Universidade Federal de Ouro Preto (UFOP)
Instituto de Ciências Exatas e Aplicadas (ICEA)

Autômatos Finitos Determinísticos: Operações

Disciplina: Fundamentos Teóricos da Computação
Professor: Vinícius Dias

Aluno: Marcos Henrique Santos Cunha - 19.1.8003

João Monlevade, Agosto/2021.

Introdução

Autômatos Finitos Determinísticos (AFDs) são formalismos reconhecedores capazes de reconhecer a classe das linguagens regulares. As linguagens regulares estão constantemente presentes no contexto de detecção de padrões em texto, sendo de grande importância para o mundo da tecnologia.

Existem diversas operações que podem ser feitas sobre AFDs, tais como complemento, união, interseção e minimização. Além disso, é possível também executar os AFDs para verificar se uma determinada palavra é ou não reconhecida pelo mesmo, e a possibilidade de visualizar um AFD de forma gráfica se mostra interessante, dada a complexidade de visualização de uma definição formal por extenso.

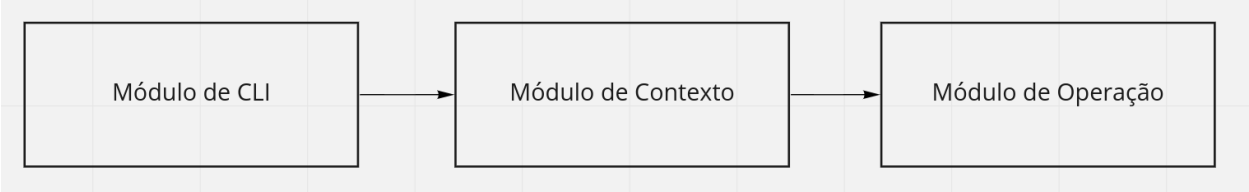
Dessa forma, o intuito deste trabalho é desenvolver um software capaz de executar todas as operações descritas acima sobre AFDs, além de permitir a visualização de forma gráfica de AFDs e também a execução de AFDs sobre palavras, verificando se elas são ou não reconhecidas. Para isso, foi desenvolvido um software dividido em módulos, executado por meio de linha de comando (CLI), que utiliza de diversas técnicas e estruturas de dados para atender todos os requisitos descritos acima.

Solução do Problema

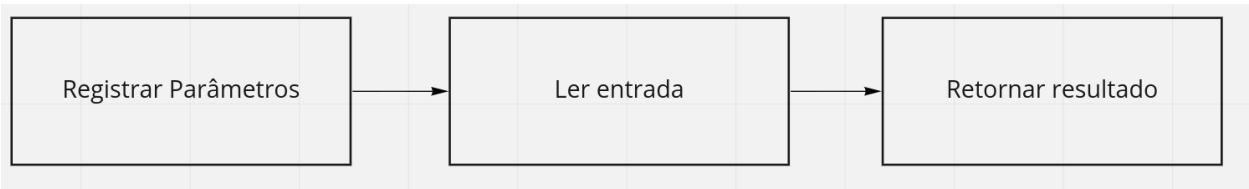
Funcionamento Geral

Para uma melhor estruturação do funcionamento do software, ele foi dividido em três módulos fundamentais, que operam de maneira independente. O módulo de CLI, é responsável por lidar com a entrada do usuário e com os parâmetros esperados na linha de comando. O módulo de contexto é responsável por pegar os resultados do módulo de CLI e fazer validações, verificando se não houveram erros de utilização pelo usuário. Finalmente, o módulo de operações é o responsável por pegar os dados de contexto e executar a operação desejada pelo usuário, bem como realizar as

operações de saída em arquivos, ao final da execução da operação desejada, permitindo que o usuário acesse os resultados obtidos. Na figura abaixo, é mostrado um diagrama que ilustra a forma como os módulos interagem entre si, e a direção do fluxo de dados na aplicação. O módulo de CLI é acionado primeiro, logo em seguida o módulo de contexto e só então o módulo de operação, que finaliza a execução do software.

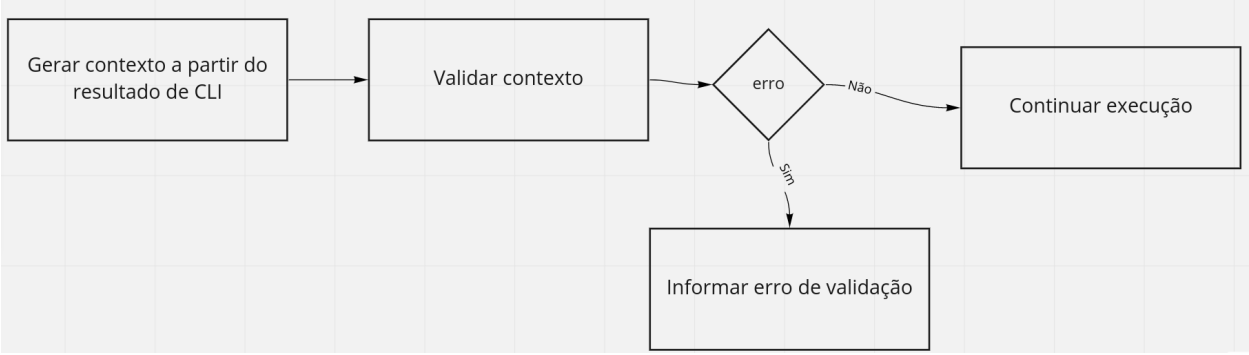


O módulo de CLI consiste em três partes fundamentais. Primeiramente, é necessário que o utilizador do módulo registre, de forma prévia, quais parâmetros espera receber do usuário, e quantos valores espera para cada parâmetro. Dessa forma, o módulo consegue executar o segundo e terceiro estágios, que envolvem a leitura e processamento da entrada, registrando os valores encontrados para cada parâmetro numa estrutura de resultado, que é posteriormente retornada para o usuário. As operações desse módulo são simples, com complexidade O(n) de tempo e espaço, onde n é o número de parâmetros registrados. Abaixo é mostrado um fluxograma que ilustra o fluxo de dados e processamento no módulo de CLI.

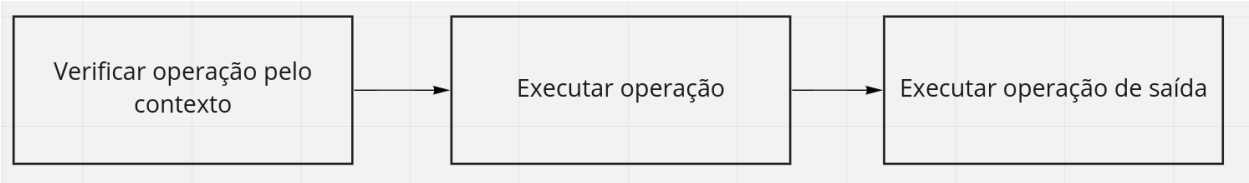


Ao final da execução do módulo de CLI, o módulo de contexto entra em ação. Ele pega os resultados do módulo de CLI e gera uma estrutura de contexto, verificando quais parâmetros foram preenchidos pelo usuário, e quais valores foram inseridos. Posteriormente, é feita uma validação dos dados inseridos, verificando principalmente se foi informado um arquivo de saída, se apenas uma operação foi escolhida, e se todos os valores necessários para a operação escolhida foram fornecidos. Caso não exista nenhum erro, o módulo continua a execução normalmente, e caso existam erros o módulo informa o usuário e interrompe a execução imediatamente. Esse módulo

apresenta complexidade O(n) de tempo, onde n é o número de valores lidos no CLI, e O(1) de espaço, sendo sempre igual independente da execução. Abaixo pode ser verificado um fluxograma que mostra de forma simples o funcionamento do módulo de contexto.



Finalmente, o módulo de operações é o responsável por receber um contexto validado e executar a operação desejada pelo usuário. Basicamente esse é o módulo que implementa os requisitos descritos anteriormente na introdução. A complexidade de tempo desse módulo é O(n), sendo n o número de operações disponíveis, e O(1) de espaço, pois precisa guardar dados já definidos que não mudam.



Operações

O software suporta cinco tipos de funcionalidades diferentes: visualização, complemento, interseção e união, minimização e reconhecimento de palavras. É importante notar que, ao final da execução de todos os algoritmos, eles escrevem seus resultados em um arquivo, de nome informado pelo usuário. A funcionalidade de escrita em arquivos é independente da execução das operações, ou seja, elas não estão amarradas de nenhuma maneira. O executor das operações é capaz de executar a operação, pegar o resultado e acionar as funções de escrita em arquivos para registrar a saída para o usuário. Cada uma das funcionalidades será explicada com detalhes abaixo, bem como a análise de complexidade dos algoritmos implementados.

Visualização

A visualização consiste em gerar um arquivo .dot a partir do afd, para que ele possa ser visualizado posteriormente utilizando uma ferramenta como o GraphViz. A solução utilizada foi basicamente projetar uma estrutura de dados que contém todas as informações necessárias para a geração de um arquivo .dot, e popular essa estrutura com os valores do AFD correspondente. É importante notar que, ao final da geração da estrutura de dados, é feita uma filtragem nas transições da estrutura de dados, juntando transições que saem do mesmo estado e levam ao mesmo estado, pois isso faz com que apenas um arco seja representado na visualização via GraphViz. A complexidade de tempo da geração da estrutura de dados é de O(n), onde n é o número de elementos do AFD, juntando estados, símbolos, transições, estados finais e estado inicial. Já a complexidade da filtragem é $O(n^2)$, onde n é o número de transições do AFD. A complexidade de espaço é também linear, de acordo com o número de elementos contidos no afd de entrada.

Complemento

A operação de complemento consiste basicamente em gerar uma cópia do AFD de entrada, trocando os estados iniciais pelos estados finais. A complexidade de tempo é linear para a cópia dos elementos, em função do número de elementos do AFD de entrada. A troca de estados finais para estados iniciais tem complexidade O(s * f), onde s é a quantidade de estados e f é a quantidade de estados finais. A complexidade de espaço é linear, uma vez que o AFD gastará basicamente o mesmo espaço que o outro para ser armazenado.

Interseção e União

Para execução de interseção e união, foi primeiro necessário o projeto de uma função que fosse capaz de gerar um autômato correspondente ao produto de dois AFDs, deixando somente o preenchimento dos estados finais a cargo do usuário. A complexidade de tempo da função de geração de produtos de AFDs é $O(n^2)$, onde n é o número médio de estados entre os dois AFDs. A complexidade de espaço é linear,

uma vez que será gerado um AFD que possui armazenamento equivalente aos dois AFDs de entrada.

Após gerar o produto de autômatos, basta escolher os estados que incluem estados finais dos dois AFDs para gerar a interseção entre eles. Já para gerar a união, basta selecionar os estados que incluem estados finais de pelo menos um dos AFDs. Esse processo tem complexidade O(s * x), onde s é o número de estados de um dos AFDs e x é o número de estados do outro AFD. A complexidade de espaço é linear em função da soma do número de estados dos AFDs, pois os estados precisam ser armazenados como estados finais, caso necessário.

Minimização

A minimização é o processo mais complexo executado por este software, uma vez que ela envolve fazer diversas operações com strings e muitos laços. Basicamente o processo de minimização consiste na criação de grupos de equivalência, onde posteriormente elementos do grupo passarão por um teste onde verificarão se todos eles caem nos mesmos grupos de equivalência, para todos os símbolos possíveis. Caso contrário, é criado um novo grupo de equivalência onde são colocados os elementos fora da regra. Antes de todo esse processo ocorrer, é necessário também remover os estados inalcançáveis do AFD, pois se eles são inalcançáveis eles são, portanto, inúteis.

Primeiramente, o software gera uma cópia do AFD original, porém sem estados inalcançáveis, e posteriormente executa o processo de minimização descrito acima. O processo de remoção de estados inalcançáveis tem complexidade linear de tempo e espaço, de acordo com o tamanho do AFD de entrada. Já o processo de geração de grupos de equivalência tem complexidade de tempo $O(n^2 * s)$, onde n é o número de estados do AFD de entrada e s é o número de símbolos do mesmo. Já a complexidade de espaço é linear em função da quantidade de estados do AFD de entrada. Ao final do processo, é gerado um AFD resultante da junção de cada um dos grupos de equivalência em um respectivo estado, e na redução das transições para transições entre grupos de equivalência. A memória alocada para o AFD intermediário é liberada ao final, pois ele não é mais necessário para nenhum processo.

Reconhecimento de palavras

O reconhecimento de palavras é uma funcionalidade especialmente interessante, que pode receber como parâmetro um AFD e uma lista de palavras, e retornar o resultado da execução do AFD sobre cada uma das palavras, isto é, dizer se o AFD aceita ou rejeita a palavra. A implementação dessa funcionalidade foi feita de forma recursiva, implementando uma versão computacional da função de transição estendida dos AFDs. Devido ao armazenamento em forma de lista dos estados, que foi uma escolha de projeto, a complexidade de tempo deste algoritmo é de O(w * n), onde w é o tamanho da palavra de entrada e n é o tamanho do conjunto de estados. Isso acontece pois a função de transição tem tempo linear em função dos estados, e ela será executada para cada termo da palavra, devido à utilização da função de transição estendida. A complexidade de tempo é linear em função do número de palavras, pois precisam ser armazenados os resultados do reconhecimento de cada uma das palavras. Ao final do processo, uma lista de resultados é retornada ao usuário.

Guia de utilização

O software é muito simples de ser utilizado, e funciona com base na interface de linha de comando. Primeiramente, basta entrar na pasta onde encontram-se os códigos fonte e compilar o projeto:

```
marcoshsc@marcoshsc:~/Programs/afd-operacoes$ make -B
```

Agora que o software está compilado, é possível utilizar suas funcionalidades. Como dito anteriormente, o software funciona em linha de comando, e consegue filtrar as operações desejadas por meio de flags. Cada uma das flags será descrita com detalhes a seguir:

- --output: Representa a saída do programa. Aceita somente um argumento, que deve ser o nome do arquivo de saída do programa.
- --dot: Funcionalidade de visualização. Aceita um argumento que deve ser o nome do arquivo com o AFD de entrada. Essa funcionalidade escreve na saída um arquivo DOT que pode ser visualizado por meio do GraphViz.

- --complemento: Funcionalidade de complementação de AFD. Aceita um argumento que deve ser o nome do arquivo com o AFD de entrada. Essa funcionalidade escreve um AFD na saída.
- --intersecao: Funcionalidade de interseção entre AFDs. Aceita dois argumentos, que devem ser os nomes dos arquivos com os dois AFDs de entrada. Essa funcionalidade escreve um AFD na saída.
- --uniao: Funcionalidade de união entre AFDs. Aceita dois argumentos, que devem ser os nomes dos arquivos com os dois AFDs de entrada. Essa funcionalidade escreve um AFD na saída.
- --minimizacao: Funcionalidade de minimização de AFD. Aceita um argumento que deve ser o nome do arquivo com o AFD de entrada. Essa funcionalidade escreve um AFD na saída.
- --reconhecer: Funcionalidade de reconhecimento de palavra. Aceita dois argumentos, onde o primeiro é o nome do arquivo com o AFD de entrada, e o segundo é o nome do arquivo com as palavras que serão testadas no AFD de entrada. Escreve na saída o resultado do reconhecimento das palavras.

É importante salientar algumas regras para a execução do software: a flag --output deve estar presente em toda execução do software, e somente uma das outras flags podem estar presentes ao mesmo tempo. Por exemplo, não é permitido que se utilize a flag --reconhecer e --dot ao mesmo tempo. Caso essas regras não sejam atendidas o software notificará o usuário.

Exemplos de uso

Visualização

```
marcoshsc@marcoshsc:~/Programs/afd-operacoes$ ./afdtool --dot afd.txt --output afd.dot
marcoshsc@marcoshsc:~/Programs/afd-operacoes$
```

Complemento

```
marcoshsc@marcoshsc:~/Programs/afd-operacoes$ ./afdtool --complemento afd.txt --output afd_complemento.txt
marcoshsc@marcoshsc:~/Programs/afd-operacoes$
```

Interseção e União

```
marcoshsc@marcoshsc:~/Programs/afd-operacoes$ ./afdtool --intersecao afd.txt afdCopy.txt --output afd_new.txt
marcoshsc@marcoshsc:~/Programs/afd-operacoes$ ./afdtool --uniao afd.txt afdCopy.txt --output afd_new.txt
marcoshsc@marcoshsc:~/Programs/afd-operacoes$
```

Minimização

```
marcoshsc@marcoshsc:~/Programs/afd-operacoes$ ./afdtool --minimizacao afdCopy.txt --output afd_new.txt
marcoshsc@marcoshsc:~/Programs/afd-operacoes$
```

Reconhecimento de Palavras

```
marcoshsc@marcoshsc:~/Programs/afd-operacoes$ ./afdtool --reconhecer afd.txt palavras.txt --output palavras-reconhecidas.txt
marcoshsc@marcoshsc:~/Programs/afd-operacoes$
```

Combinação de funcionalidades

Naturalmente, um software com múltiplas funcionalidades pode permitir que uma sequência delas possa ser feita em sequência, para atingir um determinado objetivo. A combinação principal de ações deste software consiste em executar alguma das operações que retornam AFDs e posteriormente executar a funcionalidade de visualização em cima do resultado, para posteriormente executar o GraphViz em cima dela e conseguir visualizar graficamente os resultados. Segue abaixo um exemplo disso:

```
marcoshsc@marcoshsc:~/Programs/afd-operacoes$ ./afdtool --minimizacao afdCopy.txt --output afd_new.txt
marcoshsc@marcoshsc:~/Programs/afd-operacoes$ ./afdtool --dot afd_new.txt --output afd_new.dot
marcoshsc@marcoshsc:~/Programs/afd-operacoes$ dot -Tpdf afd_new.dot > afd_new.pdf
marcoshsc@marcoshsc:~/Programs/afd-operacoes$
```

Ressalta-se que essa é uma combinação particularmente útil, porém que qualquer sequência pode ser executada, tanto com as operações sozinhas ou juntas, desde que respeite as restrições de cada operação.

Conclusão

Neste trabalho, foi desenvolvido um software capaz de executar múltiplas operações sobre AFDs, que é particularmente útil para todos os profissionais e estudantes que precisam trabalhar com tal ferramenta. Seu desenvolvimento gerou um grande aprendizado, e uma melhora nas habilidades de desenvolvimento de software em linguagem C. Como sugestão de trabalho futuro, poderiam ser implementadas funcionalidades sobre AFNs e AFNs com transições lambda.