**Programação de Linguagens**   
  
Instituto Politécnico do Cávado e do Ave  
Escola Superior de Tecnologia  
Licenciatura Engenharia de Sistemas Informáticos  
Ano Letivo: 2023/2024

**Relatório**

**Trabalho Prático**

Joana Freitas Pimenta – 22999

Diogo Manuel Marques – 23000

Ludgero Miguel Simões – 23135

Índice

[Introdução 3](#_Toc163776021)

[Desenvolvimento 4](#_Toc163776022)

[Autómatos Finitos Deterministas 4](#_Toc163776023)

[Expressão Regular para AFND 6](#_Toc163776024)

[Conversão de AFND para AFD 7](#_Toc163776025)

[Conclusão 10](#_Toc163776026)

[Figura 1 – Execução do script ‘afd\_main.py’ com o argumento ‘-help’ 4](#_Toc163776031)

[Figura 2 - Execução do script ‘afd\_main.py’ com o argumento ‘-rec’ submetendo uma string não reconhecida 4](#_Toc163776032)

[Figura 3 - Execução do script ‘afd\_main.py’ com o argumento ‘-rec’ submetendo uma string reconhecida 5](#_Toc163776033)

[Figura 4 - Execução do script ‘afd\_main.py’ com o argumento ‘-graphviz’ 5](#_Toc163776034)

[Figura 5 - Preview do gráfico utilizando a extenção ‘Graphviz’ do ‘VSCode’ 6](#_Toc163776035)

[Figura 6 - Execução do script ‘er\_main.py’ com o argumento ‘-help’ 6](#_Toc163776036)

[Figura 7 - Execução do script ‘er\_main.py’ com um arquivo JSON 7](#_Toc163776037)

[Figura 8 - Execução do script ‘afnd\_main.py’ com o argumento ‘-help‘ 7](#_Toc163776038)

[Figura 9 - Execução do script ‘afnd\_main.py’ com o argumento ‘-output‘ 8](#_Toc163776039)

[Figura 10 - Execução do script ‘afnd\_main.py’ com o argumento ‘-graphviz‘ 8](#_Toc163776040)

[Figura 11 - Preview do gráfico gerado utilizando a extenção ‘Graphviz’ do ‘VSCode’ 9](#_Toc163776041)

# Introdução

No presente relatório, aborda-se o trabalho prático realizado em contexto da unidade curricular de Processamento de Linguagens, no qual propõem-se explorar a implementação de reconhecedores de linguagens regulares. O foco reside na aplicação dos algoritmos e estruturas de dados discutidos ao longo das aulas, com especial destaque para a linguagem de programação Python, conforme exemplificado durante a UC.

Como objetivos deste trabalho, primeiramente, pretende-se demonstrar a relevância das expressões regulares no processamento de linguagens. Em seguida, busca-se definir expressões regulares para o reconhecimento de elementos simples e implementar analisadores baseados em autômatos finitos deterministas. Além disso, visa-se compreender o processo de desenvolvimento de ferramentas reconhecedoras de expressões regulares, bem como a implementação e adaptação de autômatos finitos, seja determinista ou não determinista.

Ao longo deste relatório, detalhar-se-á cada etapa do desenvolvimento, desde a representação dos autômatos em formato JSON até a implementação das regras de conversão entre expressões regulares, autômatos finitos não deterministas e deterministas.

Segue-se uma análise detalhada das etapas dos trabalhos práticos, desde a representação das expressões regulares até a implementação dos algoritmos de conversão entre diferentes tipos de autômatos finitos.

# Desenvolvimento

## Autómatos Finitos Deterministas

Começou-se por projetar um script, executado a partir da linha de comando, recebendo argumentos específicos para realizar operações como reconhecimento de palavras em um autômato ou geração de gráficos utilizando Graphviz.

Inicialmente, o script verifica os argumentos passados pela linha de comando, procurando pelas opções '-help', para exibir informações sobre como utilizar o script, '-rec', para especificar uma palavra a ser reconhecida pelo autômato, '-graphviz', para gerar um diagrama do autômato utilizando Graphviz.

A black background with white text

Description automatically generated

Figura 1 – Execução do script ‘afd\_main.py’ com o argumento ‘-help’

Em seguida, o script lê um arquivo JSON que descreve a definição do autômato finito determinista (AFD). Esse arquivo contém informações como o conjunto de estados (Q), o alfabeto (V), a função de transição (delta), o estado inicial (q0) e o conjunto de estados finais (F).

Após carregar as informações do autômato, o script define uma função 'reconhece' que recebe uma palavra como entrada e determina se essa palavra é reconhecida ou não pelo autômato. Para isso, a função percorre a palavra, verificando se cada símbolo pertence ao alfabeto e se existe uma transição definida para esse símbolo a partir do estado atual. Caso a palavra seja reconhecida, a função retorna uma mensagem que indica o reconhecimento e o caminho percorrido pelo autômato. Caso contrário, retorna uma mensagem indicando que a palavra não foi reconhecida e o motivo, como a falta de uma transição definida.

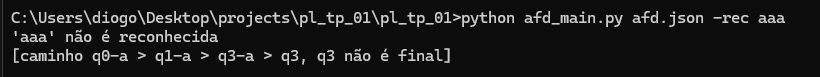


Figura 2 - Execução do script ‘afd\_main.py’ com o argumento ‘-rec’ submetendo uma string não reconhecida

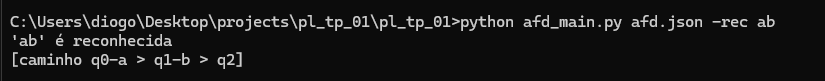


Figura 3 - Execução do script ‘afd\_main.py’ com o argumento ‘-rec’ submetendo uma string reconhecida

Além disso, o script define uma função chamada 'graphviz\_gen' que gera um arquivo no formato DOT para representar o autômato utilizando Graphviz. Essa função percorre as transições do autômato e gera as instruções necessárias para desenhar as setas correspondentes no diagrama. Se nenhum caminho for especificado, a função imprime o código DOT na saída padrão. Caso contrário, escreve o código DOT em um arquivo especificado pelo caminho fornecido.

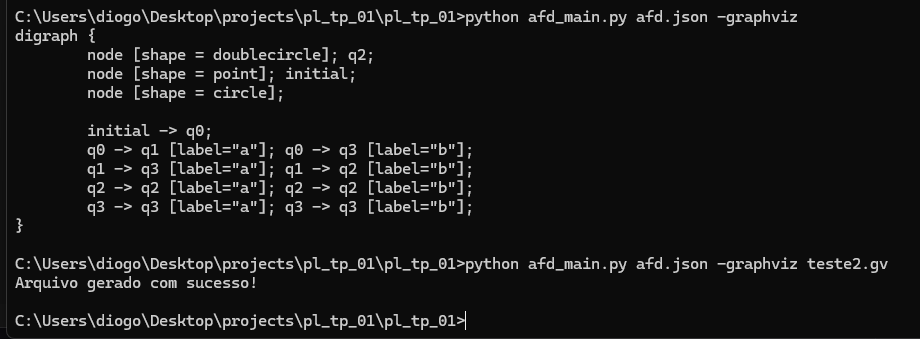


Figura 4 - Execução do script ‘afd\_main.py’ com o argumento ‘-graphviz’

Por fim, o script verifica se foi especificada uma palavra para reconhecimento ou se a geração do diagrama Graphviz foi solicitada. Dependendo das opções escolhidas, o script chama a função 'reconhece' ou 'graphviz\_gen' para realizar a operação desejada.

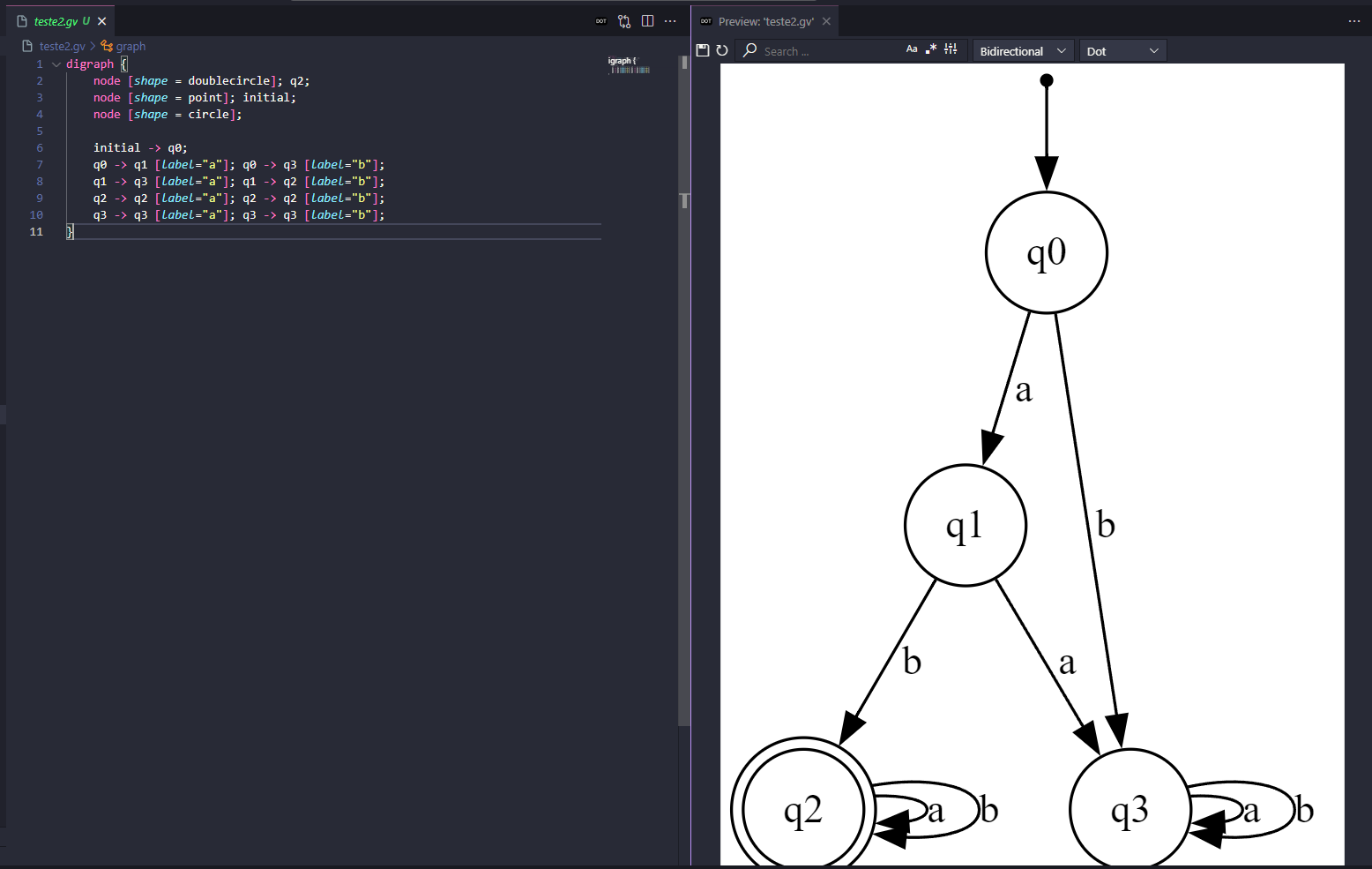


Figura 5 - Preview do gráfico utilizando a extenção ‘Graphviz’ do ‘VSCode’

## Expressão Regular para AFND

Seguiu-se então com o desenvolvimento do sript ‘er\_main.py’, que recebe como argumentos um arquivo JSON que descreve a expressão regular e um parâmetro '--output' para especificar o arquivo onde o AFND convertido será salvo.

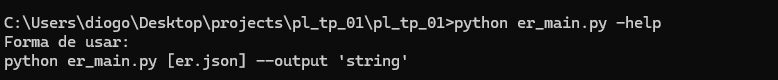


Figura 6 - Execução do script ‘er\_main.py’ com o argumento ‘-help’

Inicialmente, o script verifica os argumentos passados pela linha de comando, procurando pelas opções pre-definidas. Em seguida, lê o arquivo JSON que descreve a expressão regular.

Após carregar a expressão regular, o script define várias funções para processar os diferentes tipos de operadores presentes na expressão regular, como símbolos individuais, alternância ('alt'), sequência ('seq'), fecho de Kleene ('kle') e fecho transitivo ('trans'). Cada função é responsável por criar os estados e transições necessários para representar o operador correspondente no AFND.

Além disso, o script define uma função principal chamada 'convertERToAFND', que inicia o processo de conversão da expressão regular para o AFND. Esta função utiliza as funções auxiliares mencionadas anteriormente para construir o AFND com base na estrutura da expressão regular.

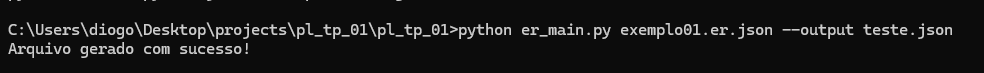


Figura 7 - Execução do script ‘er\_main.py’ com um arquivo JSON

Após construir o AFND, o script o salva em um arquivo JSON especificado pelo parâmetro '--output'.

Em resumo, o script é capaz de converter uma expressão regular em um Autômato Finito Não Determinístico, seguindo as regras e operadores definidos na expressão regular.

## Conversão de AFND para AFD

Continuou-se para o script ‘afnd\_main.py’ projetado para converter um Autômato Finito Não Determinístico (AFND) em um Autômato Finito Determinístico (AFD). Inicialmente, o script verifica os argumentos passados pela linha de comando, procurando pela opção '-help', ‘output’ ou ‘graphviz’. Em seguida, lê o arquivo JSON que descreve o AFND.

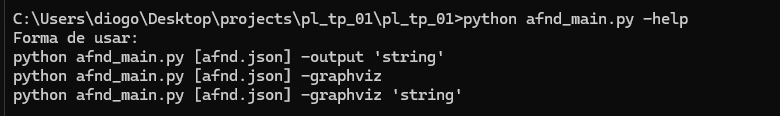


Figura 8 - Execução do script ‘afnd\_main.py’ com o argumento ‘-help‘

Após carregar o AFND, o script define várias variáveis para armazenar os componentes do AFND, como os conjuntos de estados (Q), símbolos do alfabeto (V), função de transição (delta), estado inicial (q0) e estados finais (F).

O script inclui uma função principal chamada 'convertAFNDtoAFD', responsável por realizar a conversão do AFND para o AFD. Esta função utiliza um algoritmo de conversão que explora todos os possíveis estados alcançáveis a partir do estado inicial do AFND, construindo as transições correspondentes no AFD. Ele utiliza uma fila para controlar os estados a serem processados e uma abordagem iterativa para construir as transições do AFD.

Após converter o AFND para o AFD, o script salva o AFD em um arquivo JSON especificado pelo parâmetro '-output'. Além disso, se a opção '-graphviz' for especificada, o script pode gerar um arquivo Graphviz para visualizar o AFND.



Figura 9 - Execução do script ‘afnd\_main.py’ com o argumento ‘-output‘

A computer screen with white text

Description automatically generatedEm resumo, o script é capaz de converter um Autômato Finito Não Determinístico em um Autômato Finito Determinístico, seguindo as regras e componentes definidos no AFND. Ele fornece uma maneira conveniente de converter e visualizar autômatos finitos, facilitando a compreensão e análise de expressões regulares e linguagens formais.

Figura 10 - Execução do script ‘afnd\_main.py’ com o argumento ‘-graphviz‘

A screenshot of a computer

Description automatically generated

Figura 11 - Preview do gráfico gerado utilizando a extenção ‘Graphviz’ do ‘VSCode’

# Conclusão

Em conclusão, o código desenvolvido demonstra uma abordagem eficiente para a conversão de Autômatos Finitos Não Determinísticos (AFNDs) em Autômatos Finitos Determinísticos (AFDs). Ao utilizar uma combinação de estruturas de dados adequadas e algoritmos bem definidos, o script é capaz de processar autômatos complexos e gerar representações determinísticas equivalentes de forma precisa e eficaz.

Através da análise detalhada dos componentes do AFND e da aplicação de um algoritmo iterativo de conversão, o código constrói as transições necessárias do AFD, garantindo que todas as transições possíveis sejam exploradas a partir do estado inicial. Além disso, a capacidade de gerar visualizações gráficas do AFND por meio do Graphviz aumenta a utilidade e a compreensão do processo de conversão.

Essa abordagem oferece uma solução versátil e poderosa para lidar com expressões regulares e linguagens formais, permitindo a análise e a manipulação eficiente de autômatos finitos em contextos diversos, desde a compilação de linguagens de programação até a análise de linguagens naturais. Em suma, o código representa uma contribuição significativa para a área de processamento de linguagens formais, proporcionando uma ferramenta valiosa para engenheiros, pesquisadores e entusiastas interessados em autômatos finitos e teoria da computação.