UNIP

UNIVERSIDADE PAULISTA

Aspectos Teóricos da Computação

Autor: Prof. Hugo Gava Insua
Colaboradoras: Profa. Vanessa Santos Lessa
Profa. Larissa Rodrigues Damiani

Professor conteudista: Hugo Gava Insua

Mestre em Engenharia de Produção pela Universidade Paulista (UNIP), especialista em Ensino de Matemática pela Universidade Cruzeiro do Sul (2017), graduado em Matemática pela Universidade Metodista de São Paulo (2000). Atua como professor do curso de Ciência da Computação da UNIP, nas disciplinas *Matemática Discreta*, *Cálculo para Computação*, *Álgebra Linear*, *Estatística*, *Computação Gráfica*, *Processamento Digital de Imagem*, *Linguagens Formais e Autômatos* e *Aspectos Teóricos da Computação*. Atuou como professor na rede oficial e privada de ensino do estado de São Paulo.

Dados Internacionais de Catalogação na Publicação (CIP)

I59a Insua, Hugo Gava.

Aspectos Teóricos da Computação / Hugo Gava Insua. – São Paulo: Editora Sol, 2023.

164 p., il.

Nota: este volume está publicado nos Cadernos de Estudos e Pesquisas da UNIP, Série Didática, ISSN 1517-9230.

1. Linguagem. 2. Computabilidade. 3. Complexidade. I. Título.

CDU 681.3

U518.65 – 23

Sumário

Aspectos Teóricos da Computação

APRESENTAÇÃO

Caros alunos, é com enorme satisfação que apresento a disciplina tema deste livro-texto. Nas linhas que seguem, mostrarei que esta é uma disciplina fundamental, pois estuda os fundamentos teóricos da ciência da computação. Sua importância se dá por ajudar a entender as limitações dos dispositivos de computação e a desenvolver algoritmos eficientes para resolver problemas computacionais. É uma área muito rica e em constante evolução. Ela é essencial para a compreensão dos limites e possibilidades da computação, e é usada em várias outras áreas, como criptografia, inteligência artificial, processamento de linguagem natural e computação quântica. A disciplina tem como objetivo entender quais problemas são solucionáveis por computadores e quais não o são, bem como o que é possível e o que é impossível um computador realizar.

Alguns dos temas a que essa disciplina se dedica incluem:

- teoria dos autômatos e linguagens formais;
- computabilidade e complexidade;
- lógica matemática e prova de teoremas;
- algoritmos e estruturas de dados.

Neste livro-texto nos debruçaremos sobre os dois primeiros temas, visto que os demais são também objeto de outras disciplinas do nosso curso. Autômatos, computabilidade e complexidade são conceitos fundamentais da teoria da computação, e estudam os limites e possibilidades da computação.

Os autômatos são modelos teóricos de máquinas que manipulam símbolos. Eles são usados para descrever a capacidade de processamento de diferentes modelos de computação. Existem vários tipos de autômatos, como os finitos determinísticos e não determinísticos, os de pilha e máquinas de Turing. Cada tipo de autômato tem diferentes capacidades de processamento e é adequado para resolver diferentes tipos de problemas.

A computabilidade, por sua vez, se refere à capacidade de resolver problemas por meio de algoritmos, ou seja, por meio de um conjunto finito de instruções bem definidas. A teoria da computabilidade investiga os limites da computação e quais problemas podem ser resolvidos por meio de algoritmos. Um problema é considerado computável se um algoritmo que o resolve pode ser expresso em termos de um autômato. Por outro lado, um problema é considerado intratável se não houver algoritmo eficiente para resolvê-lo.

Já a complexidade se refere à análise do desempenho dos algoritmos, ou seja, o quanto de recursos computacionais é necessário para resolver um problema. A complexidade é medida em termos de tempo e espaço requeridos para executar um algoritmo. Sua importância se dá na medida em que fornece uma maneira de classificar problemas de acordo com sua dificuldade. Problemas com complexidade de tempo polinomial são considerados fáceis, enquanto problemas com complexidade exponencial

são considerados difíceis. A classe de problemas que podem ser resolvidos em tempo polinomial é conhecida como P, enquanto a classe de problemas que são pelo menos tão difíceis quanto os problemas NP-completos é conhecida como NP.

INTRODUÇÃO

O presente livro-texto foi dividido em duas unidades com quatro títulos cada.

A primeira unidade contempla o estudo dos fundamentos matemáticos da formação das linguagens, base para o correto entendimento da disciplina, como conjuntos, sequências, relações, funções, grafos, cadeias e linguagens. Estudaremos também os diversos tipos de autômatos, como os autômatos finitos determinísticos e não determinísticos, autômatos de pilha e máquina de Turing. A partir desta última, iniciaremos o estudo da computabilidade.

Na segunda unidade aprofundaremos os conceitos de computabilidade através do estudo dos problemas que são ou não decidíveis. Estudaremos também a complexidade computacional, que trata da quantidade de recursos, como tempo e espaço, necessários para resolver um problema computacionalmente. A complexidade é medida em notações como O-grande e Ω-grande, que descrevem o limite superior e inferior do tempo ou espaço necessários para resolver um problema.

Bons estudos.

Unidade I

1 PRESSUPOSTOS MATEMÁTICOS NECESSÁRIOS

1.1 Conjuntos

Um conjunto é uma coleção ou reunião de objetos, elementos ou valores distintos que são considerados uma única entidade. Em matemática, os conjuntos são usados para agrupar elementos com características comuns ou relacionadas.

A ordem dos elementos em um conjunto não tem impacto em sua definição, e o número de vezes que um mesmo elemento aparece em um conjunto não é relevante. Portanto, podemos afirmar que: A = {maçã, banana, laranja} = {banana, laranja, maçã} = {maçã, maçã, banana, banana, laranja, laranja}.

Exemplo: B = {papagaio, arara, tucano}. B é uma coleção de três pássaros.

Exemplo: C = {0, 523}. C é uma coleção de dois números reais.

Exemplo: D = {÷, ≥, %, +}. D é uma coleção de quatro símbolos matemáticos.

Chamamos de pertinência a relação de um elemento qualquer pertencer ou não a determinado conjunto. Considerando o conjunto D anterior, dizemos que + pertence a D e denotamos por + ∈ D. Para indicar que arara não é pertence a D, escreve-se arara ∉ D.

Os conjuntos podem ser classificados segundo a quantidade de elementos que possuem. Assim:

- **Finitos**: são conjuntos cujos elementos podem ser enumerados. Exemplo: seja o conjunto A = {a, e, i, o, u}. A é um conjunto finito, representado pela enumeração de seus elementos a, e, i, o, u.

- **Infinitos**: os conjuntos numéricos (naturais, inteiros, racionais, irracionais, reais e complexos) são exemplos de conjuntos infinitos. Exemplo: N = {0, 1, 2, 3, 4, ...}, Z = {..., -3, -2, -1, 0, 1, 2, 3, ...}.

- **Vazio**: é o conjunto que não possui elementos. Há duas formas para representá-lo: { } ou Ø.

Os conjuntos também podem ser descritos com base em suas propriedades, como exemplificado a seguir. Considere o conjunto E definido como {y | y ∈ R e y > 0}; nesse caso, E representa o conjunto de números reais positivos. Além disso, o conjunto dos pontos no plano cartesiano pode ser representado como {P | P = (x, y) e x + y = 5}, onde P denota pontos cujas coordenadas x e y somam 5.

Chamamos de relação de inclusão entre conjuntos a maneira de descrever como um conjunto pode estar contido em outro, ou seja, quando um conjunto é subconjunto de outro. Essa relação é fundamental na teoria dos conjuntos e é representada por dois símbolos principais: $\subseteq$ e $\subset$, que denotam diferentes tipos de inclusão.

O símbolo $\subseteq$ é usado quando estamos indicando que um conjunto é subconjunto do outro. Formalmente, se $A \subseteq B$ (A está contido em B), isso significa que todos os elementos de A também são elementos de B, e A pode ser igual a B. Isso implica que qualquer conjunto é subconjunto de si mesmo.

O símbolo $\subset$ representa uma inclusão própria, o que significa que um conjunto é um subconjunto estrito do outro. Se $A \subset B$ (A está contido propriamente em B ou A é subconjunto próprio de B), isso implica que todos os elementos de A estão em B, mas A não é igual a B. Em outras palavras, A está contido em B, mas B tem pelo menos um elemento que não está em A.

Exemplo: sejam os conjuntos $A = \{0, 1, 3, 5, 15, 25, 30, 40\}$ e $B = \{0, 1, 15, 25\}$, é possível estabelecer que $B \subseteq A$, $B \subset A$, $A \subseteq A$, $B \subseteq B$.

Dois conjuntos são iguais se, e somente se, $A \subseteq B$ e $B \subseteq A$.

O conjunto vazio, por definição, é um conjunto que não contém nenhum elemento. Devido a essa característica, é considerado um subconjunto de qualquer conjunto. Essa propriedade é uma consequência direta das definições de subconjuntos e da lógica da teoria dos conjuntos.

A definição de subconjunto diz que um conjunto A é subconjunto de um conjunto B se, e somente se, todos os elementos de A também são elementos de B. Como o conjunto vazio não tem elementos, não há elementos em A que não estejam em B, independentemente de qual seja o conjunto B. Em outras palavras, não importa qual conjunto B você escolha, não há nenhum elemento em A que não esteja em B, porque A não possui elementos. Portanto, o conjunto vazio atende à condição de ser subconjunto de qualquer conjunto.

Na teoria dos conjuntos, há diversas operações que, quando aplicadas a dois conjuntos A e B quaisquer, formam um terceiro conjunto. As quatro principais operações são:

- **União**: sejam A e B quaisquer dois conjuntos. A união de A e B, denotada como $A \cup B$, consiste em todos os elementos que pertencem a A ou a B, ou seja, $A \cup B = \{x \mid x \in A \text{ ou } x \in B\}$. Exemplo: considerando A = {maçã, banana, laranja} e B = {banana, uva, kiwi}, a união de A e B é {maçã, banana, laranja, uva, kiwi}.

- **Interseção**: sejam A e B quaisquer dois conjuntos. A interseção de A e B, denotada como $A \cap B$, contém todos os elementos que pertencem tanto a A quanto a B, ou seja, $A \cap B = \{x \mid x \in A \text{ e } x \in B\}$. Exemplo: considerando A = {maçã, banana, laranja} e B = {banana, uva, kiwi}, a interseção de A e B é $A \cap B$ = {banana}.

- **Diferença**: sejam A e B quaisquer dois conjuntos. A diferença de A por B, denotada como A - B, consiste em todos os elementos que pertencem a A, mas não a B, ou seja, $A - B = \{x \mid x \in A \text{ e } x \notin B\}$. Exemplo: considerando A = {maçã, banana, laranja} e B = {banana, uva, kiwi}, a diferença A - B = {maçã, laranja}, e a diferença B - A = {uva, kiwi}.

- **Complementação**: a complementação de um conjunto A, denotada como A', é o conjunto que contém todos os elementos de um conjunto universo U que não pertencem a A, ou seja, $A' = \{x \mid x \in U \text{ e } x \notin A\}$.

Suponha que A seja o conjunto de números pares e U seja o conjunto de números inteiros. Nesse caso, a complementação de A, representada como A', seria o conjunto de números inteiros que não são pares, ou seja, os números ímpares.

A = {..., -6, -4, -2, 0, 2, 4, 6, ...} U = {..., -3, -2, -1, 0, 1, 2, 3, ...} A' = {..., -3, -1, 1, 3, ...}

1.2 Sequências e uplas

Sequências ou listas são coleções ordenadas de objetos representadas entre parênteses. Dessa forma, a lista (a, b, c) é diferente da lista (c, b, a).

Podemos classificar as sequências como finitas (uplas) ou infinitas. Uma sequência com n elementos é denominada n-upla. Assim, a sequência (a, b, c) é uma 3-upla.

Tanto listas como conjuntos podem ser elementos de outras listas ou outros conjuntos. Por exemplo, temos o conjunto potência, ou conjunto das partes, que é o conjunto de subconjuntos de determinado conjunto, e temos o produto cartesiano, que é o conjunto de listas de pares ordenados entre dois conjuntos.

- **Conjunto potência**: seja A um conjunto. Tem-se que o conjunto potência é definido como:

$2^{|A|} = \{S \mid S \subseteq A\}$, onde |A| é a quantidade de elementos de A, ou, simplesmente, cardinalidade de A.

Exemplo: seja A = {1, 2, 3}, tem-se que:

$2^{|3|} = \{\emptyset, \{1\}, \{2\}, \{3\}, \{1, 2\}, \{1, 3\}, \{2, 3\}, \{1, 2, 3\}\}$

Observe que o conjunto potência de A é o conjunto de todos os subconjuntos de A.

- **Produto cartesiano**: sejam A e B dois conjuntos. O produto cartesiano A x B define-se como:

$A \times B = \{(a, b) \mid a \in^{A \text{ e } b} \in^{B}\}$

O elemento do produto cartesiano (a, b) é uma 2-upla denominada par ordenado, ou seja, a ordem em que os componentes a e b se apresentam é relevante.

Exemplo: sejam A = {x, y} e B = {1, 2, 3}. Determine o produto cartesiano entre:

A x B = {(x, 1), (x, 2), (x, 3), (y, 1), (y, 2), (y, 3)}

B x A = {(1, x), (1, y), (2, x), (2, y), (3, x), (3, y)}

A x A = {(x, x), (x, y), (y, x), (y, y)}

B x B = {(1, 1), (1, 2), (1, 3), (2, 1), (2, 2), (2, 3), (3, 1), (3, 2), (3, 3)}

1.3 Relações e funções

Em matemática, uma relação é um conjunto de pares ordenados que relacionam elementos de dois conjuntos. Formalmente, uma relação R entre dois conjuntos A e B é definida como um subconjunto do produto cartesiano A x B, que é o conjunto de todos os pares ordenados (a, b), onde a é um elemento de A e b é um elemento de B.

Por exemplo, considere os conjuntos A = {1, 2, 3} e B = {a, b, c}. A relação R = {(1, a), (2, b), (2, c), (3, a)} é um subconjunto do produto cartesiano A x B, pois todos os seus elementos são pares ordenados cujos primeiros elementos pertencem a A e os segundos elementos pertencem a B. Observe que os elementos (2, b) e (2, c) mostram que o elemento 2 de A está relacionado a ambos os elementos b e c de B.

O conceito de função é uma das partes mais importantes da matemática, tendo aplicações não só nessa área, mas nas ciências de modo geral. A funções sempre estão presentes quando relacionamos duas grandezas variáveis.

Veja o exemplo:

Tabela 1

Litros de gasolina gasta (L)	Distância percorrida (km)
1	10
2	20
3	30

Observe que a distância percorrida em quilômetros é dada em **função** do consumo de gasolina em litros, ou seja, a distância percorrida depende dos litros gastos de combustível.

Distância percorrida = 10 vezes a quantidade de gasolina gasta, ou **km = 10 · L**. A esta fórmula chamamos de **lei da função**.

Considere dois conjuntos A e B não vazios. Denominamos função de A em B e representamos por F: $A \rightarrow B$ a regra que associa cada elemento $x \in A$ a um único elemento $y \in B$.

Dessa forma, podemos entender funções como um sistema de partida e chegada, ou seja, a função toma um valor de partida (sobre os elementos de A) que leva a um valor de chegada (sobre os elementos de B). Jamais, em uma função, um valor de partida leva a mais de um valor no conjunto de chegada.

Na figura a seguir tem-se um esquema, através de diagramas de Venn, que representa uma regra f que associa elementos do conjunto de partida A a elementos no conjunto de chegada B.

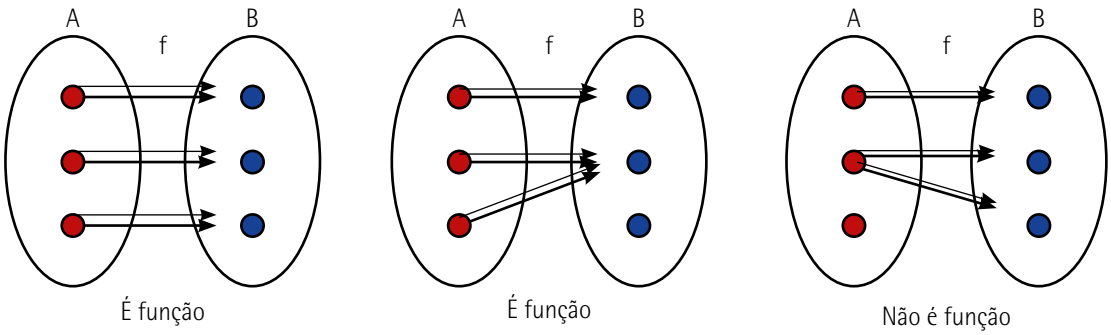

Figura 1 – Regras f: associam elementos de A a elementos de B através de diagramas de Venn

Pela figura, é possível observar que só é classificada como função a regra f que associa cada elemento do conjunto A a um único elemento do conjunto B. Dessa forma, suponha um valor de chegada y para um valor de partida x. Podemos representar essa função como f(x) = y.

Ao conjunto de partida A denominamos domínio, ao conjunto de chegada B denominamos contradomínio, e aos elementos do contradomínio, obtidos a partir da regra f, denominamos imagem.

Veja o exemplo: dados os conjuntos A = {1, 2, 3, 4} e B = {2, 3, 4, 5, 6, 7, 8}. A função F: A $\rightarrow$ B que determina a relação entre os elementos de A e B é x $\rightarrow$ x + 1. Sendo assim, f(x) = x +1; em outras palavras, cada x do conjunto A é transformado em x + 1 no conjunto B.

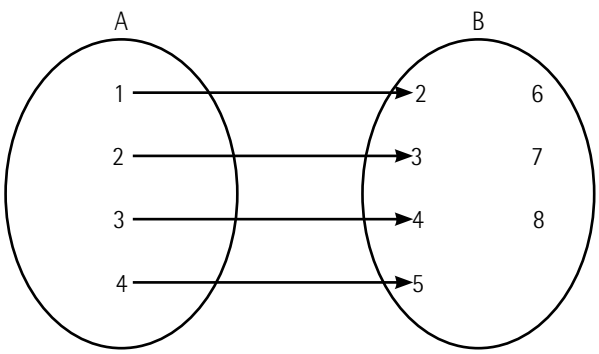

Figura 2 – Domínio, contradomínio e imagem

1.4 Grafos

Um grafo G(N,A) é definido pelo par de conjuntos N e A, onde:

N – conjunto não vazio: os nodos ou vértices do grafo;

A – conjunto de pares ordenados a = (n, m), n e m $\in$ N: as arestas do grafo.

Vejamos a seguir um exemplo. Seja o grafo G(N,A) dado por:

N = {a | a é uma indústria}

A = {(n, m) | < n é concorrente de m >}

Perceba que o conjunto N representa todas as indústrias, e o conjunto A todos os concorrentes entre si. Portanto, podemos representar inúmeros grafos que relacionam indústrias que são concorrentes entre si. Vejamos o grafo G_1, que é um elemento dessas inúmeras indústrias que são concorrentes entre si:

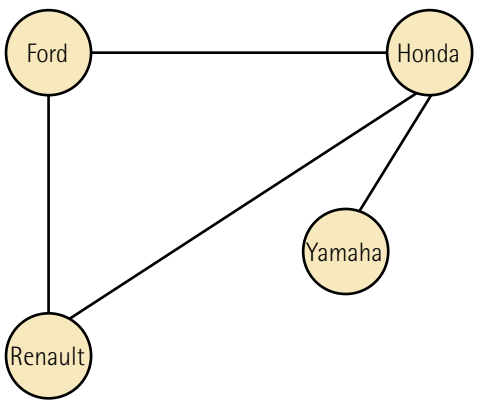

Figura 3 – Grafo G_1

V = {Ford, Honda, Renault, Yamaha}

A = {(Ford, Honda), (Honda, Ford), (Honda, Renault), (Renault, Honda), (Honda, Yamaha), (Yamaha, Honda), (Ford, Renault), (Renault, Ford)}

Observe que Ford e Renault são indústrias de automóveis; Yamaha fabrica motocicletas; e Honda é indústria de automóveis e motocicletas.

Em G_1, consideramos que a relação <n é concorrente de m> é simétrica. Assim, se <n é concorrente de m> então <m é concorrente de n>. Portanto, as arestas que ligam os vértices não possuem qualquer orientação.

Digrafo (grafo orientado)

É o grafo cujas arestas têm uma direção específica, indicando a relação entre dois vértices. Por exemplo, um grafo orientado pode ser usado para representar um sistema de trânsito, em que as ruas são os vértices e as arestas representam as direções permitidas do tráfego. Vejamos uma outra aplicação.

Considere o grafo G(N,A) definido por:

V = {r | r é funcionário da empresa E}

A = {(n, m) | < n é chefe de m >}

O conjunto V é o conjunto de todos os funcionários da empresa E. O conjunto A representa todos os chefes e seus subordinados.

O grafo G_2 representa a relação chefe-subordinado:

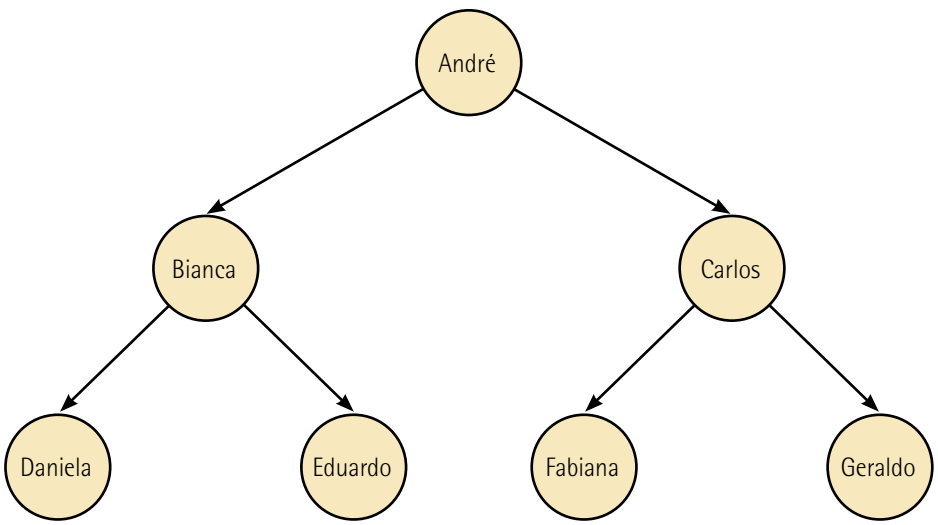

Figura 4 – Grafo G_2

V = {André, Bianca, Carlos, Daniela, Eduardo, Fabiana, Geraldo}

A = {(André, Bianca), (André, Carlos), (Bianca, Daniela), (Bianca, Eduardo), (Carlos, Fabiana), (Carlos, Geraldo)}

A relação representada pelo conjunto A é dita não simétrica, pois se <n é chefe de m>, não pode ocorrer o contrário, ou seja, <m é chefe de n>. Veja, portanto, que os pares ordenados de A têm uma orientação, acarretando que os sentidos das arestas ou arcos de G_2 não podem ser alterados.

O grafo anterior é um grafo orientado (ou digrafo), sendo que as conexões entre os vértices são chamadas de arcos.

Ordem

A ordem de um grafo é o número de vértices (ou nodos) que ele contém. Em outras palavras, a ordem é o tamanho (cardinalidade) do conjunto de vértices do grafo. Por exemplo:

ordem(G_1) = 4

ordem(G_2) = 7

A ordem de um grafo é uma medida importante, pois ela pode influenciar várias de suas propriedades e características. Por exemplo, a complexidade de algoritmos e problemas relacionados a grafos muitas vezes dependem de sua ordem.

Adjacência

A adjacência em um grafo se refere à relação de conexão entre vértices. Dois vértices em um grafo são considerados adjacentes se estiverem ligados por uma aresta (ou arco). Assim, dois vértices são adjacentes se existe uma aresta que conecta um ao outro.

A adjacência em um grafo é uma medida importante, pois ela descreve a conectividade do grafo e influencia muitas de suas propriedades e características. Por exemplo, uma adjacência pode ser usada para determinar o grau de um vértice (ou seja, o número de arestas que estão conectadas a um vértice), que é uma medida essencial de centralidade em um grafo.

Deve-se notar que a adjacência é uma relação simétrica em grafos não direcionados, ou seja, se um vértice n está conectado a um vértice m, então o vértice m também está conectado ao vértice n. Por exemplo, em G_1, são adjacentes os vértices Ford e Honda. Em grafos direcionados, no entanto, a adjacência pode ser assimétrica, ou seja, um vértice pode estar conectado a outro, mas não necessariamente o contrário. Dessa forma, estabelecemos os conceitos de:

- **Sucessor**: dados dois vértices quaisquer n e m, o vértice m é sucessor de n se existe uma aresta que parte de n e chega a m. Neste caso, percebemos que no grafo G_2 Bianca e Carlos são sucessores de André.

- **Antecessor**: dados dois vértices quaisquer n e m, o vértice n é antecessor de m se existe uma aresta que parte de n e chega a m. Neste caso, percebemos que em G_2 Bianca é antecessora de Daniela e Eduardo.

Lembrete

A adjacência em um grafo é uma medida importante, pois ela descreve a conectividade do grafo e pode ser usada para determinar o grau de um vértice.

Grau

O grau de um vértice em um grafo é o número de arestas que incidem sobre esse vértice, ou seja, é o número de arestas que estão conectadas a ele. Em G_1, por exemplo, os vértices Ford e Renault são grau 2, Honda é grau 3 e Yamaha, grau 1.

Em um grafo não direcionado, a soma dos graus de todos os vértices é igual ao dobro do número de arestas do grafo (porque cada aresta é contada duas vezes, uma para cada vértice que ela conecta). Esta propriedade é conhecida como o teorema do aperto de mão (ou *handshaking lemma*, em inglês).

Além disso, os vértices com grau mais alto tendem a ter uma maior centralidade em um grafo, o que significa que eles são mais importantes na estrutura geral do grafo e podem ser considerados pontos críticos na rede.

Em se tratando de digrafos, dizemos:

- **Grau de emissão**: quantidade de arestas que partem de um vértice. Em G_2, o grau de emissão do vértice Carlos é 2.

- **Grau de recepção**: quantidade de arestas que chegam a um vértice. Em G_2, o grau de recepção de todos os vértices é 1.

Fonte

Um vértice em um grafo é considerado uma fonte se ele não tem arestas direcionadas para ele (grau de recepção = 0). Assim, uma fonte é um vértice que só emite arestas, mas não recebe arestas de nenhum outro vértice no grafo.

A identificação de fontes em um grafo pode ser útil em vários contextos. Por exemplo, em um grafo que representa um fluxo de informações ou de recursos, as fontes podem indicar as origens de dados ou recursos que precisam ser processados ou distribuídos. Em um grafo de dependências de tarefas, as fontes podem representar ações que podem ser executadas sem depender de outras. Em G_2, o vértice André é uma fonte, pois seu grau é 0, ou seja, não há arestas chegando nele.

Sumidouro

Um vértice de um grafo é considerado um sumidouro se ele não tiver nenhuma aresta saindo dele (grau de emissão = 0), ou seja, um sumidouro é um vértice de um grafo direcionado que é o destino de um fluxo de informação ou de um processo.

Como exemplo, pode-se citar uma rede de transporte de produtos. Um sumidouro pode ser o depósito final onde os produtos são armazenados, de onde não sai mais nenhuma aresta para continuar a transportar os produtos.

Assim como as fontes, os sumidouros são importantes na análise de redes de fluxo e na identificação de falhas em sistemas complexos. Por exemplo, em um sistema de tubulação de água, um sumidouro pode ser um vazamento que está causando desperdício de água, e a identificação desse sumidouro pode ajudar a reduzir o desperdício e aumentar a eficiência do sistema.

Em G_2, os vértices Daniela, Eduardo, Fabiana e Geraldo são sumidouros.

Laço

Na teoria dos grafos, um laço é uma aresta que conecta um vértice a ele mesmo. Assim, um laço é uma aresta que tem o mesmo vértice como origem e destino. Veja um exemplo de um laço no vértice B do grafo G_3 da figura a seguir.

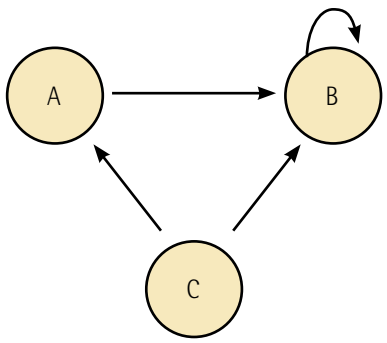

Figura 5 – Grafo G_3: exemplo de laço no vértice B

Laços são importantes na teoria dos grafos, pois um laço aumenta o grau de um vértice em 1, visto que ele é contado duas vezes: uma vez como grau de emissão e outra vez como grau de recepção. Além disso, laços são relevantes em algumas práticas, como em sistemas de controle, onde um laço pode representar uma conexão de feedback que afeta a estabilidade do sistema.

Grafo regular

Um grafo é dito regular se todos os seus vértices têm o mesmo grau. Desta forma, o número de arestas incidentes em cada vértice é o mesmo.

Formalmente, um grafo G é k-regular se cada vértice em G tem grau k. Por exemplo, um grafo 3-regular é um grafo em que cada vértice tem grau 3. Na figura a seguir, temos o grafo G_4, exemplo de um grafo 2-regular.

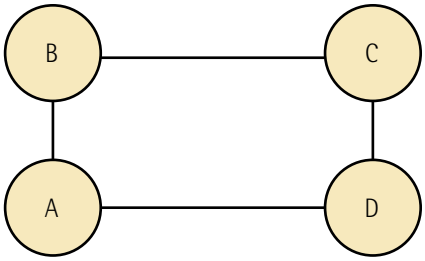

Figura 6 – Grafo G_4: 2-regular

Grafos regulares têm algumas propriedades interessantes. Por exemplo, eles podem ser usados para modelar estruturas simétricas, como polígonos regulares. Além disso, existem fórmulas para calcular o número de arestas e o número de vértices em um grafo regular, o que pode ser útil em alguns problemas de combinação e otimização.

É importante notar que nem todo grafo é regular. Muitos grafos na prática têm vértices com diferentes graus, o que os torna irregulares.

Grafo completo

Na teoria dos grafos, um grafo completo é um grafo simples em que cada par de vértices é adjacente, ou seja, existe uma aresta que liga cada par de vértices do grafo.

Formalmente, um grafo completo com n vértices é denotado por K_n. Um grafo completo K_n tem $n \cdot (n - 1)/2$ arestas e cada vértice tem grau $n - 1$.

Por exemplo, um grafo completo com quatro vértices, K_4, é representado pelo grafo G_5 da figura a seguir:

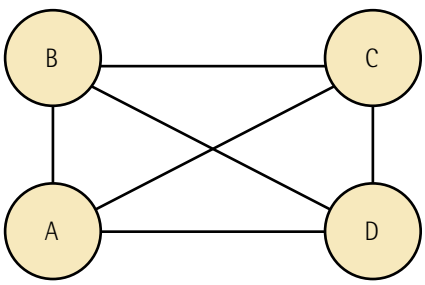

Figura 7 – Grafo G_5: grafo completo de quatro vértices

Nesse grafo, cada par de vértices (A,B), (A,C), (A,D), (B,C), (B,D) e (C,D) são adjacentes, o que significa que o grafo é completo.

Perceba que esse grafo tem $n = 4$ vértices; logo, o número de arestas é:

$n \cdot (n - 1)/2 = 4 \cdot (4 - 1)/2 = 6$ arestas, e o grau de cada vértice é dado por $n - 1 = 4 - 1 = 3$.

Outra propriedade é que grafos completos são também regulares.

Os grafos completos são frequentemente usados como modelos para situações em que todos os pares de elementos têm alguma relação entre si, como em problemas de otimização combinatória ou em sistemas de comunicação em que todos os dispositivos precisam se comunicar entre si.

Cadeia

Uma cadeia em um grafo é uma sequência de vértices e arestas alternadas, em que cada aresta conecta os vértices adjacentes na sequência. Por exemplo, considerando o grafo G_6 com os nós N = {A, B, C, D} e arestas A = {(A, B), (B, C), (C, D)}, uma cadeia poderia ser: A → B → C → D.

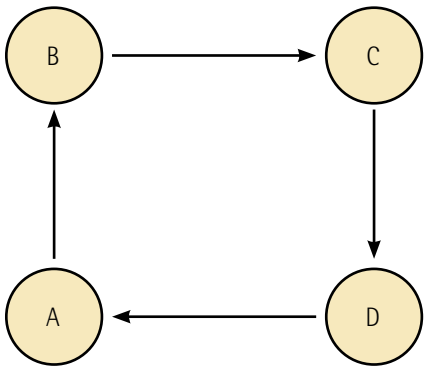

Figura 8 – Grafo G_6: A → B → C → D formam uma cadeia

Outro exemplo de cadeia é o grafo G_7:

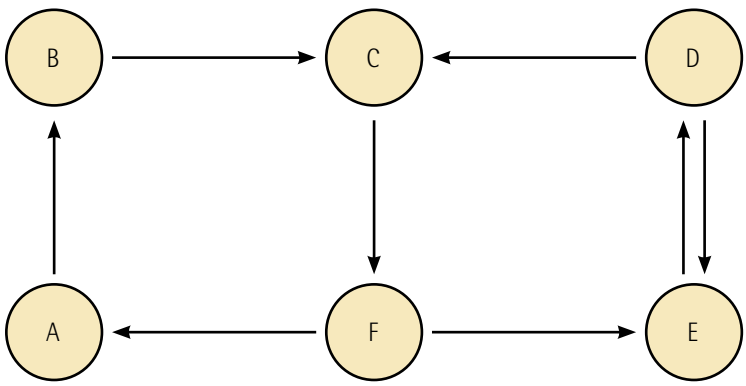

Figura 9 – Grafo G_7: E → F → A → B formam uma cadeia

O conceito de cadeia vale tanto para grafos não orientados como para grafos orientados; no segundo caso, só ignore o sentido da orientação dos arcos. A sequência de nós E → F → A → B é um exemplo de cadeia.

- **Cadeia elementar**: não passa duas vezes pelo mesmo vértice (nó).

- **Cadeia simples**: não passa duas vezes pela mesma aresta.

O comprimento de uma cadeia é a quantidade de arestas que ela possui.

Caminho

Podemos entender um caminho como uma sequência de vértices conectados em um grafo, em que cada vértice na sequência está diretamente ligado ao próximo vértice. Caminho, portanto, é uma rota que percorre, no grafo, as arestas de mesma orientação, passando por diferentes vértices, sem repeti-los.

Um caminho pode ser representado como uma sequência de vértices – por exemplo, v_1, v_2, v_3, ..., v_n, onde cada v_i está conectado ao $v_i + 1$ por uma aresta. O número de vértices em um caminho é chamado de comprimento do caminho.

Existem diferentes tipos de caminhos, dependendo das restrições e propriedades específicas. Alguns exemplos comuns incluem:

- **Caminho simples**: um caminho em que nenhum vértice é visitado mais de uma vez, ou seja, não há repetição de vértices no caminho.

- **Caminho fechado ou ciclo**: um caminho em que o primeiro e o último vértices são o mesmo, formando um ciclo.

- **Caminho mínimo**: um caminho entre dois vértices com o menor comprimento possível, geralmente medido em termos do número de arestas percorridas.

- **Caminho euleriano**: um caminho que passa por todas as arestas de um grafo exatamente uma vez.

- **Caminho hamiltoniano**: um caminho que passa por todos os vértices de um grafo exatamente uma vez.

Os caminhos são fundamentais na teoria dos grafos e têm várias aplicações em ciência da computação, otimização, redes, algoritmos de busca, entre outros campos. Eles são usados para resolver problemas como encontrar uma rota mais curta entre dois pontos, encontrar ciclos em um grafo e determinar a conectividade entre vértices.

A sequência de vértices (B, C, F, E, D) é um exemplo de caminho em G_7.

Circuito

Um circuito é um caminho simples e fechado. A sequência de vértices (B, C, F, A, B) é um exemplo de circuito em G_7.

2 LINGUAGENS REGULARES

Sabemos que um computador é um dispositivo ou sistema capaz de executar operações de processamento de informações. Ele é projetado para receber dados, realizar operações específicas sobre esses dados e fornecer resultados. Entretanto, para nossa disciplina, o que nos interessa são os modelos computacionais, abstrações matemáticas ou teóricas de sistemas computacionais usadas para estudar a capacidade de computação e as propriedades dos algoritmos. Esses modelos fornecem uma representação formal dos processos de computação e ajudam a compreender o que é computacionalmente possível e o que não é.

Existem vários modelos computacionais amplamente estudados, cada um com suas características e poder computacional específico. Neste momento, estudaremos os autômatos finitos determinísticos (AFD) ou máquina de estados finitos.

2.1 Autômatos finitos determinísticos

Os AFDs representam um modelo computacional com memória extremamente restrita – mas, apesar da limitação de memória, um AFD pode executar funções de muita importância. Como exemplo, podemos citar os controladores: computadores que exercem funções específicas, como portas automáticas e freios ABS.

Também um sistema de controle de semáforos é um controlador que pode ser relacionado a um AFD. Considere um cruzamento de ruas com semáforos para os veículos em cada direção. O sistema de controle de semáforos precisa decidir quando cada sinal deve mudar de verde para amarelo e, em seguida, de amarelo para vermelho, seguindo um padrão.

Podemos modelar esse sistema de controle de semáforos como um AFD. Cada estado do AFD representa uma configuração dos semáforos, indicando quais sinais estão verdes, amarelos ou vermelhos em determinado momento. As transições entre os estados ocorrem quando há uma mudança nos semáforos de acordo com as regras predefinidas.

Por exemplo, podemos ter os seguintes estados no AFD do sistema de controle de semáforos:

- **Estado 1**: semáforo para veículos na rua A está verde, semáforo para veículos na rua B está vermelho.
- **Estado 2**: semáforo para veículos na rua A está amarelo, semáforo para veículos na rua B está vermelho.
- **Estado 3**: semáforo para veículos na rua A está vermelho, semáforo para veículos na rua B está verde.
- **Estado 4**: semáforo para veículos na rua A está vermelho, semáforo para veículos na rua B está amarelo.

As transições entre esses estados ocorrem com base nas condições, como o tempo decorrido desde a última mudança de sinal, a detecção de veículos nas ruas ou um botão de pedestre interceptado.

Assim, o AFD do sistema de controle de semáforos descreve o comportamento determinístico do controlador. Dado um estado atual e uma entrada (por exemplo, uma detecção de veículo ou o tempo decorrido), o AFD determina qual será o próximo estado e, consequentemente, quais semáforos serão ativados.

Esse exemplo ilustra como um controlador pode ser modelado usando um AFD, cujas transições de estado são determinadas por entradas específicas e regras predefinidas, geradas em um comportamento controlado e previsível.

Os autômatos finitos são modelos matemáticos simples e abstratos que podem ser representados por grafos direcionados, nos quais os nós são os estados e as arestas são as transições entre os estados. Esses autômatos possuem uma quantidade finita de estados e podem ler uma entrada de símbolos do alfabeto, movendo-se de um estado para outro de acordo com as transições definidas.

Podemos também entender os AFDs como abstrações usadas para reconhecer palavras (sequência finita de símbolos) em linguagens regulares. De outro modo, esses AFDs reconhecem todas as sequências pertencentes à linguagem para a qual foram criados e rejeitam todas as sequências que não pertencem a ela.

Os autômatos finitos consistem em dois componentes principais:

- **Fita de entrada**: é uma memória finita que contém a cadeia (palavra ou string) a ser reconhecida pelo autômato. A fita é dividida em células, e cada célula armazena um símbolo da cadeia de entrada. A leitura dos símbolos na fita é feita usando um cursor que aponta para o próximo símbolo a ser processado, movendo-se apenas da esquerda para a direita.

- **Unidade de controle finito ou máquina de estados**: é o controlador central do autômato. Possui uma lista finita de estados e transições que definem as possíveis movimentações do cursor. Um estado armazena informações relevantes do passado, necessárias para o processamento posterior da cadeia de entrada. O autômato inicia em um estado inicial único, com o cursor apontado para o símbolo mais à esquerda da cadeia. Cada transição no AFD é especificada por uma tripla (estado atual, símbolo atual, próximo estado), em que o símbolo pode ser um símbolo vazio (ε) ou qualquer elemento do alfabeto da cadeia de entrada na qual a linguagem está definida. Veja a figura a seguir.

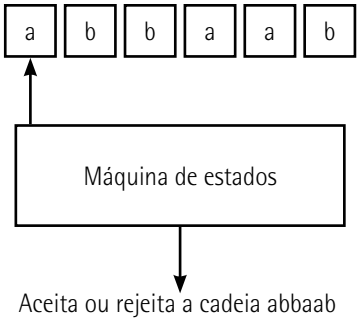

Figura 10 – Estrutura de um autômato finito

Formalmente, um AFD é uma 5-upla $(Q, \Sigma, \delta, q_0, F)$, onde:

- **Conjunto de estados (Q)**: é um conjunto finito de estados nos quais o AFD pode estar. Cada estado representa uma configuração específica do autômato.
- **Alfabeto (Σ)**: é um conjunto finito de símbolos que são aceitos como entrada pelo autômato. Pode incluir, por exemplo, caracteres, números ou qualquer outro símbolo relevante para o problema.
- **Função de transição (δ)**: é uma função que mapeia um estado atual e um símbolo de entrada para o próximo estado. Essa função define as transições possíveis que o autômato pode fazer. Matematicamente, $\delta = Q \times \Sigma \rightarrow Q$.
- **Estado inicial (q_0)**: é o estado em que o AFD está antes de processar qualquer entrada. Matematicamente, $q_0 \in Q$.
- **Conjunto de estados finais (F)**: é um conjunto de estados que indicam que uma determinada sequência de entrada foi reconhecida pelo autômato. Esses estados representam o sucesso do processamento. Matematicamente, $F \subseteq Q$.

O comportamento de um AFD é determinístico, o que significa que para um estado atual e um símbolo de entrada específico há uma única transição definida. Isso diferencia o AFD de um autômato finito não determinístico (AFN), que pode ter múltiplas transições para um mesmo estado e símbolo de entrada.

Observação

AFDs são usados na computação e têm aplicações como análise léxica, processamento de linguagem natural, reconhecimento de padrões e validação de entradas, fundamentais para entender linguagens formais e computabilidade.

As linguagens regulares são um tipo de linguagem formal que pode ser descrita ou reconhecida por AFDs ou AFNs. Essas linguagens são estudadas no campo da teoria das linguagens formais e autômatos.

Uma linguagem é considerada regular se, e somente se, ela puder ser gerada por uma gramática regular ou reconhecida por um autômato finito. Em outras palavras, uma linguagem é regular se existe um autômato finito capaz de reconhecer todas as palavras pertencentes a essa linguagem e rejeitar todas as palavras que não pertencem a ela.

As linguagens regulares possuem propriedades específicas que podem ser expressas nos termos da definição formal de um AFD ou das gramáticas regulares, como a propriedade de serem fechadas em operações como união, interseção e complemento.

Além disso, as linguagens regulares podem ser descritas por meio de expressões regulares (ER), que são padrões formais utilizados para representar conjuntos de palavras. ERs são amplamente usadas em linguagens de programação, processamento de texto, busca de padrões e outras áreas relacionadas à manipulação de sequências de caracteres.

Em resumo, as linguagens regulares são um importante conceito na teoria da computação e têm aplicações práticas em diversas áreas da ciência da computação, especialmente na área de compiladores, linguagens de programação e processamento de texto.

Exemplo: seja M um AFD $(Q, \Sigma, \delta, q_0, F)$, onde:

$Q = \{q_0, q_1, q_2\}$

$\Sigma = \{x, y, z\}$

q_0 é o estado inicial

$F = \{q_2\}$

$\delta = \{((q_0, x), q_1), ((q_1, y), q_2), ((q_2, z), q_2)\}$

Verificar se palavra xyzz pertence à linguagem L reconhecida pelo AFD M.

O processamento da cadeia deve iniciar-se no estado inicial q_0 com o cursor sob o símbolo mais à esquerda, ou seja, o símbolo x.

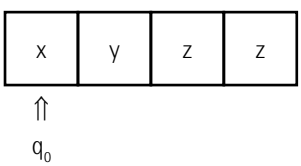

Figura 11

Lido o símbolo x, o cursor move-se uma célula à direita. A função de transição δ indica que, para o par (q_0, x), o autômato deve alcançar o estado q_1.

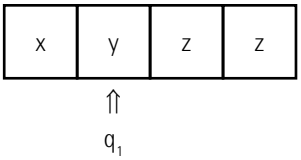

Figura 12

Lido o símbolo y, o cursor move-se uma célula à direita. A função de transição δ indica que, para o par (q_1, y), o autômato deve alcançar o estado q_2.

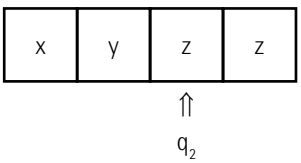

Figura 13

Lido o símbolo z, o cursor move-se uma célula à direita. A função de transição δ indica que, para o par (q_2, z), o autômato se mantém em q_2.

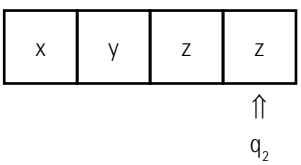

Figura 14

Lido o segundo símbolo z, o cursor move-se à direita. A função de transição δ indica que, para o par (q_2, z), o autômato se mantém em q_2.

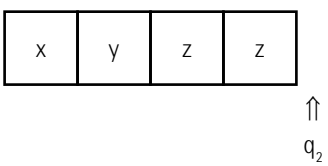

Figura 15

Como o cursor está à direita do último símbolo da fita e no estado final q_2, o AFD alcançou sua configuração final. Nesta configuração, diz-se que o autômato finito reconhece a cadeia de entrada, ou seja, a cadeia de entrada pertence à linguagem reconhecida pelo autômato.

É importante frisar que o reconhecimento de qualquer cadeia só ocorre se, após lidos todos os símbolos, o cursor parar à direita do último símbolo e o AFD estiver no estado final.

Vejamos agora como se dá o processamento de algumas palavras que não pertencem à linguagem L reconhecida por M.

Exemplo: verificar se a palavra xyyz pertence à linhagem L reconhecida pelo AFD M.

Como no exemplo anterior, o AFD inicia a leitura no estado inicial q_0 com o cursor sob a célula mais à esquerda da fita, ou seja, o símbolo x.

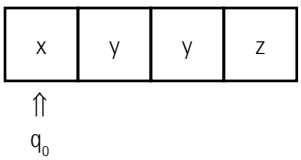

Figura 16

Lido o símbolo x, o cursor move-se uma célula à direita. A função de transição δ indica que, para o par (q_0, x), o autômato deve alcançar o estado q_1.

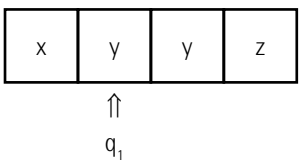

Figura 17

Lido o símbolo y, o cursor move-se uma célula à direita. A função de transição δ indica que, para o par (q_1, y), o autômato deve alcançar o estado q_2.

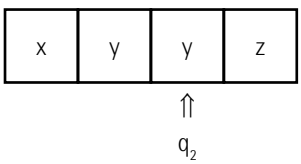

Figura 18

Note que em δ não existe uma transição que indique o par (q_2, y). Dessa forma, a leitura não consegue prosseguir ao atingir o segundo y, e o AFD atinge a configuração final com o cursor não estando à direita do último símbolo da fita. Logo, a palavra xyyz não pertence à linguagem reconhecida pelo AFD M. Dizemos então que M rejeita a cadeia xyyz.

Vejamos outro caso em que M rejeita a cadeia de entrada. Vamos verificar se a palavra x pertence à linhagem L reconhecida pelo AFD M.

O AFD inicia a leitura no estado inicial q_0 com o cursor sob a célula mais à esquerda da fita, ou seja, o símbolo x.

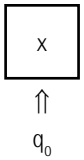

Figura 19

Lido o símbolo x, o cursor é movido à direita, com o autômato no estado q_1.

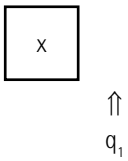

Figura 20

O AFD M atinge a configuração final, com o cursor à direita do último símbolo da fita. Entretanto, M rejeita a cadeia de entrada pois ele não alcançou o estado final q_2. Logo, a palavra x não pertence à linguagem reconhecida pelo autômato M.

Sobre esses três exemplos, são apresentadas algumas conclusões:

> O autômato sempre para ao processar qualquer que seja a cadeia de entrada, pois apenas uma das seguintes condições ocorrem:
>
> 1. A cadeia é processada até o último símbolo e o autômato assume um estado final reconhecendo a cadeia.
>
> 2. Após o processamento do último símbolo da fita, o autômato finito assume um estado não final. O autômato para e a cadeia de entrada é rejeitada.
>
> 3. A função de transição é indefinida para um símbolo da cadeia de entrada. O autômato para e a cadeia de entrada é rejeitada (Moraes, 2013, p. 29).

Nos exemplos anteriores, verificamos através da máquina de estados finitos M como um AFD processa determinada cadeia de entrada, reconhecendo-a ou rejeitando-a. Entretanto, não definimos qual é a linguagem reconhecida por M. Vamos agora estudar as operações regulares, e depois retomamos o AFD M determinando a linguagem reconhecida por ele.

Saiba mais

Para saber mais sobre autômatos, leia:

HOPCROFT J.; ULLMAN, J.; MOTWANI, R. *Introdução à teoria dos autômatos, linguagens e computação*. Rio de Janeiro: Campus, 2002.

2.2 Operações regulares

Em linguagens formais, as operações regulares referem-se a um conjunto de operações que podem ser aplicadas a linguagens regulares para formar novas linguagens regulares. As linguagens regulares são aquelas que podem ser reconhecidas por autômatos finitos ou, equivalentemente, podem ser descritas por meio de ERs.

- **União (ou soma)**: dadas duas linguagens regulares L_1 e L_2, a união de L_1 e L_2, denotada por $L_1\ L_2$ ou $L_1 + L_2$, é a linguagem que contém todas as palavras que pertencem a L_1 ou a L_2 (ou a ambas). Exemplo: se a linguagem $L_1 = \{0, 01, 001\}$ e a linguagem $L_2 = \{\varepsilon, 11, 111\}$, então $L_1 \cup L_2 = \{\varepsilon, 0, 01, 001, 11, 111\}$.

- **Concatenação**: dadas duas linguagens regulares L_1 e L_2, a concatenação de L_1 e L_2, denotada por $L_1\ L_2$ ou L_1L_2, é uma linguagem formada pela concatenação de cada palavra de L_1 com cada palavra de L_2. Exemplo: se a linguagem $L_1 = \{0, 01, 001\}$ e a linguagem $L_2 = \{\varepsilon, 10\}$, então $L_1L_2 = \{0\ \varepsilon, 01\ \varepsilon, 001\ \varepsilon, 010, 0110, 00110\} = \{0, 01, 001, 010, 0110, 00110\}$.

- **Fechamento de Kleene (ou estrela)**: o fechamento de Kleene aplicado a uma linguagem L resulta em uma nova linguagem L*, que consiste em todas as possíveis concatenações de zero ou mais strings pertencentes à linguagem L. Em outras palavras, L* inclui todas as strings que podem ser formadas pela concatenação de qualquer número (incluindo zero) de strings em L. Por exemplo, se tivermos a linguagem L = {a, b}, então L* incluirá todas as combinações possíveis de sequências formadas por a e b, incluindo strings vazias. Assim, L* seria {ε, a, b, aa, ab, ba, bb, aaa, ...}.

- **Fechamento transitivo**: é o conjunto formado por todas as cadeias do alfabeto Σ, com exceção da cadeia vazia. Representa-se o fechamento transitivo por L^+. Exemplo: seja o alfabeto unário $\Sigma = \{0\}$, tem-se que $L^+ = \{0, 00, 000, 0000, 00000, ...\}$.

Agora que estudamos as operações, retomaremos o AFD M e determinaremos a linguagem reconhecida por ele.

Exemplo: seja M um AFD $(Q, \Sigma, \delta, q_0, F)$, onde:

$Q = \{q_0, q_1, q_2\}$

$\sum = \{x, y, z\}$

q_0 é o estado inicial

$F = \{q_2\}$

$\delta = \{((q_0, x), q_1), ((q_1, y), q_2), ((q_2, z), q_2)\}$

Determine a linguagem L reconhecida por M.

Uma maneira apropriada de se determinar a linguagem é montarmos um grafo orientado que represente a função de transição δ.

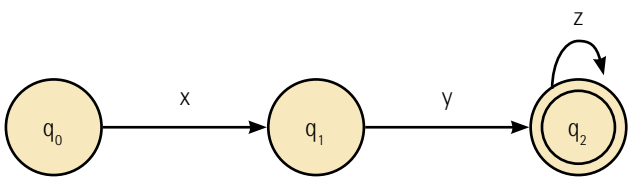

Figura 21

No grafo, os nós representam os estados q_0, q_1 e q_2. Em q_2 temos um nó representado por um círculo duplo, indicando o estado final. Cada aresta (setas orientadas) representam o símbolo do alfabeto ($\sum$) lido.

O processamento de todas as palavras pertencentes à linguagem se inicia no estado q_0. Dessa forma, segundo a função de transição, o processamento das palavras pertencentes à linguagem L reconhecida por M se dá em três passos:

- estando inicialmente o AFD M no estado inicial q_0 e lendo o símbolo x, ele alcança o estado q_1;
- estando o AFD no estado q_1 e lendo o símbolo y, ele alcança o estado q_2;
- estando o AFD no estado q_2 e lendo o símbolo z, ele permanece em q_2.

Note, no grafo, que no estado q_2 temos um laço, ou seja, o terceiro passo pode se repetir indefinidamente e, portanto, M reconhece qualquer palavra iniciada com os símbolos xy e finalizada, obrigatoriamente, com um ou mais símbolo z.

Matematicamente, temos que a linguagem L reconhecida pelo autômato M é $L = \{\omega \mid \omega = xyz^n \text{ e } n > 1\}$.

Vamos agora resolver o exemplo a seguir.

Exemplo de aplicação

Seja M_1 um AFD $(Q, \Sigma, \delta, q_0, F)$, onde:

$Q = \{q_0, q_1, q_2\}$

$\Sigma = \{0, 1\}$

q_0 é o estado inicial

$F = \{q_1\}$

$\delta = \{((q_0, 0), q_0), ((q_0, 1), q_1), ((q_1, 1), q_1), ((q_1, 0), q_2), ((q_2, 0), q_1), (q_2, 1), q1\}$

Determine:

a) O que ocorre com M_1 ao processar a cadeia 1101.

Resolução

Iniciamos a leitura da cadeia de entrada com M_1 no estado inicial e o cursor sob o símbolo mais à esquerda da fita, ou seja, o símbolo 1.

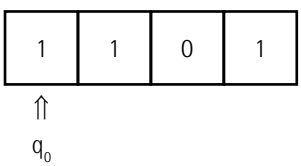

Figura 22

Lido o símbolo 1, o cursor move-se uma célula à direita. A função de transição δ indica que, para o par $(q_0, 1)$, M_1 alcança o estado q_1.

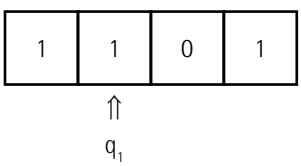

Figura 23

Lido o segundo símbolo 1, o cursor move-se uma célula à direita. A função de transição δ indica que, para o par $(q_1, 1)$, M_1 se mantém no estado q_1.

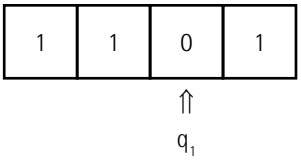

Figura 24

Lido o símbolo 0, o cursor move-se uma célula à direita. A função de transição δ indica que, para o par (q_1, 0), M_1 alcança o estado q_2.

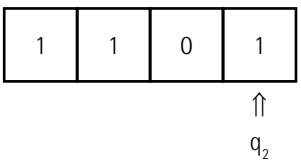

Figura 25

Lido o terceiro símbolo 1, o cursor move-se uma célula à direita. A função de transição δ indica que, para o par (q_2, 1), M_1 reestabelece o estado q_1.

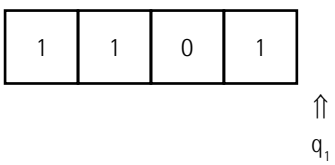

Figura 26

O autômato atingiu a configuração final, visto que o estado q_1 é o estado final e o cursor aponta para a direita do último símbolo da cadeia de entrada. Nesta configuração, diz-se que o autômato finito reconhece a cadeia de entrada 1101, ou seja, a cadeia de entrada pertence à linguagem reconhecida pelo autômato.

b) Através da função de transição δ, monte o grafo que representa o autômato M_1.

Resolução

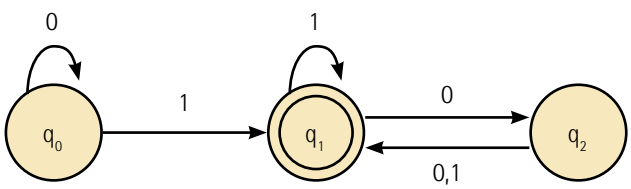

Figura 27

c) Determine a linguagem L_1 reconhecida por M_1.

Resolução

Observando o grafo, é possível concluir que ele reconhece qualquer cadeia terminada com o símbolo 1, como {1, 01, 11, 0101010101, ...}, pois o AFD sempre se move para o estado final q_1 ao ler o símbolo 1. Ele também reconhece qualquer cadeia que tenha um símbolo 1 seguido de uma quantidade par do símbolo 0, como {100, 0100, 110000, 0101000000, ...}.

Matematicamente, L1 = {ω /ω = 0*1x ou 0*1x0^y, com x ≥ 1 e y par}.

2.3 Autômato finito não determinístico

Um AFN é um tipo de autômato em que, a partir de um estado e um símbolo de entrada, pode haver mais de uma transição possível. Isso é, os AFNs permitem que o par (estado corrente, símbolo corrente) apresente diversos próximos estados. Assim, o "não determinismo" significa que o autômato pode estar em vários estados ativos concomitantemente.

Ao receber uma entrada, um AFN pode navegar por diferentes caminhos ou possibilidades de transição ao mesmo tempo, considerando todas as opções disponíveis. Isso torna os AFNs mais expressivos do que os AFDs, pois podem compreender linguagens mais complexas.

Um exemplo prático de aplicação de AFN está no reconhecimento de ERs em processamento de linguagens de programação. Por exemplo, é comum utilizar um AFN para analisar um código-fonte em busca de padrões específicos, como identificar todas as ocorrências de palavras-chave, números ou símbolos em uma linguagem de programação. O AFN pode percorrer diferentes caminhos, testando várias combinações de transições, a fim de reconhecer corretamente os padrões desejados.

Formalmente, um AFN é uma quíntupla (Q, $\sum$, δ, q0, F), onde:

Q é um conjunto finito de estados.

$\sum$ é um alfabeto finito de símbolos de entrada.

δ é uma função de transição que mapeia $Q \times (\sum \cup \{\varepsilon\}) \; 2^{|Q|}$, ou seja, para cada estado $q_n \in Q$ e símbolo de entrada $a \in (\sum \cup \{\varepsilon\})$, $\delta(q_n, a)$ levará a um conjunto de estados.

q0 é o estado inicial, onde $q0 \in Q$.

F é um conjunto de estados finais ou aceitos, onde $F \subseteq Q$.

A função de transição δ permite que um AFN tenha várias transições possíveis para um dado estado e símbolo de entrada, incluindo a transição vazia (ε-transição). Isso dá ao AFN a capacidade de considerar várias possibilidades de caminho ao processar uma entrada.

A seguir serão mostrados grafos que representam AFNs para que possamos reconhecê-los, defini-los e entender o processamento das cadeias de entrada.

Observe a figura a seguir.

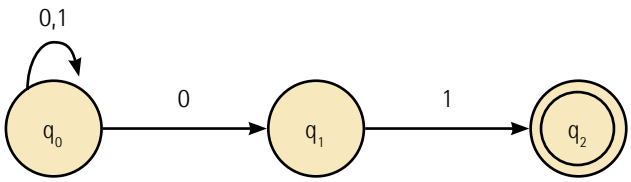

Figura 28 – Exemplo de AFN (N_1)

Note que no estado inicial q_0, lendo o símbolo 1, ela permanece em q_0, ou seja, existe **δ** = ((q0,1), q1), portanto, nenhuma diferença com um AFD. Entretanto, estando N_1 no estado q_0, lendo o símbolo 0, a função de transição δ permite que N_1 permaneça no estado q_0 ou avance para o estado q_1, ou seja, existe **δ** = ((q0,0), {q0,q1}), diferentemente de um AFD, que só permitiria a transição para um único estado.

A transição **δ** = ((q0,0), {q0,q1}) ocorre em paralelo, como se N_1 criasse uma cópia de si mesmo; assim, uma delas permaneceria em q_0 e a outra avançaria a q_1.

Definindo formalmente o autômato N_1:

$N_1 = (Q, \Sigma, \delta, q0, F)$, onde:

$Q = \{q_0, q_1, q_2\}$

$\Sigma = \{0, 1\}$

Uma outra maneira de representar a função de transição δ é através de uma tabela:

δ	0	1
q_0	$\{q_0, q_1\}$	$\{q_0\}$
q_1	∅	$\{q_2\}$
q_2	∅	∅

Onde:

q_0 é o estado inicial

$F = \{q_2\}$

Agora entenderemos o processamento em paralelo através da cadeia **00101** no AFN N_1.

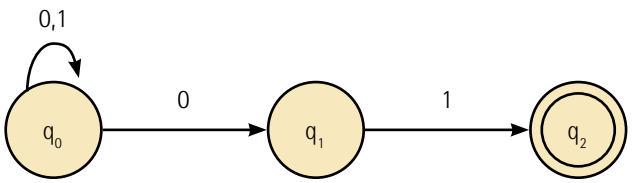

Figura 29

Começamos processando o símbolo **0** (grafado na cor preta).

Observe que estando N_1 em q_0, lendo **0**, pode avançar a q_1 ou permanecer em q_0.

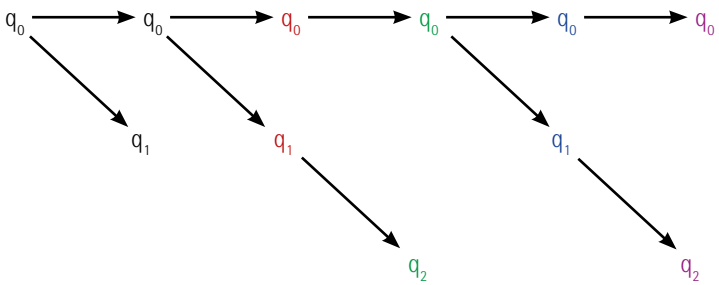

Figura 30

Neste momento, uma cópia de N_1 está em q_0, e a outra cópia está em q_1, ambas lendo o símbolo **0**. Perceba que a cópia de N_1 que está em q_1 lendo **0** rejeita a cadeia, pois não existe a transição (q1,0). Já a cópia que está em q_0 lendo **0** mais uma vez se duplica, criando uma cópia que permanece em q_0 e outra que avança a q_1.

Agora começa a leitura do símbolo **1**. A cópia de N_1 que está em q_1 avança a q_2 e rejeita a cadeia, pois atingiu o estado final sem processar todos os símbolos. Já a cópia que está em q_0, lendo **1**, permanece em q_0.

Estando N_1 no estado q_0, lendo **0**, se duplica, criando uma cópia que permanece em q_0 e outra que avança a q_1.

A cópia que está em q_0 inicia a leitura do último símbolo, ou seja, **1**, e permanece em q_0. Neste momento, ela rejeita a cadeia, pois a leitura acabou em um estado não final. Já a cópia que está em q_1

também inicia a leitura do símbolo 1 e avança para q_2, reconhecendo a cadeia 00101, pois ela leu todos os símbolos de entrada e parou no estado final.

Em outro exemplo, observe a figura a seguir.

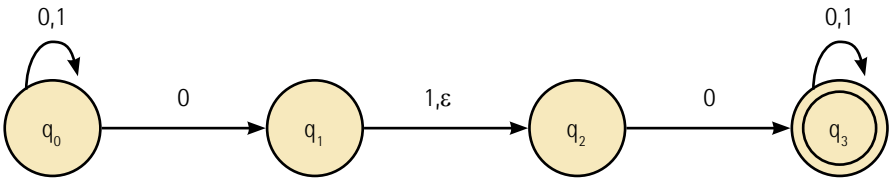

Figura 31 – Exemplo de AFN (N_2)

Estando N_2 no estado q_0, lendo os símbolos de entrada 0 ou 1, as transições ocorrerão como no exemplo anterior. A novidade aqui se dá quando N_2 atinge o estado q_1: neste caso ocorre a leitura do símbolo 1 e também, simultânea e paralelamente, a leitura de nenhum símbolo (palavra vazia ε), e alcançam o estado q_2.

Definindo formalmente o autômato N_2:

$N_2 = (Q, \Sigma, \delta, q0, F)$, onde:

$Q = \{q_0, q_1, q_2, q_3\}$

$\Sigma = \{0, 1\}$

δ	0	1	ε
q_0	$\{q_0, q_1\}$	q_0	Ø
q_1	Ø	$\{q_2\}$	$\{q_2\}$
q_2	$\{q_3\}$	Ø	Ø
q_3	$\{q_3\}$	$\{q_3\}$	Ø

Onde:

q_0 é o estado inicial

$F = \{q_3\}$

Vejamos agora o processamento da cadeia de entrada 00110. Resolveremos este exemplo montando as cópias de N_2. Devemos lembrar que as cópias processam a cadeia de entrada simultaneamente. Nos grafos que se seguirão, o vértice (nó) em vermelho indica o estado ativo.

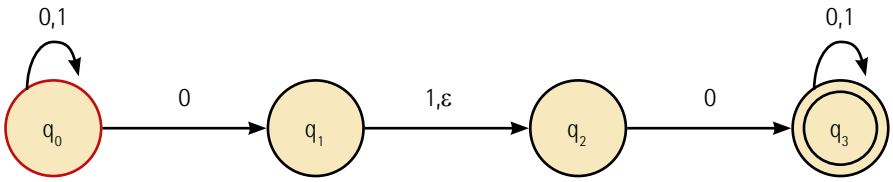

Figura 32 – Cópia 1

O estado inicial q_0 é o estado ativo. Assim, inicia-se a leitura do primeiro símbolo 0 da cadeia de entrada. Há com isso duas transições possíveis: permanecer em q_0 e avançar para q_1. Assim, a cópia 1 permanece em q_0 e cria-se a cópia 2, em que o estado ativo será q_1.

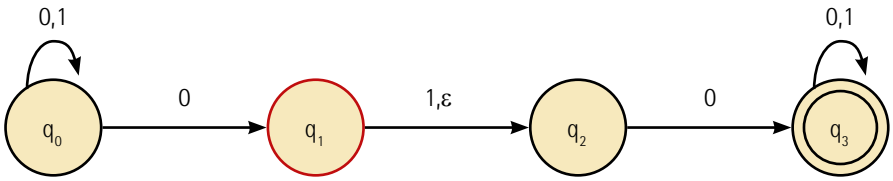

Figura 33 – Cópia 2

Em q_1, perceba que há uma transição em vazio (muda-se de estado sem efetuar a leitura de nenhum símbolo). Assim, automaticamente cria-se a cópia 3, em que o estado q_2 fica ativo.

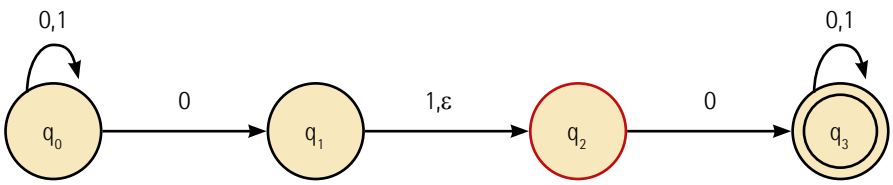

Figura 34 – Cópia 3

Note que ao fazer a leitura do primeiro símbolo 0 da cadeia de entrada, N_2 criou três cópias de si mesmo que trabalham processando a cadeia de entrada de forma simultânea.

Iniciando a leitura do segundo 0 da cadeia de entrada, em todas as cópias existentes, verifica-se na cópia 1 que há duas transições possíveis: permanecer em q_0 e avançar para q_1. Assim, a cópia 1 permanece em q_0 e cria-se a cópia 4, onde o estado ativo será q_1.

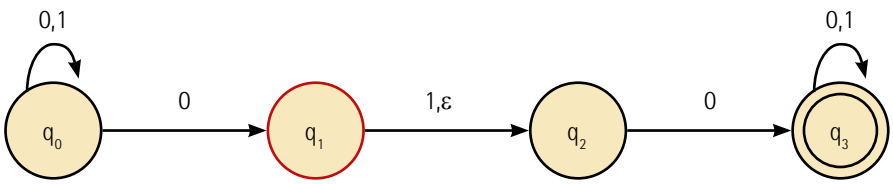

Figura 35 – Cópia 4

Em q_1, na cópia 4, há a transição em vazio. Assim, automaticamente cria-se a cópia 5, onde o estado q_2 fica ativo.

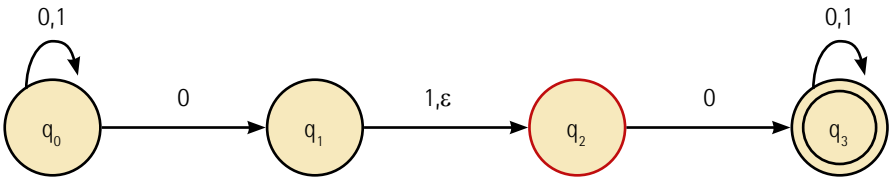

Figura 36 – Cópia 5

Como todas as cópias processam a cadeia simultaneamente e, neste momento, N_2 está efetuando a leitura do segundo zero da cadeia de entrada, note que para a cópia 2, onde o estado ativo é q_1, não existe transição lendo o símbolo 0. Logo, a cópia 2 para e rejeita a cadeia de entrada, finalizando, assim, sua existência. Dizemos informalmente que a cópia 2 morreu.

Na cópia 3, o estado ativo é q_2 e, lendo o símbolo 0, ativa-se o estado q_3. Passamos agora a denominar a cópia 3 de cópia 3'.

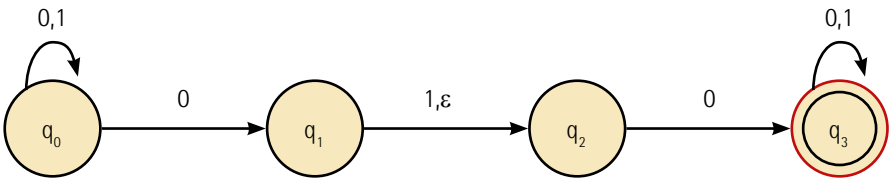

Figura 37 – Cópia 3'

Neste momento, temos então quatro cópias de N_2 processando a cadeia de entrada: cópia 1, cópia 3', cópia 4, cópia 5.

Partimos para a leitura do primeiro símbolo 1 da cadeia de entrada. Na cópia 1, estando q_0 ativo, a única transição possível é ela permanecer no estado q_0. O mesmo ocorre na cópia 3': estando q_3 ativo, a única transição possível é permanecer em q_3.

Na cópia 4, o estado ativo é q_1. Lendo o símbolo 1, ela avança a q_2. Passamos agora a denominar a cópia 4 de cópia 4'.

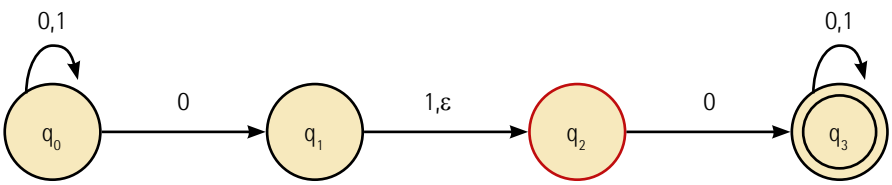

Figura 38 – Cópia 4'

Já na cópia 5, q_2 está ativo e não existe transição para o símbolo 1. Assim, a cópia 5 para e rejeita a cadeia de entrada.

Portanto, temos agora apenas três cópias de N_2 processando a cadeia de entrada: cópia 1, cópia 3', cópia 4'.

Avançamos agora para a leitura do segundo símbolo 1 da cadeia de entrada. Na cópia 1, o estado ativo é q_0, e, lendo o símbolo 1, permanece em q_0. Na cópia 3', o estado ativo é q_3 e, lendo o símbolo 1, permanece em q_3. Finalizando a leitura do segundo símbolo 1 da cadeia de entrada, na cópia 4' ela para e rejeita a cadeia de entrada, pois não existe transição para o símbolo 1.

Agora, portanto, temos apenas duas cópias de N_2 processando a cadeia de entrada: cópia 1 e cópia 3'.

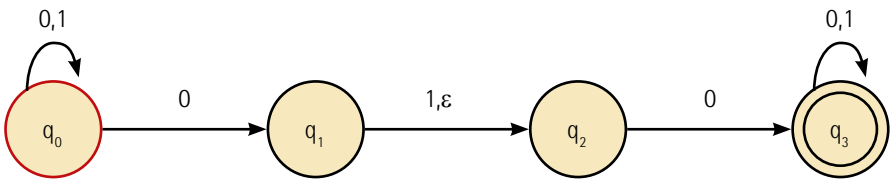

Figura 39 – Cópia 1

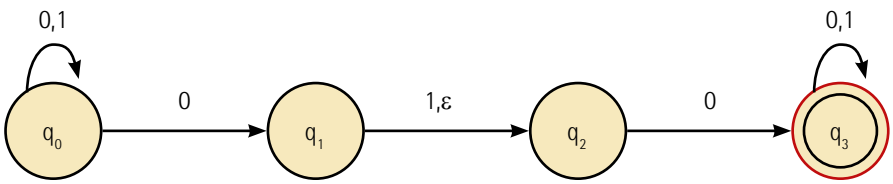

Figura 40 – Cópia 3'

Finalizando a leitura da cadeia de entrada, ou seja, lendo o terceiro símbolo 0, na cópia 1 o estado ativo é q_0. Vimos que as transições possíveis são permenecer em q_0 e avançar a q_1. Note que, em ambos os casos, a leitura da cadeia de entrada terminou, mas N_2 não está no estado final (q3). Logo, a cópia 1 para e rejeita a cadeia de entrada 00110.

Na cópia 3', o estado ativo é q_3 e, lendo o símbolo 0, ela permene em q_3. Como o estado q_3 é o estado final e a leitura da cadeia de entrada finalizou, dizemos que N_2 parou e aceitou a cadeia de entrada 00110.

2.4 Expressões regulares

As ERs constituem-se em uma alternativa para representar as linguagens regulares. Assim, usando as operações regulares (união, concatenação e estrela, vistas no item 2.2 deste livro-texto), podemos representar algebricamente as linguagens regulares. Portanto, o valor de uma ER é uma linguagem regular.

Exemplo: determine a linguagem regular definida pela expressão regular $(\varepsilon \cup 1)0^*$.

Aqui, temos a união da palavra vazia ε com 1 concatenada com 0*. Significa que as palavras (strings ou cadeias) pertencentes à expressão anterior podem ou não começar com um único símbolo 1 e terminar com nenhum ou mais símbolos 0s. Portanto, palavras pertencentes à linguagem regular que essa expressão representa são:

L = {ε, 0, 00, 000, 0000, ..., 1, 10, 100, 1000, 10000, ...}.

Outra ER que representa essa mesma linguagem pode ser escrita como $0^* \cup 10^*$.

Assim, podemos representar, na notação de conjuntos, a linguagem como $L = \{\omega / \omega \in \{0, 1\}$, onde $\omega = 0^k$ ou $\omega = 10^k$, com $k \geq 0\}$.

Observação

Na expressão regular $0^* \cup 10^*$, mais especificamente em 10^*, a operação estrela recai somente sobre o símbolo 0.

Definindo formalmente ER, temos: seja R uma expressão regular sobre um alfabeto Σ que representa a linguagem L(R) se, e somente, se:

R = x para $x \in \sum$, então L(R) = {x}

R = ε, então L(R) = {ε}

R = Ø, então L(R) = Ø

$R = R_1 R_2$, então $L(R) = L(R_1) L(R_2)$

$R = R_1R_2$, então $L(R) = L(R_1)L(R_2)$

$R = R_1^*$, então $L(R) = L(R_1)^*$

Observação

Atenção às ERs ε e Ø. Enquanto ε descreve a linguagem contendo somente a cadeia vazia, a expressão Ø descreve a linguagem que não possui nenhuma cadeia.

As operações contidas nas ER obedecem a uma ordem de precedência: respectivamente, estrela, concatenação e, por fim, união; a utilização de parêntese indica a operação prioritária, mudando a ordem estabelecida.

Vejamos agora um exemplo de como aplicar a precedência entre as operações contidas nas ER.

Exemplo de aplicação

Determine a linguagem descrita pelas ERs a seguir:

a) $R = ab \cup ca^*$

Resolução

Esta expressão regular descreve todas as cadeias formadas pela concatenação dos símbolos a e b união com a concatenação dos símbolos c e a*.

A concatenação de a com b resulta em {ab}.

A operação estrela em a resulta em {ε, a, aa, aaa, aaaa, ...}.

A concatenação de c com a* resulta em {c, ca, caa, caaa, caaaa, ...}.

Por fim, a união das duas concatenações descreve L(R), a saber:

L(R) = {ab, c, ca, caa, caaa, caaaa, ...}.

b) $R = a(b \cup ca^*)$

Resolução

Esta ER descreve todas as cadeias formadas pela concatenação do símbolo a com a união entre o símbolo b e a concatenação dos símbolos c e a*.

A operação estrela em a resulta em {ε, a, aa, aaa, aaaa, ...}.

A concatenação de c com a* resulta em {c, ca, caa, caaa, caaaa, ...}.

A união entre b e a concatenação de c com a* resulta em {b, c, ca, caa, caaa, caaaa, ...}.

Por fim, a linguagem L(R) descrita pela concatenação do símbolo a com a união entre o símbolo b e a concatenação dos símbolos c e a* é:

L(R) = {ab, ac, aca, acaa, acaaa, acaaaa, ...}.

c) $a(b \cup c)a^*$

Resolução

Esta expressão regular descreve todas as cadeias formadas pela concatenação do símbolo a com a união entre o símbolo b e c concatenada com a^*.

Isso quer dizer que todas as cadeias inicializarão com o símbolo a e finalizarão com zero ou mais símbolos a. Entre o início e o fim da cadeia deverá haver um b ou um c. Portanto:

L(R) = {ab, aba, abaa, abaaa, ..., ac, aca, acaa, acaaa, ...}.

Equivalência entre expressão regulares

Duas ERs R1 e R2 são equivalentes se, e somente se, descreverem a mesma linguagem. Denotamos por $R_1 \equiv R_2 \Leftrightarrow L(R_1) \equiv L(R_2)$.

Exemplo de aplicação

Determine uma expressão regular equivalente a:

a) $R_1 = (0 \cup \varepsilon)1$

Resolução

A linguagem descrita por essa ER é a cadeia gerada pela concatenação do símbolo 0 com o símbolo 1 ou pela concatenação da palavra vazia ε com o símbolo 1. Logo, a linguagem descrita é $L(R_1) = \{01, 1\}$. Agora fica relativamente fácil criar uma expressão regular equivalente: $(0 \cup \varepsilon)1 \equiv 01 \cup 1$.

b) $R_2 = (0 \cup 1)^*$

Resolução

Se considerarmos somente a ER dentro do parêntese, $0 \cup 1$, desconsiderando a operação *, teríamos que $L(R_2) = \{0,1\}$. Entretanto, quando inserimos a operação estrela a {0,1}, significa que estamos concatenando zero ou mais vezes o símbolo 0 com ele mesmo e com o símbolo 1, como também concatenando zero ou mais vezes o símbolo 1 com ele mesmo e com 0.

Basicamente, o operador * permite que qualquer combinação possível de zeros e uns possa ser gerada por essa ER. Isso inclui cadeias com qualquer quantidade de zeros e uns, em qualquer ordem.

Logo, a linguagem descrita é $L(R_2) = \{ \omega | \omega \in \{0,1\}^* \}$. Imaginar uma ER equivalente a $(0 \cup 1)^*$ talvez não seja tão automático como no exemplo anterior. Façamos, então, uma tentativa com a expressão regular (0*1*) e verificaremos as cadeias reconhecidas por ela:

- Reconhece cadeia vazia.
- Reconhece cadeias só com 0.
- Reconhece cadeias só com 1.

Reconhece cadeias iniciadas com uma ou mais ocorrências de 0 seguidas de uma ou mais ocorrências de 1.

Até o momento, (0*1*) reconheceu as mesmas cadeias reconhecidas por R_2. Entretanto, R_2 também reconhece cadeias com qualquer quantidade de zeros e uns, em qualquer ordem, fato que (0*1*) não é capaz de reconhecer. Para resolvermos, basta incluir o operador estrela fora dos parênteses. Assim, (0*1*)*. Portanto:

$(0 \cup 1)^* \equiv (0^*1^*)^*$

2.5 Lema do bombeamento para linguagens regulares

Primeiramente, devemos nos lembrar que linguagem regular é aquela que pode ser representada por um autômato finito.

O lema do bombeamento é frequentemente utilizado como uma ferramenta para mostrar que certas linguagens não são regulares. Ao assumir que uma linguagem é regular, podemos aplicar a lema do bombeamento e chegar a uma contradição, demonstrando que a linguagem não pode ser regular.

No entanto, é importante observar que o lema do bombeamento não pode ser usado para provar que uma linguagem é regular. Ele apenas nos fornece um método para identificar quando uma linguagem não é regular. Existem outras técnicas e propriedades para provar que uma linguagem é regular – por exemplo, construindo AFDs ou AFNs, ou utilizando a equivalência entre expressões regulares e autômatos finitos.

Antes de definir formalmente o lema do bombeamento para linguagens regulares, vamos ver intuitivamente como ele funciona. Começamos mostrando um AFD através de um grafo. Neste AFD vamos apenas denotar seus estados, sem mostrar as transições.

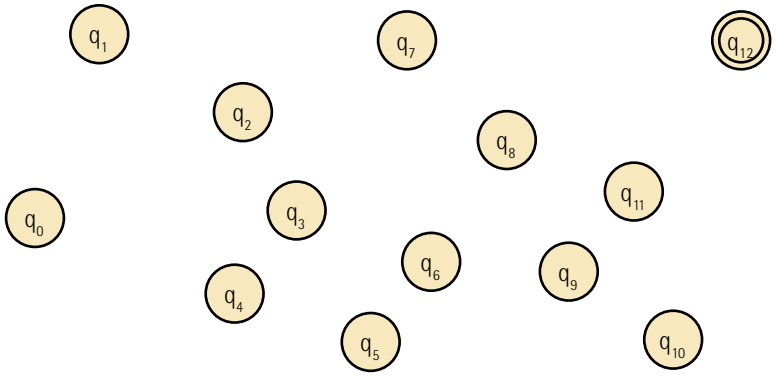

Figura 41 – Representação dos estados de um AFD

Na figura, temos 13 estados, sendo q_0 o estado inicial e q_{12} o estado final.

Devemos agora determinar o tamanho da maior cadeia possível sem repetir estados (sem que o grafo contenha ciclos).

Lembrete

Ciclo: um caminho em que o primeiro e o último vértices são o mesmo.

Obviamente, a maior cadeia será obtida ligando os vértices na sequência e, portanto, tamanho 12, ou seja, |Q| - 1 = 12. Assim:

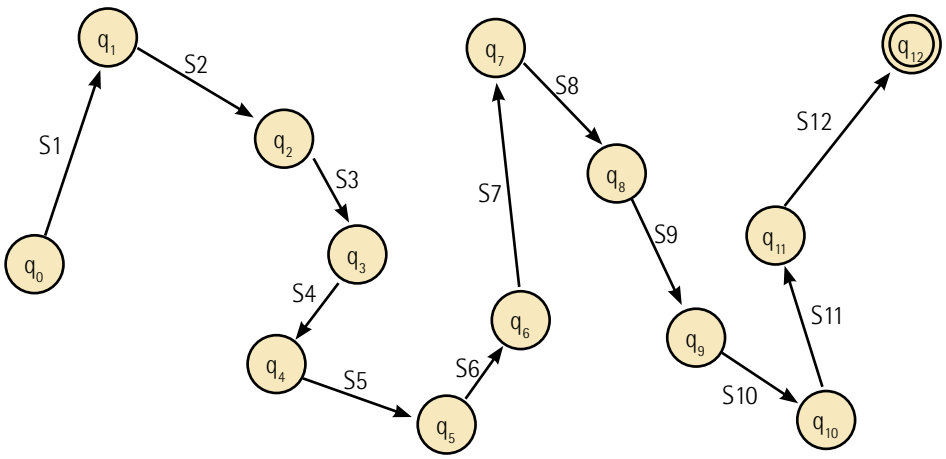

Figura 42 – Representação da maior cadeia sem repetir estados

A relação |Q| - 1 vale para qualquer AFD que não repita estados.

Outra questão sobre a qual devemos refletir é se é possível representar uma linguagem com cadeias tão grandes quanto se queira usando um AFD sem ciclos.

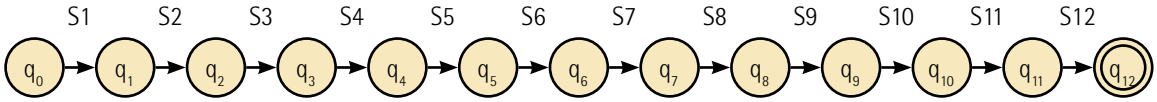

Figura 43 – Representação da maior cadeia na disposição horizontal

Fica claro, observando a figura anterior, que a resposta é que não é possível representar uma cadeia tão grande quanto se queira usando um AFD sem ciclos. Isso porque já vimos que a cadeia representada na figura, que é de comprimento 12, é a maior possível. Portanto, para representar uma cadeia com comprimento maior que o número de estados, é necessário que no AFD haja um ciclo.

Agora, imaginemos um AFD genérico. Para poder representar uma cadeia de tamanho tão grande quanto se queira, vamos inserir um ciclo neste AFD, ligando dois estados distintos quaisquer, e destacaremos o processamento dos símbolos em três partes, representadas por: α, β e γ.

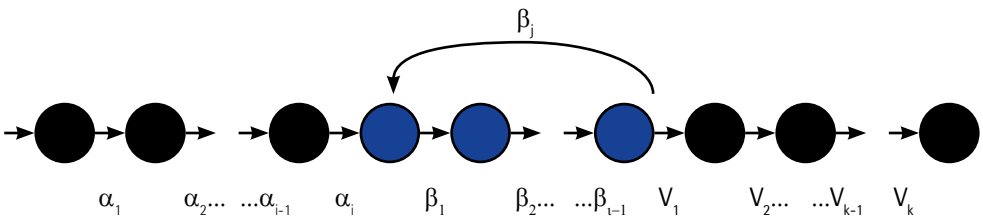

Figura 44 – Representação do AFD genérico com ciclo

Considere que o estado inicial é o vértice mais à esquerda e que o estado final é o vértice mais à direita.

Sabemos que o processamento de cadeias reconhecidas por este AFD pode passar pelo ciclo zero ou mais vezes. Assim, observe uma cadeia que passa zero vezes pelo ciclo:

$$\omega_0 = \alpha_1 \ldots \alpha_i \beta_1 \ldots \beta_{j-1} Y_1 Y_2 \ldots Y_K$$

Agora, a cadeia que passa uma vez pelo ciclo:

$$\omega_1 = \alpha_i \ldots \alpha_{j-1} \alpha (\beta_1 \ldots \beta_j)_1 \beta_1 \ldots \beta_{j-1} Y_1 \ldots Y_K$$

Perceba que a cada nova passagem pelo ciclo insere-se (**bombeia-se**) uma nova subcadeia $\beta_1 \ldots \beta_j$. Assim:

$$\omega_1 = \alpha_1 \ldots \alpha_{i-1} \alpha_i \ldots (\beta_1 \ldots \beta_j)_1 \beta_1 \ldots \beta_{j-1} Y_1 \ldots Y_K$$

$$\omega_2 = \alpha_1 \ldots \alpha_{i-1} \alpha_i \ldots (\beta_1 \ldots \beta_j)_1 (\beta_1 \ldots \beta_j)_2 \beta_1 \ldots \beta_{j-1} Y_1 \ldots Y_K$$

$$\omega_3 = \alpha_1 \ldots \alpha_{i-1} \alpha_i \ldots (\beta_1 \ldots \beta_j)_1 (\beta_1 \ldots \beta_j)_2 (\beta_1 \ldots \beta_j)_3 \beta_1 \ldots \beta_{j-1} Y_1 \ldots Y_K$$

$$\vdots$$

$$\omega_n = \alpha_1 \ldots \alpha_{j-1} \alpha_i \ldots (\beta_1 \ldots \beta_j)_1 (\beta_1 \ldots \beta_j)_2 (\beta_1 \ldots \beta_j)_3 \ldots (\beta_1 \ldots \beta_j)_n \beta_1 \ldots \beta_{j-1} Y_1 \ldots Y_K$$

Outra forma de representar a cadeia ω_n e que facilitará a compreensão da definição formal do lema do bombeamento é:

$$\omega_n = \alpha_1 \ldots \alpha_{i-1} \alpha_i (\beta_1 \ldots \beta_j)^n \beta_1 \ldots \beta_{j-1} Y_1 \ldots Y_K$$

Dessa forma, foi apresentado intuitivamente o conceito do lema do bombeamento, isto é, se existe um AFD em cujo ciclo se pode bombear quantas vezes se queira uma subcadeia, significa que a linguagem reconhecida por esse AFD é regular e infinita.

Resumidamente, o lema do bombeamento diz: é um resultado que estabelece uma relação entre as linguagens regulares e uma determinada propriedade. Ele afirma que toda linguagem regular possui essa propriedade. Portanto, se uma linguagem não possui essa propriedade, ela não é regular.

Contudo, é importante destacar que o fato de uma linguagem ter essa propriedade não implica necessariamente que ela seja regular. A presença da propriedade é uma condição necessária, mas não suficiente para que uma linguagem seja regular.

A forma mais segura de provar que uma linguagem é regular é apresentar um AFD, um AFN ou uma ER que a reconheça. Essas são as formas protegidas de demonstrar a regularidade de uma linguagem de maneira precisa e rigorosa.

Definição e forma do lema do bombeamento

Se L é uma linguagem regular, isso significa que existe um número p, chamado comprimento de bombeamento, tal que para toda cadeia ω pertencente a L ($\omega \in L$) com comprimento $|\omega|$ maior ou igual a p ($|\omega| \geq p$), ela pode ser decomposta na forma $\omega = \alpha\beta y$, obedecendo às seguintes condições:

- β não é uma cadeia vazia ($\beta \neq \varepsilon$);
- o comprimento de $\alpha\beta$ é menor ou igual a p ($|\alpha\beta| \leq p$);
- para qualquer número inteiro não negativo n, a cadeia $\alpha\ \beta^n\ y$ também pertence à linguagem L (se $n \geq 0$, então $\alpha\ \beta^n\ y \in L$).

Essencialmente, o lema do bombeamento para linguagens regulares afirma que, se uma linguagem é regular, então existe um comprimento mínimo a partir do qual é possível bombeá-la repetindo uma parte da cadeia (representada por β), mantendo-a na linguagem.

Vejamos uma aplicação do lema do bombeamento.

Exemplo de aplicação

Aplique o lema do bombeamento para determinar se a linguagem $L = \{\omega / \omega = a^n b^n, n > 0\}$ é não regular.

Resolução

A linguagem L reconhece cadeias formadas pelos símbolos a e b, de modo que a quantidade de símbolos a seja igual à quantidade de símbolos b. O valor de n é indeterminado, acarretando que ele pode assumir qualquer valor, e, portanto, L é infinita.

Sendo L uma linguagem infinita, vamos assumir, por absurdo, que ela é regular e pode ser representada por um AFD que contenha um ciclo. Assim, as cadeias reconhecidas por L devem assumir a forma:

$$\omega_n = \alpha_1 \ldots \alpha_{j-1}\, \alpha_i\, (\beta_1 \ldots \beta_j)^n\, \beta_1 \ldots \beta_{j-1}\, Y_1 \ldots Y_K$$

Assim, para o exemplo, representaremos uma cadeia com o seguinte formato:

$\omega_1 = \alpha_1 \ldots \alpha_i\, (\alpha_1 \ldots \alpha_j)^1\, \alpha_1 \ldots \alpha_{j-1}\, b_1 \ldots b_K$, onde $k = i + j + (j - 1)$, pois a quantidade de símbolos a tem que ser igual à quantidade de símbolos b.

Podemos agora escrever esta cadeia na forma de potência e representá-la através de um grafo que passa uma única vez pelo ciclo, onde o estado inicial é o vértice mais à esquerda e o estado final é o vértice mais à direita:

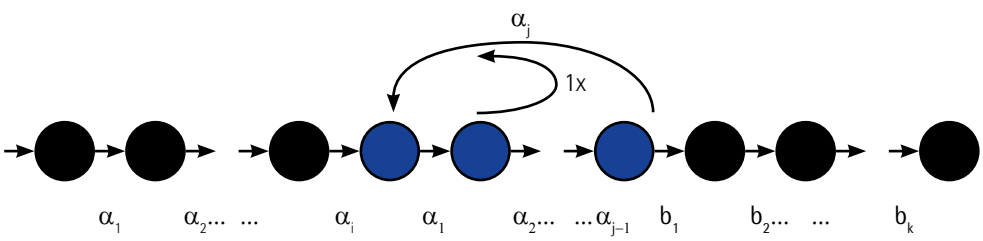

Figura 45

$$\omega_1 = a^i\, a^j\, a^{j-1}\, b^{i+j+(j-1)} = a^{i+j+(j-1)} b^{i+j+(j-1)}$$

Observe que a quantidade de símbolos a é exatamente igual à quantidade de símbolos b.

Representamos agora a cadeia ω_2, através do grafo que passa duas vezes pelo ciclo e vejamos o que ocorre:

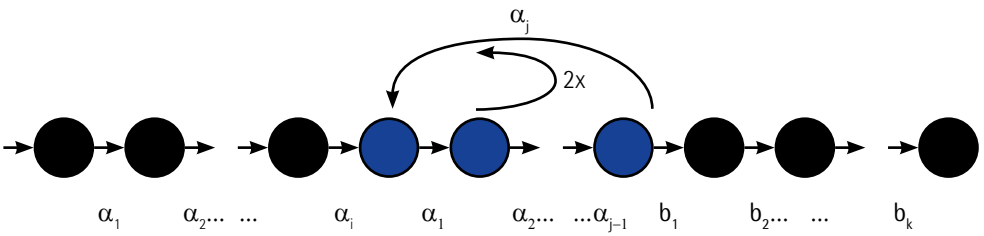

Figura 46

$\omega_2 = a^i a^{2j} a^{j-1} b^{i+j+(j-1)}$

Note agora que, ao passar duas vezes pelo ciclo, aumentamos a quantidade de símbolos a j vezes. Assim, ao chegar no estado final, a quantidade de símbolos a será maior que a quantidade de símbolos b.

O lema do bombeamento propõe que não importa quantas vezes passamos pelo ciclo, a cadeia resultante sempre pertencerá à linguagem. Entretanto, ao passar duas vezes pelo ciclo, a cadeia resultante não é da forma $a^n b^n$; logo, por contradição, L não é regular.

3 LINGUAGENS LIVRES DE CONTEXTO

Este é um conceito importante na teoria das linguagens formais e na computabilidade (que estudaremos no item 4). As linguagens livres de contexto são uma classe de linguagens que podem ser descritas por gramáticas livres de contexto, que são formalismos matemáticos utilizados para definir a estrutura sintática de uma linguagem.

Esse tipo de linguagem possui regras de produção que especificam como construir suas cadeias a partir de símbolos não terminais e terminais. Os símbolos não terminais representam elementos que podem ser expandidos em outras cadeias, enquanto os terminais são símbolos que não podem ser expandidos.

As linguagens livres de contexto têm várias propriedades interessantes. Elas são fechadas sob operações de união, concatenação e fecho de Kleene, o que significa que, ao realizar essas operações entre linguagens livres de contexto, o resultado também será uma linguagem livre de contexto.

As gramáticas livres de contexto e as linguagens livres de contexto têm aplicações em várias áreas, como análise sintática de linguagens de programação, processamento de linguagem natural, compiladores, entre outras. Elas fornecem uma maneira formal de descrever a estrutura de uma linguagem e são fundamentais para entender a natureza das linguagens e sua manipulação computacional.

No estudo das linguagens formais, as linguagens livres de contexto são uma classe amplamente estudada, e compreendê-las é essencial para o desenvolvimento de algoritmos e técnicas que envolvem o processamento de linguagens estruturadas.

3.1 Gramática livre de contexto

Uma gramática livre de contexto é formalmente definida por uma quádrupla G = (V, Σ, R, S), em que:

V é um conjunto finito de símbolos não terminais.

Σ é um conjunto finito de símbolos terminais, onde $V \cap \Sigma = \emptyset$.

R é um conjunto finito de regras de produção, em que cada regra é da forma A -> α, onde A é um símbolo não terminal e α é uma cadeia de símbolos terminais ou não terminais em que $\alpha \in (\Sigma \cup V)^*$.

S é o símbolo inicial da gramática, na qual S pertence a V.

Formalmente, uma derivação em uma gramática livre de contexto é uma sequência finita de substituições de símbolos, começando com o símbolo inicial S e aplicando as regras de produção para então substituir os não terminais até que se obtenha uma cadeia de símbolos terminais.

Uma linguagem gerada por uma gramática livre de contexto é o conjunto de todas as cadeias que podem ser derivadas do símbolo inicial S.

Exemplo: a gramática geradora da linguagem $L = \{\omega \mid \omega = a^n b^n, n \geq 0\}$ é G = (V, Σ, R, S), onde:

V = {S} (conjunto de símbolos não terminais)

Σ = {a, b} (alfabeto composto de símbolos terminais)

R = {S → aSb | ε} (conjunto de regras de produção)

S é o símbolo não terminal inicial da derivação

Determine se a linguagem L é livre de contexto:

Antes de iniciarmos a resolução, é importante relembrar que na expressão S → aSb | ε o símbolo | significa que S pode gerar aSb ou S pode gerar ε.

Vamos então às regras de produção e verificaremos se é possível gerar cadeias onde o número de símbolos a seja igual ao número de símbolos b.

S ⇒ ε

S ⇒ aSb ⇒ aεb = ab

S ⇒ aSb ⇒ aaSbb ⇒ aaεbb = aabb

S ⇒ aSb ⇒ aaSbb ⇒ aaaSbbb ⇒ aaaεbbb = aaabbb

Portanto, L = {ε, ab, aabb, aaabbb, aaaabbbb, ...}

Como a regra de produção R permitiu a formação de cadeias em que a quantidade de símbolos a é igual à quantidade de símbolos b, verifica-se que L é uma linguagem livre de contexto.

Observação

Ao efetuarmos as derivações, devemos utilizar a seta dupla $\Rightarrow$.

Exemplo: verifique se a gramática G_2 gera uma linguagem livre de contexto.

Dados:

$G_2 = (V, \Sigma, R, S)$, onde:

$V = \{S\}$ (conjunto de símbolos não terminais)

$\Sigma = \{a, b\}$ (alfabeto composto de símbolos terminais)

$R = \{S \rightarrow aSa \mid bSb \mid a \mid b \mid \varepsilon\}$ (conjunto de regras de produção)

S é o símbolo não terminal inicial da derivação

Pode-se verificar se G_2 é ou não uma gramática livre de contexto se as produções coadunam com a definição.

1ª produção $S \Rightarrow aSb$

Neste caso, temos um símbolo não terminal S que pertence a V gerando símbolos terminais (a), o que é permitido pelo item 3 da definição.

2ª produção $S \Rightarrow bSb$

É o mesmo caso da primeira produção: um símbolo não terminal S que pertence a V gera símbolos terminais (b), o que é permitido pelo item 3 da definição.

3ª produção $S \Rightarrow a$

Neste caso, a produção gerou somente um símbolo terminal (a), e o item 3 da definição diz que o símbolo não terminal $a \in (\Sigma \cup V)^*$. Logo, esta produção é válida, pois $(\Sigma \cup V)^*$ significa que S pode gerar somente símbolos terminais.

4ª produção $S \Rightarrow b$

Mesmo caso da terceira produção; logo, é válida.

5ª produção $S \Rightarrow \varepsilon$

O símbolo não terminal ε é uma produção válida, pois, na expressão $\alpha \in (\Sigma \cup V)^*$, a operação * define que são válidas todas as combinações de cadeias possíveis dos símbolos a e b, inclusive a cadeia vazia ε.

Portanto, G_2 é uma gramática livre de contexto.

Exemplo: qual é a linguagem gerada por G_2?

Para descobrir a linguagem, basta efetuarmos as derivações das regras de produção:

$S \Rightarrow \varepsilon$

$S \Rightarrow a$

$S \Rightarrow b$

$S \Rightarrow aSa \Rightarrow a\varepsilon a = aa$

$S \Rightarrow bSb \Rightarrow b\varepsilon b = bb$

$S \Rightarrow aSa \Rightarrow abSba \Rightarrow abaSaba \Rightarrow aba\varepsilon aba = abaaba$

$S \Rightarrow bSb \Rightarrow baSab \Rightarrow babSbab \Rightarrow bab\varepsilon bab = babbab$

Obviamente, é possível fazer infinitas derivações. Aqui estão apenas alguns exemplos que demonstram que, independentemente das derivações efetuadas, as palavras formadas têm a característica de apresentarem a mesma leitura da esquerda para a direita ou da direita para a esquerda. Palavras com essa característica são denominadas palíndromos.

Logo, a linguagem gerada por G_2 é $L(G_2) = \{\omega \in \Sigma / \omega = \omega^R\}$, ou seja, qualquer cadeia ω é igual à sua reversa.

Exemplo: as regras 1 a 6 pertencem ao conjunto de produções R de uma gramática G=({E,T,F}, {id, +,*, ( )}, P, E).

1) $E \rightarrow E + T$

2) $E \rightarrow T$

3) $T \rightarrow T * F$

4) $T \rightarrow F$

5) $F \rightarrow (E)$

6) $F \rightarrow id$

Mostre que G gera a cadeia (id * id + id) * (id + id).

O símbolo E é o símbolo inicial da gramática, portanto iniciamos as derivações por ele:

$E \Rightarrow T$ (regra de produção 2)

$T \Rightarrow T * F$ (regra de produção 3)

$T * F \Rightarrow F * F$ (regra de produção 4)

$F * F \Rightarrow (E) * F$ (regra de produção 5)

$(E) * F \Rightarrow (E + T) * F$ (regra de produção 1)

$(E + T) * F \Rightarrow (T + T) * F$ (regra de produção 2)

$(T + T) * F \Rightarrow (T * F + T) * F$ (regra de produção 3)

$(T * F + T) * F \Rightarrow (F * F + T) * F$ (regra de produção 4)

$(F * F + T) \Rightarrow (id * id + T) * F$ (regra de produção 6)

$(id * id + T) * F \Rightarrow (id * id + F) * F$ (regra de produção 4)

$(id * id + F) * F \Rightarrow (id * id + id) * F$ (regra de produção 6)

$(id * id + id) * F \Rightarrow (id * id + id) * (E)$ (regra de produção 5)

$(id * id + id) * (E) \Rightarrow (id * id + id) * (E + T)$ (regra de produção 1)

$(id * id + id) * (E + T) \Rightarrow (id * id + id) * (T + T)$ (regra de produção 2)

$(id *id + id) * (T + T) \Rightarrow (id * id + id) * (F + T)$ (regra de produção 4)

$(id *id + id) * (F + T) \Rightarrow (id * id + id) * (id + T)$ (regra de produção 6)

(id *id + id) * (id + T) $\Rightarrow$ (id * id + id) * (id + F) (regra de produção 4)

(id *id + id) * (id + F) $\Rightarrow$ (id * id + id) * (id + id) (regra de produção 6)

Esse é um exemplo prático de aplicação das linguagens livres de contexto na representação e análise de expressões aritméticas simples.

Árvores de derivação

Uma árvore de derivação é uma representação gráfica usada para visualizar o processo de derivação de uma cadeia de símbolos em uma linguagem livre de contexto, de acordo com as regras de produção de uma gramática livre de contexto. Ela ilustra como a cadeia pode ser seguida a partir do símbolo inicial da gramática, aplicando as regras de produção sequencialmente.

Cada nó da árvore de derivação representa um símbolo da cadeia sendo derivado durante o processo. O nó raiz é rotulado com o símbolo inicial da gramática, e os nós filhos são rotulados com os símbolos obtidos após a aplicação das regras de produção. Essa estrutura em árvore mostra uma sequência de derivações, da esquerda para a direita, que leva do símbolo inicial até a cadeia final.

Vamos considerar uma gramática G = (V, Σ, R, S) e uma cadeia ω pertencente à linguagem gerada por G. A árvore de derivação de ω é construída da seguinte forma:

- o nó raiz é rotulado com o símbolo inicial S;
- para cada regra de produção que pode ser aplicada ao não terminal atual, criamos um ramo na árvore;
- substituímos o não terminal atual por sua produção correspondente na regra escolhida, criando um nó filho para cada símbolo na cadeia produzida.

Esse processo é repetido até que todos os símbolos não terminais tenham sido substituídos por cadeias compostas apenas de símbolos terminais. A árvore de derivação resultante mostrará o caminho completo de derivação da cadeia ω, desde o símbolo inicial até a cadeia final formada pelos símbolos terminais.

A árvore de derivação é uma representação útil para entender a estrutura sintática de uma cadeia e como ela é derivada da gramática. Ela fornece uma visão passo a passo do processo de derivação, o que facilita a análise da estrutura gramatical e pode ajudar a detectar erros ou ambiguidades na gramática.

As árvores de derivação também são usadas em análise sintática de linguagens de programação, onde são conhecidas como árvores de análise sintática ou árvores de análise de sintaxe abstrata (AST). Nesse contexto, a árvore de derivação é construída para representar a estrutura hierárquica da expressão ou instrução do programa, o que permite uma análise mais eficiente e precisa da estrutura sintática do código-fonte.

Exemplo: seja a gramática G = ({S, B}, {a, b}, P, S), onde: P = {S → aB | aSB; B → b}. A derivação para a cadeia aabb é:

S ⇒ aSB ⇒ aaBB ⇒ aabB ⇒ aabb

A árvore de derivação desta cadeia é representada na figura a seguir:

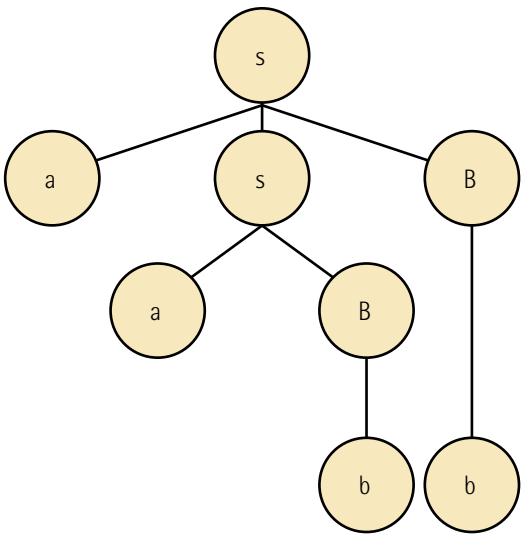

Figura 47

Ambiguidade das gramáticas livres de contexto

Uma gramática livre de contexto é ambígua quando existem duas ou mais sequências de substituições que podem gerar a mesma cadeia a partir do símbolo inicial da gramática. Essas diferentes derivações podem levar a diferentes árvores de derivação, cada uma representando uma análise sintática alternativa da mesma cadeia.

A ambiguidade pode ser problemática em algumas aplicações de processamento de linguagem natural e análise sintática. Por exemplo, ao projetar um compilador, a ambiguidade pode dificultar a geração de uma árvore de análise única e coerente para uma determinada cadeia, o que pode levar a resultados incorretos na compilação.

A ambiguidade pode surgir em gramáticas por causa de diferentes regras de produção que podem ser aplicadas para derivar uma mesma cadeia ou devido à falta de especificidade nas regras de produção. Além disso, o uso de recursão à esquerda ou produções vazias também pode contribuir para uma ambiguidade.

Exemplo: considere a gramática G = ({E}, {x, +, *}, R, E), onde R = {E → E + E | E * E | x}. A cadeia x + x * x é gerada pela gramática G. Tem-se que:

$E \Rightarrow E + E$

$E + E \Rightarrow x + E$

$x + E \Rightarrow x + E * E$

$x + E * E \Rightarrow x + x * E$

$x + x * E \Rightarrow x + x * x$

A árvore de derivação correspondente é:

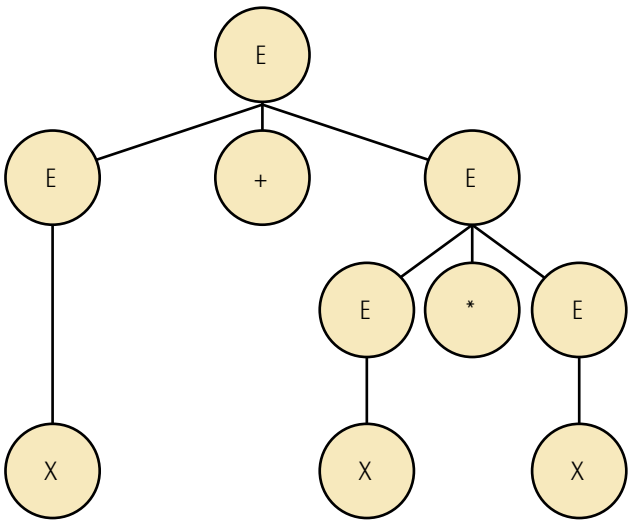

Figura 48

Por outro lado, a mesma sentença x + x * x pode ser derivada:

$E \Rightarrow E * E$

$E * E \Rightarrow E + E * E$

$E + E * E \Rightarrow x + E * E$

$x + E * E \Rightarrow x + x * E$

$x + x * E \Rightarrow x + x * E$

$x + x * E \Rightarrow x + x * x$

A árvore de derivação correspondente é:

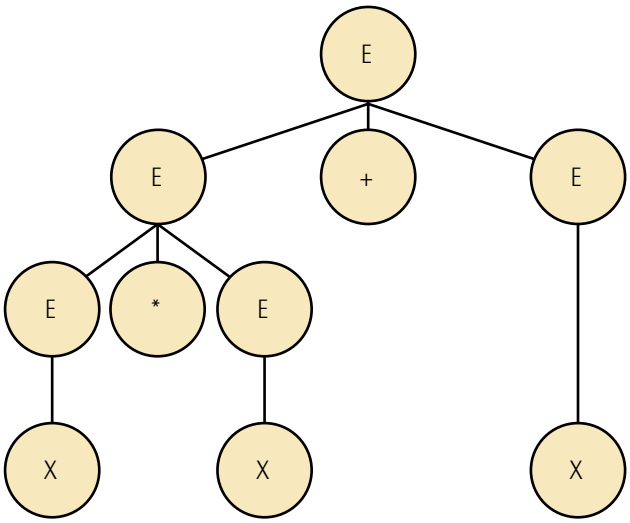

Figura 49

3.2 Autômatos de pilha

Os autômatos de pilha, ou autômatos de pilha não determinísticos (APND), são uma extensão dos autômatos finitos e possuem uma pilha como memória adicional. Entenda a pilha do APND como uma pilha de pratos, blocos ou qualquer outro objeto, ou seja, como uma fila vertical, na qual só temos acesso ao objeto que está no topo. Os APNDs desempenham um papel fundamental na teoria da computação e linguagens formais, sendo especialmente relevantes para compreender e analisar linguagens livres de contexto.

O APND é o dispositivo reconhecedor das linguagens livres de contexto. É importante destacar que os AFDs têm uma capacidade de reconhecimento limitada, abrindo apenas um subconjunto das linguagens livres de contexto.

O APND é composto por diferentes elementos:

- A fita de entrada, que é análoga ao AFD e armazena a cadeia de entrada a ser ou não reconhecida. Essa fita é dividida em células, cada uma correspondente a um símbolo da cadeia de entrada. A leitura da fita ocorre do lado esquerdo para o lado direito, e os símbolos nela presentes não podem ser modificados, permitindo apenas operações de leitura.

- A unidade de controle, que possui um número finito e pré-definido de estados. As transições de estado ocorrem com base nas especificações definidas por uma relação de transição. Essa relação associa o estado atual do autômato a um novo estado, sendo a mudança de estado determinada pela leitura do símbolo corrente da cadeia de entrada (que pode ser a palavra vazia ε) seguida pela leitura e remoção (que pode não ocorrer, a depender do símbolo presente no topo da pilha) de um símbolo no topo da pilha. Essas transições podem implicar a inserção de um novo símbolo

na pilha. Além disso, os símbolos a serem lidos da cadeia de entrada, do topo da pilha e gravados nessa estrutura de dados, podem coincidir com uma cadeia vazia (ε).

- A memória auxiliar organizada em pilha, na qual os símbolos a serem consultados, removidos e armazenados não necessariamente coincidem com os símbolos do alfabeto de entrada. Os símbolos da pilha possuem um alfabeto próprio.

O APND é capaz de lidar com a memória adicionalmente fornecida pela pilha, permitindo linguagens mais complexas, como as linguagens livres de contexto.

Recordando que a linguagem $L = \{\omega \mid \omega = a^n b^n, n \geq 0\}$ é livre de contexto, observe as figuras a seguir:

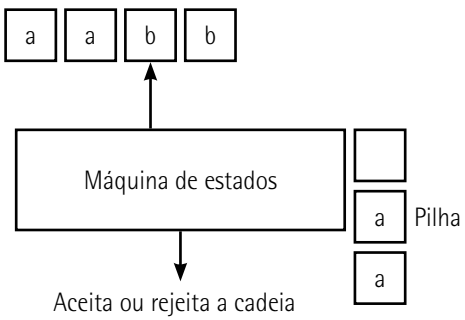

Figura 50 – APND: conteúdo da pilha após a leitura da subcadeia aa

Na figura anterior, após a leitura de cada um dos símbolos a, foi inserido o símbolo A na pilha.

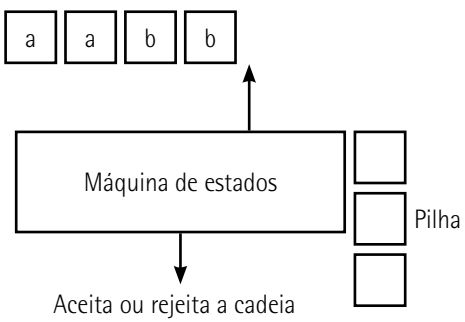

Figura 51 – APND: conteúdo da pilha após a leitura da cadeia aabb

Note, na figura anterior, que a cada leitura do símbolo b foi retirado da pilha, respectivamente, cada um dos símbolos A. Isso significa que o APND, através da memória auxiliar (pilha), contou a quantidade de símbolos a e b, reconhecendo a cadeia aabb como pertencente à linguagem $L = \{\omega \mid \omega = a^n b^n, n \geq 0\}$.

Dessa forma, as transições em um APND funcionam da seguinte maneira:

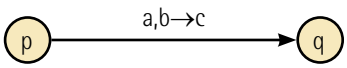

Figura 52

Sendo o estado atual p, leia o símbolo a do alfabeto de entrada, desempilhe o símbolo b do alfabeto da pilha (somente se b estiver no topo da pilha) e empilhe o símbolo c do alfabeto da pilha. Caso b não esteja no topo da pilha, esta transição não será realizada.

Também podem ocorrer as seguintes situações:

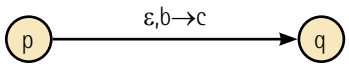

Figura 53

Caso ε pertença ao alfabeto de entrada, significa que apenas vamos desempilhar, se possível, b e empilhar c.

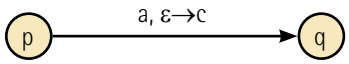

Figura 54

Nesta posição de ε, significa que não desempilhamos nenhum símbolo e empilhamos c.

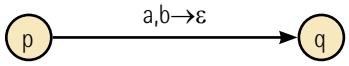

Figura 55

Por fim, esta transição indica que não empilharemos nenhum símbolo.

A definição formal de um autômato de pilha é dada por uma sêxtupla $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$, onde:

Q é um conjunto finito de estados.

Σ é o alfabeto de entrada, ou seja, o conjunto de símbolos de entrada que a máquina lê.

Γ é o alfabeto da pilha, ou seja, o conjunto de símbolos que podem ser colocados na pilha.

δ é a função de transição, dada por $Q \times (\Sigma \cup \{\varepsilon\}) \times (\Gamma \cup \{\varepsilon\}) \rightarrow P(Q \times (\Gamma \cup \{\varepsilon\}))$.

$q_0 \in Q$ é o estado inicial.

$F \subseteq Q$ é o conjunto de estados finais.

Vale aqui uma explicação sobre a função de transição δ. Como se trata de uma função, obrigatoriamente existe o conjunto domínio de partida e o conjunto contradomínio de chegada (reveja o item 1.3 deste livro-texto). O domínio é representado por $Q \times (\Sigma \cup \{\varepsilon\}) \times (\Gamma \cup \{\varepsilon\})$, que significa o produto cartesiano entre o estado atual (que pertence a Q), o símbolo a ser lido (que pertence a $\Sigma \cup \{\varepsilon\}$) e o símbolo no topo

da pilha (que pertence a $\Gamma \cup \{\varepsilon\}$), determinando a sequência de movimentos do APND. Esta tripla resulta no contradomínio, que é o conjunto potência (reveja o item 1.2 deste livro-texto) do produto cartesiano entre o conjunto de estados Q e os símbolos do alfabeto da pilha Γ , incluindo a palavra vazia ε.

Portanto, como exemplo, seja a função de transição $\delta\ (q_2,x,Y) \rightarrow \{q_3,Z)$ válida para uma determinada cadeia de entrada, lemos δ da seguinte maneira: estando no estado atual q_2, lendo o símbolo do alfabeto de entrada x, desempilhamos Y, avançamos para o estado q_3 e empilhamos Z.

Agora que vimos detalhadamente como um APND reconhece uma gramática livre de contexto, vejamos alguns exemplos.

Exemplo de aplicação

Determine se o APND representado pelo grafo a seguir reconhece as seguintes cadeias de entrada:

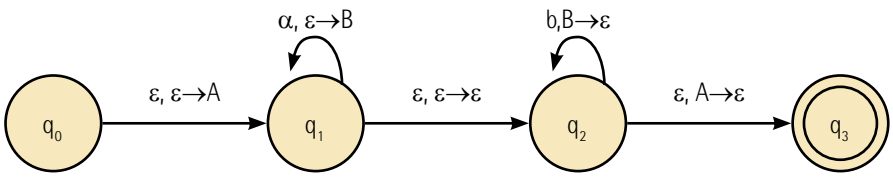

Figura 56

a) aaabbb

Resolução

Para a leitura de qualquer cadeia, o APND inicializa no estado inicial q_0. Neste estado, a função de transição $\varepsilon\varepsilon \rightarrow A$ determina que, lendo nenhum símbolo da cadeia de entrada, não se desempilha nenhum símbolo da pilha, empilha-se o símbolo A, avança-se para o q_1.

Figura 57

Como o autômato de pilha é não determinístico, note que em q_1 há duas transições possíveis. Todavia, como queremos ler a cadeia aaabbb, optaremos pela transição $a\varepsilon \rightarrow R$. Desse modo, estando o APND no estado q_1, lendo o primeiro símbolo a da cadeia de entrada, não se desempilha nenhum símbolo da pilha, empilha-se o símbolo B e permanece-se em q_1.

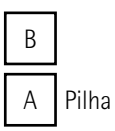

Figura 58

Ainda na transição aε→R, estando o APND no estado q_1, lendo o segundo símbolo a da cadeia de entrada, não se desempilha nenhum símbolo da pilha, empilha-se o símbolo B e permanece-se em q_1.

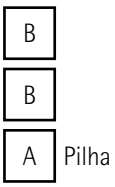

Figura 59

Ainda na transição aε→R, estando o APND no estado q_1, lendo o terceiro símbolo a da cadeia de entrada, não se desempilha nenhum símbolo da pilha, empilha-se o símbolo B e permanece-se em q_1.

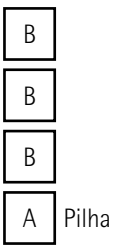

Figura 60

Lidos todos os símbolos a, podemos usar a transição ε,ε. Dessa forma, estando o APND no estado q_1, não lendo nenhum símbolo da cadeia de entrada, não desempilhando nenhum símbolo da pilha, não empilhando nenhum símbolo, avança-se para o estado q_2.

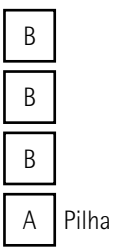

Figura 61

Note que em q_2 temos duas transições possíveis. Entretanto, como queremos ler os três símbolos b restantes da cadeia de entrada, optaremos pela transição b,B→ε. Desse modo, estando o APND no estado q_2, lendo o primeiro símbolo b da cadeia de entrada, desempilha-se o símbolo B, não se empilha nenhum símbolo e permanece-se em q_2.

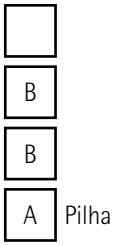

Figura 62

Ainda na transição b,B→ε, estando o APND no estado q_2, lendo o segundo símbolo b da cadeia de entrada, desempilha-se o símbolo B, não se empilha nenhum símbolo e permanece-se em q_2.

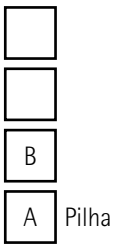

Figura 63

Ainda na transição b,B →ε, estando o APND no estado q_2, lendo o terceiro símbolo b da cadeia de entrada, desempilha-se o símbolo B, não se empilha nenhum símbolo e permanece-se em q_2.

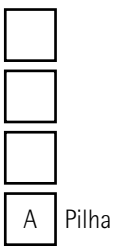

Figura 64

Vamos agora para a outra transição existente em q_2, εA→ε. Estando o APND no estado q_2 e lendo nenhum símbolo da cadeia de entrada, desempilha-se o símbolo A, não se empilha nenhum símbolo e avança-se para o estado final q_3. Como a cadeia foi lida em sua totalidade, o APND está no estado final e a pilha está vazia, conclui-se que o APND aceitou (reconheceu) a cadeia de entrada aaabbb.

b) aaabb

Resolução

Neste caso, faremos inicialmente a leitura do símbolo igual ao item anterior, visto que as duas cadeias possuem os símbolos aaa em seu início.

Estando o APND no estado inicial q0, a função de transição ε,ε→A determina que, lendo nenhum símbolo da cadeia de entrada, não se desempilha nenhum símbolo da pilha, empilha-se o símbolo A e avança-se para o q_1.

Figura 65

Em q_1, iniciamos a leitura do primeiro a da cadeia de entrada aaabb, optando pela transição a,ε→B. Desse modo, estando o APND no estado q_1, lendo o primeiro símbolo a da cadeia de entrada, não se desempilha nenhum símbolo da pilha, empilha-se o símbolo B e permanece-se em q_1.

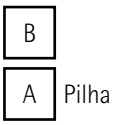

Figura 66

Ainda na transição a,ε→B, estando o APND no estado q_1, lendo o segundo símbolo a da cadeia de entrada, não se desempilha nenhum símbolo da pilha, empilha-se o símbolo B e permanece-se em q_1.

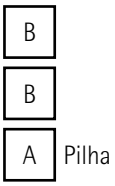

Figura 67

Ainda na transição a,ε→B, estando o APND no estado q_1, lendo o terceiro símbolo a da cadeia de entrada, não se desempilha nenhum símbolo da pilha, empilha-se o símbolo B e permanece-se em q_1.

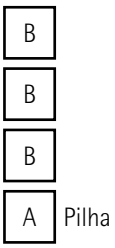

Figura 68

Lidos todos os símbolos a, podemos usar a transição ε,ε→ε. Dessa forma, estando o APND no estado q_1, não lendo nenhum símbolo da cadeia de entrada, não desempilhando nenhum símbolo da pilha, não empilhando nenhum símbolo, avança-se para o estado q_2.

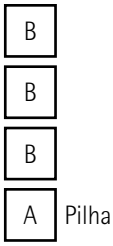

Figura 69

Iniciamos neste momento a leitura dos dois símbolos b, e optaremos pela transição b,B→ε. Desse modo, estando o APND no estado q_2, lendo o primeiro símbolo b da cadeia de entrada, desempilha-se o símbolo B, não se empilha nenhum símbolo e permanece-se em q_2.

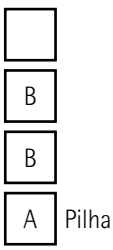

Figura 70

Ainda na transição b,B→ε, estando o APND no estado q_2, lendo o segundo símbolo b da cadeia de entrada, desempilha-se o símbolo B, não se empilha nenhum símbolo e permanece-se em q_2.

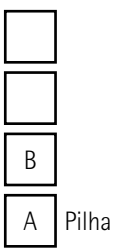

Figura 71

Note que a leitura da cadeia de entrada se encerrou, mas o APND está no estado q_2, que não é o estado final, e a pilha não está vazia. Logo, o APND para e rejeita a cadeia aaabb.

c) abab

Resolução

Com o APND no estado inicial q0, a função de transição ε,ε→A determina que, lendo nenhum símbolo da cadeia de entrada, não se desempilha nenhum símbolo da pilha, empilha-se o símbolo A, avança-se para o q_1.

Figura 72

Como o autômato de pilha é não determinístico, note que em q_1 há duas transições possíveis. Todavia, como queremos ler a cadeia abab, optaremos pela transição $a,\varepsilon \rightarrow B$. Desse modo, estando o APND no estado q_1, lendo o primeiro símbolo a da cadeia de entrada, não se desempilha nenhum símbolo da pilha, empilha-se o símbolo B e permanece-se em q_1.

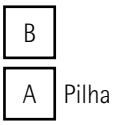

Figura 73

Agora, como devemos ler o primeiro símbolo b, optamos pela transição $\varepsilon,\varepsilon \rightarrow \varepsilon$. Assim, com o APND no estado q_1, avançaremos à q_2 sem ler, desempilhar ou empilhar nenhum símbolo.

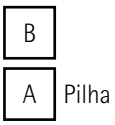

Figura 74

Com o APND no estado q_2, lendo o primeiro símbolo b da cadeia de entrada, a transição $b,B \rightarrow \varepsilon$ determina que se desempilha o símbolo B, não se empilha nenhum símbolo e permanece-se em q_2.

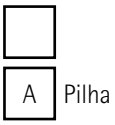

Figura 75

Neste momento, temos que proceder à leitura do segundo símbolo a. Entretanto, nenhuma das transições disponíveis para o estado q_2, ou seja, $b,B \rightarrow \varepsilon$ e $\varepsilon,A \rightarrow \varepsilon$, possibilitam a leitura desse símbolo. Logo, o APND para e rejeita a cadeia abab.

3.3 Lema do bombeamento para linguagens livres de contexto

Antes de entrarmos no tema deste tópico, devemos nos recordar que uma linguagem só é livre de contexto se existir uma gramática que a gere ou um autômato de pilha que a reconheça.

Lembramos ainda que uma gramática livre de contexto é da forma G = (V, ∑, R, S), e que uma derivação nessa gramática é uma sequência finita de substituições de símbolos, começando com o símbolo inicial S e aplicando as regras de produção para substituir os não terminais até que se obtenha uma cadeia de símbolos terminais.

Lembrete

Na forma G = (V, ∑, R, S):

V é um conjunto finito de símbolos não terminais.

∑ é um conjunto finito de símbolos terminais, onde $V \cap \sum = \emptyset$.

R é um conjunto finito de regras de produção, em que cada regra é da forma A -> α, onde A é um símbolo não terminal e α é uma cadeia de símbolos terminais ou não terminais em que $\alpha \in (\sum \cup V)^*$.

S é o símbolo inicial da gramática, na qual S pertence a V.

Assim como para as linguagens regulares, o lema do bombeamento para linguagens livres de contexto serve como uma ferramenta para mostrar que certas linguagens não são livres de contexto.

Vejamos a seguinte situação problema: é possível determinar uma gramática livre de contexto ou um APND para a linguagem $L = \{\{\omega / \omega = a^n b^n c^n / n \geq 0\}$.

Observa-se facilmente que a linguagem L é composta por cadeias com qualquer quantidade dos símbolos a, b e c, nesta ordem, desde que as quantidades de a, de b e de c sejam iguais.

Nos tópicos anteriores, estudamos que para a linguagem do tipo $\{\omega \mid \omega = a^n b^n, n \geq 0\}$ existe uma APND que, para cada símbolo a lido, era empilhado, por exemplo, um símbolo A, e que para cada símbolo b lido se desempilhava o símbolo A até que a pilha se esvaziasse e, assim, se garantia que as quantidades de a e b eram iguais.

Entretanto, para a linguagem $L = \{\{\omega / \omega = a^n b^n c^n / n \geq 0\}$, assim que um APND iniciar a leitura dos símbolos c, a pilha estará vazia e não será possível garantir que a quantidade dos três símbolos seja a mesma. Isso acontece porque a linguagem L não é livre de contexto. Dessa forma, é impossível determinar uma gramática que a gere ou um APND que a reconheça.

Contudo, não basta dizer que L não é livre de contexto: devemos tirar a prova. Como para as linguagens regulares, que dizem que se existe um AFD em cujo ciclo se pode bombear quantas vezes se queira uma subcadeia, as novas palavras sempre serão reconhecidas por esse AFD, de forma semelhante podemos dizer das linguagens livres de contexto em relação a um APND – mas com algumas diferenças, que veremos a seguir.

Considere a gramática G = (V, Σ, R, S), onde:

V = {S, A, B} (conjunto de símbolos não terminais)

$\sum$ = {a, b} (alfabeto composto de símbolos terminais)

R = {S $\rightarrow$ aAa|aSa; A $\rightarrow$ bBb|bAb; B $\rightarrow$ cBc|cxc} (conjunto de regras de produção)

S é o símbolo inicial da gramática

Determine para essa gramática uma palavra de tamanho 7, ou seja, $|\omega| = 7$, sem repetir variáveis (símbolos não terminais).

Obrigatoriamente, devemos iniciar a derivação pelo símbolo S.

S $\Rightarrow$ aAa $\Rightarrow$ abBba abcxcba

Sob outra perpectiva, montemos a árvore de derivação da cadeia abcxcba:

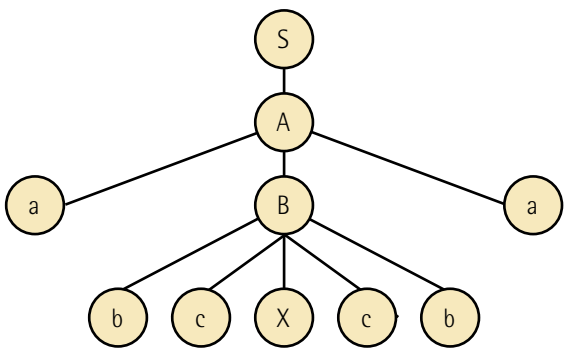

Figura 76

Será possível para essa mesma gramática determinar uma palavra de tamanho 8 ou maior sem repetir variáveis? Observando as regras de derivação e a árvore para esta gramática, vê-se que é impossível, pois a palavra de tamanho 7 é o limite para não se repetir um símbolo não terminal.

Vejamos agora uma derivação para uma gramática qualquer a partir do símbolo inicial T.

Como vimos, em algum momento teremos que repetir variáveis. Assim, iniciamos a derivação a partir de T. Em alguma fase da derivação, repetimos o símbolo não terminal R, formando uma certa palavra que pode ser dividida em cinco partes, $\alpha\beta$y λ μ, conforme a imagem a seguir.

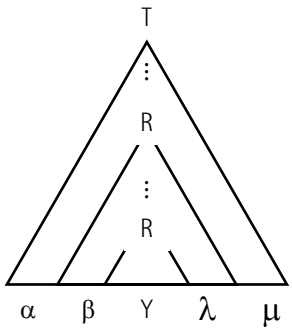

Figura 77

Todavia, como houve repetição de variável (símbolo não terminal R), nada impede que a repitamos quantas vezes quisermos. Assim, podemos substituir a repetição da derivação denotada pelo triângulo menor pela derivação do triângulo intermediário, conforme as imagens a seguir.

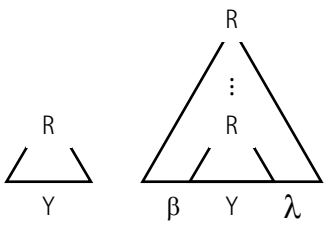

Figura 78

Isso resulta na seguinte derivação:

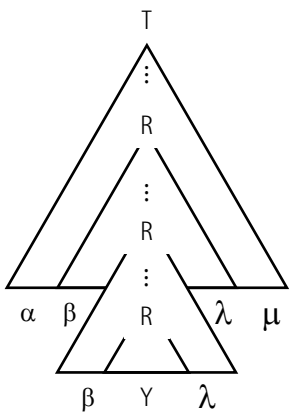

Figura 79

O fato de repetir variáveis é o que chamamos de bombeamento.

Sipser (2007) define formalmente o lema do bombeamento para linguagens livres de contexto da seguinte forma: se L é uma linguagem livre de contexto, isso significa que existe um número p, chamado

comprimento de bombeamento, tal que toda cadeia ω pertencente a L ($\omega \in L$) com comprimento $|\omega|$ maior ou igual a p ($|\omega| \geq p$) pode ser decomposta na forma $\omega = \alpha\beta\gamma\lambda\mu$, obedecendo às seguintes condições:

- $\beta\lambda$ não é uma cadeia vazia ($\beta\lambda \neq \varepsilon$);
- o comprimento de $\beta\gamma\lambda$ é menor ou igual a p ($|\beta\gamma\lambda| \leq p$);

para qualquer número inteiro não negativo n, a cadeia $\alpha\beta^n\gamma\lambda^n\mu$ também pertence à linguagem L (se $n \geq 0$, então, $\alpha\beta^n\gamma\lambda^n\mu \in L$.

Exemplo: dada a cadeia ω = 0000011111, dividida em cinco partes, $\alpha\beta\gamma\lambda\mu$, conforme as tabelas a seguir. Vejamos como ω se comporta nos bombeamentos:

00	0	00	11	111
α	β	γ	λ	μ

$\alpha\beta^2\gamma\lambda^2\mu = \underbrace{00}_{\alpha}\underbrace{00}_{\beta^2}\underbrace{00}_{\gamma}\underbrace{1111}_{\lambda^2}\underbrace{111}_{\mu}$

0	000	011	111
α	γ	λ	μ

$\beta = \varepsilon$

$\alpha\beta^2\gamma\lambda^2\mu = \underbrace{0}_{\alpha}\underbrace{000}_{\gamma}\underbrace{011011}_{\lambda^2}\underbrace{111}_{\mu}$

000	00	11111
β	λ	μ

$\alpha = \gamma = \varepsilon$

$\alpha\beta^0\gamma\lambda^0\mu = \underbrace{11111}_{\mu}$

Exemplo de aplicação

Mostre que a linguagem $L = \{\{\omega / \omega = a^n b^n c^n / n \geq 0\}$ não é livre de contexto.

Resolução

Da mesma forma que para linguagens regulares, por contradição, suponha que L é livre de contexto.

Se L é livre de contexto, então vale o lema. Seja p o valor dado pelo lema e $\omega = a^p b^p c^p$. Como $|\omega| \geq p$, ω pode ser escrita como tal $\alpha\beta\gamma\lambda\mu$, que valem (1), (2) e (3). Assim, temos:

$\overbrace{aa...a}^{p}\overbrace{bb...b}^{p}\overbrace{cc...c}^{p}$

Perceba que, caso $\beta\lambda$ seja cadeia vazia ou o comprimento de β y λ seja maior que p, ou, por fim, para qualquer número inteiro não negativo n a cadeia $\alpha\beta^n y\lambda^n \mu$ resultante não for da forma $a^n b^n c^n$, então L não é livre de contexto.

Para iniciarmos propriamente a resolução, analisaremos os casos em que β y λ contenha somente símbolos a, somente símbolos b, somente símbolos c ou qualquer combinação de símbolos ab e bc. Isso porque, no caso de β y λ conter os três símbolos (a, b, c), ele terá comprimento maior que p, o que, pela condição 2 do lema, não pode ocorrer.

Assim, qualquer divisão em $|\beta y \lambda| \leq p$ e $\beta \lambda \neq \varepsilon$ acarretará $\beta y \lambda$ no máximo dois símbolos diferentes. Neste caso, há duas possibilidades:

Se β e λ têm apenas um símbolo cada:

$\beta = a^k$ e $\lambda = a^j \mid b^j \mid \varepsilon$

$\beta = b^k$ e $\lambda = b^j \mid c^j \mid \varepsilon$

$\beta = c^k$ e $\lambda = c^j \mid \varepsilon$

$\beta = \varepsilon$ e $\lambda = a^j \mid b^j \mid c^j$

Com k e j ≥ 1. Em qualquer caso, $\alpha\beta^2 y\lambda^2\mu$ terá quantidades diferentes dos três símbolos.

Se β e λ têm dois símbolos:

$\beta = a^k b^j$ e $\lambda = b^t \mid \varepsilon$

$\beta = b^k c^j$ e $\lambda = c^t \mid \varepsilon$

$\beta = a^k \mid \varepsilon$ e $\lambda = a^j b^t$

$\beta = b^k \mid \varepsilon$ e $\lambda = b^j c^t$

Com k, j e t ≥ 1. Em qualquer caso, $\alpha\beta^2\lambda^2\mu$ terá os símbolos fora de ordem.

Nos dois casos, não importa como ω é dividida, temos que $\alpha\beta^2 y\lambda^2\mu \notin$ L. Logo, L não é livre de contexto.

4 COMPUTABILIDADE

Antes de estudarmos propriamente a computabilidade, recordaremos os diferentes tipos de linguagens formais, conhecidas como hierarquia de Chomsky.

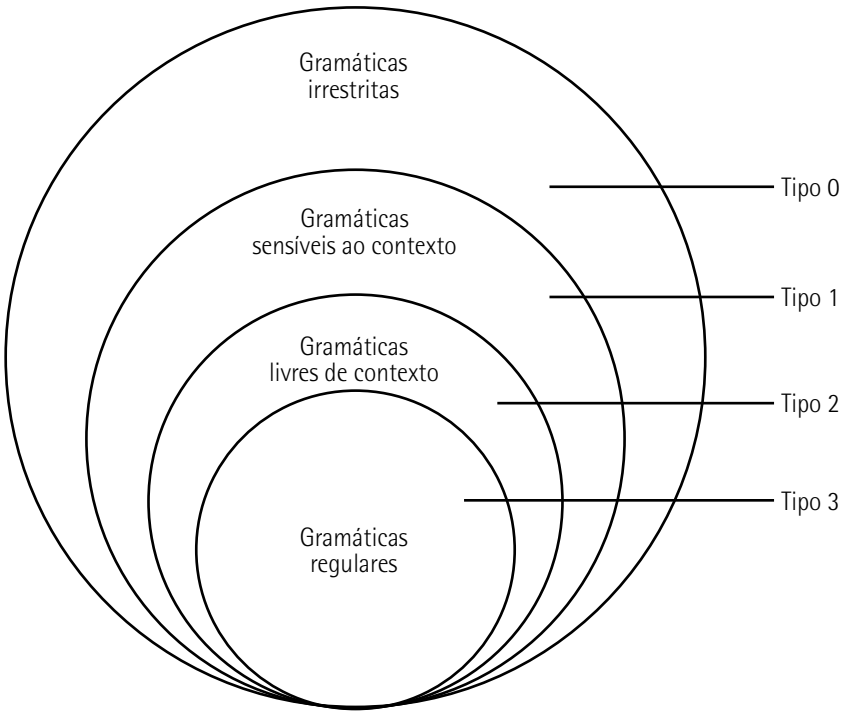

Figura 80 – Hierarquia de Chomsky para linguagens formais

A hierarquia de Chomsky é uma classificação de linguagens formais proposta pelo linguista Noam Chomsky na década de 1950. Ela categoriza diferentes tipos de gramáticas e linguagens de acordo com sua capacidade de gerar expressões e símbolos. É dividida em quatro níveis:

- **Tipo 0 – gramáticas irrestritas**: têm regras de produção muito poderosas, permitindo a substituição de qualquer sequência de símbolos por outra. Elas podem gerar linguagens que não são reconhecíveis por nenhum dispositivo de computação finito, incluindo as chamadas linguagens recursivamente enumeráveis e são reconhecidas por uma máquina de Turing (MT) com fita ilimitada.

- **Tipo 1 – gramáticas sensíveis ao contexto**: nesse nível, as regras de produção têm restrições contextuais que garantem que as substituições mudam de acordo com a estrutura do contexto circundante. Elas podem gerar linguagens sensíveis ao contexto, que são mais expressivas que as livres de contexto, mas menos poderosas que as recursivamente enumeráveis. São reconhecidas por uma MT com fita limitada ou um autômato com duas pilhas.

- **Tipo 2 – gramáticas livres de contexto**: possuem regras de produção mais restritas, permitindo apenas substituições simples independentemente do contexto. Elas podem gerar linguagens livres

de contexto, amplamente utilizadas na descrição de estruturas hierárquicas em linguagens de programação e análise sintática. São reconhecidas por autômatos de pilha.

- **Tipo 3 – gramáticas regulares**: no nível mais restrito, elas são características por regras de produção que permitem apenas substituições diretas e simples, sem qualquer tipo de contexto. Elas podem gerar linguagens regulares, que são aquelas reconhecíveis por autômatos finitos e expressões regulares.

A hierarquia de Chomsky estabelece uma ordem crescente de poder computacional, desde as linguagens regulares, que são as mais simples, até as recursivamente enumeráveis, que são as mais complexas. Cada tipo de gramática é capaz de gerar um conjunto específico de linguagens, e essa classificação tem sido fundamental na compreensão das capacidades e da limitação dos diferentes tipos de sistemas de processamento de linguagem.

No processo de compilação de linguagens de programação, as gramáticas regulares, as livres de contexto e as sensíveis ao contexto, segundo Moraes (2013), desempenham papéis distintos em diferentes etapas do processo, desde a análise léxica até a análise sintática e semântica. Observe a seguir a explicação do papel de cada tipo de gramática:

- **Gramáticas regulares (tipo 3 da hierarquia de Chomsky)**: são usadas principalmente na **análise léxica**. Elas descrevem padrões simples de tokens, como identificadores, números, símbolos e palavras-chave. A análise léxica transforma o código-fonte em sequências de tokens, e as gramáticas regulares auxiliam na definição dos padrões léxicos da linguagem. Expressões regulares são frequentemente usadas para especificar esses padrões, e os autômatos finitos (ou autômatos de estados finitos) podem ser usados para esses padrões durante a análise léxica.

- **Gramáticas livres de contexto (tipo 2 da hierarquia de Chomsky)**: desempenham um papel vital na **análise sintática**. Elas são usadas para descrever a estrutura hierárquica das construções sintáticas da linguagem. Durante uma análise sintática, os algoritmos de parsing (análise) são aplicados para verificar se a sequência de tokens segue as regras gramaticais. Se uma sequência de tokens for derivável a partir da gramática, uma árvore sintática será construída para representar a estrutura do código-fonte. Essa árvore sintática é usada como uma representação preservada para análise semântica e otimizações posteriores.

- **Gramáticas sensíveis ao contexto (tipo 1 da hierarquia de Chomsky)**: são mais poderosas do que as gramáticas livres de contexto e podem ser usadas em **análises sintáticas mais complexas**, que desenvolvem um entendimento mais profundo do contexto. Embora não sejam tão comuns quanto às gramáticas livres de contexto na compilação de linguagens de programação, elas podem ser usadas para capturar regras sintáticas e semânticas mais sofisticadas em linguagens específicas.

O componente dependente de contexto na linguagem de programação pode ser identificado em situações que envolvem a concordância entre diferentes partes do código, como na concordância entre a declaração de tipos de variáveis e seu uso, na concordância entre os parâmetros declarados em

um método e os argumentos transmitidos quando esse método é chamado, bem como em casos de sobrecarga de métodos.

A concordância de tipos e a consistência de informações em diferentes partes do código são exemplos claros de dependência de contexto. Isso ocorre porque o contexto do uso de uma variável ou método afeta diretamente a maneira como essas entidades devem ser interpretadas e utilizadas. Observe a seguir alguns exemplos:

- **Concordância entre declaração de tipos e uso de variáveis**: quando você declara uma variável com um determinado tipo, o contexto exige que essa variável seja usada de maneira consistente com relação aos tipos envolvidos. Qualquer uso subsequente da variável deve estar em conformidade com o tipo declarado, garantindo que as operações realizadas na variável sejam legais e coerentes.

- **Concordância de parâmetros e argumentos de método**: ao declarar um argumento com um certo número e tipo de parâmetros, o contexto exige que, ao chamá-lo, você dê um número igual de argumentos e que estes estejam de acordo com os tipos esperados. Isso garante que os dados sejam transmitidos de forma adequada e coerente entre o chamador e o método.

- **Sobrecarga de métodos**: ocorre quando uma classe tem vários métodos com o mesmo nome, mas com parâmetros diferentes, e o contexto determinará qual será chamado de método com base nos tipos e na quantidade de argumentos transmitidos na chamada do método. Isso demonstra claramente a dependência de contexto na resolução da chamada de método gentil.

Em resumo, as gramáticas regulares são usadas para especificar padrões léxicos durante a análise léxica; as livres de contexto são essenciais para a análise sintática, e as sensíveis ao contexto têm um papel menos comum, mas podem ser empregadas em análises sintáticas mais avançadas e num contexto específico. Cada tipo de gramática desempenha um papel crucial na transformação do código-fonte em uma representação preservada e na realização de verificações semânticas e otimizações durante o processo de construção.

4.1 A tese de Church-Turing

É um princípio fundamental na teoria da computação que estabelece uma conexão entre diferentes noções de computabilidade. Ela foi formulada independentemente por Alonzo Church e Alan Turing na década de 1930 e é um dos pilares conceituais da ciência da computação e da matemática.

Essa tese afirma que qualquer função que seja intuitivamente considerada como computável pode ser executada por uma MT. Em outras palavras, se há um procedimento bem definido para resolver um problema, mesmo que ele seja complexo, a tese sugere que ele pode ser reduzido a uma série de passos que uma MT pode realizar.

Uma MT é um modelo abstrato de um dispositivo computacional, consistindo em uma fita ilimitada à direita dividida em células, um cabeçote de leitura/gravação que pode mover-se para a esquerda

ou direita, e um conjunto finito de estados e regras de transição. A tese de Church-Turing sugere que todas as abordagens viáveis de computação, incluindo algoritmos manuais, MT e outras máquinas computacionais equivalentes levarão aos mesmos resultados computacionais.

A importância da tese de Church-Turing reside em sua capacidade de definir os limites da computabilidade. Ela sugere que há um conjunto de problemas para os quais não existe um algoritmo geral para resolvê-los, independentemente da abordagem computacional utilizada. Esses problemas são conhecidos como **incomputáveis** ou **indecidíveis**. Exemplos clássicos incluem o problema da parada, que consiste em determinar se um programa de computador eventualmente para ou entra em um loop infinito.

A tese de Church-Turing não é uma prova matemática, mas sim uma conjectura amplamente aceita e suportada por diversas formulações equivalentes de modelos de computação, como MT, máquinas de registro, máquinas de Post, cálculo lambda e outras. Ela fornece uma base teórica para a compreensão da natureza e dos limites da computação, servindo como um guia para determinar o que é e o que não é computacionalmente possível.

Segundo Copeland (2020):

> A tese de Church-Turing diz respeito ao conceito de um método eficaz ou sistemático ou mecânico em lógica, matemática e ciência da computação. "Eficaz" e seus sinônimos, "sistemático" e "mecânico", são termos artísticos nessas disciplinas: eles não carregam seu significado cotidiano. Um método, ou procedimento, M, para alcançar algum resultado desejado, é chamado de "eficaz" (ou "sistemático" ou "mecânico") apenas no caso de:
>
> 1. M é definido em termos de um número finito de instruções exatas (cada instrução sendo expressa por meio de um número finito de símbolos).
>
> 2. M irá, se executado sem erro, produzir o resultado desejado em um número finito de passos.
>
> 3. M pode (na prática ou em princípio) ser realizado por um ser humano sem a ajuda de qualquer máquina, exceto com papel e lápis.
>
> 4. M não exige discernimento, intuição ou engenhosidade por parte do ser humano que executa o método.

4.2 Definição formal de uma MT

A título de revisão, considere-se o alfabeto $\Sigma = \{a, b, c\}$. A partir desse alfabeto, podem ser definidas diferentes linguagens.

Exemplos:

$L_R = \{w \mid w = a^n\ bb\ c^m, n > 0, m > 0\}$

Trata-se de uma **linguagem regular**, pois não existe um vínculo entre o número de ocorrência dos diferentes símbolos a, b e c.

$L_L = \{w \mid w = a^n b^n, n > 0\}$

É uma **linguagem livre de contexto**, pois o número de ocorrências do símbolo c deve ser igual ao do símbolo a. Ao se verificar se uma palavra pertence a L_L, um reconhecedor deveria, ao identificar cada símbolo c, lembrar-se que anteriormente foi encontrado um símbolo a correspondente. Para tal, sabe-se que uma memória organizada em pilha resolve conceitualmente o problema.

$L_s = \{w \mid w = a^n b^n c^n, n > 0\}$

Trata-se de uma **linguagem sensível de contexto**. Ao se verificar se uma palavra pertence a L_s, um reconhecedor deveria memorizar não somente cada ocorrência do símbolo a, como também cada ocorrência do símbolo b para poder, finalmente, reconhecer cada ocorrência do símbolo c. Pode-se constatar que, conceitualmente, para se resolver esse problema, seriam necessárias duas pilhas ou uma MT, haja vista que as linguagens sensíveis ao contexto são também chamadas de Turing decidíveis ou Turing reconhecíveis.

As MT são similares aos autômatos finitos e de pilha e apresentam fita de entrada e unidade de controle finito. A figura a seguir ilustra sua estrutura.

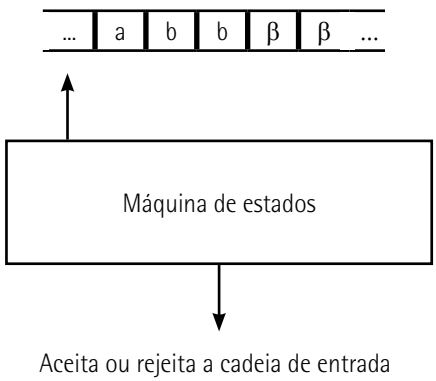

Figura 81 – Esquema de uma MT

Destaquemos a seguir seus componentes.

A **fita de entrada**, ilimitada à direita (ou infinita, segundo alguns autores), contém os símbolos do alfabeto de entrada, Σ, e do alfabeto da fita, Γ. O alfabeto da fita tem o símbolo de célula em branco, β, e outros símbolos. Alguns autores, no caso de fita ilimitada à direita, também inserem, no alfabeto da fita, o símbolo de início da fita, denotado por •.

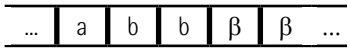

Figura 82

Por sua vez, o **cabeçote de leitura e escrita** tem a função de ler e sobrescrever um símbolo e movimentar-se para a esquerda ou para a direita.

Figura 83

A **máquina de estado** registra o estado corrente do conjunto **Q**, podendo parar, ao final da leitura, reconhecendo ou rejeitando a cadeia de entrada, ou ainda não parar e entrar em loop.

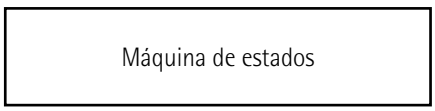

Figura 84

A MT lê a cadeia de entrada a partir da função de transição δ. Vejamos como funciona a função da transição.

Considere o grafo orientado a seguir, no qual os vértices representam os estados p e q e cuja função de transição é da forma $a \rightarrow b,\alpha$.

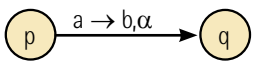

Figura 85

A leitura da função de transição é feita da seguinte maneira:

Estando a máquina no estado p, lendo o símbolo a, sobrescreva o símbolo b e mova o cabeçote de leitura para o lado α (esquerda ou direita) e então vá para o estado q.

Entendido como se dá a leitura da função de transição, montemos uma MT que representa o grafo anterior:

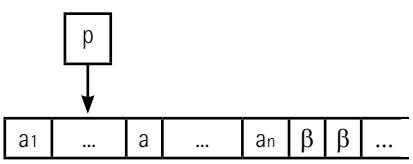

Figura 86

Observe que temos a fita ilimitada à direita, o cursor sobre o símbolo a e a máquina de estados mostrando que ela está no estado p.

Como visto, o cabeçote de leitura pode se mover para a esquerda ou para a direita, assim, temos duas situações possíveis:

- O cursor apaga o símbolo a e na mesma célula escreve o símbolo b, move-se à esquerda e a máquina de estados alcança o estado q.

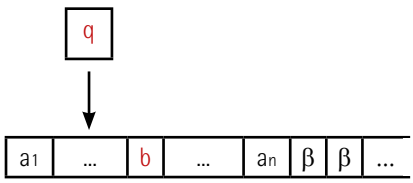

Figura 87

- O cursor apaga o símbolo a e na mesma célula escreve o símbolo b, move-se à direita e a máquina de estados alcança o estado q.

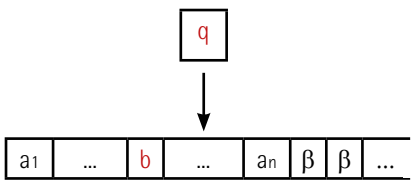

Figura 88

Há ainda o caso de o cabeçote de leitura sobrescrever o mesmo símbolo sobre a fita. Nessa conjuntura, a função de transição seria da forma $a \rightarrow a,_{\alpha}$ e as duas situações possíveis seriam:

O cursor mantém o símbolo a, move-se à esquerda e a máquina de estados alcança o estado q.

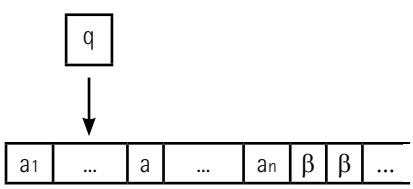

Figura 89

- O cursor mantém o símbolo a, move-se à direita e a máquina de estados alcança o estado q.

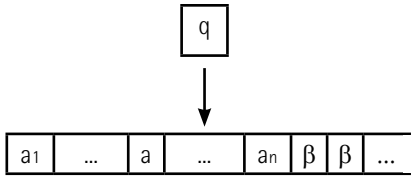

Figura 90

Exemplo de aplicação

Dada a linguagem L = {{ω / ω = $a^n b^n c^n$/ n ≥ 1}, determine se a cadeia aaaabbbbcccc pertence a L.

Resolução

Observe que a fita só contém a cadeia e o restante das células está em branco.

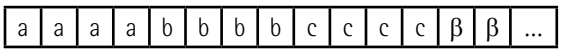

Figura 91

Considerando que o cabeçote de leitura está no primeiro símbolo à esquerda da fita, temos:

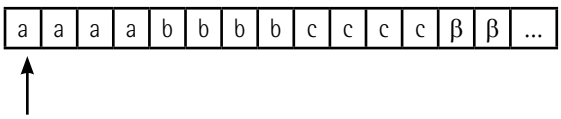

Figura 92

Lembrando que a linguagem determina que, a cada símbolo a lido, deve constar, nessa ordem, um símbolo b e um símbolo c. Dessa maneira, o cabeçote de leitura, após ler o primeiro símbolo a, andará para a direita, ignorando os demais símbolos a até encontrar o primeiro símbolo b. Após ler o primeiro símbolo b, andará à direita, ignorando todos os símbolos, até encontrar o primeiro símbolo c e lê-lo. Assim, marcaremos um x para cada símbolo a lido, um y para cada símbolo b lido e um z para cada símbolo c lido.

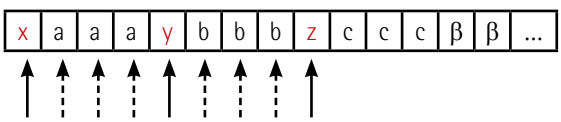

Figura 93

Lidos os primeiros símbolos a, b e c e substituídos por x, y e z, o cabeçote de leitura se moverá para a esquerda, ignorando todos os símbolos, até encontrar o primeiro x, e o cabeçote se moverá uma célula à direita.

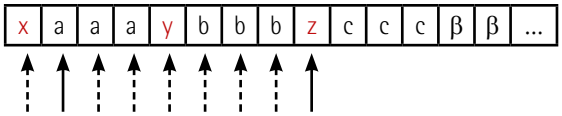

Figura 94

Lido o segundo símbolo a, será marcado um x e então o cursor se moverá à direita, ignorando todos os símbolos até encontrar o segundo símbolo b. Lido o segundo símbolo b, será marcado um y e então o cursor se moverá à direita, ignorando todos os símbolos, até encontrar o segundo símbolo c, o qual é lido e substituído por um z.

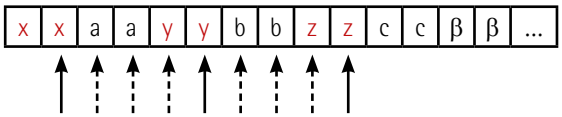

Figura 95

Lidos os segundos símbolos a, b e c e substituídos por x, y e z, o cabeçote de leitura se moverá para a esquerda, ignorando todos os símbolos até encontrar o segundo x, e então o cabeçote se moverá à direita.

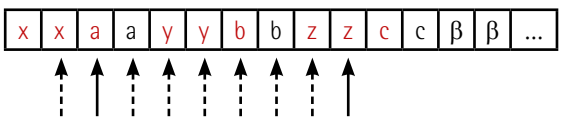

Figura 96

Lido o terceiro símbolo a, será marcado um x e então o cursor se moverá à direita, ignorando todos os símbolos até encontrar o terceiro símbolo b. Lido o terceiro símbolo b, será marcado um y e então o cursor se moverá à direita, ignorando todos os símbolos, até encontrar o terceiro símbolo c, o qual é lido e substituído por um z.

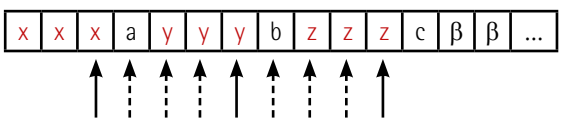

Figura 97

Lidos os terceiros símbolos a, b e c e substituídos por x, y e z, o cabeçote de leitura se moverá para a esquerda, ignorando todos os símbolos, até encontrar o terceiro x e então o cabeçote se moverá à direita.

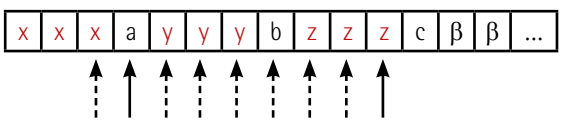

Figura 98

Lido o quarto símbolo a, será marcado um x e então o cursor se moverá à direita, ignorando todos os símbolos, até encontrar o quarto símbolo b. Lido o quarto símbolo b, será marcado um y e então o cursor se moverá à direita, ignorando todos os símbolos, até encontrar o terceiro símbolo c, o qual é lido e substituído por um z.

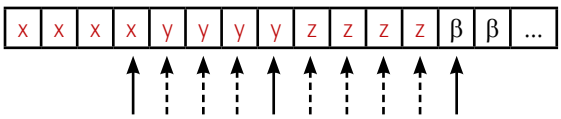

Figura 99

Lidos os quartos e últimos símbolos a, b e c e substituídos por x, y e z, o cabeçote se move à esquerda, ininterruptamente, até encontrar a quarto x.

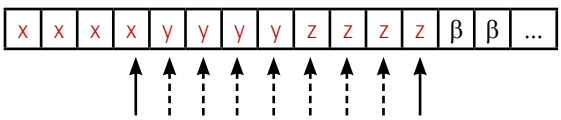

Figura 100

Nesse momento, o cabeçote de leitura começa a se mover à direita com o objetivo de encontrar algum símbolo não lido, até encontrar o símbolo de branco. Como não há mais símbolos para serem lidos, ele para e reconhece a cadeia.

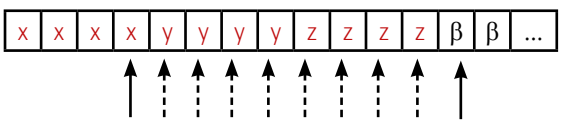

Figura 101

Esse reconhecimento só foi possível porque a cadeia pertence à linguagem.

Observe agora o exemplo a seguir com atenção.

Exemplo de aplicação

Dada a linguagem $L = \{\{\omega / \omega = a^n b^n c^n / n \geq 1\}$, determine se a cadeia aaabbccaa pertence a L.

Resolução

Considerando que o cabeçote de leitura está no primeiro símbolo à esquerda da fita, temos:

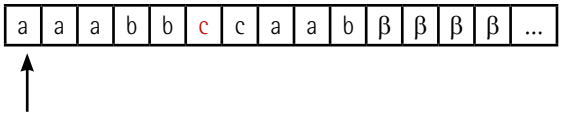

Figura 102

A linguagem determina que, a cada símbolo a lido, deve constar, nessa ordem, um símbolo b e um símbolo c. Dessa maneira, o cabeçote de leitura, após ler o primeiro símbolo a, andará para a direita, ignorando os demais símbolos a, até encontrar o primeiro símbolo b. Após ler o primeiro símbolo b, andará à direita, ignorando todos os símbolos, até encontrar o primeiro símbolo c e lê-lo. Assim, marcaremos um x para cada símbolo a lido, um y para cada símbolo b lido e um z para cada símbolo c lido.

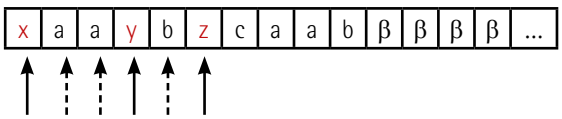

Figura 103

Lidos os primeiros símbolos a, b e c e substituídos por x, y e z, o cabeçote de leitura se moverá para a esquerda, ignorando todos os símbolos, até encontrar o primeiro x, e o cabeçote se moverá uma célula à direita.

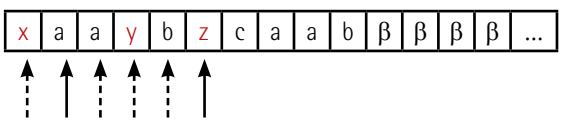

Figura 104

Lido o segundo símbolo a, será marcado um x e então o cursor se moverá à direita, ignorando todos os símbolos, até encontrar o segundo símbolo b. Lido o segundo símbolo b, será marcado um y e então o cursor se moverá à direita, ignorando todos os símbolos, até encontrar o segundo símbolo c, o qual é lido e substituído por um z.

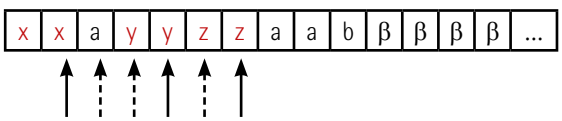

Figura 105

Lidos os segundos símbolos a, b e c e substituídos por x, y e z, o cabeçote de leitura se moverá para a esquerda, ignorando todos os símbolos, até encontrar o segundo x, e então o cabeçote move-se à direita.

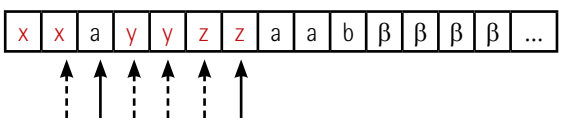

Figura 106

Lido o terceiro símbolo a, será marcado um x, o cursor move-se à direita, passando pelo primeiro e segundo y e, na sequência, pelo primeiro z. Nesse momento, o cabeçote de leitura para a cadeia é rejeitado.

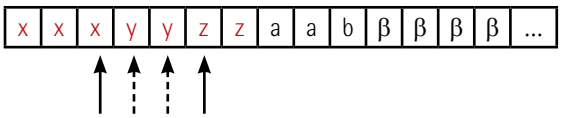

Figura 107

Você deve estar se perguntando porque a cadeia foi rejeitada.

Observe que a linguagem L = {{ω / ω = $a^n b^n c^n$/ n ≥ 1} determina que a quantidade de símbolos a, b e c, **nessa ordem**, tem que ser igual.

Nesse exemplo, a cadeia de entrada se inicia com três símbolos a. Isso significa que, obrigatoriamente nessa ordem, o cabeçote de leitura, ao ler o terceiro símbolo a, vai passar pelo primeiro símbolo b, pelo segundo símbolo b e deveria encontrar, na sequência, o terceiro símbolo b. Entretanto, após o segundo símbolo b, o cabeçote de leitura encontra o símbolo c e, como consequência, a leitura se encerra e a cadeia é rejeitada.

Perceba, portanto, que a marcação dos símbolos a, b e c por x, y e z funciona como uma maneira de contar as sequências dos símbolos.

Concluímos que o reconhecimento das cadeias pertencentes à linguagem L se ocorre assim:

- Percorra a fita marcando, para cada símbolo a, um b e um c.
- Se não for possível, rejeite.
- Se todos os símbolos foram marcados, aceite.
- Caso contrário, rejeite.

Exemplo de aplicação

No exemplo a seguir, um grafo de uma MT representa a linguagem L = {{ω / ω = $a^n b^n c^n$/ n ≥ 1}.

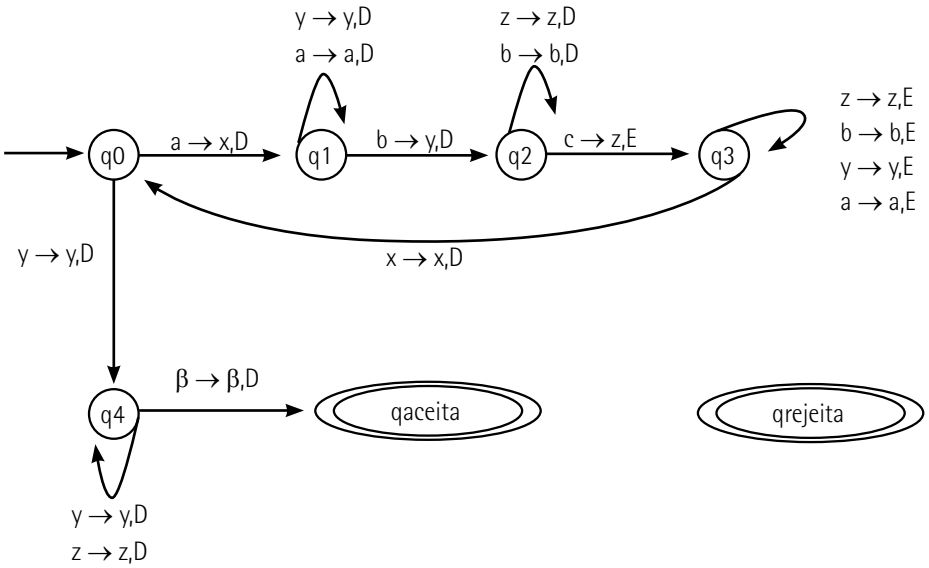

Figura 108

Para que o entendimento da interpretação do grafo fique mais claro, veremos, concomitantemente, o processamento da cadeia aaaabbbbcccc, pertencente a L, através da MT propriamente dita.

Resolução

Iniciamos a leitura do grafo na parte à esquerda, onde temos:

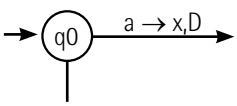

Figura 109

Paralelamente, na MT:

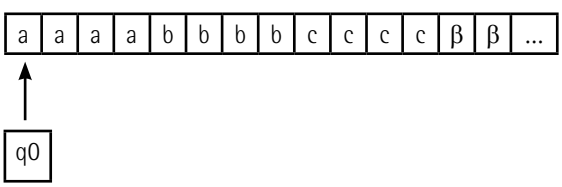

Figura 110

Aqui o processamento determina que, estando no estado q_0, lendo o símbolo a, sobrescreva x e mova uma célula à direita, alcançando o estado q_1. Assim, a MT fica com a seguinte configuração:

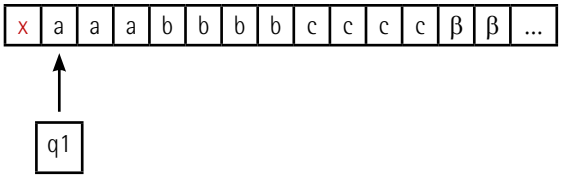

Figura 111

No grafo, a leitura será no vértice:

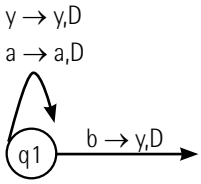

Figura 112

Observe que, nessa parte do grafo, temos três possíveis transições: duas no loop (y→y,D; a→a,D) e a transição b→y,D. Entretanto, a linguagem L determina que, para cada símbolo a encontrado, devemos encontrar, nessa ordem, um símbolo b e um símbolo c. Portanto, a transição escolhida deve ser a→a,D, o que possibilitará ao cabeçote de leitura andar à direita, ignorando os demais símbolos, até encontrar o primeiro símbolo b.

Dessa forma, com a MT no estado q_1, lendo o símbolo a, sobrescreva o símbolo a, mova uma célula à direita e mantenha o estado q_1. Mantém-se essa transição até que o cabeçote de leitura alcance o primeiro símbolo b. Assim:

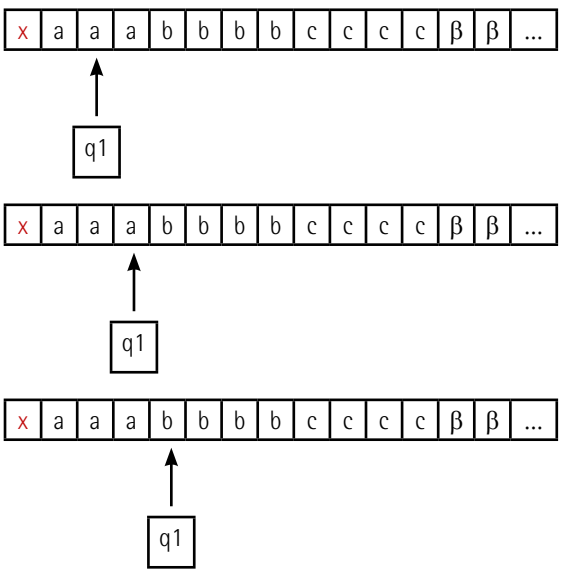

Figura 113

Alcançado o primeiro símbolo b, deve-se agora alcançar o primeiro símbolo c. Assim, a função de transição que torna isso possível é b→y,D, localizada em:

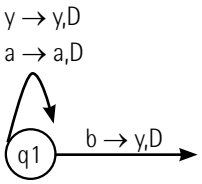

Figura 114

Dessa forma, com a MT no estado q_1, lendo o símbolo b, sobrescreva y e avance ao estado q_2.

No grafo, alcançamos a transição:

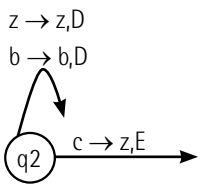

Figura 115

Na MT:

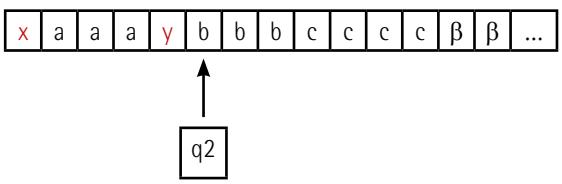

Figura 116

Como se deve alcançar o primeiro símbolo c, a MT deve prosseguir se movendo à direita, ignorando os demais símbolos até alcançá-lo. Assim, a função de transição b→b,D, que está no loop do grafo, deve ser aplicada por três vezes.

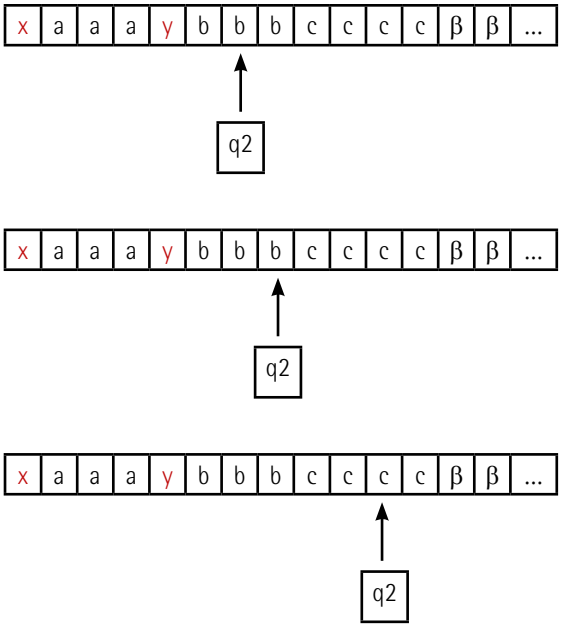

Figura 117

Alcançado o primeiro símbolo c, deve-se sobrescrever o símbolo z. Portanto, a função de transição que será utilizada é c→z,E. Dessa forma, estando a MT no estado q_2, lendo o símbolo c, sobrescreva z, mova uma célula à esquerda e alcance o estado q_3.

No grafo:

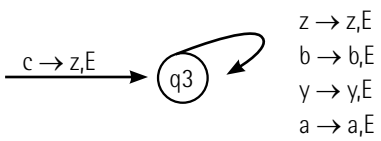

Figura 118

Assim, a configuração da MT será:

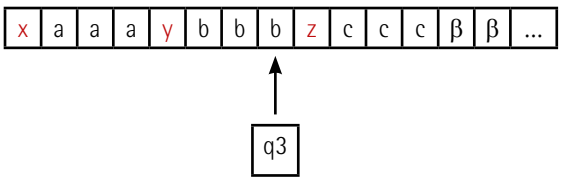

Figura 119

Nesse momento, finalizou-se a contagem dos primeiros símbolos a, b e c. Portanto, o cabeçote de leitura deve retornar ao símbolo x e inicializar a contagem do segundo símbolo a, do segundo b e do segundo c.

Como o cabeçote de leitura está apontando para o símbolo b e a máquina está no estado q_3, usaremos a função de transição b→b,E. Dessa forma, com a MT no estado q_3, lendo o símbolo b, sobrescreva b, mova-se à esquerda e permaneça no estado q_3.

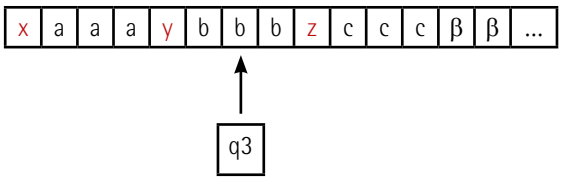

Figura 120

E, por mais duas vezes, usa-se a mesma transição:

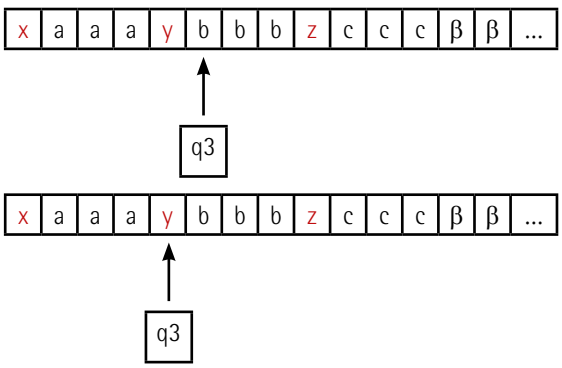

Figura 121

Como o cabeçote de leitura está no símbolo y e a máquina no estado q_3, usa-se a função de transição y→y,E. Logo, com a MT no estado q_3, lendo o símbolo y, sobrescreva y, mova uma célula à esquerda e permaneça no estado q_3.

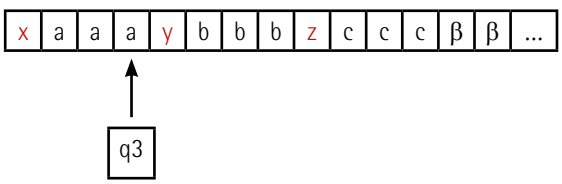

Figura 122

Lembrando que a MT deve retornar até o símbolo x. Com o cabeçote de leitura no estado q_3, lendo o símbolo a, deve-se usar a função de transição a→a,E por três vezes. Assim, com a MT no estado q_3, lendo o símbolo a, sobrescreva a, mova uma célula à esquerda e permaneça no estado q_3.

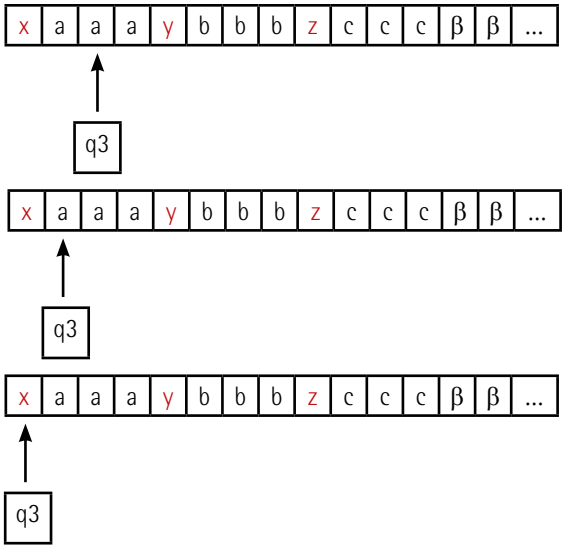

Figura 123

Alcançado o símbolo x, a máquina está pronta para inicializar a leitura e a contagem dos segundos símbolos a, b e c nessa ordem. Assim, como o cabeçote de leitura está lendo x e a máquina está em q_3, usa-se a transição x→x,D, que liga o vértice q_3 ao vértice q_0 do grafo.

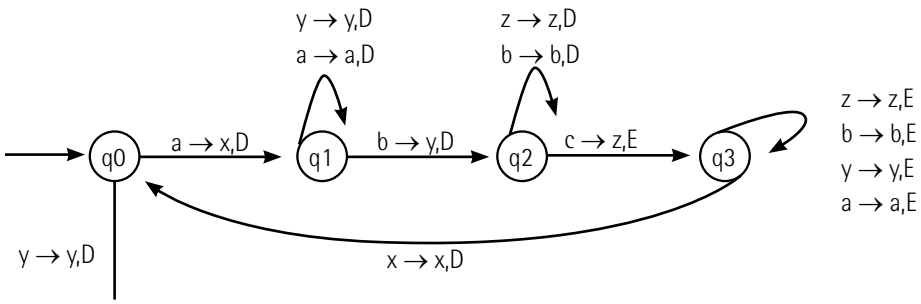

Figura 124

Portanto, com a MT no estado q_3, lendo x, sobrescreva x, mova uma célula à direita e alcance o estado q_0. Assim, no grafo retornamos à posição

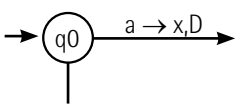

Figura 125

Feito o processamento da transição x→x,D, a nova configuração da MT será:

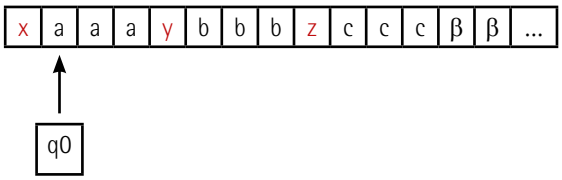

Figura 126

A única transição possível para a MT quando ela está no estado q_0, lendo o símbolo a, é a transição a→x,D. Assim, com a MT no estado q_0, lendo o símbolo a, sobrescreva x, mova uma célula à direita e alcance o estado q_1.

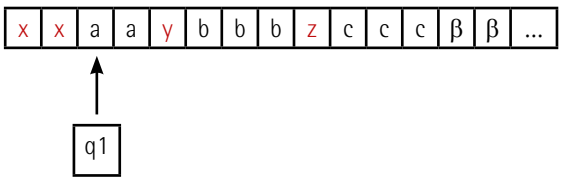

Figura 127

No grafo:

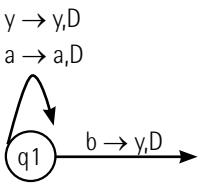

Figura 128

Nesse momento, o cabeçote de leitura terá que se mover à direita, ignorando todos os símbolos até que encontre o segundo símbolo b. Dessa forma, usa-se a transição a→a,D do loop por duas vezes. Assim, com a MT no estado q_1, lendo o símbolo a, sobrescreva a, mova-se à direita e permaneça no estado q_1.

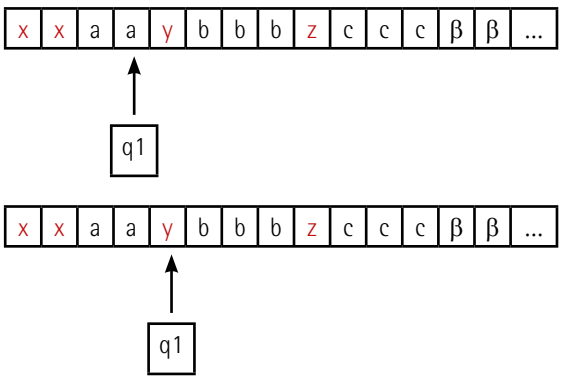

Figura 129

Agora, com a MT no estado q_1, lendo o símbolo y, ela deve alcançar o segundo símbolo b, e a transição usada será y→y,D do loop. Assim, com a MT no estado q_1, lendo o símbolo y, sobrescreva y, mova-se à direita e permaneça no estado q_1.

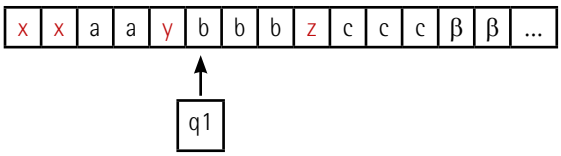

Figura 130

Encontrado o segundo símbolo b, a MT deve continuar se movendo à direita para encontrar o segundo símbolo c. Nesse caso, usa-se a transição b→y,D e alcança-se o estado q_2. Portanto, com a MT no estado q_1, lendo o símbolo b, sobrescreva y, mova-se à direita e alcance o estado q_2.

No grafo:

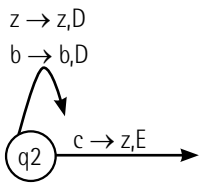

Figura 131

Na MT:

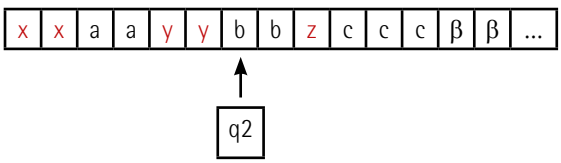

Figura 132

Nesse ponto, usa-se a transição b→b,D por duas vezes.

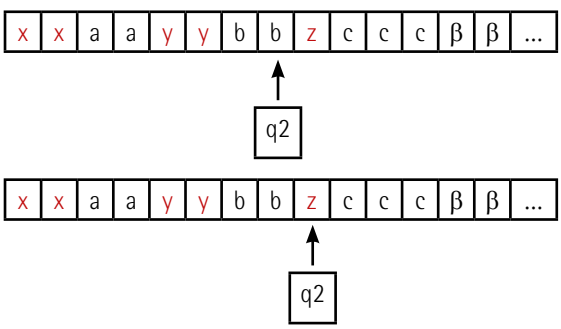

Figura 133

Agora, o cabeçote de leitura vai ler o símbolo z e a máquina ficará no estado q_2. Dessa forma, a MT usará a transição z→z,D do loop. Assim, com a MT no estado q_2, lendo o símbolo z, sobrescreva z, mova-se à direita e permaneça no estado q_2.

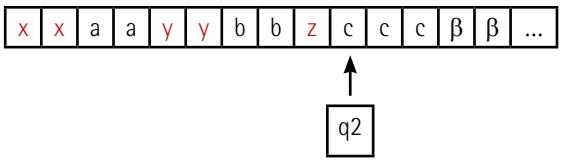

Figura 134

Alcançado o segundo símbolo c, a máquina efetuará a leitura desse símbolo e deverá mover-se à esquerda, ignorando todos os demais símbolos, até alcançar o segundo símbolo x. Dessa maneira, a MT poderá efetuar a leitura dos terceiros símbolos a, b e c nessa ordem.

Assim, a máquina usará a transição c→z,E, que a levará ao estado q_3. Com a MT no estado q_2, lendo o símbolo c, sobrescreva z, mova-se à direita e alcance o estado q_3.

No grafo:

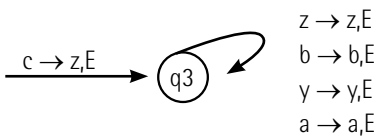

Figura 135

Na MT:

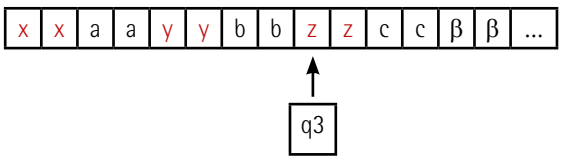

Figura 136

Como a máquina está no estado q_3 e deverá mover-se à esquerda até encontrar o segundo x, usa-se a transição z→z,E do loop. Assim, com a MT no estado q_3, lendo o símbolo z, sobrescreva z, mova-se à esquerda e permaneça no estado q_3.

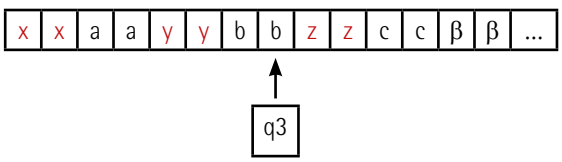

Figura 137

Como o cabeçote de leitura está sob o símbolo b, a MT usará a transição b→b,E do loop por duas vezes. Assim, com a MT no estado q_3, lendo o símbolo b, sobrescreva b, mova-se à esquerda e permaneça no estado q_3.

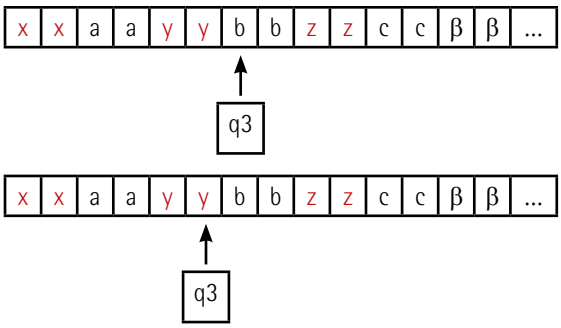

Figura 138

Nesse momento, o cabeçote de leitura está sob o símbolo y e a máquina no estado q_3, logo, a transição a ser usada, por duas vezes, é a y→y,E do loop. Assim, com a MT no estado q_3, lendo o símbolo y, sobrescreva y, mova-se à esquerda e permaneça no estado q_3.

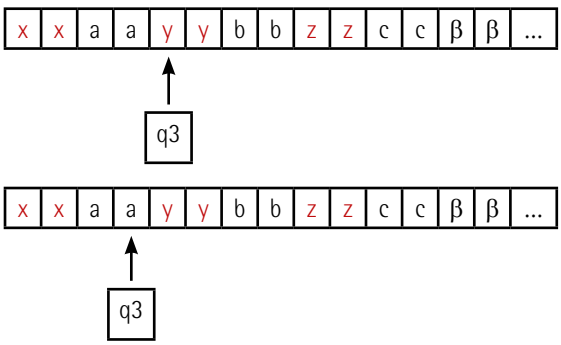

Figura 139

Estando o cabeçote de leitura no símbolo a e a máquina no estado q_3, a transição a ser usada, por duas vezes, é a→a,E do loop. Portanto, com a MT no estado q_3, lendo o símbolo a, sobrescreva a, mova-se à esquerda e permaneça no estado q_3.

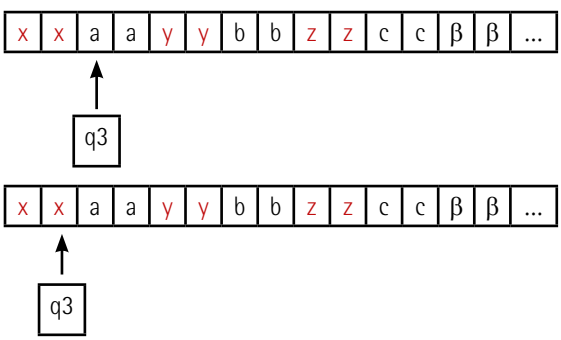

Figura 140

Alcançado o segundo símbolo x, a MT efetuará a leitura desse símbolo e assim estará apta a efetuar a leitura/contagem dos terceiros símbolos a, b e c. Portanto, a transição a ser usada é x→x,D, que liga o vértice q_3 ao vértice q_0 do grafo.

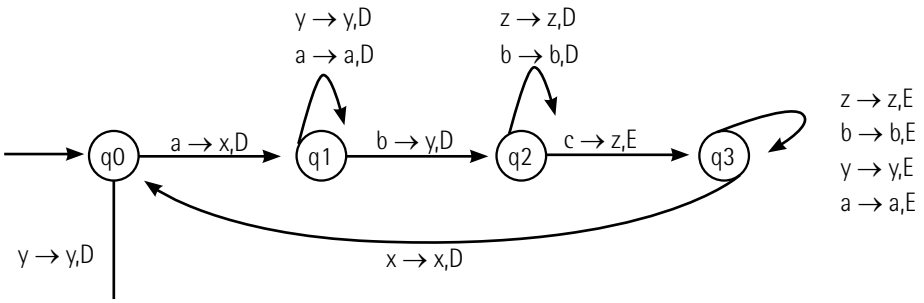

Figura 141

Assim, com a MT no estado q_3, lendo x, sobrescreva x, mova-se à direita e alcance o estado q_0.

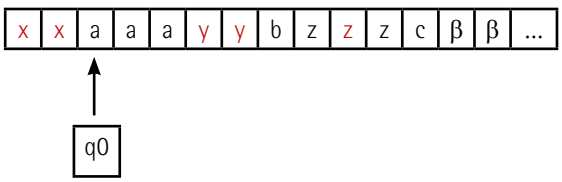

Figura 142

Assim, no grafo retornamos à posição

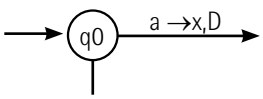

Figura 143

A única transição possível para a MT quando ela está no estado q_0, lendo o símbolo a, é a transição a→x,D. Assim, com a MT no estado q_0, lendo o símbolo a, sobrescreva x, mova uma célula à direita e alcance o estado q_1.

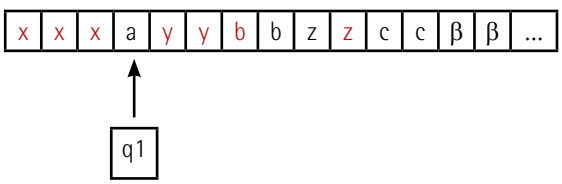

Figura 144

No grafo:

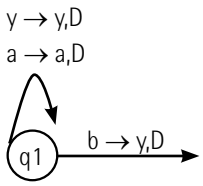

Figura 145

Nesse momento, o cabeçote de leitura terá que se mover à direita, ignorando todos os símbolos até encontrar o terceiro símbolo b. Dessa forma, usa-se a transição a→a,D do loop. Assim, com a MT no estado q_1, lendo o símbolo a, sobrescreva a, mova-se à direita e permaneça no estado q_1.

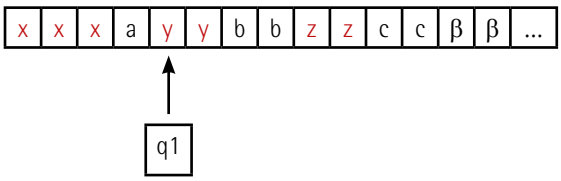

Figura 146

Agora, como a MT está no estado q_1, lendo o símbolo y, e ela deve alcançar o terceiro símbolo b, a transição usada será y→y,D do loop por duas vezes. Assim, com a MT no estado q_1, lendo o símbolo y, sobrescreva y, mova-se à direita e permaneça no estado q_1.

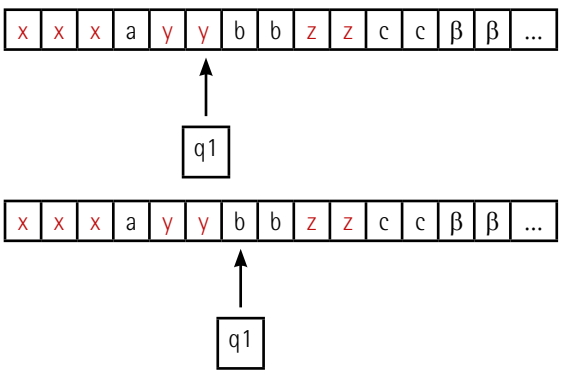

Figura 147

Encontrado o terceiro símbolo b, a MT deve continuar se movendo à direita para encontrar o terceiro símbolo c. Nesse caso, usa-se a transição b→y,D e alcança-se o estado q_2. Portanto, com a MT no estado q_1, lendo o símbolo b, sobrescreva y, mova-se à direita e alcance o estado q_2.

No grafo:

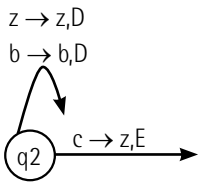

Figura 148

Na MT:

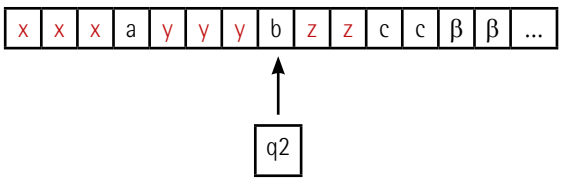

Figura 149

Nesse ponto, usa-se a transição b→b,D do loop.

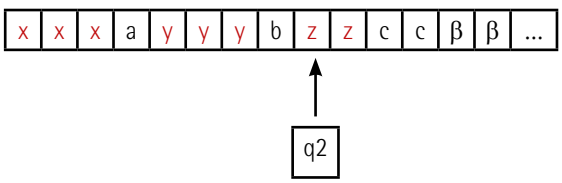

Figura 150

Agora, o cabeçote de leitura vai ler o símbolo z e a máquina ficará no estado q_2. Dessa forma, a MT usará a transição z→z,D do loop por duas vezes. Assim, com a MT no estado q_2, lendo o símbolo z, sobrescreva z, mova-se à direita e permaneça no estado q_2.

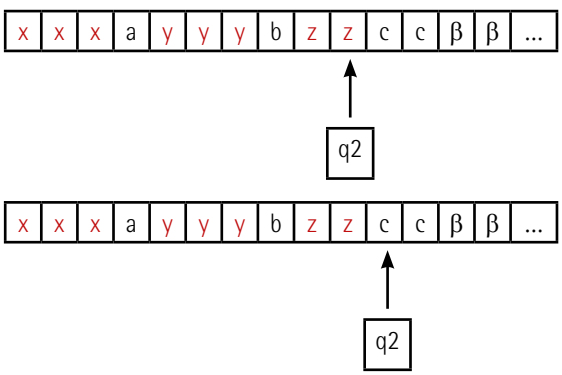

Figura 151

Alcançado o terceiro símbolo c, a máquina efetuará a leitura desse símbolo e deverá mover-se à esquerda, ignorando todos os demais símbolos, até alcançar o terceiro símbolo x. Dessa maneira, a MT poderá efetuar a leitura dos quartos símbolos a, b e c nessa ordem.

Assim, a máquina usará a transição c→z,E, que a levará ao estado q_3. Com a MT no estado q_2, lendo o símbolo c, sobrescreva z, mova-se à direita e alcance o estado q_3.

No grafo:

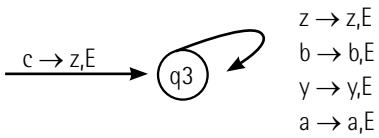

Figura 152

Na MT:

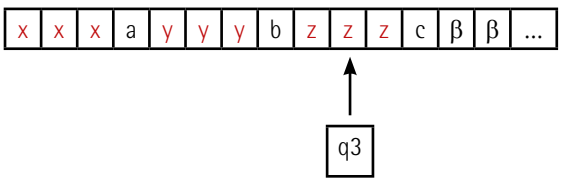

Figura 153

Como a máquina está no estado q_3 e deverá mover-se à esquerda até encontrar o terceiro x, usa-se a transição z→z,E do loop por duas vezes. Assim, com a MT no estado q_3, lendo o símbolo z, sobrescreva z, mova-se à esquerda e permaneça no estado q_3.

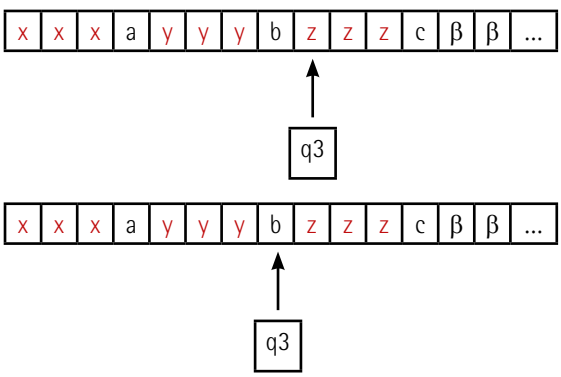

Figura 154

Como o cabeçote de leitura está sob o símbolo b, a MT usará a transição b→b,E do loop. Assim, com a MT no estado q_3, lendo o símbolo b, sobrescreva b, mova-se à esquerda e permaneça no estado q_3.

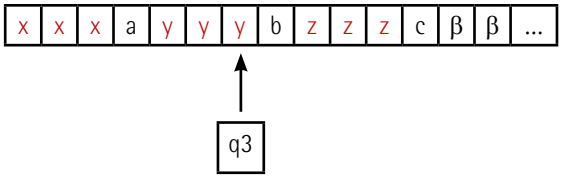

Figura 155

Nesse momento, o cabeçote de leitura está sob o símbolo y e a máquina ficará no estado q_3, logo a transição a ser usada, por três vezes, é a y→y,E do loop. Assim, com a MT no estado q_3, lendo o símbolo y, sobrescreva y, mova-se à esquerda e permaneça no estado q_3.

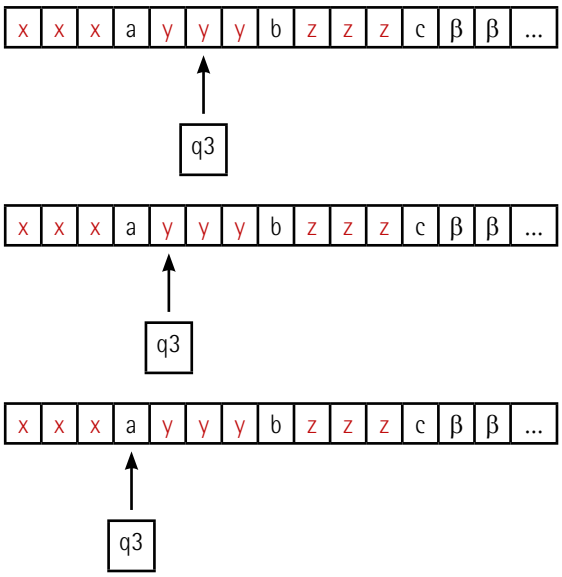

Figura 156

Estando o cabeçote de leitura no símbolo a e a máquina no estado q_3, a transição a ser usada é a→a,E do loop. Portanto, com a MT no estado q_3, lendo o símbolo a, sobrescreva a, mova-se à esquerda e permaneça no estado q_3.

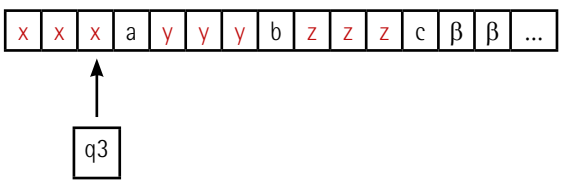

Figura 157

Alcançado o terceiro símbolo x, a MT efetuará a leitura desse símbolo e assim estará apta a fazer a leitura/contagem dos quartos símbolos a, b e c. Portanto, a transição a ser usada é x→x,D, que liga o vértice q_3 ao vértice q_0 do grafo.

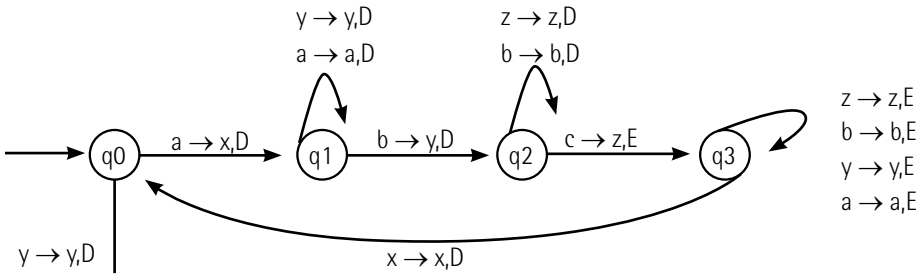

Figura 158

Assim, com a MT no estado q_3, lendo x, sobrescreva x, mova-se à direita e alcance o estado q_0.

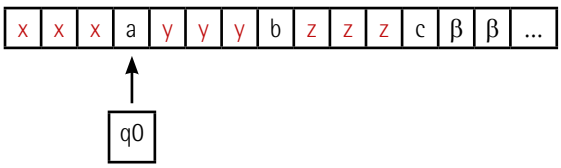

Figura 159

Assim, no grafo, retornamos à posição

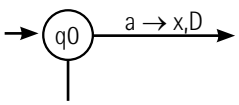

Figura 160

A única transição possível para a MT quando ela está no estado q_0, lendo o símbolo a, é a transição a→x,D. Assim, com a MT no estado q_0, lendo o símbolo a, sobrescreva x, mova uma célula à direita e alcance o estado q_1.

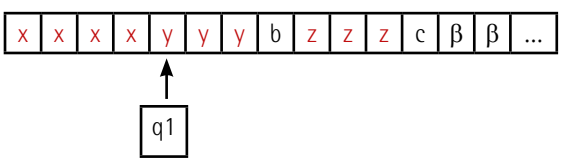

Figura 161

No grafo:

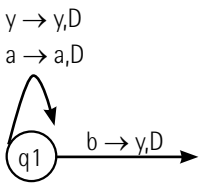

Figura 162

Nesse momento, o cabeçote de leitura terá que se mover à direita, ignorando todos os símbolos até encontrar o quarto símbolo b. Dessa forma, usa-se a transição y→y,D do loop por três vezes. Assim, com a MT no estado q_1, lendo o símbolo y, sobrescreva y, mova-se à direita e permaneça no estado q_1.

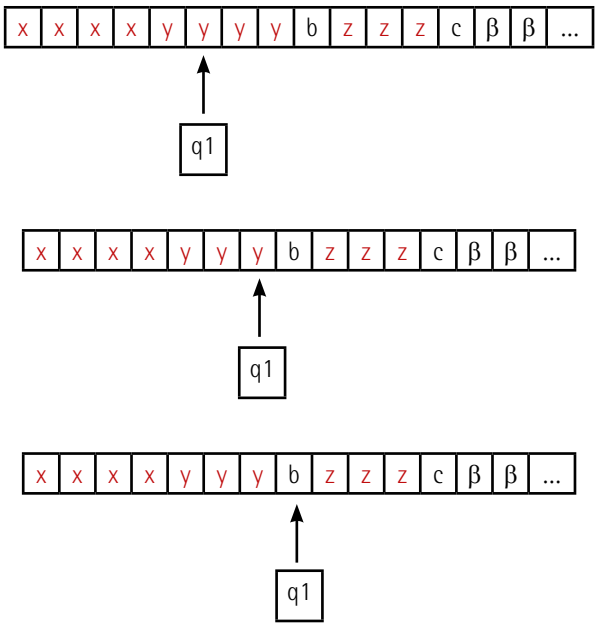

Figura 163

Encontrado o terceiro símbolo b, a MT deve continuar se movendo à direita para encontrar o quarto símbolo c. Nesse caso, usa-se a transição b→y,D e alcança-se o estado q_2. Portanto, com a MT no estado q_1, lendo o símbolo b, sobrescreva y, mova-se à direita e alcance o estado q_2.

No grafo:

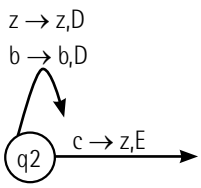

Figura 164

Na MT:

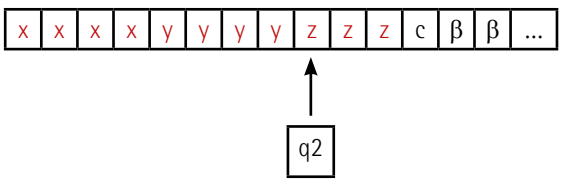

Figura 165

Agora, o cabeçote de leitura vai ler o símbolo z e a máquina ficará no estado q_2. Dessa forma, a MT usará a transição z→z,D do loop por três vezes. Assim, com a MT no estado q_2, lendo o símbolo z, sobrescreva z, mova-se à direita e permaneça no estado q_2.

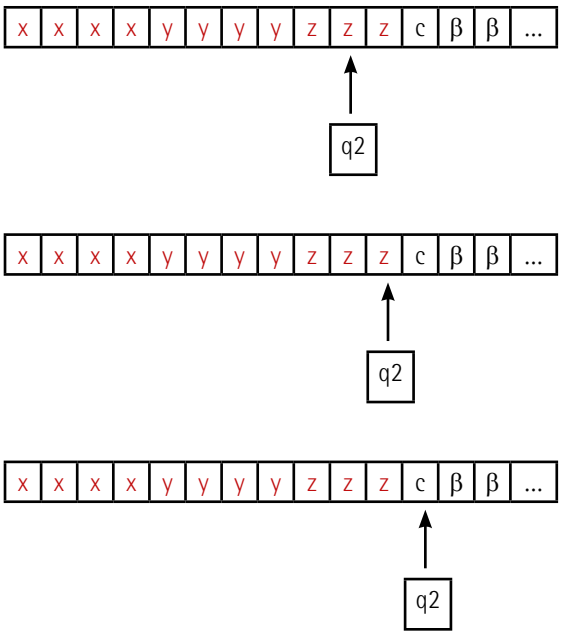

Figura 166

Alcançado o quarto símbolo c, a máquina efetuará a leitura desse símbolo e deverá mover-se à esquerda, ignorando todos os demais símbolos, até alcançar o quarto símbolo x. Esse movimento à esquerda é necessário porque a MT precisa verificar se não há mais alguma sequência de símbolos a, b e c. Assim, a máquina usará a transição c→z,E, que a levará ao estado q_3. Com a MT no estado q_2, lendo o símbolo c, sobrescreva z, mova-se à direita e alcance o estado q_3.

No grafo:

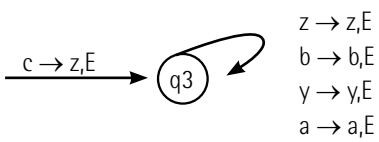

Figura 167

Na MT:

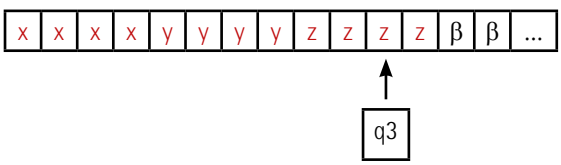

Figura 168

Como a máquina está no estado q_3 e deverá mover-se à esquerda até encontrar o quarto x, usa-se a transição z→z,E do loop por três vezes. Assim, com a MT no estado q_3, lendo o símbolo z, sobrescreva z, mova-se à esquerda e permaneça no estado q_3.

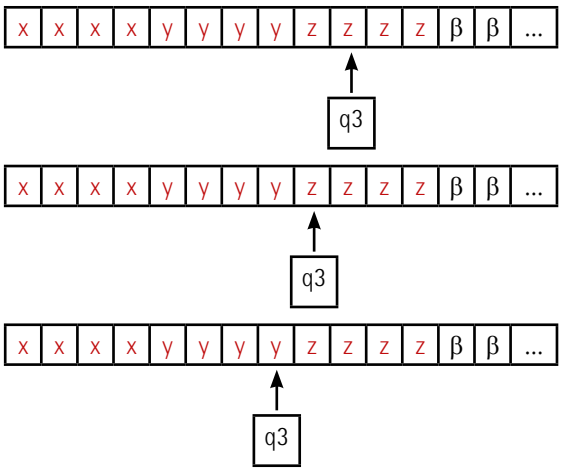

Figura 169

Nesse momento, o cabeçote de leitura está sob o símbolo y e a máquina ficará no estado q_3, logo a transição a ser usada, por quatro vezes, é a y→y,E do loop. Assim, com a MT no estado q_3, lendo o símbolo y, sobrescreva y, mova-se à esquerda e permaneça no estado q_3.

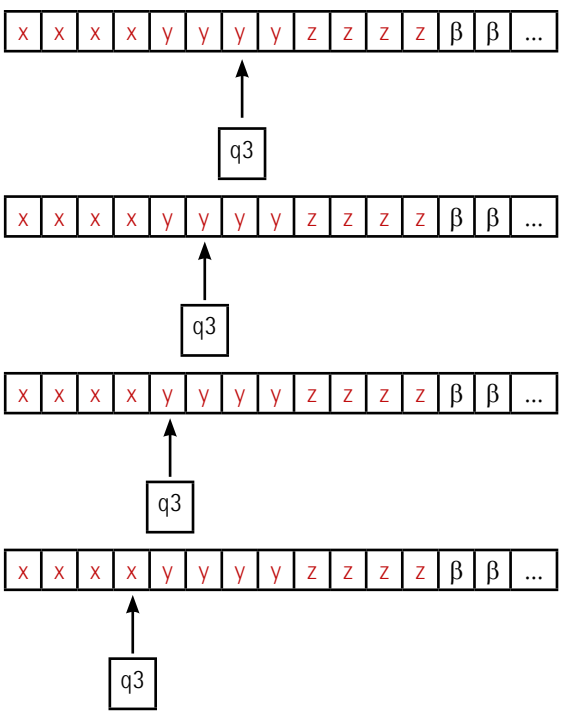

Figura 170

Alcançado o quarto símbolo x e com a MT no estado q_3, ela efetuará a leitura desse símbolo através da transição x→x,D, que liga o vértice q_3 ao vértice q_0 do grafo.

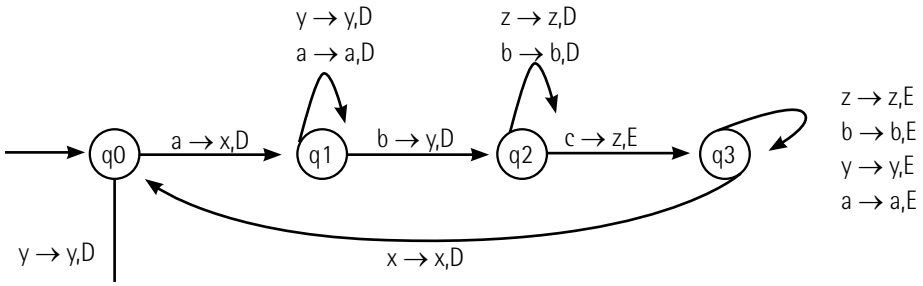

Figura 171

Assim, com a MT no estado q_3, lendo x, sobrescreva x, mova-se à direita e alcance o estado q_0.

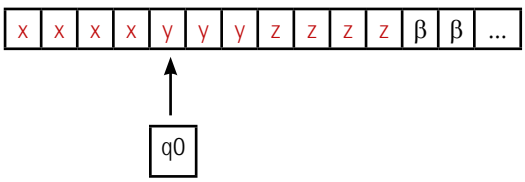

Figura 172

Observe que o cabeçote de leitura está sob o símbolo y, logo a única transição possível para o estado q_0, lendo y, é a transição que liga o vértice q_0 ao vértice q_4, ou seja, y→y,D.

No grafo:

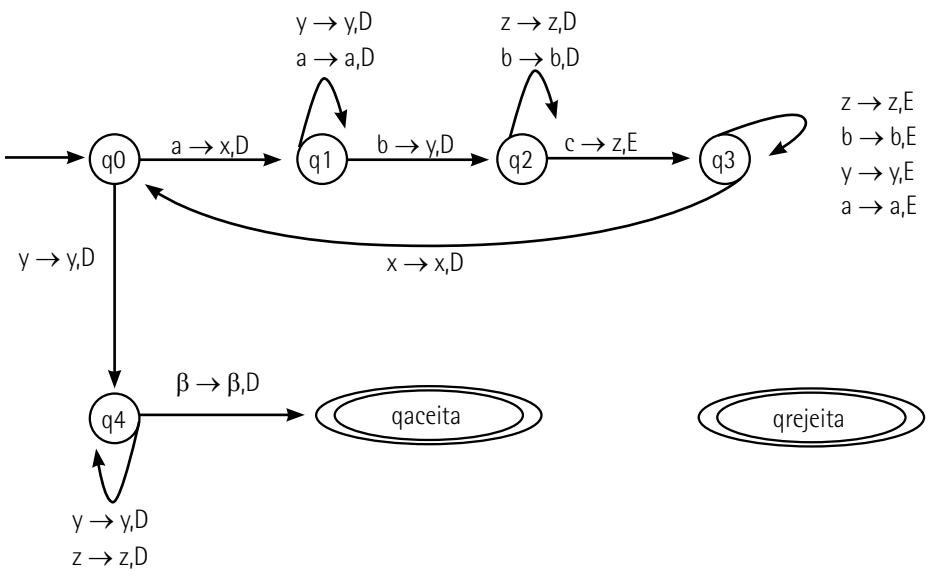

Figura 173

Portanto, com a MT no estado q_0, lendo o símbolo y, sobrescreva y, mova-se à direita e avance ao estado q_4.

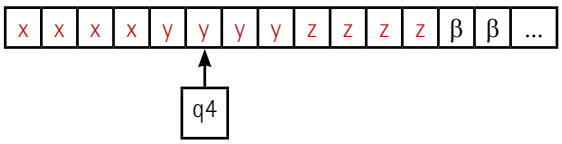

Figura 174

Com a MT no estado q_4, lendo um y, ela usará a transição y→y,D do loop por três vezes. Assim, com a MT no estado q_4, lendo o símbolo y, sobrescreva y, mova-se à direita e permaneça no estado q_4.

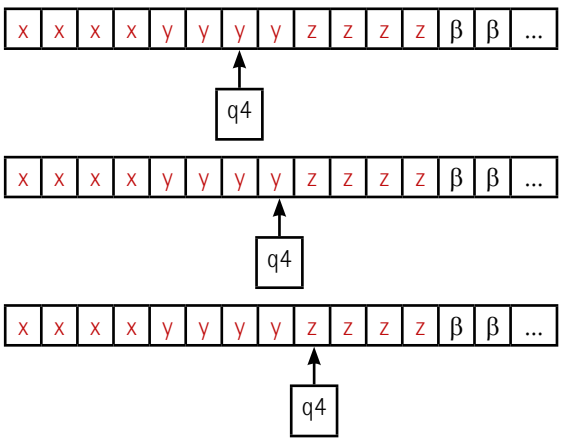

Figura 175

Alcançado o símbolo z, será usada a transição z→z,D do loop por quatro vezes. Assim, com a MT no estado q_4, lendo o símbolo z, sobrescreva z, mova-se à direita e permaneça no estado q_4.

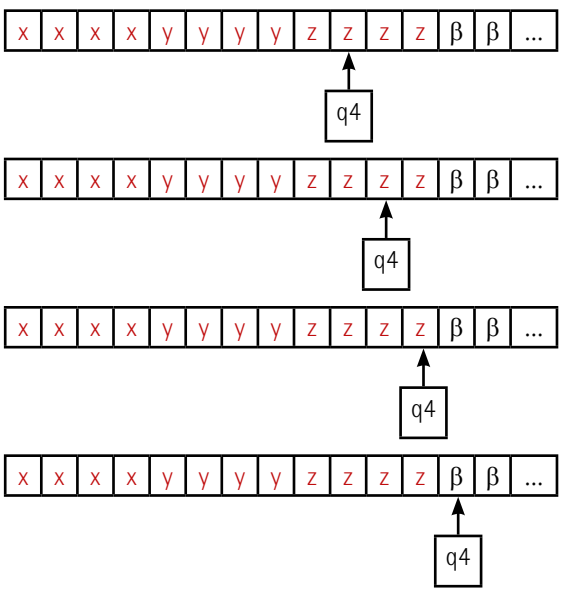

Figura 176

Alcançada a célula em branco, β, a transição possível é a que liga o vértice q_4 ao vértice q_{aceita}, ou seja, a transição $\beta \rightarrow \beta, D$.

No grafo:

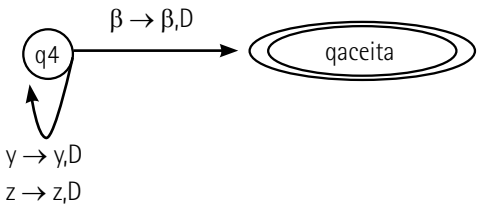

Figura 177

Assim, com a MT no estado q_4, lendo o símbolo β, sobrescreva β, mova-se à direita e alcance o estado q_{aceita}.

Nesse momento, a MT para e aceita a cadeia aaaabbbbcccc.

Definição formal

Agora que conhecemos o funcionamento de uma MT, vamos à sua definição.

Formalmente, uma MT é uma 7-upla $(Q, \Sigma, \Gamma, \delta, q_0, q_a, q_r)$, onde:

Q é o conjunto finito de estados

Σ é o alfabeto de entrada, onde $\beta \notin \Sigma$ e $\bullet \notin \Sigma$

Γ é o alfabeto da fita, tal que $\Sigma \subseteq \Gamma$, $\bullet \in \Gamma e \beta \in \Gamma$

δ é a função de transição, com δ: $Q \times \Gamma \rightarrow Q \times \Gamma \times \{E,D\}$

$q_0 \in Q$ é o estado inicial

q_a é o estado de aceitação

q_r é o estado de rejeição, $q_r \neq q_a$

Observe a função de transição δ. Por se tratar de uma função, o conjunto de partida é um par ordenado que contém um determinado estado e um símbolo do alfabeto da fita, por exemplo, (q_1,x), e o conjunto de chegada é uma 3-upla contendo um estado, um símbolo do alfabeto da fita e a direção a seguir pelo cabeçote da fita, por exemplo, (q_2,y,D). Dessa forma, temos:

$\delta (q_1,x) = (q_2,y,D)$

Essa é a notação formal da função de transição, não alterando a interpretação. Assim, com a MT no estado q1, lendo x, sobrescreva y, mova uma célula à direita e alcance o estado q_2.

Voltando ao autômato do exemplo anterior, você deve ter notado que não há nenhuma transição para o estado de rejeição q_r.

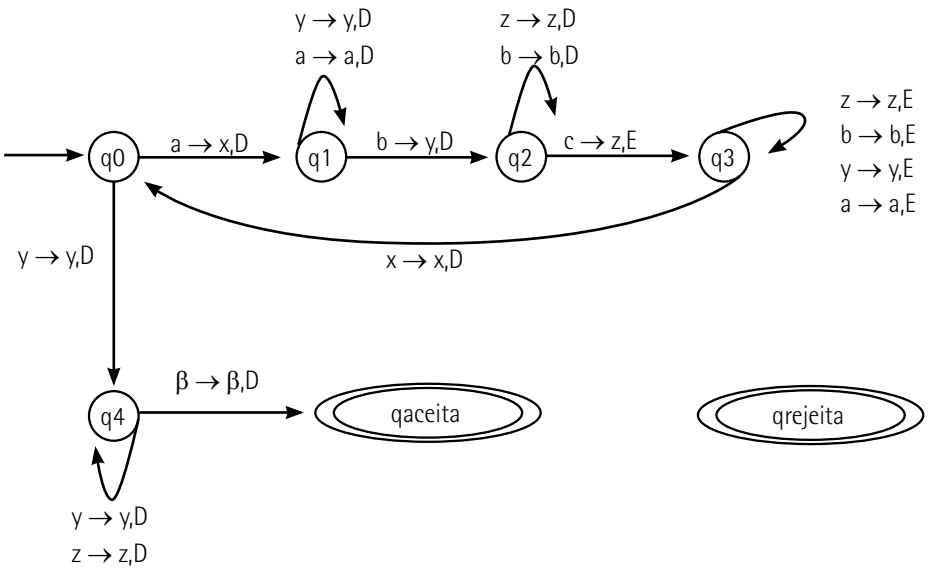

Figura 178

Na realidade, essas transições existem, só não foram inseridas para não poluir o desenho do autômato. Entretanto, podemos facilmente identificar as transições que levam a qr.

No autômato, no estado q_0, nas transições ele lê os símbolos a e y. Assim, se o autômato estiver no estado q_0, lendo qualquer dos símbolos que não sejam a e y, obrigatoriamente, o levará ao estado qr. Da mesma forma, com o autômato no estado q_1, lendo qualquer dos símbolos que não sejam a, b e y, o levará também ao estado qr.

Pode-se, portanto, aplicar esse mesmo raciocínio aos estados q_2, q_3 e q_4.

Configurações da MT

Cada posição do cabeçote de leitura obtida com as transições representa uma nova configuração da MT.

Veja a seguir algumas configurações especiais:

Considere $M = (Q, \Sigma, \Gamma, \delta, q_0, q_a, q_r)$, uma MT e $\omega \in \Sigma^*$

- Qualquer $q_0\ \omega$ é a configuração inicial.
- Qualquer αq_{ay} é a configuração de aceitação.
- Qualquer αq_{ry} é a configuração de rejeição.

Configuração de aceitação e rejeição são denominadas configurações de parada. Isso quer dizer que, considerando uma MT e uma cadeia qualquer, deve-se iniciar o processamento no estado q_0; alcançando q_a, o processamento para e aceita-se a cadeia; alcançando o estado q_r, o processamento para e rejeita-se a cadeia.

Computabilidade

Recordando os autômatos finitos e os de pilha, vimos que eles aceitavam a cadeia quando eles tinham alcançado o estado final.

No caso das MT, como há dois estados especiais, q_a e q_r, insere-se nova nomenclatura. Assim, considerando uma máquina de MT e uma cadeia, dizemos que:

- MT **aceita** a cadeia se partir da configuração inicial; se efetuar zero ou mais transições, alcança a configuração de aceitação.
- MT **decide** a cadeia se partir da configuração inicial; se efetuar zero ou mais transições, alcança a configuração de aceitação e a de rejeição.

Saiba mais

O filme indicado a seguir é um longa-metragem de 2014 que se baseia na história real do pai da computação, Alan Turing (1912-1954).

O JOGO da imitação. Direção: Morten Tyldum. EUA: Netflix, 2014. 114 min.

4.3 Linguagens recursivas e recursivamente enumeráveis

Nesse momento, é possível apresentar duas novas classes de linguagem. Menezes (2000) as define como:

- A linguagem L é recursiva, ou Turing – decidível, se existe uma MT que a decide:
  - Dado $\omega \in \Sigma^*$, a MT decide se $\omega \in L$ ou $\omega \notin L$

- A linguagem L é recursivamente enumerável, ou Turing – reconhecível, se existe um MT que a reconhece:

  – Dado $\omega \in \Sigma^*$, a MT aceita se $\omega \in L$ e rejeita ou entra em loop se $\omega \notin L$

Se uma Linguagem é recursiva, então também é recursivamente enumerável.

Observe a figura a seguir:

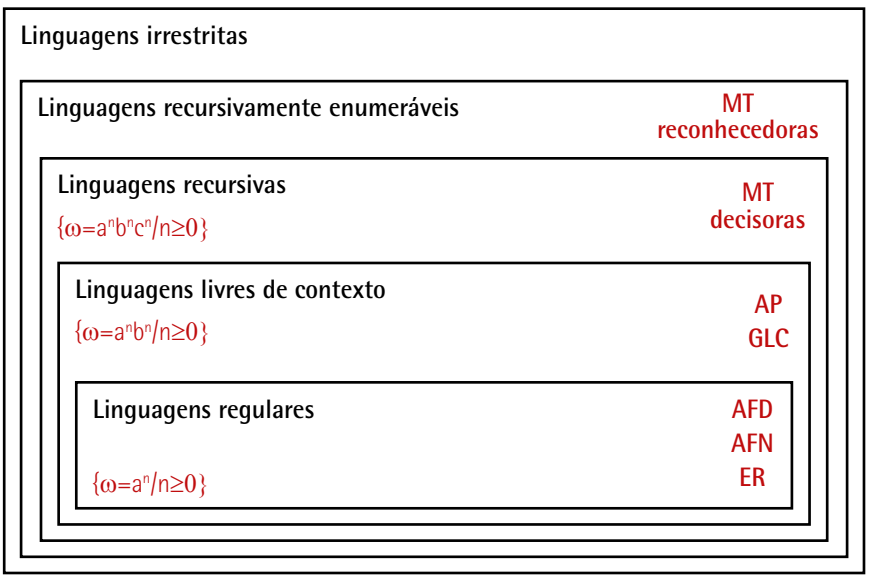

Figura 179 – Relação entre as classes de linguagem

Note que as linguagens regulares estão contidas nas linguagens livres de contexto, assim como estas estão nas linguagens recursivas, que, por sua vez, estão contidas nas linguagens recursivamente enumeráveis. Isso denota que podemos simular um AFD, ou AFN, AP em uma MT.

Exemplo de aplicação

Considere a linguagem A $=\{\omega\#\omega \,/\, \omega \in \{0,1\}^*\}$

a) Que tipo de cadeias formam A?

Resolução

As cadeias formadas por A são constituídas de três partes: a primeira e a última são constituídas por todas as concatenações de símbolos 0 e 1; a parte central é formada por um único símbolo de cerquilha (jogo da velha). Entretanto, é condição que a primeira e a última parte sejam iguais.

b) A cadeia de entrada 01101#01101 pertence à linguagem A?

Resolução

Observe que a cadeia pode ser separada em três partes:

$$\underbrace{01101}_{1^a \text{ parte}}\ \underbrace{\#}_{2^a \text{ parte}}\ \underbrace{01101}_{3^a \text{ parte}}$$

Como a 1ª e a 3ª partes são iguais, elas são separadas por um único símbolo de cerquilha. Logo, a cadeia 01101#01101 pertence à linguagem A.

c) Uma MT decide ou reconhece a linguagem A?

Resolução

Considere a MT M_A, que processará a cadeia dada na fita a seguir.

0	1	1	0	1	#	0	1	1	0	1	ß	ß

Em vez de resolvermos essa questão somente através das funções de transições de M_A, usaremos o que chamamos de **notação intermediária** da MT. Ela permite que imaginemos todo o processamento de cadeias de determinada linguagem, facilitando o processo de criação de uma MT.

Assim, o cabeçote de leitura de M_A executará:

- Um vaivém ao longo da fita, checando posições correspondentes de ambos os lados do símbolo # para verificar se essas posições contêm os mesmos símbolos.
- Se alguma posição não contém o mesmo símbolo ou o símbolo de # não for encontrado, rejeite.
- Marque os símbolos à medida que eles são verificados.
- Quando todos os símbolos à esquerda do símbolo # forem marcados, verifique se há símbolos não marcados à direita do símbolo #.
- Se houver, rejeite; caso contrário, aceite.

A seguir, temos o diagrama de M_A, que se inicia no estado q_1, lendo o símbolo 0 mais à esquerda.

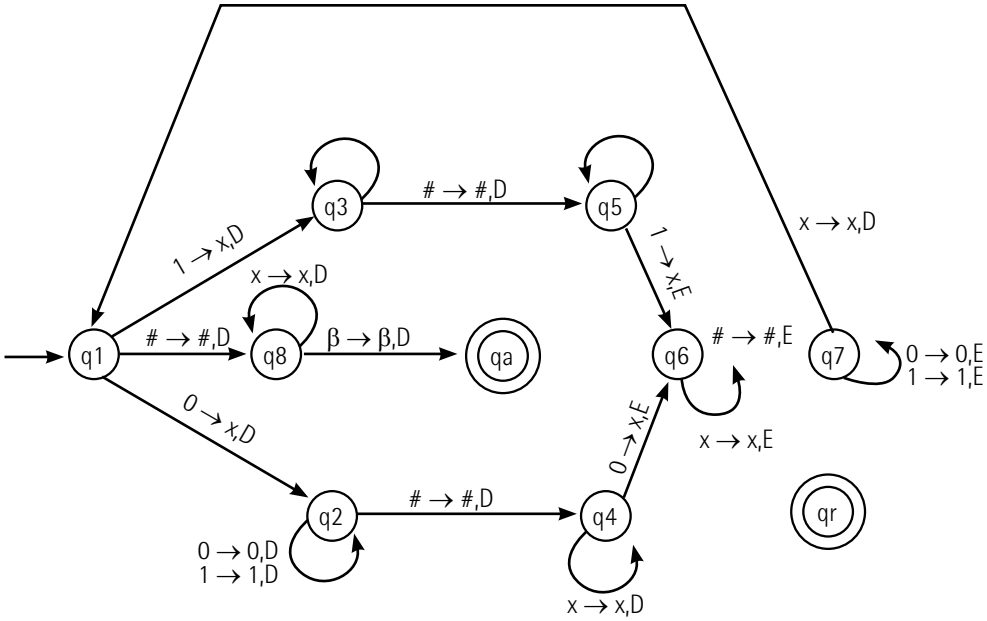

Figura 180

Veja o processamento:

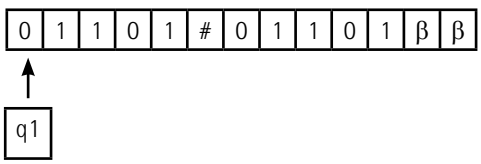

Figura 181

Estando em q_1, lendo 0, sobrescreva x, mova-se à direita e avance para q_2:

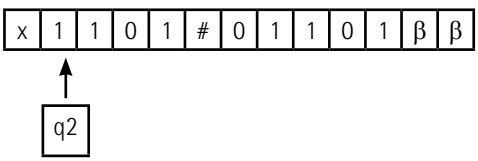

Figura 182

Estando em q_2, lendo 1, sobrescreva 1, mova-se à direita e permaneça em q_2:

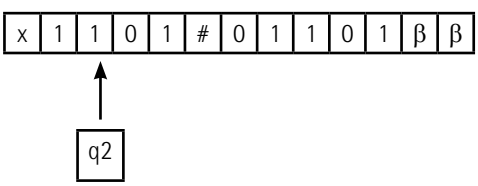

Figura 183

Estando em q_2, lendo 1, sobrescreva 1, mova-se à direita e permaneça em q_2:

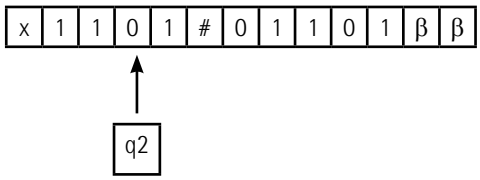

Figura 184

Estando em q_2, lendo 0, sobrescreva 0, mova-se à direita e permaneça em q_2:

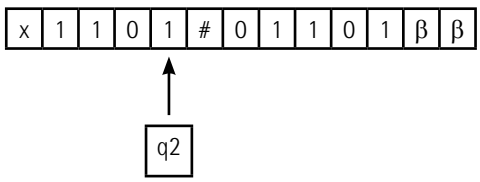

Figura 185

Estando em q_2, lendo 1, marque 1, mova-se à direita e permaneça em q_2:

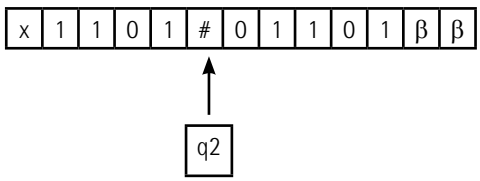

Figura 186

Estando em q_2, lendo #, sobrescreva #, mova-se à direita e avance para q_4:

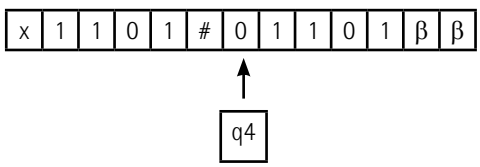

Figura 187

Estando em q_4, lendo 0, sobrescreva x, mova-se à esquerda e avance para q_6:

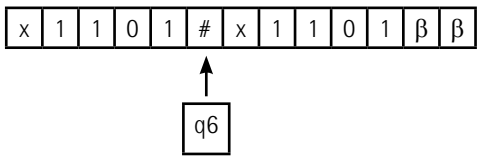

Figura 188

Estando em q_6, lendo #, sobrescreva #, mova-se à esquerda e avance para q_7:

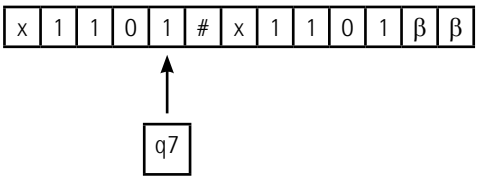

Figura 189

Estando em q_7, lendo 1, sobrescreva 1, mova-se à esquerda e permaneça em q_7:

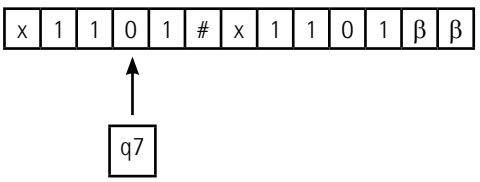

Figura 190

Estando em q_7, lendo 0, sobrescreva 0, mova-se à esquerda e permaneça em q_7:

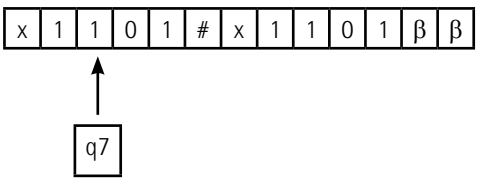

Figura 191

Estando em q_7, lendo 1, sobrescreva 1, mova-se à esquerda e permaneça em q_7:

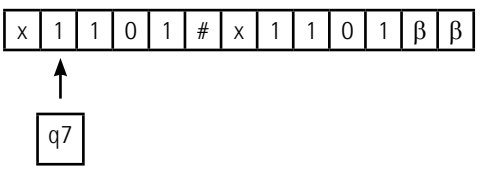

Figura 192

Estando em q_7, lendo 1, sobrescreva 1, mova-se à esquerda e permaneça em q_7:

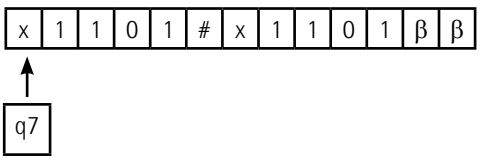

Figura 193

Estando em q_7, lendo x, sobrescreva x, mova-se à direita e avance para q_1:

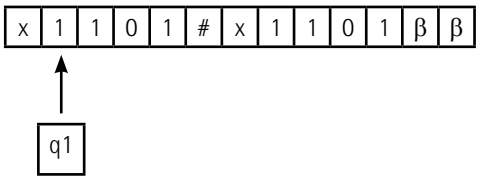

Figura 194

Estando em q_1, lendo 1, sobrescreva x, mova-se à direita e avance para q_3:

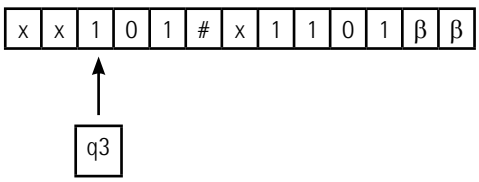

Figura 195

Estando em q_3, lendo 1, sobrescreva 1, mova-se à direita e permaneça em q_3:

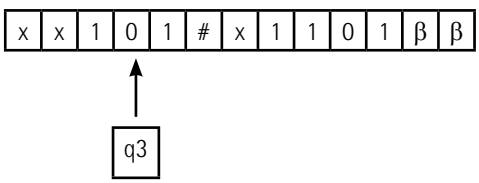

Figura 196

Estando em q_3, lendo 0, sobrescreva 0, mova-se à direita e permaneça em q_3:

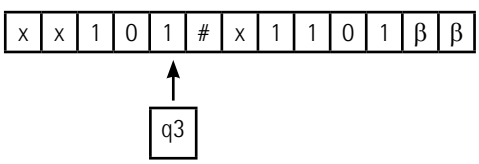

Figura 197

Estando em q_3, lendo 1, sobrescreva 1, mova-se à direita e permaneça em q_3:

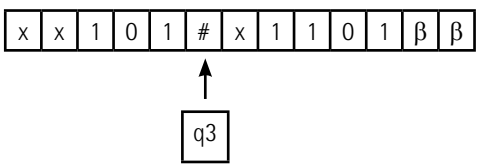

Figura 198

Estando em q_3, lendo #, sobrescreva #, mova-se à direita e avance para q_5:

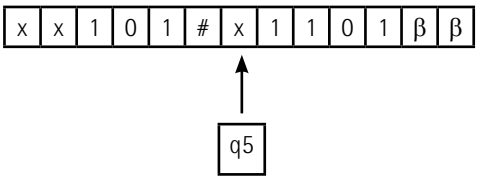

Figura 199

Estando em q_5, lendo x, sobrescreva x, mova-se à direita e permaneça em q_5:

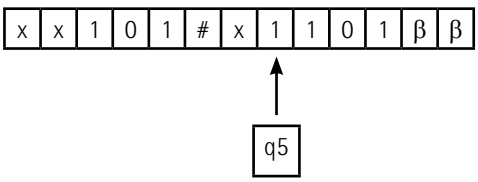

Figura 200

Estando em q_5, lendo 1, sobrescreva x, mova-se à esquerda e avance para q_6:

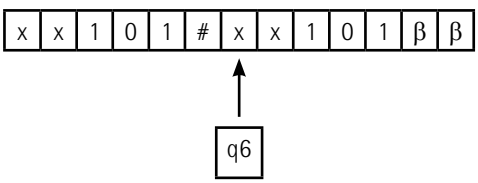

Figura 201

Estando em q_6, lendo x, sobrescreva x, mova-se à esquerda e permaneça em q_6:

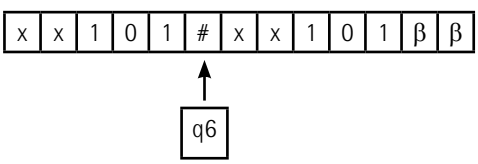

Figura 202

Estando em q_6, lendo #, sobrescreva #, mova-se à esquerda e avance para q_7:

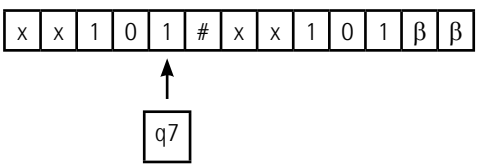

Figura 203

Estando em q_7, lendo 1, sobrescreva 1, mova-se à esquerda e permaneça em q_7:

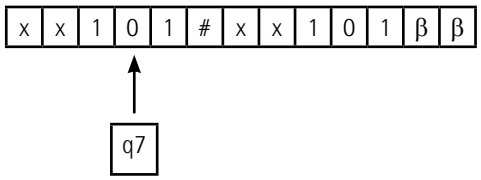

Figura 204

Estando em q_7, lendo 0, sobrescreva 0, mova-se à esquerda e permaneça em q_7:

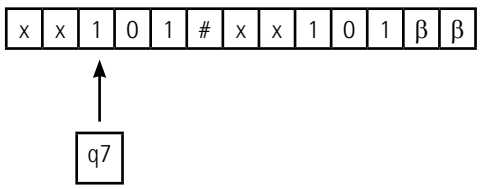

Figura 205

Estando em q_7, lendo 1, sobrescreva 1, mova-se à esquerda e permaneça em q_7:

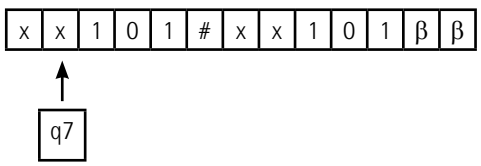

Figura 206

Estando em q_7, lendo x, sobrescreva x, mova-se à direita e avance para q_1:

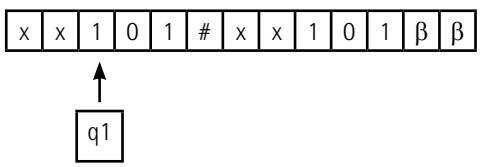

Figura 207

Estando em q_1, lendo 1, sobrescreva x, mova-se à direita e avance para q_3:

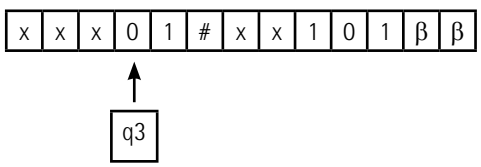

Figura 208

Estando em q_3, lendo 0, sobrescreva 0, mova-se à direita e permaneça em q_3:

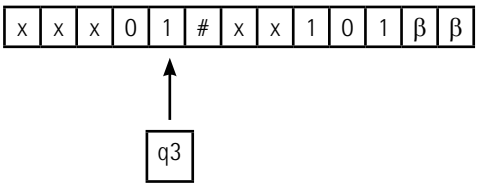

Figura 209

Estando em q_3, lendo 1, sobrescreva 1, mova-se à direita e permaneça em q_3:

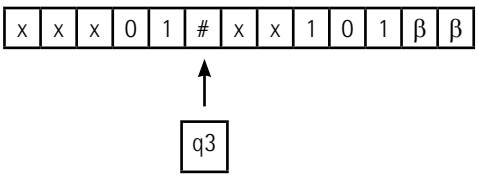

Figura 210

Estando em q_3, lendo #, sobrescreva #, mova-se à direita e avance para q_5:

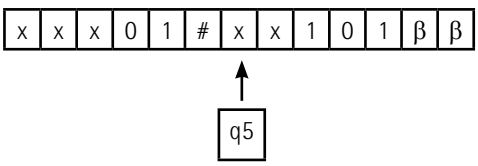

Figura 211

Estando em q_5, lendo x, sobrescreva x, mova-se à direita e permaneça em q_5:

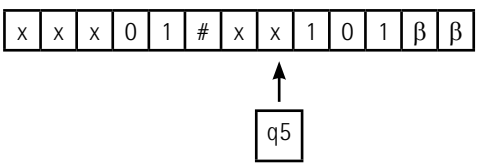

Figura 212

Estando em q_5, lendo x, sobrescreva x, mova-se à direita e permaneça em q_5:

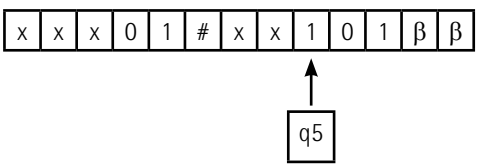

Figura 213

Estando em q_5, lendo 1, sobrescreva x, mova-se à esquerda e avance para q_6:

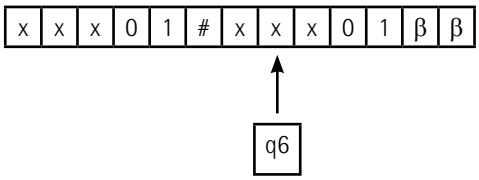

Figura 214

Estando em q_6, lendo x, sobrescreva x, mova-se à esquerda e permaneça em q_6:

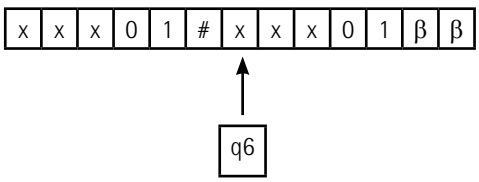

Figura 215

Estando em q_6, lendo x, sobrescreva x, mova-se à esquerda e permaneça em q_6:

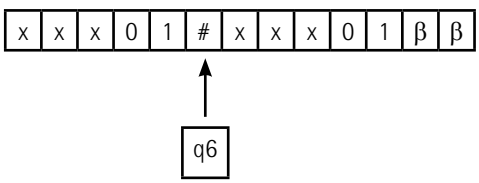

Figura 216

Estando em q_6, lendo #, sobrescreva #, mova-se à esquerda e avance para q_7:

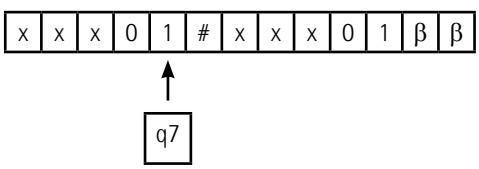

Figura 217

Estando em q_7, lendo 1, sobrescreva 1, mova-se à esquerda e permaneça em q_7:

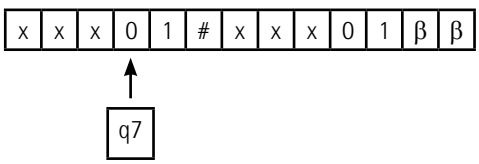

Figura 218

Estando em q_7, lendo 0, sobrescreva 0, mova-se à esquerda e permaneça em q_7:

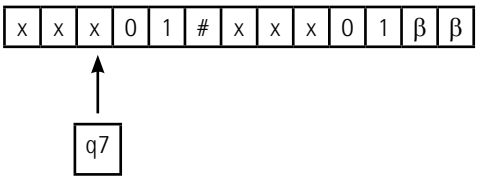

Figura 219

Estando em q_7, lendo x, sobrescreva x, mova-se à direita e avance para q_1:

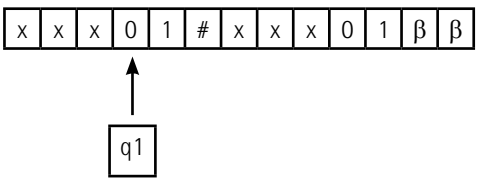

Figura 220

Estando em q_1, lendo 0, sobrescreva x, mova-se à direita e avance para q_2:

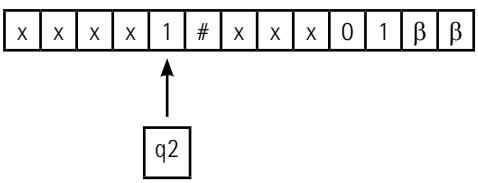

Figura 221

Estando em q_2, lendo 1, sobrescreva 1, mova-se à direita e permaneça em q_2:

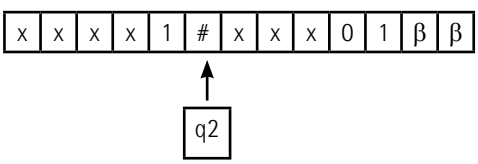

Figura 222

Estando em q_2, lendo #, sobrescreva #, mova-se à direita e avance para q_4:

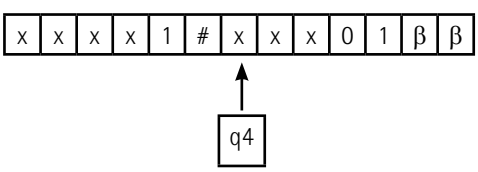

Figura 223

Estando em q_4, lendo x, sobrescreva x, mova-se à direita e permaneça em q_4:

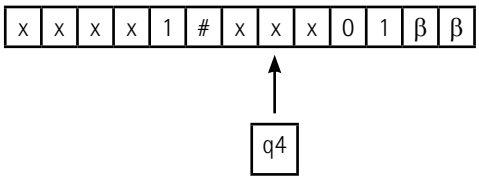

Figura 224

Estando em q_4, lendo x, sobrescreva x, mova-se à direita e permaneça em q_4:

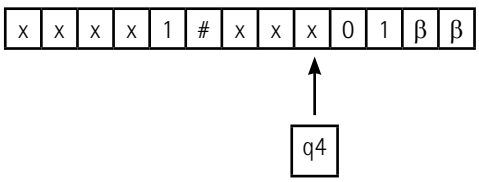

Figura 225

Estando em q_4, lendo x, sobrescreva x, mova-se à direita e permaneça em q_4:

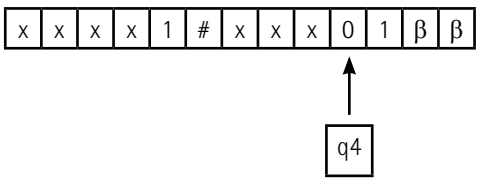

Figura 226

Estando em q_4, lendo 0, sobrescreva x, mova-se à esquerda e avance para q_6:

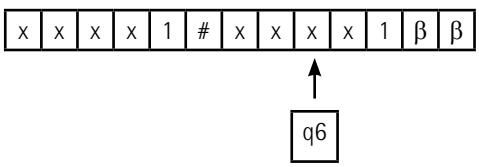

Figura 227

Estando em q_6, lendo x, sobrescreva x, mova-se à esquerda e permaneça em q_6:

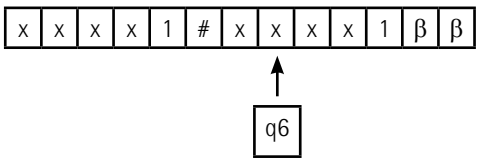

Figura 228

Estando em q_6, lendo x, sobrescreva x, mova-se à esquerda e permaneça em q_6:

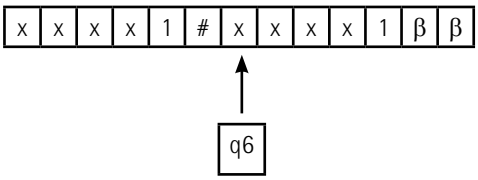

Figura 229

Estando em q_6, lendo x, sobrescreva x, mova-se à esquerda e permaneça em q_6:

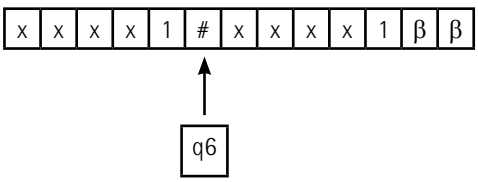

Figura 230

Estando em q_6, lendo #, sobrescreva #, mova-se à esquerda e avance para q_7:

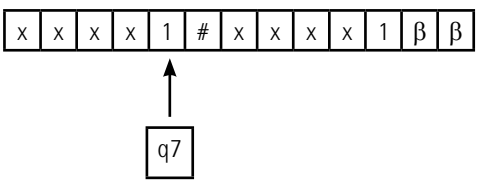

Figura 231

Estando em q_7, lendo 1, sobrescreva 1, mova-se à esquerda e permaneça em q_7:

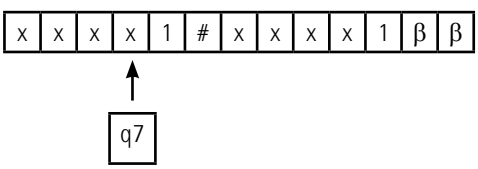

Figura 232

Estando em q_7, lendo x, sobrescreva x, mova-se à direita e avance para q_1:

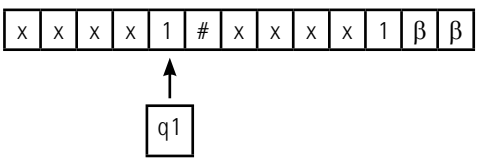

Figura 233

Estando em q_1, lendo 1, sobrescreva x, mova-se à direita e avance para q_3:

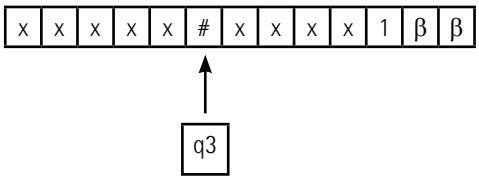

Figura 234

Estando em q_3, lendo #, sobrescreva #, mova-se à direita e avance para q_5:

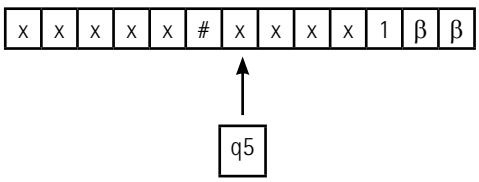

Figura 235

Estando em q_5, lendo x, sobrescreva x, mova-se à direita e permaneça em q_5:

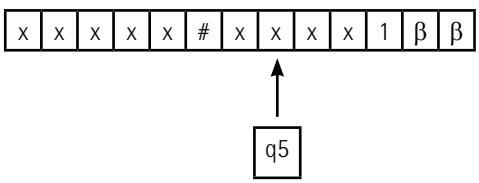

Figura 236

Estando em q_5, lendo x, sobrescreva x, mova-se à direita e permaneça em q_5:

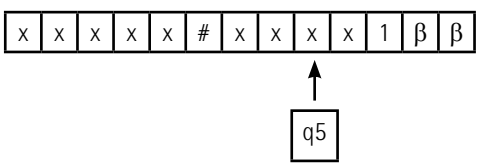

Figura 237

Estando em q_5, lendo x, sobrescreva x, mova-se à direita e permaneça em q_5:

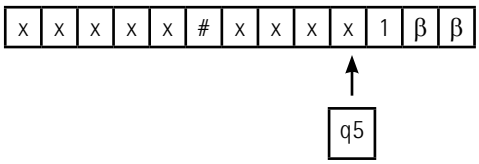

Figura 238

Estando em q_5, lendo x, sobrescreva x, mova-se à direita e permaneça em q_5:

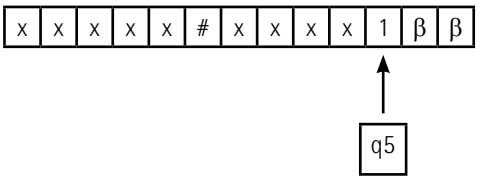

Figura 239

Estando em q_5, lendo 1, sobrescreva x, mova-se à esquerda e avance para q_6:

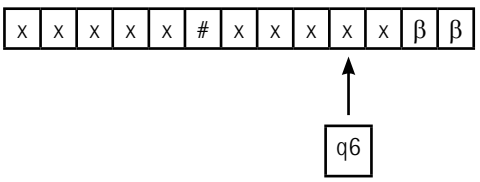

Figura 240

Estando em q_6, lendo x, sobrescreva x, mova-se à esquerda e permaneça em q_6:

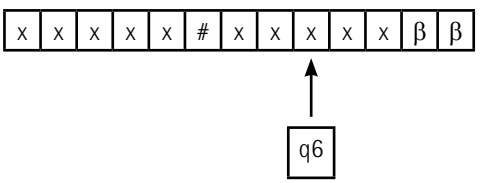

Figura 241

Estando em q_6, lendo x, sobrescreva x, mova-se à esquerda e permaneça em q_6:

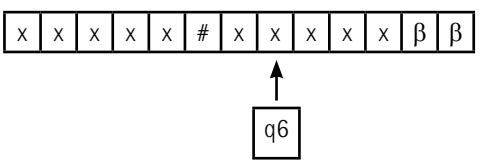

Figura 242

Estando em q_6, lendo x, sobrescreva x, mova-se à esquerda e permaneça em q_6:

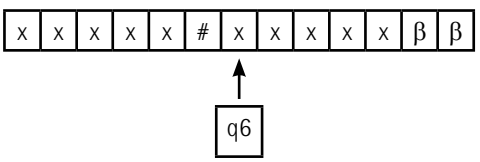

Figura 243

Estando em q_6, lendo x, sobrescreva x, mova-se à esquerda e permaneça em q_6:

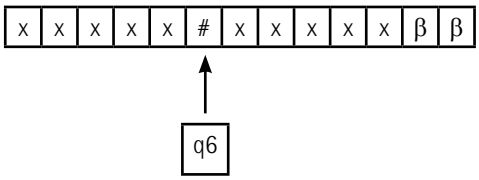

Figura 244

Estando em q_6, lendo #, sobrescreva #, mova-se à esquerda e avance para q_7:

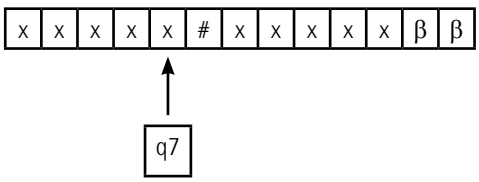

Figura 245

Estando em q_7, lendo x, sobrescreva x, mova-se à direita e avance para q_1:

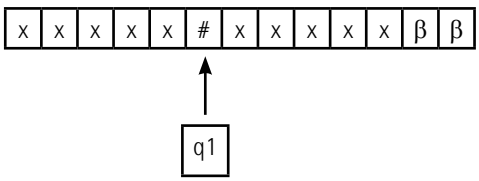

Figura 246

Estando em q_1, lendo #, sobrescreva #, mova-se à direita e avance para q_8:

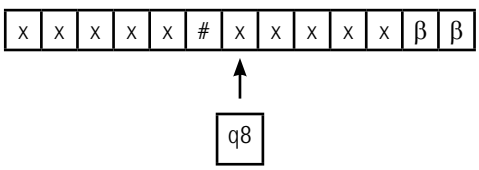

Figura 247

Estando em q_8, lendo x, sobrescreva x, mova-se à direita e parmaneça em q_8:

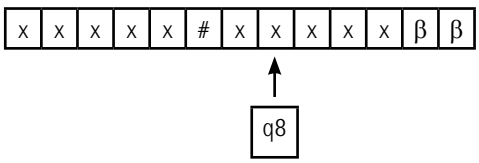

Figura 248

Estando em q_8, lendo x, sobrescreva x, mova-se à direita e parmaneça em q_8:

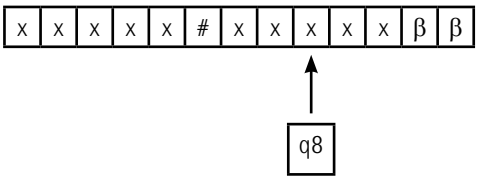

Figura 249

Estando em q_8, lendo x, sobrescreva x, mova-se à direita e parmaneça em q_8:

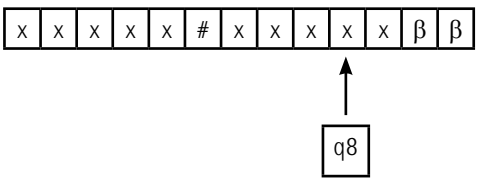

Figura 250

Estando em q_8, lendo x, sobrescreva x, mova-se à direita e parmaneça em q_8:

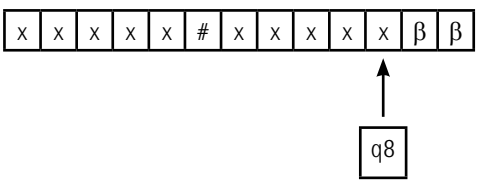

Figura 251

Estando em q_8, lendo x, sobrescreva x, mova-se à direita e parmaneça em q_8:

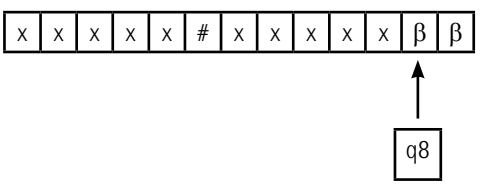

Figura 252

Estando em q_8, lendo β, sobrescreva β, mova-se à direita e avance para q_a.

Nesse momento, a MT M_A para e decide a linguagem A aceitando a cadeia 01101#01101.

A linguagem aceita por M_A é Turing decidível, pois qualquer transição não especificada levaria ao estado de rejeição, por exemplo, estando no estado q_8, lendo 0.

4.4 Variações da MT

Nas MT é possível implementar diversas variações, sem, contudo, aumentar o poder computacional. Isso quer dizer que uma MT tradicional é capaz de computar os mesmos algoritmos de suas variações, entretanto os desempenhos podem ser distintos.

Vejamos agora duas possíveis variações e seus processamentos. Paralelamente, será mostrado como uma MT tradicional também é capaz de cumprir a mesma tarefa.

MT cujo cabeçote de leitura pode não se mover

A definição da função de transição δ é:

$\delta = Q \times \Gamma \rightarrow Q \times \Gamma \times \{E, D, P\}$

Assim, além do cabeçote poder se mover à esquerda (E) e à direita (D), pode ficar parado (P).

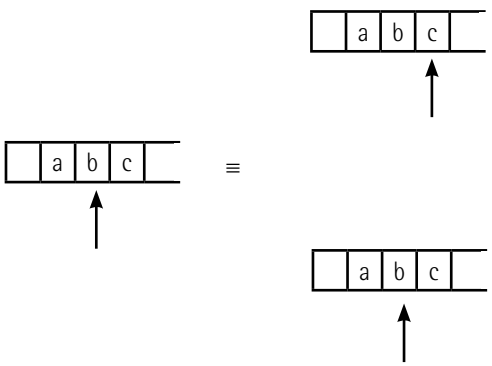

Figura 253

Nesse exemplo, à esquerda temos a variante da MT e, à direita, a MT tradicional. Dessa forma, supondo que na variante exista uma transição, cujo comando é o cabeçote de leitura ficar parado em b, equivalentemente, na MT tradicional, pode-se determinar uma transição de se mover, consecutivamente, à direita e à esquerda, sem sobrescrever nenhum símbolo.

MT com múltiplas (K) fitas

A definição da função de transição é:

$\delta = Q \times \Gamma^k \rightarrow Q \times \Gamma^k \times \{E, D\}^k$

Assim, considerando um certo estado e o conteúdo das K fitas, muda-se o estado, sobrescreve-se sobre as K fitas e sobre cada uma delas, e o cabeçote move-se à esquerda ou à direita.

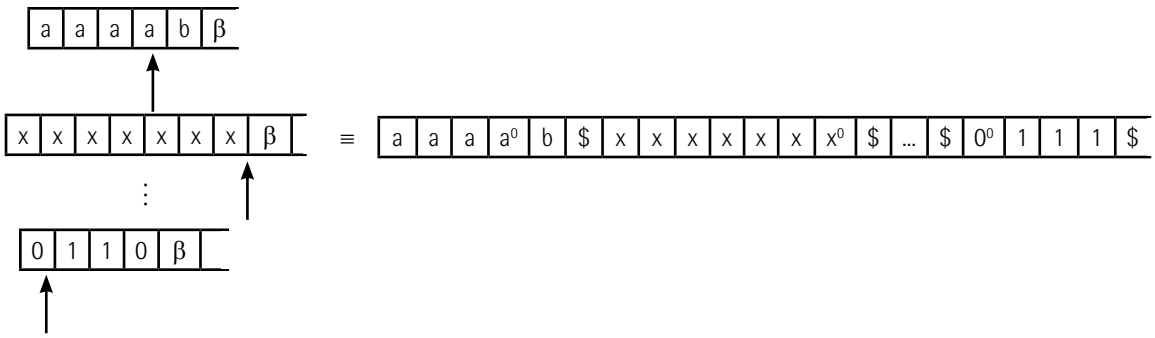

Figura 254

No exemplo anterior, à esquerda temos a variante da MT; à direita, a MT tradicional.

Da mesma forma, pode-se simular uma MT de K fitas com uma MT tradicional. Para isso, tome cada uma das K fitas da variante e compute-as uma por vez na tradicional, processo que ocorre incluindo um símbolo qualquer ($), que indique a mudança de fita.

Outra questão a acentuar é que uma MT de K fitas possui K cabeçotes. Assim, para representar, na MT tradicional, o local onde os K cabeçotes estão, adiciona-se, por exemplo, ^{0}.

Outras variações possíveis são as MT não determinísticas, as MT com fita ilimitada à esquerda e à direita e o enumerador. Para essa última variação, vale um adendo, pois é a partir dela que surge o conceito de linguagem recursivamente enumerável ou Turing reconhecível.

Enumerador

É uma MT tradicional com impressora. Começa com fita em branco e computa imprimindo cadeias. As cadeias impressas fazem parte das linguagens reconhecidas, denominadas recursivamente enumeráveis ou Turing reconhecíveis.

Resumo

Nesta unidade, estudamos os conjuntos, as sequências e as uplas, que desempenham papéis cruciais na estruturação e análise de dados, seja na matemática pura, seja na ciência da computação, seja em muitas outras disciplinas. Eles fornecem maneiras essenciais de organizar elementos e informações, permitindo-nos compreender e manipular dados de forma mais eficaz.

Em seguida, vimos que uma relação é um conjunto de pares ordenados que indicam conexões entre elementos de diferentes conjuntos. Essas relações podem ser usadas para conexões modelares, como "é parente de", "é maior que", ou "é amigo de". Elas são vitais na representação de estruturas complexas e na análise de conexões entre diferentes entidades.

Nesse contexto, acentuamos que uma função é uma relação especial em que cada elemento de um conjunto de partida (domínio) está associado exatamente a um elemento de um conjunto de chegada (contradomínio). As funções descrevem mapeamentos precisos e são fundamentais em muitos campos, desde matemática aplicada até ciência da computação. Elas permitem a modelagem de processos que transformam entradas em saídas específicas.

Um grafo é composto de vértices (ou nós) e arestas (ou conexões) que ligam os vértices. Grafos podem ser direcionados (com arestas com direção) ou não direcionados, ponderados (com valores nas arestas) ou não ponderados. Eles são usados para modelar uma variedade de situações do mundo real, desde redes sociais até sistemas de transporte. A teoria dos grafos fornece ferramentas para analisar propriedades encontradas, achar caminhos, estudar conexões e muito mais.

Destacamos também as linguagens livres de contexto, um tipo de linguagem na teoria das linguagens formais que pode ser descrito por uma gramática livre de contexto da gramática livre de contexto. Acentuamos, depois, a computabilidade, que é a capacidade de resolver problemas ou calcular funções matemáticas de forma algorítmica, geralmente usando um modelo teórico de computação, como a MT. Um problema é considerado computável se existir um algoritmo que pode produzir uma resposta correta para todas as entradas possíveis desse problema. A teoria da computabilidade explora os limites e possibilidades desses algoritmos e modelos de computação.

Prosseguindo, estudamos a tese de Church-Turing, um princípio fundamental na teoria da computação que postula que qualquer função computável pode ser calculada por uma MT. Por sua vez, a MT é um modelo teórico de um dispositivo de computação que consiste em uma fita infinita dividida em células, uma cabeça de leitura/gravação e um conjunto de estados finitos. Ela é capaz de resolver qualquer problema que possa ser computado algoritmicamente.

Por fim, vimos que existem diversas variações da MT, como as MT não determinísticas, as MT multifita, as MT quânticas etc. Essas variações são modelos teóricos que ajudam a explorar as fronteiras da computação e a compreender melhor as capacidades e limitações dos sistemas de computação.

Exercícios

Questão 1. (Cespe-Cebraspe/2022, adaptada) Um autômato finito determinístico (AFD) é um modelo matemático, utilizado na área computacional, que aceita ou rejeita cadeias de símbolos, gerando um único ramo de computação para cada cadeia de entrada. Uma das aplicações desse modelo é o processamento de linguagem natural. Nesse contexto, avalie as afirmativas a seguir.

I – Um AFD pode, para cada entrada, transitar a partir do seu estado atual em somente um estado.

II – Um AFD tem a capacidade de adivinhar algo sobre sua entrada ao testar valores.

III – Um AFD consegue estar em vários estados ao mesmo tempo.

É correto o que se afirma em:

A) I, apenas.

B) III, apenas.

C) I e III, apenas.

D) II e III, apenas.

E) I, II e III.

Resposta correta: alternativa A.

Análise das afirmativas

I – Afirmativa correta.

Justificativa: um AFD possui transições determinísticas. Isso significa que, para cada símbolo de entrada, ele pode transitar para um único estado seguinte.

II – Afirmativa incorreta.

Justificativa: um AFD não tem a capacidade de adivinhar valores. Ele apenas segue um conjunto fixo de regras de transição determinísticas, com base nos símbolos de entrada e no seu estado atual.

III – Afirmativa incorreta.

Justificativa: em um AFD, o autômato está em um único estado em determinado momento. O seu estado sucessor é definido pelo seu estado atual e pela condição de transição.

Questão 2. (IFB/2017) Considerando a definição de autômatos finitos, assinale a única alternativa que contém somente cadeias de caracteres totalmente aceitas pelo autômato finito da figura.

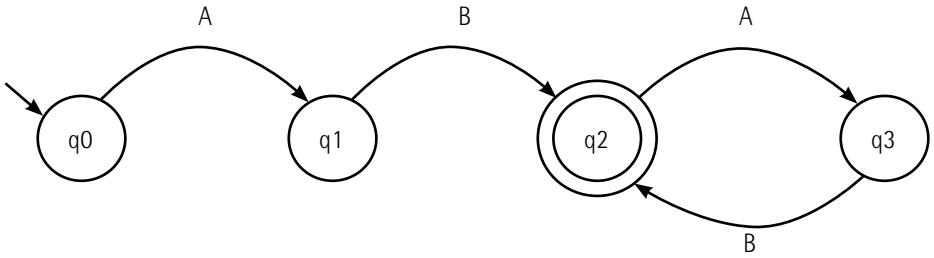

Figura 255

A) AB, ABAB, ABABAB.

B) AB, ABBA, ABABAB.

C) AB, ABAA, ABABAB.

D) AB, ABAB, ABBAAB.

E) AB, ABAB, ABAABA.

Resposta correta: alternativa A.

Análise da questão

O grafo da figura representa a função de transição de um autômato finito. O estado inicial é q_0, conforme indicado pela seta. O estado final, apontado pela circunferência dupla, é q_2.

A função de transição indica que, para que uma cadeia seja totalmente aceita, ela precisa ser composta apenas de caracteres A e B, sem repetições adjacentes do mesmo símbolo (como AA ou BB). Além disso, ela deve ser iniciada em A e terminada em B. Para entendermos melhor, vamos acompanhar o processamento das cadeias presentes na alternativa correta, conforme segue.

- **AB**: iniciando em q_0 e lendo o caractere A, ocorre a transição para q_1. Em q_1, lemos o caractere B, que ocasiona o estado final em q_2. Logo, essa cadeia é aceita.

- **ABAB**: iniciando em q_0 e lendo o caractere A, ocorre a transição para q_1. Em q_1, lemos o caractere B, e ocorre a transição para q_2. Em seguida, em q^2, lemos o caractere A, que ocasiona a transição para q_3. Em q_3, lemos o caractere B, e isso ocasiona o retorno a q_2. Lemos todos os caracteres da cadeia e encerramos a leitura no estado final q_2. Portanto, essa cadeia é aceita.

- **ABABAB**: iniciando em q^0 e lendo o caractere A, ocorre a transição para q^1. Em q^1, lemos o caractere B, e ocorre a transição para q^2. Em seguida, em q^2, lemos o caractere A, que ocasiona a transição para q^3. Em q^3, lemos o caractere B, e isso ocasiona o retorno a q^2. Novamente em q^2, lemos o caractere A, e vamos para q^3. Em q^3, lemos o último caractere B, que ocasiona um retorno a q^2. Lemos todos os caracteres da cadeia e encerramos a leitura no estado final q^2. Portanto, essa cadeia é aceita.