Implementação de um Gerador de Autômatos Finitos Determinísticos a partir de Gramáticas Regulares e Tokens

Erickson G. Müller
Universidade Federal da Fronteira Sul - UFFS
erickson.muller@estudante.uffs.edu.br

11 de dezembro de 2025

1 Resumo

Este trabalho apresenta o desenvolvimento de uma ferramenta computacional para a geração de Autômatos Finitos Determinísticos a partir de tokens e gramáticas regulares. O sistema, implementado na linguagem Python, processa um arquivo de entrada contendo definições de tokens e gramáticas regulares em notação BNF. A metodologia adotada consistiu em duas etapas principais: primeiramente, a construção de um Autômato Finito Não-Determinístico através da conversão direta das regras gramaticais e léxicas, garantindo a criação de novos estados para cada transição de tokens e, secundariamente, a conversão deste para um AFD utilizando o algoritmo de construção de subconjuntos. Por fim, o autômato resultante foi totalizado com a inclusão de um estado de erro, garantindo que o reconhecedor seja completo. O projeto demonstra a aplicação prática de conceitos fundamentais da Teoria da Computação na automação de analisadores léxicos.

2 Introdução

A manipulação de autômatos finitos envolve desafios algorítmicos clássicos, notadamente a conversão de modelos não-determinísticos para modelos determinísticos. Enquanto um Autômato Finito Não-Determinístico permite múltiplas transições para um mesmo símbolo a partir de um único estado, sua implementação computacional direta é menos eficiente do que a de um Autômato Finito Determinístico. Portanto, a conversão entre esses modelos é um passo necessário para a construção de reconhecedores de linguagens performáticos.

O presente projeto visa a implementação de um software que realiza o ciclo completo de geração de um reconhecedor para linguagens regulares. A solução foi projetada para processar um arquivo de entrada ("fonte.txt") contendo tokens simples e regras em notação BNF, convertendo-os inicialmente para um AFND. A partir desta estrutura, aplica-se o algoritmo de construção de subconjuntos para obter um AFD equivalente.

Diferente de abordagens que focam na minimização de estados via classes de equivalência, este trabalho prioriza a completude do autômato. Dessa forma, uma etapa final de processamento é dedicada à inserção de estados de erro para a totalização, garantindo que a máquina de estados resultante possua transições definidas para todo o alfabeto da linguagem em todos os seus estados, conforme especificado nos requisitos do projeto.

3 Referencial Teórico

Esta seção apresenta os fundamentos matemáticos e computacionais necessários para a compreensão do projeto, abrangendo desde a definição de linguagens regulares até os algoritmos de transformação de autômatos.

3.1 Linguagens Regulares e Gramáticas

As Linguagens Regulares constituem a classe mais simples na Hierarquia de Chomsky. Elas podem ser descritas por Expressões Regulares ou geradas por Gramáticas Regulares. Uma gramática é definida formalmente por uma quádrupla $G = (V, \Sigma, P, S)$ [1][2], onde V é um conjunto finito de variáveis (não-terminais), Σ é o alfabeto de terminais, P são as regras de produção e S é o símbolo inicial. Neste trabalho, utiliza-se a notação BNF para representar tais gramáticas.

3.2 Autômatos Finitos

Um Autômato Finito é um modelo matemático de computação utilizado para reconhecer linguagens regulares. Ele pode ser classificado em:

- **Não-Determinístico (AFND):** Permite que, para um dado estado e símbolo de entrada, existam zero, uma ou múltiplas transições possíveis.
- **Determinístico (AFD):** Para cada par (estado, símbolo), existe exatamente uma transição definida.

3.3 Determinização

A equivalência entre AFNDs e AFDs é comprovada pelo teorema que afirma que toda linguagem aceita por um AFND também é aceita por um AFD. O algoritmo utilizado para realizar essa conversão é conhecido como *Subset Construction* (Construção de Subconjuntos). O método consiste em criar novos

estados no AFD correspondentes a conjuntos de estados do AFND original, eliminando a ambiguidade das transições.

4 Especificação e Implementação da Solução para Gerar AFD

A implementação da solução abordou desafios específicos da manipulação de autômatos, notadamente a gestão de nomes de estados e a resolução de não-determinismo. Abaixo, detalham-se as estratégias algorítmicas adotadas.

4.1 Geração Dinâmica de Estados

Quando um token ou gramática está sendo processado, respectivamente nas funções *processarToken()* e *processarGramatica()*, é chamada a função *obterProxNome()* que vai usar o número de estados no autômato para criar o nome do próximo estado, que segue uma ordem alfabética. A função que faz essa conversão é chamada *gerarNomeEstado()*.

Para essa geração de nomes de estados, foi adotada a nomenclatura sequencial no padrão A−Z → AA−AZ..., conforme a interpretação dada pela solução do autômato indicada na descrição do projeto. Diferente de abordagens que utilizam identificadores numéricos, esta solução mantém um contador global e uma tabela de símbolos para mapear não-terminais das gramáticas para seus respectivos nomes no autômato gerado. Isso assegura a consistência entre referências futuras e passadas durante a leitura do arquivo.

4.2 Algoritmo de Determinização

A conversão de AFND para AFD foi realizada através do algoritmo de construção de subconjuntos. O processo inicia com o estado {`S`} e, iterativamente, calcula a união das transições possíveis para cada símbolo do alfabeto.

- **Gerenciamento de Estados Compostos:** Cada novo conjunto de estados descoberto é convertido em uma chave única e ordenadas alfabeticamente para evitar duplicidade, ligados por uma linha. No formato "A_B".
- **Propagação de Estados Finais:** Um estado composto no AFD é marcado como final se, e somente se, contiver pelo menos um estado final do AFND original.

A função *determinizar()* cria uma fila de estados a serem analisados. Todos os estados que são destinados a mais de um estado com o mesmo terminal, resultará na criação de um novo estado na variável *nome_estado*. Após, verifica-se se esse estado é final verificando se os estados do conjunto são finais. Em seguida, os destinos do novo estado são criados e esse estado é adicionado à fila para ser processado.

4.3 Tratamento de Erros e Totalização

Após a determinização, o autômato passa por uma verificação de completude. O algoritmo percorre todos os estados e o alfabeto da linguagem. Para qualquer estado sem transição definida, cria-se uma transição para um novo estado de erro (nomeado como "_" para facilitar a visualização das transições no terminal). Esse estado possui transições reflexivas para todo o alfabeto.

5 Conclusão

O desenvolvimento deste trabalho permitiu consolidar os conhecimentos teóricos adquiridos na disciplina de Linguagens Formais e Autômatos, demonstrando a aplicabilidade prática de conceitos abstratos como estados, transições e determinismo. A solução mostrou-se eficaz no processamento de arquivos de entrada mistos, realizando corretamente o *parsing* de regras BNF e a tokenização de palavras reservadas.

Foi possível observar, durante a implementação, a complexidade inerente ao tratamento do não-determinismo. A estratégia de gerar novos estados sequenciais para cada token revelou-se eficaz para isolar conflitos iniciais, delegando a resolução da ambiguidade ao algoritmo de construção de subconjuntos. Embora a etapa de minimização não tenha sido aplicada no escopo final, a implementação da totalização com o Estado de Erro garantiu que o AFD gerado seja robusto, rejeitando explicitamente cadeias que não pertencem à linguagem especificada.

A escolha de utilizar dicionários aninhados e conjuntos do tipo `frozenset` facilitou a manipulação algébrica necessária para o algoritmo de determinização. Um desafio técnico superado foi a padronização da nomenclatura dos estados, o que exigiu um mapeamento dinâmico entre os não-terminais da gramática e os estados do autômato. Os testes realizados confirmaram que o sistema gera tabelas de transição coerentes, onde estados compostos resolvem corretamente as ambiguidades presentes no AFND original.

Conclui-se que o software desenvolvido cumpriu os requisitos funcionais estabelecidos para a geração de AFDs. O sistema é capaz de carregar corretamente as definições de linguagem a partir de um arquivo texto, respeitando a regra de compartilhamento único do estado inicial (S) e a criação de novos estados para transições subsequentes. O autômato final apresentado pelo programa é, portanto, um Autômato Finito Determinístico e Total.

Referencial Bibliográfico

[1] SCHEFFEL, Roberto M. *Apostila Linguagens Formais e Autômatos.*

[2] HOPCROFT, J. E. and ULLMAN, J. D.: *Introduction to Automata Theory, Languages and Computation.* Addison-Wesley, 1979