Relatório do trabalho da disciplina de Processamento de Linguagens

TP01 - Linguagens Regulares

Hugo Filipe Nogueira Silva – a16368

Hugo Daniel Neves Poças - a26339

Pedro Ricardo Rocha Silva - a26342

ESIPL

Barcelos, 2024

|  |
| --- |
| Afirmo por minha honra que não recebi qualquer apoio não autorizado na realização deste trabalho prático. Afirmo igualmente que não copiei qualquer material de livro, artigo, documento web ou de qualquer outra fonte exceto onde a origem estiver expressamente citada. |

Hugo Filipe Nogueira Silva – a16368

Hugo Daniel Neves Poças - a26339

Pedro Ricardo Rocha Silva - a26342

**Índice**

[Autómatos Finitos Deterministas (AFD) – Exercício A 5](#_Toc164270425)

[Introdução 5](#_Toc164270426)

[Estrutura do Código 5](#_Toc164270427)

[Leitura da Definição do AFD 5](#_Toc164270428)

[Validações do Autómato 6](#_Toc164270429)

[Gerar Representação Gráfica do Grafo 8](#_Toc164270430)

[Reconhecimento de Palavras 9](#_Toc164270431)

[Utilização do Código 10](#_Toc164270432)

[Exemplos de utilização: 10](#_Toc164270433)

[Conclusão 10](#_Toc164270434)

[Expressão Regular para AFND – Exercício B 11](#_Toc164270435)

[Introdução 11](#_Toc164270436)

[Estrutura do Código 11](#_Toc164270437)

[Operações Básicas nos AFNDs 11](#_Toc164270438)

[Concatenação: 11](#_Toc164270439)

[União: 12](#_Toc164270440)

[Fecho de Kleene: 13](#_Toc164270441)

[Transição Vazia: 14](#_Toc164270442)

[Conversão da Expressão Regular num AFND 15](#_Toc164270443)

[Utilização do Código 16](#_Toc164270444)

[Exemplo de Utilização: 16](#_Toc164270445)

[Conclusão 16](#_Toc164270446)

[Conversão de AFND para AFD – Exercício C 17](#_Toc164270447)

[Introdução 17](#_Toc164270448)

[Estrutura do Código 17](#_Toc164270449)

[Subfunções Utilizadas: 17](#_Toc164270450)

[Processo de Conversão: 18](#_Toc164270451)

[Utilização do Código 21](#_Toc164270452)

[Exemplo de Utilização: 21](#_Toc164270453)

[Conclusão: 21](#_Toc164270454)

# Autómatos Finitos Deterministas (AFD) – Exercício A

## Introdução

Este capítulo apresenta uma implementação em Python de um Algoritmo de Reconhecimento de Linguagens baseado em Autómatos Finitos Deterministas (AFD). O código fornece funcionalidades para ler a definição de um AFD a partir de um ficheiro JSON, gerar a representação gráfica do grafo associado ao AFD utilizando a biblioteca Graphviz e reconhecer palavras de acordo com as regras definidas pelo AFD.

## Estrutura do Código

O código está estruturado em três partes principais: a leitura da definição do AFD a partir de um ficheiro JSON, a geração da representação gráfica do grafo do AFD e o reconhecimento de palavras.

### Leitura da Definição do AFD

O método carregar\_automato(ficheiro\_definicao) é responsável por carregar a definição do AFD a partir de um ficheiro JSON. Este método utiliza a biblioteca padrão json do Python para carregar o conteúdo do ficheiro e retorná-lo como um dicionário Python.



### Validações do Autómato

A função validaAutomato tem como objetivo validar a definição de um autómato representado por um dicionário em Python. Esta função verifica diversos critérios importantes para garantir a correta definição do autómato, abrangendo desde a estrutura básica até detalhes específicos relacionados às transições e símbolos.

**Verificação do Tipo de Dados:**

A função inicia verificando se o argumento passado é de facto um dicionário. Isso é fundamental, pois a função espera receber um dicionário que represente um autómato. Caso contrário, o programa é encerrado com uma mensagem de erro indicando que o ficheiro JSON deve ser um dicionário.



**Verificação dos Estados Finais:**

Em seguida, verifica-se se o autómato possui estados finais definidos. Se não houver estados finais definidos, o programa é encerrado com uma mensagem de erro informando que o autómato não tem estados finais definidos.



**Verificação dos Estados Finais no Autómato:**

Posteriormente, a função verifica se todos os estados finais definidos no autómato realmente existem nos estados do autómato. Se algum estado final definido não existir, o programa é encerrado com uma mensagem de erro indicando que nem todos os estados finais estão definidos no autómato.



**Verificação dos Estados do Autómato:**

Após isso, verifica-se se o autómato possui estados definidos. Se não houver nenhum estado definido, a função adiciona os estados com base nas transições presentes. Por outro lado, se já houver estados definidos, a função verifica se todos os estados presentes nas transições também estão definidos como estados do autómato.



**Verificação dos Símbolos do Autómato:**

Em seguida, a função verifica se o autómato possui símbolos definidos. Se não houver nenhum símbolo definido, a função obtém todos os símbolos das transições, seleciona apenas os valores únicos e ordena a lista.



**Verificação do Símbolo 'ε':**

É realizada uma verificação adicional para garantir que o símbolo 'ε' não esteja presente nos símbolos do autómato. Se estiver presente, o programa é encerrado com uma mensagem de erro indicando que o autómato não é um AFD (Autómato Finito Determinista).



**Verificação de Transições Múltiplas:**

Por fim, a função verifica se há mais de uma transição para o mesmo símbolo em alguma transição do autómato. Se essa condição for verificada, o programa é encerrado com uma mensagem de erro indicando que o autómato é um AFND (Autómato Finito Não-Determinista).



### Gerar Representação Gráfica do Grafo

O método gerar\_grafo(automato) utiliza a biblioteca Graphviz para gerar a representação gráfica do grafo associado ao AFD. Este método percorre os estados e as transições definidas no AFD e utiliza os métodos da biblioteca Graphviz para adicionar os nós e as arestas correspondentes ao grafo.



### Reconhecimento de Palavras

O método reconhecer\_palavra(automato, palavra) é responsável por reconhecer se uma palavra pertence à linguagem definida pelo AFD. Este método percorre a palavra, verificando se existe uma transição definida para cada símbolo da palavra. Se a palavra for reconhecida, o método retorna verdadeiro juntamente com o caminho percorrido. Caso contrário, retorna falso e indica a situação de erro encontrada.



### Utilização do Código

Para utilizar o código, pode-se fornecer o caminho para um ficheiro JSON contendo a definição do AFD e opcionalmente especificar as operações desejadas, como a geração da representação gráfica do grafo ou o reconhecimento de uma palavra.

### Exemplos de utilização:



Este comando irá gerar a representação gráfica do grafo associado ao AFD definido no ficheiro automato.json.



Este comando irá reconhecer a palavra "101" utilizando o AFD definido no ficheiro automato.json.

## Conclusão

A implementação apresentada neste capítulo oferece uma forma simples e eficiente de trabalhar com Autómatos Finitos Deterministas, permitindo a definição, visualização e reconhecimento de linguagens através de AFDs.

# Expressão Regular para AFND – Exercício B

## Introdução

Este capítulo apresenta uma implementação em Python de um Algoritmo de Conversão de Expressões Regulares para Autómatos Finitos Não-Determinísticos (AFND). O código apresenta funcionalidades para converter uma expressão regular num AFND, permitindo representar linguagens regulares de forma não determinística.

## Estrutura do Código

O código está estruturado em três partes principais: a definição das operações básicas nos AFNDs, a conversão da expressão regular num AFND e a execução do programa principal.

### Operações Básicas nos AFNDs

As operações básicas nos AFNDs incluem a concatenação, a união e o fecho de Kleene. Cada operação é implementada como uma função que recebe dois AFNDs, um como entrada e retorna um novo AFND como saída.

### Concatenação:

A função concatenação(afnd1, afnd2) recebe dois AFNDs como entrada e concatena-os, adicionando transições vazias do estado final de afnd1 para o estado inicial de afnd2.



### União:

A função união(afnd1, afnd2) recebe dois AFNDs como entrada e realiza a união entre eles, criando um novo estado inicial e definindo transições vazias deste estado para os estados iniciais de afnd1 e afnd2.



### Fecho de Kleene:

A função kleene(afnd) recebe um AFND como entrada e aplica o fecho de Kleene criando um novo estado inicial e um novo estado final, e definindo transições vazias entre estes estados e os estados originais do AFND.



### Transição Vazia:

Esta função transição\_vazia(afnd) adiciona uma transição vazia ao AFND, criando dois estados e atualizando a estrutura do AFND para incluir essa transição. Isso permite que o AFND represente transições entre estados sem utilizar nenhum símbolo de entrada.  
  


operadores: Este é o nome do dicionário que é usado para mapear as operações.

"seq": Esta chave está associada à operação de concatenação. Quando o código encontrar a operação "seq" numa expressão regular, ele chama a função concatenacao.

"alt": Esta chave está associada à operação de união. Quando o código encontrar a operação "alt" numa expressão regular, ele chama a função uniao.

"kle": Esta chave está associada à operação de fecho de Kleene. Quando o código encontrar a operação "kle" numa expressão regular, ele chama a função kleene.  
  
“seq”: Esta chave está associada a uma transição vazia.



### Conversão da Expressão Regular num AFND

A função converter\_afnd(expressao\_regular) recebe uma expressão regular como entrada e converte-a num AFND utilizando as operações básicas nos AFNDs.



### Utilização do Código

Para utilizar o código, deve-se fornecer o caminho para um ficheiro JSON que contém a expressão regular. O programa converterá a expressão regular num AFND e imprimirá o AFND resultante no terminal.

### Exemplo de Utilização:



## Conclusão

A implementação apresentada neste capítulo oferece uma forma simples e eficiente de converter expressões regulares em Autómatos Finitos Não-Determinísticos, permitindo a representação e manipulação de linguagens regulares de forma não determinística.

# Conversão de AFND para AFD – Exercício C

## Introdução

Este script Python é uma ferramenta projetada para converter autómatos finitos não-determinísticos (AFND) em autómatos finitos determinísticos (AFD). O processo de conversão é essencial para simplificar a análise e o reconhecimento de linguagens regulares, permitindo a implementação de reconhecedores mais eficientes. A conversão é realizada através da construção de conjuntos de estados alcançáveis e aplicação do fecho-ε.

## Estrutura do Código

O código é dividido em várias funções principais que tratam da leitura, a conversão e a gravação de autómatos:

**carrega\_automato(caminho)**: Lê um AFND de um ficheiro JSON.

Uma imagem com texto, captura de ecrã, Tipo de letra

Descrição gerada automaticamente

**AFNDtoAFD(automatoAFND)**: Converte o AFND lido em AFD.

A função **AFNDtoAFD** é responsável por converter um autómato finito não-determinístico (AFND) num autómato finito determinístico (AFD). Ela utiliza conceitos de teoria de autómatos como o fecho-ε e a construção de subconjuntos.

### Subfunções Utilizadas:

**-encontrar\_conjunto(estadoAtual, simbolo, transicoes)**: Esta função interna é usada para determinar todos os estados que podem ser alcançados a partir de um conjunto de estados atual dado um símbolo específico. Utiliza um conjunto (set) para garantir que os estados não sejam repetidos e retorna um frozenset imutável dos estados alcançáveis.

Uma imagem com texto, captura de ecrã, software, Tipo de letra

Descrição gerada automaticamente

**- fecho\_epsilon(conjunto, transicoes)**: Calcula o fecho-ε de um conjunto de estados, que são todos os estados alcançáveis a partir do conjunto dado, considerando apenas transições ε (epsilon). A função usa uma stack para explorar todos os estados alcançáveis e adiciona novos estados ao fecho à medida que são encontrados.

Uma imagem com texto, captura de ecrã, Tipo de letra

Descrição gerada automaticamente

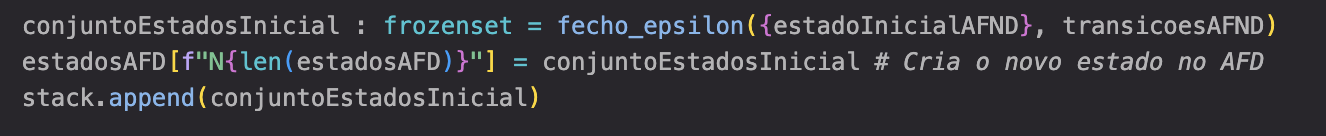
### Processo de Conversão:

**- Inicializações**: A função começa por configurar as variáveis iniciais, como os estados e transições do AFND, além do estado inicial e os estados finais.

Uma imagem com texto, captura de ecrã, Tipo de letra

Descrição gerada automaticamente

**- Construção dos Estados do AFD**: A determinização inicia com o fecho-ε do estado inicial. Este conjunto de estados representa o primeiro estado do AFD. A função usa uma stack para verificar os conjuntos de estados que ainda precisam ser processados.



**- construir\_AFD(listaEstadosAtuais)**

No loop principal, a função processa cada conjunto de estados na stack. Para cada símbolo do alfabeto, exceto ε, determina-se os estados alcançáveis e aplica-se o fecho-ε. Novos conjuntos encontrados são colocados na stack para análise subsequente.

Uma imagem com texto, captura de ecrã, software, Tipo de letra

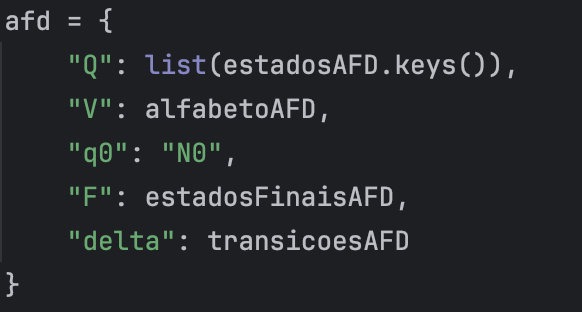
Descrição gerada automaticamente

**- Identificação dos Estados Finais do AFD**: Qualquer novo estado do AFD que contenha pelo menos um estado final do AFND é considerado um estado final do AFD.

Uma imagem com texto, captura de ecrã, Tipo de letra, file

Descrição gerada automaticamente

**- Construção do Objeto AFD**: O AFD é então construído como um dicionário contendo seus estados, alfabeto, estado inicial, estados finais, e transições.



**- Retorno**: A função retorna o autómato finito determinístico(AFD) construído.

**- gravar\_automato(afd, caminho):** Grava o AFD convertido num ficheiro JSON.

Uma imagem com texto, captura de ecrã, Tipo de letra

Descrição gerada automaticamente

### Utilização do Código

Para utilizar o script, deve-se fornecer o caminho para um ficheiro JSON que contém a definição AFND e o nome para o output do script.



### Exemplo de Utilização:



## Conclusão:

Esta implementação fornece um método prático e eficiente para a conversão de Autómatos Finitos Não-Determinísticos em Determinísticos.