Autômatos Finitos Determinísticos

Guilherme Cardoso Silva Renata Caroline Cunha

[1]Linguagens Formais e Autômatos – Instituto Federal de Minas Gerais (IFMG)
Caixa Postal 35570-000 – Formiga – MG – Brasil

1. Introdução

Este Trabalho Prático aborda a implementação dos algoritmos para manipulação de Autômatos Finitos Determinísticos (AFDs):

- Minimização de Autômatos;
- Equivalência de Autômatos;
- União de Autômatos;
- Interseção de Autômatos;
- Complemento de Autômatos;
- Diferença de Autômatos;
- Lista de Pares Equivalentes;
- Aceita Palavra;
- Anda no Autômato;
- Verifica se Estado é Final;
- Lê Autômato;
- Salva Autômato;
- Entre outros para manipulação de AFDs;

2. Compilação

Existem duas formas para compilar o projeto, sendo elas:

- Importar o projeto Java utilizando a ferramenta Eclipse;
- Executar através do terminal a classe Principal. *$ java Principal*

Para testar os algoritmos implementados é necessário fazer alterações na classe Principal. Caso a execução seja feita no terminal é necessário recompilar esta classe *$ javac Principal.java*.

3. Implementação

Foi utilizado a linguagem Java na IDE Eclipse Neon 3.0 para o desenvolvimento deste trabalho.

A execução de todos os algoritmos consideram que como entrada receberão um AFD completo e todos autômatos devem ter o mesmo alfabeto para serem utilizados nos métodos que realizam operações entre dois AFDs.

Os métodos Load e Save foram implementados seguindo o padrão de arquivo da ferramenta JFlap 7.0.

O Projeto foi dividido em várias classes para modularizar o código e facilitar a manutenção.

Junto do relatório está anexado um diagrama com todas as Classes implementadas no projeto.

3.1. Classe AFD

Esta é a classe para representar um AFD. Nela são armazenados uma lista de estados, uma lista contendo cada letra do alfabeto, uma lista de transições, o estado inicial e uma lista contendo os estados finais.

Todos os métodos que realizam operações com AFDs foram implementados nesta classe.

3.2. Classe ENodoVerificacao

Classe Enum que contém constantes para os nodos das matrizes de equivalência e minimização.

Definindo se estes já foram verificados, se não são equivalentes ou são equivalentes.

3.3. Classe NodoEquivalenciaAFD

Utilizada para representar a estrutura de um nodo que é utilizado para construir uma matriz para equivalência de autômatos. Ela contém métodos para armazenar o par de estados com origem do AFD a ser analisado nesta posição, uma propriedade Enum que mostra se o nodo não é equivalente ou não foi verificado e uma lista de pares pendentes para definir se esta posição é ou não equivalente.

3.4. Classe NodoMinimizacao

Utilizada para representar a estrutura de um nodo que é utilizado para construir uma matriz para minimização. Ela contém métodos para armazenar o par de estados a ser analisado nesta posição, uma propriedade Enum que mostra se o nodo não é equivalente ou não foi verificado e uma lista de pares pendentes para definir se esta posição é ou não equivalente.

3.5. Classe ParOrdenado

Utilizada para representar a estrutura de um par ordenado. Ela contém métodos para armazenar um estado x e um estado y.

Esta Classe é necessária para o método de lista de estados equivalentes ou minimização.

3.6. Classe ParOrdenadoOrigem

Utilizada para representar a estrutura de um par ordenado com AFD de origem. Ela contém métodos para armazenar um estado x e um estado y com seus respectivos AFD de origem.

Esta Classe é necessária para o método de verificação de autômatos equivalentes, onde cada índice do par pode pertencer a autômatos distintos.

3.7. Classe State

Utilizada para representar a estrutura de um estado. Ela contém métodos para armazenar o id, o nome, se é um estado inicial e final.

3.8. Classe Transition

Utilizada para representar a estrutura de transição. Ela contém métodos para armazenar a origem, o destino e a letra consumida.

4. Validação

Foram construídos vários AFDs utilizando a ferramenta JFlap 7.0, que foram utilizados para uma bateria de testes.

Para validar os resultados encontrados, estes foram verificados no JFlap e manualmente, comparando os resultados.

Todos os testes utilizados estão anexados junto ao código fonte.

5. Conclusão

Este trabalho resultou em uma grande experiência e conhecimento com relação as formas de implementar um AFD. Além disto, pode-se exercitar com um problema prático a implementação dos algoritmos trabalhados em sala de aula.