CENTRO UNIVERSITÁRIO
FMU
LEVEL UP

LINGUAGENS FORMAIS E AUTÔMATOS

https://portal.fmu.br

LINGUAGENS FORMAIS E AUTÔMATOS

Público-alvo: Estudantes do Centro Universitário FMU.

Objetivo: Esta apostila tem como objetivo fornecer a base matemática e teórica necessária para compreender como as linguagens — tanto naturais quanto de programação — são estruturadas, geradas e reconhecidas por máquinas. Ao estudar a Hierarquia de Chomsky, o aluno desenvolve a capacidade de abstrair problemas complexos e identificar o nível de computabilidade de cada desafio, estabelecendo os fundamentos para a construção de compiladores, analisadores léxicos e sistemas de processamento de texto.

Além do rigor teórico, a disciplina visa capacitar o estudante na aplicação prática de formalismos como Autômatos Finitos, Expressões Regulares e Gramáticas Livres de Contexto. Essa competência é essencial para o desenvolvimento de softwares eficientes, pois permite que o desenvolvedor escolha a ferramenta com a menor complexidade computacional necessária para uma tarefa, garantindo otimização de recursos e segurança na validação de dados.

Por fim, a disciplina explora os limites da computação através do estudo das Máquinas de Turing e da Teoria da Decidibilidade. O objetivo é provocar uma reflexão crítica sobre o que pode ou não ser resolvido por algoritmos, preparando o futuro profissional para lidar com problemas de alta complexidade lógica. Ao unir teoria e prática, a disciplina consolida o pensamento analítico indispensável para a criação de soluções tecnológicas inovadoras e robustas no campo da Ciência e Engenharia da Computação.

Professor: Robson Carmo, mestre pelo programa de mestrado profissional em Gestão e Tecnologia em Sistemas Produtivos do Centro Paula Souza, concluiu 3 MBAs (Inovação e Transformação Digital, Gestão de Empresarial, Engenharia de Software). Instrutor de certificações da SAFE, auto e coautor de vários livros e ebooks.

Lattes: http://lattes.cnpq.br/8753112683856964

Orcid: https://orcid.org/0009-0002-7102-4993

LLM utilizada: Gemini 3 – Fevereiro de 2026.

Módulos e Conteúdo

Unidade 1: Fundamentos Matemáticos e Introdução às Linguagens

- **Foco:** Atender aos objetivos de definição de conjuntos, relações, funções e a base das linguagens.
- **Conteúdo:** Revisão de matemática discreta (Menezes, Cap. 1); Definição formal de Alfabetos, Strings e Linguagens; A relação crucial entre as linguagens formais e a sintaxe das linguagens de programação reais.

Unidade 2: Autômatos Finitos Determinísticos (AFD)

- **Foco:** Modelagem e resolução de problemas práticos.
- **Conteúdo:** Características do AFD; Como mapear problemas do mundo real para estados e transições; Exercícios de aplicação prática em lógica de sistemas.

Unidade 3: Autômatos Finitos Não-Determinísticos (AFN) e Transições-ε

- **Foco:** Conceituação de AFN e o processo de conversão.
- **Conteúdo:** Funcionamento do AFN e AFN-ε; Algoritmo de conversão (subconjuntos) de AFN para AFD; Critérios de decisão: quando usar cada modelo?

Unidade 4: Expressões Regulares (ER) e Análise Léxica

- **Foco:** Operadores aritméticos, precedência e analisadores léxicos.
- **Conteúdo:** Sintaxe das ERs; Operadores e precedência; Construção de analisadores léxicos (base para compiladores); Equivalência entre ER e Autômatos Finitos (Teorema de Kleene).

Unidade 5: Gramáticas Regulares e a Hierarquia de Chomsky

- **Foco:** Regras gramaticais e classificação de linguagens.
- **Conteúdo:** Definição de Gramáticas Regulares; O Teorema do Bombeamento (Pumping Lemma) para provar que uma linguagem não é regular; Estudo detalhado da Hierarquia de Chomsky.

Unidade 6: Linguagens Livres de Contexto (LLC) e Gramáticas (GLC)

- **Foco:** Elaboração de gramáticas de linguagens de programação.
- **Conteúdo:** Funcionamento e aplicabilidade de GLC; Estratégias para resolver problemas de LLC; Como as linguagens que vocês usam (Java/C) são definidas por estas gramáticas.

Unidade 7: Autômatos de Pilha (AP)

- **Foco:** Análise sintática e recursividade.
- **Conteúdo:** A estrutura da pilha; Processamento de strings com memória auxiliar; Aplicação em compiladores para verificar o balanceamento de parênteses e estruturas recursivas.

Unidade 8: Máquinas de Turing (MT) e Decidibilidade

- **Foco:** Limites da computação e resolutividade.
- **Conteúdo:** Características e importância da MT; Linguagens enumeráveis recursivamente; Introdução à Teoria da Decidibilidade (o que o computador consegue ou não resolver).

Unidade 9: Prática Integrada - Modelagem e Implementação

- **Foco:** Abstração de problemas práticos e codificação.
- **Conteúdo:** Escolha do formalismo apropriado; Relação entre o modelo teórico e o código implementado; Uso intensivo do simulador JFLAP.

.

Unidade 1: Fundamentos Matemáticos, Alfabetos e Linguagens

Contexto Histórico

Para entendermos as Linguagens Formais, precisamos voltar ao início do século XX. Matemáticos como David Hilbert buscavam formalizar a matemática computacional. No entanto, foi Alan Turing e Alonzo Church que, independentemente, começaram a definir o que significa **"compute/computar"**.

De acordo com **Diverio (2011)**, a Teoria da Computação nasceu antes mesmo dos computadores físicos existirem. O objetivo era responder: ***O que pode ser resolvido de forma automática?*** Para responder a isso, precisávamos de uma linguagem que não tivesse as ambiguidades da língua portuguesa ou inglesa.

Se eu digo "O banco está ocupado", você pode pensar em um assento ou em uma instituição financeira. Na computação, a ambiguidade é um erro catastrófico. Por isso, as **Linguagens Formais**, como bem define **Menezes (2011)**, são conjuntos de cadeias de símbolos construídos sobre regras matemáticas estritas.

A Modelagem

Segundo **Menezes (2011)**, a computação não trata de máquinas físicas, mas de **MODELOS**. Um modelo é uma **abstração** que captura o **essencial** de um **sistema**. As Linguagens Formais são os modelos matemáticos que usamos para descrever a estrutura de tudo o que é processado.

Diverio (2011) complementa que esta teoria busca a "Máquina Universal". Antes de programarmos em Python ou Java, precisamos entender que o computador "enxerga" apenas sequências de símbolos organizadas sob regras que vamos estudar agora.

1.1 A Base Matemática: Conjuntos, Relações e Funções

Para entender como um computador processa informações, não podemos começar pelo código, mas pela matemática que o sustenta. **Menezes (2011)** afirma que a Teoria da Computação é, essencialmente, a aplicação de estruturas matemáticas discretas.

1.2.1 Teoria dos Conjuntos

Um conjunto é uma coleção de objetos distintos, chamados de elementos. Na nossa disciplina, o conjunto mais importante é o **Alfabeto**.

- **Pertencimento:** Se **x** é um elemento do conjunto **A**, denotamos $x \in A$.
- **Subconjuntos:** $A \subseteq B$ se todo elemento de **A** também pertence a **B**.

- **Conjunto das Partes *P*(A):** É o conjunto de todos os subconjuntos de **A**. Se **A = {0, 1}**, então ***P*(A) = {∅, {0}, {1}, {0, 1}}**. Este conceito é vital para entendermos como um Autômatos Não-Determinístico explora múltiplos estados simultaneamente.

O que é o Conjunto das Partes *P*(A)?

Imagine que você tem uma sacola com itens (o conjunto **A**). O **Conjunto das Partes** é uma lista de todas as **combinações possíveis** de itens que você pode tirar dessa sacola, incluindo a opção de não tirar nada (conjunto vazio) e a opção de tirar tudo.

Exemplo prático:

Se A = {0, 1}, as combinações são:

1. **Vazio:** ∅ (Não peguei nada)
2. **Unitários:** {0} e {1} (Peguei um de cada vez)
3. **Total:** {0, 1} (Peguei todos)

Assim, ***P***(A) = {∅, {0}, {1}, {0, 1}}.

Curiosidade Matemática: Se um conjunto tem **n** elementos, o seu Conjunto das Partes terá sempre 2^n elementos. No nosso exemplo, $2^2 = 4$.

O que é um Autômato?

De forma simples, um **autômato** é um modelo matemático de uma máquina que processa informações seguindo um conjunto de regras lógicas. Ele recebe uma entrada (como uma sequência de símbolos ou letras), muda de "estado" conforme lê essa entrada e, ao final, nos diz se aquela sequência é válida ou não.

Pense no autômato como um **fluxograma inteligente**.

1.1.2 Relações e Sequências

Diferente de um conjunto, onde a ordem não importa **({a, b} = {b, a})**, em uma **sequência** a ordem é fundamental. Uma string (ou cadeia) é, formalmente, uma **sequência finita** de símbolos.

- **Produto Cartesiano: A x B** é o conjunto de todos os pares ordenados **(a, b)**. A função de transição de um autômato opera sobre o **produto cartesiano** do conjunto de estados pelo alfabeto.

1.1.3 Funções

Uma função $f : A \rightarrow B$ associa cada elemento de **A** a exatamente um elemento de **B**. Na Unidade 2, veremos a função $\delta(q,\sigma) = q'$ (delta de **q** e **sigma** é igual a **q linha**), que é o "motor" lógico de qualquer autômato. Se essa função não for bem definida (determinística), o comportamento do software torna-se imprevisível.

Esta é a leitura direta dos símbolos gregos e latinos.

- δ **(delta minúsculo):** Representa a função de transição.
- σ **(sigma minúsculo):** Representa um símbolo do alfabeto.
- **q' (q linha):** O apóstrofo em matemática é lido como "linha", indicando um novo estado que deriva do original.

1.2 Alfabetos, Cadeias (Strings) e Linguagens

1.2.1 O Alfabeto (Σ) (sigma maiúsculo)

Como define **Sipser (2007)**, um alfabeto é um conjunto finito e não vazio de símbolos.

- **Exemplos Acadêmicos:** $\Sigma_1 = \{0, 1\}$ (binário) $\Sigma_2 = \{a, b, c, ..., z\}$ (latino).
- **Exemplos Práticos:** O conjunto de instruções de um processador ou o conjunto de caracteres válidos em uma URL.

1.2.2 Cadeias ou Strings

Uma cadeia é uma sequência finita de símbolos de um alfabeto.

- **Comprimento (|w|):** O número de símbolos.

  Se **w = abba**, $|w| = 4$.

  Quando colocamos algo entre barras verticais, no contexto de strings ou conjuntos, estamos falando de **comprimento** (ou cardinalidade).

  - Se w = "bola", então $|w| = 4$.
  - Se w = "A", então $|w| = 1$.

- **A Cadeia Vazia (ϵ):** É a sequência que não contém símbolos.

  Na computação e na matemática, a letra grega **ϵ** (epsilon) representa a **palavra vazia** (ou *string* vazia).

  - Imagine uma caixa de texto onde você não digitou nada.
  - Imagine uma lista que não contém nenhum caractere.

  Isso é o **ϵ**. Ele existe como um conceito (uma sequência de zero símbolos), mas não "ocupa espaço" visual.

|ϵ| lê-se como: **"O comprimento da palavra vazia"**.

A explicação da expressão |ϵ| = 0

A expressão afirma que **o comprimento da palavra vazia é igual a zero**.

Pode parecer óbvio, mas essa é uma regra crucial para algoritmos de processamento de texto e autômatos. Ela define que a palavra vazia é o **elemento neutro** da concatenação.

Se você juntar (concatenar) a palavra **"casa"** com a palavra vazia (**ϵ**), o resultado continua sendo "casa", e o comprimento total não muda:

$$|casa| + |\epsilon| = 4 + 0 = 4$$

- **Potência de Símbolos:** a^n significa o símbolo **a** repetido **n** vezes. $a^0 = \epsilon$.

1.2.3 Operações com Cadeias

1. **Concatenação:** Juntar ou concatenar **w = "casa"** e **v = "belo"** resulta em **wv = "casabelo"**.
2. **Reverso (w^R):** Se **w = "roma"**, **w^R = "amor"**.
3. **Prefixos e Sufixos:** "comp" é prefixo de "computador", enquanto "dor" é sufixo.

1.3 O Conceito Formal de Linguagem

Uma linguagem **L** sobre um alfabeto **Σ** é qualquer subconjunto de **Σ*** (lê-se sigma estrela). Ou seja, **$L \subseteq \Sigma^*$**.

O que é Σ*?

É o conjunto de todas as cadeias possíveis que podem ser geradas a partir de **Σ**, incluindo **ϵ**. Isso é chamado de **Fechamento de Kleene**.

- Se **Σ = {0, 1}**, então **Σ* = {ϵ, 0, 1, 00, 01, 10, 11, 000, ...}**.

A frase afirma que, embora o alfabeto (**Σ**) seja sempre finito, o conjunto de combinações que podemos formar com ele (**Σ***) é infinito.

Exemplo Prático e Didático

Pense no alfabeto da língua portuguesa: ele é finito (26 letras). No entanto, a regra para criar "palavras" (cadeias de caracteres) permite que você crie sequências de qualquer tamanho.

Se definirmos uma linguagem **L** como "todos os números pares representados em binário", o alfabeto é **Σ = {0, 1}**.

- As palavras seriam: {0, 10, 100, 110, 1000, ...}.
- Como a sequência de números pares nunca acaba, a linguagem **L** resultante é **infinita**, embora use apenas dois símbolos.

Diferença Importante: Σ* vs Σ⁺

Às vezes você verá um sinal de mais no lugar da estrela:

- **Σ* (estrela):** Inclui a palavra vazia (**ϵ**).
- **Σ⁺ (mais): Não** inclui a palavra vazia. Começa a partir de palavras com pelo menos um símbolo.

1.3.1 Tipos de Linguagens e a Hierarquia de Chomsky

Neste ponto, introduzimos a visão de **Diverio (2011)** sobre a classificação das linguagens. Nem todas as linguagens são iguais em complexidade:

1. **Linguagens Regulares (Tipo 3):** As mais simples, reconhecidas por Autômatos Finitos.

São as linguagens de menor complexidade. Elas não possuem memória de longo prazo ou capacidade de contagem infinita.

- **O Reconhecedor:** Autômatos Finitos (AFD/AFN).
- **A "Lógica":** Funcionam baseadas em estados. "Se estou aqui e recebo isso, vou para ali".
- **Exemplo Prático:** Validar um CPF, um e-mail ou uma senha que deve ter pelo menos uma letra maiúscula. Elas lidam com **padrões rígidos**.

2. **Linguagens Livres de Contexto (Tipo 2):** Usadas para descrever a estrutura de linguagens de programação.

Aqui a complexidade aumenta, pois precisamos de uma memória auxiliar (Pilha). Elas permitem **recursividade** e estruturas aninhadas.

- **O Reconhecedor:** Autômato de Pilha.
- **A "Lógica":** Conseguem lidar com "abre e fecha". Elas lembram o que foi aberto para garantir que será fechado na ordem correta.
- **Exemplo Prático:** A estrutura de um código em C ou Java. Por exemplo, garantir que cada chaves { tenha um par } correspondente, não importa quão profundo seja o aninhamento.

3. **Linguagens Sensíveis ao Contexto (Tipo 1) e Recursivamente Enumeráveis (Tipo 0):** As mais complexas, que exigem o poder de uma Máquina de Turing.

São o topo da pirâmide. Elas descrevem tudo o que é computável por um algoritmo moderno.

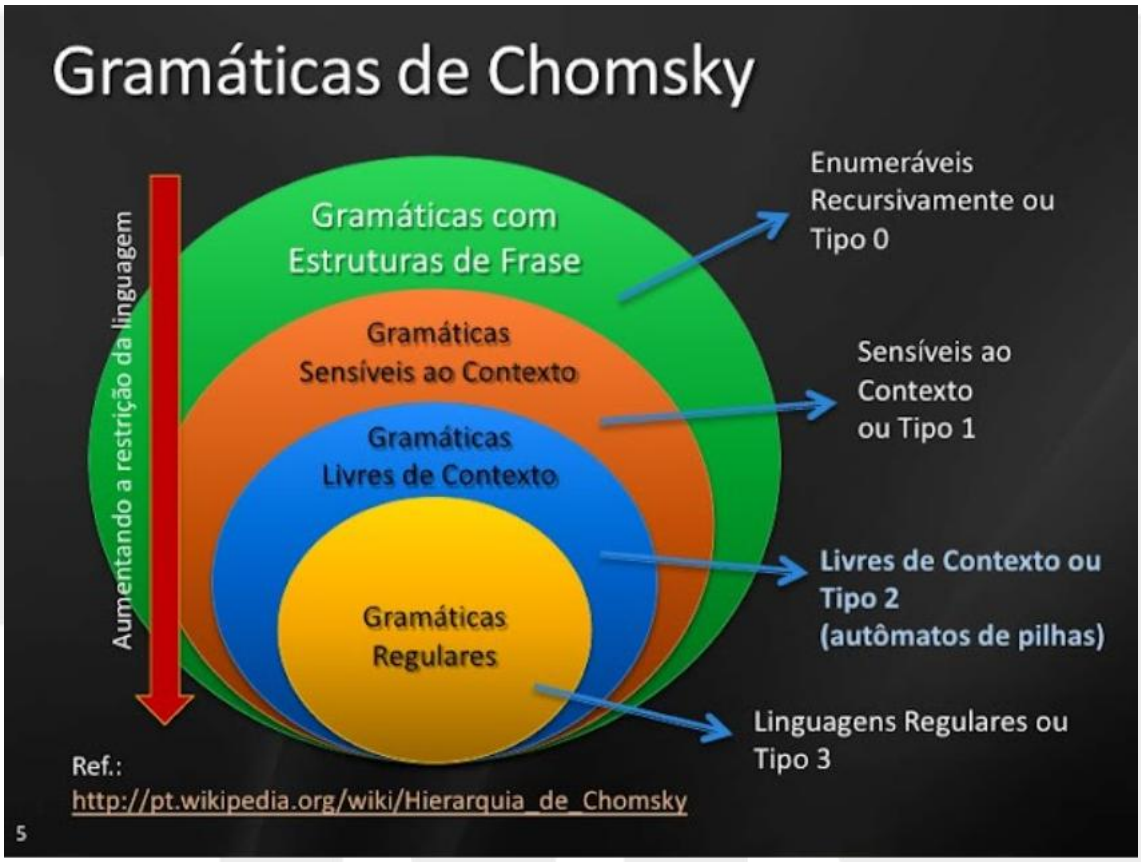
- **O Reconhecedor:** Máquina de Turing.
- **A "Lógica":** A máquina pode ler e escrever em qualquer lugar de uma fita infinita. Ela pode "voltar atrás" e revisar o que fez.
- **Linguagens Sensíveis ao Contexto:** A regra depende do que está em volta (contexto). Ex: Verificar se uma variável foi declarada antes de ser usada.
- **Recursivamente Enumeráveis:** É a classe de tudo o que um computador consegue resolver. Se existe um passo a passo (algoritmo) para resolver, a Máquina de Turing consegue processar.

Por que isso é importante?

A **Hierarquia de Chomsky** prova que uma linguagem de nível superior pode fazer tudo o que uma de nível inferior faz, mas o contrário não é verdadeiro.

Analogia Prática:

- Um **Autômato Finito** (Tipo 3) é como um termostato (simples decisão).
- Um **Autômato de Pilha** (Tipo 2) é como uma calculadora de parênteses.
- A **Máquina de Turing** (Tipo 0) é o seu computador completo.



Fonte: https://radiomaffei.blogspot.com/2012/12/esse-cara-vale-pena-ser-lido.html

Linguagens Sensíveis ao Contexto (LSC) (Tipo 1)

Essas linguagens exigem que o autômato consiga comparar quantidades de três ou mais elementos diferentes ao mesmo tempo. Um autômato com pilha comum não consegue fazer isso (ele só compara dois), por isso usamos um **Autômato Linearmente Limitado (LBA)**.

- Exemplo Clássico: $L = \{a^n b^n c^n \mid n \geq 1\}$
  - **O que é:** Palavras que têm uma quantidade **n** de **a**, seguida pela **mesma** quantidade **n** de **b** e a **mesma** quantidade **n** de **c**.
  - **Exemplos de palavras:** abc, aabbcc, aaabbbccc.

- **Por que é sensível ao contexto?** Porque para saber se o número de 'c' está correto, você precisa "lembrar" do contexto anterior de 'a' e 'b'.

- Outro exemplo: $L = \{ ww \mid w \in \{a, b\}^* \}$ (o problema da cópia)
  - **O que é:** Uma palavra seguida por uma cópia idêntica de si mesma.
  - **Exemplos:** abab, baabaa, aaaa.

Linguagens Recursivamente Enumeráveis (LRE) (Tipo 0)

Este é o nível máximo. Elas são reconhecidas pela **Máquina de Turing**. Se uma linguagem pode ser descrita por um algoritmo (por mais complexo que seja), ela está aqui.

- Exemplo: O Problema da Parada (Halting Problem)
  - Imagine um conjunto de todos os programas de computador que eventualmente terminam de rodar (não entram em loop infinito). Tentar decidir se um programa qualquer pertence a esse conjunto é o exemplo supremo de LRE.
- **Exemplo Matemático:** L = { w | w descreve um grafo que possui um caminho hamiltoniano}.
  - Basicamente, qualquer problema que um computador moderno consegue "verificar" ou "tentar resolver" entra nesta categoria.

O que é um Caminho Hamiltoniano?

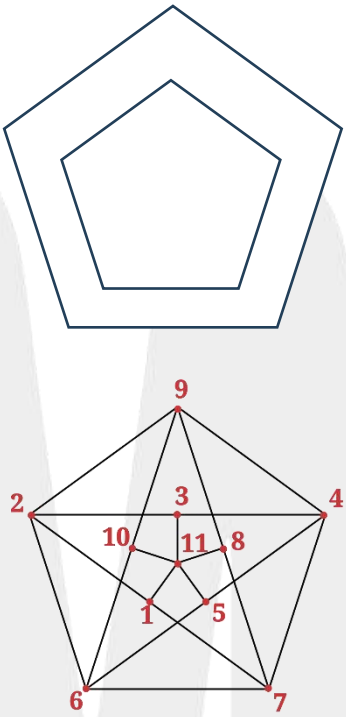
Um caminho hamiltoniano em um grafo é um trajeto que passa por **todos os vértices exatamente uma vez**.

- **A dificuldade:** Encontrar esse caminho é um problema clássico de alta complexidade (NP-completo). Não existe uma fórmula simples ou uma Expressão Regular que resolva isso; é necessário testar combinações e seguir uma lógica algorítmica profunda.

Decomposição da Frase

- **L:** É a linguagem (o conjunto de todas as respostas válidas).
- **w:** É a "palavra" ou entrada. Neste caso, w não é apenas uma sequência de letras, mas uma **codificação de um grafo** (pense em uma matriz de adjacência ou uma lista de conexões convertida em binário).

- **w descreve um grafo...:** A frase diz que a linguagem L só aceita strings que, quando traduzidas, formam grafos que possuem o tal caminho.



Fonte: https://pt.wikipedia.org/wiki/Caminho_hamiltoniano

Tipo de Linguagem	Autômatos (Máquina)	Exemplo Típico
Sensível ao Contexto	Autômato Linearmente Limitado	$a^n b^n c^n$
Recursivamente Enumerável	Máquina de Turing	Qualquer algoritmo computável

Onde isso se encaixa na prática?

Linguagens Sensíveis ao Contexto são usadas na análise de linguagens de programação reais, onde você precisa verificar se uma variável foi declarada antes de ser usada (o "contexto" da declaração).

1.4 Linguagens Formais vs. Linguagens de Programação

Toda linguagem de programação (Python, C, Java) tem duas partes:

1. **Sintaxe:** Definida por Linguagens Formais. O compilador usa um autômato para verificar se você esqueceu um ; ou se escreveu if corretamente.
2. **Semântica:** O significado do que foi escrito.

Quando você define que uma variável deve começar com uma letra e pode conter números, você está definindo uma **Linguagem Regular**. O processo de "tokenização" em um compilador é, puramente, a aplicação de Autômatos Finitos e Expressões Regulares para validar strings.

I. Perguntas de Múltipla Escolha

1. **O que é o "Conjunto das Partes" P(A) de um conjunto A = {a, b}?**

a) {a, b}

b) {Ø, {a}, {b}}

c) {Ø, {a}, {b}, {a, b}}

d) {a, b, ab}

2. **Qual a principal diferença entre um conjunto e uma sequência?**

a) Conjuntos são sempre infinitos.

b) Em sequências, a ordem dos elementos importa; em conjuntos, não.

c) Sequências não podem conter números.

d) Conjuntos não podem ser vazios.

3. **O que representa o símbolo ε na Teoria das Linguagens Formais?**

a) Um erro de sintaxe.

b) O alfabeto completo.

c) A cadeia vazia (comprimento zero).

d) O símbolo final de uma string.

4. Se Σ = {0, 1}, qual das cadeias abaixo NÃO pertence a Σ*?

a) 00000

b) ϵ

c) 1021

d) 111

5. A operação de Fechamento de Kleene (Σ^*) resulta sempre em um conjunto:

a) Finito.

b) Vazio.

c) Infinito (se o alfabeto não for vazio).

d) Composto apenas por cadeias de mesmo tamanho.

6. O que é um prefixo próprio de uma cadeia w?

a) É qualquer subcadeia de **w**.

b) É um prefixo de **w**, tal que o prefixo é diferente da própria **w**.

c) É o reverso da cadeia **w**.

d) É a cadeia vazia concatenada com **w**.

7. Segundo Menezes (2011), uma linguagem formal é:

a) Um software de tradução.

b) Qualquer subconjunto de **Σ***.

c) Uma sequência de comandos em linguagem Assembly.

d) Um alfabeto sem símbolos repetidos.

8. Qual a relação entre Autômatos Finitos e Compiladores?

a) Autômatos são usados para criar o design gráfico do compilador.

b) Autômatos são usados na análise léxica para reconhecer palavras válidas (tokens).

c) Compiladores não utilizam autômatos, apenas inteligência artificial.

d) Autômatos servem para aumentar a velocidade do processador.

9. Se $|u| = 3$ e $|v| = 4$, qual o comprimento da concatenação $|uv|$?

a) 12

b) 7

c) 34

d) 1

10. O que Diverio (2011) define como Alfabeto?

a) Um conjunto infinito de caracteres Unicode.

b) Um conjunto finito e não vazio de símbolos.

c) Uma lista de palavras reservadas do Java.

d) Qualquer sequência de bits.

11. Dada a cadeia w = "abc", qual o seu reverso w^R?

a) "abc"

b) "cba"

c) "bca"

d) "ϵ"

12. As linguagens de programação como C++ são classificadas majoritariamente como:

a) Linguagens Regulares.

b) Linguagens Livres de Contexto (na sua estrutura sintática).

c) Linguagens de Sinais.

d) Linguagens Finitas.

II. Perguntas Reflexivas

1. **Abstração:** Como a matemática de conjuntos permite que um mesmo compilador funcione em computadores com hardwares totalmente diferentes? Reflita sobre a independência do modelo formal em relação à máquina física.

2. **Linguagem Natural vs. Formal:** Por que não usamos o Português para programar computadores? Quais os riscos da ambiguidade linguística para a computação segundo a visão de Sipser?

III. Exercícios Práticos

1. **Operações de Conjuntos:** Dado o alfabeto $\Sigma = \{x, y, z\}$, liste todos os elementos de Σ^2 (cadeias de comprimento 2) e calcule o tamanho do conjunto das partes $P(\Sigma)$.

2. **Concatenação e Potência:** Se $L = \{01, 10\}$, determine o conjunto resultante de L^2.

3. **Identificação de Padrões:** Escreva uma definição formal (usando notação de conjuntos) para a linguagem L sobre $\Sigma = \{a, b\}$ que contém todas as cadeias que começam e terminam com a mesma letra.

Unidade 2: Autômatos Finitos Determinísticos (AFD)

2.1 A Natureza dos Autômatos Finitos

Segundo **Menezes (2011)**, um autômato finito é o modelo mais simples de computação. Ele é ideal para sistemas que possuem uma "memória extremamente limitada". Ao contrário de um computador moderno que tem Gigabytes de RAM, um autômato finito "lembra" apenas em qual **estado** ele está no momento.

A palavra **"Determinístico"** é a chave: significa que, para qualquer estado atual e qualquer símbolo lido, existe **apenas um** caminho possível. Não há sorte, escolha ou ambiguidade.

Características Principais

- **Sem Memória Extra:** Diferente de um computador comum, o AFD não tem um "HD" ou "Pilha" para guardar dados extras. Sua única memória são os estados.
- **Tempo Real:** Ele processa a entrada um símbolo por vez, da esquerda para a direita, e nunca volta atrás.
- **Eficiência:** É o modelo mais rápido de processamento, usado por compiladores para a fase de Análise Léxica (identificar palavras-chave, números e variáveis).

2.1.1 O que é um Estado?

Um estado representa uma etapa específica no processamento de uma informação. De acordo com **Diverio (2011)**, a mudança de um estado para outro é disparada por um evento externo (um símbolo do alfabeto).

Exemplo Clássico: O Semáforo

- **Estado q_0:** Verde.
- **Evento:** Sensor de tempo esgotado.
- **Transição:** O sistema muda para o Estado q_1 (Amarelo).
- **Determinismo:** No sistema ideal, não há dúvida. Se o tempo acabou e está no Verde, o *único* destino possível é o Amarelo.

2.2 Definição Formal (a 5-tupla) (quíntupla)

Para garantir o rigor acadêmico solicitado por **Sipser (2007)**, definimos um AFD como uma estrutura matemática composta por cinco elementos:

$$M = (Q, \Sigma, \delta, q_0, F):$$

1. **Q (Conjunto de Estados):** Um conjunto finito. Ex: $\{q_0, q_1, q_2\}$.
2. **Σ (Alfabeto):** Conjunto de símbolos que a máquina pode ler. Ex: {0, 1}.
3. **δ (Função de Transição):** É o mapa da máquina.

   Formalmente: $\delta: Q \times \Sigma \rightarrow Q$

   Lê-se:

   Delta leva **Q vezes sigma em Q)**

   ou

   A função de **transição delta** mapeia o **produto cartesiano** do **conjunto de estados** pelo **alfabeto de entrada** no próprio **conjunto de estados**.

   Leitura Funcional (lógica):

   Se estou no **estado X** e leio o **símbolo Y**, vou para o **estado Z.**

   ou

   A função de transição recebe um **estado atual** e um **símbolo do alfabeto** e resulta em um **novo estado.**
4. **q_0 (Estado Inicial):** Onde tudo começa. $q_0 \in Q$.
5. **F (Conjunto de Estados de Aceitação):** Um subconjunto de Q que contém os estados onde, se a leitura terminar, a palavra é considerada "válida" ou "aceita".

2.3 Modelagem de Problemas com AFD

Problema para ser modelado:

Projetar um sistema que só aceite números binários com uma quantidade par de 0

Passo 1: Definir os Estados

Precisamos de dois estados:

- q_{par}: Estado onde o número de zeros lidos até agora é par (incluindo zero 'zeros').
- q_{impar}: Estado onde o número de zeros lidos é ímpar.

Passo 2: Definir as Transições

Tabela de Transição de Estados (δ)

Estado Atual (δ)	Entrada '0'	Entrada '1'	Descrição Lógica
→*q_{par}	q_{impar}	q_{par}	Se ler '0', a contagem vira ímpar (vai para Q_{impar}). Se ler '1', a quantidade de zeros não muda.
q_{impar}	q_{par}	q_{impar}	Se ler '0', a contagem volta a ser par (vai para S_{par}). Se ler '1', a quantidade de zeros continua ímpar.

i. **O Símbolo** → (estado inicial)

A seta apontando para o estado indica que ele é o **ponto de partida** da máquina.

- Significa que, assim que o autômato é ligado (ou começa a processar uma cadeia), ele se encontra obrigatoriamente neste estado.

ii. **O Símbolo** * (estado de aceitação)

O asterisco indica que este é um **estado final** ou de **aceitação**.

- Se a leitura da sua sequência de bits terminar em um estado marcado com *, a máquina dirá "Sim" (cadeia aceita).

Sem o *, você sabe como a máquina se move, mas não sabe se a palavra que ela leu é válida ou inválida.

iii. **3. Os nomes q_{par} e q_{impar}**

Este é apenas o rótulo (nome) que foi dado ao estado para facilitar a compreensão do problema.

- Neste contexto, ele indica que a máquina está em uma condição onde a contagem de zeros lidos até o momento é **par**.

Para ler esta matriz, cruzamos a linha do estado onde a máquina se encontra com a coluna do símbolo que ela acabou de ler:

1. **Linha q_{par}** (estado inicial e de aceitação):
    - Se ler **0**: A contagem de zeros, que era par, torna-se ímpar. O sistema migra para q_{impar}.
    - Se ler **1**: A quantidade de zeros não muda. O sistema permanece em q_{par}.

2. **Linha q_{impar}:**
    - Se ler **0**: A contagem de zeros, que era ímpar, torna-se par novamente. O sistema retorna para q_{par}.
    - Se ler **1**: A quantidade de zeros permanece ímpar. O sistema permanece em q_{impar}.

Na ciência da computação, essa matriz é a base para a implementação de **Matrizes de Adjacência** em grafos e para a otimização de analisadores léxicos. Em vez de múltiplos if/else, o computador simplesmente consulta as coordenadas da matriz (estado, símbolo) para saber o próximo passo, o que torna o processamento extremamente veloz.

Passo 3: Definir Aceitação

Como queremos "quantidade par", o estado de aceitação **F** será $\{q_{par}\}$.

Definição Formal do Autômato *M*

O autômato finito determinístico que reconhece a linguagem de cadeias binárias com quantidade par de zeros é definido pela **sétupla:**

$$M = (Q, \Sigma, \Gamma, \delta, q_0, F, \sqcup)$$

$Q = \{q_{par}, q_{impar}\}$: É o conjunto finito de estados.

$\Sigma = \{0, 1\}$: É o alfabeto de entrada (símbolos permitidos na cadeia).

$\Gamma = \{0, 1, \sqcup\}$: É o alfabeto da fita (em AFDs simplificados, usamos apenas Σ, mas a sétupla formal prevê o alfabeto completo de trabalho).

$q_0 = q_{par}$: É o estado inicial, onde a máquina começa o processamento.

$F = \{q_{par}\}$: É o conjunto de estados de aceitação (finais).

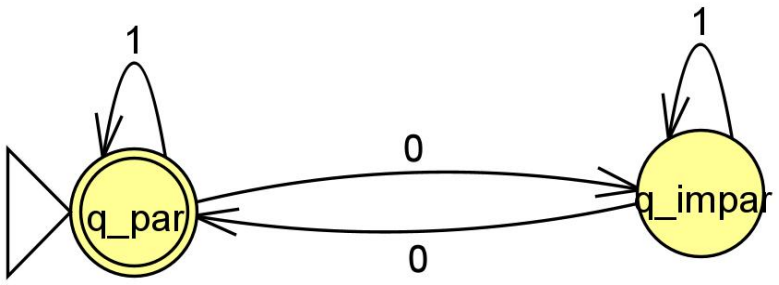
$\sqcup$: Símbolo branco (espaço vazio), relevante para a terminação da leitura.

δ: É a função de transição, definida abaixo:\

Matriz de Transição (δ) – entrada: 1010

Passo	Símbolo Lido	Transição	Estado Resultante
0	-	(Início)	q_{par}
1	1	$\delta(q_{par}, 1)$	q_{par}
2	0	$\delta(q_{par}, 0)$	q_{impar}
3	1	$\delta(q_{impar}, 1)$	q_{impar}
4	0	$\delta(q_{impar}, 0)$	q_{par} (aceita)

Desenho deste modelo no JFLAP



Modelagem do problema no JFLAP

Exemplos de Resultados

Entrada	Resultado
0001	Reject
0010	Reject
0100	Reject
1000	Reject

1100	Accept
1110	Reject
1111	Accept
0000	Accept
0101	Accept

Poque aceitou 1111???

Justificativa Técnica: O Conceito de Paridade e o Elemento Neutro

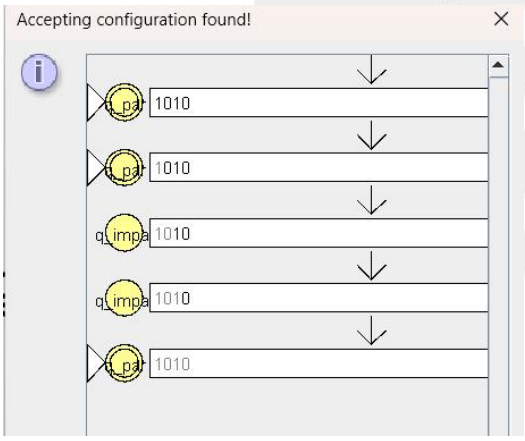
Ao analisar os resultados obtidos no simulador JFLAP, observa-se que cadeias compostas exclusivamente pelo símbolo '1' (como 1111) ou a própria cadeia vazia são classificadas como **ACEITAS**. Este comportamento é teoricamente correto e esperado para um Autômato Finito Determinístico (AFD) que modela a **paridade par de zeros**.

Segundo a definição matemática de paridade, o número **zero é um número par** (0 (mod 2) = 0). Visto que o AFD utiliza o estado inicial q_{par} também como estado de aceitação, o sistema assume que, antes de processar qualquer símbolo '0', a contagem acumulada é zero. Como o símbolo '1' não altera o estado de contagem de zeros (representado por um *self-loop* nos estados), uma cadeia sem zeros mantém o autômato em sua configuração inicial de aceitação. Portanto, a aceitação de **1111** valida que o modelo respeita rigorosamente a propriedade de paridade para o conjunto de números naturais, incluindo o zero.

2.4 Representações: Diagramas vs. Tabelas

Menezes (2011) destaca que o **Diagrama de Estados** é melhor para a compreensão humana, enquanto a **Tabela de Transição** é superior para a implementação computacional.

Diagrama de Estado



Novo Problema

Modelar um autômato onde **qualquer entrada que não seja o símbolo '0'** force a transição da máquina **para o estado q_1**. Assumindo que q_0 é o nosso estado inicial, aqui está a configuração dessa lógica:

Tabela de Transição de Estado para o alfabeto {0, 1}:

Estado Atual	Símbolo '0'	Símbolo '1'
→ q_0	q_0	q_1
*q_1	q_0	q_1

Análise do Comportamento

1. **Regra do '0':** Nesta modelagem, convencionamos que o '0' mantém a máquina no estado q_0 ou a faz retornar para ele. Ele atua como o símbolo que "reseta" ou mantém a estabilidade no estado inicial.
2. **Regra do '1' (diferente de 0):** Conforme solicitado, qualquer valor diferente de 0 (neste caso, o '1') dispara o gatilho que move a máquina obrigatoriamente para o estado **q_1**.
3. **Persistência em q_1:** Se a máquina já estiver em q_1 e ler outro '1', ela permanece em q_1. Se ler um '0', ela volta para q_0.

Representação Matemática (Função δ)

A leitura formal desta tabela seria:

- $\delta\ (q_0, 0) = q_0$
- $\delta\ (q_0, 1) = q_1$
- $\delta(q_1, 0) = q_0$
- $\delta\ (q_1, 1) = q_1$

Nesta tabela, qualquer palavra que contenha pelo menos um 'a' levará a máquina ao estado q_1 e lá ela permanecerá (estado de aceitação).

Definição Formal do Autômato *M*

$Q = \{q_0, q_1\}$: É o conjunto finito de estados.

$\Sigma = \{0, 1\}$: É o alfabeto de entrada.

$\Gamma = \{0, 1, \sqcup\}$: É o alfabeto da fita.

q_0 = É o estado inicial.

$F = \{q_1\}$: É o conjunto de estados de aceitação (indicado pelo *).

$\sqcup$: Símbolo branco de preenchimento.

δ: É a função de transição mapeada.

Novo Problema

Em muitas linguagens, uma instrução só é válida se terminar com uma sequência específica. Vamos projetar um AFD que aceite apenas cadeias binárias que **terminem com a sequência "01"**.

Passo 1: Definição dos Estados (Q)

Precisamos de estados que representem o "progresso" da máquina na leitura da sequência:

- q_0: Estado inicial (não leu nada de relevante ainda).
- q_1: "Encontrei um **0**". A máquina está em alerta, esperando um '1' para completar a sequência.
- q_2: "Encontrei **01**". Estado de aceitação.

Passo 2: Matriz de Transição (δ)

A lógica aqui é de "retrocesso": se a máquina estiver quase completando a sequência e ler o símbolo errado, ela deve voltar para o estado que faça sentido.

Estado Atual	Entrada '0'	Entrada '1'	Descrição
→ q_0	q_1	q_0	Se ler '0', avança. Se ler '1', ignora.
q_1	q_1	q_2	Se ler outro '0', continua esperando o '1'. Se ler '1', aceita!
*q2	q_1	q_0	Se ler '0', volta para o alerta (q1). Se ler '1', volta ao início.

Passo 3: A Sétupla Formal

$$M = (\{q_0, q_1, q_2\}, \{0,1\}, \{\mathbf{0}, \mathbf{1}, \sqcup\}, \delta, q_0, \{q_2\}, \sqcup)$$

M = (Q, Σ, Γ, δ, q0, F, Ш)

2.5 Processamento de Cadeias e a Função de Transição Estendida

Como a máquina processa uma string inteira? **Sipser (2007)** introduz a ideia da função **δ*** (delta estrela). Ela não olha apenas para um símbolo, mas para uma cadeia inteira.

Se w = 011, a máquina faz:

$\delta(q_0, 0) \rightarrow q_1$

$\delta(q_1, 1) \rightarrow q_2$

$\delta(q_2, 1) \rightarrow q_3$

Se $q_3 \in F$, a palavra **w** é aceita. Caso contrário, é rejeitada.

Importante: Em um AFD, a leitura é sempre da esquerda para a direita e a máquina nunca volta atrás.

I. Perguntas de Múltipla Escolha

1. **Qual o papel da função de transição δ em um AFD?**

a) Definir quais são os símbolos do alfabeto.

b) Determinar o próximo estado com base no estado atual e no símbolo lido.

c) Armazenar o resultado final do processamento.

d) Reiniciar a máquina em caso de erro.

2. **Quantos estados iniciais um AFD pode ter segundo a definição formal?**

a) Quantos forem necessários para o problema.

b) No mínimo dois.

c) Exatamente um.

d) Nenhum, se a linguagem for vazia.

3. O que caracteriza um estado como sendo "de aceitação"?

a) Ele é o último estado do diagrama.

b) Ele possui uma seta de entrada, mas nenhuma de saída.

c) Ele é representado por dois círculos concêntricos e indica que a cadeia lida é válida.

d) É o estado que consome mais memória.

4. Em um AFD, o que acontece se um estado não tiver uma transição definida para um símbolo do alfabeto?

a) O autômato aceita a cadeia por padrão.

b) A definição de AFD exige que todo estado tenha exatamente uma transição para cada símbolo do alfabeto.

c) O autômato entra em modo de espera.

d) A cadeia é processada novamente do início.

5. Se a cadeia vazia (ϵ) é aceita por um autômato, o que podemos afirmar sobre q_0?

a) q_0 deve ser um estado de aceitação ($q_0 \in F$).

b) q_0 não pode ter transições de saída.

c) O autômato não possui alfabeto.

d) q_0 é um "estado morto".

6. A Tabela de Transição é uma forma de representar:

a) Apenas os estados finais.

b) A função δ de forma matricial.

c) O histórico de todas as cadeias já lidas.

d) A hierarquia de Chomsky.

7. Segundo Diverio (2011), o que é um "Estado Morto" (ou estado de erro)?

a) Um estado que não pertence ao conjunto Q.

b) Um estado de onde não é possível alcançar nenhum estado de aceitação.

c) O estado inicial quando a máquina é desligada.

d) Um estado que aceita todas as cadeias possíveis.

8. **Dada a 5-tupla, o conjunto F é sempre:**

a) Igual ao conjunto Q.

b) Um subconjunto de Q.

c) Um conjunto com apenas um elemento.

d) O conjunto vazio.

9. **O determinismo em um AFD garante que:**

a) A máquina nunca erre o resultado.

b) Para um mesmo par (estado, símbolo), o destino seja sempre único e previsível.

c) O tempo de execução seja sempre zero.

d) A linguagem seja sempre infinita.

10. **Qual destes é um exemplo prático de aplicação de AFD na computação?**

a) Renderização de vídeos em 3D.

b) Reconhecimento de palavras reservadas em um compilador (Análise Léxica).

c) Cálculo de rotas em GPS complexos.

d) Mineração de criptomoedas.

11. **Se um alfabeto é $\Sigma = \{0, 1\}$ e o conjunto de estados é $Q = \{q_0, q_1\}$, quantas transições a função δ deve definir ao todo?**

a) 2

b) 4 (2 estados times 2 símbolos)

c) 1

d) Infinitas

12. **Menezes (2011) afirma que os Autômatos Finitos reconhecem a classe das Linguagens:**

a) Livres de Contexto.

b) Regulares.

c) Sensíveis ao Contexto.

d) Naturais.

II. Perguntas Reflexivas

1. **Limitação de Memória:** Como a impossibilidade de um AFD "voltar atrás" na fita de leitura limita o tipo de problema que ele pode resolver? (Pense em como você verificaria se uma expressão matemática tem parênteses balanceados).

2. **Modelagem Real:** Por que é mais seguro usar um AFD para validar uma senha ou um CPF do que simplesmente escrever vários comandos if/else soltos no código?

II. Exercícios Práticos

1. **Desenho Técnico:** Modele um AFD para o alfabeto $\Sigma = \{a, b\}$ que aceite apenas cadeias que terminem com o sufixo "ba". Forneça o conjunto de estados Q, a função δ e o estado final F.

2. **Análise de Linguagem:** Descreva, em linguagem natural, qual o padrão de strings aceito por um autômato que possui dois estados (q_0 e q_1), onde q_0 é inicial e final, e qualquer símbolo do alfabeto {0, 1} alterna entre os dois estados.

Unidade 3: Autômatos Finitos Não-Determinísticos (AFN) e Transições-ϵ

3.1 O Conceito de Não-Determinismo

De acordo com **Sipser (2007)**, o não-determinismo é uma generalização do determinismo. Enquanto no AFD, para cada estado e símbolo, existe exatamente uma transição, no **Autômato Finito Não-Determinístico (AFN)**, podem existir:

- **Várias transições** para o mesmo símbolo a partir de um único estado.
- **Nenhuma transição** para um determinado símbolo.
- **Transições sem ler nenhum símbolo** (transições vazias ou ϵ).

AFN, formalmente, difere dos **AFDs** apenas por sua função da transição:

$$\delta: Q \times \Sigma \rightarrow P(Q)$$

Lê-se: A **função de transição** recebe um **estado** e um **símbolo** e retorna um **subconjunto de estados possíveis**, ou, a função de transição delta mapeia o produto cartesiano do conjunto de estados pelo alfabeto no conjunto das partes (ou power set) do conjunto de estados."

δ (delta): A regra de transição.

Q × Σ: O domínio. Indica que a função opera sobre a combinação de um estado atual e um símbolo lido.

→ (seta): Indica o mapeamento para o contradomínio.

P(Q): Aqui está a grande diferença para o AFD. Enquanto no AFD o resultado é um único estado (Q), no AFN o resultado é um **conjunto de estados**.

Menezes (2011) explica que o não-determinismo não existe fisicamente no hardware convencional, mas é uma ferramenta de modelagem poderosíssima. Imagine um software que precisa decidir entre vários caminhos possíveis: o AFN "explora" todos simultaneamente. Se *qualquer* um desses caminhos levar a um estado de aceitação, a cadeia é aceita.

3.2 AFN com Transições Vazias (AFN-ϵ)

O **AFN-ϵ** é uma extensão que permite que o autômato mude de estado "espontaneamente", ou seja, sem consumir nenhum símbolo da entrada.

Diverio (2011) destaca que as transições-ϵ facilitam a união de diferentes linguagens. Se temos um autômato que reconhece "nomes" e outro que reconhece "números",

podemos criar um novo estado inicial que aponta para ambos via transição-ϵ, permitindo que o sistema aceite qualquer um dos dois de forma simplificada.

3.2.1 Fecho-ϵ (Fecho-Vazio) (ϵ -closure)

O conceito de Fecho-ϵ de um estado q é o conjunto de todos os estados que podem ser alcançados a partir de **q** usando apenas **transições vazias**. Esta é a base para a conversão de modelos.

3.3 Conversão de AFN para AFD (Algoritmo de Construção de Subconjuntos)

AFDs e AFNs são equivalentes em poder de reconhecimento. Todo **AFD** é um **AFN** e, portanto, o não-determinismo não representa um aumento de capacidade dos autômatos.

No entanto, o AFN é muitas vezes muito menor e mais simples de desenhar. O processo de conversão, detalhado por **Menezes (2011)**, segue estes passos:

1. O estado inicial do novo AFD é o Fecho-ϵ do estado inicial do AFN.
2. Para cada novo estado criado (que na verdade é um conjunto de estados do AFN), calculamos para onde ele vai com cada símbolo do alfabeto.
3. Um estado no AFD será de aceitação se pelo menos um dos estados do AFN contidos nele for de aceitação.

Característica	AFD	AFN
Destino da Transição	Para cada par (estado, símbolo), existe **exatamente um** próximo estado.	Para um par (estado, símbolo), podem existir **zero, um ou vários** próximos estados.
Cadeia Vazia (ϵ)	Não permite transições sem ler um símbolo (**ϵ** não é permitido).	Permite mudar de estado sem ler nenhum símbolo (Transições- **ϵ**).
Função de Transição (δ)	$\delta : Q \times \Sigma \rightarrow Q$ ** A função de transição resulta em um elemento	$\delta: Q \times \Sigma \rightarrow P(Q)$ ** A função de transição resulta em um conjunto
Implementação	Mais fácil (direta)	Mais difícil (exige simulação)

Tamanho	Geralmente maior	Pode ser muito menor
Previsibilidade	O caminho é único e fácil de rastrear (Determinístico).	Permite "palpites" ou múltiplos caminhos simultâneos.

3.4 Quando escolher AFD ou AFN?

- **Use AFN quando:** O problema envolve múltiplos padrões possíveis ou quando a simplicidade do desenho for prioritária. É excelente para a fase de **concepção e abstração**.

- **Use AFD quando:** For implementar o autômato em código. Como o hardware é determinístico, o AFD traduz-se diretamente em tabelas de busca eficientes e rápidas (Complexidade O(n) onde n é o tamanho da string).

3.4.1 A notação O(n)

A notação **O(n)**, lida como **"Ordem de n"** ou **"Complexidade Linear"**, é um dos conceitos mais fundamentais na análise de algoritmos. Em termos simples: **O(n) significa que o tempo de execução do seu código cresce na mesma proporção que o tamanho dos seus dados.**

Imagine que você tem uma lista de nomes e precisa encontrar um específico:

- Se a lista tem **10** nomes, o computador faz (no pior caso) **10** operações.
- Se a lista cresce para **1 milhão** de nomes, o computador fará **1 milhão** de operações.

O gráfico dessa complexidade é uma **linha reta ascendente**.

Comparação com outras ordens:

Notação	Nome	Exemplo Real	Performance
O(1)	Constante	Acessar um item pelo índice em um array.	Ultrarrápido
O(log n)	Logarítmica	Busca binária (dividir para conquistar).	Muito eficiente
O(n)	Linear	Percorrer uma lista; busca simples.	Justo/Aceitável??
O(n2)	Quadrática	Dois loops aninhados (um for dentro de outro).	Perigoso / Lento

Problema usando AFN

Projetar um autômato que aceite qualquer cadeia de caracteres que contenha a subcadeia **"web"**.

Por que usar um AFN aqui?

Se usássemos um AFD, teríamos que prever, para cada estado, o que fazer com todas as 26 letras do alfabeto. No **AFN**, podemos usar o "não-determinismo" a nosso favor:

- O estado inicial q_0 simplesmente ignora qualquer letra e continua em q_0.
- Mas, de forma não-determinística, se ele ler um 'w', ele também pode "palpitar" que a palavra começou e avançar para q_1.

Modelagem do AFN (Σ = {a, b, ..., z})

i. **Estados (Q):**

- **q_0:** Estado de busca. Continua lendo qualquer letra.
- **q_1:** "Encontrei um **w**".
- **q_2:** "Encontrei **we**".
- **q_3:** "Encontrei **web**" (Estado de Aceitação).

ii. **Função de Transição (δ):**

Aqui vemos a mágica do AFN na primeira linha. Ao ler 'w', o autômato está em dois lugares ao mesmo tempo: continua procurando o início da palavra e inicia a validação da palavra encontrada.

Tabela de Transição de Estados (δ)

Estado Atual	Entrada 'w'	Entrada 'e'	Entrada 'b'	Outras letras
→q_0	{q_0 ,q_1 }	{q_0 }	{q_0 }	{q_0 }
q_1	Ø	{q_2 }	Ø	Ø
q_2	Ø	Ø	{q_3 }	Ø
q_3*	{q_3 }	{q_3 }	{q_3 }	{q3 }

** A Tabela de Transição de um AFN deve ser interpretada como um **mapa de computação paralela**. Cada célula representa todos os estados possíveis que o sistema pode assumir após processar um símbolo. Se, ao final da leitura, pelo menos um dos estados no conjunto atual for um estado de aceitação (com *), a entrada é considerada válida.

Passo a passo para ler e interpretar a Tabela de Transição

a) O Significado das Chaves { }

Sempre que você vir um conjunto como $\{q_0, q_1\}$, a interpretação correta é: **"A máquina se divide"**.

- **Na prática:** Ao ler o símbolo 'w' no estado q_0, o autômato não escolhe entre ficar ou ir; ele cria uma cópia de si mesmo e passa a existir nos dois estados ao mesmo tempo.

b) Leitura das Linhas e Colunas

Vamos analisar a primeira linha (Estado q_0):

- **Cruzamento q_0 com 'w' → $\{q_0, q_1\}$**: Significa que o 'w' pode ser apenas uma letra qualquer da frase (mantendo a busca em q_0) **OU** pode ser o início da palavra "web" (iniciando o rastro em q_1).
- **Cruzamento q_0 com 'e' → $\{q_0\}$**: Significa que um 'e' lido isoladamente não inicia a palavra "web", então a única opção é continuar procurando no estado inicial.

c) O Símbolo de Conjunto Vazio Ø

Em um AFN, o símbolo Ø (vazio) é muito comum.

- **Interpretação:** Significa que aquele caminho "morreu".
- **Exemplo:** Na linha do q_1 (que significa "acabei de ler um 'w'"), se a próxima letra for um 'w', o resultado é Ø. Isso indica que a tentativa de formar a palavra "web" naquele ponto específico falhou, e aquela "cópia" da máquina é descartada.

d) O Estado de "Captura" (Self-loop)

Observe a linha do estado q_3*:

- Todas as entradas resultam em $\{q_3\}$.

- **Interpretação:** Uma vez que a subcadeia "web" foi encontrada, a máquina entra em um estado de "sucesso permanente". Não importa o que venha depois (outras letras ou espaços), a cópia que chegou em q_3 garante que a string inteira será aceita no final.

iii. Sétupla Formal

M = (Q, Σ, Γ, δ, q0, F, ⊔), onde:

- Q = $\{q_0, q_1, q_2, q_3\}$
- Σ = {a, b, ..., z}
- $q_0 = q_0$
- F = $\{q_3\}$
- δ é a função mapeada na tabela acima

Problema usando AFN: Validador de Complexidade de Senhas

Um sistema de segurança exige que uma senha binária (para simplificar nosso alfabeto) contenha a sequência 010 (um padrão de segurança) OU a sequência 101 (um padrão alternativo). Se qualquer uma dessas sequências aparecer em qualquer lugar da senha, ela é considerada "forte o suficiente".

Modelagem do AFN (Σ = {0, 1})

i. Estados (Q):

- **q1:** Estado de busca inicial.
- **q1, q2:** Caminho para reconhecer 010.
- **q3, q4:** Caminho para reconhecer 101.
- **Q5:** Estado de aceitação (encontrou um dos padrões).

ii. Tabela de Transição (δ)

Aqui, as chaves { } mostram o não-determinismo. Note como o q_0 pode iniciar os dois caminhos simultaneamente:

Estado Atual	Entrada '0'	Entrada '1'	Descrição do Momento
→q0	{q0 ,q1 }	{q0 ,q3 }	Continua em q0 ou tenta os caminhos q1 ou q3
q1	Ø	{q2 }	Leu 0, espera 1
q2	{q5 }	Ø	Leu 01, espera 0 para aceitar
q3	{q4 }	Ø	Leu 1, espera 0
q4	Ø	{q5 }	Leu 10, espera 1 para aceitar
q5*	{q5 }	{q5 }	Padrão encontrado! Aceita o restante da cadeia

iii. Sétupla Formal

$$M = (\{q0, q1, q2, q3, q4, q5\}, \{0, 1\}, \{0, 1, ⊔\}, \delta, q0, \{q5\}, ⊔)$$

$$M = (Q, \Sigma, \Gamma, \delta, q0, F, ⊔)$$

Como ler o processamento da senha 0010?

Imagine que o sistema processa a senha bit a bit:

1. **Início:** A máquina está em {q0}.
2. **Lê 0:** Ela se divide. Agora está em {q0, q1}. (Uma cópia continua procurando, outra começou o caminho do 010).
3. **Lê 0:**
   - De q0 com 0 vai para {q0, q1}.
   - De q1 com 0 vai para Ø (esse caminho "morre").
   - **Estado atual:** {q0, q1}.
4. **Lê 1:**
   - De q0 com 1 vai para {q0, q3}.
   - De q1 com 1 vai para {q2}.
   - **Estado atual:** {q0, q2, q3}.

5. **Lê 0:**
    - De q0 com 0 vai para {q0, q1}.
    - De q2 com 0 vai para {q5}.
    - De q3 com 0 vai para {q4}.
    - **Estado atual:** {q0, q1, q4, q5}.
6. **Conclusão:** Como o estado final **q5** está no conjunto de estados atuais, a senha é **ACEITA**.

Problema usando AFN: Sincronização de Dados

Cenário: Em redes de computadores, os dados viajam como um fluxo contínuo de bits. Para saber onde começa uma informação importante, usamos um "preâmbulo" ou "cabeçalho".

Objetivo: Projetar um autômato que aceite apenas fluxos de dados que contenham a sequência de sincronia **1010**.

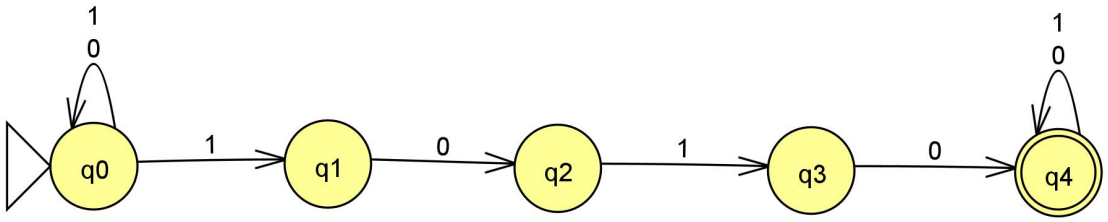


Tabela de Transição (δ)

Estado Atual	Entrada '0'	Entrada '1'	Comentário Didático
→q_0	q_0	q_0 , q_1	Pode ignorar o '1' ou começar a validar o frame.
q_1	q_2	Ø	Esperava '0', se vier '1' este caminho morre.
q_2	Ø	q_3	Esperava '1'.
q_3	q_4	Ø	Esperava '0'.
Q_0*	q_4	q_4	Frame detectado. Aceita o restante dos dados.

Sétupla Formal

$$M = (\{q_0, q_1, q_2, q_3, q_4\}, \{0, 1\}, \{0, 1, ⊔\}, \delta, q_0, \{q_4\}, ⊔)$$

M = (Q, Σ, Γ, δ, q0, F, ⊔)

Tabela Símbolos (teoria da computação)

Símbolo	Nome Técnico	Definição e Função no Autômato
Σ	Alfabeto de Entrada	Conjunto finito de símbolos que compõem as cadeias a serem lidas (ex: {0,1}).
Γ	Alfabeto da Fita	Conjunto de todos os símbolos que podem aparecer na fita. $\Sigma \subset \Gamma$, pois inclui símbolos de controle como o branco.
Q	Conjunto de Estados	Espaço de estados finito da máquina (ex: {$q0$,$q1$,...,qn }).
δ	Função de Transição	O "motor" lógico: $\delta: Q \times \Sigma \to Q$ (AFD) ou $\delta: Q \times \Sigma \to P(Q)$ (AFN).
q_0	Estado Inicial	O estado onde a computação obrigatoriamente começa ($q0$ $\in Q$).
F	Estados de Aceitação	Subconjunto de estados ($F \subseteq Q$) que validam a cadeia ao final da leitura.
$\sqcup$	Símbolo Branco	Representa o espaço vazio na fita de memória. Essencial para delimitar o fim da entrada.
ϵ	Épsilon	Cadeia vazia ou transição nula. Permite salto de estado sem consumir entrada (AFN).
$P(Q)$	Conjunto das Partes	Representa todas as combinações possíveis de estados. No AFN, indica que o destino é um subconjunto.
M	Máquina (Modelo)	A tupla completa que define o sistema: $M=(Q,\Sigma,\Gamma,\delta,q0$,$F,\sqcup)$.
Σ^*	Fecho de Kleene	O conjunto infinito de todas as palavras possíveis sobre Σ, incluindo ϵ.
$\Sigma+$	Fecho Positivo	O conjunto de todas as palavras possíveis sobre Σ, excluindo a cadeia vazia ϵ.
2^Q	Notação de Potência	Uma forma alternativa de escrever $P(Q)$, indicando o número de subconjuntos (2^n).
$\emptyset$	Conjunto Vazio	Representa a ausência de transição ou uma linguagem que não aceita nenhuma cadeia.

No **AFD/AFN** convencional, muitas vezes trabalhamos apenas com a **Quíntupla** ($Q, \Sigma, \delta, q0, F$), no entanto, em nossos exercícios, exploramos a **Sétupla** ($Q, \Sigma, \Gamma, \delta, q0, F, \sqcup$) para ter um rigo acadêmico maior.

Tabela Símbolos de Relação e Operação de Conjuntos

Símbolo	Nome Técnico	Aplicação na Teoria da Computação
$\in$	Pertence	Indica que um elemento faz parte de um conjunto. Ex: $q_0 \in Q$ (O estado inicial pertence ao conjunto de estados).
$\notin$	Não Pertence	Indica que um elemento está fora do conjunto. Ex: $\epsilon \notin \Sigma$ (A cadeia vazia não faz parte do alfabeto de entrada).
$\subseteq$	Subconjunto ou Igual	Indica que um conjunto está contido em outro. Ex: $F \subseteq Q$ (O conjunto de estados finais é uma parte do conjunto total de estados).
$\subset$	Subconjunto Próprio	Indica que um conjunto está contido, mas não é idêntico ao outro. Ex: $\Sigma \subset \Gamma$ (O alfabeto de entrada é estritamente menor que o da fita).
$\cup$	União	Une as possibilidades de dois conjuntos. Usado no AFN: $\delta(q_0,a)=\{q_0\} \cup \{q_1\}$.
$\cap$	Interseção	Elementos comuns a dois conjuntos. Usado para verificar se uma computação terminou em aceitação: $Q_{atuais} \cap F \neq \emptyset$.
P(Q)	Conjunto das Partes	O conjunto de todos os subconjuntos de Q. No AFN, a transição aponta para um elemento de P(Q).
$\times$	Produto Cartesiano	Combina conjuntos. Na função δ, indica que a entrada é um par (Estado $\times$ Símbolo).
$\setminus$	Diferença	Elementos que estão em um conjunto mas não no outro. Ex: Estados que não são de aceitação ($Q \setminus F$).
$\emptyset$	Conjunto Vazio	Representa uma transição impossível no AFN ou uma linguagem que não reconhece nenhuma palavra.

3.5. Notação para modelagem de autômatos (Convenção JFLAP)

Para modelar autômatos de forma que sejam compreendidos por simuladores e outros acadêmicos, utilizamos elementos gráficos padronizados. No JFLAP, a construção segue estas quatro convenções principais:

I. Estados (Q)

Os estados são representados por **círculos**. No JFLAP, eles são nomeados automaticamente como q_0, q_1, q_2, ... conforme são criados.

- **Estado Inicial:** Identificado por uma **seta (triângulo)** apontando para o círculo. Existe apenas um estado inicial por autômato.

- **Estado Final (ou de Aceitação):** Identificado por um **círculo duplo** (um círculo dentro do outro). Pode haver múltiplos estados finais.

II. Transições (δ)

As transições são as **setas** que conectam os estados. Elas indicam a mudança de um estado para outro após a leitura de um símbolo.

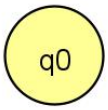
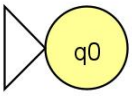
- **Rótulo da Transição:** O símbolo do alfabeto (Σ) que causa a mudança é escrito sobre a seta.
- **Auto transição (Self-loop):** Uma seta que sai e volta para o mesmo estado, indicando que o símbolo lido não altera o estado atual.
- **Transição de Saída:** Cada estado pode ter várias setas saindo dele (no caso de AFN) ou exatamente uma para cada símbolo do alfabeto (no caso de AFD).

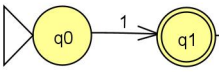
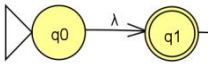
III. Transições de Cadeia Vazia (λ ou ϵ)

Embora a literatura utilize frequentemente o **ϵ** (épsilon), o JFLAP utiliza por padrão o símbolo **λ (lambda)** para representar transições vazias.

- Essas transições permitem que o autômato mude de estado sem consumir nenhum caractere da fita de entrada.
- No JFLAP, para criar uma transição vazia, basta deixar o campo do rótulo em branco ao criar a seta.

IV. Tabela de Símbolos do JFLAP

Elemento Visual	Significado	Como criar no JFLAP
 Círculo Simples	Estado comum	Ferramenta State Creator.
 Seta Entrante	Início do processamento	Botão direito no estado > Initial.
 Círculo Duplo	Aceitação da cadeia	Botão direito no estado > Final.

 Seta com Rótulo	Regra de transição	Ferramenta Transition Creator.
 Rótulo Vazio	Salto ϵ (λ)	Clique na seta e não digite nada.

Exercício de Fixação

Exemplo 1 - Desenhe um autômato com o alfabeto {0,1} e aceite apenas cadeias que terminem em 1.

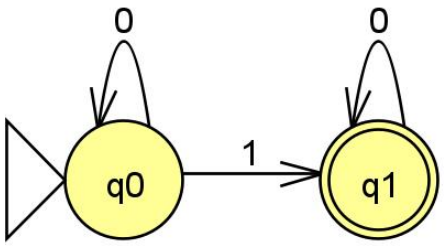


Tabela de Transição (δ)

Estado Atual	Entrada '0'	Entrada '1'
→q0	q0	q1
*q1	q0	q1

Exemplo 2 - Desenhe um autômato com o alfabeto {a,b,c,d} e aceite apenas cadeias que terminem em a.

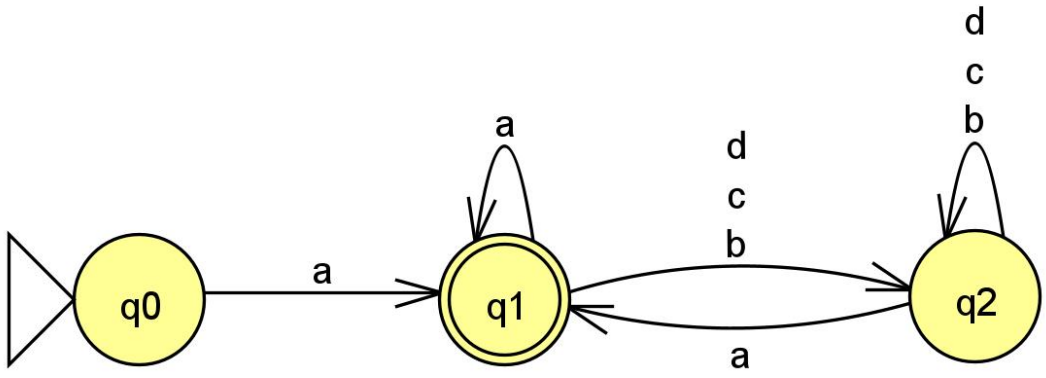


Tabela de Transição (δ)

Estado Atual	Entrada 'a'	Entrada 'b'	Entrada 'c'	Entrada 'd'
→q0	q1			
*q1	q1	q2	q2	q2
q2	q31	q2	q2	q2

Exemplo 3 - Desenhe um autômato com o alfabeto {a,b,0,1} e aceite apenas cadeias que terminem em a1.

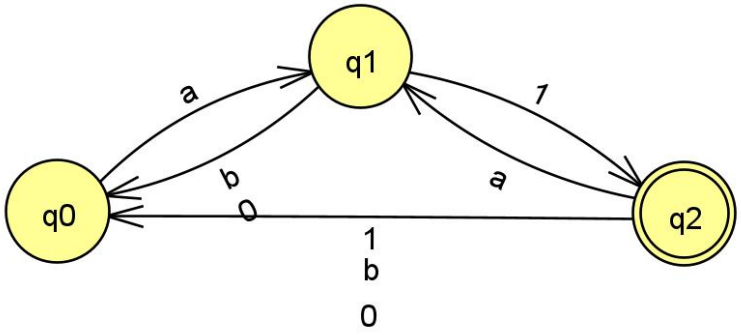


Tabela de Transição (δ)

Estado Atual	Entrada 'a'	Entrada '1'	Entrada 'b' / '0'
→q0	q1	q0	q0
q1	q1	q2	q0
*q2	q1	q0	q0

Exemplo de transformação de AFN em AFD

Vamos modelar um AFN que reconhece cadeias binárias ($\Sigma = \{0, 1\}$) que **contêm a sequência "01"**.

O AFN é muito mais simples de desenhar, pois ele "tenta" encontrar o 01 em qualquer ponto da cadeia. Já o AFD precisa de regras rígidas para cada bit lido.

1. O Modelo AFN (Não-Determinístico)

Neste modelo, o estado q_0 tem um laço para ignorar qualquer bit até que ele "decida" que o 0 atual é o início da sequência 01.

Tabela de Transição AFN:

Estado	Entrada '0'	Entrada '1'
→q_0	$\{q_0, q_1\}$	$\{q_0\}$
q_1	Ø	$\{q_2\}$
*q_2	$\{q_2\}$	$\{q_2\}$

2. Transformação para AFD (Construção de Subconjuntos)

Para converter, analisamos para quais **conjuntos de estados** a máquina vai. Chamaremos os novos estados do AFD pelas letras A, B e C.

Estado A $\{q_0\}$ (Inicial):

Estado B $\{q_0, q_1\}$:

Estado C $\{q_0, q_2\}$ (Final, pois contém q_2):

Estado (Subconjunto)	Entrada '0'	Entrada '1'
→A:$\{q0\}$	B	A
B:$\{q0, q_1\}$	B	C
*C:$\{q_0, q_2\}$	C	C

I. Questões de Múltipla Escolha

1. Sobre a natureza do não-determinismo em um AFN, assinale a alternativa correta:

a) Um AFN é mais potente que um AFD, pois consegue reconhecer linguagens não-regulares.

b) O não-determinismo permite que, a partir de um estado e um símbolo, o autômato alcance um conjunto de estados possíveis.

c) Um AFN não pode ser convertido em um AFD devido à sua complexidade.

d) O não-determinismo é uma característica física do hardware onde o autômato é executado.

2. O que representa o símbolo ϵ (épsilon) em um diagrama de transição de um AFN?

a) Um erro fatal que interrompe a execução.

b) O final obrigatório da cadeia de entrada.

c) Uma transição que ocorre sem o consumo de qualquer símbolo da fita de entrada.

d) O alfabeto completo da linguagem.

3. Considere a função de transição $\delta: Q \times \Sigma \rightarrow P(Q)$. O termo $P(Q)$ indica que:

a) O resultado da transição é sempre um único estado final.

b) O resultado é um elemento do conjunto das partes de Q, ou seja, um subconjunto de estados.

c) O autômato possui infinitos estados.

d) A transição depende de uma pilha de memória extra.

4. No processo de conversão de AFN para AFD (Construção de Subconjuntos), se um AFN possui n estados, qual o número máximo de estados que o AFD resultante pode ter?

a) $n + 1$

b) n^2

c) 2^n

d) $n!$ (n fatorial)

5. Uma cadeia é considerada "aceita" por um AFN se:

a) Pelo menos um dos caminhos possíveis terminar em um estado de aceitação.

b) Todos os caminhos possíveis terminarem em estados de aceitação.

c) O autômato consumir a cadeia em tempo recorde.

d) A cadeia não possuir o símbolo ϵ.

6. Qual a utilidade prática das transições- ϵ na modelagem de sistemas?

a) Aumentar a velocidade de processamento do computador.

b) Substituir a necessidade de estados iniciais.

c) Impedir que a máquina entre em loops infinitos.

d) Facilitar a união de dois ou mais autômatos em um único sistema.

7. Sobre a equivalência entre AFDs e AFNs, é correto afirmar que:

a) Todo AFD é um AFN, mas nem todo AFN possui um AFD equivalente.

b) O AFN é usado em compiladores por ser mais rápido que o AFD.

c) Ambos reconhecem exatamente a mesma classe de linguagens: as Linguagens Regulares.

d) Linguagens Livres de Contexto são reconhecidas por AFNs.

8. O que acontece em um AFN se, para um determinado símbolo, não houver transição definida ($\delta(q, s) = \emptyset$)?

a) Aquele caminho específico da computação é encerrado ("morre").

b) O autômato aceita a cadeia automaticamente.

c) A máquina reinicia do estado q_0.

d) O símbolo é ignorado e a fita avança.

9. Ao converter um AFN para um AFD, quais estados do novo AFD serão considerados "de aceitação" (F)?

a) Apenas o estado que for idêntico ao estado final do AFN.

b) Apenas o último estado gerado no processo de conversão.

c) Todos os estados, exceto o estado inicial.

d) Todos os novos estados (subconjuntos) que contiverem ao menos um estado final do AFN original.

10. O " Fecho-Vazio" (ε-closure) de um estado q é definido como:

a) O conjunto de todos os estados alcançáveis a partir de q seguindo apenas transições-ε.

b) O conjunto de símbolos que o estado q não consegue ler.

c) O estado final após a remoção de todos os zeros da fita.

d) A inversão de todas as setas do diagrama.

II. Perguntas Reflexivas

1. **Paralelismo Teórico:** Discuta a analogia de que um AFN funciona como um sistema de computação paralela massiva. Como a ideia de "múltiplas cópias" da máquina simplifica a vida do projetista de sistemas em comparação ao pensamento linear e restrito de um AFD?

2. **Custo de Conversão:** Vimos que um AFN com 10 estados pode gerar um AFD com até 1.024 estados (21^0). Reflita sobre o impacto disso no desenvolvimento de software: Por que muitas vezes preferimos modelar com AFN, mesmo sabendo que a execução final (pelo computador) poderá exigir muito mais memória após a conversão para AFD?

III. Exercícios Práticos

1. **Conversão de Modelo:** Dado um AFN com alfabeto $\Sigma = \{a, b\}$, estado inicial q_0 e as transições $\delta(q_0, a) = \{q_0, q_1\}$ e $\delta(q_0, b) = \{q_0\}$, converta-o manualmente para um AFD. Apresente a tabela de transição resultante com os estados nomeados como subconjuntos (ex: $\{q_0, q_1\}$).

2. **Modelagem AFN- ϵ:** Projete um AFN com transições- ϵ que aceite a união de duas linguagens: L1 (cadeias que começam com '0') e L2 (cadeias que terminam com '1'). Use um novo estado inicial q_{start} que se conecte às duas lógicas via ϵ.

Unidade 4: Expressões Regulares (ER) e Análise Léxica

4.1 Conceituação e Definição Formal

Uma **Expressão Regular** é uma forma compacta e puramente algébrica de definir uma **Linguagem Regular**. Enquanto **o Autômato Finito** é uma máquina que *reconhece* se uma string é válida, a Expressão Regular ***descreve*** o conjunto de todas as **strings válidas**.

Menezes (2011) define recursivamente uma ER sobre um alfabeto **Σ**:

1. **Ø** é uma **ER** (representa a linguagem vazia).
2. **ϵ** é uma **ER** (representa a linguagem contendo apenas a cadeia vazia).
3. Para cada **a ∈ Σ**, **a** é uma ER (representa a linguagem **{a}**).
4. Se **R_1** e **R_2** são **ERs**, então (R_1 + R_2), (R_1 · R_2) e R_1* também são.

1. Ø (ER que representa a Linguagem Vazia)

O conceito: Representa um conjunto que **não possui nenhuma cadeia**, nem mesmo a vazia. É uma máquina que rejeita tudo o que recebe.

- **Exemplo Real:** A interseção de duas linguagens que não têm nada em comum.
    - L_1 = {a} e L_2 = {b}. A linguagem comum entre elas é **Ø**.

2. ϵ (ER que representa a Linguagem {ϵ})

O conceito: Representa uma linguagem que contém exatamente **uma cadeia de comprimento zero**.

- **Exemplo Real:** Um campo de "Complemento" em um formulário de endereço que o usuário decide não preencher. A entrada foi enviada, mas está vazia.

3. a onde a ∈ Σ (ER que representa {a})

O conceito: É a forma mais simples de reconhecimento: a máquina espera exatamente um caractere específico do alfabeto.

- **Exemplo Real:** Se **Σ** = {0, 1}, a **ER 0** aceita apenas a cadeia "0" e rejeita "1", "00" ou vazio.

4. Operações Combinadas (Se R1 e R_2 são ERs)

A. União: $(R_1 + R_2)$ ou $(R_1 | R_2)$

O conceito: Representa a escolha ou alternativa. Aceita cadeias que pertencem a R_1 OU a R_2.

- Exemplo: Se $R_1 = a$ e $R_2 = b$, a ER $(a + b)$ aceita as cadeias {a, b}.
- Aplicação: Validar se um usuário digitou "M" ou "F" em um campo de gênero.

B. Concatenação: $(R_1 \cdot R_2)$

O conceito: Representa o sequenciamento. Aceita cadeias que começam com algo de R_1 seguido imediatamente por algo de R_2.

- Exemplo: Se R_1 = a e R_2 = b, a ER $(a \cdot b)$ aceita apenas a cadeia {ab}.
- Aplicação: Formar palavras reservadas em linguagens de programação, como if, for ou int.

Se R_1 = {a,b,c,d} e R_2={e,f,g,h}, quais seriam as combinações possíveis de R1 · R2?

Como R_1 possui 4 elementos e R_2 possui 4 elementos, o resultado terá exatamente 16 combinações (4 X 4).

Lista das Combinações Possíveis:

As combinações são formadas pelo par (u, v) onde $u \in R_1$ e $v \in R_2$:

1. Com 'a': {ae, af, ag, ah}
2. Com 'b': {be, bf, bg, bh}
3. Com 'c': {ce, cf, cg, ch}
4. Com 'd': {de, df, dg, dh}

Representação em Conjunto Final:

L = {ae, af, ag, ah, be, bf, bg, bh, ce, cf, cg, ch, de, df, dg, dh}

C. Fecho de Kleene: R_1*

O conceito: Representa repetição e opcionalidade. Aceita R_1 repetido zero ou mais vezes.

- Exemplo: Se $R_1 = a$, então a* aceita {ϵ, a, aa, aaa, ...}.

- **Aplicação:** Validar uma sequência de dígitos em um número inteiro, onde pode haver qualquer quantidade de algarismos.

Expressão Regular	Linguagem Gerada (L)	Descrição Narrativa
∅	∅	"Nada é aceito."
ϵ	{ϵ}	"Aceita apenas o vazio."
a	{a}	"Aceita exatamente a letra 'a'."
a+b	{a,b}	"Aceita 'a' ou 'b'."
a·b	{ab}	"Aceita 'a' seguido de 'b'."
a*	{ϵ,a,aa,...}	"Aceita qualquer quantidade de 'a', inclusive nenhuma."

4.2 Operadores e Precedência

Para resolver problemas sem ambiguidades, devemos seguir a ordem de precedência dos operadores, assim como na aritmética (onde a multiplicação vem antes da soma).

1. **Fechamento (Estrela de Kleene - *):** Possui a maior precedência. Representa zero ou mais ocorrências.
2. **Concatenação (. ou justaposição):** Possui precedência média.
3. **União (ou Alternância - + ou |):** Possui a menor precedência. Representa a escolha entre padrões.

Nível	Direção	Comportamento
Fluxo de Entrada	Esquerda → Direita	Os caracteres são apresentados ao motor nesta ordem.
Precedência de Operadores	Hierárquica	Parênteses **()** primeiro, depois Fechos *, depois Concatenação ·, e por fim União |.
Busca (Match)	Não-Linear	O motor pode "saltar" para trás para validar grupos de captura ou alternativas.

Exemplo Prático: A expressão $0 + 1 \cdot 0^*$

- Primeiro resolve-se 0^* (zero ou mais zeros).
- Depois a concatenação $1 \cdot 0^*$ (um '1' seguido de zero ou mais zeros).
- Por fim a união: ou o símbolo '0', ou o padrão anterior.

Conjunto Final: L = {0, 1, 10, 100, 1000, 10000, ...}

Exemplo Prático: A expressão $(0 + 1) \cdot 0^*$

- Primeiro resolve-se **(0 + 1)** (zero ou 1).
- Depois a concatenação **0*** (qualquer quantidade de zeros, inclusive nenhuma).

Caminho 1: Se você escolher o '0' no parêntese

- 0 (seguido de zero '0's)
- 00 (seguido de um '0')
- 000 (seguido de dois '0's)
- 0000 (seguido de três '0's)
- ... (infinitos zeros)

Caminho 2: Se você escolher o '1' no parêntese

- 1 (seguido de zero '0's)
- 10 (seguido de um '0')
- 100 (seguido de dois '0's)
- 1000 (seguido de três '0's)
- ... (infinitos zeros)

IMPORTANTE

Embora o fluxo de dados seja processado da esquerda para a direita, a lógica de decisão de uma Expressão Regular moderna é **não-linear**, permitindo que o algoritmo explore múltiplos caminhos e retorne a pontos anteriores da cadeia para garantir a validade da regra.

4.3 Tabela Completa de Operadores e Notações

Abaixo, apresento uma tabela com os operadores fundamentais (formais) e os metacaracteres estendidos utilizados na computação moderna (POSIX/PCRE), fundamentais para a análise léxica.

PCRE -> Perl Compatible Regular Expressions

POSIX -> Portable Operating System Interface

Os metacaracteres estendidos são o que permitem criar validações complexas com poucas linhas de código. Diferente dos símbolos básicos da Teoria da Computação (∪, ·, *), os estendidos focam em **praticidade e legibilidade**.

Categoria	Símbolo	Nome Técnico	Aplicação e Significado
Formal	Σ	Alfabeto	Conjunto finito de símbolos de entrada (ex: {0,1}).
Formal	Γ	Alfabeto da Fita	Inclui Σ + símbolos de controle como o branco (␣).
Formal	δ	Função de Transição	A regra de movimento: Q×Σ→Q (AFD) ou P(Q) (AFN).
Formal	ϵ	Épsilon	Cadeia vazia. Transição que não consome caracteres.
Formal	R*	Estrela de Kleene	Repetição de zero a infinitas vezes.
Formal	R+	Fecho Positivo	Repetição de uma a infinitas vezes (R·R*).
Formal	∪ ou +	União / Alternância	Escolha entre dois caminhos ou linguagens.
Formal	·	Concatenação	Sequenciamento de símbolos (geralmente implícito na computação).
Formal	P(Q)	Conjunto das Partes	Usado na conversão AFN → AFD para agrupar estados.
Computação	.	Ponto (Wildcard)	Casa com qualquer único caractere (exceto nova linha).
Computação	|	Pipe	O equivalente prático à união (∪). Ex: (cat|dog).
Computação	?	Opcional	O elemento anterior aparece 0 ou 1 vez.
Computação	^	Circunflexo	Âncora: força o casamento no início da string/linha.
Computação	$	Cifrão	Âncora: força o casamento no final da string/linha.
Computação	[ ]	Classe de Caracteres	Define um conjunto permitido. Ex: [a-z] ou [0-9].
Computação	[^]	Negação de Classe	Casa com qualquer caractere exceto os listados.
Computação	( )	Grupo de Captura	Agrupa elementos para prioridade ou extração posterior.
Computação	\d	Digit	Atalho para qualquer dígito numérico [0-9].
Computação	\w	Word	Atalho para alfanuméricos e underline [a-zA-Z0-9_].
Computação	\s	Space	Atalho para espaços, tabs e quebras de linha.

Computação	\b	Boundary	Fronteira de palavra (não consome caractere, apenas valida posição).
Computação	{n,m}	Quantificador	Define limites: {3} (exatos), {3,} (mínimo), {3,5} (faixa).
Computação	\	Escape	Anula o poder do metacaractere. Ex: \. busca um ponto literal.

DICA:

Utilize o site https://regex101.com para criar e testar expressões regulares.

O **PRCE** (originalmente PCRE1) evoluiu para o **PCRE2 em 2015** (padrão atual).

Característica	PCRE (Versão 1)	PCRE2 (Versão Atual)
Lançamento	1997 (Legado)	2015 (Padrão atual)
Status de Suporte	Apenas correções de bugs críticos.	Desenvolvimento ativo e melhorias.
Performance	Rápida, mas limitada em strings longas.	JIT Compilation (Just-In-Time) muito superior.
Segurança	API de C mais simples, mas menos protegida.	Melhor proteção contra transbordamento de pilha.
Suporte Unicode	Requer configurações extras (UTF-8).	Suporte nativo e robusto (UTF-8, 16 e 32).
API de C	Funções como pcre_compile().	Funções renomeadas como pcre2_compile().

4.4 Exemplos de uso de ER

a. Validação de Nome de Usuário (Username)

Cenário: O usuário deve criar um login que comece com uma letra, tenha entre 3 e 16 caracteres e aceite apenas letras, números e *underline*.

- **RE:** ^[a-zA-Z]\w{2,15}
- **Símbolos Utilizados:**
  - ^ e $: **Âncoras** de início e fim (garantem que a string inteira siga a regra).
  - [a-zA-Z]: **Classe de Caracteres** (o primeiro caractere deve ser letra).
  - \w: **Atalho** para alfanumérico (letras, números e _).

- {2,15}: **Quantificador** (como já temos o 1º caractere, faltam de 2 a 15 para completar o intervalo de 3 a 16).

b. Endereço IPv4

Cenário: Validar um endereço IP simples (formato x.x.x.x). *Nota: Esta versão simplificada foca na estrutura de dígitos e pontos.*

- **RE:** ^\d{1,3}(\.\d{1,3}){3}$
- **Símbolos Utilizados:**
  - \d{1,3}: **Atalho + Quantificador** (1 a 3 dígitos).
  - \.: **Escape** (o ponto . sozinho é um metacaractere coringa, a barra \ o torna um ponto literal).
  - ( ): **Grupo de Captura** (agrupa o conjunto "ponto + números").
  - {3}: **Repetição** (o grupo se repete exatamente 3 vezes após o primeiro bloco).

c. Valor Monetário (Real Brasileiro)

Cenário: Reconhecer valores como R$ 1.250,00 ou 150,50. Útil para o seu sistema ISY.ONE.

- **RE:** ^(R\$\s?)?\d+(\.\d{3})*,\d{2}$
- **Símbolos Utilizados:**
  - ?: **Opcional** (O "R$" e o espaço \s podem ou não existir).
  - \d+: **Fecho Positivo** (Pelo menos um dígito antes da vírgula).
  - (\.\d{3})*: **Fecho de Kleene** (Pode haver múltiplos blocos de milhar com ponto, ou nenhum).
  - ,\d{2}: **Concatenação** fixa para os centavos.

d. Extração de Datas (Formato ISO)

Cenário: Buscar em um arquivo de log datas no formato AAAA-MM-DD.

- **RE:** \b\d{4}-\d{2}-\d{2}\b
- **Símbolos Utilizados:**

- \b: **Boundary** (Garante que a data não seja parte de um número maior, como um ID).
- -: Caractere literal de **Concatenação**.
- \d{n}: Precisão numérica para anos, meses e dias.

e. Senha "Forte" (Requisito de Complexidade)

Cenário: A senha deve ter no mínimo 8 caracteres, incluindo pelo menos uma letra maiúscula, uma minúscula e um número.

- **RE:** ^(?=.*[a-z])(?=.*[A-Z])(?=.*\d).{8,}$
- **Símbolos Utilizados:**
  - (?=...): **Lookahead Positivo** (Verifica se algo existe à frente sem "consumir" a string).
  - .*: **Coringa + Fecho de Kleene** (Qualquer caractere em qualquer quantidade).
  - .{8,}: **Mínimo de caracteres** (Garante o comprimento total após as verificações de lookahead).

f. Modelagem de estruturas de decisão do Python (if, elif, else)

Cenário: Modelar as instruções de decisão do Python (como o if, elif e else) usando Expressões Regulares. Considere a sintaxe específica da linguagem: a palavra-chave, os espaços e o caractere obrigatório de dois pontos ":".

A Estrutura da Regra (Python Decision)

Uma instrução de decisão básica em Python segue este padrão:

1. Inicia com if, elif ou else.
2. Pode ter espaços em branco ao redor.
3. No caso de if/elif, existe uma **condição** (comparação, variável, etc.).
4. Termina obrigatoriamente com :.

- **RE:** ^\s*(if|elif)\s+.+:\s*$|^\s*else\s*:\s*$

- Símbolos Utilizados:
  - ^ e $: Âncoras que garantem que a linha inteira seja a instrução de decisão (evita pegar um if dentro de um comentário ou string).
  - \s*: Atalho + Estrela de Kleene. Aceita qualquer nível de indentação (espaços ou tabs no início).
  - (if|elif): União / Alternância. Define que o token inicial deve ser um desses dois.
  - \s+: Fecho Positivo. Obriga a existência de pelo menos um espaço após o if/elif (ex: if x > 0:).
  - .+: Coringa + Fecho Positivo. Representa a condição (qualquer caractere, pelo menos um).
  - :: Caractere Literal. O terminador sintático do Python.
  - |: Operador de União. Separa a lógica do if/elif da lógica do else (que não recebe condição).
  - else\s*:: O else permite espaços antes dos dois pontos, mas nunca uma condição.

4.5 Equivalência: ER ó Autômatos

Sipser (2007) prova que para toda ER existe um AFN-ϵ equivalente (pelo método de Thompson) e, consequentemente, um AFD.

- ER para AFN: Fácil de construir manualmente (regras de colagem).
- AFN para AFD: Algoritmo de subconjuntos (visto na Unidade 3).
- AFD para ER: Método de eliminação de estados.

Esta equivalência garante que podemos escrever uma regra de validação complexa em uma única linha de texto (ER) e o computador saberá transformá-la em uma máquina de estados ultraveloz para processar milhões de registros por segundo.

I. Perguntas de Múltipla Escolha

1. Qual a função do operador "Estrela de Kleene" (*)?

a) Obrigar a presença de pelo menos um símbolo.

b) Permitir zero ou mais repetições do padrão anterior.

c) Negar todos os símbolos do alfabeto.

d) Unir dois alfabetos distintos.

2. Na análise léxica, o que é um "Token"?

a) Um erro de sintaxe grave.

b) Uma unidade básica com significado léxico (ex: um número ou operador).

c) O nome do arquivo fonte.

d) O resultado final da execução do programa.

3. Qual a ordem correta de precedência (da maior para a menor)?

a) União, Concatenação, Estrela.

b) Concatenação, União, Estrela.

c) Estrela, Concatenação, União.

d) União, Estrela, Concatenação.

4. A expressão regular (a|b)*abb aceita qual tipo de linguagem?

a) Apenas a string "abb".

b) Strings que começam com "abb".

c) Strings sobre {a, b} que terminam obrigatoriamente em "abb".

d) Qualquer string que não contenha "c".

5. O que o metacaractere ? representa na computação?

a) Uma pergunta ao usuário.

b) Que o elemento precedente é opcional (0 ou 1 ocorrência).

c) Repetição infinita.

d) Início de um comentário.

6. De acordo com Menezes (2011), as ERs descrevem qual classe de linguagens?

a) Linguagens Livres de Contexto.

b) Linguagens Regulares.

c) Linguagens Naturais.

d) Linguagens de Baixo Nível apenas.

7. Qual ER representa o conjunto de todas as strings binárias que começam com '1'?

a) 0*1

b) 1(0+1)*

c) (0|1)*1

d) 1*

8. O analisador léxico é responsável por qual tarefa?

a) Verificar se a lógica do programa está correta.

b) Agrupar caracteres em tokens e descartar espaços/comentários.

c) Traduzir o código para linguagem de máquina.

d) Alocar memória para as variáveis.

9. A ER [0-9]+(\.[0-9]+)? é comumente usada para reconhecer:

a) Datas.

b) Números decimais (inteiros ou com ponto).

c) Endereços de e-mail.

d) Nomes de variáveis em C.

10. O Teorema de Kleene afirma que:

a) Todo autômato finito é impossível de converter para ER.

b) ERs e Autômatos Finitos são equivalentes em poder de expressão.

c) Linguagens Regulares não podem ter repetições.

d) A cadeia vazia não pode ser descrita por ER.

11. Qual a diferença entre a* e a+?

a) Nenhuma, são sinônimos.

b) a* aceita a cadeia vazia; a+ exige pelo menos um 'a'.

c) a+ é usado apenas em alfabetos numéricos.

d) a* é mais rápido para o processador.

12. **O que acontece se uma string não for reconhecida por nenhuma ER no analisador léxico?**

a) O compilador ignora e segue adiante.

b) É gerado um erro léxico (caractere inválido).

c) O programa entra em loop infinito.

d) A string é convertida para binário automaticamente.

II. Perguntas Reflexivas

1. **Eficiência:** Por que é mais eficiente usar Expressões Regulares na fase léxica do que usar uma Gramática Livre de Contexto (mais complexa) para tudo?

2. **Segurança:** Como falhas na escrita de Expressões Regulares de validação em formulários web podem permitir ataques de Injeção de SQL ou XSS?

III. Exercícios Práticos

1. **Criação de ER:** Escreva uma Expressão Regular que reconheça identificadores que devem começar com uma letra (maiúscula ou minúscula), seguidos por qualquer número de letras ou dígitos, mas que devem ter no mínimo 3 caracteres.

2. **Conversão Visual:** Tente desenhar um AFN simples que represente a expressão (0+1)*11.

3. **Análise de Padrão:** Dada a ER [a-z]+@[a-z]+\.com, identifique quais das seguintes strings são aceitas e por que:
    - contato@fmu.com
    - admin123@google.com
    - suporte@empresa.org

Unidade 5: Gramáticas Regulares e a Hierarquia de Chomsky

5.1 O conceito de Gramática Formal

Enquanto os autômatos são **reconhecedores** (leem uma string e dizem "sim" ou "não"), as gramáticas são **geradores**. Segundo **Menezes (2011)**, uma gramática é um sistema de regras de substituição que permite criar todas as cadeias válidas de uma linguagem.

Formalmente, uma gramática G é uma **quádrupla:**

$$(V, \Sigma, P, S)$$

- **V (Variáveis/Não-terminais):** Símbolos auxiliares (geralmente letras maiúsculas) que aparecem durante o processo de geração, mas não na string final.
- **Σ (Terminais):** O alfabeto real da linguagem (letras minúsculas, números, etc.).
- **P (Regras de Produção):** O coração da gramática. Regras do tipo $A \rightarrow \alpha$, que indicam que a variável A pode ser substituída pela sequência alpha.
- **S (Símbolo Inicial):** A variável por onde toda geração começa ($S \in V$).

5.1.1 Variáveis (ou não-terminais) — O "Esqueleto"

As variáveis são símbolos intermediários. Elas **não aparecem no resultado final** (na string que o usuário lê), mas servem para dar estrutura e organização às regras.

- **Representação:** Geralmente escritas com letras **maiúsculas** (A, B, S).
- **Função:** Elas são "promessas" de conteúdo. Elas dizem: "Aqui deve entrar alguma coisa que ainda será definida pelas regras de produção".
- **Analogia:** Pense nas categorias gramaticais: <Substantivo>, <Verbo>, <Adjetivo>. Você não diz a palavra "Substantivo" em uma frase, você a substitui por uma palavra real (como "Carro").

5.1.2. Terminais — O "Produto Final"

Os terminais são os símbolos que compõem as cadeias (palavras) da linguagem. Eles são o ponto final da derivação: uma vez que você chega a um terminal, não pode mais substituí-lo por nada.

- **Representação:** Geralmente escritos com letras **minúsculas**, números ou símbolos especiais (a, b, 0, 1, +, *).

- **Função:** Formar a string final que será “processada” pelo computador ou “lida” pelo usuário.
- **Analogia:** São as palavras reais da nossa língua: "casa", "correr", "azul".

Exemplo Prático: Gerando uma Operação Matemática

Imagine uma gramática para **somar números**:

1. <Expressão> → <Número> + <Número>
2. <Número> → 0 | 1 | 2

- **Variáveis (não-terminais):** { <Expressão>, <Número> } — **São os moldes**.
- **Terminais:** { 0, 1, 2, + } — São os valores e o operador final.

O processo de derivação:

⟨Expressão⟩ ⇒ ⟨ Número⟩ + ⟨Número⟩ ⇒ 1 + 2

No final, as variáveis sumiram e restaram apenas os **terminais** (1+2).

Exemplo 1: Gramática de uma Linguagem "Pai e Filho" ($a^n b^n$)

Este é um exemplo clássico que um AFD simples **não consegue resolver** (pois exige contagem), mas uma gramática resolve facilmente.

- **V (Variáveis/Não-terminais):** {S}
- **Σ (Terminais):** {a, b}
- **P (Regras de Produção):**
    1. S → aSb
    2. S → ab
- **S (Símbolo Inicial):** S

OBS

** Para a linguagem $a^n b^n$ (onde a quantidade de 'a' deve ser igual à de 'b'), essas duas regras trabalham juntas para garantir o **equilíbrio** e a **parada** do processo.

A Regra Recursiva: S → aSb

Esta regra é responsável pelo crescimento da cadeia. Ela deve ser lida da seguinte forma:

"Para cada 'a' que eu coloco na esquerda, sou obrigado a colocar um 'b' na direita, mantendo o molde S no meio para continuar o processo."

- **O "truque" da contagem:** Como o AFD (Autômato Finito) não tem memória, ele não consegue "contar" quantos 'a's leu para depois exigir a mesma quantidade de 'b's. A gramática resolve isso de forma visual: ela expande de dentro para fora.
- **Recursividade:** O S aparece do lado direito da seta, o que permite aplicar essa regra repetidas vezes.

A Regra de Parada: S → ab

Esta regra é o ponto final da produção. Ela deve ser lida como:

"Substitua o molde S pelo par final 'ab' e encerre a geração."

- **Condição de Saída:** Sem esta regra, você ficaria preso na regra 1 para sempre, gerando uma cadeia infinita que nunca termina em símbolos terminais (letras minúsculas).
- **Base da Linguagem:** Ela também define a menor cadeia possível que essa gramática pode gerar, que é exatamente ab (n=1).

Exemplo de cadeia válida: aaabbb

Exemplo de cadeia inválida: aaaabbb

Exemplo 2: Gramática para Identificadores de Variáveis (Sintaxe Simplificada)

Imagine que no seu projeto, você queira definir como um nome de variável pode ser escrito.

- **V:** {S, L, D} (S = Sentença, L = Letra, D = Dígito)
- **Σ:** {x, y, 0, 1}
- **P:**
    1. S → L | LS | LD (Começa com letra, pode seguir com mais letras ou pode seguir com mais dígitos)
    2. L → x | y
    3. D → 0 | 1
- **S:** S

Exemplo de Derivação para x0:

$$S \Rightarrow LD \Rightarrow xD \Rightarrow x0$$

5.2 Gramáticas Regulares (Tipo 3)

As Gramáticas Regulares são o tipo mais restrito de gramática e geram exatamente as mesmas linguagens que os AFDs e as ERs. **Sipser (2007)** as classifica em dois subtipos:

1. **Gramática Linear à Direita:** As regras devem seguir o padrão:
    - A → aB (Um terminal seguido de um não-terminal)
    - A → a (Apenas um terminal)
    - A → ϵ (Cadeia vazia)

2. **Gramática Linear à Esquerda:** O padrão é o inverso (A → Ba), mas nunca misturamos as duas na mesma gramática.

Exemplo Prático: Identificadores que começam com 'a'

S → aA

A → aA | bA | a | b | ϵ

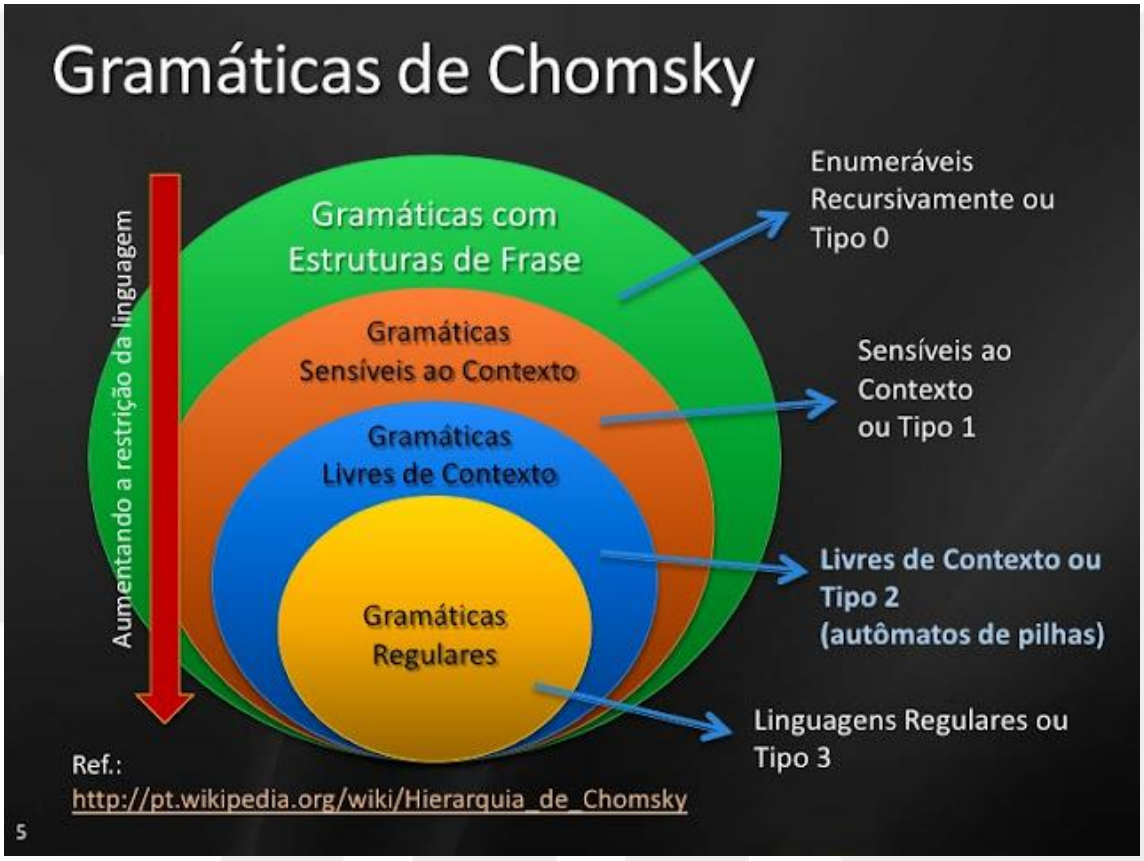
Aqui, o S obriga o primeiro caractere a ser 'a'. Depois, a variável A permite qualquer combinação de 'a' ou 'b'.

5.3 A Hierarquia de Chomsky

No final da década de 50, Noam Chomsky propôs uma classificação que organiza as linguagens em níveis de complexidade. **Diverio (2011)** ressalta que cada nível superior contém todos os níveis inferiores (inclusão).

Tipo	Linguagem	Reconhecedor	Exemplo Prático
Tipo 3	**Regular**	Autômato Finito	Senhas, CPFs, Tokens de código.
Tipo 2	**Livre de Contexto**	Autômato de Pilha	Estrutura de linguagens (Java, C, JSON).

Tipo 1	Sensível ao Contexto	Autômato Linearmente Limitado	Verificação de tipos e variáveis declaradas.
Tipo 0	Recursivamente Enumerável	Máquina de Turing	Qualquer algoritmo computável.



Fonte: https://radiomaffei.blogspot.com/2012/12/esse-cara-vale-pena-ser-lido.html

5.4 Curiosidade: O Teorema do Bombeamento (Pumping Lemma)

Um erro comum é achar que tudo pode ser resolvido com Autômatos ou Expressões Regulares. O **Teorema do Bombeamento**, detalhado por **Sipser (2007)**, é a ferramenta matemática usada para provar que uma linguagem **NÃO é regular**.

A lógica é simples: Se uma linguagem é regular e infinita, ela deve ter um ciclo no seu autômato. Se ela tem um ciclo, eu posso repetir esse ciclo quantas vezes eu quiser e a palavra deve continuar pertencendo à linguagem.

Exemplo Prático de Falha: $L = \{a^n b^n \mid n \geq 0\}$ (Ex: "ab", "aabb", "aaabbb").

Para reconhecer isso, a máquina precisaria "contar" quantos 'a's leu para garantir que lerá a mesma quantidade de 'b's. Como o Autômato Finito tem memória finita, ele **não consegue contar**. Logo, via Pumping Lemma, provamos que esta linguagem não é Tipo 3.

I. Perguntas de Múltipla Escolha

1. **Qual a função do Símbolo Inicial (S) em uma gramática?**

a) Indicar o fim da geração da string.

b) Ser o ponto de partida para todas as derivações da gramática.

c) Armazenar os símbolos terminais.

d) Definir o alfabeto da linguagem.

2. **Uma Gramática Linear à Direita caracteriza-se por:**

a) Ter não-terminais apenas do lado esquerdo da seta.

b) Ter o não-terminal sempre à direita do terminal na regra de produção.

c) Não permitir a cadeia vazia.

d) Ter regras que podem ser lidas de trás para frente.

3. **Na Hierarquia de Chomsky, qual o tipo das Linguagens Regulares?**

a) Tipo 0

b) Tipo 1

c) Tipo 2

d) Tipo 3

4. **O que o Teorema do Bombeamento (Pumping Lemma) permite provar?**

a) Que uma linguagem é regular.

b) Que uma linguagem não é regular.

c) Que um autômato tem muitos estados.

d) Que a Máquina de Turing é invencível.

5. Qual componente de uma gramática representa o "alfabeto final" da string gerada?

a) Variáveis (V)

b) Regras de Produção (P)

c) Terminais (Σ)

d) Símbolo Inicial (S)

6. De acordo com Menezes (2011), qual o reconhecedor das linguagens de Tipo 2?

a) Máquina de Turing.

b) Autômato Finito Determinístico.

c) Autômato de Pilha.

d) Analisador Léxico.

7. A regra de produção A → aB | b pertence a que tipo de gramática?

a) Regular (Linear à Direita).

b) Sensível ao Contexto.

c) Irrestrita.

d) Natural.

8. Por que a linguagem $\{a^n b^n\}$ não é regular?

a) Porque usa letras proibidas.

b) Porque exige memória infinita para contar a paridade entre 'a' e 'b'.

c) Porque o alfabeto é muito pequeno.

d) Porque Chomsky não a incluiu na lista.

9. O que é uma "derivação" em gramáticas?

a) O processo de converter um autômato em ER.

b) Um erro de lógica no código-fonte.

c) A exclusão de regras inúteis.

d) A sequência de substituições de variáveis até chegar a uma string de terminais.

10. **As linguagens de Tipo 0 são também chamadas de:**

a) Linguagens Regulares.

b) Linguagens de Programação de Alto Nível.

c) Linguagens Recursivamente Enumeráveis.

d) Linguagens Mortas.

11. **Qual a relação de inclusão correta na Hierarquia de Chomsky?**

a) Tipo 3 ⊂ Tipo 2 ⊂Tipo 1 ⊂Tipo 0.

b) Tipo 0 ⊂Tipo 1 ⊂Tipo 2 ⊂Tipo 3.

c) Tipo 2 ⊂Tipo 3 ⊂Tipo 0.

d) Não existe inclusão, são classes separadas.

12. **Se uma linguagem pode ser descrita por uma Expressão Regular, ela obrigatoriamente:**

a) É de Tipo 2.

b) Possui uma Gramática Regular equivalente.

c) Precisa de uma Máquina de Turing para ser lida.

d) Não possui estados finais.

II. Perguntas Reflexivas

1. **A Natureza da Linguagem:** Se as Linguagens Regulares (Tipo 3) são tão limitadas que não conseguem nem "contar" ou "balancear parênteses", por que elas ainda são a base de toda a análise léxica de compiladores modernos?

2. **Abstração vs. Realidade:** Como a Hierarquia de Chomsky ajuda um engenheiro de software a escolher a ferramenta certa para validar um dado (ex: decidir entre usar um Simples if/Regex ou um Parser completo)?

III. Exercícios Práticos

1. **Construção de Gramática:** Escreva as regras de produção para uma Gramática Regular que gere a linguagem L = { w ∈ {0, 1}* | w { possui um número ímpar de '1's}.

2. **Classificação Técnica**: Dada a gramática abaixo, identifique seu tipo na Hierarquia de Chomsky e justifique: $S \rightarrow aSb \mid \epsilon$

3. **Aplicação do Pumping Lemma:** Explique, com suas palavras, por que um autômato finito falharia ao tentar reconhecer se uma string de código tem todos os parênteses abertos e fechados corretamente.

Unidade 6: Linguagens Livres de Contexto (LLC) e Gramáticas Livres de Contexto (GLC)

6.1 A Necessidade de ir além do Regular

Como vimos na unidade anterior através do Teorema do Bombeamento, os Autômatos Finitos e as Expressões Regulares falham em tarefas simples como **contar** ou **balancear**.

Menezes (2011) explica que a estrutura de uma linguagem de programação como Java ou C exige "aninhamento" (um if dentro de outro if, parênteses dentro de parênteses). Para modelar essa memória de curto prazo necessária para saber se todos os parênteses abertos foram fechados, precisamos das **Linguagens Livres de Contexto (Tipo 2)**.

6.2 Definição Formal de GLC

Uma **Gramática Livre de Contexto (GLC)**, segundo **Sipser (2007)**, mantém a estrutura de 4-tupla **(V, Σ, P, S)**, mas as suas regras de produção (P) são muito mais flexíveis que as regulares. Em uma GLC, a regra é definida como:

$$V \rightarrow (V \cup \Sigma)^*$$

Isso significa que o lado esquerdo deve ser **sempre uma única variável**, mas o lado direito pode ser qualquer combinação de variáveis e terminais. O termo "Livre de Contexto" vem do fato de que a substituição da variável no lado esquerdo não depende dos símbolos ao seu redor.

Exemplo Prático: A Linguagem dos Parênteses Balanceados

$$S \rightarrow (S) \mid SS \mid \epsilon$$

Com esta gramática simples, podemos gerar:

- ()
- (())
- ()(())

Um Autômato Finito jamais conseguiria descrever esta linguagem para um número infinito de aninhamentos.

6.3 Derivações e Árvores Sintáticas (Parse Trees)

Para **Diverio (2011)**, a forma mais clara de visualizar como uma GLC gera uma cadeia é através da **Árvore Sintática**.

1. **Derivação à Esquerda (Leftmost):** Substitui-se sempre a variável mais à esquerda primeiro.
2. **Derivação à Direita (Rightmost):** Substitui-se a variável mais à direita primeiro.

Se uma gramática consegue gerar **duas árvores sintáticas diferentes** para a mesma cadeia, dizemos que a gramática é **Ambígua**.

Por que a ambiguidade é um problema?

Em compiladores, a ambiguidade pode levar o computador a calcular 2 + 3 * 4 de duas formas: **(2+3)*4 = 20** ou **2+(3*4) = 14**. Por isso, a elaboração de GLCs para linguagens de programação exige regras de precedência rígidas.

6.4 GLC na Construção de Linguagens de Programação

As linguagens que utilizamos no dia a dia são definidas quase inteiramente por GLCs, muitas vezes utilizando a notação **BNF (Backus-Naur Form)**. A **BNF (Backus-Naur Form)**, ou Forma de Backus-Naur, é uma **meta-linguagem** (uma linguagem usada para descrever outra linguagem) utilizada para definir formalmente a **sintaxe** de linguagens de programação, protocolos de rede e formatos de dados.

Em termos práticos, se uma Gramática Livre de Contexto (GLC) é o conceito matemático, a BNF é a **notação de engenharia** usada para escrever essa gramática no mundo real. Ela foi criada por John Backus e Peter Naur na década de 1960 para documentar a linguagem ALGOL 60.

A Sintaxe Básica da BNF

A notação baseia-se em regras de produção que utilizam um conjunto simples de metacaracteres:

- **< > (Chaves angulares):** Enclausuram os símbolos **não-terminais** (variáveis ou estruturas abstratas).
- **::= (Sinal de atribuição):** Significa "é definido como" ou "produz".
- **| (Barra vertical / Pipe):** Indica uma escolha/alternativa ("ou").
- **Texto puro (fora de < >):** Representa os símbolos **terminais** (caracteres literais, palavras-chave).

Exemplo de Elaboração: Estrutura de um IF

Em um compilador, a regra para um comando condicional poderia ser:

<comando_if> ::= if (<expressao>) { <bloco> } [else { <bloco> }]

Aqui, vemos a recursividade: o <bloco> pode conter outros <comando_if>. Essa capacidade recursiva é o que permite a criação de algoritmos complexos e é o que diferencia as LLC das linguagens regulares.

6.5 Estratégias para Resolução de Problemas

Ao avaliar um problema prático, como saber se devo usar GLC ou ER?

- **Use ER (Regular):** Se o problema for busca de padrões fixos, validação de campos simples (datas, telefones) ou separação de palavras (tokens).
- **Use GLC (Livre de Contexto):** Se o problema exigir paridade (ex: a^n b^n), aninhamento de estruturas, ou se a ordem dos elementos depender de um "abre e fecha".

I. Perguntas de Múltipla Escolha

1. **Qual a principal limitação das Linguagens Regulares que as LLC conseguem superar?**

a) A incapacidade de processar números binários.

b) A incapacidade de lidar com recursividade e aninhamento infinito.

c) A falta de suporte para letras maiúsculas.

d) A velocidade de processamento lenta.

2. **Na regra de produção A → α, o que define uma GLC?**

a) alpha deve ser obrigatoriamente um único terminal.

b) A deve ser uma única variável não-terminal.

c) A deve ser uma combinação de terminais e variáveis.

d) alpha não pode conter a cadeia vazia.

3. O que é uma gramática ambígua?

a) Uma gramática que aceita duas línguas diferentes ao mesmo tempo.

b) Uma gramática que gera mais de uma árvore sintática para a mesma string.

c) Uma gramática que não possui estado inicial.

d) Uma gramática escrita em mais de um idioma.

4. A notação BNF (Backus-Naur Form) é frequentemente utilizada para:

a) Desenhar autômatos finitos.

b) Definir a sintaxe de linguagens de programação.

c) Calcular a complexidade de algoritmos de ordenação.

d) Codificar imagens em binário.

5. Qual destas linguagens NÃO é regular, mas é Livre de Contexto?

a) $L = \{ a^n b^m \mid n, m \geq 0 \}$

b) $L = \{ w \mid w$ termina em $00\}$

c) $L = \{ a^n b^n \mid n \geq 0\}$

d) $L = \{ a, b, c \}$

6. Segundo Menezes (2011), uma árvore sintática mostra:

a) A hierarquia de diretórios de um computador.

b) A estrutura de derivação de uma cadeia a partir do símbolo inicial.

c) O caminho percorrido por um autômato finito.

d) A lista de todas as variáveis globais de um programa.

7. O que representa o termo "Livre de Contexto"?

a) Que a linguagem pode ser usada em qualquer sistema operativo.

b) Que as regras de substituição de uma variável não dependem dos símbolos vizinhos.

c) Que a linguagem não possui regras gramaticais fixas.

d) Que o programador é livre para inventar novos comandos.

8. Qual o componente de uma GLC que permite a recursividade?

a) O uso de variáveis no lado direito das produções.

b) A presença de símbolos terminais.

c) O símbolo inicial S.

d) O alfabeto Σ.

9. Em compiladores, a análise baseada em GLC ocorre em qual fase?

a) Análise Léxica.

b) Análise Sintática (Parsing).

c) Otimização de Código.

d) Ligação (Linking).

10. Diverio (2011) associa as LLC ao reconhecimento por qual máquina?

a) Autômato Finito.

b) Autômato de Pilha.

c) Máquina de Turing.

d) Processador RISC.

11. Se uma gramática tem a regra S → aSb | ab, qual string ela gera?

a) abab

b) aabb

c) aba

d) baab

12. A estratégia de "Derivação à Esquerda" consiste em:

a) Começar a ler a string pelo final.

b) Expandir sempre o não-terminal mais à esquerda na sentença atual.

c) Excluir todas as regras que usam a letra 'a'.

d) Mover todas as variáveis para o lado esquerdo da equação.

II. Perguntas Reflexivas

1. **Ambiguidade na Vida Real:** Imagine se a gramática do Java fosse ambígua. Quais seriam as consequências para um desenvolvedor que escrevesse um sistema bancário? Como garantir que o compilador interpreta o código exatamente como o programador pensou?

2. **Evolução:** Por que não usamos apenas GLC para tudo, já que elas são mais poderosas que as Expressões Regulares? O que ganhamos em manter a análise léxica (ER) separada da análise sintática (GLC)?

III. Exercícios Práticos

1. **Construção de GLC:** Desenvolva uma GLC que gere a linguagem $L = \{ a^n b^{2n} \mid n \geq 1 \}$. (Ou seja, para cada 'a', deve haver sempre o dobro de 'b's).

2. **Árvore Sintática:** Dada a gramática:

$$E \rightarrow E + E \mid E * E \mid id$$

Desenhe (ou descreva) as duas árvores possíveis para a expressão id + id * id.

3. **Modelagem de Bloco:** Escreva uma regra de produção simplificada para representar um bloco de código que começa com {, termina com } e pode conter uma lista de comandos ou outros blocos aninhados.

Unidade 7: Autômatos de Pilha (AP)

7.1 A Necessidade da Memória Auxiliar

Como discutido por **Menezes (2011)**, os Autômatos Finitos (AFD/AFN) possuem uma "memória" limitada aos seus estados. Eles conseguem lembrar onde estão, mas não conseguem "contar" ou "armazenar" informações para uso posterior.

Para reconhecer linguagens como $L = \{a^n b^n \mid n \geq 0\}$, a máquina precisa de uma forma de lembrar quantos 'a's ela leu para depois compará-los com a quantidade de 'b's. A solução teórica e prática para isso é o acréscimo de uma **Pilha (Stack)** ao autômato.

7.2 Definição Formal do Autômato de Pilha (AP)

De acordo com **Sipser (2007)**, um Autômato de Pilha é uma 7-tupla (sétupla):

$$M = (Q, \Sigma, \Gamma, , \delta, q_0, Z_0, F):$$

1. **Q:** Conjunto finito de estados.
2. **Σ:** Alfabeto de entrada (os símbolos da string).
3. **Γ (Gama):** Alfabeto da pilha (os símbolos que podem ser guardados na memória).
4. **δ:** Função de transição.
5. **q_0:** Estado inicial.
6. **Z_0:** Símbolo inicial da pilha (marcador de fundo).
7. **F:** Conjunto de estados de aceitação.

A Função de Transição (δ)

A função de transição mapeia a configuração atual para o próximo passo:

$$\delta: Q \times (\Sigma \cup \{\epsilon\}) \times \Gamma \rightarrow P(Q \times \Gamma^*).$$

δ: Diferente do AFD, no **AP** considera:

Estado Atual, Símbolo Lido, Símbolo no Topo da Pilha → Novo Estado, Símbolo a ser empilhado.

7.3 O Funcionamento da Pilha: LIFO e Recursividade

A pilha opera sob a lógica **LIFO** (*Last In, First Out* - o último a entrar é o primeiro a sair). **Diverio (2011)** destaca que essa estrutura é a base da **recursividade** na computação.

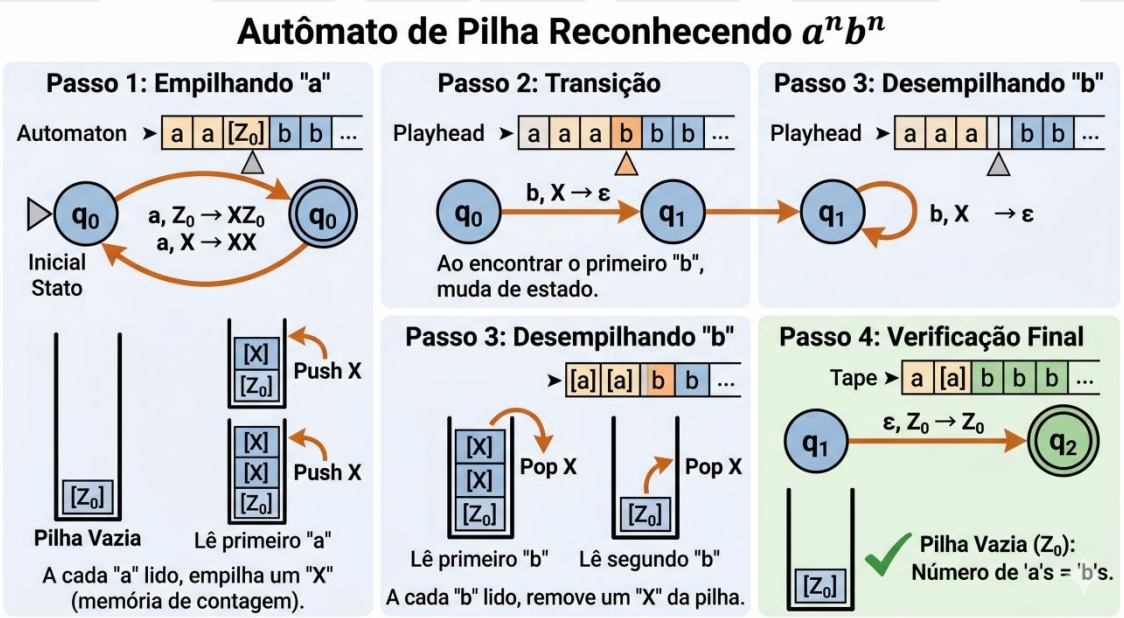
Mecanismo de Transição

A cada passo, o AP pode realizar três operações na pilha:

- **Push (Empilhar):** Adiciona um símbolo ao topo.
- **Pop (Desempilhar):** Remove o símbolo do topo.
- **No-op (Manter):** Lê o topo mas não altera a pilha.

Exemplo Prático: Reconhecendo $a^n b^n$

1. Para cada 'a' lido na entrada, a máquina faz um **Push** de um símbolo (ex: 'X') na pilha.
2. Ao encontrar o primeiro 'b', a máquina muda de estado.
3. Para cada 'b' lido, ela faz um **Pop** de um 'X'.
4. Se ao final da string a pilha estiver vazia (apenas com o marcador Z_0), significa que a quantidade de 'a's e 'b's era idêntica.



Definição Formal do AP $(Q, \Sigma, \Gamma, , \delta, q_0, Z_0, F)$:

$Q = \{q_0, q_1, q_2\}$: É o conjunto finito de estados do autômato.

q_0: Estado de leitura e empilhamento dos símbolos a.

q_1: Estado de leitura e desempilhamento com os símbolos b.

q_2: Estado de aceitação/parada final.

Σ = {a, b}: É o alfabeto de entrada (os símbolos que a fita de entrada pode conter).

Γ = {X, Z_0}: É o alfabeto da pilha (os símbolos que podem ser armazenados na memória auxiliar).

X: Símbolo utilizado como unidade de contagem para cada a lido.

Z_0: Símbolo especial que marca o fundo da pilha.

$q_0 \in Q$: É o estado inicial onde o processamento começa.

$Z_0 \in \Gamma$: É o símbolo inicial da pilha (a pilha começa a computação contendo apenas este elemento).

F = {q^2}: É o conjunto de estados finais ou de aceitação (representado graficamente pelo círculo duplo).

7.4 Autômatos de Pilha e Análise Sintática

Na construção de compiladores, o Autômato de Pilha é o motor do **Parser** (Analisador Sintático). Enquanto o **analisador léxico** entrega "palavras", o **analisador de pilha** verifica se a estrutura faz sentido. Por exemplo, ao encontrar um abre parênteses (, o compilador faz um **Push**. Ele pode ler mil linhas de código (recursividade), mas ele só aceitará o programa se, ao final, houver um **Pop** correspondente para cada (empilhado. Se a pilha não terminar vazia, o compilador gera o erro: *"Unbalanced parentheses"*.

7.5 Determinismo vs. Não-Determinismo em APs

Aqui reside uma diferença crucial apontada por **Menezes (2011)**:

- **AP Determinístico (APD):** Existe apenas um caminho. É o que usamos em linguagens de programação para garantir compilação rápida.
- **AP Não-Determinístico (APN):** Pode tentar vários caminhos (útil para linguagens naturais).
- **Diferença Vital:** Diferente dos Autômatos Finitos, um AP Não-Determinístico é **mais poderoso** que um Determinístico. Existem linguagens livres de contexto que *só* podem ser reconhecidas por um APN.

I. Perguntas de Múltipla Escolha

1. **O que diferencia fundamentalmente um Autômato de Pilha de um Autômato Finito?**

a) O número de estados iniciais.

b) A presença de uma memória auxiliar do tipo pilha.

c) A capacidade de ler a fita de trás para frente.

d) O uso de alfabetos binários apenas.

2. Qual a política de acesso aos dados em uma pilha?

a) FIFO (First In, First Out).

b) Aleatória.

c) LIFO (Last In, First Out).

d) Apenas leitura.

3. Na 7-tupla do AP, o que representa o símbolo Γ (Gama)?

a) O alfabeto de símbolos que podem ser armazenados na pilha.

b) O conjunto de estados finais.

c) A função de transição.

d) O tempo de execução da máquina.

4. Para que serve o símbolo Z_0 na base da pilha?

a) Para indicar que a máquina deve parar e dar erro.

b) Para marcar o fundo da pilha e permitir que a máquina saiba quando ela está vazia.

c) Para representar a cadeia vazia na entrada.

d) Para aumentar o poder de processamento do processador.

5. Um Autômato de Pilha pode reconhecer a linguagem $\{a^n b^n c^n\}$?

a) Sim, usando duas pilhas simultâneas.

b) Sim, é uma linguagem regular simples.

c) Apenas se o número n for menor que 10.

d) Não, pois uma única pilha só consegue comparar dois elementos por vez (Tipo 2).

6. O que acontece na operação de "Pop"?

a) Um novo símbolo é adicionado ao topo da pilha.

b) O símbolo do topo da pilha é removido.

c) A máquina muda de estado sem ler nada.

d) O alfabeto da pilha é reiniciado.

7. **Segundo Sipser (2007), os APs são os reconhecedores de qual classe de linguagens?**

a) Linguagens Livres de Contexto (Tipo 2).

b) Linguagens Regulares.

c) Linguagens de Máquina.

d) Linguagens Sensíveis ao Contexto.

8. **Qual a aplicação prática mais comum dos Autômatos de Pilha?**

a) Validação de nomes de usuário em sites.

b) Análise sintática de códigos-fonte em compiladores.

c) Ordenação de grandes bancos de dados.

d) Compactação de arquivos ZIP.

9. **Na função de transição δ(q, a, X) = (p, Γ), o que o 'X' representa?**

a) O próximo símbolo da fita de entrada.

b) Um erro de hardware.

c) O símbolo que está atualmente no topo da pilha.

d) O estado final.

10. **Diverio (2011) afirma que a recursividade é implementada via pilha porque:**

a) A pilha economiza energia.

b) A pilha permite "empilhar" o estado de uma função e voltar para ele após a execução.

c) A pilha é a estrutura mais rápida da memória RAM.

d) A pilha impede que o programa sofra ataques de hackers.

11. Dizemos que um AP aceita uma cadeia por "Pilha Vazia" quando:

a) A máquina para em qualquer estado e a pilha não tem mais nenhum símbolo além de Z_0.

b) A fita de entrada termina com um zero.

c) O programador deleta o código.

d) A máquina entra em loop infinito.

12. Sobre o não-determinismo nos APs, é correto afirmar:

a) APNs e APDs têm exatamente o mesmo poder.

b) O APN é estritamente mais poderoso que o APD.

c) O AFD é mais poderoso que o APN.

d) Não existe não-determinismo em máquinas com pilha.

II. Perguntas Reflexivas

1. **Hierarquia de Memória:** Como a adição de uma estrutura simples (Pilha) altera drasticamente o tipo de problema que um computador pode resolver? Por que uma "Pilha" é melhor que uma "Fila" para validar linguagens de programação?

2. **O Limite:** Se o Autômato de Pilha consegue contar dois elementos ($a^n b^n$), por que ele falha em contar três ($a^n b^n c^n$)? O que isso nos diz sobre a estrutura de dados pilha?

III. Exercícios Práticos

1. **Simulação de Pilha:** Descreva o estado da pilha passo a passo ao processar a cadeia (( ) ( )). Considere que cada (faz um push de 'X' e cada) faz um pop de 'X'.

2. **Modelagem:** Desenhe (ou descreva) um AP que aceite a linguagem dos palíndromos sobre {a, b} de comprimento par (cadeias como abba, baab, aaaa). Dica: use o não-determinismo para decidir quando a cadeia chegou ao meio e começar a desempilhar.

3. **Análise de Erro:** Explique por que um AP falharia ao processar a cadeia ()). O que aconteceria com a pilha e qual seria o veredito final da máquina?

Unidade 8: Máquina de Turing (MT) e a Computabilidade

8.1 Origem e História: O Sonho de Alan Turing

Em 1936, muito antes dos computadores de silício existirem, um jovem matemático britânico chamado **Alan Turing** publicou o artigo *"On Computable Numbers"*. O seu objetivo não era construir uma máquina física, mas responder a uma pergunta filosófica e matemática: **Existe algum limite para o que pode ser calculado?**

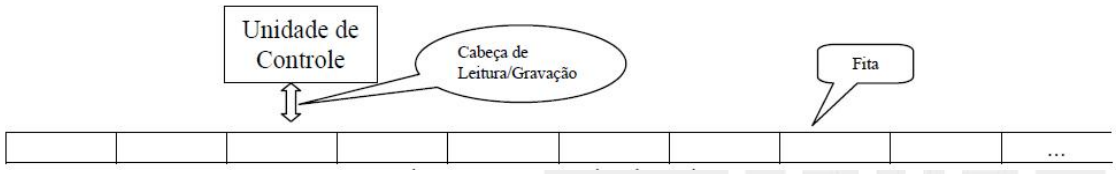
Naquela época, **"computador"** era o cargo de pessoas que faziam cálculos repetitivos em escritórios. Turing imaginou uma pessoa com um lápis, uma borracha e uma fita de papel infinita.

A pessoa poderia ler um símbolo, apagá-lo, escrever um novo e mover-se para o quadrado ao lado. Esse exercício mental deu origem à **Máquina de Turing**, provando que qualquer cálculo matemático que um ser humano possa fazer seguindo um algoritmo, uma máquina também pode. Essa é a base de toda a arquitetura moderna de computadores.

8.2 Anatomia e Funcionamento de uma MT

Segundo **Menezes (2011)** e **Sipser (2007)**, uma Máquina de Turing é composta por:

1. **Fita Infinita:** Dividida em células, servindo como memória de entrada, saída e trabalho. Cada célula contém um símbolo de um alfabeto **Γ** (Gama).
2. **Cabeça de Leitura/Gravação:** Pode ler o símbolo atual, escrever um novo símbolo e mover-se para a **Esquerda (L)** ou **Direita (R)**.
3. **Unidade de Controle:** Um conjunto finito de estados que decide o que fazer com base no símbolo lido.




```
Fita Inicial:  | 1 | 0 | 1 | 1 |
                 ^ (Cabeça aqui)

Passo 1 (R):   | 1 | 0 | 1 | 1 |
                     ^ (Moveu 1 quadrado)

Passo 2 (R):   | 1 | 0 | 1 | 1 |
                         ^ (Moveu +1 quadrado)
```

Componentes da Sétupla (M)

Para **Sipser (2007)**, a formalização é o que permite provar o que é impossível computar. A MT é definida por:

$$M = (Q, \Sigma, \Gamma, \delta, q_0, q_{aceita}, q_{rejeita}):$$

1. **Q:** Conjunto finito de **estados**.
2. **Σ:** Alfabeto de **entrada** (não contém o símbolo branco ⊔).
3. **Γ (Gama):** Alfabeto da **fita**. Inclui **Σ** e o símbolo branco ⊔ .
4. **δ:** Função de transição: $Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$.
    - *Tradução:* (Estado atual, Símbolo lido) → (Novo estado, Símbolo escrito, Direção do movimento Esq/Dir).
5. **q_0:** Estado inicial.
6. **q_{aceita}:** O estado de parada onde a máquina diz "Sim".
7. **$q_{rejeita}$:** O estado de parada onde a máquina diz "Não".

Abaixo, explicamos detalhadamente o significado e a função prática de cada um destes 7 componentes:

i. Q — Conjunto Finito de Estados

É o "cérebro" interno da máquina. Representa todos os estados possíveis em que o controle do sistema se pode encontrar durante um cálculo. Embora a fita da máquina seja infinita, o número de estados lógicos internos é sempre fixo e finito (ex: $Q = \{q_0, q_1, q_2\}$).

ii. Σ (Sigma) — Alfabeto de Entrada

É o conjunto de símbolos que podem ser utilizados para escrever a palavra inicial na fita (a entrada do programa). Uma regra crucial do formalismo é que o símbolo branco não pode pertencer a este alfabeto (⊔ ∉ **Σ**). Geralmente, utiliza-se o alfabeto binário **Σ** = {0, 1}.

iii. Γ (Gamma) — Alfabeto da Fita

É o alfabeto completo de símbolos que a máquina consegue ler ou escrever na fita de memória. Ele é sempre maior do que o alfabeto de entrada, pois obedece à seguinte regra de inclusão:

- Contém todos os símbolos do alfabeto de entrada (**Σ** ⊂ **Γ**).

➢ Contém obrigatoriamente o símbolo especial branco ($\sqcup \in \Gamma$), que indica uma célula vazia.

➢ Pode conter caracteres auxiliares de marcação (como X, Y, *) que a máquina utiliza para sinalizar posições já processadas.

iv. δ (Delta) — Função de Transição

É o motor lógico que dita o comportamento passo a passo da máquina. Ao contrário dos autômatos simples, a função de transição da Máquina de Turing lê um símbolo, escreve um novo símbolo sobre a fita e move a cabeça de leitura.

O seu mapeamento matemático é definido como:

$$\delta: Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}.$$

Como ler uma transição prática:

Se a máquina possuir a regra $\delta(q_0, 0) = (q_1, 1, R)$, a leitura mecânica é:

1. Se o estado atual for q_0 e a cabeça ler o símbolo 0 na fita:
2. A máquina altera o seu estado interno para q_1.
3. Substitui (escreve) o 0 por 1 naquele quadrado da fita.
4. Move a cabeça de leitura exatamente um quadrado para a direita (R - Right). Se fosse L (Left), moveria um quadrado para a esquerda.

v. q_0 — Estado Inicial

É o estado pertencente a Q ($q_0 \in Q$) por onde a Máquina de Turing inicia obrigatoriamente a sua execução assim que é ligada com a fita contendo a palavra de entrada.

vi. q_{aceita} — Estado de Aceitação

É o estado final de paragem positiva ($q_{aceita} \in Q$). No exato momento em que o fluxo de controlo da máquina entra neste estado, a execução é interrompida imediatamente e a cadeia de entrada é considerada válida (aceite pela linguagem).

vii. $q_{rejeita}$ — Estado de Rejeição

É o estado final de paragem negativa ($\$q_{rejeita} \in Q$). À semelhança do estado de aceitação, se a máquina transitar para $\$q_{rejeita}$, a computação para na hora e a

cadeia de entrada é considerada inválida (rejeitada). Por rigor formal, o estado de rejeição deve ser diferente do estado de aceitação ($q_{aceita} \neq q_{rejeita}$).

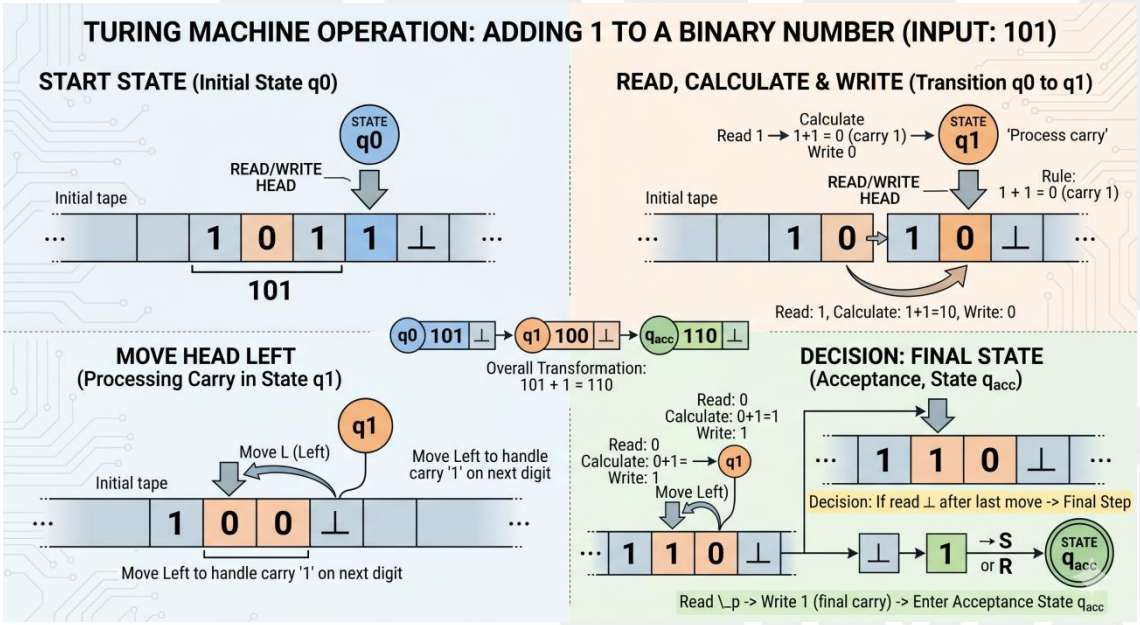
8.3 Funcionamento Didático: O Passo a Passo

Diferente do Autômato de Pilha, a MT pode **ler e escrever em qualquer lugar da fita**.

Exemplo: Adicionar 1 a um número binário

Imagine a fita com a entrada 101 e a cabeça de leitura no último dígito.

- **Etapa 1 (Leitura):** A máquina lê 1.
- **Etapa 2 (Cálculo/Escrita):** Pela regra da soma, 1 + 1 = 10 e "vai um". A máquina escreve 0 na fita.
- **Etapa 3 (Movimento):** A cabeça move-se para a esquerda (L) para processar o próximo dígito com o "vai um".
- **Etapa 4 (Decisão):** Se a máquina ler um branco ␣ após todos os movimentos, ela escreve o último "vai um" e entra em q_{aceita}.



8.4 Tipos de Máquinas de Turing

Embora o modelo básico seja o mais estudado, existem variações que facilitam a programação, embora **todas tenham o mesmo poder computacional**:

- **MT com Múltiplas Fitas:** Útil para copiar dados rapidamente.
- **MT Não-Determinística:** Pode explorar vários caminhos (importante na teoria P vs NP).
- **MT Enumeradora:** Uma máquina que "imprime" todas as strings de uma linguagem.
- **Máquina de Turing Universal (UTM):** Uma MT que pode ler a descrição de *outra* MT e simulá-la. É o conceito de **Software**.

8.5 Teoria da Decidibilidade: O que é Impossível?

Menezes (2011) e **Diverio (2011)** enfatizam que nem tudo pode ser resolvido por algoritmos. Aqui entramos na distinção prática:

1. **Linguagem Decidível:** Existe uma MT que, para qualquer entrada, sempre para e dá uma resposta (Aceita ou Rejeita).
    - *Exemplo:* Verificar se um número é primo. O computador sempre terminará o cálculo.
2. **Linguagem Reconhecível (mas não decidível):** A máquina para se a resposta for "Sim", mas pode entrar em **loop infinito** se a resposta for "Não".

O Problema da Parada (Halting Problem)

Este é o exemplo prático mais famoso. É possível criar um programa que leia o código de *qualquer* outro programa e diga se ele vai travar (loop infinito) ou não?

Alan Turing provou que NÃO.

Se tentarmos criar esse "super verificador", chegaremos a uma contradição lógica. Isso prova que existem limites fundamentais para o que podemos programar.

I. Perguntas de Múltipla Escolha

1. **Qual a principal inovação da Máquina de Turing em relação ao Autômato de Pilha?**

a) O uso de múltiplas pilhas.

b) Uma fita infinita onde se pode ler e escrever em qualquer direção.

c) A capacidade de processar apenas números decimais.

d) O fato de ser uma máquina física feita de metal.

2. **O que o símbolo ␣ (branco) representa na fita da MT?**

a) Um espaço vazio da memória ainda não utilizado.

b) Um erro de processamento.

c) O fim obrigatório do alfabeto de entrada.

d) O estado inicial da máquina.

3. **Na função de transição da MT, o que significam os símbolos {L, R}?**

a) *Low* e *Rich* (níveis de energia).

b) *Loop* e *Reset*.

c) *Left* (Esquerda) e *Right* (Direita), as direções de movimento da cabeça.

d) *Language* e *Recursion*.

4. **Uma Máquina de Turing Universal (UTM) é capaz de:**

a) Resolver problemas que nem Alan Turing resolveu.

b) Simular qualquer outra Máquina de Turing a partir de sua descrição.

c) Funcionar sem energia elétrica.

d) Ler apenas linguagens regulares.

5. **Segundo a Tese de Church-Turing:**

a) Computadores quânticos são mais fracos que a MT.

b) Autômatos finitos são superiores a máquinas de Turing.

c) A internet é uma linguagem de Tipo 3.

d) Qualquer algoritmo computável pode ser executado por uma Máquina de Turing.

6. **Uma linguagem é considerada "Decidível" quando:**

a) A máquina entra em loop infinito para entradas falsas.

b) A máquina sempre para e decide entre aceitação ou rejeição para qualquer entrada.

c) O programador decide as regras na hora.

d) Ela possui menos de 10 estados.

7. O "Problema da Parada" provou que:

a) Todos os programas de computador podem ser verificados contra loops.

b) A memória RAM deve ser sempre par.

c) Turing era um excelente programador de Java.

d) Existem problemas que são logicamente impossíveis de serem resolvidos por algoritmos.

8. Qual componente da sétupla define o alfabeto que pode ser escrito na fita?

a) Σ

b) Γ

c) Q

d) δ

9. Diverio (2011) afirma que as linguagens aceitas pela MT são de qual tipo na Hierarquia de Chomsky?

a) Tipo 3 (Regulares).

b) Tipo 2 (Livres de Contexto).

c) Tipo 0 (Recursivamente Enumeráveis).

d) Tipo 4 (Naturais).

10. Diferente dos autômatos anteriores, quando a MT atinge o estado q_{aceita}, ela:

a) Volta para o início da fita.

b) Para imediatamente e termina a execução.

c) Apaga toda a fita.

d) Entra em standby.

11. O que é um "loop" em uma Máquina de Turing?

a) Quando a máquina nunca atinge um estado de parada (q_{aceita} ou $q_{rejeita}$).

b) Uma estrutura de repetição for otimizada.

c) Um tipo de fita circular.

d) Uma transição-ε.

12. **A importância da MT para a computação moderna é que ela:**

a) Serve como modelo teórico para a CPU e a memória RAM.

b) É a linguagem mais usada para criar sites.

c) Substitui o uso de bancos de dados.

d) Foi o primeiro sistema operacional Windows.

II. Perguntas Reflexivas

1. **O Limite do Pensamento:** Se a Máquina de Turing representa o limite do que é computável, você acredita que a inteligência humana pode realizar cálculos que uma MT não consegue, ou nosso cérebro é apenas uma MT biológica muito complexa?

2. **Praticidade:** Como o conhecimento de que existem problemas indecidíveis (como o Problema da Parada) afeta o dia a dia de um engenheiro de software ao criar ferramentas de teste automatizado?

III. Exercícios Práticos

1. **Rastreamento de Fita:** Imagine uma MT que inverte bits (0 vira 1 e 1 vira 0). Se a fita inicial é 110, descreva o conteúdo da fita após 3 passos de execução.

2. **Desenho Lógico:** Projete as transições (δ) para uma MT que deve mover a cabeça de leitura até encontrar o primeiro símbolo branco ␣ à direita.

3. **Decidibilidade:** O problema "Verificar se um programa tem mais de 100 linhas de código" é decidível ou indecidível? Justifique com base nos conceitos da unidade.

Unidade 9: Prática Integrada – Modelagem e Implementação

9.1 A Arte da Abstração e Escolha do Formalismo

Como engenheiro ou cientista da computação, a sua primeira tarefa não é programar, mas **identificar a complexidade do problema**. Segundo **Menezes (2011)**, se escolhermos um formalismo mais complexo do que o necessário (ex: usar uma Máquina de Turing para validar um e-mail), desperdiçamos recursos. Se escolhermos um mais simples (ex: usar um AFD para validar expressões matemáticas), o problema não será resolvido.

Guia de Decisão Profissional:

- **Problema de Reconhecimento de Padrão Simples:** (Ex: Datas, Placas de Carro, Palavras-chave).
    - **Formalismo:** Linguagens Regulares (AFD, AFN ou Expressões Regulares).
    - **Porquê:** São extremamente rápidos e ocupam pouca memória.

- **Problema de Estrutura Aninhada ou Balanceada:** (Ex: XML, JSON, Fórmulas Matemáticas).
    - **Formalismo:** Linguagens Livres de Contexto (Gramáticas GLC ou Autômatos de Pilha).
    - **Porquê:** Exigem memória para "lembrar" o que foi aberto e o que deve ser fechado.

- **Problema de Lógica Complexa ou Cálculo Geral:** (Ex: Algoritmos de ordenação, IA, Compiladores completos).
    - **Formalismo:** Máquina de Turing / Linguagens Recursivamente Enumeráveis.
    - **Porquê:** Representam a computabilidade total.

9.2 Do Formalismo ao Código (Implementação)

Na sua **APS**, será solicitado que relacione o formalismo com o código. Veja como isso se traduz na prática profissional:

1. **Expressões Regulares (ER):** Traduzem-se diretamente em funções de biblioteca (ex: re.match() em Python ou Pattern.matches() em Java).

2. **Autômatos (AFD):** Traduzem-se em estruturas de controle switch-case dentro de um laço while, onde cada "case" é um estado.
3. **Gramáticas (GLC):** Traduzem-se em funções recursivas (Descida Recursiva) ou tabelas de parsing em ferramentas como **Yacc** ou **ANTLR**.

9.3 O Uso do Simulador JFLAP

Download: https://www.jflap.org/jflaptmp/

https://www2.cs.duke.edu/csed/jflap/jflapbook/jflapbook2006.pdf

Exercício 1: Validação de Padrões Binários

Contexto: No desenvolvimento de protocolos de comunicação de baixo nível, é comum o uso de bits de controle no início e no fim de uma mensagem para garantir que os dados não foram corrompidos.

No JFLAP:
Desenhe um **Autômato Finito Determinístico (AFD)** sobre o alfabeto Σ = {0, 1} que reconheça apenas cadeias que **comecem com 0 e terminem com 1**.

- **Exemplos de cadeias aceitas:** `01`, `001`, `0101`, `01111`
- **Exemplos de cadeias rejeitadas:** `11`, `00`, `101`, `0`

Diretrizes para o estudante no JFLAP:

1. Utilize a ferramenta *State Creator* para modelar os estados necessários (tente resolver usando o menor número de estados possível para evitar redundância).
2. Certifique-se de que o autômato seja **determinístico** (cada estado deve ter exatamente uma transição de saída para **0** e uma para **1**).
3. Abra o menu **Input -> Multiple Runs**, insira as cadeias de teste acima e verifique se o simulador exibe *Accept* e *Reject* corretamente.

Tabela de Transição

Estado Atual	Entrada '0'	Entrada '1'	Tipo de Estado / Significado
→q0	q1	qdead	Inicial: Cadeia vazia. Se ler 0, vai para o fluxo correto; se ler 1, vai para a lixeira.
q1	q1	q2	Monitor: Começou com 0, mas o último caractere lido foi 0 (Rejeição momentânea).
* q2	q1	q2	Final (Aceitação): Começou com 0 e o último caractere lido foi 1.
qdead	qdead	qdead	Lixeira (Dead State): Começou com 1. Rejeição definitiva para qualquer entrada subsequente.

Exercício 2: Validação de Tokens de Segurança (Modelagem de AFD)

Contexto: Em sistemas de autenticação de dois fatores (2FA), os servidores geram tokens numéricos temporários. Imagine que o seu sistema gere um token de segurança composto por exatamente 3 caracteres, e que, por exigência da equipe de criptografia, esse token deve conter pelo menos um dígito 1. Qualquer padrão que fuja dessa regra deve ser rejeitado para mitigar ataques de força bruta..

No JFLAP:
Utilizando o módulo **Finite Automaton** do JFLAP, construa um Autômato Finito Determinístico (AFD) sobre o alfabeto **Σ= {0, 1}**. O autômato deve aceitar apenas cadeias que tenham comprimento correspondente a exatamente **3 bits** e que possuam pelo menos **um bit 1** em qualquer posição.

- **Exemplos de cadeias aceitas** (comprimento 3 e tem '1'): 001, 010, 100, 101, 111
- **Exemplos de cadeias rejeitadas** (tamanho errado ou sem '1'): 000 (comprimento 3, mas sem '1'), 01 (comprimento menor que 3), 1101 (comprimento maior que 3), 1 (comprimento menor que 3).

Tabela de Transição

Estado Atual	Entrada '0'	Entrada '1'	Tipo de Estado / Significado
→q0	q1	q2	Inicial: Nenhum bit lido.
q1	q3	q4	1 bit lido. Nenhuma ocorrência de 1.
q2	q4	q5	1 bit lido. Já possui 1.
q3	q6	q7	2 bits lidos. Nenhuma ocorrência de 1.
q4	q7	q7	2 bits lidos. Já possui 1.

q5	q7	q7	2 bits lidos. Já possui 1.
q6	qdead	qdead	3 bits lidos. Rejeitado (é o padrão 000).
* **q7**	qdead	qdead	Final (Aceitação): 3 bits lidos e possui 1.
qdead	qdead	qdead	Lixeira (Dead State): String com mais de 3 bits.

I. Perguntas de Múltipla Escolha

1. Qual o primeiro passo ao receber um problema de computação?

a) Começar a digitar o código imediatamente.

b) Abstrair o problema e identificar a sua classe na Hierarquia de Chomsky.

c) Comprar um novo computador.

d) Ignorar as regras gramaticais.

2. Para validar se um arquivo XML tem todas as tags fechadas corretamente, qual o formalismo mais apropriado?

a) Expressões Regulares.

b) Autômato Finito Determinístico.

c) Gramática Livre de Contexto.

d) Tabela Verdade.

3. Por que as Expressões Regulares são tão usadas em validação de formulários web?

a) Porque são coloridas.

b) Porque são o formalismo mais eficiente para reconhecimento de padrões de strings sem aninhamento.

c) Porque podem resolver qualquer problema matemático.

d) Porque não precisam de processador para rodar.

4. No JFLAP, o que acontece quando uma string termina e o autômato está num estado que não é de aceitação?

a) A string é aceita.

b) O computador trava.

c) A string é rejeitada.

d) O JFLAP apaga o desenho.

5. A implementação de um AFD em código através de switch-case foca em:

a) Criar uma interface gráfica.

b) Mapear estados e transições de forma determinística.

c) Aumentar a velocidade da internet.

d) Reduzir o tamanho do disco rígido.

6. Se um problema exige que você compare a quantidade de 'A's, 'B's e 'C's (todos iguais), qual máquina deve usar?

a) Autômato Finito.

b) Autômato de Pilha.

c) Máquina de Turing.

d) Calculadora simples.

7. O que é o "rastreio" (trace) de uma execução?

a) É a busca por vírus no código.

b) É o acompanhamento passo a passo das mudanças de estado e memória da máquina.

c) É a tradução do código para inglês.

d) É a exclusão de comentários.

8. Na APS, a relação entre formalismo e código serve para demonstrar:

a) Que você sabe copiar código da internet.

b) Que você compreende a base teórica por trás da implementação prática.

c) Que o código é mais importante que a teoria.

d) Que os autômatos não servem para nada na prática.

9. Qual formalismo é a base para a criação de "Tokens" num compilador?

a) Gramática Livre de Contexto.

b) Expressões Regulares.

c) Máquina de Turing Universal.

d) Lógica Fuzzy.

10. Diverio (2011) sugere que a escolha errada de um formalismo pode causar:

a) Falta de memória ou loops infinitos desnecessários.

b) Quebra do monitor.

c) Melhoria no desempenho.

d) Mudança de cor no JFLAP.

11. Uma das vantagens de usar Gramáticas (GLC) em vez de Autômatos de Pilha no design de linguagens é:

a) A gramática é mais fácil de ler e manter por seres humanos.

b) A gramática é mais rápida que a máquina.

c) A gramática não precisa de regras.

d) A gramática só funciona para números.

12. O formalismo "mais apropriado" é aquele que:

a) É o mais complexo possível.

b) Resolve o problema com o menor nível de complexidade necessário na Hierarquia de Chomsky.

c) Foi inventado mais recentemente.

d) Não usa matemática.

II. Perguntas Reflexivas

1. **Simulação vs. Realidade:** Como o uso de ferramentas como o JFLAP impacta a confiança de um engenheiro antes de subir um código para produção?

2. **O Papel do Engenheiro:** Num mundo com IAs que geram código, por que o conhecimento profundo da Teoria da Computação e da Hierarquia de Chomsky ainda é o maior diferencial de um profissional de alto nível?

III. Exercícios Práticos (Simulados para a APS)

1. **Avaliação:** Identifique o formalismo mais simples para reconhecer a linguagem de CPFs (formato xxx.xxx.xxx-xx). Justifique.

2. **Modelagem Integrada:** Escreva uma GLC e depois desenhe um Autômato de Pilha equivalente para a linguagem de strings balanceadas compostas apenas por [e].

3. **Relação com Código:** Escreva um pequeno trecho de código (em qualquer linguagem) que implemente o comportamento de um AFD que aceita strings binárias que terminam em 1.

Unidade 1: Fundamentos Matemáticos e Introdução às Linguagens

1. Se um determinado alfabeto Σ possui exatamente 3 símbolos distintos, o número total de elementos que farão parte do seu Conjunto das Partes P(Σ) será de:

a) 3 elementos.

b) 6 elementos.

c) 8 elementos.

d) 9 elementos.

2. Na Teoria das Linguagens Formais, a palavra vazia (ϵ) possui propriedades matemáticas bem definidas. Em relação ao seu comprimento e comportamento na operação de concatenação, assinale a alternativa correta:

a) Seu comprimento é $|\epsilon| = 1$ e ela altera o tamanho de qualquer string que sofra concatenação com ela.

b) Seu comprimento é $|\epsilon| = 0$ e ela atua como o elemento neutro na operação de concatenação de cadeias.

c) Seu comprimento é variável de acordo com a cardinalidade do alfabeto Σ adotado pelo autômato.

d) Seu comprimento é zero, mas quando concatenada com uma string w, o resultado final será sempre ϵ.

3. Considere o alfabeto binário Σ = {0, 1}. Qual a diferença fundamental entre as linguagens geradas pelas operações de Fechamento de Kleene (Σ*) e Fechamento Positivo (Σ^+)?

a) O conjunto Σ* é infinito, enquanto o conjunto Σ_+ possui tamanho finito.

b) O conjunto Σ^+ inclui a cadeia vazia (ϵ), enquanto o conjunto Σ* a exclui obrigatoriamente.

c) O conjunto Σ* inclui a cadeia vazia (ϵ), enquanto o conjunto Σ^+ diferencia-se por não incluí-la.

d) Ambos os conjuntos são idênticos em estrutura, diferenciando-se apenas na nomenclatura acadêmica utilizada.

4. Na Hierarquia de Chomsky, as Linguagens Livres de Contexto pertencem a qual classificação e qual é o seu modelo computacional reconhecedor padrão?

a) Tipo 3 — Autômatos Finitos Determinísticos.

b) Tipo 2 — Autômatos de Pilha.

c) Tipo 1 — Autômatos Linearmente Limitados.

d) Tipo 0 — Máquinas de Turing.

5. Qual classe de linguagens da Hierarquia de Chomsky representa o nível máximo de abrangência computacional, englobando tudo o que pode ser resolvido por meio de um algoritmo moderno?

a) Linguagens Regulares (Tipo 3).

b) Linguagens Livres de Contexto (Tipo 2).

c) Linguagens Sensíveis ao Contexto (Tipo 1).

d) Linguagens Recursivamente Enumeráveis (Tipo 0).

Unidade 2: Autômatos Finitos Determinísticos (AFD)

6. Em termos de capacidade de armazenamento, qual a principal limitação de um Autômato Finito Determinístico (AFD) que o diferencia de um computador moderno?

a) O AFD não consegue processar cadeias construídas com caracteres alfabéticos, apenas bits binários.

b) O AFD possui uma memória extremamente limitada, sendo capaz de "lembrar" apenas em qual estado se encontra no momento.

c) O AFD exige que a fita de entrada seja apagada por completo a cada transição de estado realizada.

d) O AFD depende obrigatoriamente de uma memória auxiliar do tipo fila para processar cadeias.

7. Na representação gráfica de um autômato finito, as convenções visuais para o Estado Inicial e para o Estado de Aceitação são estruturadas, respectivamente, por:

a) Um círculo duplo e uma seta apontada para o estado.

b) Uma seta apontada para o estado e um círculo duplo.

c) Um asterisco ao lado do estado e uma seta bidirecional.

d) Uma linha tracejada e um círculo preenchido.

8. Seja um AFD configurado com uma função de transição $\delta(q_0, 0) = q_1$. Isso significa que o determinismo da máquina impõe que:

a) A partir do estado q_0, ao ler o símbolo 0, o autômato possui caminhos paralelos para múltiplos estados.

b) A partir do estado q_0, ao ler o símbolo 0, a máquina pode escolher entre mudar para q_1 ou permanecer em q_0 sem consumir a entrada.

c) Para o par composto pelo estado q_0 e o símbolo 0, existe exatamente um único próximo estado destino, que é q_1.

d) A máquina irá realizar uma transição espontânea para q_1 mesmo se a fita estiver vazia.

9. Para processar uma cadeia de caracteres completa de uma única vez em vez de analisar símbolo por símbolo, a literatura estende a função de transição padrão δ para a chamada:

a) Função de Transição Nula (δ_0).

b) Função de Transição Estendida (δ^*).

c) Função de Equivalência Direta (δ^+).

d) Matriz de Transição Paralela (δ^p).

10. Suponha um AFD operando sobre o alfabeto $Σ = {0, 1\}. Ao final da leitura de uma string w, a máquina para em um estado q_k. Para que essa string w seja considerada oficialmente "aceita" pela linguagem do autômato, o estado q_k deve:

a) Ser o estado inicial q_0.

b) Ser um estado isolado sem nenhuma transição de saída.

c) Pertencer ao conjunto de estados finais/aceitação (F).

d) Ser um "estado morto" (lixeira).

Unidade 3: Autômatos Finitos Não-Determinísticos (AFN) e Transições-ε

11. Uma das características que distinguem o Autômato Finito Não-Determinístico (AFN) do Determinístico (AFD) é a possibilidade de o AFN:

a) Reconhecer linguagens de complexidade superior à das linguagens regulares.

b) Apresentar zero, um ou vários estados destinos possíveis para um mesmo par de estado e símbolo de entrada.

c) Voltar e reler caracteres que já foram processados e deixados para trás na fita de entrada.

d) Funcionar sem a necessidade de definir um estado inicial de partida.

12. As transições vazias ou transições-ϵ (ϵ-transitions) presentes em modelos de AFN-ϵ permitem que a máquina:

a) Mude de estado de forma espontânea, sem a necessidade de consumir qualquer símbolo da fita de entrada.

b) Aceite automaticamente qualquer cadeia que contenha erros de sintaxe graves.

c) Ignore as regras de restrição impostas pela quíntupla formal do autômato.

d) Realize a escrita de novos símbolos na fita de leitura.

13. O conceito de Fecho-ϵ (ϵ-closure) de um estado q em um autômato não-determinístico define:

a) O conjunto de todas as strings que fazem o autômato travar a partir de q.

b) O conjunto de todos os estados que podem ser alcançados a partir de q realizando exclusivamente transições vazias (ϵ).

c) O número total de vezes que o autômato pode ignorar a leitura do bit zero.

d) O estado final de aceitação que encerra a execução do sistema paralelo.

14. Em relação ao poder matemático de reconhecimento de linguagens, a aplicação do Algoritmo de Construção de Subconjuntos prova que:

a) O AFN é estritamente mais poderoso que o AFD, reconhecendo linguagens que o AFD não consegue.

b) Todo AFD pode ser convertido em um AFN, mas o inverso é impossível.

c) AFDs e AFNs são equivalentes em poder de reconhecimento, gerando exatamente a mesma classe de linguagens regulares.

d) A conversão de um AFN para AFD reduz drasticamente o número de estados do autômato resultante para metade.

15. Ao mapear um AFN para uma Tabela de Transição Matricial, cada célula da tabela não conterá apenas um único estado, mas sim:

a) Uma lista de ações semânticas de erro.

b) Um conjunto contendo todos os estados possíveis que o sistema pode assumir em paralelo após processar aquele símbolo.

c) O valor numérico correspondente ao comprimento total da string.

d) Uma instrução de parada definitiva da fita de leitura.

Unidade 4: Expressões Regulares (ER) e Análise Léxica

16. Na construção de compiladores, as Expressões Regulares são ferramentas matemáticas utilizadas em qual fase do ciclo de tradução e com qual objetivo principal?

a) Análise Semântica — para validar os tipos de dados de variáveis declaradas.

b) Análise Sintática — para estruturar a árvore de derivação hierárquica do código.

c) Análise Léxica — para identificar padrões rígidos de texto e agrupar sequências de caracteres em tokens.

d) Otimização de Código — para reduzir o consumo de memória RAM do software.

17. O Teorema de Kleene estabelece uma equivalência fundamental na Teoria da Computação. Esse teorema prova a equivalência entre as classes de linguagens descritas por:

a) Gramáticas Livres de Contexto e Autômatos de Pilha.

b) Expressões Regulares e Autômatos Finitos (AFD/AFN).

c) Máquinas de Turing e Linguagens Naturais.

d) Expressões Regulares e Linguagens Sensíveis ao Contexto.

18. Qual é a ordem hierárquica correta de precedência dos operadores das Expressões Regulares teóricas, listada da maior prioridade para a menor prioridade?

a) União, Concatenação, Estrela (Fecho de Kleene).

b) Concatenação, União, Estrela (Fecho de Kleene).

c) Estrela (Fecho de Kleene), Concatenação, União.

d) União, Estrela (Fecho de Kleene), Concatenação.

19. Dada a Expressão Regular teórica R = (a|b)*abb, assinale a alternativa que apresenta uma cadeia que pertence à linguagem gerada por R:

a) abab

b) bbaabb

c) abba

d) aabbb

20. O que acontece se, durante a fase de análise léxica de um programa de computador, o fluxo de caracteres da fita de entrada apresentar um caractere que não é reconhecido por nenhuma das Expressões Regulares dos tokens cadastrados?

a) O compilador ignora o caractere silenciosamente e avança para a próxima linha.

b) É gerado um erro léxico, indicando a presença de um caractere inválido para a linguagem.

c) O analisador léxico converte o caractere automaticamente para binário.

d) O programa é forçado a entrar em uma estrutura de loop infinito.

Unidade 5: Gramáticas Regulares e a Hierarquia de Chomsky

21. Uma gramática regular linear à direita caracteriza-se estruturalmente por apresentar regras de produção onde o lado direito da seta possui:

a) Apenas variáveis não-terminais nas duas extremidades.

b) Um símbolo terminal seguido (ou não) de uma única variável não-terminal posicionada na extremidade direita.

c) Uma cadeia vazia obrigatoriamente precedida por duas variáveis não-terminais.

d) Apenas símbolos terminais dispostos à esquerda do axioma.

22. Para provar com rigor matemático que uma determinada linguagem de interesse NÃO é regular, os cientistas da computação utilizam qual ferramenta teórica?

a) O Teorema de Kleene.

b) O Algoritmo de Construção de Subconjuntos.

c) O Teorema do Bombeamento (Pumping Lemma).

d) A Notação BNF (Backus-Naur Form).

23. O processo de aplicação sucessiva das Regras de Produção (P) a partir do Símbolo Inicial (S) até que a cadeia resultante contenha apenas símbolos terminais é denominado formalmente como:

a) Concatenação Estrutural.

b) Derivação.

c) Minimização de Estados.

d) Fecho Relacional.

24. A linguagem clássica $L = \{a^n b^n \mid n \geq 0\}$ não pode ser classificada como uma linguagem regular (Tipo 3) porque sua estrutura exige:

a) Um alfabeto infinito composto por símbolos gregos.

b) Uma capacidade de memória de contagem emparelhada infinita que os estados fixos de um autômato finito não possuem.

c) A validação de contextos semânticos dependentes da arquitetura do processador.

d) O uso obrigatório de transições-ϵ bidirecionais.

Unidade 6: Linguagens Livres de Contexto (LLC) e Gramáticas Livres de Contexto (GLC)

25. Na regra de produção formal do tipo $A \rightarrow \alpha$, o que define o critério de uma gramática ser classificada especificamente como "Livre de Contexto" (GLC)?

a) O termo α à direita deve conter apenas símbolos terminais minúsculos.

b) O termo A à esquerda deve ser obrigatoriamente composto por uma única variável não-terminal isolada.

c) O lado direito da regra não pode conter, sob nenhuma hipótese, a palavra vazia (ϵ).

d) A substituição da variável depende diretamente dos símbolos que estão posicionados ao redor dela.

26. Quando uma determinada Gramática Livre de Contexto consegue gerar duas ou mais árvores sintáticas (árvores de derivação) distintas para uma mesmíssima cadeia válida, essa gramática é classificada tecnicamente como:

a) Linear.

b) Ambigua.

c) Enumerável.

d) Irrestrita.

27. A notação BNF (Backus-Naur Form), amplamente utilizada para definir a sintaxe oficial de linguagens de programação modernas, constitui um equivalente prático de engenharia para qual conceito teórico?

a) Autômatos Finitos Determinísticos.

b) Expressões Regulares de POSIX.

c) Gramáticas Livres de Contexto (GLC).

d) Máquinas de Turing Universais.

28. Durante o processo de parsing (Análise Sintática) em um compilador moderno, a estratégia de "Derivação à Esquerda" (Leftmost Derivation) dita que o algoritmo deve expandir sempre:

a) O símbolo terminal mais próximo do fim da fita.

b) O não-terminal (variável) posicionado mais à esquerda na sentença atual de reescrita.

c) Apenas as regras de produção que geram a palavra vazia (ϵ).

d) O símbolo que foi inserido por último na tabela de tokens.

Unidade 7: Autômatos de Pilha (AP)

29. O Autômato de Pilha (AP) supera a limitação de memória dos autômatos finitos ao incorporar uma memória auxiliar baseada na política de acesso do tipo:

a) LIFO (Last In, First Out).

b) FIFO (First In, First Out).

c) Acesso Aleatório Direto.

d) Indexação por Chave Semântica.

30. Na especificação formal da sétupla de um Autômato de Pilha, o símbolo especial Z_0 desempenha um papel operacional crítico, que consiste em:

a) Provocar uma parada por erro imprevisto de hardware sempre que atingido.

b) Representar o caractere em branco da fita de escrita aleatória.

c) Marcar o fundo da pilha para que a máquina identifique com segurança quando a memória auxiliar retornou ao seu estado vazio.

d) Atuar como o único estado de aceitação final do sistema de controle.

Unidade 7: Máquina de Turing (MT)

31. No modelo formal da Máquina de Turing, a paragem da computação e o veredito sobre a aceitação ou rejeição de uma cadeia de entrada funcionam de uma forma marcadamente diferente dos autômatos finitos e de pilha. Na Máquina de Turing, a execução é encerrada:

a) Apenas quando a cabeça de leitura atinge o último quadrado preenchido da fita e não há mais símbolos para processar.

b) Sempre que a fita de leitura é completamente limpa e reescrita com o caractere especial branco (␣).

c) Após a máquina realizar um número fixo de movimentos para a esquerda (L) correspondente ao tamanho inicial da string.

d) No exato momento em que o fluxo de controle transita para o estado q_{aceita} ou para o estado $q_{rejeita}$, interrompendo a máquina imediatamente.

32. Uma linguagem é classificada formalmente como "Decidível" (ou Recursiva) quando a Máquina de Turing associada a ela se comporta como um algoritmo total. Isto significa que a máquina:

a) Deve garantir a paragem tanto para as cadeias que pertencem à linguagem (aceitando-as) quanto para as que não pertencem (rejeitando-as).

b) Pode entrar em loop infinito para cadeias que não pertencem à linguagem, desde que pare para as cadeias válidas.

c) Não necessita de utilizar o símbolo branco (␣) nas suas células de memória da fita.

d) É capaz de simular simultaneamente qualquer outra Máquina de Turing a partir de uma Expressão Regular equivalente.

Bibliografia Básica

MENEZES, Paulo F. B., Linguagens Formais e Autômatos [recurso eletrônico, Minha Biblioteca]. Porto Alegre, Grupo A, 2011.

DIVERIO, Tiaraju A., Teoria da Computação: Máquinas universais e computabilidade [recurso eletrônico, Minha Biblioteca]. Porto Alegre, Grupo A, 2011.

SIPSER, Michael., Introdução à teoria da computação [recurso eletrônico, Minha Biblioteca]. São Paulo: Cengage Learning, 2007..

Bibliografia Complementar

ASCENCIO, Ana F. Gomes, Estruturas de Dados: algoritmos, análise da complexidade e implementações em Java e C/C++ [recurso eletrônico, Biblioteca Virtual]. São Paulo: Pearson Prentice Hall, 2010.

TOSCANI, Laira Vieira., Complexidade de Algoritmos [recurso eletrônico, Minha Biblioteca]. Porto Alegre, Bookman, 2012. 3a ed.

DOBRUSHKIN, Vladimir A., Métodos para Análise de Algoritmos [recurso eletrônico, Minha Biblioteca]. Rio de Janeiro: LTC, 2012.

AHO, Alfred V., LAM, Mônica S., SETHI, Ravi, ULLMAN, Jeffrey D., Compiladores: Princípios, Técnicas e Ferramentas [recurso eletrônico, Biblioteca Virtual]. São Paulo: Pearson Addison-Wesley, 2008.

STEIN, Clifford., Matemática discreta para ciência da computação [recurso eletrônico, Biblioteca Virtual]. São Paulo: Pearson Education do Brasil, 2013.

Periódicos Científicos

Advances in Software Engineering - ISSN 1687-8655
Computación y Sistemas - ISSN 1405-5546
Computational & Applied Mathematics - ISSN 1807-0302
Journal of the Brazilian Computer Society - ISSN 0104-6500