Construção de um Autômato Finito Determinístico livre de épsilon transições e mínimo

Adriel Schmitz J. de Paula[1], **Leonardo Gonçalves**[2]

[1]Universidade Federal da Fronteira Sul (UFFS)

[2]Curso de Ciência da Computação – Chapecó, SC – Brasil

adrielschmitz10@gmail.com[1], leoojg@gmail.com[2]

Abstract. *This meta-article aims to describe how the construction of a finite deterministic automaton was minimal and free of epsilon transitions. Python3 is used as the programming language for the creation of the algorithm, and code fragments and test outputs are shown in order to better elucidate the subject. It also contains brief explanations on finite automaton, in order to make the reader aware of the subject. Finally, it is important to remember that the article follows all the standards imposed by the SBC.*

Resumo. *Este meta-artigo tem por objetivo descrever como foi a construção de um autômato finito determinístico mínimo e livre de épsilon transições. Utiliza-se python3 como linguagem de programação para a criação do algoritmo e são mostrados fragmentos de códigos e resultados das saídas dos testes afim de elucidar melhor acerca do assunto. Contém também breves explicações sobre autômatos finitos, com o intuito de deixar o leitor inteirado no assunto. Por fim, é importante lembrar que o artigo segue todas as normas impostas pela SBC.*

1. Autômato Finito Determinístico

Existe um ramo na parte teórica de Ciência da Computação chamado Teoria dos Autômatos, e segundo Hopcroft, pode-se dizer que um Autômato Finito Determinístico (AFD) é uma máquina de estados finita que aceita ou que rejeita cadeiras de símbolos gerando um único ramo de computação para cada cadeia de entrada. A parte do ”Determinístico”indica à unidade do processamento. O autômato finito foi criado com o intuito de reproduzir, através de estruturas mais simples, máquinas de estado finitas.

2. Passos para a criação do AFD

Como já foi dito, utilizou-se Python3 para a criação do algoritmo. A seguir será mostrado e explanado alguns fragmentos de códigos.

2.1. Leitura dos Tokens e da Expressão Regular

A função get-tokensExpressao é responsável por ler os tokens até que seja encontrado uma linha. Os tokens lidos são adicionados em uma lista, para serem usados futuramente. O mesmo é feito para a leitura da Expressão Regular, faz-se a leitura e armazena em uma lista até que seja encontrado o fim do arquivo.

2.2. Criação do Autômato

Tendo os tokens salvos em uma lista, para cada token é feita o mapeamento no autômato. Em seguida, é feita a cópia da expressão regular para o autômato.

2.3. Eliminação de épsilon transições

É feito uma varredura no autômato verificando se existe alguma épsilon transição em alguma regra criando os conjuntos de cada produção. Em seguida, verifica se existe alguma épsilon transição, se existir faz a copia os estados dessa transição para a produção que ele achou a épsilon transição. Após isso, verifica se existe alguma épsilon transição indireta, varrendo os conjuntos feitos a parir do autômato. Se existir, adiciona as produções que faltam na regra em questão.

2.4. Determinização

Para tornar o autômato determinístico, varre o autômato verificando se existe o indeterminismo. Uma vez encontrado, cria um estado novo referente aos casos de não determinismo e verifica se o estado criado ainda não existe no autômato. Se o estado não existir, ele é adicionado à uma fila, terminando a varredura no autômato, é necessário criar os estados que estão na fila. Então é repetido o processo de determinização para todos os itens da fila. Caso no processo anterior, algum estado novo seja encontrado, esse estado também é adiciono à fila. Com isso, tem-se um autômato finito determinístico (AFD).

2.5. Minimização

Nessa etapa, o objetivo é a eliminação dos estados inalcançáveis e/ou mortos. Para fazer isso, cria-se uma lista dos estados alcançados a partir de ¡S¿. Cria-se também uma lista contendo os estados finais, em seguida, para cada produção do AFD, é verificado se essa produção é alcançável a partir de ¡S¿. Após isso, é gerado uma lista com as produções que a produção em questão alcança, então, verifica-se se pelo menos um item da lista de estados finais está contido na lista de alcançáveis da lista da produção. Uma vez estando, significa que esse estado tem algum caminho que levará à um estado final. Caso alguma produção não seja alcançável a partir do ¡S¿, ou se ela não tenha algum caminho para uma produção que tenha um estado final, essa produção é eliminada. Por fim, verifica-se se dentre as produções restantes existe algum caminho que leve a uma produção que já fora removida, então, esse caminho é removido.

3. Teste realizados

Entrada:

```
se
senao

<S> ::= a<A> | e<A> | i<A> | o<A> | n<A>
<A> ::= a<A> | e<A> | i<A> | o<A> | n<A> | eps
```

Acima tem-se o arquivo de carga contendo os tokens e a gramática regular.

Saída:

```
<S> ::= a<A> | e<A> | i<A> | n<A> | o<A> | s<-13>
<5> ::= a<6> | e<-> | i<-> | n<-> | o<-> | s<->
<6> ::= a<-> | e<-> | i<-> | n<-> | o<7> | s<->
<7> ::= a<-> | e<-> | i<-> | n<-> | o<-> | s<-> | eps
```

```
<A> ::= a<A> | e<A> | i<A> | n<A> | o<A> | s<-> | eps
<-13> ::= a<-> | e<-24> | i<-> | n<-> | o<-> | s<->
<-24> ::= a<-> | e<-> | i<-> | n<5> | o<-> | s<-> | eps
<-> ::= a<-> | e<-> | i<-> | n<-> | o<-> | s<->
```

A partir da entrada, obteve-se a saída acima. As produções marcadas com -", indica que essas foram obtidas a partir da minimização.

4. Conclusão

É possível concluir que o maior desafio desse trabalho foi entender o que deveria ser feito, como fazer para que o algoritmo fosse o mais genérico e correto possível. Outro ponto é o estado de erro e a eliminação de épsilon transições, nesses foi preciso um pouco mais de planejamento de como seria implementado corretamente. A escolha da linguagem de programação foi um ponto positivo, foi possível focar mais tempo na construção do autômato e não aprendendo uma linguagem nova. Os testes feitos provaram o ótimo funcionamento do autômato criado, o que demonstra um bom prognóstico. Por fim, é importante salientar a enorme contribuição do trabalho no entendimento do assunto, ajudou muito a entender o passo a passo de como o AFD funciona e quão útil ele pode ser.

5. Referências

Hopcroft, John E.; Motwani, Rajeev; Ullman, Jeffrey D. (2001). Introduction to Automata Theory, Languages, and Computation 2 ed. [S.l.]: Addison Wesley. ISBN 0-201-44124-1. Consultado em 09 de julho de 2018.