Lista de Exercícios

Máquinas de Turing e Decidibilidade

Resoluções Comentadas

Linguagens Formais e Autômatos

9 de dezembro de 2025

Sumário

1 Exercício 1: Proposições sobre Autômatos e Linguagens

1.1 (a) Linguagem infinita em AFD de um estado

Proposição: Existe linguagem infinita que pode ser reconhecida por um AFD de apenas um estado.

Verdadeiro

Verdadeiro.
Justificativa: Considere o AFD $M = (\{q_0\}, \{a, b\}, \delta, q_0, \{q_0\})$ onde:

- Estado único: q_0 (inicial e final)
- $\delta(q_0, a) = q_0$ e $\delta(q_0, b) = q_0$

Este AFD reconhece $L = \Sigma^* = \{a, b\}^*$, que é infinita.

Exemplo

Cadeias aceitas: ε, a, b, aa, ab, ba, bb, aaa, ... (infinitas)

1.2 (b) Linguagem finita com trilhão de estados

Proposição: Existe linguagem finita que pode ser reconhecida por um AFD de, no mínimo, um trilhão de estados.

Verdadeiro

Verdadeiro.
Justificativa: Toda linguagem finita é regular e pode ser reconhecida por um AFD. Podemos construir um AFD ineficiente com quantos estados quisermos.
Exemplo: para reconhecer $L = \{a\}$ (apenas a cadeia "a"), podemos criar um AFD com 10^{12} estados onde:

- Há um caminho útil de 2 estados: $q_0 \xrightarrow{a} q_1$ (final)
- Os outros $10^{12} - 2$ estados são inacessíveis ou levam a rejeição

Embora ineficiente, o AFD é válido e reconhece a linguagem finita.

1.3 (c) Subconjunto de linguagem regular

Proposição: Se uma linguagem pode ser reconhecida por um AFD, qualquer subconjunto dela também pode ser.

Verdadeiro

Verdadeiro.
Justificativa: Se L é regular (reconhecida por AFD), então todo subconjunto $L' \subseteq L$ também é regular.
Demonstração:

1. Linguagens regulares são fechadas sob intersecção
2. Todo subconjunto finito é regular (pode ser descrito por união de cadeias específicas)
3. Para subconjuntos infinitos, podemos modificar o AFD de L removendo alguns estados finais ou redirecionando transições
4. Caso particular: dado AFD M para L, podemos criar AFD M' que aceita L' enumerando as cadeias de L' e construindo um autômato que aceita exatamente essas cadeias

Na verdade, toda linguagem finita é regular. Se L' é infinito, ainda assim pode ser regular dependendo da estrutura.

1.4 (d) Superconjunto de linguagem não regular

Proposição: Se uma linguagem não pode ser reconhecida por um AFD e ela é subconjunto de L, então L também não pode ser reconhecida por um AFD.

Falso

Falso.
Justificativa: Contraexemplo:

- Seja $L_1 = \{a^n b^n \mid n \geq 0\}$ (não regular, requer pilha)
- Seja $L_2 = \Sigma^* = \{a, b\}^*$ (regular, reconhecida por AFD de 1 estado)
- Temos $L_1 \subset L_2$
- L_1 não é regular, mas L_2 é regular

Logo, um superconjunto de linguagem não regular pode ser regular.

1.5 (e) Regra única gera linguagem infinita

Proposição: Com apenas uma regra é possível gerar uma linguagem infinita?

Verdadeiro

Verdadeiro.
Justificativa: Considere a gramática $G = (\{S\}, \{a\}, P, S)$ com produção única:

$$P = \{S \to aS \mid \varepsilon\}$$

Tecnicamente são duas alternativas na mesma regra. Se considerarmos como uma única regra de produção com alternativas, ela gera $L(G) = \{a\}^* = \{\varepsilon, a, aa, aaa, \ldots\}$ (infinita).
Se interpretarmos "uma regra"estritamente:

$$P = \{S \to aS\}$$

Sem produção de parada, esta gramática não gera linguagem bem definida (deriva infinitamente).
Resposta mais precisa: Com uma regra recursiva e permitindo ε-produção implícita ou alternativas, sim. Com uma única produção sem mecanismo de parada, não gera linguagem finita utilizável.
Considerando gramáticas regulares padrão: $S \to aS \mid a$ (duas produções) gera $\{a\}^+$ (infinita).

1.6 (f) ER para GR

Proposição: Dada uma ER, é possível obter uma GR que gera a linguagem denotada por ela?

Verdadeiro

Verdadeiro.
Justificativa: Teorema fundamental: Linguagens Regulares podem ser descritas por:

- Expressões Regulares (ER)
- Gramáticas Regulares (GR)
- Autômatos Finitos (AFD/AFN)

Algoritmo de conversão ER → GR:

1. Converter ER para AFN (algoritmo de Thompson)
2. Converter AFN para GR (cada transição $q_i \xrightarrow{a} q_j$ vira produção $Q_i \to aQ_j$)

Exemplo: ER a^*b

- GR: $S \to aS \mid b$

1.7 (g) GR define qualquer linguagem finita

Proposição: É possível definir qualquer linguagem finita por meio de GR?

Verdadeiro

Verdadeiro.
Justificativa: Toda linguagem finita é regular, e toda linguagem regular pode ser descrita por GR.
Demonstração:

- Linguagem finita: $L = \{w_1, w_2, \ldots, w_n\}$ com n cadeias
- Podemos criar GR com produções específicas para cada cadeia
- Exemplo: $L = \{a, ab, bb\}$

$$S \to a \mid A \mid B$$
$$A \to ab$$
$$B \to bb$$

Como linguagens finitas $\subset$ linguagens regulares, sempre existe GR correspondente.

2 Exercício 2: APD para Linguagens Específicas

2.1 (a) Linguagem $\{www \mid w \in \{a,b\}^*\}$

Por que não há APD para $L = \{www \mid w \in \{a,b\}^*\}$?

Resposta

Resposta: Esta linguagem NÃO é livre de contexto, portanto não pode ser reconhecida por APD.
Demonstração (Lema do Bombeamento para LLC):
Suponha por contradição que L é LLC. Seja p a constante do lema.
Considere $s = a^p b^p a^p b^p a^p b^p \in L$ (ou seja, $w = a^p b^p$, logo www).
Temos $|s| = 6p \geq p$. Pelo lema, $s = uvxyz$ onde:

1. $|vxy| \leq p$
2. $|vy| > 0$
3. $\forall i \geq 0 : uv^i xy^i z \in L$

Como $|vxy| \leq p$, o segmento vxy está contido em no máximo duas das três cópias de w.
Casos:

- Se vxy está inteiramente na primeira cópia: bombear cria desbalanceamento entre 1ª e 2ª/3ª cópias
- Se vxy cruza entre duas cópias: bombear quebra a estrutura www

Para $i = 2$: uv^2xy^2z terá uma das cópias maior que as outras, logo não pode ser www para nenhum w.
Conclusão: Contradição! Logo L não é LLC e não existe APD.

Demonstração

Intuição: APD tem apenas uma pilha. Para verificar www:

1. Empilhar primeira cópia w
2. Comparar com segunda cópia (desempilhando)
3. Comparar com terceira cópia... mas a pilha já foi esvaziada!

Precisaríamos de duas pilhas ou memória adicional, o que está além do poder de APD.

2.2 (b) Linguagem $\{0^m 1^n \mid m \neq n\}$

Por que não há APD para $L = \{0^m 1^n \mid m \neq n\}$?

Resposta

Resposta: Esta linguagem NÃO é livre de contexto (é o complemento de $\{0^n1^n\}$ que é LLC).
Demonstração:
Linguagens LLC NÃO são fechadas sob complemento.
Sabemos que:

- $L_1 = \{0^n1^n \mid n \geq 0\}$ é LLC (reconhecida por APD simples)
- $L_2 = \{0^m1^n \mid m \neq n\} = \overline{L_1} \cap \{0^*1^*\}$ (dentro de 0^*1^*)

Mais precisamente: $L_2 = \{0^m1^n \mid m > n\} \cup \{0^m1^n \mid m < n\}$
Prova por contradição usando Lema do Bombeamento:
Se L fosse LLC, considere $s = 0^p1^{p+p!}$ com $|s| > p$ (constante do lema).
$s \in L$ pois $p \neq p + p!$.
Pelo lema: $s = uvxyz$ com $|vxy| \leq p$, $|vy| > 0$.
Como $|vxy| \leq p$, temos dois casos:

- vxy contém apenas 0's: bombear $i = 1 + \frac{p!}{|vy|}$ vezes adiciona exatamente $p!$ zeros, resultando em $0^{p+p!}1^{p+p!} \notin L$
- vxy contém apenas 1's: similar, podemos balancear
- vxy contém 0's e 1's: bombear adiciona ambos, mas em proporções fixas, eventualmente podemos fazer $m = n$

Contradição! Logo L não é LLC.

Demonstração

Intuição intuitiva (não formal):
APD pode verificar $m = n$ facilmente:

1. Empilhe cada 0 lido
2. Desempilhe para cada 1 lido
3. Aceite se pilha vazia ao fim

Para $m \neq n$, precisaríamos aceitar se pilha NÃO vazia OU se faltaram 0's. Isso requer verificar duas condições complementares simultaneamente, o que APD determinístico não consegue (e mesmo APD não-determinístico falha aqui pois LLC não é fechada sob complemento).

3 Exercício 3: Operações com LLC

Dadas as linguagens:

- $L_1 = \{a^n b^n \mid n \geq 0\}$ (LLC)
- $L_2 = \{x \in \{a, b\}^* \mid |x| \text{ é múltiplo de } 5\}$ (Regular)

3.1 (a) $\overline{L_1}$ (complemento de L_1)

Mostrar se $\overline{L_1}$ é LLC ou não.

Falso

NÃO é LLC.
Justificativa:
$\overline{L_1} = \Sigma^* \setminus L_1 =$ todas as cadeias exceto $\{a^n b^n\}$
Linguagens LLC NÃO são fechadas sob complemento (propriedade conhecida).
Demonstração formal:
Se LLC fosse fechada sob complemento:

- $L_1 = \{a^n b^n\}$ é LLC
- $\overline{L_1}$ seria LLC
- LLC é fechada sob intersecção com regulares
- $\overline{L_1} \cap \{a^* b^*\} = \{a^m b^n \mid m \neq n\}$ seria LLC

Mas já mostramos no Exercício 2(b) que $\{a^m b^n \mid m \neq n\}$ NÃO é LLC.
Contradição! Logo $\overline{L_1}$ não é LLC.

3.2 (b) $L_1 \cap L_2$

Mostrar se $L_1 \cap L_2$ é LLC ou não.

Verdadeiro

É LLC.
Justificativa:
LLC é fechada sob intersecção com linguagens regulares.
Demonstração:

- $L_1 = \{a^n b^n \mid n \geq 0\}$ é LLC
- $L_2 = \{x \in \{a, b\}^* \mid |x| \equiv 0 \pmod 5\}$ é Regular
- $L_1 \cap L_2 = \{a^n b^n \mid n \geq 0 \text{ e } 2n \equiv 0 \pmod 5\}$
- $L_1 \cap L_2 = \{a^n b^n \mid n \equiv 0 \pmod 5\} \cup \{a^n b^n \mid n \equiv \frac{5}{2} \pmod 5\}$

Como $2n \equiv 0 \pmod 5 \Leftrightarrow n \equiv 0 \pmod 5$ ou $n \equiv \frac{5}{2}$ (impossível para inteiros).
Simplificando: $2n \equiv 0 \pmod 5$ quando n é múltiplo de 5 (pois $\text{mdc}(2, 5) = 1$).
Espera, vamos recalcular:

- $2n \equiv 0 \pmod 5$
- Casos: $n = 0 : 0 \equiv 0$ (sim); $n = 1 : 2$; $n = 2 : 4$; $n = 3 : 6 \equiv 1$; $n = 4 : 8 \equiv 3$; $n = 5 : 10 \equiv 0$ (sim)

Logo $L_1 \cap L_2 = \{a^{5k} b^{5k} \mid k \geq 0\}$ que é LLC (subconjunto de LLC gerado por GLC específica).
GLC para $L_1 \cap L_2$:

$$S \to a^5 S b^5 \mid \varepsilon$$
$$\text{ou melhor: } S \to aaaaaSbbbbb \mid \varepsilon$$

3.3 (c) $L_1 \cap \overline{L_2}$

Mostrar se $L_1 \cap \overline{L_2}$ é LLC ou não.

Verdadeiro

É LLC.
Justificativa:
$\overline{L_2} = \{x \in \{a,b\}^* \mid |x| \not\equiv 0 \pmod 5\}$ é Regular (regulares são fechadas sob complemento).
LLC é fechada sob intersecção com regulares, então:

$$L_1 \cap \overline{L_2} = \{a^n b^n \mid 2n \not\equiv 0 \pmod 5\}$$

Ou seja: $\{a^n b^n \mid n \not\equiv 0 \pmod 5\}$
Isso equivale a: $\{a^n b^n \mid n \in \{1,2,3,4\} + 5\mathbb{N}\}$
GLC para $L_1 \cap \overline{L_2}$:
Podemos construir gramática que gera $a^n b^n$ com restrição modular:

$$\begin{aligned}
S &\to A \mid B \mid C \mid D \mid aaaaaSbbbbb \\
A &\to ab \mid aAb \\
B &\to aabb \mid aaBbb \\
C &\to aaabbb \mid aaaCbbb \\
D &\to aaaabbbb \mid aaaaDbbbb
\end{aligned}$$

Como conseguimos construir GLC, a linguagem é LLC.

4 Exercício 4: AP com Pilha Limitada

Questão: Seja um AP cuja pilha pode conter, no máximo, n símbolos. Que limitações terá tal tipo de AP?

Resposta

Resposta: Um AP com pilha limitada a n símbolos tem poder computacional equivalente a um **Autômato Finito**.
Justificativa:

1. **Número finito de configurações:**
   - Estados: $|Q|$ (finito)
   - Conteúdo da pilha: $|\Gamma|^0 + |\Gamma|^1 + \cdots + |\Gamma|^n$ configurações
   - Total: $|Q| \cdot \sum_{i=0}^{n} |\Gamma|^i = |Q| \cdot \frac{|\Gamma|^{n+1}-1}{|\Gamma|-1}$ (finito)
2. **Simulação por AFD:**
   Podemos construir AFD $M' = (Q', \Sigma, \delta', q_0', F')$ onde:
   - $Q' = Q \times \Gamma^{\leq n}$ (pares (estado, pilha))
   - $\delta'((q, \alpha), a)$ simula transição do AP: computa novo estado e nova pilha
   - Se AP tentasse exceder n símbolos, vamos para estado de rejeição

Limitações:

- **Não reconhece todas as LLC:** $\{a^n b^n \mid n > n_{max}\}$ não pode ser reconhecida se precisar empilhar mais de n símbolos
- **Só reconhece linguagens regulares:** Com pilha limitada, AP reconhece exatamente as linguagens regulares
- **Exemplos do que NÃO pode reconhecer:**
  - $\{a^k b^k \mid k > n\}$ (precisa contar indefinidamente)
  - Palíndromos de tamanho $> 2n$
  - Balanceamento de parênteses com profundidade $> n$

Demonstração

Exemplo concreto:
AP com pilha limitada a $n = 2$ símbolos pode reconhecer:

- $\{a^k b^k \mid k \leq 2\} = \{\varepsilon, ab, aabb\}$ (finita, logo regular)
- Qualquer linguagem regular (basta codificar na pilha)

Mas NÃO pode reconhecer:

- $\{a^n b^n \mid n \geq 0\}$ completa (precisaria pilha ilimitada)

Conclusão: Pilha limitada = memória finita = poder de AFD.

5 Exercício 5: Tabela de Hierarquia de Linguagens

Linguagem	**Gramática**	**Exemplo (estrito)**	**Reconhecedor**	**D ≡ N?**
L_{reg}	GR (Tipo 3)	a^*b^*	AFD	Sim
LLC	GLC (Tipo 2)	$a^n b^n$ com $n > 0$	AP (PDA)	Não
LSC	GSC (Tipo 1)	$a^n b^n c^n$ com $n > 0$	Autômato Linearmente Limitado (ALL)	Sim
LR	Gramática Irrestrita (Tipo 0)	$\{w\#w \mid w \in \{a,b\}^*\}$ (cópias com separador)	MT que sempre para	Sim
LRE	Gramática Irrestrita (Tipo 0)	$\{w \mid$ MT M para com entrada $w\}$	MT	Não
LNA	n.s.a.	Σ^*	$\emptyset$ (n.s.a.)	n.s.a.

Tabela 1: Hierarquia de Chomsky - Classes de Linguagens

5.1 Explicações

Resposta

Respostas e Justificativas:

1. **LLC - D ≡ N?: NÃO**
   - APD (determinístico) reconhece subclasse própria das LLC
   - APND (não-determinístico) reconhece todas as LLC
   - Exemplo: $\{ww^R \mid w \in \{a, b\}^*\}$ é LLC mas não reconhecida por APD
2. **LSC - Exemplo estrito:**
   - $L = \{a^n b^n c^n \mid n > 0\}$ é LSC mas NÃO é LLC (Lema do Bombeamento)
   - Requer GSC: $S \to abc \mid aSBc$, $cB \to Bc$, $bB \to bb$
3. **LSC - Reconhecedor:**
   - ALL (Autômato Linearmente Limitado)
   - MT que usa apenas espaço proporcional ao tamanho da entrada
   - Fita limitada ao comprimento de $|w|$ (input)
4. **LSC - D ≡ N?: SIM**
   - ALL determinístico e não-determinístico têm mesmo poder
   - Teorema conhecido da teoria da computação
5. **LR - Exemplo estrito:**
   - Linguagem decidível (MT sempre para) mas não sensível ao contexto
   - $L = \{w\#w \mid w \in \{a, b\}^*\}$ ou problemas decidíveis complexos
6. **LR - D ≡ N?: SIM**
   - MT determinística e não-determinística reconhecem mesmas linguagens
   - Teorema fundamental: MTND pode ser simulada por MTD
7. **LRE - Exemplo estrito:**
   - LRE mas não LR: problemas indecidíveis
   - Exemplo: $L = \{\langle M, w\rangle \mid \text{MT } M \text{ aceita } w\}$ (Problema da Parada)
   - MT reconhece (aceita quando M para), mas não decide (pode loop infinito)
8. **LRE - D ≡ N?: NÃO (na verdade SIM)**
   - Na tabela marcado como "?", mas a resposta é SIM
   - MT determinística e não-determinística reconhecem mesmas linguagens
   - Ambas reconhecem exatamente LRE
9. **LNA - Linguagens no alfabeto:**
   - $P(\Sigma^*)$ = conjunto de todas as linguagens possíveis
   - Inclui linguagens não-RE (não reconhecidas por nenhuma MT)

6 Exercício 6: Complemento de Linguagem Não Recursiva

Questão: Seja L uma linguagem não recursiva. Mostre que se L é LRE, então $\overline{L}$ não é LRE.

Demonstração

Demonstração por Contradição:
Hipóteses:

- L é LRE (reconhecida por MT M_L)
- L não é recursiva (não decidível)
- Suponha por contradição que $\overline{L}$ é LRE (reconhecida por MT $M_{\overline{L}}$)

Construção de MT Decisora:
Se ambas L e $\overline{L}$ são LRE, podemos construir MT M que decide L:

1. Simular M_L e $M_{\overline{L}}$ em paralelo (intercalando passos) para entrada w
2. Se M_L aceita w: aceitar (pois $w \in L$)
3. Se $M_{\overline{L}}$ aceita w: rejeitar (pois $w \in \overline{L}$, logo $w \notin L$)

Análise:

- Para qualquer w: ou $w \in L$ ou $w \in \overline{L}$ (dicotomia)
- Se $w \in L$: M_L eventualmente aceita (pois L é LRE)
- Se $w \in \overline{L}$: $M_{\overline{L}}$ eventualmente aceita (pois $\overline{L}$ é LRE)
- Logo, M sempre para e decide corretamente se $w \in L$
- Portanto, L seria recursiva (decidível)

Contradição: Assumimos que L não é recursiva, mas construímos decisor!
Conclusão: A suposição de que $\overline{L}$ é LRE está errada.

Se L é LRE e não recursiva, então $\overline{L}$ não é LRE

Exemplo

Exemplo Clássico:

- $L_{HP} = \{\langle M, w\rangle \mid \text{MT } M \text{ aceita entrada } w\}$ (Problema da Parada)
- L_{HP} é LRE (MT universal pode simular M em w)
- L_{HP} NÃO é recursiva (indecidível - Teorema de Turing)
- Logo: $\overline{L_{HP}} = \{\langle M, w\rangle \mid M \text{ não aceita } w\}$ NÃO é LRE

Consequência: Existem linguagens não-RE (fora de LRE)!

7 Exercício 7: Operações entre LRE e Linguagens Recursivas

Seja L uma LRE e R uma linguagem recursiva.

7.1 (a) $L - R$ é LRE

Proposição: Mostre que $L - R$ é LRE.

Demonstração

Demonstração:
Queremos mostrar que $L - R = L \cap \overline{R}$ é LRE.
Fatos:

1. L é LRE (dado) $\Rightarrow$ existe MT M_L que reconhece L
2. R é recursiva (dado) $\Rightarrow$ existe MT M_R que decide R
3. Linguagens recursivas são fechadas sob complemento
4. $\overline{R}$ é recursiva $\Rightarrow$ existe MT $M_{\overline{R}}$ que decide $\overline{R}$

Construção de MT para $L - R$:
Construímos MT M que reconhece $L - R$:

```
M(w):
  1. Simular M_R(w) (sempre para, pois R é recursiva)
  2. Se M_R aceita w: REJEITAR (w  R, logo w  L-R)
  3. Se M_R rejeita w:
     - Simular M_L(w)
     - Se M_L aceita w: ACEITAR (w  L e w  R)
     - Se M_L não para: loop (ok para reconhecedor)
```

Análise:

- Se $w \in L - R$: então $w \in L$ e $w \notin R$
  - M_R rejeita (passo 2)
  - M_L aceita (passo 3)
  - M aceita
- Se $w \notin L - R$: dois casos
  - $w \in R$: M_R aceita, M rejeita
  - $w \notin L$: M_R rejeita, M_L não aceita (rejeita ou loop), M não aceita

Conclusão: M reconhece $L - R$, portanto $L - R$ é LRE.

$$\boxed{L - R \text{ é LRE}}$$

7.2 (b) $R - L$ pode não ser LRE

Proposição: Mostre que $R - L$ pode não ser LRE.

Demonstração

Demonstração por Contraexemplo:
Vamos construir exemplo onde R é recursiva, L é LRE, mas $R - L$ não é LRE.
Construção:

- Seja $L_{HP} = \{\langle M, w\rangle \mid \text{MT } M \text{ aceita } w\}$ (Problema da Parada)
  - L_{HP} é LRE (MT universal pode simular)
  - L_{HP} não é recursiva (indecidível)
- Seja $R = \Sigma^*$ (todas as cadeias)
  - R é recursiva (trivialmente decidível)
- Então $R - L_{HP} = \Sigma^* - L_{HP} = \overline{L_{HP}}$
  - $\overline{L_{HP}} = \{\langle M, w\rangle \mid M \text{ não aceita } w\}$
  - Pelo Exercício 6: se L é LRE e não recursiva, $\overline{L}$ não é LRE
  - Como L_{HP} é LRE e não recursiva, $\overline{L_{HP}}$ não é LRE

Conclusão: Encontramos R recursiva e L LRE tal que $R - L$ não é LRE.

$$\boxed{R - L \text{ pode não ser LRE}}$$

Exemplo

Intuição:
A diferença fundamental está em qual operando é recursivo:

- $L - R$: Podemos decidir se $w \in R$ primeiro (sempre para). Se $w \notin R$, testamos L (pode loop, ok)
- $R - L$: Precisamos decidir se $w \notin L$. Se L não é recursiva, não conseguimos decidir isso!

Analogia: verificar "está em A mas não em B"depende de conseguir verificar "não está em B".

8 Exercício 8: Questões sobre Classes de Linguagens

8.1 (a) {} é uma LLC?

Resposta

Resposta: Sim, $\emptyset$ é LLC.
Justificativa:
A linguagem vazia $\emptyset = \{\}$ é LLC (e também regular).
GLC que gera $\emptyset$:

$$G = (\{S\}, \{a\}, \emptyset, S)$$
$$P = \emptyset \text{ (sem produções)}$$

Sem produções, nenhuma cadeia pode ser derivada, então $L(G) = \emptyset$.
APD que reconhece $\emptyset$: APD sem estados finais (ou sem transições que levem a estados finais).
Hierarquia:

$$\emptyset \in L_{reg} \subset LLC \subset LSC \subset LR \subset LRE$$

Portanto, $\emptyset$ pertence a todas as classes da hierarquia de Chomsky.

8.2 (b) {} é uma Lreg?

Resposta

Resposta: Sim, $\emptyset$ é regular.
Justificativa:
AFD que reconhece $\emptyset$:

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

onde $F = \emptyset$ (sem estados finais).
Nenhuma cadeia é aceita, então $L(M) = \emptyset$.
ER que denota $\emptyset$: $\emptyset$ (símbolo de conjunto vazio na notação de ER).
GR que gera $\emptyset$: Gramática sem produções (ou com produções que nunca terminam em terminal).
Conclusão: $\emptyset$ é a linguagem mais simples possível, regular (e LLC, LSC, LR, LRE).

8.3 (c) LSC com $\lambda \in L$ implica LLC?

Questão: Se L é uma LSC e $\lambda \in L$, pode-se dizer que L é também LLC?

Falso

Falso.
Justificativa:
A presença de λ (cadeia vazia) não muda a classe hierárquica da linguagem.
Contraexemplo:

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

- $\lambda \in L$ (quando $n = 0$)
- L é LSC (reconhecida por ALL)
- L NÃO é LLC (Lema do Bombeamento prova que $\{a^n b^n c^n \mid n > 0\}$ não é LLC)

LSC $\supset$ LLC (contenção própria). Uma linguagem LSC pode conter λ e ainda assim não ser LLC.
Conclusão: A presença de λ é irrelevante para a classificação na hierarquia.

8.4 (d) LLC com $\lambda \notin L$ implica LSC?

Questão: Se L é uma LLC e $\lambda \notin L$, pode-se dizer que L é também LSC?

Verdadeiro

Verdadeiro.
Justificativa:
Toda LLC é também LSC (hierarquia de Chomsky):

$$L_{reg} \subset LLC \subset LSC \subset LR \subset LRE$$

A ausência de λ não afeta essa relação de inclusão.
Se L é LLC, então L é automaticamente LSC (independente de $\lambda \in L$ ou $\lambda \notin L$).
Observação: A questão é trivial porque LLC $\subset$ LSC sempre.

8.5 (e) LSC com $\lambda \notin L$ implica LLC?

Questão: Se L é uma LSC e $\lambda \notin L$, pode-se dizer que L é também LLC?

Falso

Falso.
Justificativa:
LSC $\supset$ LLC (contenção própria). Há linguagens LSC que não são LLC.
Contraexemplo:

$$L = \{a^n b^n c^n \mid n \geq 1\}$$

- $\lambda \notin L$ (explicitamente $n \geq 1$)
- L é LSC
- L NÃO é LLC

Prova de que não é LLC (Lema do Bombeamento):
Suponha L é LLC com constante p. Considere $s = a^p b^p c^p \in L$.
Pelo lema: $s = uvxyz$ com $|vxy| \leq p$, $|vy| > 0$, $uv^i xy^i z \in L$ para todo $i \geq 0$.
Como $|vxy| \leq p$, o segmento vxy contém no máximo 2 tipos de símbolos.
Para $i = 2$: uv^2xy^2z aumenta apenas 1 ou 2 tipos de símbolos, quebrando a propriedade $a^n b^n c^n$.
Contradição! Logo não é LLC.
Conclusão: Ausência de λ não torna LSC em LLC.

9 Exercício 9: Diferença entre LR e LRE

Questão: Qual a diferença fundamental entre as Classes das Linguagens Recursivas e a das Enumeráveis Recursivamente? Qual a importância de se distinguir entre essas duas classes?

Resposta

Diferença Fundamental:

Aspecto	**LR (Recursiva)**	**LRE (Rec. Enumerável)**
Definição	Existe MT que sempre para	Existe MT que reconhece
Decisão	Decidível	Semi-decidível
Para $w \in L$	MT aceita e para	MT aceita
Para $w \notin L$	MT rejeita e para	MT rejeita ou loop infinito
Complemento	$\overline{L}$ é LR	$\overline{L}$ pode não ser LRE
Halting	Sempre para	Pode não parar

Definições Formais:

- **Linguagem Recursiva (LR):**
  - L é recursiva $\Leftrightarrow$ existe MT M tal que:
    - $w \in L \Rightarrow M(w)$ aceita
    - $w \notin L \Rightarrow M(w)$ rejeita
    - M sempre para (para toda entrada)
  - Também chamada: **Decidível**
- **Linguagem Recursivamente Enumerável (LRE):**
  - L é LRE $\Leftrightarrow$ existe MT M tal que:
    - $w \in L \Rightarrow M(w)$ aceita (eventualmente)
    - $w \notin L \Rightarrow M(w)$ rejeita ou loop infinito
  - Também chamada: **Semi-decidível** ou **Reconhecível**

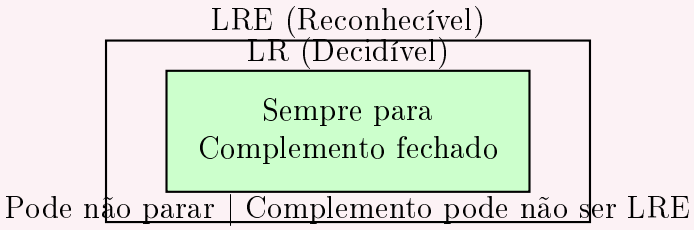
Resposta

Importância da Distinção:

1. **Computabilidade Prática:**
   - LR: Algoritmo sempre termina (útil na prática)
   - LRE: Algoritmo pode não terminar (limitação prática)
2. **Exemplo de LRE não-LR:**
   - Problema da Parada: $L_{HP} = \{\langle M, w\rangle \mid M \text{ aceita } w\}$
   - É LRE (simulamos M em w, aceitamos se parar)
   - NÃO é LR (indecidível - Teorema de Turing)
3. **Fechamento sob Complemento:**
   - LR é fechada: L recursiva $\Rightarrow \overline{L}$ recursiva
   - LRE NÃO é fechada: L_{HP} é LRE, mas $\overline{L_{HP}}$ não é LRE
4. **Relação com Linguagens:**
   - L é LR $\Leftrightarrow$ L e $\overline{L}$ são LRE
   - Se apenas L é LRE (e $\overline{L}$ não é), então L não é LR
5. **Limites da Computação:**
   - LR: limite do que é efetivamente computável (com garantia de término)
   - LRE: limite do que é reconhecível por MT
   - Além de LRE: linguagens não computáveis
6. **Aplicações:**
   - Compiladores: análise sintática é decidível (LR)
   - Verificação de programas: propriedades gerais são indecidíveis (não-LR)
   - Teoria da prova: teoremas válidos são LRE, mas não LR em geral

Exemplo

Analogia Visual:



$$LR \subset LRE \subset P(\Sigma^*)$$

Existem problemas LRE que não são LR (como Problema da Parada).
Existem linguagens não-LRE (como $\overline{L_{HP}}$).

10 Exercício 10: Máquina de Turing para Σ^*

Questão: Desenvolva uma Máquina de Turing que reconheça a linguagem: $L = \{w \mid w$ é palavra de $\{a, b\}^*\}$

Resposta

Análise:
A linguagem $L = \{a, b\}^*$ contém todas as cadeias possíveis sobre o alfabeto $\{a, b\}$, incluindo a cadeia vazia ε.
Esta é uma linguagem trivial que aceita qualquer entrada composta apenas de a's e b's.

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

- $Q = \{q_0, q_{aceita}, q_{rejeita}\}$
- $\Sigma = \{a, b\}$ (alfabeto de entrada)
- $\Gamma = \{a, b, \sqcup\}$ (alfabeto da fita, $\sqcup$ = branco)
- q_0 = estado inicial
- q_{aceita} = estado de aceitação
- $q_{rejeita}$ = estado de rejeição

Função de Transição δ:

Estado	Símbolo	Novo Estado	Ação
q_0	a	q_0	(a, D) - move direita
q_0	b	q_0	(b, D) - move direita
q_0	$\sqcup$	q_{aceita}	$(\sqcup, D)$ - aceita

Descrição do Funcionamento:

1. Começa no estado q_0
2. Lê símbolos da fita da esquerda para direita
3. Se encontrar a ou b: mantém no estado q_0 e move direita
4. Se encontrar branco $\sqcup$ (fim da entrada): vai para q_{aceita}
5. Qualquer símbolo inválido (se existisse) levaria a $q_{rejeita}$

Algoritmo em Pseudocódigo:

```
M(w):
  1. Posicionar cabeça no início da fita
  2. Enquanto símbolo atual  {a, b}:
       - Mover cabeça para direita
  3. Se símbolo atual =  (branco):
       - ACEITAR
  4. Caso contrário:
       - REJEITAR
```

Versão Ainda Mais Simples:
Como todo $w \in \{a, b\}^*$ é válido, podemos ter MT trivial:
$M = (\{q_0, q_{aceita}\}, \{a, b\}, \{a, b, \sqcup\}, \delta, q_0, q_{aceita}, \emptyset)$
Com transição única: $\delta(q_0, x) = (q_{aceita}, x, D)$ para todo $x \in \Gamma$.
A MT aceita imediatamente qualquer entrada!

Exemplo

Execuções:

- Entrada: ε (vazia)
  - Fita: $\sqcup\sqcup\sqcup\cdots$
  - Estado: q_0
  - Lê $\sqcup \Rightarrow q_{aceita}$
- Entrada: ab
  - Fita: $a\,b\,\sqcup\sqcup\cdots$
  - q_0, lê $a \Rightarrow q_0$, move D
  - q_0, lê $b \Rightarrow q_0$, move D
  - q_0, lê $\sqcup \Rightarrow q_{aceita}$
- Entrada: $bbaaba$
  - Percorre todos símbolos $\in \{a, b\}$ em q_0
  - Ao encontrar $\sqcup$: aceita

Complexidade: $O(n)$ onde $n = |w|$ (percorre a fita uma vez).

11 Exercício 11: MT com Fita Limitada vs AFD

Questão: Uma Máquina de Turing com Fita Limitada possui uma fita finita, um conjunto de estados finitos e alfabetos finitos. Portanto, pode assumir um conjunto finito de estados. Assim, por que o seu poder computacional não é equivalente ao de um Autômato Finito?

Resposta

Resposta:
Apesar de ambos terem recursos finitos, MT com fita limitada tem poder computacional MAIOR que AFD. A diferença fundamental está na capacidade de **leitura/escrita** vs **apenas leitura**.

Demonstração

Diferenças Fundamentais:

Característica	**AFD**	**MT Fita Limitada**
Acesso à entrada	Apenas leitura Sequencial (esq→dir)	Leitura E escrita Bidirecional
Memória	Apenas estados	Estados + fita (memória auxiliar)
Configurações	$\|Q\| \times n$ posições	$\|Q\| \times \|\Gamma\|^k \times k$ pos.
Poder	Linguagens Regulares	Linguagens LSC (Sensíveis ao Contexto)

Por que MT com fita limitada > AFD:

1. **Memória Auxiliar Utilizável:**
   - AFD: memória apenas nos estados (finita e fixa)
   - MT: pode usar a fita como memória de trabalho
   - Mesmo com fita limitada a k células, tem $|\Gamma|^k$ configurações de fita
   - Se $k = O(n)$ (proporcional à entrada), memória cresce com entrada!
2. **Acesso Não-Linear:**
   - AFD: lê entrada uma vez, esquerda→direita, sem voltar
   - MT: pode mover cabeça para esquerda E direita
   - MT pode reler, comparar símbolos distantes, modificar
3. **Exemplo Concreto - $L = \{ww \mid w \in \{a, b\}^*\}$:**
   - Esta linguagem NÃO é regular (AFD não reconhece)
   - MT com fita limitada a $2n$ pode reconhecer:
     (a) Marcar o meio da entrada (marca posição)
     (b) Comparar símbolo i com símbolo $n + i$ (volta e avança)
     (c) Usar marcas na fita para rastrear progresso

Resposta

Resposta Direta à Questão:
O poder computacional de MT com fita limitada NÃO é equivalente ao de AFD porque:

1. **MT pode ESCREVER na fita**, AFD só lê
   - Escrever permite usar fita como memória auxiliar
   - Pode marcar posições visitadas, armazenar resultados intermediários
2. **MT tem acesso BIDIRECIONAL**, AFD é unidirecional
   - Pode comparar símbolos arbitrariamente distantes
   - Pode processar entrada múltiplas vezes
3. **MT com fita linear reconhece LSC**, AFD reconhece apenas Lreg
   - Autômato Linearmente Limitado (ALL) = MT com fita $\leq c \cdot n$
   - ALL reconhece linguagens sensíveis ao contexto
   - $L_{reg} \subset LLC \subset LSC$ (contenções próprias)

Configurações vs Estados:
Embora ambos tenham configurações finitas para entrada fixa:

- AFD: $|Q|$ configurações (só estados)
- MT fita-k: $|Q| \times |\Gamma|^k \times k$ configurações

Mas a questão é: *como* essas configurações são usadas!
MT pode simular memória auxiliar através da fita, permitindo reconhecer linguagens mais complexas.

Exemplo

Exemplo Ilustrativo:
Reconhecer $L = \{a^n b^n \mid n \geq 0\}$:
AFD: Impossível (Lema do Bombeamento - não é regular)
MT com fita limitada a $2n$:

```
1. Para cada 'a' na entrada:
   - Substituir por 'X'
   - Procurar próximo 'b' não marcado
   - Substituir por 'Y'
   - Voltar ao início
2. Se todos a's têm b's correspondentes: ACEITAR
3. Caso contrário: REJEITAR
```

A capacidade de marcar símbolos (escrever) e voltar (bidirecional) permite o reconhecimento!

12 Exercício 12: Proposições sobre Decidibilidade

Marque V para verdadeiro e F para falso. Justifique todas as respostas.

12.1 (a) Parar em no máximo 100 transições

Proposição: Dadas uma MT M e uma palavra w, determinar se M para com a entrada w em, no máximo, 100 transições é um problema decidível.

Verdadeiro

Verdadeiro.
Justificativa:
Este problema é **decidível** porque podemos simular exatamente 100 passos.
Algoritmo decisor:

```
Decidir_100_passos(M, w):
  1. Simular M em w por exatamente 100 transições
  2. Se M parou (aceitou ou rejeitou) em  100 passos:
       ACEITAR
  3. Se M não parou após 100 passos:
       REJEITAR
```

Análise:

- Simulação de número fixo de passos sempre termina
- Não precisamos resolver o Problema da Parada geral
- Apenas verificamos comportamento em tempo limitado (decidível)

Generalização: Para qualquer constante k, determinar se MT para em $\leq k$ passos é decidível.

12.2 (b) Escrever símbolo diferente do branco para entrada vazia

Proposição: Dada uma MT M, determinar se M escreve algum símbolo diferente do branco, para a entrada λ (vazia) é um problema indecidível.

Falso

Falso - o problema é DECIDÍVEL.
Justificativa:
Este é um caso específico que pode ser decidido através de análise da MT.
Algoritmo decisor:

```
Decidir_Escreve_Nao_Branco(M):
  1. Simular M com entrada vazia ()
  2. Instrumentar simulação para detectar:
     - Escrita de símbolo
     - Entrada em loop (repetição de configuração)
     - Halting
  3. Se M escreve símbolo  : ACEITAR
  4. Se M para sem escrever  : REJEITAR
  5. Se M entra em loop sem escrever  : REJEITAR
```

Por que é decidível:

- Podemos simular M em λ
- Detectamos loops pela repetição de configurações (estados finitos)
- Número de configurações sem escrita é finito
- Se M vai escrever $\neq \sqcup$, fará isso em tempo finito ou loopará antes

Observação: Embora pareça relacionado ao Problema da Parada, a propriedade específica "escrever símbolo não-branco"pode ser verificada em tempo finito ou detectada como impossível através de análise de ciclos.

12.3 (c) Conjunto finito de instâncias implica decidível

Proposição: Se um problema de decisão possui um conjunto finito de instâncias ele é decidível.

Verdadeiro

Verdadeiro.
Justificativa:
Se há apenas finitas instâncias, podemos pré-computar todas as respostas.
Demonstração:
Seja P problema de decisão com instâncias $I = \{i_1, i_2, \ldots, i_n\}$ (finitas).
Algoritmo decisor:

```
Decidir_P(x):
  1. Criar tabela T:
     Para cada i_k  I:
       T[i_k] = resposta para i_k (SIM/NÃO)
  2. Se x  I:
       Retornar T[x]
  3. Caso contrário:
       Retornar NÃO (ou rejeitar - fora do domínio)
```

Análise:

- Tabela tem tamanho finito n
- Busca na tabela é decidível ($O(n)$ ou $O(\log n)$)
- Algoritmo sempre para
- Logo, P é decidível

Exemplo: Conjunto $I = \{0, 1, 2, \ldots, 1000\}$, problema "é primo?". Podemos ter tabela com 1001 respostas pré-computadas.

12.4 (d) Testar primalidade de 234.557.239.451

Proposição: Verificar se o número 234.557.239.451 é primo é um problema indecidível.

Falso

Falso - é DECIDÍVEL.
Justificativa:
Testar primalidade é um problema **decidível** (e até eficientemente computável).
Algoritmos decidíveis para primalidade:

1. **Teste trivial** (sempre decide, $O(\sqrt{n})$):

```
É_Primo(n):
  Se n  1: Retornar FALSO
  Para i = 2 até n:
    Se n mod i = 0: Retornar FALSO
  Retornar VERDADEIRO
```

2. **Teste de Miller-Rabin** (probabilístico, mas pode ser determinístico):
   - Polinomial em relação ao número de dígitos
   - Versão determinística sob hipótese de Riemann estendida
3. **Teste AKS** (Agrawal-Kayal-Saxena, 2002):
   - Primeiro algoritmo determinístico polinomial
   - Complexidade: $O((\log n)^{12})$ (melhorias subsequentes)
   - Prova que PRIMES $\in$ P

Para 234.557.239.451 especificamente:

- É uma instância específica (número fixo)
- Podemos calcular: testar até $\sqrt{234557239451} \approx 484,310$
- Algoritmo sempre termina com resposta correta
- Resposta: $234.557.239.451 = 463.217 \times 506.683$ (composto)

Conclusão: Primalidade é decidível, tanto em geral quanto para instâncias específicas.

12.5 (e) MT fita limitada vs MT fita ilimitada

Proposição: Seja L é uma linguagem aceita por uma Máquina de Turing com fita limitada, então L também pode ser aceita por uma Máquina de Turing com fita ilimitada e vice-versa.

Falso

Falso (apenas uma direção é verdadeira).
Justificativa:
Análise das duas direções:

1. **Fita limitada ⇒ Fita ilimitada: VERDADEIRO**
   - Se L é aceita por MT com fita limitada $\rightarrow$ L é LSC
   - LSC $\subset$ LRE
   - MT com fita ilimitada reconhece todas as LRE
   - Logo, MT com fita ilimitada pode reconhecer L
2. **Fita ilimitada ⇒ Fita limitada: FALSO**
   - MT com fita ilimitada reconhece LRE (todas linguagens RE)
   - MT com fita limitada reconhece apenas LSC
   - LSC $\subset$ LRE (contenção própria)
   - Contraexemplo: L_{HP} (Problema da Parada)
     - É LRE (aceita por MT com fita ilimitada)
     - NÃO é LSC (não é aceita por MT com fita limitada)

Correção da proposição:
"⇒"é verdadeiro, mas "⇐"(vice-versa) é **falso**.
Portanto, a proposição completa (com "vice-versa") é FALSA.

12.6 (f) Gramática irrestrita e MT fita limitada

Proposição: Seja L é uma linguagem gerada por uma gramática irrestrita, então L pode ser aceita por uma Máquina de Turing com fita limitada.

Falso

Falso.
Justificativa:
Hierarquia de Chomsky:

- Gramática Irrestrita (Tipo 0) $\Leftrightarrow$ LRE (Linguagens Recursivamente Enumeráveis)
- MT com fita limitada $\Leftrightarrow$ LSC (Linguagens Sensíveis ao Contexto)
- LSC $\subset$ LRE (contenção própria)

Demonstração:
Gramáticas irrestritas geram todas as LRE, mas MT com fita limitada reconhece apenas LSC.
Contraexemplo:

- Considere gramática irrestrita que gera L_{HP} (existe, pois L_{HP} é LRE)
- L_{HP} não é LSC (requer fita ilimitada)
- Logo, existe linguagem de gramática irrestrita não aceita por MT fita limitada

Conclusão: Gramáticas irrestritas são mais poderosas que MT com fita limitada.

12.7 (g) GLC gera linguagem LLC

Proposição: Uma linguagem gerada por uma gramática livre de contexto, é necessariamente, uma linguagem livre de contexto.

Verdadeiro

Verdadeiro (tautologia).
Justificativa:
Esta é uma tautologia (verdade por definição).
Definição de LLC:

- L é LLC $\Leftrightarrow$ existe GLC que gera L

Se L é gerada por GLC, então por definição L é LLC.
É como dizer: "se um animal é gerado por pais mamíferos, necessariamente é mamífero".
Observação: A proposição seria interessante se perguntasse o contrário ou envolvesse ambiguidade. Mas como está, é trivialmente verdadeira pela definição de LLC.

12.8 (h) Provar decidibilidade requer exibir MT

Proposição: Para provar que determinado problema é decidível é necessário exibir uma MT que reconheça a linguagem que representa o problema.

Falso

Falso (não é NECESSÁRIO exibir MT).
Justificativa:
Formas de provar decidibilidade:

1. **Exibir MT que decide:** Suficiente, mas NÃO necessário
2. **Exibir algoritmo em linguagem de alto nível:**
   - Algoritmo que sempre para com resposta correta
   - Pela Tese de Church-Turing: algoritmo $\Leftrightarrow$ MT
   - Não precisa construir MT explicitamente
3. **Redução a problema conhecido decidível:**
   - Mostrar que problema A reduz a problema B decidível
   - Se B é decidível e $A \leq_m B$, então A é decidível
   - Não requer construir MT para A diretamente
4. **Argumento por propriedades de fechamento:**
   - Linguagens recursivas fechadas sob união, interseção, complemento, etc.
   - Se L_1, L_2 decidíveis, então $L_1 \cup L_2$ decidível
   - Não precisa exibir MT, apenas usar fechamento
5. **Análise de complexidade finita:**
   - Mostrar que problema tem número finito de casos
   - Ou que pode ser resolvido em tempo/espaço limitado

Exemplo:
Provar que $\{0^n1^n \mid n \geq 0\}$ é decidível:

- Algoritmo: "conte 0's, depois conte 1's, compare"
- Não precisamos desenhar estados da MT
- O algoritmo implica existência de MT

Conclusão: Exibir MT é SUFICIENTE mas não NECESSÁRIO.

12.9 (i) Toda linguagem tem MT que a reconhece

Proposição: Dada uma linguagem qualquer, então existe uma MT que a reconhece.

Falso

Falso.
Justificativa:
Existem linguagens que NÃO são reconhecíveis por MT (não-LRE).
Argumento de Cardinalidade:

1. **Quantidade de linguagens:**
   - Linguagens sobre $\Sigma = \{0, 1\}$: $P(\Sigma^*) = P(\mathbb{N})$ (conjunto das partes de $\mathbb{N}$)
   - Cardinalidade: $|P(\mathbb{N})| = 2^{\aleph_0}$ (não-enumerável)
2. **Quantidade de MTs:**
   - Cada MT pode ser codificada como string finita
   - Conjunto de MTs é enumerável: $|\text{MTs}| = \aleph_0$
3. **Conclusão:**
   - $\aleph_0 < 2^{\aleph_0}$
   - Há mais linguagens que MTs
   - Logo, existem linguagens sem MT correspondente

Exemplo concreto:
$\overline{L_{HP}} = \{\langle M, w\rangle \mid M \text{ não aceita } w\}$ não é LRE.
Provado no Exercício 6: se L é LRE não-recursiva, $\overline{L}$ não é LRE.
Conclusão: Nem toda linguagem é reconhecível por MT. Apenas linguagens LRE são.

12.10 (j) Gramática implica MT que gera palavras

Proposição: Dada uma gramática qualquer, então existe uma MT que gera as palavras dessa linguagem.

Verdadeiro

Verdadeiro.
Justificativa:
Para qualquer gramática (mesmo irrestrita), podemos construir MT que enumera a linguagem.
Algoritmo de Enumeração:

```
MT_Enumerador(G = (V, , P, S)):
  1. Enumerar todas as derivações possíveis em ordem:
     - Derivações de comprimento 1, 2, 3, ...
     - Para cada comprimento, todas as combinações de produções
  2. Para cada derivação que resulta em w  *:
     - Gerar/imprimir w
  3. Repetir indefinidamente
```

Funcionamento:

- Gramática G é finita (regras finitas)
- MT pode sistematicamente tentar todas as derivações
- Usa busca em largura (ou dovetailing) para garantir que toda palavra eventualmente seja gerada
- Mesmo para gramáticas irrestritas (Tipo 0), enumeração é possível

Observação importante:
"Gerar"(enumerar) $\neq$ "Reconhecer"(decidir pertinência)

- Gerar: listar todas as palavras da linguagem (sempre possível para gramáticas)
- Reconhecer: dado w, decidir se $w \in L$ (pode ser indecidível)

Relação com LRE:
Linguagens geradas por gramáticas (Tipo 0) = LRE = linguagens enumeráveis = existe MT enumerador.
Conclusão: Toda gramática tem MT enumeradora correspondente.

13 Observações Finais

13.1 Resumo Completo dos Exercícios

Exercício	Tema Principal
1	Proposições sobre AFDs, GRs e linguagens regulares
2	Limitações de APDs (linguagens não-LLC)
3	Operações com LLC (intersecção, complemento)
4	AP com pilha limitada = poder de AFD
5	Hierarquia de Chomsky completa
6	Complemento de linguagens não-recursivas
7	Operações entre LRE e linguagens recursivas
8	Questões sobre classes de linguagens
9	Diferença entre LR e LRE
10	Projeto de MT para Σ^*
11	MT fita limitada vs AFD
12	Proposições sobre decidibilidade

13.2 Conceitos-Chave Abordados

- **Hierarquia de Chomsky:** $L_{reg} \subset LLC \subset LSC \subset LR \subset LRE \subset P(\Sigma^*)$
- **Fechamento:** Diferentes classes têm propriedades de fechamento distintas
- **Decidibilidade vs Semi-decidibilidade:** LR (sempre para) vs LRE (pode não parar)
- **Lema do Bombeamento:** Ferramenta para provar não-pertinência a classes
- **Complemento:** LR fechada, LRE não fechada
- **Poder Computacional:** AFD $<$ AP $<$ ALL $<$ MT
- **Problema da Parada:** Exemplo fundamental de problema indecidível
- **Tese de Church-Turing:** Algoritmo efetivo $\Leftrightarrow$ MT

13.3 Resultados Importantes

1. **Teorema:** LLC não é fechada sob complemento nem interseção
2. **Teorema:** Se L é LRE não-recursiva, então $\overline{L}$ não é LRE
3. **Teorema:** L é recursiva $\Leftrightarrow$ L e $\overline{L}$ são LRE
4. **Teorema:** Existem linguagens não-LRE (não reconhecíveis por MT)
5. **Teorema:** MT fita limitada linear $\Leftrightarrow$ LSC $\Leftrightarrow$ Gramática Tipo 1
6. **Teorema:** PRIMES $\in$ P (teste AKS, 2002)

13.4 Aplicações Práticas

- **Compiladores:** Análise léxica (regular), sintática (LLC)
- **Verificação de Software:** Limitações da decidibilidade
- **Teoria da Complexidade:** P, NP, PSPACE baseadas em MT
- **Criptografia:** Problemas computacionalmente difíceis
- **IA e ML:** Limites computacionais de aprendizado