Construção, Determinização e Minimização de Autômato Finito a partir de Gramáticas Regulares

Jackson Henrique Hochscheidt, Matheus Henrique Trichez, Patric Dalcin Venturini

***Resumo*—No presente artigo discutimos uma pequena introdução e algumas noções básicas acerca do estudo de Gramáticas Regulares e Autômatos Finitos. Ainda, buscamos explanar sobre o processo de Construção, Determinização e Minimização de um Autômato Finito para Gramáticas Regulares. Também explicamos como a nossa implementação realiza essa funções, e de que maneira interpreta e modela as gramáticas e autômatos.**

***Palavras-chave*—Autômatos Finitos, Linguagens Formais, Determinização, Minimização, Gramáticas Regulares.**

I. Introdução

Formulada originalmente na década de 1950, a *teoria das linguagens* tinha como proposta explorar o campo das linguagens naturais. No entanto, ela acabou se mostrando de grande uso no estudo das ditas *linguagens artificiais*, tais como as linguagens originárias da computação e informática [1].

Esse artigo traz um software que opera sobre esse contexto de *linguagens artificiais* na parte de reconhecimento léxico destas. Mais precisamente, atua na criação, determinização e minimização de um Autômato Finito sobre a gramática de uma linguagem regular.

Um autômato finito é, além de um simples grafo que descreve uma máquina de estados, um formalismo matemático que representa um analisador léxico. Ou seja, uma máquina de estados usada no reconhecimento de *sentenças* de uma linguagem regular. Sentenças estas alcançáveis através do *fechamento transitivo* da máquina gerativa — de símbolos produtivos de tal linguagem — que é a gramática regular.

Por sua vez, o processo de determinização do autômato finito recebe como entrada esse conjunto de sentenças alcançáveis da linguagem e onde houver indeterminização, ou seja, onde um não-terminal da linguagem alcançar mais de um não-terminal através de um único terminal, é necessário fazer a determinização, criando assim novos estados de transição e determinizando o autômato.

Finalmente, a minimização de uma linguagem tem por objetivo remover do autômato os estados que, após a determinização, tornaram-se inalcançáveis; remover os estados mortos, ou seja, que nunca levam a um estado final; e só então aplicar a minimização por "Classe de Equivalência". O processo de minimização através da Classe de Equivalência separa os estados em finais e não-finais e, entre os grupos de semelhantes, procura aqueles que, para cada terminal, eles levem para uma mesma classe de estado. Explicando melhor, levando em consideração os dois estados que estão sendo analisados, se um mesmo terminal levar para um estado qualquer final ou um não-final, não precisando ser o mesmo estado nos dois casos, eles são equivalentes. Caso os terminais levem para um estado final e um não-final, os dois não são equivalentes.

II. Referencial Teórico

Ao entrarmos nesse contexto, existem alguns conceitos que precisam ser definidos para evitar a má interpretação do seu significado. Isso acontece pois além de serem termos técnicos neste contexto, são palavras comuns e recorrentes no cotidiano das *linguagens naturais*. As definições que seguem são do livro de Paulo Blauth Menezes [1].

- Alfabeto:
  "Um *alfabeto* é um conjunto finito de símbolos ou caracteres." Assim, consideramos que:
  - Um conjunto infinito *não* é um alfabeto
  - O conjunto vazio é um alfabeto
- Linguagem Formal:
  Uma linguagem formal ou apenas *linguagem* sobre um alfabeto é um conjunto de palavras que pertencem ao alfabeto da linguagem.
- Palavra:
  "Uma *palavra*, *cadeia de caracteres* ou *sentença* sobre um alfabeto, é uma sequência finita de símbolos do alfabeto concatenados."
  Portanto, admite-se que uma cadeia sem símbolos seja uma palavra válida e utilizamos o símbolo ε (*Epsilon*) para representar o que chamamos de uma *cadeia vazia* ou *palavra vazia*.
- Prefixo:
  "Qualquer sentença inicial dos símbolos da palavra".
- Sufixo:
  "Qualquer sentença final dos símbolos da palavra".
- Subpalavra:
  "Uma sequência qualquer de símbolos contíguos da palavra"

As Gramáticas Regulares são, assim como as demais gramáticas, uma máquina gerativa referente a uma linguagem de classe equivalente a sua. Assim, uma gramática regular deve gerar sentenças que pertencem ao conjunto de sentenças de uma linguagem regular. Representada por um finito conjunto de regras que quando aplicadas geram sentenças da linguagem, a gramática deve ser capaz de gerar todas as sentenças de uma linguagem através do se conjunto de regras.

Através do mapeamento das transições de estados descritos pelas regras de uma gramática, podemos gerar o que chamamos de Autômato Finito. Modelado por um formalismo matemático, o autômato trata de reconhecer cada símbolo de uma palavra levando em consideração o estado atual, e para

onde essa combinação de símbolo/estado podem o levar. Ou seja, a combinação símbolo/estado é quem determina para qual será o próximo estado. Podendo responder assim, de forma a *aceitar* uma palavra como uma sentença que pertence ao conjunto de sentenças da *linguagem* (caso as transições sucessivas chegarem a um estado final válido), ou *rejeitar* a palavra em análise caso encontrar, através das transições sucessivas, um estado de erro.

Entretanto, ao ser gerado um autômato como mencionado no parágrafo anterior, o resultado será na maioria das vezes, um Autômato Finito *Não-Determinístico* (AFND). O que significa dizer que em alguns casos, uma mesma combinação de símbolo/estado deste autômato, podem levar a sequência de transições a mais de um estado. Podendo ser evitado se aplicado um processo de determinização, que transforma um Autômato Finito Não-Determinístico e um Autômato Finito *Determinístico* (AFD) equivalente. E este, por sua, vez alcança um, e apenas um, estado para cada combinação símbolo/estado.

Porém, um AFD pode ter, nos piores casos, um número de até 2^n estados enquanto o seu AFND equivalente teria apenas n estados [2].

Uma alternativa para o problema do elevado número de estados, é o processo de *Minimização* do Autômato Finito Determinístico. Esse processo consiste em remover os estados ditos *inalcançáveis*, *mortos* ou em *classe de equivalência*. Em nosso caso em específico trataremos apenas os dois primeiros itens:

- *Inalcançáveis* são ditos os estados que não podem ser alcançáveis pelo estado inicial, direta ou indiretamente.
- São considerados *Mortos* os estados os estados que não alcançam pelo menos um estado final, direta ou indiretamente.

III. Desenvolvimento

Primeiro de tudo, para o programa começar a trabalhar em cima do autômato, é feita a leitura do arquivo que contém o mesmo e que será determinizado. O Arquivo precisa estar formatado da seguinte forma para ser aceito pelo programa:

$$< A >::= a < B > |\&|a;$$

Dessa forma, os não-terminais sempre ficam entre '$<$' e '$>$'. O estado atual sempre vem seguido dos símbolos de identificação '::=' e precedido de um terminal com um não-terminal, um terminal sozinho ou um símbolo de palavra vazia '&', terminando a linha com um ';', como no exemplo a seguir.

Lendo linha a linha do arquivo, é armazenado cada não-terminal do lado esquerdo do autômato como um objeto do autômato, juntamente com um conjunto de terminais e seus respectivos terminais. Ao mesmo tempo, é feito também uma tabela com os terminais que aparecem, para a questão de controle de como será montada a tabela do autômato.

Com isso feito, é gerado um arquivo de saída com o autômato finito ainda não determinado, podendo contar mais de um não-terminal atribuído a um terminal.

Então, o código faz o processo de determinização do autômato, verificando linha a linha do autômato quais terminais

```
<S> ::= a<B> | a<A> | b<B> | c<D>;
<A> ::= a<A> | b<B> | b<C>;
<B> ::= b<B> | b<C> | &;
<C> ::= a<B> | &;
<D> ::= d<E>;
<E> ::= d<D>;
```

Fig. 1. Autômato Finito utilizado para exemplificação.

	a	b	c	d
S	B, A	B	D	X
A	A	B, C	X	X
*B	X	B, C	X	X
*C	B	X	X	X
D	X	X	X	E
E	X	X	X	D
X	X	X	X	X

Fig. 2. Autômato resultante do processo de leitura.

possuem mais de um não-terminal, junta os mesmos em um só estado, organizando por ordem alfabética, e cria um estado novo com a soma desses estados. Para cada vez que uma alteração acontece no autômato, o código é rodado de novo, pois novos indeterminações podem surgir na criação dos novos estados.

	a	b	c	d
S	[AB]	B	D	X
A	A	[BC]	X	X
*B	X	[BC]	X	X
*C	B	X	X	X
D	X	X	X	E
E	X	X	X	D
*[AB]	A	[BC]	X	X
*[BC]	B	[BC]	X	X

Fig. 3. Autômato determinizado a partir do autômato dado.

A seguir, para cada linha do autômato determinizado, começando pelo estado inicial, é observado quais outros estados são chamados pelos terminais e adicionados aos estados alcançáveis. Sempre que um novo não-terminal é adicionado, ele é verificado para ver se outros não-terminais passam a ser alcançáveis pelo autômato, eliminando assim aqueles que nunca são alcançados pelos terminais.

Por fim, encerrando o processo de determinização, vem a eliminação dos estados mortos. Para o processo de eliminação desses estados, é armazenado um vetor com todos os estados finais do autômato. Para cada linha do autômato, então, a analisado quais alcançam um estado final e adicionado eles

	a	b	c	d
S	[AB]	B	D	X
A	A	[BC]	X	X
*B	X	[BC]	X	X
D	X	X	X	E
E	X	X	X	D
*[AB]	A	[BC]	X	X
*[BC]	B	[BC]	X	X

Fig. 4. Autômato sem os devidos estados inalcançáveis.

para os “estados finais”, pois, se eles alcançam um estado final, um estado que alcance ele leva a um estado final. Isso é feito sempre que acontece um acréscimo nos novos “estados finais” e, assim que é terminado esse processo, os estados mortos são removidos do autômato.

	a	b	c	d
S	[AB]	B	D	X
A	A	[BC]	X	X
*B	X	[BC]	X	X
*[AB]	A	[BC]	X	X
*[BC]	B	[BC]	X	X
X	X	X	X	X

Fig. 5. Autômato final, determinizado e sem os estados mortos e inalcançáveis.

IV. Conclusão

O trabalho como um todo serviu para uma maior aprendizagem quanto aos autômatos em si, assim como auxiliou a reforçar a compreensão do processo de minimização dos mesmos e sua complexidade computacional, visto que cada alteração ocorrida no processo gera um nova revisão inteira do que foi gerado. Ajudou também a compreender o passo a passo da determinização de um autômato finito, com a remoção dos estados inalcançáveis e mortos, assim como a importância dessas etapas do processo.

REFERÊNCIAS

[1] Paulo Blauth Menezes. *Linguagens formais e autômatos*.

[2] JE Hopcroft and JD Ullman. Introdução à teoria de autômatos, linguagens e computação. segunda edição, ed. *Campus\Elsevier*, 2003.

[3] Kenneth C Louden. *Compiladores-Princípios e Práticas*. Cengage Learning Editores, 2004.

[4] Ravi Sethi, Jeffrey D Ullman, and monica S. Lam. *Compiladores: princípios, técnicas e ferramentas*. Pearson Addison Wesley, 2008.