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

Desenvolvimento de um interpretador completo de expressões de teoria dos conjuntos para fins didáticos

Development of a complete set theory expression interpreter for didactic purposes

Fellipe de Souza Reis
fellipe@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

A escassez de software para auxílio no aprendizado dos alunos dos cursos superiores da área de computação, principalmente em matérias introdutórias como matemática discreta e lógica para computação, motiva a implementação de ferramentas do gênero. Entre os diversos tipos de softwares que podem auxiliar no aprendizado tem-se os interpretadores. Interpretadores são programas que executam instruções escritas em uma determinada linguagem. Neste trabalho foi abordada a construção de um interpretador completo de expressões de teoria dos conjuntos para fins didáticos. Para isso foi utilizada a literatura básica de compiladores em conjunto com a linguagem Python. Como resultado do trabalho foi implementado o interpretador de expressões de teoria dos conjuntos. Sua corretude foi evidenciada por experimentos de natureza empírica.

PALAVRAS-CHAVE: Interpretador. Expressão. Conjuntos.

ABSTRACT

The scarcity of software to assist students and lecturers in higher education in the field of computing, especially in introductory subjects such as discrete mathematics and computer logic, motivates the implementation of tools of this kind. Among the various types of software that can assist in learning we have interpreters. Interpreters are programs that execute instructions written in a specific language. In this work, the construction of a complete interpreter of set theory expressions for didactic purposes was addressed. For that, the basic compiler literature was used in conjunction with the Python language. As a result of the work, the set theory expression interpreter was implemented and its correctness was evidenced by experiments of empirical nature.

KEYWORDS: Interpreter. Expression. Sets.

Recebido: 19 ago. 2020.
Aprovado: 01 out. 2020.

INTRODUÇÃO

A escassez de software para auxílio do aprendizado dos alunos dos cursos superiores da área de computação, principalmente em matérias introdutórias como matemática discreta e lógica para computação, motiva a implementação de ferramentas do gênero. Um dos tópicos que surgem com frequência considerável em ementas de disciplinas básicas como matemática discreta e em disciplinas intermediárias como teoria da computação é a "teoria dos conjuntos", dado o notável papel que a mesma tem em firmar as bases da matemática e, por consequência, da computação.

Entretanto, mesmo quando se trata de ferramentas didáticas nos tópicos referentes a conjuntos, não há muitas ferramentas comuns como geradores e interpretadores de expressões que podem auxiliar nas tarefas de construção de provas, listas de exercício e slides. Como exemplo de interpretador semelhante já implementado se tem o "Interpretador de expressões lógicas" de Moreira, Silva, Ricardi, Penachio, Silva, Trindade e Rocha (2001, p.1) também voltado para fins didáticos, porém o mesmo tem seu foco em expressões da lógica boolena.

Entre os diversos tipos de softwares que podem trabalhar com as expressões da teoria dos conjuntos estão os interpretadores. Interpretadores, quando implementados por meio de uma linguagem de programação, são programas que executam instruções escritas em uma determinada linguagem, podendo ou não converter código fonte para código intermediário, de forma a facilitar construções complexas na mesma, como explica Farias (2013, p.66-72).

É importante notar que, mesmo comumente tendo uma estrutura mais simples e sendo mais fáceis de desenvolver, os interpretadores ainda têm a necessidade de passar pelas etapas de análise e validação de entradas que é dividida em três. Primeiramente se tem análise léxica, que possui a função de identificar os símbolos básicos da linguagem (*tokens*) a partir dos caracteres de entrada. Após a *tokenização* da entrada é necessária a análise sintática que tem como objetivo reconhecer a estrutura das sequências de *tokens*, gerados na análise léxica, validando e organizando para a próxima etapa. Ao final a análise semântica verifica a coerência e significado da entrada validada pelas etapas anteriores gerando a saída desejada.

Neste trabalho foi abordada a construção de um interpretador completo de expressões de teoria dos conjuntos para fins didáticos. Além disso, estão aqui expressos os processos de implementação de cada uma das análises citadas anteriormente e seus devidos testes de validação e, quando cabível, desempenho.

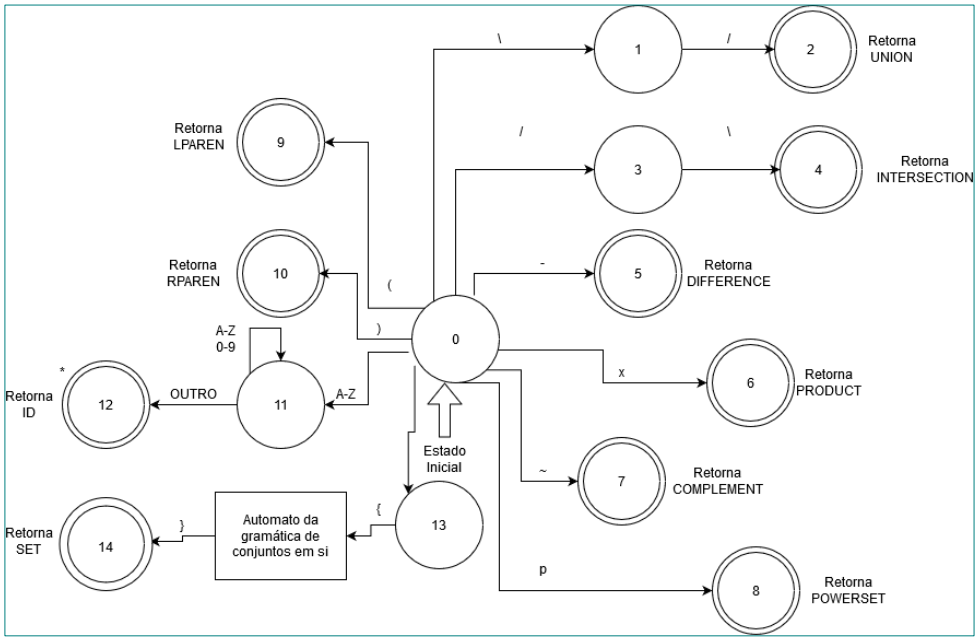
MATERIAIS E MÉTODOS

Para implementação da ferramenta foi escolhido o uso da linguagem Python (em sua versão 3.7) por ser adequada para uso no ensino de iniciantes, possuir uma grande comunidade de usuários ativos e ser usada no meio profissional, como dito por Van Rossum (1999, p.6). A priori, optou-se pelo uso da biblioteca PLY, uma implementação das ferramentas de *parsing lex* (que realiza análise lexical) e Yacc (que realiza análise sintática) voltada ao Python e criada por Beazley (2020, p.1), para construção das análises léxica e sintática, entretanto a Yacc demonstrou diversos erros de compilação relacionados a versão em que a PLY se encontrava e

a versão do Python utilizada. Por fim optou-se pela criação de implementações autorais de analisadores léxicos e sintáticos.

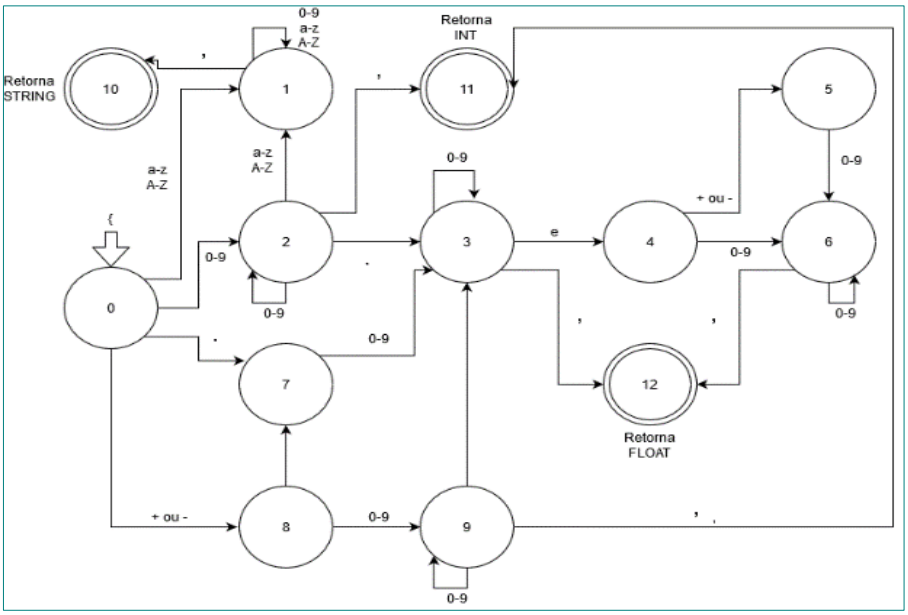
A construção do analisador léxico usou como base a teoria do capítulo 3 do livro “Compiladores: Princípios, técnicas e ferramentas” de Aho, Sethi e Ullman (1995, p.38-86), que é uma leitura clássica e fundamental a qualquer um que trabalhe com construção de compiladores. Mais especificamente foi implementado o autômato finito determinístico que representa as expressões de teoria dos conjuntos, representado nas Figuras 1 e 2 abaixo, em uma estrutura de “if e else” para extração de *tokens*, também foram utilizados dois apontadores para marcar o início e o fim dos lexemas nas cadeias de caracteres.

Figura 1 – Autômato que representa a gramática que rege expressões de teoria dos conjuntos. O “*” acima do estado 12 significa que o apontador fim de lexema deve retornar uma posição na leitura da entrada.



Fonte: Autoria própria

Figura 2 – Autômato que rege a gramática de declarações de conjuntos. Os retornos nos estados finais são os tipos dos elementos inseridos no conjunto. Este autômato está presente na figura 1 entre os estados 14 e 13.



Fonte: Autoria própria

É importante salientar que o autômato só trabalha com conjuntos que não possuam em si outros conjuntos como subelemento, aceita conjuntos vazios, ignora espaços excedentes que vem antes e após elementos do conjunto e não trabalha com os operadores “para todo” e “existe algum”. A Tabela 1 ilustra exemplos de expressões admissíveis e inadmissíveis pela ferramenta criada neste projeto.

Tabela 1 – Exemplo de entradas admitidas ou não pela análise léxica

Entrada	É admitida?
A = {}	Sim
~{0} – {1, 2} \/ {3} /\ p{0, 1} x {}	Sim
{3, 4, 5, {0, 1}}	Não
∀∈A∃0	Não
{ 0,1,2,azul , -23.4, fibonacci1123 }	Sim
B0 = {0, 1, 2} – {3}	Sim

Fonte: Autoria própria (2020).

Em seguida foram construídos 4 algoritmos de análise sintática. Primeiro foram implementados o “método de Unger sem produções vazias” e o “algoritmo Cocke-Younger-Kasami” com base na teoria do livro “Parsing Techniques-A Practical Guide” de Grune e Jacobs (1990, p.104-129). Posteriormente, utilizando as definições do capítulo 4 do mesmo livro usado na construção do analisador léxico, foi implementado o “analisador sintático preditivo não-recursivo” em conjunto com uma gramática LL(1) capaz de gerar as expressões de teoria dos conjuntos. Por fim o último método implementado foi um autômato com pilha em conjunto com o “algoritmo Shunting-yard”, explicado por Wolf (2011, p.1) na página da Pace University, para validação e análise sintática de expressões. Todos os algoritmos de análise sintática foram implementados utilizando a estrutura dicionário do Python, o que permitiu também que os mesmos trabalhassem com

diversos tipos de gramática. Neste artigo não será abordada a capacidade dos diversos algoritmos sintáticos trabalharem com gramáticas variadas, pois a única gramática de interesse dos autores é a de teoria dos conjuntos.

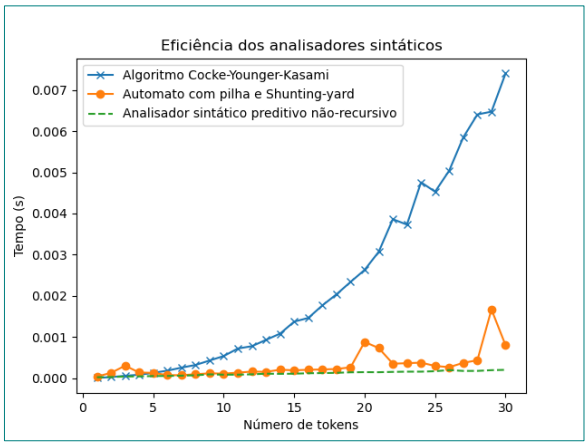
Após implementados, foram realizados testes de desempenho com os algoritmos de análise sintática e, por fim, optou-se em utilizar a abordagem do autômato com pilha em conjunto com "Shunting-yard" devido ao desempenho linear do mesmo. O uso do "Shunting-yard" também é justificado por sua saída ser a expressão em notação polonesa reversa (RPN), pois a mesma não precisa ser convertida de uma árvore sintática abstrata para uma arvore útil, facilitando a análise semântica e a obtenção do conjunto resultante da expressão.

Assim como expresso em suas respectivas literaturas fonte (referenciadas nos diversos parágrafos anteriores desta mesma sessão), o "algoritmo CYK" possui complexidade computacional não linear (assim como ilustrado na Figura 4), o "método de Unger" possui complexidade exponencial (observado na Figura 3), e tanto o "Shunting-yard" quanto o "analisador sintático preditivo não-recursivo" são lineares (exposto empiricamente na Figura 4).

Figura 3 – Desempenho do "método de Unger sem produções vazias". Ressalta-se que o método de Unger foi separado dos outros algoritmos no gráfico, pois o mesmo demonstrou comportamento assintótico exponencial e dificultaria a visualização do desempenho dos demais algoritmos.

Fonte: Autoria própria

Figura 4 – Desempenho dos algoritmos de análise sintática. O eixo x representa o número de tokens da expressão. O eixo y é relacionado ao tempo de execução necessário para análise das expressões (em segundos).



Fonte: Autoria própria

A construção da análise semântica consistiu em resolver as expressões em RPN buscando ou inserindo variáveis na tabela de endereços quando necessário, entretanto devido ao gasto de tempo de processamento e de memória que as operações conjunto das partes e produto cartesiano possuem optou-se por limitar ambas nesta parte da análise. A operação conjunto das partes só pode trabalhar com conjuntos de cardinalidade 8 ou menos. Já a operação produto cartesiano só pode ser efetuada caso o produto da cardinalidade dos conjuntos envolvidos na mesma seja 1000 ou menos. Em caso de erro durante a análise semântica, a mesma retornará o valor nulo e não um conjunto como resposta.

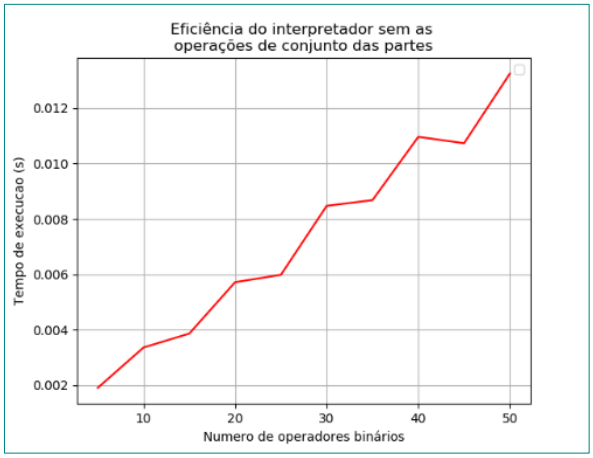
RESULTADOS E DISCUSSÕES

Por fim foram realizados testes de validação em cada uma das partes da análise das expressões e no sistema como um todo. Os testes consistiram em gerar entradas juntamente com as respostas esperadas e em seguida alimentar o sistema com as mesmas e comparar a saída gerada. Os testes trabalharam com expressões geradas aleatoriamente em função do número de operadores binários da linha, começando de 5 operações binárias até 50 em um intervalo de 5. Para cada número de operadores binários e cada parte do analisador foi gerado 10000 expressões de teste.

Mais expecificamente, os testes da análise léxica consistiam em gerar as expressões de entrada e as saídas tokenizadas e compará-las com os resultados produzidos pelo analisador léxico. Já os testes da análise sintática, de forma análoga, geravam as entradas e as saídas esperadas em RPN e comparavam as mesmas com as saídas produzidas pelo analisador sintático. Os testes da análise semântica e do sistema (como um todo) se basearam em gerar as expressões de entrada e suas saídas esperadas e comparar com as saídas obtida através do interpretador.

Tanto nos testes de validação de cada parte da análise quanto o teste do sistema como um todo não identificaram erros, além disso foi obtido o resultado ilustrado na Figura 5 do desempenho geral do interpretador. O desempenho em cada ponto do gráfico foi calculado através da média simples do tempo de execução de cada expressão gerada. Ressalta-se que o gráfico da figura 5 sugere que o interpretador possui desempenho linear.

Figura 5 – Desempenho do interpretador como um todo, ou seja, desde o processo de análise léxica até a obtenção do conjunto resposta.



Fonte: Autoria própria.

CONCLUSÃO

Como resultado do trabalho o interpretador de expressões de teoria dos conjuntos completo foi implementado e seu desempenho exposto como linear através de experimentos de natureza empírica. Como produtos secundários do projeto temos os algoritmos genéricos “CYK”, “método de Unger sem produções vazias”, “analisador sintático preditivo não-recursivo”, autômato com pilha e o “algoritmo Shunting-yard” que podem trabalhar com gramáticas diversas. Também foi obtido o desempenho de cada um dos algoritmos de análise sintática testados.

Salienta-se que a corretude do interpretador, em termos práticos, foi fortemente evidenciada em todos os testes. Sumarizando foram feitos experimentos de corretude da análise léxica que averiguou o processo de tokenização, os experimentos de desempenho dos diversos analisadores sintáticos e o experimento de corretude da abordagem “Shunting-yard” com autômato com pilha e o experimento de desempenho/corretude do interpretador como um todo.

Devido à falta de interface gráfica a ferramenta ainda não é adequada para o uso didático e devido à natureza assintótica das operações produto cartesiano e conjunto das partes é importante fazer uso das mesmas com certa prudência, pois o desempenho (tempo e gasto de memória) do analisador pode ser prejudicado. Como continuidade do trabalho será implementada uma interface gráfica para o interpretador de forma a reforçar o caráter didático do mesmo, verificar a corretude dos demais analisadores sintáticos e disponibilizar o projeto em uma plataforma de versionamento de código que permita sua distribuição livre para comunidade.

AGRADECIMENTOS

Os autores agradecem ao apoio financeiro do Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq).

REFERÊNCIAS

AHO, Alfred V.; SETHI, Ravi; ULLMAN, Jeffrey D. Compiladores: Princıpios, técnicas e ferramentas. LTC, Rio de Janeiro, Brasil, p.38-86, 1995.

BEAZLEY, David. Ply (python lex-yacc). Disponível em: http://www.dabeaz.com/ply, Acesso em: 25 ago. 2020.

FARIAS, Gilberto; SANTANA, Eduardo. Introdução à computação. v1. 0, Universidade Aberta do Brasil, p.66-72, 2013.

GRUNE, Dick; JACOBS, Ceriel. Parsing Techniques–A Practical Guide, p.104-129, 1990. VU University. Amsterdam.

MOREIRA, Almir; SILVA, Andrey P.; RICARDI, Cláudio; PENACHIO, Alisson M.; SILVA, Celso M.; TRINDADE, Luiz C.R.; ROCHA, Reginaldo, et al. Interpretador de Expressões Lógicas. 2001.

VAN ROSSUM, Guido et al. Computer programming for everybody. Proposal to the Corporation for National Research Initiatives, 1999.

WOLF, Carol E.; Infix to postfix conversion algorithm. Disponível em: http://csis.pace.edu/~wolf/CS122/infix-postfix.html, Acesso em: 25 ago. 2020.