SEI-SICITE 2021
Pesquisa e Extensão para um mundo em transformação

XI Seminário de Extensão e Inovação
XXVI Seminário de Iniciação Científica e Tecnológica
08 a 12 de Novembro - Guarapuava/PR

UTFPR
UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
CAMPUS GUARAPUAVA

Desenvolvimento de um Analisador Semântico para a Lógica Proposicional

Development of a Semantic Analyzer for Propositional Logic

Guilherme Poleto (orientado) *, Gustavo Henrique Paetzold (orientador)†

RESUMO

Lógica proposicional é um formalismo matemático pelo qual podemos descrever sentenças declarativas a partir de uma linguagem formal composta por um conjunto de regras que busca remover ambiguidades presentes na linguagem natural. Tais sentenças são chamadas de proposições ou enunciados e podem assumir dois valores: verdadeiro ou falso, propriedade esta chamada de valor-verdade. Os estudos de suas aplicações se estendem muito além de meras expressões matemáticas, sendo vastamente utilizadas nos campos da ciência da computação e áreas correlatas, como algoritmos de otimização e análise de circuitos lógicos digitais, portanto, é de interesse didático a facilitação da exposição desse conceito. Este trabalho descreve as etapas de desenvolvimento de um analisador voltado a interpretação da estrutura semântica de expressões lógicas proposicionais com o intuito de atenuar a curva de aprendizado de tais operações, abordando conceitos como máquinas de estados, pilha e fila e algoritmos de análise clássicos como *Shunting-yard* e Árvores de Resolução Semântica. O resultado final foi uma estrutura de análise de expressões completa implementada em *Python* de fácil compreensão para estudantes em seu primeiro contato com as etapas de processamento de expressões.

Palavras-chave: Lógica proposicional. Interpretador. Análise semântica. Teoria da Computação.

ABSTRACT

Propositional logic is a mathematical formalism by which we can describe declarative sentences from a formal language composed of a set of rules that seek to remove ambiguities present in natural language. Such sentences are called propositions or statements and can take two values: true or false, a property called the truth-value. The studies of its applications extend far beyond mere mathematical expressions, being widely used in the fields of computer science and related areas, such as optimization algorithms and analysis of digital logic circuits, therefore, facilitating the exposure of this concept is of educational interest. . This work describes the stages of development of an analyzer aimed at interpreting the semantic structure of propositional logical expressions in order to smooth the learning curve of such operations, covering concepts such as state machines, stack and queue and classical analysis algorithms such as textitShunting-yard and Semantic Resolution Trees. The end result was a complete expression analysis framework implemented in *Python* that is easy to understand for students in their first contact with expression processing steps.

Keywords: Propositional logic. Interpreter. Semantic analysis. Computer Theory.

* Departamento de Engenharia de Computação; gpoleto@alunos.utfpr.edu.br; https://orcid.org/0000-0002-7894-2669.
† Departamento de Engenharia de Computação; ghpaetzold@outlook.com; https://orcid.org/0000-0001-9951-050X.

1 INTRODUÇÃO

Expressões lógicas são definidas por literais e conectivos, também chamados *tokens*, definindo a chamada logica proposicional, cujo objetivo é remover ambiguidades presentes na linguagem natural (SOARES CORRÊA DA SILVA; FINGER; VIEIRA DE MELO, 2018). Essas expressões são comumente chamadas proposições ou enunciados, tais proposições possuem uma propriedade chamada valor-verdade, o que implica que serão valoradas como verdadeiras ou falsas (F. BISPO; B. CASTANHEIRA; MELO S. FILHO, 2017).

Como interpretamos, validamos e valoramos essas expressões? A análise de tais expressões consiste de 3 etapas fundamentais: análise léxica, sintática e semântica. O primeiro passo da análise se preocupa com a validação dos *tokens*, verificando apenas se existem dentro da linguagem definida, sem se preocupar com a sequência dos caracteres ou significado da expressão. O passo seguinte valida a ordem em que os *tokens* estão dispostos, de forma a analisar se a sequência destes é concisa com a gramática definida. O último passo, por sua vez, tem a função de verificar se tal expressão se trata de uma contradição, contingência ou tautologia (REIS SANTOS; LANGLOIS, 2018).

2 MÉTODO

Para que seja realizada a análise semântica de uma expressão é necessário que tenham sido definidas as regras da linguagem a qual essas expressões irão obedecer, neste caso, as regras serão as mesmas da álgebra *booleana* (DAGHLIAN, 1995), é necessário também que o enunciado tenha sido validada pelas etapas de análise léxica e sintática, portanto, a implementação destes métodos é essencial para o desenvolvimento desta última etapa de análise.

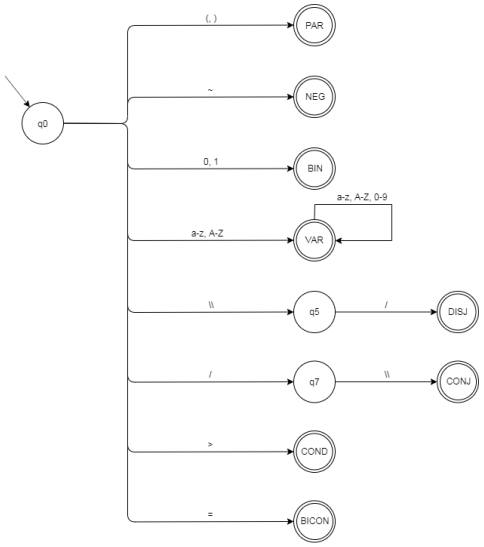
Para a análise léxica foi definido um *lexicon*, ou seja, um conjunto de *tokens* que pertencem a nossa linguagem, esse conjunto consiste de palavras que são classificadas em três possíveis tipos: operadores, valores binários e variáveis, também chamadas proposições. Proposições seguem regras léxicas definidas pela linguagem, como, por exemplo, variáveis não devem ser iniciadas por números e a sequência de caracteres dos operadores deve ser completa (ALBRIGHT, 2020). Para tanto foi implementada em *Python* a máquina de estados descrita na Figura 1.

Esta implementação recebe os caracteres um a um e os valida e agrupa em tuplas de *tokens*, se algum caractere não pertencer a nossa linguagem, então a sequência de caracteres é descartada e a expressão é considerada inválida, no entanto, a análise segue para os próximos valores da proposição de forma que o máximo de erro léxicos sejam descriminados, a análise nos retorna uma tupla contendo um *booleano* indicando se a expressão é válida, uma lista de tuplas de *tokens* válidos (caso haja algum) e seus respectivos valores e, se houver, uma lista de erros encontrados.

Para a análise sintática foi implementado também em *Python* o algoritmo *Shunting-Yard* que consiste em receber a lista de tuplas citada anteriormente, retornada pela saída da análise léxica e inserir sequencialmente as tuplas em uma estrutura de pilha para gerar uma saída em forma de fila com a expressão convertida de notação infixa para pós-fixa, também chamada de notação polonesa reversa, seguindo as condições do algoritmo e a precedência de operadores lógicos. Então, utiliza-se outra pilha, como na Figura 2 para a validação da sintaxe, checando se as operações presentes na expressão são válidas (TILIKSEW et al., 2020).

No processo de análise semântica é dada como entrada a fila de expressões em notação polonesa reversa, sendo

Figura 1 – Autômato Base



Fonte: (POVALUK DA SILVA; PAETZOLD, 2020)

Figura 2 – Exemplo de pilha de notação polonesa reversa

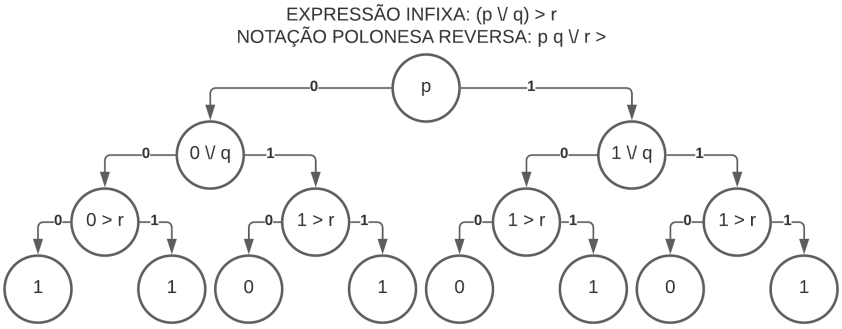
Fonte: Autoria própria (2021)

inserida em uma estrutura de árvore chamada árvore de resolução semântica, onde é analisada a valoração de cada expressão levando em consideração todas as combinações de valores possíveis para as variáveis declaradas, de forma que os nós-folha representem a tabela verdade da expressão analisada, p número de nós-folha será 2^N onde N é o número de variáveis na expressão (LOUDEN, 2004).

Cada nó intermediário da árvore representa uma possível valoração de uma proposição, e cada nó folha representa um possível resultado final da expressão. Se todos os nós folhas forem verdadeiros, se trata de uma tautologia, se todos forem falsos, se trata de uma contradição, já, se ao menos um nó folha for verdadeiro, se trata de uma contingência, também chama de expressão satisfazível.

É notável na Figura 3 a possibilidade de otimização do algoritmo de forma que não sejam valoradas todas as proposições possíveis, visto que podem ser equivalentes, e, portanto, resultarem na mesma valoração final.

Figura 3 – Árvore gramatical de uma contingência



Fonte: Autoria própria (2021)

3 RESULTADOS

Temos como resultado dessa pesquisa a implementação das etapas de análise léxica, sintática e semântica, que recebe a sequência de *tokens* que representam a expressão lógica e valida cada um deles, dando como saída a valoração da expressão em cada etapa.

A aplicação implementada possibilitará a atenuação da curva de aprendizado de algoritmos aqui apresentados e facilitará a abordagem prática desses assuntos para os interessados, além de possibilitar futuras expansões no código, como otimização de seus algoritmos e a implementação de algoritmos de redução de expressões lógicas tendo como entrada a saída da aplicação aqui descrita.

4 CONCLUSÕES

Este analisador pode ser utilizado para a valoração e criação da tabela verdade de expressões lógicas bem como para estudo dos procedimentos básicos de análise de um interpretador, além de utilizar a saída deste analisador como entrada para o estudo e desenvolvimento de algoritmos de otimização de expressões lógicas, como Mapas de *Karnaugh* ou *Quine-McCluskey*.

Há ainda a possibilidade de otimização do processo de análise semântica por métodos como *hash maps* e programação dinâmica, no qual expressões equivalentes não seriam computadas, evitando redundância e aumentando o desempenho do algoritmo. O código será aberto e disponibilizado sob licença *GPLv3*.

AGRADECIMENTOS

Agradeço ao professor Gustavo H. Paetzold por sua orientação e paciência, a Greici K. Heimerdinger por seu incentivo e apoio emocional e a Luana C. G. Campos por sua grande ajuda.

REFERÊNCIAS

ALBRIGHT, Larry. Compilers. **Salem Press Encyclopedia of Science**, 2020. Disponível em: .

DAGHLIAN, Jacob. **Lógica e álgebra de Boole**. 4. ed. [S.l.]: Grupo GEN, 1995.

F. BISPO, Carlos Alberto; B. CASTANHEIRA, Luiz; MELO S. FILHO, Oswaldo. **Introdução à Lógica Matemática**. [S.l.]: Cengage Learning Brasil, 2017.

LOUDEN, Kenneth C. **Compiladores: princípios e práticas**. [S.l.]: Cengage Learning Brasil, 2004.

POVALUK DA SILVA, Felipe Arthur; PAETZOLD, Gustavo Henrique. LEXF: Um Analisador Lexical Eficiente e Multipropósito. **XXV Seminário De Iniciação Científica E Tecnológica Da UTFPR**, 2020. Disponível em: .

REIS SANTOS, Pedro; LANGLOIS, Thibault. **Compiladores - Da Teoria à Prática**. [S.l.]: Grupo GEN, 2018.

SOARES CORRÊA DA SILVA, Flávio; FINGER, Marcelo; VIEIRA DE MELO, Ana Cristina. **Lógica para computação**. 2. ed. [S.l.]: Cengage Learning Brasil, 2018.

TILIKSEW, Beakal et al. Shunting Yard Algorithm, 2020. Disponível em: .