

Relatório Trabalho 01 de L.F.A

Samuel Terra¹ Matheus Calixto²

¹Instituto Federal de Ciência e Tecnologia de Minas Gerais
São Luiz Gonzaga, s/n - Formiga / MG - Brasil

calixtinn@gmail.com, samuelterra22@gmail.com

Resumo. *Este é um trabalho da disciplina de Linguagens Formais e Autômatos Finitos, que aborda a implementação de algoritmos para a manipulação de autômatos finitos.*

Introdução

Dentre os problemas propostos para serem resolvidos neste trabalho prático, ambos foram concluídos. O primeiro problema foi criar uma classe para representar os Autômatos Finitos Determinísticos, e o segundo problema foi implementar as funções que continham os algoritmos para a manipulação deste AFD.

O programa foi construído na linguagem Python 3, por proporcionar facilidade de implementação, diversas funções de manipulação de objetos, interface gráfica e estruturas de dados.

A estrutura do programa foi construída em cima do padrão MVC (*Model, View and Controller*) de orientação a Objetos. O MVC é um padrão de arquitetura de software onde realiza a separação da aplicação em três camadas. Com a camada *Model* é possível elaborar a modelagem dos objetos mais simples no sistema (ex. Automato, Estado, Transição). Já na camada *Controller*, é onde fica todas as regras de negócio, os métodos que realmente realizam todo o esforço com a implementação de todas as funcionalidades. E camada que é chamada de *View* é possível realizar a interação com o usuário, nela apenas é solicitado as informações de entrada e passadas para o *Controller* que é também instanciado.

A interação com o usuário é intuitiva e realizada através do terminal com menus e sub-menus contendo todas as funcionalidades exigidas na especificação do trabalho.

Implementação

A implementação do trabalho foi realizada pelos dois alunos de maneira online, que utilizaram recursos como: IDE *PyCharm* e controle de versão com *Git*. A divisão das tarefas foi realizada de maneira igual e justa entre os integrantes, o que contribuiu de maneira excelente para o bom andamento do trabalho. As dificuldades foram solucionadas rapidamente através da troca de ideias, e as decisões de implementação foram discutidas de maneira saudável.

A Classe AFD

Do que diz respeito ao código, o objeto AFD foi construído a partir de outros objetos: Estados (*States*) e Transições (*Transitions*). O objeto *State*, que representa um estado de um autômato, possui os seguintes atributos:

- **ID:** Um número inteiro salvo como carácter, que é a identificação do estado.

- **Name:** O nome do estado.
- **PosX:** Um número real, representando a coordenada do eixo X referente à posição do estado no plano cartesiano do software JFLAP.
- **PosY:** Um número real, representando a coordenada do eixo Y referente à posição do estado no plano cartesiano do software JFLAP.
- **Initial:** Uma flag booleana, indicando se o estado é um estado inicial (*True*) ou não (*False*).
- **Final:** Uma flag booleana, indicando se o estado é um estado final (*True*) ou não (*False*).

Já o objeto *Transition* que também possui uma classe própria, assim como o objeto *State*, representa as transições entre os estados desse AFD. Cada transição possui os seguintes atributos:

- **ID:** Um número inteiro salvo como carácter, que é a identificação da transição.
- **From:** Um número inteiro salvo como caractere, que indica o estado de partida da transição.
- **To:** Um número inteiro salvo como caractere, que indica o estado de destino da transição
- **Read:** Um caractere que é consumido ao se realizar uma transição de um estado a outro.

Por fim, através desses objetos, o objeto AFD, que representa o autômato, é construído. A classe AFD possui os seguintes atributos:

- **States:** Uma lista de objetos do tipo *State*, que comporta todos os estados do AFD.
- **Transitions:** Uma lista de objetos do tipo *Transition*, que comporta todas as transições do AFD.
- **Initial:** Um número inteiro, salvo como caractere, que representa o estado inicial do AFD.
- **Finals:** Uma lista de caracteres, contendo o ID de todos os estados que são finais.
- **Alphabet:** Uma lista contendo todos os caracteres que fazem parte do alfabeto do referido AFD.

Com esses objetos, conclui-se a constituição da interface *Model* do modelo MVC, e com isso a primeira parte do trabalho que era criar uma classe que representasse um Autômato, foi concluída.

Manipulação do AFD

Para a segunda etapa do trabalho, foi solicitado que se criasse funções para a manipulação deste AFD. Estas manipulações se dão através dos algoritmos vistos em sala de aula, e podem ser descritas a seguir:

- **Entrada de Dados:** A entrada de dados é obtida através do Software JFLAP, que permite criar autômatos e salva-los no formato *.jff*, que nada mais é que o formato XML. portanto, o usuário deve construir o autômato primeiro, através do JFLAP, e depois utilizar a opção de Carregamento no programa. Para tal, foi implementada uma função de leitura de arquivos XML, onde foi utilizada a biblioteca *ElementTree*. O arquivo *.jff*, possui *tags* pré definidas que permitem a leitura

fácil e obtenção das informações através do arquivo. Foram extraídas as seguintes informações deste arquivo:

- ID do estado
- Nome do Estado
- Posição X (plano cartesiano)
- Posição Y (plano cartesiano)
- Flag de estado inicial
- Flag de estado final
- ID do estado fonte da transição
- ID do estado destino da transição
- Caractere consumido na transição

Através da extração destes dados, foram criados objetos *State*, *Transitions*, o estado inicial do autômato, a lista de estados finais e o alfabeto, possibilitando então a criação do objeto AFD. Buscando facilitar ainda mais a interação com o usuário, na entrada de dados foi utilizada a biblioteca *TkInter* que fornece uma interface simples e funcional, tornando assim, mais fácil selecionar o arquivo desejado e no formato correto.

- **Saída de dados:** Na saída de dados, diferente da entrada, foi utilizada uma biblioteca chamada Dom que também realiza a manipulação de arquivo em formato xml. As mesmas tags utilizadas para a leitura do arquivo em formato JFlap, foram utilizadas, desta forma, é possível salvar o autômato no mesmo formato de entrada e se tornando totalmente "compatível" com o software JFlap. A interface fornecida pela biblioteca *TkInter* também foi utilizada na saída de dados, facilitando ainda mais a escolha de onde salvar o autômato. Além da opção de exportar os dados em formato do JFlap, é possível realizar a impressão em forma de *debug* pelo próprio terminal.
- **Estados equivalentes:** Esta foi a função mais complicada e a principal do trabalho, pois através dela, possibilitou-se implementar as outras funções de maneira fácil. O algoritmo para a implementação desta função, foi o mesmo utilizado em sala de aula. De maneira sucinta, foram realizados os seguintes passos:
 1. Criação de uma lista com os ID's dos estados do AFD.
 2. A partir desta lista, testa-se um a um, para verificar uma possível equivalência. É criada uma chave, contendo os ID's dos dois estados separados por vírgula. Ex: 1,2 (Estado 1 equivalente ao 2?), em uma tabela Hash, denominada Tabela de Equivalência. O valor desta chave pode ser X ou N, onde X representa que não são equivalentes e N que por hora são. Em um teste inicial, é verificado se as flags destes estados são finais. Caso um for final e o outro não, já não são equivalentes, logo a chave recebe o valor X. Caso contrário, recebe N
 3. Para cada estado, salva-se uma tabela hash contendo como chave o carácter lido, e como valor o destino.
 4. Para cada letra do alfabeto, obtém-se o destino da transição de cada estado, a partir das tabelas de cada estado montadas anteriormente.

5. Cria-se então uma chave (destino_estado1, destino_estado2), para se realizar o teste de equivalência na Tabela de Equivalências. Se na tabela, estes dois estados não forem equivalentes, logo os estados que estão sendo testados não são também. Então a tabela, na chave correspondente à esses estados recebe o valor X. Caso ainda não se souber se estes estados destino são equivalentes (valor N na tabela), adiciona-se à uma tabela denominada "amarrados", que possui como chave estes destinos, e como valor os estados que estão sendo testados no momento. Ou seja, Se num futuro os destinos não forem equivalentes, todos os estados que estão "amarrados" a eles também terão de ser marcados como não equivalentes
6. No final, do algoritmo, as chaves da tabela de equivalência que contém o valor N, são considerados Equivalentes.

- **Minimização de AFD's:** A função de minimização de AFD's funciona com base na função de equivalência de estados, onde, obtida a lista de estados equivalentes, convencionou-se em eliminar o primeiro estado da chave. Ex: Chave (1,3), elimina-se o estado 1. Caso o primeiro estado da chave for o inicial, elimina-se o segundo. Porém, antes de eliminar o estado do AFD, primeiro, as transições deste estado a ser eliminado são modificadas. Ou seja, todas as transições que tinham ele como destino, terão como destino o estado 2 da chave. E todas as transições que tinham o estado 1 como fonte, terão o estado 2 como fonte. Por fim, retorna-se o AFD mínimo.
- **Complemento de AFD's:** A implementação do método que realiza o complemento do autômato se tornou fácil depois que toda a estrutura de decisão de projeto proporcionou a lista de todos os estados. Tal método recebe como parâmetro o AFD que deseja realizar o complemento, desta forma, se torna fácil obter a lista dos estados através do método *getStates*. Tendo todos os estados retornados pelo método do modelo, bastou apenas percorrer a lista de estados e mudar o valor da variável booleana que sinalizava se o estado era final ou não. Por fim, o método retorna o objeto autômato com os estados finais atualizados.
- **União de AFD's:**
- **Intercessão de AFD's:**
- **Diferença de AFD's:**
- **Adiciona estado ao AFD:**
- **Remove estado do AFD:**
- **Adiciona transição ao AFD:**
- **Remove Transição do AFD:**

Validação

Para a validação dos dados, foram usados exemplos simples de exercícios que foram resolvidos em sala de aula. Para uma boa visualização, desde o início do desenvolvimento foi implementado a função de saída para arquivo de entrada do software JFlap. Desta forma, os testes e validações dos métodos que ainda iriam ser implementados, ficariam mais fáceis de se visualizar.

Demais testes que acabaram sendo mais elaborados, foram validados usando o próprio software JFlap.

Conclusão