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 3](#_Toc163754800)

[Introdução 3](#_Toc163754801)

[Estrutura do Código 3](#_Toc163754802)

[Leitura da Definição do AFD 3](#_Toc163754803)

[Validações do Autómato 4](#_Toc163754804)

[Gerar Representação Gráfica do Grafo 6](#_Toc163754805)

[Reconhecimento de Palavras 7](#_Toc163754806)

[Utilização do Código 8](#_Toc163754807)

[Exemplos de utilização: 8](#_Toc163754808)

[Conclusão 8](#_Toc163754809)

[Expressão Regular para AFND – Exercício B 9](#_Toc163754810)

[Introdução 9](#_Toc163754811)

[Estrutura do Código 9](#_Toc163754812)

[Operações Básicas nos AFNDs 9](#_Toc163754813)

[Concatenação: 9](#_Toc163754814)

[União: 10](#_Toc163754815)

[Fecho de Kleene: 11](#_Toc163754816)

[Conversão da Expressão Regular num AFND 13](#_Toc163754817)

[Utilização do Código 14](#_Toc163754818)

[Exemplo de Utilização: 14](#_Toc163754819)

[Conclusão 14](#_Toc163754820)

[Conversão de AFND para AFD – Exercício C 15](#_Toc163754821)

# 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.



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.



### 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

## Estrutura do Código