AUTÔMATO FINITO DETERMINÍSTICO:
CONSTRUÇÃO DE APLICAÇÃO GERADORA[1]

Karen Ruver Mentges
Izabela Fusieger

RESUMO

O presente artigo apresenta um breve referencial teórico, o qual embasa o desenvolvimento de uma aplicação de geração de autômatos finitos determinísticos a partir de tokens ou gramáticas regulares. A linguagem utilizada na aplicação é php, combinada a banco de dados MySQL para armazenamento dos dados.

Palavras-chave: Gramática Regular; Autômatos Finitos.

1 INTRODUÇÃO

Com este artigo se busca apresentar um referencial teórico que apresenta conceitos necessários para o desenvolvimento de uma aplicação que receba um arquivo com tokens e gramáticas regulares para então gerar um autômato finito não-determinístico, a partir deste com a utilização do teorema da deteminização se obterá um autômato finto determinístico.

As etapas do desenvolvimento do código-fonte da aplicação são descritas de maneira sucinta no desenvolvimento.

2 REFERENCIAL TEÓRICO

Alfabeto e sentença, os quais podem ser descritos, respectivamente, como um conjunto finito de símbolos e uma sequência finita de símbolos de um alfabeto, permitem construir a definição de linguagem: um conjunto finito ou infinito de sentenças de um alfabeto.

Um conjunto finito de regras que geram sentenças quando aplicadas é o que chamamos de gramática. Gramáticas formais são classificadas em quatro tipos segundo a hierarquia de Chomsky, sendo esses: regulares, livres de contexto, sensíveis ao contexto e irrestritas.

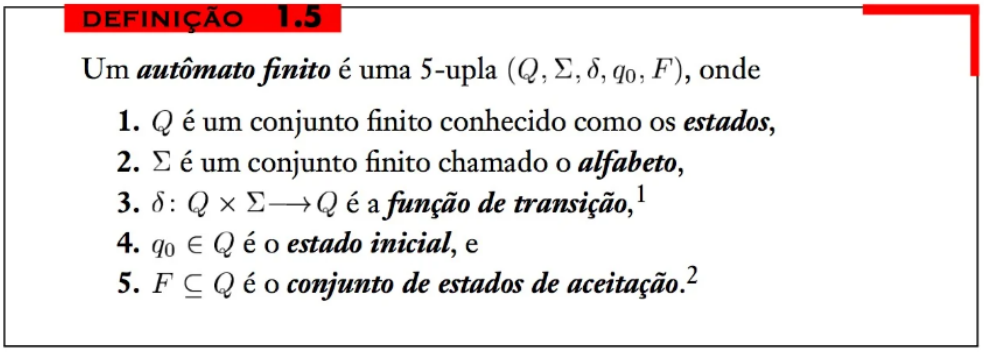
Gramáticas regulares geram linguagens regulares, que segundo Menezes (2011) são linguagens mais simples, o que permite desenvolver algoritmos de geração e reconhecimento de baixa complexidade e fácil implementação.

Dispositivos que fazem o reconhecimento para verificar se uma determinada sentença pertence a uma certa linguagem são chamados de reconhecedores de linguagens, sendo um exemplo os autômatos finitos que reconhecem as linguagens geradas por gramáticas regulares. Sua definição formal é apresentada na figura a seguir.

---

[1] Artigo elaborado com orientação do professor Braulio Adriano de Mello para a disciplina de Linguagens Formais e Autômatos, do curso de Ciência da Computação, na Universidade Federal da Fronteira Sul.

Figura 1 – Definição formal de autômato finito



Fonte: Sipser (2007)

Autômatos Finitos podem ser classificados em Determinísticos (AFD) e Não-Determinísticos (AFND), que se diferenciam na função de transição, sendo que essa ao receber um estado e símbolo como argumentos, para AFDs retorna apenas um único estado, enquanto para AFNDs pode retornar nenhum, um ou mais estados (Menezes, 2011).

Menezes (2011, p.85) apresenta o teorema "A classe dos autômatos finitos determinísticos é equivalente à classe dos autômatos finitos não determinísticos" e o prova por indução.

Por vezes é mais fácil descrever um AFND para uma linguagem do que descrever um AFD para essa linguagem, afirma Sipser (2007), tornando a equivalência entre AFND e AFD muito útil.

3 DESENVOLVIMENTO

Para o desenvolvimento da aplicação se optou pelo uso da linguagem php com a utilização de um banco de dados MySQL para armazenar as informações.

A aplicação pode ser dividida nas seguintes etapas: o tratamento do arquivo de entrada, a geração de um AFND e a aplicação de teorema da determinização para obter o AFD correspondente.

3.1 LEITURA E ARMAZENAMENTO DA ENTRADA

Para armazenamento dos dados temos a criação do banco e de três tabelas: alfabeto, estado e transição. O arquivo txt recebido na entrada é analisado em partes, primeiramente identificando se esta entrada é composta por tokens ou se é uma gramática para então armazenar os dados.

Operadores como "<", ">", "::=" e "|" são identificados na aplicação de modo a armazenar de maneira correta os dados de entrada de uma grámatica regular. Enquanto na análise de tokens, é importante identificar o último estado crido como final.

A análise de cada linha da entrada e a posterior gravação de seu conteúdo no banco já faz parte da criação do AFND.

3.2 CRIAÇÃO DO AFND

A identificação de qual é o tipo de entrada, tokens ou gramática, é o primeiro passo. Independente do tipo de entrada, o estado S já se encontra gravado no banco, pois ele é o único que pode ser compartilhado, os demais devem ser únicos. Então, ao se criar estados e

transições, são realizadas verificações para evitar que se crie um estado S repetido ou se sobrescreva seus dados.

Figura 2 – Armazenamento estado S

```
$newstate->setContent('S');
$newstate->setReference('S');
$bdstate->s_insert($newstate);
```

Fonte: Autoria própria

Para gramáticas, se analisam as regras, iniciando com o armazenamento do símbolo não terminal à esquerda do operador "::=" como estado de referência. Uma verificação é realizada para encontrar qual é o estado que será inserido na tabela de transição.

Então se analisa cada trecho separado pelo operador "|" individualmente, de modo a identificar as transições em cada produção, buscando verificar se existe épsilon produções e produções com apenas símbolos terminais. A Figura 3 apresenta trecho de código que separa as produções de uma regra e as dividem entre símbolos terminais e não terminais.

Os símbolos terminais são analisados posteriormente, e caso ainda não estejam cadastrados no banco, se realiza esta etapa. Para os trechos não terminais, é verificado se existem estados cadastrados que referenciem o estado encontrado, caso não exista, se cria um estado.

Para estados finais, ou seja, estados com apenas símbolos terminais ou "ε", é feita verificações e estes são adicionados ao vetor dos estados finais.

Dessa forma, uma regra é integrada ao banco com todos os símbolos, estados e transições gravadas para que se possa imprimir a tabela de transições do AFND gerado pela gramática.

Figura 3 – Divisão de produções para análise

```
for ($k=0; $k < count($arr); $k++) {
    if($arr[$k] != '<' && $arr[$k] != '>' && $arr[$k] != ' ' && $arr[$k] != '\n'){
        if(ord($arr[$k]) >= 65 && ord($arr[$k]) <= 90) {
            $s = $s.$arr[$k];
        }
        else {
            $a = $a.$arr[$k];
        }
    }
}
```

Fonte: Autoria própria

Entradas do tipo tokens, os símbolos são lidos individualmente, sendo sempre criado um estado, e da mesma maneira que entradas do tipo gramática, é feita busca para verificar a presença dos símbolos na tabela do alfabeto, e caso não estejam presentes, é feito seu armazenamento. O último estado criado é inserido ao vetor dos estados finais.

Para as transições, é verificado caso seja a primeira transição, para que se armazene em S, e nas demais tem-se o cuidado para evitar sobrescrever dados de S.

Figura 4 – Criação de transição do primeiro token

```
// Se for o primeiro simbolo do token
if($j == 0) {
    // Cria uma nova transição
    $newtransition->setAlphabet($array[$j]);
    $newtransition->setStartState('S');
    $newtransition->setEndState(chr($r));
    $bdtransition->t_insert($newtransition);
}
```

Fonte: Autoria própria

3.3 GERAÇÃO DE AFD A PARTIR DE AFND

A matriz gerada a partir da entrada que contêm o AFND é analisada para saber se existe conteúdo em cada posição, caso exista conteúdo maior que 1 o conteúdo é concatenado.

Figura 5 – Concatenação de elementos da matriz AFND

```php
// Verifica se o conteúdo tem tamanho maior que 1
if(strlen($array['S'][$alphabet->getContent()]) > 1) {
    $arr = explode(", ", $array['S'][$alphabet->getContent()]);
    for($a=0; $a<count($arr); $a++) {
        // Se não existe conteúdo na posição
        if(!isset($array2['S'][$alphabet->getContent()])) {
            $array2['S'][$alphabet->getContent()] = $arr[$a];
        }
        // Se existe conteúdo na posição, concatena
        else {
            $array2['S'][$alphabet->getContent()] = $array2['S'][$alphabet->getContent()].$arr[$a];
        }
    }
}
```

Fonte: Autoria própria

Para realizar a determinização, em uma nova matriz se insere o estado S e novamente se analisa os elementos, desta vez da matriz gerada com elementos concatenados. Quando existem transições para mais de um estado, se armazena um novo estado que concatena os dois estados, criando um estado. Para esses novos estados é preciso verificar os estados da transição, e gerar novas transição na matriz.

Dessa forma, a função de transição para certo estado e símbolo que anteriormente poderia retornar mais de um estado, retorna apenas um único estado.

Por fim se verifica os estados novos para identificar se algum possui estado final, e insere estes estados no vetor de estados finais.

4 CONSIDERAÇÕES FINAIS

Para o desenvolvimento da aplicação foi necessário entender de maneira clara os conteúdos apresentados em aula, principalmente linguagens, gramáticas regulares e autômatos finitos. A compreensão de como é feita a determinização de um AFND foi fundamental para que a aplicação tivesse o resultado esperado e construísse um AFD a partir de uma entrada de tokens ou gramática regular.

O projeto foi uma excelente oportunidade para solidificar os conceitos teóricos adquiridos ao longo do semestre.

REFERÊNCIAS

MENEZES, Paulo B. Linguagens Formais e Autômatos - V3 - UFRGS. [Digite o Local da Editora]: Grupo A, 2011. E-book. ISBN 9788577807994. Disponível em: https://app.minhabiblioteca.com.br/#/books/9788577807994/. Acesso em: 13 fev. 2023.

SIPSER, Michael. Introdução à Teoria da Computação: Trad. 2ª ed. norte-americana. [Digite o Local da Editora]: Cengage Learning Brasil, 2007. E-book. ISBN 9788522108862. Disponível em: https://app.minhabiblioteca.com.br/#/books/9788522108862/. Acesso em: 13 fev. 2023.

SOUSA, Carlos E B.; NASCIMENTO, Leonardo B G.; MARTINS, Rafael L.; et al. Linguagens Formais e Autômatos. [Digite o Local da Editora]: Grupo A, 2021. E-book. ISBN 9786556901138. Disponível em: https://app.minhabiblioteca.com.br/#/books/9786556901138/. Acesso em: 13 fev. 2023.