PROJETO LFA - Construção de Automato Finito Determinístico Livre de Épsilon Transições, Determinizado e Minimizado

Anthony Cassol Douglas Davy Breyer

2018

Resumo

O seguinte trabalho é o relatório do desenvolvimento de uma aplicação que, recebendo como entrada tokens e gramáticas regulares de uma linguagem hipotética devolve como saída um autômato finito deterministico (AFD). Também será apresentado brevemente alguns pontos necessários para melhor entendimento do trabalho desenvolvido, abordando conceitos e definições importantes assim como explicar o passo a passo desde a entrada, estrutura do algorítmo, logica de desenvolvimento e o resultado obtido.

Palavras-chaves: Gramática regular, Autômato, algoritmo.

Introdução

Dentro do estudo da Computação ou da Ciência da Computação, a teoria dos autômatos é o estudo de máquinas abstratas ou autômatos e os problemas que podem ser resolvidos se utilizando destas maquinas. Esses autômatos ou maquinas são compostos de estados (representados pelos circulos) e transições (representadas pelas setas) como pode ser visto na imagem 1.1.

Usando como exemplo a maquina da imagem acima, podemos ver que no momento em que ela recebe uma entrada ela toma uma das transições e parte parte para um outro estado, que as vezes pode ser ele mesmo. Neste caso, são possiveis duas opções, se o automato recebe "0"ele sai do estado S1, toma a transição pelo valor 0 e vai para o estado S2. Senão ele tomaria a transição 1 e voltaria para o estado de origem. No estado S2 acontecerá a mesma coisa. No momento que receber uma valor como entrada, esse valor é analizado e assim o maquina irá para o novo estado correspondente com a entrada.

Neste trabalho será desenvolvido uma aplicação em PHP que receberá uma entrada, a partir dela criará um autômato finito não deterministico, depois disso serão removidas as épsilon transições, depois esse AFND, depois esse AF será determinizado gerando um automato finito deterministico (AFD. Depois ainda esse AFD será minimizado e por ultimo os estados não mapeados serão ajustados à levar a um estado de erro.

1 Tipos de Autômatos

1.1 AFN - Autômato Finito Não Determinístico

Um AFN (Autômato finito não deterministico) possui as mesmas caracteristicas de outros autômatos como os citados na seção anterior. Esse autômato receberá uma entrada e, de acordo com ela, ele se comportará de certa maneira. Porem o que o torna um AFN é que existe pelo menos um estado que quando lido um simbolo possui mais de um estado destino como é possivel ver no exemplo abaixo na imagem 1.2.

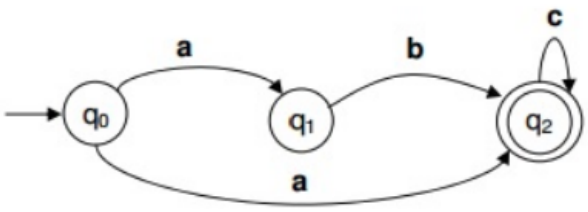

Figura 1 – Exemplo de autômato finito não deterministico

Pela definição formal:

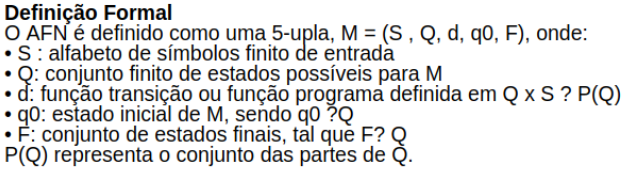

Figura 2 – Definição formal de uma AFN

1.2 AFD - Autômato Finito Determinístico

AFD ou também conhecido como maquina de estados finita deterministica é uma maquina/automato que possui um numero de estados finitos que aceita ou rejeita cadeias de simbolos/caracteres gerando um ramo de computação que numa analogia com arvores seria como partir da raiz até um nó

folha em que essa linguagem seria aceita ou rejeitada. Esse automato é finito pois o conjunto de estados possiveis é finito e deterministico pois ao ler uma entrada este possui apenas um proximo estado possivel, diferente dos AFN.

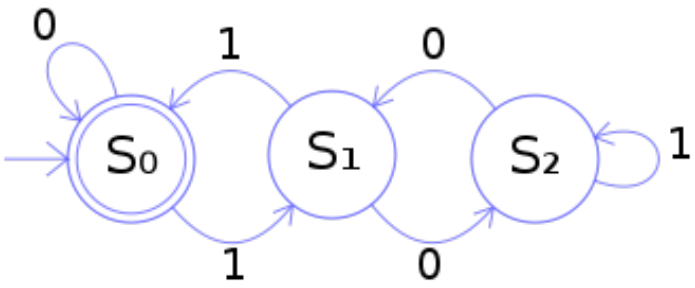

Figura 3 – Exemplo de um Autômato Finito Determinístico

Definição Formal
Um Autômato Finito Determinístico (AFD) é definido por uma quíntupla M = (S , Q, d, q0, F), onde:
• S : alfabeto de símbolos finitos de entrada
• Q: conjunto finito de estados possíveis para M
• d: função transição ou função programa definida em Q x S ? Q
• q0: estado inicial de M, sendo q0 ?Q
• F: conjunto de estados finais, tal que F? Q

Figura 4 – Definição formal de um Autômato finito não deterministico

2 Desenvolvimento da Aplicação

Pelo tipo de entrada de tratamento (strings) e pela afinidade, foi escolhido fazer o trabalho em PHP. O código se divide em dois arquivos, um é o *index.php* que é a função main do código. É nesse arquivo que ocorrem a leitura de arquivos e as chamadas das demais funções que estão no outro arquivo chamado *funcoes.php*.

2.1 Estrutura e Construção do Autômato Finito não Deterministico

Primeiramente o código lê o arquivo txt *entrada.txt* e percorre cada linha. Logo após, cada cada linha encontrada é mandada para a função *insereSimbolos*. Nesta função, a partir do simbolo inicial das gramaticas «"distingue entre gramatica e token (os tokens seriam as palavras da entrada). Se a linha correspondente for um token então é chamada a função de *insereToken*, senão é chamada a função *insereGR*. Estas duas funções são distintas pois elas necessitam de tratamentos diferentes para depois elas comporem a tabela de símbolos que é a primeira linha da tabela.

Na função *insereToken* os caracteres da linha que é uma palavra são desconcatenados e cada um assume um indice no vetor da tabela de simbolos *tabSimb*. A função *insereGR* se assemelha a anterior, entretanto nas gramáticas se encontram caracteres que não necessitam estar na tabela de simbolos,

então temos que descartalos. Pega-se apenas os caracteres importantes da gramática. Essas duas funções retornam a variável *tabSimb* para a função anterior *insereSimbolos* que retorna de forma recursiva para o foreach no arquivo index. O retorno *tabSimb* é um array que possui todos os não terminais encontrados na gramática, a posição que ele está nesse array será usado para trata-lo na matriz do autômato finitio que será criado em seguida.

Com a tabela de símbolos finalizada, faz-se o mesmo processo anterior de leitura das linhas da entrada. Porém agora são chamadas as funções *leToken* e *leGR*. Verifica-se se a linha em questão é token ou GR e depois faz-se o tratamento para inseri-las na matriz do autômato. O retorno é a varíavel *afnd* que contém o autômato em forma matricial.

O processo se repete até o fim do arquivo.

2.2 Transições Vazias

O primeiro passo para tornar um autômato determinístico é eliminar as transicões vazias. Para remover as episolon transições(transições vazias), no arquivo index é chamada a função *transicoesvazias* passando como parâmetro as variáveis *afnd* e *tabSimb*. O símbolo de vazio será considerado como o *"*"*. No inicio a flag *transicaovazia* inicia como 0. Depois é verificado se existe uma transição vazia '*' na tabela de simbolos. Se ele tiver o símbolo '*', verifica-se na coluna do símbolo '*' se existe um estado que tenha uma transição vazia. No caso da existência, trata-se a transição vazia, fazendo com que o estado que contém receba todas as transições do estado em que ele chega através do vazio e a linha na coluna '*' é zerada (NULL). Quando encontrado a flag *transicaovazia* recebe 1. É verificado a existência de transição até o momento em que não ser encontrada mais nenhuma, ou seja, até que a flag termine o processo com 0. Assim a variavel *afnd* retorna para a index com o automato finito não deterministico formado e livre de transições vazias.

2.3 Determinização

Na função *determinizar* é enviado o automato e a tabela de simbolos. É percorrida a matriz e verificado os indices da matriz que possuem indeterminismo (quando através de uma produção não final pode-se ir para mais de um estado distintos). Para solucionar isso, é criado um novo estado nomeado com a concatenação dos causadores do indeterminismo. Este novo estado recebe todas as transições seguintes dos estados encontrados em indeterminismo e suas caracteristicas. Quando afunção acaba, os valores passam para a variável *afd* pois o autômato já se configura como um autômato finito determinístico.

2.4 Eliminação de Mortos

Para efetuar a eliminação dos mortos no autômato é chamada a função *eliminamortos* que lê os estados alcançáveis do autômato e adiciona estes em um array chamado *alcancaveis*. Depois é feita a comparação entre esse array

com os estados alcançáveis, os que estão contidos no autômato mas não estão contidos no array de alcançaveis são removidos do autômato já que, se não estão contidos no array de alcançáveis eles nunca serão alcançados, portanto são mortos.

2.5 Equivalência

Na função chamada *equivalencia* o algoritmo se resume a verificar entre as linhas do autômato quais estados são equivalentes, ou seja, levam aos mesmo estados a partir das mesmas entradas. O algoritmo também compara se eles são terminais ou não. Só ocorre equivalência se eles manterem a propriedade a cima e os dois forem terminais ou se os dois não forem terminais. Encontrando essas equivalências são geradas classes que são printadas na tela exibindo quais são semelantes.

Considerações finais

Pode ser visto com esse trabalho a importância de se estudar e trabalhar estes assuntos e algoritmos já que estes podem ser aplicados nas mais diversas finalidades como linguagens de programação, análise sintática, análise lexica, na modelagem de redes e circuitos lógicos como muitas outras. Com este o entendimento da disciplina foi ampliado e colocado em prática o que ajuda na fixacão do mesmo. Ainda salientamos que o projeto não foi desenvolvido em sua totalidade e que ainda faltam alguns detalhes e alguns passos finais para a conclusão completa do trabalho, entretanto o tempo e certas dificuldades na implementação acabaram acarretando no atraso dele.

Referências

http://www.dsc.ufcg.edu.br/ pet/jornal/junho2014/materias/recapitulando.html