico.

Símbolos Não Terminais: Conjunto de variáveis utilizadas no conjunto de palavras linguagem, formadas pelos terminais e não terminais.

Símbolo Inicial: Variável, não terminal usada para definir inicio da linguagem.

Regras de Produção: Conjunto de regras sintáticas da definição da linguagem, indica como símbolos terminais e não terminais podem ser agrupados.

A GLC usada pelos autores foi a seguinte:

```
<ATR> ::= id = <E>;
<E> ::= <E> + <T> | <E> - <T> | <T>
<T> ::= <T> * <F> | <T> / <F> | <F>
```

```
<F> ::= id | num
<SE> ::= se <CLAUSULA> entao <BLOCO><SENAO>
<ENQUANTO> :: = enquanto <CLAUSULA> entao <BLOCO> fimenquanto
<DECLARA> ::= int id<DECLARACAO>
<BLOCO> ::= <ATR><BLOCO> | <SE><BLOCO> | <ENQUANTO><BLOCO> |
<DECLARA><BLOCO>
<DECLARACAO> ::= , id <DECLARACAO> | ;
<CLAUSULA> ::= <E><OPREL><E> | <E>
<OPREL> ::= } | { | ==
-><S> ::= <BLOCO>
<SENAO> ::= senao <BLOCO> fimse | fimse
```

Para construir o analisador sintático, foi necessário minerar os dados de um documento *.xml* produzido pelo *Gold Parser*. Com esses dados, foram criadas três tabelas, são elas:

Reconhecedor de símbolos: usado para transformar os símbolos de saída da análise léxica para a saída do *Gold Parser*, de acordo com os identificadores gerados por ele.

Tabela de reduções: utilizada para consultar qual a redução a ser feita e quantos elementos da pilha remover.

Tabela de *parsing*: usada para verificar para qual estado transicionar, além de que ação deve ser tomada.

O algoritmo que usará essas tabelas pode ser verificado abaixo:

Verifica o topo da pilha Se o topo da pilha não for um estado:

Salva o topo da pilha em símbolo atual.

Procura pelo próximo estado e salva ele em estado atual.

Se o topo da pilha for um estado:

Salvar este estado em estado atual e procura na fita o símbolo atual.

Verifica na tabela de parsing o que fazer com o estado atual e o símbolo atual.

Se for um empilhamento

Coloca o que está em símbolo atual na pilha, dizendo que ele é um símbolo.

Coloca o que está em próximo estado na pilha, dizendo que ele é um estado.

Move o ponteiro da fita uma casa.

Se for uma redução, procura o id da redução na tabela de redução.

Desempilha o quanto reduz.

Empilha o novo símbolo dizendo que ele é um símbolo.

Se for um salto, coloca o que está em próximo estado na pilha, dizendo que ele é um estado.

4. Resultados

Como resultado, obtivemos um Analisador Léxico que reconhecerá cada um dos *tokens* se ele pertencer a linguagem, do contrário, resultará em um erro que será mostrado no terminal, identificando sua linha e qual o lexema que está em desacordo. Além disso, foi criado um analisador sintático, que imprimirá na saída padrão se ele reconheceu o código sintaticamente ou não. Por fim, se o código fonte não for reconhecido pelo analisador léxico, ele não será enviado para a análise sintática.

Referências

AHO, A., SETHI, R., and LAM, S. Compiladores: princípios, técnicas e ferramentas.[sl]: Longman do brasil, 2008. *ISBN*, 358171665:1.