Relatório da implementação do método de conversão de ERs para AFDs

Bruno Pena Baêta
Instituto de Ciências Exatas e Informática (ICEI)
Pontifícia Universidade Católica de Minas Gerais
Belo Horizonte, Brasil
bbaeta@sga.pucminas.br

Felipe Nepomuceno Coelho
Instituto de Ciências Exatas e Informática (ICEI)
Pontifícia Universidade Católica de Minas Gerais
Belo Horizonte, Brasil
felipe.coelho.1265277@sga.pucminas.br

I. Introdução

Uma das formas de representar uma linguagem é através de autômatos finitos (AF), eles podem ser determinísticos (AFD) ou não determinísticos (AFN). Ainda há os autômatos não determinísticos com transições vazias, ou transições λ (AFN-λ). Para todo autômato, existe uma ou mais expressões regulares (ER) que representam a mesma linguagem. Esses autômatos podem ser convertidos em expressões e vice-versa. O objetivo deste trabalho é implementar e analisar um algoritmo criado para transformar expressões regulares em autômatos finitos determinísticos. Na seção 2, será apresentado o programa que foi desenvolvido para executar os testes. Na seção 3, será explicado o funcionamento do algoritmo. Já na seção 4, iremos apresentar os testes feitos e resultados obtidos. Por fim, na seção 5 será apresentada a conclusão deste trabalho.

II. Programa

Para testar e executar os algoritmos, foi criado um programa escrito na linguagem de programação C++. O programa consiste em 5 arquivos e 1 biblioteca. Cada um será explicado a seguir.

A. main.cpp

É o arquivo principal, ele cria um menu que permite que o usuário manipule o programa criado e faça seus testes. O menu contém opções para ler ERs, transformar ERs em AFNs-λ, transformar AFNs-λ em AFNs, transformar AFNs em AFDs, exportar AFDs para o JFlap [1] (versão 7.0), testar uma sentença e testar múltiplas sentenças.

B. fa.cpp

Esse arquivo é responsável por conter a classe "FA", que é a classe que representa um autômato finito (AF). Ela tem as propriedades e métodos necessários para criar e manipular um AF. Assim como o método para testar uma sentença no AF.

C. superFa.cpp

Esse arquivo é responsável por conter a classe "SuperFA", que representa um super AF. Um super AF é um autômato finito que contém "Super Estados" no lugar de estados comuns e "Super Transições" no lugar de transições comuns. Super estados são estados que representam um conjunto de estados, eles valem como múltiplos estados simultaneamente. Super transições são como transições comuns, mas transitam de um super estado para outro super estado.

D. algorithms.cpp

Arquivo que contém os algoritmos. Esse arquivo é responsável por fornecer as funções dos algoritmos que serão chamados pelo arquivo *main.cpp*.

E. utils.cpp

Arquivo que contém as funções auxiliares do programa. As funções que não são relacionadas à função principal do programa e ainda sim são muito utilizadas.

F. Biblioteca pugixml

A biblioteca *pugixml* [2] é uma biblioteca que foi utilizada para criar arquivos do tipo XML que são lidos pelo JFlap [1] versão 7.0. O programa tem a funcionalidade de importar e exportar arquivos XML para o JFlap, para que os autômatos possam ser mostrados na ferramenta.

III. Algoritmo

O algoritmo criado é dividido em 3 partes: conversão de ER para AFN-λ, AFN-λ para AFN e AFN para AFD. Nesta seção iremos passar por cada uma dessas 3 partes.

A. Conversão ER para AFN-λ

A primeira etapa de conversão é a conversão de expressão regular para um autômato não determinístico com transições vazias. Esse algoritmo consiste em ler a expressão e ir criando sub-autômatos e os unindo, criado o autômato final.

Composto por um método recursivo, o algoritmo lê os caracteres da expressão e verifica o que deve fazer para cada um deles. Primeiramente, o caractere pode cair em 1 de 4 casos: ser o caractere "+", ")", "(" ou outro.

Se o caractere não for "+" ou ")" ou "(", o caractere é classificado como um caractere do alfabeto do autômato, ou seja, ele será responsável por uma nova transição. Para essa situação, é criado um sub-autômato onde esse caractere faz uma transição do estado inicial para o estado inicial. Se o caractere for "λ", essa transição é vazia. Em seguida, é verificado se o próximo caractere é um Fecho de Kleene (*), se for, é feito um sub-autômato do sub-autômato já existente,

mas para um Fecho de Kleene. A partir disso, o sub-autômato é concatenado ao autômato já existente até agora e é lido o próximo caractere da expressão.

Se o caractere for "(", o método é chamado recursivamente para criar o sub-autômato responsável pela expressão que está dentro dos parênteses. Por isso, se o caractere for ")", o método apenas retorna o sub-autômato atual. Importante salientar que, após os parenteses, também é feita a verificação do Fecho de Kleene como explicado anteriormente.

Se o caractere for "+", o método faz a união do autômato atual com o sub-autômato que será gerado como uma chamada recursiva.

Caso não haja mais caracteres na expressão, o autômato gerado é retornado, concluindo o algoritmo e gerando um AFN-λ a partir da ER dada.

B. Conversão AFN-λ para AFN

Inicialmente, o algoritmo gera os fechos vazios de cada estado do autômato. Os fechos vazios são conjuntos de estados de podem ser alcançados a partir de um determinado estado sem que seja necessário nenhum símbolo, ou seja, todos os estados alcançáveis apenas com transições λ.

Em seguida, são criadas as novas transições δ' a partir das transições δ já existentes. A transição δ' de um estado E com o símbolo s ($\delta'(E, s)$) é dada pela união $\cup$ dos fechos vazios ($f\lambda$) do δ de cada estado que pertence ao conjunto de estados do fecho vazio de E. Isso é feito para todo estado e símbolo no autômato.

$$\delta'(E, s) = \bigcup_{e \in E} f\lambda(\delta(e, s))$$

Caso haja algum estado final do fecho vazio do estado inicial, o estado inicial também é adicionado ao conjunto dos estados finais.

Ao final, é criado um novo autômato com novas transições e estados finais, transformando um AFN-λ em um AFN.

C. Conversão AFN para AFD

Antes de descrever essa conversão, é importante ressaltar que sua implementação também teve como referência o livro de José Newton, Introdução aos fundamentos da computação: Linguagens e máquinas [3].

Para essa conversão, será criado um Super AF inicialmente vazio que será completado ao longo do algoritmo. Como primeiro passo o algoritmo passa por todas as transições do AFN inicial, adicionando super estados no super AF. Esses super estados são criados a partir do conjunto de estados os quais determina transição leva. Em AFNs, uma transição pode levar a múltiplos estados, e esses múltiplos estados são convertido em um único super estado nesse super AF. A partir disso, em vez da transição levar a múltiplos estados, ela levará a um super estado, se tornando uma super transição.

Para o segundo passo, são criadas novas super transições. Essas novas super transições δ' são criadas a partir da união $\cup$ dos conjuntos os quais as transições comuns δ levavam para cada estado e dentro do super estado E para um símbolo s. Isso é feito para todo super estado e símbolo no autômato.

$$\delta'(E, s) = \bigcup_{e \in E} \delta(e, s)$$

Caso essa nova super transição δ' leve a um conjunto de estados que ainda não foi representado por um super estado, é criado um novo super estado para esse conjunto. Esse passo se repete até que todos os super estados do autômato tenham suas super transições tratadas.

Os estados finais serão a união entre os estados finais do AFN com os super estados que contém esses estados finais do AFN.

Ao final, o super AF é convertido em um autômato finito determinístico comum, concluindo a conversão de AFN para AFD.

IV. Testes e Resultados

Para fazer os testes, foram criadas diversas expressões regulares, assim como sentenças para serem testadas nos autômatos de tais expressões. Nesta seção, serão apresentados alguns casos e analisado o tempo tomado para as conversões e testes de sentenças. Para os testes menores, os AFs gerados também terão uma representação visual. Para as representações visuais, foi usado o software JFlap [1] na versão 7.0.

Está sendo usada uma máquina com o sistema operacional Windows 10, compilador g++ versão 11.2.0, 12GB de memória RAM e um processador Intel Core i7 de 10ª geração.

A fins de apresentação visual, os AFDs gerados estarão minimizados, mas essa minimização não será levada em conta no cálculo de tempo do algoritmo.

A. Teste 1

Expressão Regular: ab

AFN-λ:

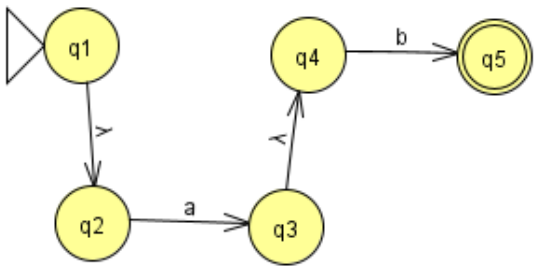

Fig. 1. AFN-$lambda$ do teste 1

AFN:

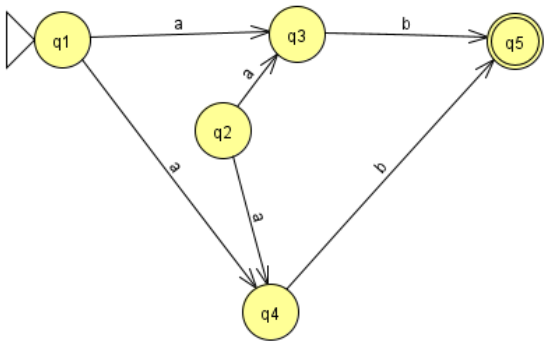

Fig. 2. AFN do teste 1

AFD:

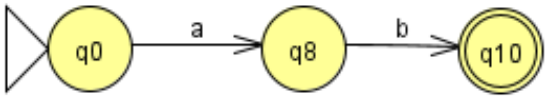

Fig. 3. AFD do teste 1

Sentenças:

- aab → Rejeita
- ab → Aceita
- ba → Rejeita

Tempo de conversão ER → AFN-λ: 1ms

Tempo de conversão AFN-λ → AFN: 0ms

Tempo de conversão AFN → AFD: 0ms

B. *Teste 2*

Expressão Regular: $a + b^*$

AFN-λ:

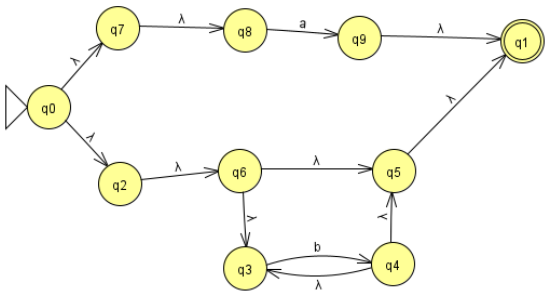

Fig. 4. AFN-*lambda* do teste 2

AFN:

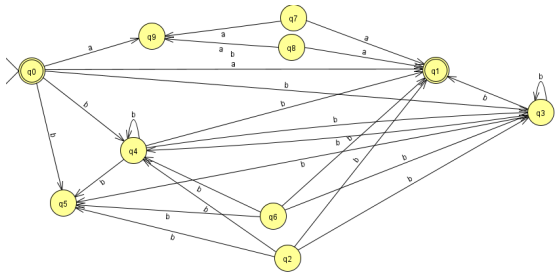

Fig. 5. AFN do teste 2

AFD:

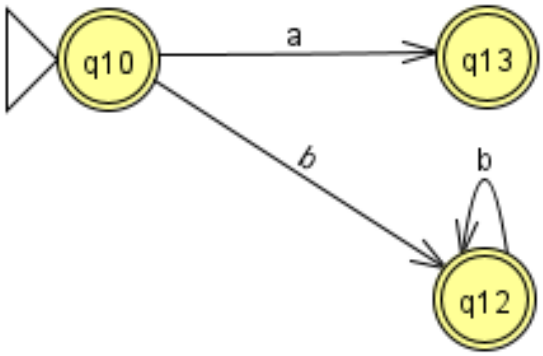

Fig. 6. AFD do teste 2

Sentenças:

- abbbb → Rejeita
- b → Aceita
- baaab → Rejeita
- a → Aceita
- bbbbb → Aceita

Tempo de conversão ER → AFN-λ: 1ms

Tempo de conversão AFN-λ → AFN: 0ms

Tempo de conversão AFN → AFD: 1ms

C. *Teste 3*

Expressão Regular: $a + \lambda + ab + (bb)^*$

AFN-λ:

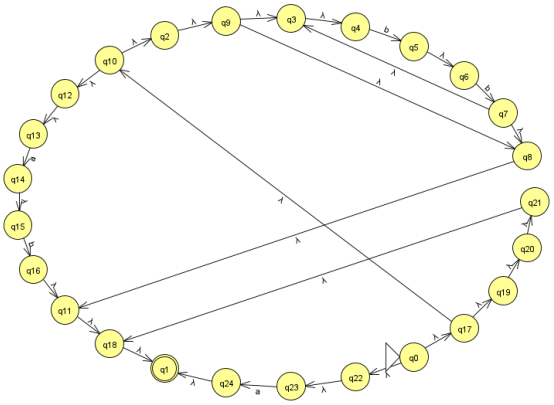

Fig. 7. AFN-$lambda$ do teste 3

AFN:

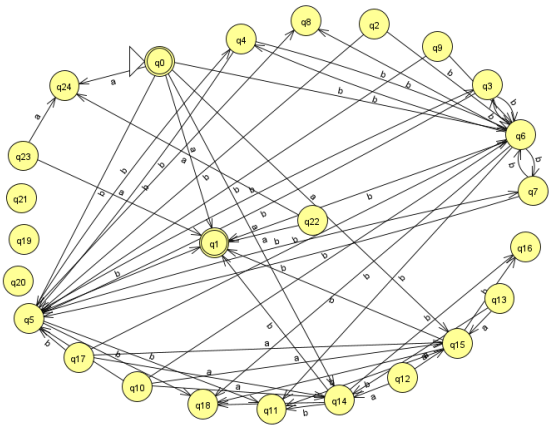

Fig. 8. AFN do teste 3

AFD:

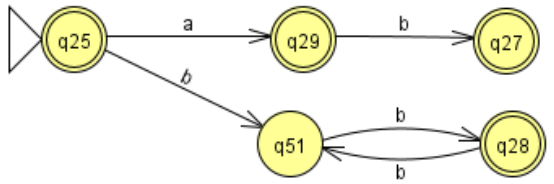

Fig. 9. AFD do teste 3

Sentenças:

- $\lambda \rightarrow$ Aceita
- a $\rightarrow$ Aceita
- abb $\rightarrow$ Rejeita
- bbb $\rightarrow$ Rejeita
- bbbbbb $\rightarrow$ Aceita

Tempo de conversão ER $\rightarrow$ AFN-λ: 3ms
Tempo de conversão AFN-$\lambda \rightarrow$ AFN: 1ms
Tempo de conversão AFN $\rightarrow$ AFD: 6ms

V. Conclusão

A partir dos testes feitos e resultados obtidos, concluímos que o método proposto é um sucesso. O tempo de execução é perfeitamente aceitável e a acurácia foi de 100%.

É proposta para pesquisas futuras a otimização e melhoria desse algoritmo, visto que a análise de complexidade não era o objetivo deste trabalho.

References

[1] S. Rodger and T. Finley, "Jflap," 2015.

[2] J. Weinbub, R. Heinzl, P. Schwaha, F. Stimpfl, and S. Selberherr, "A lightweight material library for scientific computing in c++," in *Proceedings of the European Simulation and Modelling Conference (ESM)*, 2010, pp. 454–458.

[3] N. Vieira, Ed., *"Introdução aos fundamentos da computação: Linguagens e máquinas"*. Cengage Learning, 2006.