https://eventos.utfpr.edu.br//sicite/sicite2020

LEXF: Um Analisador Lexical Eficiente e Multipropósito

LEXF: An Efficient and Multipurpose Lexical Analyser

Felipe Arthur Povaluk da Silva
felipesilva.2017@alunos.utfpr.edu.br
Universidade Tecnológica federal do Paraná, Toledo, Paraná, Brasil

Gustavo Henrique Paetzold
ghpaetzold@utfpr.edu.br
Universidade Tecnológica Federal do Paraná, Toledo, Paraná, Brasil

RESUMO

Todo compilador precisa necessariamente passar pelos processos de análise léxica, sintática e semântica, respectivamente, para termos a compilação de um código em uma linguagem de máquina. Com isto, foi desenvolvida a biblioteca LEXF que possui um analisador léxico, sendo que na prática este constrói um autômato com base nas inserções dos estados nesse autômato, e então analisa de forma eficiente se essa entrada possui *tokens* válidos ou não para a linguagem determinada por este autômato e retorna a lista de *tokens* resultantes. Foi comparado o desempenho em tempo da LEXF com o analisador léxico da biblioteca PLY onde se constatou que o mesmo possui um nível de eficiência próximo ao desta ferramenta já existente.

PALAVRAS-CHAVE: Análise lexical. Autômato. Compilador. *Token*.

Recebido: 04 set. 2020.
Aprovado: 01 out. 2020.

Direito autoral: Este trabalho está licenciado sob os termos da Licença Creative Commons-Atribuição 4.0 Internacional.

ABSTRACT

Every compiler must necessarily go through the processes of lexical, syntactic and semantic analysis, respectively, to have the compilation of code in a machine language. With this, the LEXF library was developed, which has a lexical analyser, and in practice it builds an automaton based on the state insertions in that automaton, and then efficiently analyse whether this entry has valid tokens or not for the language determined by this automaton and returns the list of resulting tokens. The time performance of LEXF was compared with the lexical analyser of the PLY library, where it was found that it has an efficiency level close to that of this existing tool.

KEYWORDS: Lexical analysis. Automata. Compiler. Token.

INTRODUÇÃO

Existem várias formas de comunicação entre duas entidades diferentes, como por exemplo a comunicação com linguagens naturais entre nós humanos que utilizamos do português, por exemplo, para fazermos a transmissão de informações de acordo com as regras gramaticais propostas por esta língua. Entretanto quando se fala em comunicação com máquinas, as linguagens naturais não servem para essa situação pois estas abrem espaço para dialetos e subjetividades, o que impossibilita a comunicação com uma máquina. Nesse caso necessitamos de uma linguagem não natural, que também possui regras léxicas, sintáticas e semânticas que permitem o funcionamento da mesma, mas que ao mesmo tempo é objetiva, não permitindo dupla interpretação para uma mesma frase.

Por conta disso, existe a necessidade de definir primeiramente as regras léxicas de uma linguagem, para então partirmos para uma sintaxe e uma semântica, que servem de base para a construção de um **compilador:** um programa que cria um programa em linguagem de código de máquina a partir de um código fonte escrito.

Um analisador léxi**co** pode ser definido como foi descrito por Alfred (1995, p. 6), sendo um programa que: "lê os caracteres de um programa fonte e os agrupa num fluxo de *tokens*, no qual cada *token* representa uma sequência de caracteres logicamente coesa", sendo que a captação dos mesmos se faz com base em uma regra léxica preestabelecida onde podemos definir se um conjunto de caracteres pertence a algum *token* de uma determinada linguagem.

Uma linguagem não precisa necessariamente ser uma linguagem de programação convencional (como C, Python, Java, entre outras), mas pode também ser uma linguagem que interprete, por exemplo, uma expressão lógica, ou seja, uma linguagem definida em termos gramaticais que, nesse caso, sustente o objetivo de interpretar uma expressão lógica, sendo que primeiramente deve se passar por um analisador léxico que defina se os caracteres utilizados pertencem a um conjunto de *tokens* válidos para uma expressão lógica (ou então, para esta linguagem).

Ao buscar por analisadores léxicos já existentes para Python foi encontrada a PLY, que conforme Beazley (2020) define: "PLY is a 100% Python implementation of the lex and yacc tools commonly used to write parsers and compilers.", ou seja, se trata de uma biblioteca que contém ferramentas de análise léxica já consolidada, porém foi identificado certas limitações em relação à sua facilidade de uso e flexibilidade. Com isso foi criado a nova biblioteca LEXF, também feita em Python, porém mais simples e que disponibiliza mecanismos que permitem a utilização de forma didática, sobre como esse processo (de análise léxica) ocorre de forma eficiente na prática para diferentes tipos de linguagem.

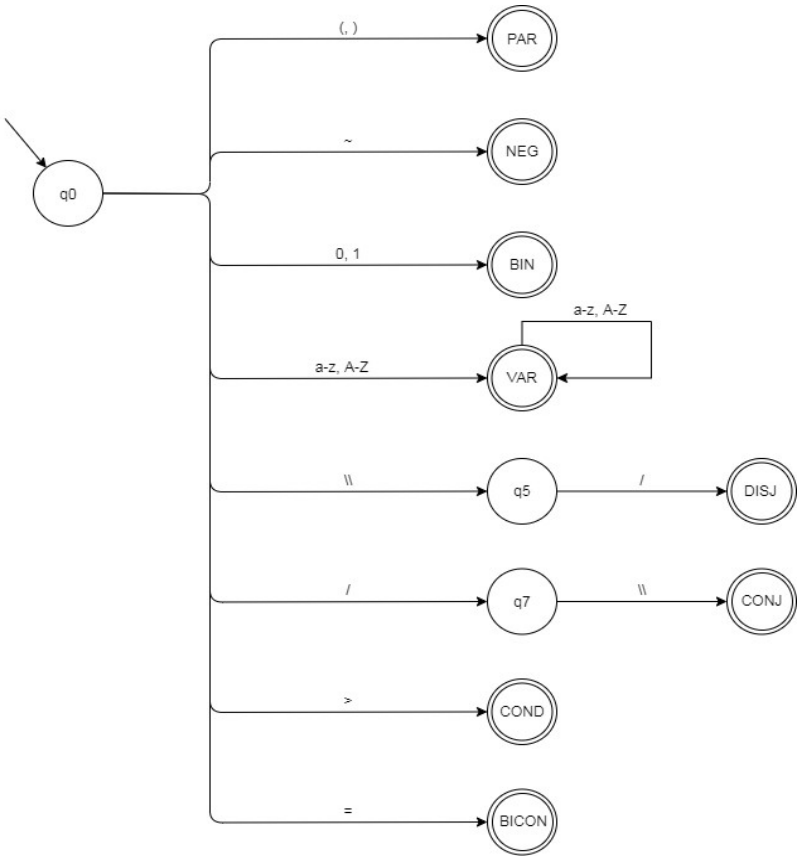
MATERIAL E MÉTODOS

Conforme foi dito acima, um analisador léxico é uma ferramenta que analisa sequências de caracteres e identifica "componentes lexicais de uma determinada linguagem de programação"(PAETZOLD; SCHEMBERGUER, 2014, p. 27), os chamados *tokens*, e com base nisso foi construído a **LEXF:** uma biblioteca que

possui um analisador léxico, feito com foco em expressões lógicas inspirada no artigo do "EXTLex: UM ANALISADOR LÉXICO EXTENSÍVEL CAPAZ DE DETECTAR ERROS LEXICAIS" (PAETZOLD; SCHEMBERGUER, 2014, p. 27).

Antes de iniciar a programação do próprio foi necessário definir o **autômato** que representaria a estrutura das definições dos *tokens*, ou seja, uma imagem de um grafo que demonstre como será feita a transição de um estado (correspondente a um vértice do grafo) da máquina (autômato) para outro estado, de acordo com o estado atual e o caractere lido, com o objetivo de definir se estes caracteres pertencem a um *token* (um estado final da máquina), o que resulta na utilização prática de um Autômato Finito Determinístico (AFD) e em como essa máquina de estados se comporta dada uma entrada. O AFD utilizado como base pode ser visto na Figura 1 abaixo:

Figura 1 – Autômato a ser implementado com a LEXF



Essa figura mostra um **autômato** que identifica *tokens* da lógica proposicional, que foi criado para realização de testes com a LEXF. Este autômato se inicia no **estado** definido como "q0" (o estado inicial sempre possui uma flecha que não venha de outro estado apontando para si) e então, de acordo com os caracteres lidos na entrada, o estado atual da máquina se altera para outro estado do autômato. Um estado final é definido como aquele que possui dois círculos que circundam o nome do estado, e estes estados representam um *token* desta linguagem. Consequentemente os estados que possuem apenas um círculo são estados não finais, logo não representam um *token* desta linguagem, são apenas estados transitórios.

Por exemplo, ao ler o caractere “0”, o estado atual da máquina passa do estado inicial “q0” e passa a ser o “BIN” (que, neste caso, representa constantes binárias).

CODIFICAÇÃO

Assim como foi dito, a biblioteca LEXF foi implementada em Python (escolha feita devida a facilidade de uso e agilidade de prototipação que a linguagem possui) ao mesmo tempo que também foi utilizado o paradigma orientação a objetos.

Na LEXF existe a classe **Estado**, na qual definimos o nome do mesmo, a regra que esse estado possui (sendo esta regra, um dicionário) e se o mesmo é um estado final ou não. As regras da classe Estado funcionam de maneira simples, do modo: “lê o caractere X, se direciona ao estado A”.

Como um autômato possui vários estados, foi criada a classe **Automato**, a qual possui um conjunto de Estados, onde cada Estado inserido é adicionado à lista de estados do Automato. A adição dos estados no autômato é feita de forma sequencial, ou seja, deve se ter cuidado ao utilizar o mesmo pois a ordem de inserção de estados interfere na ordem de numeração dos mesmos.

Nesta classe Automato que temos a **análise léxica** implementada de fato por meio do método **lexf_corretude:** é passada a linha de caracteres a ser analisada lexicalmente e é retornada uma lista (por referência) com os nomes dos *tokens* e uma lista (por referência) com os próprios *tokens* de acordo com os caracteres lidos, caso haja algum caractere inválido (ou seja, não pertence a nenhum *token*) é retornado dentro de uma *string* de saída (como retorno do método) quais os caracteres inválidos seguidos de suas posições. Esse método percorre a *string* de entrada e de forma paralela percorre o autômato com base nas regras estabelecidas e o caractere lido. Se ao chegar em um dado estado não for possível avançar para outro estado de acordo com o caractere lido e esse estado é um estado final, toda essa *string* lida até então se trata de um *token* válido. Entretanto, caso o estado seja não final, então essa *string* não pertence a um *token* válido, logo esta será adicionada na *string* de saída como um *token* inválido. Após a *string* ser definida como um *token* válido ou não, retorna-se ao estado inicial para então percorrer o autômato novamente com os demais caracteres da entrada.

ANALISADOR LÉXICO DE EXPRESSÕES LÓGICAS

Conforme o autômato da Figura 1, a LEXF foi utilizada com o objetivo de ser um analisador léxico para processar uma dada proposição lógica:

> “Chama-se proposição todo conjunto de palavras ou símbolos que exprimem um pensamento de sentido completo. (...) Assim as proposições são expressões a respeito das quais tem sentido dizer que são verdadeiras ou falsas” (ALENCAR, 2002, p. 10,11)

Sendo que os *tokens* definidos são:

a) \/ : disjunção;

b) /\ : conjunção;

c) > : condicional;

d) = : bicondicional;

e) (a-z, A-Z) : variáveis;

f) 0, 1 : valor binário estático;

g) (,) : parênteses (indicam trecho que possui preferência de solução);

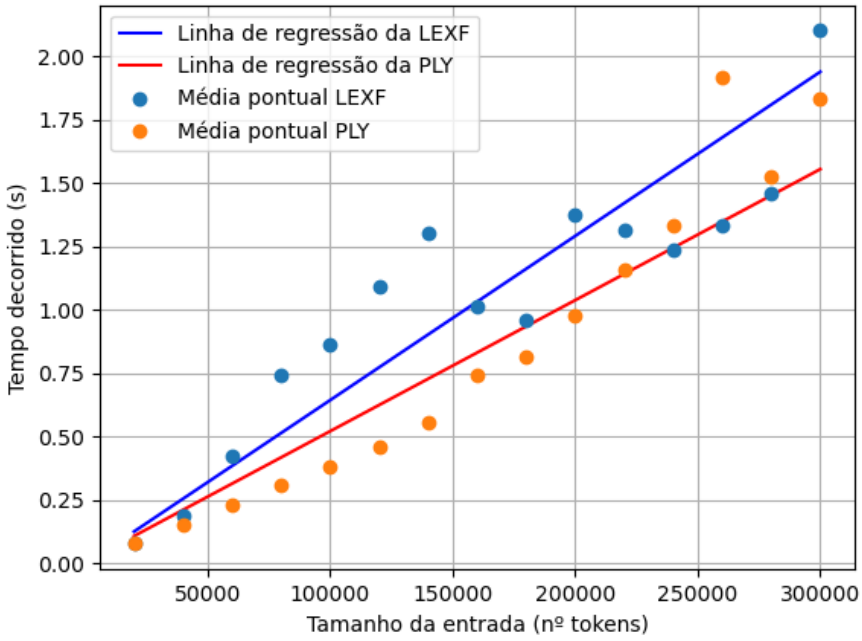
h) ~ : negação lógica;

Após o resultado da análise léxica, as listas dos *tokens* de uma entrada e seus nomes podem ser utilizados para a **análise sintática**.

RESULTADOS E DISCUSSÃO

Com objetivo de comparação, o mesmo exemplo instanciado com a LEXF foi feito no analisador léxico da PLY, uma biblioteca já existente que também possui ferramentas de análise léxica. Para este fim foi criado um *script* que monta um *dataset* de validação, ou seja, um *script* que retorna dois arquivos, sendo um a entrada e outro a saída esperada de acordo com este arquivo de entrada, feito com a intenção de testar os dois analisadores e comparar os resultados da LEXF com o da PLY. O resultado desta comparação pode ser visto na Figura 2 abaixo.

Figura 2 – Resultados da análise comparativa empírica de desempenho das bibliotecas LEXF e PLY



Assim como mostra o eixo horizontal da Figura 2, o arquivo conta com entradas (separadas em linhas) de 20.000 a 300.000 *tokens*, sendo que a linha de regressão de desempenho da LEXF (azul) se mostrou ligeiramente inferior a capacidade da PLY (vermelha), porém também teve resultado satisfatório em

questão de tempo de processamento mesmo com entradas grandes, como por exemplo com 300.000 *tokens*, mas com a característica de ser uma ferramenta mais fácil de se utilizar.

Como também se pode verificar, a linha de regressão da LEXF cresce de forma linear em relação aos pontos no gráfico (cada ponto representa a média de 3 entradas para cada tamanho N de entrada), evidenciando uma complexidade algorítmica O(N), ou seja, assintoticamente a curva de tempo de desempenho da LEXF é linear.

CONCLUSÕES

A LEXF pode ser utilizada tanto com o objetivo de se construir um AFD, o que pode servir para uma aplicação direta de autômatos, tanto para uma análise léxica, servindo assim como a primeira (léxica) das três etapas (léxica, sintática e semântica) de processamento de uma linguagem. Ela possui um ótimo tempo de processamento (se comparado com a PLY) e sua utilização é extremamente eficiente tanto na construção do AFD quanto na aplicação léxica, sendo que na PLY não temos essa percepção do que acontece por trás de sua instanciação.

A partir da utilização da LEXF para análise léxica, o retorno da mesma foi utilizado com objetivo de ser a entrada (o passo anterior) para o começo da análise sintática, desenvolvida pelo estudante Jordano Lahm, que posteriormente será a entrada da semântica, o que conclui todo o processo de criação de um compilador.

AGRADECIMENTOS

Gostaria de agradecer ao estudante Jordano Lahm pela ideia, ao professor Gustavo H. Paetzold por todo suporte, auxílio e interesse neste projeto, a Universidade Tecnológica Federal do Paraná e principalmente a Fundação Araucária por disponibilizar esse projeto e a bolsa, incentivando o interesse por assuntos científicos e disseminando o conhecimento.

REFERÊNCIAS

ALENCAR FILHO, E. **Iniciação à lógica matemática**. São Paulo: Nobel, 2002.

ALFRED, V. A.; ULLMAN D.J.; SETHI, R. **Compiladores:** Princípios, Técnicas e Ferramentas. LTC, Rio de Janeiro, 1995.

BEAZLEY, David. M. PLY (Python Lex-Yacc). Disponível em: https://ply.readthedocs.io/en/latest/. Acesso em: 31 ago. 2020.

PAETZOLD, G. H.; SCHEMBERGUER, E. E. EXTLex: UM ANALISADOR LÉXICO EXTENSÍVEL CAPAZ DE DETECTAR ERROS LEXICAIS. REVISTA ELETRÔNICA CIENTÍFICA INOVAÇÃO E TECNOLOGIA, 2, 2014, Medianeira. **Anais...** Medianeira: Universidade Tecnológica Federal do Paraná Campus Medianeira, 2014. p. 26-36.