CENTRO UNIVERSITÁRIO
FMU
LEVEL UP

LINGUAGENS FORMAIS E AUTÔMATOS

https://portal.fmu.br

Unidade 1: Introdução e Fundamentos

I. Perguntas de Múltipla Escolha

1. **c) {Ø, {a}, {b}, {a, b}}**
   - *Explicação:* O Conjunto das Partes (2^n) contém todos os subconjuntos possíveis, incluindo o vazio e o próprio conjunto.

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

3. **c) A cadeia vazia (comprimento zero).**

4. **c) 1021**
   - *Explicação:* O símbolo '2' não pertence ao alfabeto Σ = {0, 1}.

5. **c) Infinito (se o alfabeto não for vazio).**
   - *Explicação:* O fechamento de Kleene permite cadeias de qualquer comprimento $n \geq 0$.

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

7. **b) Qualquer subconjunto de Σ*.**

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

9. **b) 7**
   - *Explicação:* O comprimento da concatenação é a soma dos comprimentos: $|uv| = |u| + |v|$.

10. **b) Um conjunto finito e não vazio de símbolos.**

11. **b) "cba"**

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

II. Perguntas Reflexivas

1. **Abstração:** A matemática de conjuntos fornece um **modelo formal** (como o AFD). Como esse modelo é definido por lógica e transições, e não por voltagem ou circuitos específicos, ele é universal. Um compilador traduz essa lógica para o código de máquina específico, mas a *regra gramatical* (o modelo formal) permanece idêntica, independente se o hardware é Intel, ARM ou AMD.

2. **Linguagem Natural vs. Formal:** Não usamos Português porque linguagens naturais são inerentemente **ambíguas**. Segundo Sipser, para a computação, a precisão é vital; um comando deve ter exatamente um significado. No Português, uma frase pode ter várias interpretações dependendo do contexto, o que causaria erros imprevisíveis em um processador.

III. Exercícios Práticos

1. **Operações de Conjuntos:**
   - **Σ^2:** {xx, xy, xz, yx, yy, yz, zx, zy, zz} (Total de $3^2 = 9$ cadeias).
   - **Tamanho de P(Σ):** Como $|\Sigma| = 3$, o tamanho do conjunto das partes é $2^3=8$.

2. **Concatenação e Potência (L^2):**
   - L = {01, 10}
   - L^2 = {0101, 0110, 1001, 1010}
   - *Dica:* Basta combinar cada elemento do primeiro conjunto com cada elemento do segundo.

3. Identificação de Padrões:
    - Para Σ = {a, b}, a linguagem L que começa e termina com a mesma letra é:
    - L = {a} ∪ {b} ∪ {awa | w ∈ Σ*} ∪ {bwb | w ∈ Σ*}

    Lê-se: A linguagem L é formada pelas cadeias unitárias 'a' e 'b', ou por qualquer cadeia que comece e termine com 'a', ou qualquer cadeia que comece e termine com 'b', onde o conteúdo central 'w' pode ser qualquer sequência possível do alfabeto.

Decomposição dos Símbolos (O que você está dizendo):

- **L =** "A linguagem L é definida como..."
- **{a} ∪ {b}**: "O conjunto contendo as letras individuais 'a' e 'b'..." (Importante: strings de comprimento 1 começam e terminam com a mesma letra).
- **∪ (União)**: "...em união com..." ou "...ou...".
- **{awa | ... }**: "O conjunto de cadeias no formato 'a', seguido de algo, terminado em 'a'..."
- **| (Tal que)**: "...tal que..." ou "...onde...".
- **w ∈ Σ *:** "...'w' é qualquer cadeia pertencente ao fechamento de Kleene do alfabeto." (Ou seja, 'w' pode ser vazio, ou qualquer combinação de letras).

Unidade 2: Autômatos Finitos Determinísticos (AFD)

I. Perguntas de Múltipla Escolha

1. b) Determinar o próximo estado com base no estado atual e no símbolo lido.
2. c) Exatamente um.
3. c) Ele é representado por dois círculos concêntricos e indica que a cadeia lida é válida.
4. b) A definição de AFD exige que todo estado tenha exatamente uma transição para cada símbolo do alfabeto.
    - *Nota:* Se não houver uma transição útil, usamos o "estado morto".
5. a) q0 deve ser um estado de aceitação ($q_0 \in F$).
6. b) A função δ de forma matricial.
7. b) Um estado de onde não é possível alcançar nenhum estado de aceitação.
8. b) Um subconjunto de Q.
9. b) Para um mesmo par (estado, símbolo), o destino seja sempre único e previsível.
10. b) Reconhecimento de palavras reservadas em um compilador (Análise Léxica).
11. b) 4 (2 estados X 2 símbolos).
12. b) Regulares.

II. Perguntas Reflexivas

1. **Limitação de Memória:** Como o AFD tem memória finita (apenas os estados) e não pode retroceder, ele não consegue "contar" elementos de forma ilimitada. Para balancear parênteses, você precisaria lembrar quantos abriu para saber quantos deve fechar. Um AFD não consegue fazer isso para uma quantidade infinita de parênteses (como ((((...)))), pois exigiria infinitos estados. Para isso, precisaríamos de um **Autômato com Pilha**.

2. **Modelagem Real:** Um AFD é baseado em um modelo matemático formal e exaustivo. Enquanto if/else soltos podem conter brechas lógicas ou caminhos não tratados, o AFD força o desenvolvedor a prever o que acontece em cada estado para cada símbolo possível. Isso torna o sistema **previsível, fácil de testar e livre de efeitos colaterais**.

III. Exercícios Práticos

1. **Desenho Técnico (Sufixo "ba"):**
    - **Q = {q_0, q_1, q_2}**
        - q_0: Estado inicial (não leu "b" nem "a" como parte do sufixo final)
        - q_1: Leu o caractere "b". Está em alerta, pois o próximo "a" pode completar o sufixo.
        - q_2: Leu a sequência "ba". Estado de Aceitação.
    - **F = {q_2}**
    - **Matriz de Transição:**

Estado Atual	Entrada 'a'	Entrada 'b'
→q0	q0	q1
q1	*q2	q1
*q2	q0	q1

Explicação da Lógica de Transição:

- **No estado q_0:** Se ler 'a', continua em q_0 (não ajuda a formar "ba"). Se ler 'b', avança para q_1.
- **No estado q_1:** Se ler outro 'b', permanece em q_1 (ainda estamos à espera do 'a'). Se ler 'a', formou "ba" e vai para o estado de aceitação q_2.
- **No estado q_2 (Pós-Aceitação):**
    - Se ler 'a', a sequência vira "...baa", que não termina em "ba". Volta ao início (q_0).
    - Se ler 'b', a sequência vira "...bab". O último caractere é 'b', o que pode ser o início de um novo "ba". Portanto, volta para q_1.

2. **Análise de Linguagem:**
    - O padrão aceito é: **Cadeias de comprimento par**.
    - *Justificativa:* Como qualquer símbolo alterna o estado e a máquina começa em um estado de aceitação (q_0), após 1 símbolo ela estará em q_1 (ímpar/rejeita), após 2 símbolos voltará a q_0 (par/aceita), e assim sucessivamente.

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

I. Questões de Múltipla Escolha

1. b) O não-determinismo permite que, a partir de um estado e um símbolo, o autômato alcance um conjunto de estados possíveis.
2. c) Uma transição que ocorre sem o consumo de qualquer símbolo da fita de entrada.
3. b) O resultado é um elemento do conjunto das partes de Q, ou seja, um subconjunto de estados.
4. c) 2^n (A chamada "explosão exponencial de estados").
5. a) Pelo menos um dos caminhos possíveis terminar em um estado de aceitação.
6. d) Facilitar a união de dois ou mais autômatos em um único sistema.
7. a) Ambos reconhecem exatamente a mesma classe de linguagens: as Linguagens Regulares.
8. b) Aquele caminho específico da computação é encerrado ("morre").
9. d) Todos os novos estados (subconjuntos) que contiverem ao menos um estado final do AFN original.
10. a) O conjunto de todos os estados alcançáveis a partir de q seguindo apenas transições- ε.

II. Perguntas Reflexivas

1. **Paralelismo Teórico:** A analogia do paralelismo simplifica a modelagem porque o projetista não precisa prever todas as combinações de erro ou retornos imediatos. No AFN, você desenha o "caminho do sucesso". Se a máquina encontra uma bifurcação, ela "clona" seu estado. Isso é muito mais intuitivo do que o AFD, que exige um pensamento linear onde cada símbolo deve ter um destino único e exclusivo, muitas vezes resultando em diagramas poluídos por setas de correção.

2. **Custo de Conversão:** Preferimos o AFN na modelagem por **produtividade e clareza**. É mais fácil para um humano ler e manter um AFN. No desenvolvimento de software (como motores de Regex), o computador faz a conversão para AFD apenas uma vez (tempo de compilação). Embora o AFD gaste mais memória, a sua execução é O(n), ou seja, extremamente rápida. Trocamos memória por velocidade de execução e facilidade de design humano.

III. Exercícios Práticos

1. Conversão de Modelo (AFN para AFD)

Dado o AFN onde $\delta(q_0, a)=\{q_0, q_1\}$ e $\delta(q_0, b) = \{q_0\}$:

1º Passo: Começar pelo Início

No AFN, ao ler 'a', você está em dois lugares ao mesmo tempo: $\{q_0, q_1\}$. No AFD, não permitimos isso. Então, criamos um **único estado novo** que chamamos de $\{q_0, q_1\}$. É como se fundíssemos os dois estados em uma "bolha" (cluster).

2º Passo: Analisar o Novo Estado Criado

Agora temos que analisar o que acontece quando estamos na "bolha" $\{q_0, q_1\}$. Para descobrir o destino, você olha o que cada um faz separadamente e **une** os resultados.

Testando as entradas para $\{q_0, q_1\}$:

1. **Se ler 'a':**
    - q_0 vai para $\{q_0, q_1\}$
    - q_1 não tem transição (vazio $\emptyset$)
    - **União:** $\{q_0, q_1\} \cup \emptyset = \{q_0, q_1\}$.
    - *Resultado:* Um laço voltando para ele mesmo.
2. **Se ler 'b':**
    - q_0 vai para $\{q_0\}$
    - q_1 não tem transição (vazio $\emptyset$)
    - **União:** $\{q_0\} \cup \emptyset = q_0\}$.
    - *Resultado:* Uma seta voltando para o estado inicial $\{q_0\}$.

Tabela de Transição do AFD:

Estado (Subconjunto)	Entrada 'a'	Entrada 'b'
→A:$\{q_0\}$	$\{q_0, q_1\}$ (Estado B)	$\{q_0\}$ (Estado A)
B:$\{q_0, q_1\}$	$\{q_0, q_1\}$ (Estado B)	$\{q_0\}$ (Estado A)

Por que isso é um AFD agora?

Observe a tabela acima. Diferente do enunciado original:

- Cada célula contém **apenas um** destino (mesmo que o nome do destino tenha dois números, ele é tratado como uma unidade/estado único no AFD).
- Não há dúvidas de para onde ir.

2. Modelagem AFN- ϵ (União de L_1 e L_2)

Para aceitar cadeias que começam com '0' **OU** terminam com '1':

- **Estado Inicial:** q_{start}
- **Transições- ϵ:**
  - $\delta(q_{start}\}, \epsilon \rightarrow \{q_{L1}, q_{L2}\}$
- **Ramo L_1 (Começa com '0'):**
  - $\delta(q_{L1}, 0) = q_{f1}$ (Estado final)
  - $\delta(q_{f1}, 0|1) = q_{f1}$ (Lê qualquer coisa depois)
- **Ramo L_2 (Termina com '1'):**
  - $\delta(q_{L2}, 0|1) = q_{L2}$ (Lê qualquer coisa no início)
  - $\delta(q_{L2}, 1) = q_{f2}$ (Estado final se terminar em 1)

Unidade 4: Expressões Regulares (ER) e Análise Léxica

I. Perguntas de Múltipla Escolha

1. **b) Permitir zero ou mais repetições do padrão anterior.**
2. **b) Uma unidade básica com significado léxico (ex: um número ou operador).**
3. **c) Estrela, Concatenação, União.** (Lembre-se: Parênteses > Fecho > Concatenação > União).
4. **c) Strings sobre {a, b} que terminam obrigatoriamente em "abb".**
5. **b) Que o elemento precedente é opcional (0 ou 1 ocorrência).**
6. **b) Linguagens Regulares.**
7. **b) 1(0+1)*** (O '1' inicial é fixo, seguido de qualquer combinação de 0s e 1s).
8. **b) Agrupar caracteres em tokens e descartar espaços/comentários.**
9. **b) Números decimais (inteiros ou com ponto).**
10. **b) ERs e Autômatos Finitos são equivalentes em poder de expressão.**
11. **b) a aceita a cadeia vazia; a+ exige pelo menos um 'a'.***
12. **b) É gerado um erro léxico (caractere inválido).**

II. Perguntas Reflexivas

1. **Eficiência:** Usar ERs na fase léxica é mais eficiente porque elas são processadas por **Autômatos Finitos (AFD)**, que possuem complexidade linear O(n) e não exigem memória extra (pilha). Gramáticas Livres de Contexto exigem **Autômatos com Pilha**, que são computacionalmente mais caros. Separar o léxico (simples) do sintático (complexo) otimiza a velocidade do compilador.

2. **Segurança:** ERs mal escritas (muito permissivas) podem permitir caracteres especiais como ' (aspas simples), -- (comentários SQL) ou <script> (tags HTML). Se a ER não barrar esses metacaracteres na entrada do formulário, eles podem ser executados diretamente no banco de dados ou no navegador de outros usuários, causando vazamento de dados ou roubo de sessões.

III. Exercícios Práticos

1. **Criação de ER (Identificadores):**
   - **ER:** [a-zA-Z][a-zA-Z0-9]{2,}
   - *Explicação:* Começa com uma letra [a-zA-Z], seguida por pelo menos mais dois caracteres alfanuméricos {2,} para totalizar o mínimo de 3 exigido.

2. **Conversão Visual (AFN para (0+1)*11):**
   - **q_0:** Estado inicial com um laço (self-loop) de 0, 1.
   - **Transição de q_0 para q_1:** Lê o primeiro 1. (Aqui ocorre o não-determinismo).
   - **Transição de q_1 para q_2:** Lê o segundo 1.
   - **q_2:** Estado Final (Aceitação).

Estado Atual	Entrada '0'	Entrada '1'	Descrição do Papel do Estado
$\rightarrow q_0$	$\{q_0\}$	$\{q_0, q_1\}$	Estado Inicial: Aceita qualquer bit ou inicia a sequência final.
q_1	$\emptyset$	$\{q_2\}$	Estado Intermediário: Leu o penúltimo '1'.
$*q_2$	$\emptyset$	$\emptyset$	Estado Final: Leu o último '1' (Aceitação).

3. **Análise de Padrão:**
   - contato@fmu.com: **ACEITA**. Segue o padrão de letras minúsculas, arroba, provedor e termina em .com.
   - admin123@google.com: **REJEITADA**. A ER [a-z]+ permite apenas letras, e a string contém números (123).
   - suporte@empresa.org: **REJEITADA**. A ER exige especificamente o final .com, e esta termina em .org.

Unidade 5: Gramáticas Regulares e a Hierarquia de Chomsky

I. Perguntas de Múltipla Escolha

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

- **Justificativa:** Na definição formal da quádrupla **G = (V, Σ, P, S)**, o símbolo **S** é o axioma ou variável inicial. Toda geração de cadeia deve obrigatoriamente começar por ele.

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

- **Justificativa:** Gramáticas Lineares à Direita possuem produções do tipo A → aB ou A → a. Se o não-terminal estivesse à esquerda (A → Ba), seria Linear à Esquerda.

3. Resposta: d) Tipo 3

- **Justificativa:** Na Hierarquia de Chomsky, as Linguagens Regulares são as mais restritas e ocupam o nível mais baixo (Tipo 3).

4. Resposta: b) Que uma linguagem não é regular.

- **Justificativa:** O Pumping Lemma é uma ferramenta de prova por contradição. Se uma linguagem não satisfaz as condições do lema, podemos afirmar com certeza que ela **não** é regular.

5. Resposta: c) Terminais (Σ)

- **Justificativa:** Os terminais são os símbolos que não podem mais ser substituídos. Eles formam a cadeia final que será "lida" pelo usuário ou pela máquina.

6. Resposta: c) Autômato de Pilha.

- **Justificativa:** As linguagens de Tipo 2 (Livres de Contexto) exigem uma memória do tipo LIFO (Pilha) para reconhecer estruturas como aninhamentos e simetrias.

7. Resposta: a) Regular (Linear à Direita).

- **Justificativa:** A regra segue o padrão V → TV ou V → T, onde um não-terminal gera um terminal seguido (ou não) de outro não-terminal à direita.

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

- **Justificativa:** Autômatos Finitos não possuem memória de contagem. Para garantir que o número de 'a's seja exatamente igual ao de 'b's, a máquina precisaria de estados infinitos ou uma pilha.

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

- **Justificativa:** Derivar significa aplicar as regras de produção sucessivamente a partir de S até que não restem mais variáveis (não-terminais) na cadeia.

10. Resposta: c) Linguagens Recursivamente Enumeráveis.

- **Justificativa:** O Tipo 0 é o nível mais amplo da hierarquia, correspondendo às linguagens reconhecidas por Máquinas de Turing.

11. Resposta: a) Tipo 3 ⊂ Tipo 2 ⊂ Tipo 1 ⊂ Tipo 0.

- **Justificativa:** A hierarquia é inclusiva: toda linguagem regular é livre de contexto, toda livre de contexto é sensível ao contexto, e assim por diante.

12. Resposta: b) Possui uma Gramática Regular equivalente.

- **Justificativa:** Pelo Teorema de Kleene, as Expressões Regulares, os Autômatos Finitos e as Gramáticas Regulares são três formas diferentes de descrever a exata mesma classe de linguagens (Tipo 3).

II. Perguntas Reflexivas

1. A Natureza da Linguagem:

Apesar de limitadas, as Linguagens Regulares são utilizadas na análise léxica porque são extremamente eficientes em termos de processamento e memória (tempo linear O(n)). Na fase léxica, o objetivo não é entender a estrutura lógica do código (como o balanceamento de parênteses), mas apenas identificar "unidades de vocabulário" (tokens), como palavras-chave, números e operadores. Para essa tarefa de reconhecimento de padrões simples, a performance das ERs é imbatível.

2. Abstração vs. Realidade:

A Hierarquia de Chomsky funciona como um guia de economia de recursos. Se um engenheiro precisa validar um campo de CPF ou E-mail, a teoria mostra que uma **Regex (Tipo 3)** é suficiente e mais rápida. Tentar usar um **Parser completo (Tipo 2)** para validar um E-mail seria um desperdício de processamento. A hierarquia ajuda a escolher a ferramenta com a "menor complexidade necessária" para garantir a segurança e a performance do software.

III. Exercícios Práticos

1. Construção de Gramática (Ímpar de '1's):

Seja G = ({S, A}, {0, 1}, P, S) com P:

- S → 0S | 1A (Enquanto lê 0, continua "par". Se lê 1, torna-se "ímpar" e vai para A).
- A → 0A | 1S | ϵ (Enquanto em A, se ler 1 volta para S/par. O ϵ em A permite terminar a cadeia apenas quando ela for ímpar).

2. Classificação Técnica (S → aSb | ϵ):

Esta é uma **Gramática Livre de Contexto (Tipo 2)**.

- **Justificativa:** Ela gera a linguagem $\{a^n b^n \mid n \geq 0\}$. Note que a variável **S** está no meio de dois terminais (aSb), o que quebra a regra da Gramática Regular (onde o não-terminal deve estar apenas em uma das extremidades). Como o lado esquerdo de todas as produções tem apenas uma variável isolada, ela é Livre de Contexto.

3. Aplicação do Pumping Lemma (Parênteses):

Um autômato finito falha porque ele possui um número fixo e finito de estados (sua "memória"). Para validar se os parênteses estão corretos, o autômato precisaria "contar" quantos abriu para garantir que fechará a mesma quantidade. Como a profundidade do aninhamento de parênteses em um código pode ser teoricamente infinita, o autômato eventualmente ficaria sem estados para continuar a contagem, "esquecendo" quantos parênteses ainda faltam fechar.

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

I. Perguntas de Múltipla Escolha

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

- **Resposta correta: b) A incapacidade de lidar com recursividade e aninhamento infinito.**
- **Justificativa:** As linguagens regulares não conseguem fazer contagens emparelhadas ou validar estruturas com memórias de aninhamentos (como parênteses e chaves abertas que dependem de fechamento correspondente). As LLC superam isso através de sua estrutura recursiva e do reconhecimento por meio de pilhas.

2. Na regra de produção $A \rightarrow \alpha$ o que define uma GLC?

- **Resposta correta: b) A deve ser uma única variável não-terminal.**
- **Justificativa:** O termo "Livre de Contexto" significa exatamente que o lado esquerdo da produção deve conter apenas **uma** variável isolada ($A \in V$). Isso garante que a substituição de A por α ocorra independentemente dos símbolos que estejam ao seu redor.

3. O que é uma gramática ambígua?

- **Resposta correta: b) Uma gramática que gera mais de uma árvore sintática para a mesma string.**
- **Justificativa:** A ambiguidade formal ocorre quando uma única cadeia válida possui duas ou mais estruturas de derivação (árvores sintáticas) distintas, gerando diferentes interpretações para o mesmo código.

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

- **Resposta correta: b) Definir a sintaxe de linguagens de programação.**
- **Justificativa:** A BNF é uma meta-linguagem equivalente às GLCs, criada especificamente para descrever as regras sintáticas formais de linguagens de programação (como a especificação do bloco `if`, assinaturas de funções, etc.).

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

- **Resposta correta: c) $L = \{ a^n b^n \mid n \geq 0 \}$**
- **Justificativa:** Esta linguagem exige o emparelhamento exato (contagem) do número de símbolos **a** com o número de símbolos **b** o que viola o Teorema do Bombeamento para linguagens regulares, mas é facilmente descrita por uma GLC.

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

- Resposta correta: b) A estrutura de derivação de uma cadeia a partir do símbolo inicial.
- Justificativa: A árvore sintática (ou *parse tree*) é a representação gráfica e hierárquica do processo de derivação, mapeando como o axioma inicial se ramificou através das regras até chegar aos nós folha (terminais).

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

- Resposta correta: b) Que as regras de substituição de uma variável não dependem dos símbolos vizinhos.
- Justificativa: Significa que o não-terminal pode ser reescrito livremente, sem requerer a presença de um contexto (símbolos antes ou depois) no lado esquerdo da regra.

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

- Resposta correta: a) O uso de variáveis no lado direito das produções.
- Justificativa: A recursividade se manifesta quando uma variável chama a si mesma (direta ou indiretamente) no corpo da regra, como em A → aA.

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

- Resposta correta: b) Análise Sintática (Parsing).
- Justificativa: A análise léxica trabalha com Expressões Regulares (Tipo 3). É na análise sintática que o *parser* constrói a árvore hierárquica baseando-se nas regras de uma GLC (Tipo 2).

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

- Resposta correta: b) Autômato de Pilha.
- Justificativa: O Autômato de Pilha (AP) adiciona uma memória auxiliar padrão LIFO (pilha) ao controle finito, garantindo o poder necessário para reconhecer Linguagens Livres de Contexto.

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

- Resposta correta: b) aabb
- Justificativa: Se aplicarmos a primeira regra, temos aSb. Substituindo o S central pela segunda opção (ab), obtemos a cadeia terminal a(ab)b = aabb.

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

- Resposta correta: b) Expandir sempre o não-terminal mais à esquerda na sentença atual.
- Justificativa: Na derivação à esquerda (*leftmost derivation*), o algoritmo do compilador varre a sentença atual e sempre seleciona a variável mais à esquerda para aplicar a próxima substituição.

II. Perguntas Reflexivas

1. Ambiguidade na Vida Real:

Se a gramática de uma linguagem comercial como o Java fosse ambígua, as consequências seriam desastrosas. Um código que calcula juros bancários ou saldos de transações poderia gerar árvores sintáticas diferentes em compiladores distintos (ou até em momentos diferentes). O compilador poderia interpretar a ordem de precedência de uma operação matemática de duas formas válidas pela gramática, mas com resultados financeiros completamente diferentes (ex: `saldo + deposito * taxa` interpretado primeiro como soma ou primeiro como multiplicação).

Para garantir que o compilador interprete o código exatamente como o programador planejou, as especificações das linguagens utilizam gramáticas estritamente **não-ambíguas**, estabelecendo fatores rígidos de precedência e associatividade diretamente nas regras de produção.

2. Evolução:

Não utilizamos GLC para todas as etapas porque existe um compromisso entre o **poder de expressão** e a **complexidade computacional**. As Expressões Regulares (Tipo 3) rodam em tempo linear estável e sem consumo de memória complexa (via AFD). Separar a análise léxica (ER) da sintática (GLC) traz vantagens fundamentais de engenharia de software:

- **Desempenho:** O compilador consegue descartar espaços em branco, comentários e validar palavras-chave muito mais rápido usando ER na primeira fase, enviando um fluxo limpo e reduzido de tokens para o parser.
- **Modularidade e Manutenção:** Separar o vocabulário (léxico) da estrutura de blocos (sintático) torna o design do compilador limpo, facilitando a portabilidade e permitindo otimizações isoladas em cada módulo.

III. Exercícios Práticos

1. Construção de GLC (L = \{ $a^n b^{2n}$ | n ≥ 1 }):

Para garantir que para cada caractere **a** inserido à esquerda sejam gerados exatamente dois caracteres **b** à direita, definimos a gramática **G = ({S}, {a, b}, P, S)** com as produções P: **S → aSbb | abb**

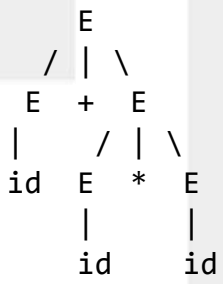
- **Explicação:** A regra **aSbb** garante o crescimento proporcional da cadeia (proporção 1 para 2). A regra **abb** funciona como a condição de parada mínima (n=1), encerrando a recursão de forma limpa.

2. Árvore Sintática (Ambiguidade de E → E + E | E * E | id):

A expressão `id + id * id` possui duas árvores sintáticas devido à falta de definição de precedência nesta gramática:

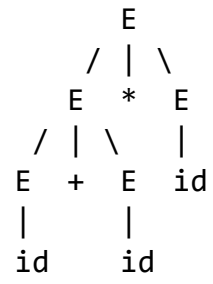
- **Árvore 1 (Multiplicação com maior precedência - Interpretação Padrão):**

O **E** inicial se divide em **E + E**. O segundo **E** se expande em **E * E**.



- **Árvore 2 (Soma com maior precedência - Interpretação Errada):**

O **E** inicial se divide em **E * E**. O primeiro **E** se expande em **E + E**.



3. Modelagem de Bloco:

Para representar um escopo estruturado (como blocos de funções ou instruções em linguagens derivadas de C/Java), podemos criar as seguintes regras de produção abstratas:

⟨ Bloco ⟩ → { ⟨ Comandos ⟩ }

⟨ Comandos ⟩ → Comando ⟩ ⟨ Comandos ⟩ | ⟨ Bloco ⟩ ⟨ Comandos ⟩ | ε

⟨ Comando ⟩ → id | atribuicao | chamada_funcao

Explicação: A regra obriga a abertura e o fechamento por chaves. A variável ⟨ Comandos⟩ permite uma sequência de instruções consecutivas e inclui o próprio ⟨ Bloco⟩ na sua definição, aceitando assim o aninhamento perfeito e infinito de blocos internos (ex: laços `if` dentro de outros laços `if`).

Unidade 7: Autômatos de Pilha (AP)

I. Perguntas de Múltipla Escolha

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

- **Resposta correta: b) A presença de uma memória auxiliar do tipo pilha.**
- **Justificativa:** O Autômato Finito possui apenas um controle interno estático (seus estados). O Autômato de Pilha (AP) expande esse modelo ao introduzir uma memória auxiliar irrestrita em tamanho, organizada de forma linear sob as regras de uma pilha.

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

- **Resposta correta: c) LIFO (Last In, First Out).**
- **Justificativa:** A estrutura de dados tipo pilha opera sob o princípio de que o último elemento a entrar na estrutura será, obrigatoriamente, o primeiro a ser removido (último a entrar, primeiro a sair).

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

- **Resposta correta: a) O alfabeto de símbolos que podem ser armazenados na pilha.**
- **Justificativa:** Enquanto **Σ** denota o alfabeto de entrada (símbolos lidos da fita), **Γ** representa o alfabeto da pilha, contendo os delimitadores e variáveis aceitos dentro da memória auxiliar.

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

- **Resposta correta: b) Para marcar o fundo da pilha e permitir que a máquina saiba quando ela está vazia.**
- **Justificativa:** Como a leitura física de uma pilha vazia causaria um erro estrutural, o marcador de fundo Z_0 (ou $ em algumas literaturas) é inserido no início da computação para que a máquina possa testar com segurança se retornou à base inicial.

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

- **Resposta correta: d) Não, pois uma única pilha só consegue comparar dois elementos por vez (Tipo 2).**
- **Justificativa:** Para validar se a quantidade de três variáveis é idêntica, a máquina precisaria empilhar os **a's**, desempilhar comparando com os **b's** (o que esvaziaria a pilha) e ficaria sem histórico para validar os **c's**. Essa linguagem pertence ao Tipo 1 (Sensível ao Contexto).

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

- **Resposta correta: b) O símbolo do topo da pilha é removido.**
- **Justificativa:** Na terminologia de pilhas, *Push* representa a operação de inserção/empilhamento no topo, enquanto *Pop* representa a remoção do elemento que se encontra no topo.

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

- **Resposta correta: a) Linguagens Livres de Contexto (Tipo 2).**
- **Justificativa:** Existe uma equivalência exata provada na Teoria da Computação: a classe de linguagens geradas por Gramáticas Livres de Contexto (GLC) é a mesma classe reconhecida por Autômatos de Pilha.

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

- **Resposta correta: b) Análise sintática de códigos-fonte em compiladores.**
- **Justificativa:** O coração de um analisador sintático (*parser*) é um autômato de pilha, responsável por verificar se estruturas recursivas do código-fonte (como escopos de blocos, expressões aritméticas e chamadas de métodos) obedecem às regras da linguagem.

9. Na função de transição $\delta(q, a, X) = (p,\Gamma)$, o que o 'X' representa?

- **Resposta correta: c) O símbolo que está atualmente no topo da pilha.**
- **Justificativa:** A função de transição de um AP necessita de três parâmetros para tomar uma decisão: o estado atual **(q)**, o símbolo lido da fita **(a)** e o símbolo presente no topo atual da pilha **(X)**.

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

- **Resposta correta: b) A pilha permite "empilhar" o estado de uma função e voltar para ele após a execução.**
- **Justificativa:** Na arquitetura de computadores, a pilha de execução (*call stack*) salva o endereço de retorno e as variáveis locais da função atual antes de mergulhar em uma chamada recursiva, permitindo o retorno correto quando a ramificação for concluída.

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

- **Resposta correta: a) A máquina para em qualquer estado e a pilha não tem mais nenhum símbolo além de Z_0 (ou está completamente vazia).**
- **Justificativa:** Existem dois critérios de aceitação equivalentes para APs: aceitação por estado final (parar em um estado de F) ou por pilha vazia (consumir toda a entrada e terminar sem símbolos na pilha, consumindo inclusive o Z_0).

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

- **Resposta correta: b) O APN é estritamente mais poderoso que o APD.**
- **Justificativa:** Diferente dos Autômatos Finitos (onde AFD e AFN são equivalentes), nos Autômatos de Pilha o não-determinismo estende o poder computacional. Existem linguagens livres de contexto (como palíndromos de comprimento par) que só podem ser reconhecidas por um AP Não-Determinístico (APN).

II. Perguntas Reflexivas

1. Hierarquia de Memória:

A adição de uma Pilha rompe a barreira dos estados fixos dos autômatos finitos, conferindo ao sistema uma capacidade de memória dinâmica teoricamente infinita. Isso muda drasticamente o cenário computacional pois permite que a máquina resolva problemas de correlação e emparelhamento temporal distante (saber que um bloco aberto na linha 1 deve fechar na linha 100).

A estrutura de **Pilha** é ideal e superior à **Fila** para linguagens de programação porque a sintaxe das linguagens é hierárquica e aninhada, e não sequencial linear. Quando abrimos uma função e, dentro dela, um laço `if`, a primeira estrutura que deve ser fechada é a mais interna (o `if`). Esse comportamento de "o último que abre é o primeiro que fecha" espelha perfeitamente a política LIFO de uma Pilha. Uma fila (FIFO) fecharia os blocos na ordem errada (de fora para dentro), quebrando o escopo do código.

2. O Limite:

O Autômato de Pilha falha ao tentar computar três contagens idênticas $\{a^n b^n c^n\}$ devido à restrição física de acesso unidimensional da pilha. Para comparar a igualdade de duas variáveis (como $a^n b^n$), usamos a pilha como um contador de balanço: empilhamos cada símbolo **a** e usamos os símbolos **b** para anular (dar *pop*) em cada a.

O problema limite ocorre quando o bloco de **b's** termina: a pilha foi completamente esvaziada para provar que $n_a = n_b$. Quando a máquina começa a ler os símbolos **c**, ela não possui mais nenhuma informação armazenada para validar se a quantidade de **c's** coincide com o valor de **n**. Para resolver esse problema, seriam necessárias duas pilhas independentes ou uma fita de acesso bidirecional (Máquina de Turing).

III. Exercícios Práticos

1. Simulação de Pilha (Cadeia (( ) ( ))):

Considerando a pilha inicializada com o marcador de fundo Z_0:

1. **Início:** Pilha = [Z_0]
2. **Lê ((1º caractere):** Realiza *Push(X)*. Pilha = [X, Z_0]
3. **Lê ((2º caractere):** Realiza *Push(X)*. Pilha = [X, X, Z_0]
4. **Lê) (3º caractere):** Realiza *Pop(X)*. Pilha = [X, Z_0]
5. **Lê ((4º caractere):** Realiza *Push(X)*. Pilha = [X, X,Z_0]
6. **Lê) (5º caractere):** Realiza *Pop(X)*. Pilha = [X, Z_0]
7. **Lê) (6º caractere):** Realiza *Pop(X)*. Pilha = [Z_0]
8. **Fim da fita:** A fita terminou e o topo da pilha contém apenas o marcador Z_0. **Veredito: Cadeia Aceita.**

2. Modelagem de AP (Palíndromos Pares sobre {a, b}):

Um Autômato de Pilha Não-Determinístico (APN) estruturado para aceitar palíndromos pares operaria com 3 estados principais (q_0, q_1, q_2) seguindo esta lógica de transições:

- **Estado q_0 (Fase de Empilhamento / Primeira metade):** Enquanto a máquina estiver em q_0, para cada caractere lido da fita (**a** ou **b**), ela realiza um *Push* do mesmo caractere na pilha.
  - $\delta(q_0, a, Z_0) = (q_0, aZ_0)$
  - $\delta(q_0, b, Z_0) = (q_0, bZ_0)$
  - $\delta(q_0, a, a) = (q_0, aa)$ | $\delta(q_0, b, a) = (q_0, ba)$ (e vice-versa)
- **O Salto Não-Determinístico (ϵ-transição):** A qualquer momento, sem ler nada da fita, a máquina pode adivinhar que chegou exatamente ao meio do palíndromo e saltar para o estado q_1. Se for o meio correto, os caminhos paralelos computacionais sobreviventes farão o casamento perfeito.
  - $\delta(q_0, \epsilon, X) = (q_1, X)$ (para qualquer topo)
- **Estado q_1 (Fase de Confronto / Segunda metade):** Em q_1, a máquina tenta casar o caractere atual da fita com o caractere do topo da pilha. Se forem iguais, ela realiza um *Pop*.
  - $\delta(q_1, a, a) = (q_1, \epsilon)$ (Lê 'a', topo 'a' → remove)
  - $\delta(q_1, b, b) = (q_1, \epsilon)$ (Lê 'b', topo 'b' → remove)
- **Aceitação (q_2):** Se a fita terminar e o topo da pilha revelar o marcador inicial Z_0, a máquina migra para o estado final q_2.
  - $\delta(q_1, \epsilon, Z_0) = (q_2, Z_0)$

3. Análise de Erro (Cadeia `())`):

Ao processar a cadeia `())`, a execução da máquina performará os seguintes passos:

1. **Lê `(`:** Executa um *Push*, adicionando um marcador à pilha. Pilha = [X, Z_0].
2. **Lê `)`:** Encontra um caractere correspondente no topo. Executa um *Pop*. Pilha = [Z_0].
3. **Lê `)`:** A fita de entrada solicita uma nova validação, porém, ao inspecionar o topo da pilha, o autômato encontra o marcador de fundo Z_0.

Como não existe nenhuma regra válida de transição que permita remover elementos abaixo do fundo da pilha ou realizar um casamento de fechamento sem um caractere de abertura correspondente, o autômato entra em **estado de pane (indefinição de transição)**. A fita de entrada é interrompida abruptamente antes da finalização ideal e a pilha fica inconsistente. **Veredito final: Cadeia Rejeitada (Erro de Sintaxe).**

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

I. Perguntas de Múltipla Escolha (com Justificativa)

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

- **Resposta correta: b) Uma fita infinita onde se pode ler e escrever em qualquer direção.**
- **Justificativa:** O autômato de pilha está restrito ao acesso linear do tipo LIFO (topo da pilha). A Máquina de Turing introduz uma fita de armazenamento infinito com acesso bidirecional aleatório, permitindo ler, escrever e sobrescrever símbolos em qualquer célula.

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

- **Resposta correta: a) Um espaço vazio da memória ainda não utilizado.**
- **Justificativa:** A fita da MT é teoricamente infinita. Inicialmente, ela contém a palavra de entrada e o restante de toda a fita é preenchido com o símbolo especial branco (␣), representando células de memória limpas e disponíveis para escrita.

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

- **Resposta correta: c) Left (Esquerda) e Right (Direita), as direções de movimento da cabeça.**
- **Justificativa:** A cada passo computacional, após ler e escrever um símbolo, a cabeça de leitura/escrita da Máquina de Turing deve obrigatoriamente deslocar-se uma posição para a esquerda L) ou para a direita (R).

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

- **Resposta correta: b) Simular qualquer outra Máquina de Turing a partir de sua descrição.**
- **Justificativa:** A UTM introduziu o conceito de "programa armazenado". Ela recebe na fita a descrição de uma MT específica (M) codificada juntamente com uma cadeia de entrada (w), operando exatamente como os computadores modernos (CPUs executando diferentes softwares).

5. Segundo a Tese de Church-Turing:

- **Resposta correta: d) Qualquer algoritmo computável pode ser executado por uma Máquina de Turing.**
- **Justificativa:** A tese estabelece que a noção intuitiva de "algoritmo" ou "processo eficaz" coincide exatamente com o que pode ser resolvido por uma Máquina de

Turing. Nenhum modelo de computação formal possui poder expressivo superior.

6. Uma linguagem é considerada "Decidível" quando:

- **Resposta correta: b) A máquina sempre para e decide entre aceitação ou rejeição para qualquer entrada.**
- **Justificativa:** Para ser decidível (ou recursiva), a máquina associada deve funcionar como um algoritmo total: para qualquer cadeia de entrada válida ou inválida, ela deve garantir a paragem em q_{aceita} ou $q_{rejeita}$ sem nunca entrar em loop infinito.

7. O "Problema da Parada" provou que:

- **Resposta correta: d) Existem problemas que são logicamente impossíveis de serem resolvidos por algoritmos.**
- **Justificativa:** Alan Turing provou, por meio da técnica da diagonalização, que é impossível construir um algoritmo genérico capaz de prever se qualquer outro programa vai parar ou travar em loop. Isso estabeleceu o primeiro limite matemático absoluto para os softwares.

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

- **Resposta correta: b) Γ**
- **Justificativa:** Na definição formal de 7 elementos da MT, **Σ** é o alfabeto de entrada (apenas leitura), enquanto **Γ** é o alfabeto da fita, que engloba todos os símbolos de **Σ**, o símbolo branco ⊔ , e quaisquer outros caracteres auxiliares de escrita que a máquina utilize para marcação.

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

- **Resposta correta: c) Tipo 0 (Recursivamente Enumeráveis).**
- **Justificativa:** O Tipo 0 ocupa o topo e a camada mais externa da hierarquia. Representa o limite da computabilidade, contendo todas as linguagens cujas cadeias podem ser reconhecidas ou geradas por uma Máquina de Turing.

10. Diferente dos autômatos anteriores, quando a MT atinge o estado qaceita, ela:

- **Resposta correta: b) Para imediatamente e termina a execução.**
- **Justificativa:** Nos autômatos finitos e de pilha, a aceitação depende do fim da fita. Na Máquina de Turing, a paragem é imediata: no momento em que o fluxo transita para q_{aceita} ou $q_{rejeita}$, a computação encerra-se na hora, independentemente de haver mais símbolos na fita.

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

- Resposta correta: a) Quando a máquina nunca atinge um estado de parada (q_{aceita} ou $q_{rejeita}$).
- Justificativa: Se uma máquina entra numa sequência infinita de transições sem alcançar um dos seus estados terminais de paragem, dizemos que ela está em loop. Esse é o comportamento das linguagens que são apenas semidecidíveis (recursivamente enumeráveis).

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

- Resposta correta: a) Serve como modelo teórico para a CPU e a memória RAM.
- Justificativa: O funcionamento mecânico da MT (controle interno decodificando instruções e alterando células de memória na fita) estabeleceu a base matemática para a Arquitetura de Von Neumann, utilizada em quase todas as CPUs e memórias RAM modernas.

II. Perguntas Reflexivas

1. O Limite do Pensamento:

Sob a perspetiva da Ciência da Computação e da Tese de Church-Turing expandida, o cérebro humano é frequentemente teorizado como um sistema informático biológico massivo. Embora operemos com redes neuronais paralelas e processos químicos, qualquer cálculo físico determinista ou probabilístico mapeável que realizamos possui uma equivalência algorítmica executável por uma Máquina de Turing.

Contudo, defensores da cognição não-computável argumentam que aspetos humanos como a intuição, a criatividade pura e os saltos de discernimento metafísico escapam ao formalismo rígido passo a passo da máquina. Academicamente, a inteligência humana destaca-se por conseguir criar novos axiomas e formular novos problemas, enquanto a Máquina de Turing está estritamente confinada a processar o conjunto de regras pré-computáveis que lhe foi fornecido.

2. Praticidade:

A prova da indecidibilidade do Problema da Parada afeta diretamente o design de ferramentas de engenharia de software, impondo limites realistas sobre o que pode ser automatizado. Sabendo que é impossível criar um validador perfeito que analise um código qualquer e garanta se ele vai travar ou não, os engenheiros de software focam-se em criar ferramentas de teste baseadas em heurísticas, restrições de tempo (*timeouts*) e aproximações.

Ferramentas de análise estática de código (como Linters ou verificadores de segurança) não tentam resolver o problema de forma universal; em vez disso, alertam para padrões perigosos comuns ou forçam a interrupção do teste caso ele ultrapasse um limite fixo de tempo de execução, contornando a barreira teórica da indecidibilidade com soluções práticas de engenharia.

III. Exercícios Práticos

1. Rastreamento de Fita (Inversor de bits para a cadeia **110**):

Supondo o estado inicial em q_0 com a cabeça de leitura posicionada no primeiro caractere:

- **Início:** Fita = `[1, 1, 0]`, Estado = q_0, Cabeça aponta para o 1º **1**.
- **Passo 1:** A máquina lê **1**, reescreve como **0**, transita para a direita (R).

Fita = `[0, 1, 0]`, Cabeça aponta para o 2º **1**.

- **Passo 2:** A máquina lê **1**, reescreve como **0**, transita para a direita (R).

Fita = `[0, 0, 0]`, Cabeça aponta para o **0**.

- **Passo 3:** A máquina lê **0**, reescreve como **1**, transita para a direita (R).

Fita = `[0, 0, 1]`, Cabeça aponta para a célula em branco (␣) à direita.

Conteúdo final da fita após 3 passos: `001`.

2. Desenho Lógico (Projetar as transições δ para encontrar o primeiro ␣ à direita):

Para realizar essa tarefa, precisamos de um estado de busca (vamos chamá-lo de q_{busca}) que ignore qualquer caractere do alfabeto e continue a mover a cabeça para a direita até atingir o branco, onde efetuará a paragem. Seja **Σ = {0, 1}**:

- $\delta(q_{busca}, 0) = (q_{busca}, 0, R)$ *(Se ler 0, mantém o símbolo 0, continua em* q_{busca} *e move para a direita)*
- $\delta(q_{busca}, 1) = (q_{busca}, 1, R)$ *(Se ler 1, mantém o símbolo 1, continua em* q_{busca} *e move para a direita)*
- $\delta(q_{busca}, ␣) = (q_{parada}, ␣, S)$ *(Se ler o branco ␣, mantém o branco, transita para o estado terminal de paragem* q_{parada} *e a cabeça fica estática)*

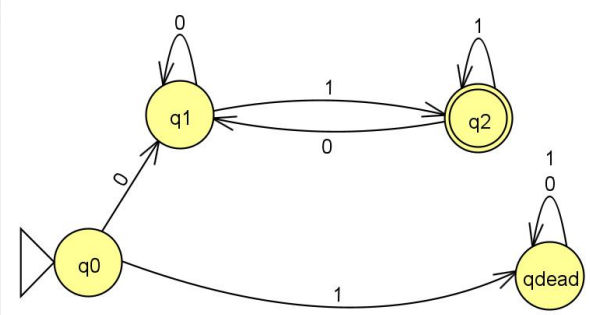
3. Decidibilidade ("Verificar se um programa tem mais de 100 linhas de código"):

Este problema é **estritamente decidível**.

- **Justificativa:** Um algoritmo projetado para essa tarefa necessita apenas ler o arquivo de texto correspondente ao código-fonte e atuar como um contador linear de caracteres de quebra de linha (como o `\n`). O tamanho do arquivo do programa é finito, o que garante que a leitura terá sempre um fim. O contador atingirá o fim do arquivo, fará uma verificação aritmética condicional básica (Se linhas > 100) e terminará sempre em aceitação ou rejeição num tempo finito, sem qualquer risco de loop infinito.

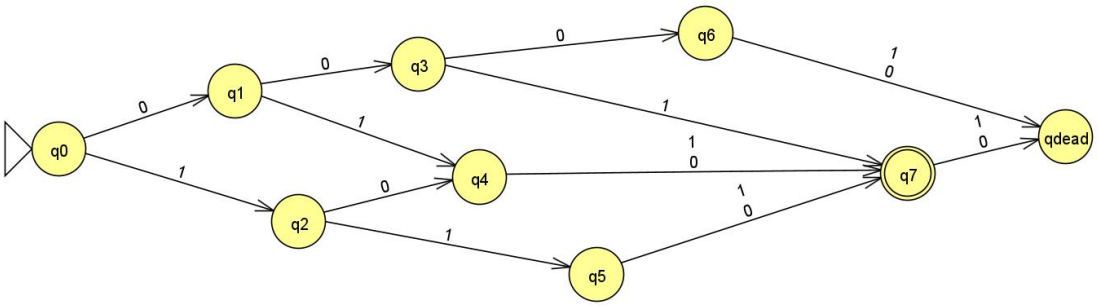
PRÁTICA INTEGRADA

Exercício 1: Validação de Padrões Binários



Input	Result
01	Accept
00101	Accept
110	Reject
11001100	Reject

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



01	Reject
001	Accept
110	Accept
111	Accept
0000	Reject
001001	Reject
1101	Reject

I. Perguntas de Múltipla Escolha (com Justificativa)

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

- Resposta correta: b) Abstrair o problema e identificar a sua classe na Hierarquia de Chomsky.
- Justificativa: Antes de codificar, o engenheiro deve entender a natureza e a complexidade do problema. Classificá-lo na Hierarquia de Chomsky dita o tipo de algoritmo, estrutura de dados e limite computacional que o problema exige.

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

- Resposta correta: c) Gramática Livre de Contexto.
- Justificativa: Estruturas hierárquicas e aninhadas (como tags HTML/XML ou chaves em blocos de código) exigem casamento de padrões e memória de profundidade. Trata-se de uma linguagem do Tipo 2 (Livre de Contexto).

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

- Resposta correta: b) Porque são o formalismo mais eficiente para reconhecimento de padrões de strings sem aninhamento.
- Justificativa: Para validar e-mails, telefones ou CEPs, não há necessidade de memória recursiva ou pilhas. As Expressões Regulares operam sob modelos de Autômatos Finitos, que validam esses formatos em tempo linear O(n) com performance ideal.

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

- Resposta correta: c) A string é rejeitada.
- Justificativa: Na teoria dos autômatos, para uma cadeia ser aceita, toda a fita de entrada deve ser consumida e o estado final do processamento deve pertencer

obrigatoriamente ao conjunto de estados de aceitação (F). Caso contrário, é rejeitada.

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

- Resposta correta: b) Mapear estados e transições de forma determinística.
- Justificativa: Em programação modular de analisadores lógicos, os blocos de `switch` externos controlam o "estado atual" da máquina, e os `switch` ou `if` internos realizam a transição exata (determinística) para o próximo estado de acordo com o caractere lido.

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

- Resposta correta: c) Máquina de Turing.
- Justificativa: A linguagem $\{A^n B^n C^n\}$ é Sensível ao Contexto (Tipo 1). Uma única pilha só consegue parear duas propriedades simultaneamente (esvaziando-se no processo). Para três contagens idênticas, é necessária uma fita bidirecional de acesso aleatório, como a da Máquina de Turing.

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

- Resposta correta: b) É o acompanhamento passo a passo das mudanças de estado e memória da máquina.
- Justificativa: O rastreio ou teste de mesa de um autômato consiste em documentar a tripla de configuração (estado atual, fita restante, conteúdo da memória) a cada pulso ou caractere processado para depurar o fluxo lógico.

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

- Resposta correta: b) Que você compreende a base teórica por trás da implementação prática.
- Justificativa: A atividade serve para ligar a abstração matemática purista (grafos de autômatos) à construção concreta de software de produção, demonstrando que o código robusto nasce de um design formal fundamentado.

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

- Resposta correta: b) Expressões Regulares.
- Justificativa: Conforme visto nas unidades de Compiladores, a fase de análise léxica faz a varredura linear do código de texto para identificar e isolar os lexemas em tokens (identificadores, pontuações, palavras-chave) utilizando exclusivamente Expressões Regulares.

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

- Resposta correta: a) Falta de memória ou loops infinitos desnecessários.
- Justificativa: Tentar resolver um problema simples (Tipo 3) usando uma ferramenta superdimensionada ou inadequada (como fazer um analisador sintático completo para validar um CPF, ou usar Expressões Regulares com backtracking complexo para ler estruturas de blocos recursivos) causa estouro de pilha, consumo indevido de CPU ou travamentos em loops.

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

- Resposta correta: a) A gramática é mais fácil de ler e manter por seres humanos.
- Justificativa: As regras de produção em texto estruturado de uma GLC (ou notação BNF) são declarativas e de fácil leitura, enquanto projetar manualmente as funções de transição detalhadas (δ) com manipulação explícita de elementos do topo de uma pilha em um grafo é um processo muito mais suscetível a erros de engenharia.

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

- Resposta correta: b) Resolve o problema com o menor nível de complexidade necessário na Hierarquia de Chomsky.
- Justificativa: Trata-se do princípio de economia e otimização em engenharia de software: utilize sempre a ferramenta menos complexa capaz de solucionar o desafio de forma segura, garantindo código veloz e de menor custo computacional.

II. Perguntas Reflexivas

1. Simulação vs. Realidade:

O uso de simuladores gráficos como o JFLAP mitiga drasticamente a incidência de falhas lógicas e cenários não mapeados antes da escrita real do código. Ele permite realizar testes de estresse em lotes (*Multiple Run*), verificando simultaneamente dezenas de cadeias limítrofes válidas e inválidas. Quando o engenheiro visualiza que o grafo valida corretamente todas as condições teóricas no simulador, a tradução para código de máquina (seja por matrizes de transição ou estruturas condicionais) torna-se puramente mecânica, elevando a confiabilidade do sistema e mitigando a necessidade de refatorações caras ou correções emergenciais em ambiente de produção.

2. O Papel do Engenheiro:

Num cenário tecnológico onde ferramentas de Inteligência Artificial automatizam a escrita de sintaxes e blocos de código redundantes com facilidade, a capacidade crítica de design arquitetural torna-se o principal ativo de um profissional sênior. Saber codificar sintaticamente não basta. O engenheiro que domina a Teoria da Computação e a Hierarquia de Chomsky é quem dita as diretrizes fundamentais: ele impede que a IA projete uma solução ineficiente baseada em expressões regulares vulneráveis a ataques de negação de serviço (*ReDoS*) para ler arquivos estruturados, e sabe exatamente quando um problema transita da fronteira do decidível para o limiar do impossível, economizando meses de desenvolvimento em soluções inviáveis.

III. Exercícios Práticos (Simulados para a APS)

1. Avaliação (CPF):

- **Formalismo mais simples: Expressão Regular (Linguagem Regular / Tipo 3).**
- **Justificativa:** O formato padrão de um CPF (`xxx.xxx.xxx-xx`) impõe estritamente um tamanho fixo estruturado em posições rígidas (3 dígitos, ponto, 3 dígitos, ponto, 3 dígitos, hífen, 2 dígitos). Não há recursividade, aninhamentos estruturais de escopo ou dependências de contagem variável infinita que requeiram pilhas ou memórias expansivas. Portanto, pode ser totalmente reconhecido por um Autômato Finito através da seguinte Regex:

```
^[0-9]{3}\.[0-9]{3}\.[0-9]{3}\-[0-9]{2}$
```

2. Modelagem Integrada (Colchetes Balanceados):

- **Gramática Livre de Contexto (GLC):** Seja G = ({S\}, {[,]}, P, S) com as regras de produção:

$$S \rightarrow [S] \mid SS \mid \epsilon$$

*(A regra **[S]** gera o aninhamento; **SS** permite a concatenação de múltiplos blocos independentes na mesma linha, ex: **[ ][ ]**; e ϵ realiza o fechamento).*

- **Autômato de Pilha (AP) Equivalente por Estado Final:**

Definimos o controle com 3 estados (q_0, q_1, q_2), onde q_0 inicializa a máquina, q_1 processa a leitura/escrita e q_2 é o estado final de aceitação.

Alfabeto de entrada **Σ = {[,]}** e alfabeto da pilha **Γ = {X, Z_0}**.

- $\delta(q_0, \epsilon, Z_0) = (q_1, Z_0)$ *(Transição inicial de configuração).*
- $\delta(q_1, [, Z_0) = (q_1, XZ_0)$ e $\delta(q_1, [, X) = (q_1, XX)$ *(Sempre que ler **[**, realiza o **Push** empilhando o marcador X).*
- $\delta(q_1,], X) = (q_1, \epsilon)$ *(Sempre que ler **]**, realiza o **Pop** eliminando um símbolo X do topo, validando o par).*
- $\delta(q_1, \epsilon, Z_0) = (q_2, Z_0)$ *(Se a fita acabar e a pilha contiver apenas o marcador de fundo Z_0, o autômato transita para o estado de aceitação q_2).*

3. Relação com Código (AFD que aceita binários terminados em 1):

Aqui está a implementação idiomática do AFD fundamental (visto na Unidade 3) utilizando uma estrutura limpa de controle de estados em **Python**:

Python

```
def token_valida_final_um(cadeia_binaria):
    # q0 representará o Estado Inicial (Último bit lido foi 0 ou vazio)
    # q1 representará o Estado Final de Aceitação (Último bit lido foi 1)
    estado_atual = 'q0'

    # Processamento linear do fluxo de caracteres (A Fita de Entrada)
    for bit in cadeia_binaria:
        if bit not in ['0', '1']:
            return False  # Caractere inválido fora do Alfabeto

        # Matriz de Transição do AFD em formato condicional
        if estado_atual == 'q0':
            if bit == '1':
                estado_atual = 'q1'
            elif bit == '0':
                estado_atual = 'q0'

        elif estado_atual == 'q1':
            if bit == '0':
                estado_atual = 'q0'
            elif bit == '1':
                estado_atual = 'q1'

    # Verificação de Aceitação ao final da leitura da fita
    return estado_atual == 'q1'

# Área de Teste de Mesa / Validação
print(token_valida_final_um("10101"))  # Retorna True (Aceita)
print(token_valida_final_um("11100"))  # Retorna False (Rejeita)
```

Gabarito de Respostas

Unidade 1: 1) C | 2) B | 3) C | 4) B | 5) D

Unidade 2: 6) B | 7) B | 8) C | 9) B | 10) C

Unidade 3: 11) B | 12) A | 13) B | 14) C | 15) B

Unidade 4: 16) C | 17) B | 18) C | 19) B | 20) B

Unidade 5: 21) B | 22) C | 23) B | 24) B

Unidade 6: 25) B | 26) B | 27) C | 28) B

Unidade 7: 29) A | 30) C

Unidade 8: 31) D | 32) A