Trabalho final para disciplina de Linguagens Formais e Autômatos

Everton de Assis Vieira[1], **Gabriel H. Moro**[1]

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

eassis.vieira@gmail.com.br, gabrielhmoro@gmail.com

Abstract. *This paper describes and explain the planning and implementation of an aplication developed to create a non-deterministic finite automaton, eliminate the epsilon transitions, convert it to a deterministic automaton and minimize the resulting automata, but not using equivalence class. This application has been implemented using the programming language Python. This task was carried out for the discipline of Formal Languages and Automata and aims to validate the knowledge acquired in class during the semester.*

Keywords– deterministic automata, minimizate

Resumo. *Este artigo descreve e explica o planejamento e a implementação de uma aplicação desenvolvida para gerar um autômato finito não deterministico, eliminar as epsilon transições existentes, determinizá-lo e minimizar o autômato resultante, porém sem o uso de classes de equivalência. A aplicação foi implementada utilizando a linguagem de programação Python. Este trabalho foi realizado para a disciplina de Linguagens Formais e Autômatos e tem como objetivo a validação dos conhecimentos adquiridos em aula durante o semestre letivo.*

Palavas-chave– autômato determinístico, minimização

1. Introdução

A proposta deste relatório é a explicação do processo e resultado final da criação de um autômato finito determinístico, gerado após a eliminação de epsilon transições, determinização e minimização (sem a aplicação de classes de equivalência).

Este trabalho foi realizado para disciplina de Linguagens Formais e Autômatos e auxilia na aprendizagem do que foi visto em sala de aula, possibilitando a visualização prática de vários conceitos estudados durante o semestre.

O estudo sobre esse assunto é importante pois possibilita a compreensão sobre uma área da computação muito relevante e que tem diversas aplicações no cotidiano. Além disso, conhecer e entender mais sobre linguagens formais e autômatos facilita o desenvolvimento de diversas aplicações em áreas variadas, como por exemplo, modelagem de circuitos, construção de compiladores, tratamento de linguagens não lineares, entre outros.

Neste artigo será apresentada a implantação de uma aplicação de um autômato finito determinístico minimizado. Os conceitos específicos e todo os processo de desenvolvimento serão apresentados nas próximas seções.

2. Referencial Teórico Básico

Para a compreensão do trabalho descrito neste artigo é necessário a compreensão dos seguintes conceitos:

Alfabeto, em *Theory of Finite Automata* é descrito como ”[...] um conjunto finito não vazio de símbolos” [Carroll and Long 1989, tradução nossa] e em *Linguagens Formais e Autômatos* também é definido como ”[...] um conjunto finito de Símbolos” [Menezes 2008], porém, diferentemente de Carroll e Long, Menezes diz que um conjunto vazio também pode ser considerado um alfabeto.

A partir de um alfabeto é possível se construir palavras, ou *strings*, definidas, segundo Carroll e Long, como uma ”[...]sequência finita de símbolos do alfabeto”
cite[tradução nossa]carroll1989theory, não diferenciando da definição de Menezes, que também define a palavra vazia, ”representada pelo símbolo ε, é uma palavra sem símbolo” [Menezes 2008], o que é explicado posteriormente por Carroll e Long como ”denotado por λ, é [...] a única palavra composta por zero letras” [Carroll and Long 1989, tradução nossa].

Uma Linguagem Formal, para Menezes, é ”um conjunto de palavras sobre um alfabeto” [Menezes 2008], e segundo Carroll e Long, uma linguagem pode ser finita ou infinita. Tem-se também que ”uma linguagem sobre um Σ [alfabeto] não não precisa incluir *strings* com todos os símbolos de Σ [alfabeto], [...]” [Hopcroft et al. 2001, tradução nossa].

Segundo Menezes, uma gramática é ”[...] uma quádrupla ordenada”que possui um ”conjunto finito de símbolos [...] *não-terminais*”V, um ”conjunto finito de símbolos *terminais*”T, disjuntos de V, um ”conjunto finito de pares, denominados *regras de produção*”P e um ”elemento de V denominado variável inicial”S.

Ainda segundo Menezes, as condições para a geração das palavras da linguagem são definidas pelas regras de produção, e ”a aplicação de uma regra de produção é denominado derivação de uma palavra”e que aplicando-se as regras de produção sucessivamente geram-se as palavras da linguagem que é representada pela gramática. A linguagem gerada pela gramática é ”composta por todas as palavras de símbolos terminais deriváveis a partir do símbolo inicial S” [Menezes 2008].

Além disso, define-se Autômato Finito, segundo Menezes como ”[...] um sistema de estados finitos [...] o qual constitui um modelo computacional do tipo sequencial muito comum em diversos estudo teórico-formais da computação em informática”. Ainda segundo Menezes, um autômato finito é uma quíntupla composta por um alfabeto que contém os símbolos da entrada, um conjunto dos possíveis estados do autômato, o que determina se o autômato é ou não finito pela quantidade destes, função de transição, um estado inicial e um conjunto de estados finais que pertence ao conjunto de estados do autômato.

Um autômato é dito não-determinístico quando um estado possui, para um mesmo símbolo do alfabeto, mais de uma transição de estado e é dito determinístico quando em cada estado e para um símbolo qualquer do alfabeto há somente uma transição.

A determinização de um autômato finito não determinístico ou conversão para autômato finito determinístico, segundo Ricarte, tem como base ”criar novos estados que representem todas as possibilidades de estados originais em um dado momento da análise da sentença em processo de reconhecimento” [Ricarte 2014].

A eliminação das epsilon transições, necessária para a determinização do autômato, que se dá ao incluir nas regras que as possuem epsilon transição as produções alcançadas por esta.

A determinização se dá com a conversão de conjuntos de estados em produções em estados únicos, criando novos estados que possuem as produções dos estados presentes no conjunto de

estados.

Por fim, podemos descrever a minimização de um autômato consiste em controlar a mudança de estado do autômato a cada vez que um símbolo é lido na entrada, de acordo com Menezes, e para isto ocorre a eliminação de estados inalcançáveis, que consiste na remoção dos estados do autômato que não são alcançáveis a partir do estado inicial, e estados inúteis, que são estados que não alcançam estados finais a partir de qualquer derivação de suas produções, não aplicando, para este trabalho, classes de equivalência.

3. Implementação

Durante a estruturação do trabalho foram levadas em considerações as características das linguagens C++ e Python para escolher qual seria a melhor escolha para o desenvolvimento da aplicação.

Para a implementação do autômato finito determinístico foi utilizado a linguagem de programação *Python*, tendo como principais motivações as facilidades e as propriedades da linguagem, bem como o interesse no aprendizado da linguagem.

As entradas planejadas para a realização dos testes referentes ao funcionamento correto da implementação foram colocadas em um arquivo *entrada.in*. A entrada é composta por um conjunto de tokens/*strings* e por um conjunto com uma ou mais gramáticas regulares.

O arquivo *main.py* realiza o tratamento das entradas já definidas, chama as funções que realizam a construção do autômato finito não determinístico, presentes no arquivo *afd.py*. Para os tokens, cada token é lido e, individualmente, enviado para a função de criação do autômato. Para as gramáticas são criados autômatos temporários que são então unidos ao autômato principal, a função de criação recebe uma gramática inteira por vez e nas produções onde há símbolo não-terminal mas não há terminal insere-se ε como símbolo terminal.

O arquivo *afd.py* cria o autômato não determinístico para os tokens e gramáticas presentes no arquivo de entrada, que serão tratados através das funções do arquivo *determinizacao.py*, onde ocorre a eliminação de epsilon transições e a determinização.

A construção do autômato ocorre, para os tokens, com a leitura de cada símbolo do token e, caso seja a regra inicial, verifica-se se já existe uma transição por este símbolo para algum estado e, caso não, cria-se no autômato uma transição deste símbolo para a próxima estado a ser criada, e caso sim, adiciona-se ao rótulo já existente uma nova transição para o próximo estado. Já para a inserção em outras regras que não a inicial, insere-se no autômato um novo estado que possui transição pelo símbolo lido para um novo estado que será criado com o próximo símbolo. Quando já não há mais caracteres/símbolos para se ler no token define-se o estado novo como final.

Já para as gramáticas criam-se autômatos temporários que serão unidos no autômato principal. Estes autômatos temporários são criados primeiramente mapeando-se o nome das regras lidas, caso não tenha sido ainda, para então ler as produções das regras. Lendo uma regra e suas produções, identificam-se os símbolos terminais e não terminais de uma produção e então, para cada produção verifica-se: caso não haja símbolo não-terminal, insere-se uma transição pelo símbolo terminal para um estado final, e caso haja símbolo não-terminal, se este não esteja mapeado, mapeia-se a regra como o próximo estado do autômato e insere-se uma transição pelo símbolo terminar para o estado mapeado pelo símbolo não-terminal. Também, caso o símbolo terminal seja ε, insere-se na lista de epsilon transições o índice que representa estado que possui uma produção com um terminal ε.

O arquivo *determinizacao.py* possui as funções de eliminação de epsilon transição e determinização, como dito anteriormente.

A função de eliminação de epsilon transição funciona da seguinte forma: a função recebe o autômato não determinístico e uma lista com o rótulo dos estados que possuem epsilon transição, da qual é feita uma cópia para verificações posteriores, e então, a partir desta lista encontram-se, os estados que possuem epsilon transições que devem ser eliminadas. Para tal, tem-se uma função recursiva que, a partir de um estado que possui epsilon transição remove-se este da lista de estados que possuem epsilon transição vê-se o estado atingido, copiam-se as produções do estado atingido para o estado que possui a transição e verifica-se se o estado atingido possui epsilon transição, e caso possua ocorre o mesmo processo, agora no estado atingido a partida da epsilon transição. Este processo ocorre até que se ache um estado que não possua epsilon transições ou que a lista de estados que possuem epsilon transição esteja vazia. Ao final do processo são removidas as epsilon transições das regras presentes na cópia da lista de estados que possuem epsilon transições.

A determinização de autômato finito indeterminístico consiste na remoção de transições que possuem indeterminismos, ou seja, transições onde um símbolo do alfabeto possui mais de uma transição em um mesmo estado. A eliminação dos indeterminismos se dá a partir da criação de um novo estado há indeterminismo, sendo que as transições do novo estado são as transições dos estados que o geram. Para a determinização, usa-se um método de busca em grafo chamado *Busca em Profundidade* (em inglês, *Depth-first search*, abreviado para DFS), onde, quando se cria um novo estado, são inseridas as transições dos estados que o geram e ocorre uma verificação nas transições do novo estado referente a necessidade da criação de novos estados, criando quando necessário e repetindo o processo enquanto existir a necessidade de criação de novos estados. O resultado destes processos é um autômato finito determinístico.

No arquivo *minimizacao.py*, como o próprio nome sugere, estão as funções para a minimização do autômato finito deterministico gerado após a aplicação das funções presentes no arquivo *determinizacao.py*. Para a minimização foram realizadas dois processo, a eliminação de estados inalcançáveis e a eliminação de estados inúteis.

Para eliminação de estados inalcançáveis foi usado também uma DFS, que consiste em marcar o estado atual como visitado e visitar o primeiro estado não visitado alcançável a partir das produções do estado atual, isto até que não seja mais possível visitar nenhum estado, tendo no final do processo uma lista com todos os estados visitados. Assim, tendo o autômato que possui todos os estados e a lista de estados visitados, é possível determinar os estados inalcançáveis, que são aqueles que não foram visitados no processo da DFS, e remover, primeiramente, as transições de outros estados para estes e então remove-los do autômato.

Foi realizada também a eliminação de estados inúteis, que são os estados a partir dos quais não é possível se atingir um estado final a partir de suas derivações. Para tal, foi utilizado uma DFS na qual, para cada estado do autômato, visita-se o primeiro estado não visitado e, quando se atinge um estado final, este é adicionado a uma lista de estados úteis. Qualquer estado que possua transição para um estado útil é marcado como útil, e com isso, ao se atingir um estado final, todos os estados visitados até atingi-lo são marcados como úteis, pois o estado que atinge o terminal é útil, o estado que atinge este também, e assim até se chegar na regra inicial. Com isto se tem uma lista dos estados úteis, a qual é utilizada numa verificação estado a estado onde, caso o estado verificado não esteja na lista, este é eliminado. Após isso são removidas nos estados remanescentes as transições para os estados eliminados e com isto se tem um autômato finito determinístico mínimo.

Há também um arquivo denominado *utilizaveis.py*, onde foram colocadas as funções que tem tarefas específicas não diretamente relacionadas ao conteúdo do trabalho, mas que facilitam a execução das funções de determinização e minimização.

4. Testes e Resultados

Para validar a implementação da aplicação são feitos testes diretamente no terminal, testando as saídas em cada execução. O arquivo de entrada (*entrada.in*) continha, para os testes, os seguintes tokens e gramáticas.

Um exemplo dos testes executados é apresentado a seguir:

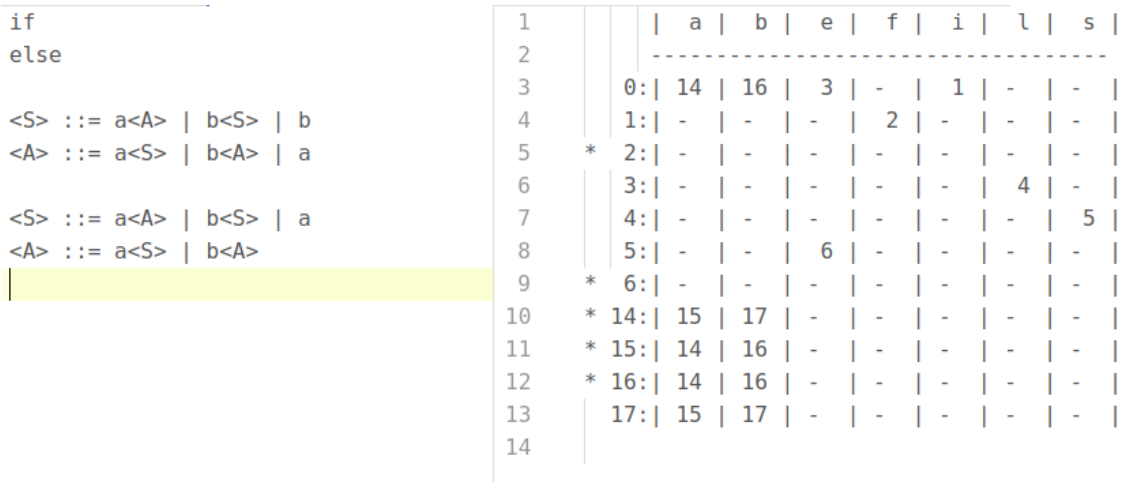

Figura 1. Teste

A imagem, dividida em duas partes, possui, a esquerda, a entrada, e, a direita, a saída. Na entrada observam-se dois tokens e duas gramáticas, sendo que a primeira gramática reconhece uma palavra com uma quantia par de *a's* encerrando por *a* ou *b*, e a segunda gramática reconhece uma palavra com uma quantia impar de *a's*, não sendo possível encerrar a palavra com o símbolo *b*.

Na saída mostra o autômato mínimo resultante, no qual os estados 2, 6, 14, 15 e 16 são finais. O estado 2 reconhece o token *if*. O estado 6 reconhece o token *else*. O estado 14 reconhece palavras com quantia impar de *a's*. O estado 15 reconhece palavras com quantia par de *a's* e terminadas em *a*. Finalmente, o estado 16 reconhece palavras com quantia par de *a's* e terminadas em *b*.

Para melhor visualização foi inserido o símbolo − para representar a transição para o estado de erro.

5. Conclusões

Podemos concluir que foi possível atingir todos os objetivos solicitados pelo professor. Algumas dificuldades foram encontradas durante a implementação da aplicação referentes a utilização da linguagem escolhida, mesmo ela apresentando mais benefícios do que a linguagem anteriormente cogitada C++, estas dificuldades se deram, principalmente, devido a inexperiência com a linguagem Python.

Foi necessário um estudo mais profundo sobre os conceitos utilizados para facilitar a organização e desenvolvimento do trabalho, bem como o estudo da linguagem escolhida e de suas funcionalidades, para decidir quais mais se adequavam ao desejado para a implementação.

É possível afirmar que este trabalho foi extremamente útil para a compreensão sobre o conteúdo. Além disso, o trabalho possibilitou a percepção sobre a importância das linguagens

formais e autômatos em áreas não só do curso de Ciência da Computação, mas também em outras áreas presentes em nosso cotidiano, segundo os artigos e livros lidos.

Referências

Arnold, A., Dicky, A., and Nivat, M. (1992). A note about minimal non-deterministic automata. *Bulletin of the EATCS*, 47:166–169.

Carroll, J. and Long, D. (1989). *Theory of finite automata with an introduction to formal languages*. Prentice-Hall, Inc.

Hopcroft, J. E., Motwani, R., and Ullman, J. D. (2001). Introduction to automata theory, languages, and computation. *Acm Sigact News*.

Menezes, P. (2008). *Linguagens formais e autômatos*. Série livros didáticos. Bookman.

Ricarte, I. L. M. (2014). Conversão para autômato finito determinístico. Disponível em: http://www.dca.fee.unicamp.br/cursos/EA876/apostila/HTML/node48.html. Último acesso: 06/07/2018.