Universidade Federal Do Maranhão - UFMA
Curso de Engenharia da Computação

Conversão de expressões regulares para autômato finito não determinístico

São Luís-MA
2025

Conversão de expressões regulares para autômato finito não determinístico

Keven Gustavo Dos Santos Gomes
Lucas Emanoel Amaral Gomes

Resumo:

O Autômato Finito Não Determinístico (AFN) é um modelo computacional utilizado na teoria da computação e na linguística para reconhecer linguagens regulares. Diferente do Autômato Finito Determinístico (AFD), o AFN permite que, a partir de um estado e um símbolo de entrada, haja múltiplas transições possíveis, incluindo transições sem consumo de símbolos (ε-transições). Essa característica introduz não determinismo, o que significa que o autômato pode estar em vários estados simultaneamente durante o processamento de uma cadeia de caracteres.

As expressões regulares (regex) são ferramentas usadas para identificar padrões em cadeias de caracteres. Elas combinam caracteres e metacaracteres para construir padrões de busca. Caracteres são elementos fundamentais que podem ser literais ou especiais, enquanto metacaracteres, como o ponto (.), o dólar ($) e o ponto de interrogação (?), conferem flexibilidade e permitem a criação de padrões mais complexos.

Esse trabalho aborda a transformação de expressões regulares em autômatos finitos determinísticos, explorando suas bases teóricas e aplicações prática

Palavras-chave: Autômato Finito Determinístico (AFD), Autômato Finito Não Determinístico (AFN), Expressões Regulares (regex)

1 INTRODUÇÃO

Um Autômato Finito Não Determinístico (AFN) é um modelo matemático utilizado para representar sistemas que possuem um conjunto finito de estados e processam uma sequência de entradas, determinando se a sequência é aceita ou rejeitada. Diferente do Autômato Finito Determinístico (AFD), o AFN permite que, a partir de um estado e um símbolo de entrada, haja múltiplas transições possíveis, incluindo transições sem consumo de símbolos (ε-transições). Ele é constituído por 3 tipos de entidades que são conjuntos de entidades, alfabeto e um conjunto de transições.

Dos estados, destacamos um como estado inicial. Destacamos também um conjunto de estados como estados finais. O conjunto de transições é definido através de uma função que pode mapear para múltiplos estados ou incluir transições ε.

Um AFN é definido pela tupla:

E: Conjunto não vazio de estados.

Σ: Conjunto que define o alfabeto.

δ: $\Sigma \times E \rightarrow P(E)$: Uma função de transição, onde P(E) representa o conjunto de partes de E (ou seja, todos os subconjuntos possíveis de E).

i: Estado inicial.

F: Um subconjunto de E, que representa o conjunto de estados finais.

A função de transição δ pode levar a múltiplos estados a partir de um único estado e símbolo de entrada, ou permitir transições sem consumir símbolos (ε-transições). A aceitação de uma cadeia ocorre se existe pelo menos um caminho de transições que leva do estado inicial a um estado final após a leitura completa da cadeia. Essa característica de não determinismo torna o AFN uma ferramenta flexível e poderosa para modelar problemas de reconhecimento de padrões.

Uma expressão regular ou também conhecido como regex, são usados para identificar padrões em uma *string.* Um regex é composto por caracteres ou metacaracteres

- Caracteres: são os elementos fundamentais que compõem os padrões de busca. Eles podem ser tanto literais (exatamente como você os digita) quanto especiais, com significados próprios dentro da sintaxe da regex.
- Metacaracteres: são caracteres com significado especiais. Eles permitem criar padrões mais complexos e flexíveis. Alguns exemplos mais comuns incluem:

  Ponto(.): Corresponde a qualquer caractere, exceto uma nova linha

  Dólar($): Indica o final da linha

  Ponto de interrogação(?): Indica zero ou mais ocorrências de caractere anterior

2 METODOLOGIA

Para ser realizada a conversão de expressões regulares(Regex) em um Autómatos Finitos Não Determinístico(AFN) foi usado a linguagem python, onde o código pega o input do usuário como entrada e analisa essa string regex quebrando-a em vários caracteres que então, dependendo do símbolo, é executada uma ação relacionada a ele, nas etapas a seguir é feito uma análise mais aprofundada da lógica do código, começando com suas operações:

2.1 Operações

Durante a transformação de uma Regex em um AFN, quando o caractere atual é diferente de um símbolo(caractere alfanumérico pertencente ao alfabeto), o código vai executar uma das seguintes operações fundamentais, que foram baseada na teoria de linguagens formais:

- **Concatenação:** combinação sequencial de dois autômatos.
- **União:** combinação alternativa entre dois autômatos.
- **Fecho de Kleene:** repetição de um padrão zero ou mais vezes.

Para cada operação, foram definidos algoritmos específicos que possibilitam a manipulação de estados e transições, respeitando as definições formais de AFN. Além disso, uma estrutura de dados em Python foi escolhida para representar as transições do autômato como um dicionário aninhado.

2.2 Estruturação do Algoritmo Geral

O algoritmo foi projetado para processar a expressão regular em etapas claras, organizadas da seguinte forma:

- **Definição da Classe AFN:**

  Foi criada uma classe Python denominada **AFN** com os seguintes atributos:

  - **estado_inicial**: Identifica o estado inicial do autômato.
  - **estado_final**: Indica o estado final do autômato.
  - **transicao**: Representa as transições como um dicionário no formato **{estado: {símbolo: {destinos}}}.**

- **Tokenização da Expressão Regular:**

  A expressão regular é processada caractere por caractere, diferenciando símbolos, operadores (“*****”, “|”, “**.**”) e delimitadores (“**(**“, “**)**”).

- **Pilhas para Operandos e Operadores:**

  Foram utilizadas duas pilhas:

  - **Operandos:** Armazena autômatos parciais que serão combinados.
  - **Operadores:** Gerencia os operadores pendentes de aplicação, garantindo a precedência correta com base na prioridade definida (* > . > |).

- **Aplicação de Operadores:**

  Para cada operador encontrado:

  - **Concatenação (“.”):** Conecta o estado final de um AFN ao estado inicial do próximo.
  - **União (“|”):** Cria um novo estado inicial com transições ε para os estados iniciais dos dois AFNs, além de um novo estado final com transições ε dos estados finais dos dois AFNs.
  - **Fecho de Kleene (“*”):** Adiciona transições ε do estado inicial ao estado final e vice-versa, criando um loop.

2.3 Criação de Funções Auxiliares

Funções auxiliares foram implementadas para modularizar o código e facilitar a manutenção:

- **criar_afn_para_simbolo(simbolo)**: Gera um AFN básico para um símbolo da expressão regular.
- **concatenar_afn(afn1, afn2)**: Implementa a operação de concatenação entre dois AFNs.
- **uniao_afn(afn1, afn2)**: Cria um AFN que representa a união de dois autômatos.

- **asterisco_afn(afn):** Realiza a operação de fecho de Kleene sobre um AFN.
- **adicionar_transicao(transicoes, origem, simbolo, destinos)**: Adiciona transições ao dicionário do AFN, evitando duplicatas.

Essas funções garantem que o algoritmo seja escalável e reutilizável para diferentes partes da expressão regular.

2.4 Fluxo de Execução do Algoritmo

O fluxo do algoritmo segue as etapas abaixo:

1. **Iteração sobre a Expressão Regular:**
    - Para cada símbolo, cria-se um AFN básico ou aplica-se um operador de acordo com sua precedência.
2. **Gerenciamento de Precedência:**
    - Antes de empilhar um operador, verifica-se se operadores de maior ou igual precedência precisam ser aplicados.
3. **Resolução de Parênteses:**
    - Operadores contidos entre parênteses são aplicados primeiro, respeitando a ordem das operações.
4. **Processamento Final:**
    - Após a iteração, os operadores remanescentes na pilha são aplicados aos operandos, resultando no AFN final.

3 RESULTADOS E DISCUSSÃO

A seguir, são apresentados os resultados obtidos ao executar o código de transformação de expressões regulares em Autómatos Finitos Não Determinísticos (AFN), a Regex é dada pelo próprio usuário como entrada através de um input e a saída é composta pelo estado inicial, estado final, e a função de transição.

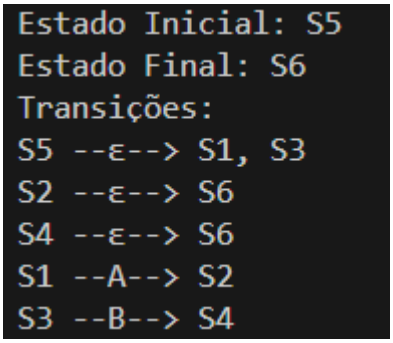
3.1 Exemplos de Transformação de Regex em AFN

Exemplo 1: Expressão Regular: **A|B**

- **Descrição:**
  A expressão **A|B** representa a união de dois símbolos: **A** e **B**.

- **AFN Gerado:**

Figura 1: AFN A|B



Fonte: Elaborado pelos autores

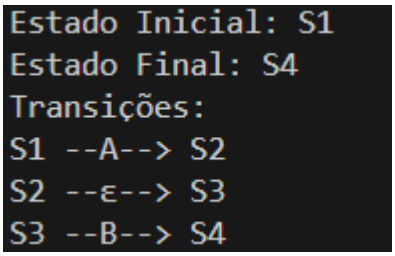
  - **Interpretação:**
    O estado inicial (**S5**) possui transições ε para os estados iniciais dos dois ramos (**S1** e **S3**), que possuem os símbolos **A** e **B** respectivamente, e então cada ramo possui transições específicas para o estado final (**S6**), passando por **S2** ou **S4**.

Exemplo 2: Expressão Regular**: AB**

- **Descrição:**
  A expressão ab representa a concatenação dos símbolos **A** e **B**.

- **AFN Gerado:**

Figura 2 - AFN AB



Fonte: Elaborado pelos autores

- **Interpretação:**
  O estado inicial (**S1**) possui uma transição direta para o próximo estado (**S2**) ao receber **A**, seguido por uma transição de **S2** para **S3**, que possui o símbolo **B,** que então se conecta ao estado final(**S4**).

Exemplo 3: Expressão Regular **A***

- **Descrição:**
  A expressão **A*** aplica o fecho de Kleene ao símbolo a, permitindo zero ou mais ocorrências.

- **AFN Gerado:**
  Figura 3: AFN A*

```
Estado Inicial: S3
Estado Final: S4
Transições:
S3 --ε--> S1, S4
S2 --ε--> S1, S4
S1 --A--> S2
```

Fonte: Elaborado pelos autores

- **Interpretação:**
  O estado inicial (**S3**) possui transições **ε** para o estado final (**S4**) e para o estado de transição de **A**(**S1**), esse estado possui transição para o estado intermediário(**S2**), que possui transições para o estado de **A** ou o estado final(**S4**).

Exemplo 4: Expressão Regular complexa **(A|B)*C**

- **Descrição:**
  A expressão **(A|B)*C**, possui todas as operações anteriores com o adicional dos parênteses formando um grupo com a união

- **AFN Gerado:**
  Figura 4: AFN (A|B).*C

```
Estado Inicial: S7
Estado Final: S10
Transições:
S7 --ε--> S8, S5
S6 --ε--> S8, S5
S5 --ε--> S3, S1
S2 --ε--> S6
S4 --ε--> S6
S1 --A--> S2
S3 --B--> S4
S8 --ε--> S9
S9 --C--> S10
```

Fonte: Elaborado pelos autores

- **Interpretação:**
  O estado inicial (**S7**) possui transições **ε** para o estado intermediário (**S8**), que permite uma transição para o estado **S9** que possui o símbolo **C**, e também para o estado intermediário(**S5**), que permite a transição para **A**(**S1**) ou **B**(**S2**), que então por meio do estado **S6**, esses símbolos podem ser repetidos indefinidamente, ou então ir para estado **S8** que é ligado ao **S9** que então pode transicionar para o estado final(**S10**).

3.2 Visualização dos Resultados

Para melhor compreensão e análise, um dos exemplos foi formatado em uma tabela:

Tabela de Transições - Exemplo **A|B**:

Estado Origem	Símbolo	Estado(s) de Destino(s)
S5	ε	S1,S3
S1	A	S2
S2	ε	S6
S3	B	S4
S4	ε	S6

3.3 Limitações Observadas

Apesar da eficiência da utilização de pilhas para operandos e operadores, foi observado algumas limitações no código:

- **Casos de Ambiguidade:** Embora a implementação atual trate operadores básicos, expressões complexas com múltiplos níveis de parênteses podem dificultar a visualização.
- **Escalabilidade:** A geração de estados adicionais para cada operação torna o modelo menos eficiente para expressões muito grandes.

3.4 Validação dos Resultados

Os autômatos gerados foram validados comparando as transições com os resultados teóricos previstos para cada exemplo, todos os casos analisados produziram autômatos corretos e consistentes com a definição formal de AFNs, esses resultados demonstram a eficácia do algoritmo na conversão de expressões regulares em autômatos finitos não determinísticos, fornecendo uma base sólida para aplicações em linguagens formais e processamento de padrões.

4 CONSIDERAÇÕES FINAIS

Este trabalho demonstrou o desenvolvimento de um algoritmo para transformar expressões regulares em Autômatos Finitos Não Determinísticos (AFN), com implementação em Python, a abordagem proposta foi eficaz na criação de autômatos que seguem as definições formais da teoria de linguagens e autômatos, sendo capazes de representar os operadores fundamentais de expressões regulares, como concatenação, união e fecho de Kleene.

Em resumo, o trabalho realizado demonstra que é possível transformar expressões regulares em AFNs de forma eficiente, fornecendo uma base sólida para estudos sobre linguagens formais e processamento de padrões,o código desenvolvido pode servir como ferramenta educacional para o ensino de autômatos e expressões regulares, além de ser um ponto de partida para o desenvolvimento de sistemas mais complexos, como uma transformação de AFN para Autômatos Finitos Determinísticos(AFD).

Por fim, os resultados alcançados destacam a importância de integrar conceitos teóricos e práticos na computação, promovendo o aprendizado e a aplicação direta de fundamentos matemáticos em soluções computacionais.

REFERÊNCIAS

LIRA, M. Autômatos Finitos em Python. Disponível em: <https://setanta.wordpress.com/2007/06/09/automatos-finitos-em-python/>. Acesso em: 10 jan. 2025.

DARKGEEKMS. GitHub - DarkGeekMS/regex-to-nfa: A python tool for Regex conversion to nondeterministic finite automaton (NFA). Disponível em: <https://github.com/DarkGeekMS/regex-to-nfa/tree/main>.

COMPUTANDO! Autômatos #01: Algoritmo 01 - Expressão Regular para Autômato Finito Não-Determinístico Teoria. Disponível em: <https://www.youtube.com/watch?v=zwcdwEQEayM>. Acesso em: 14 jan. 2025.

DAVI ROMERO DE VASCONCELOS. Linguagens Regulares: Convertendo Expressão Regular em Autômato Finito Não-Determinístico. Disponível em: <https://www.youtube.com/watch?v=JWttym_qNI0>. Acesso em: 14 jan. 2025.

Regular expression to ∈-NFA. Disponível em: <https://www.geeksforgeeks.org/regular-expression-to-nfa/>.