

# Decidibilidade

Prof. Marcus Vinícius Midená Ramos

Universidade Federal do Vale do São Francisco

15 de agosto de 2018

`marcus.ramos@univasf.edu.br`  
`www.univasf.edu.br/~marcus.ramos`

# Bibliografia

## Básica:

- ▶ *Introduction to Automata Theory, Languages and Computation (capítulo 9)*  
J. E. Hopcroft, R. Motwani e J. D. Ullman  
Addison-Wesley, 2007, 3ª edição
- ▶ *Introdução à Teoria da Computação (capítulos 4 e 5)*  
M. Sipser  
Thomson, 2006, 2ª edição

## Complementar:

- ▶ *Languages and Machines (capítulo 12)*  
T. A. Sudkamp  
Addison-Wesley, 2006, 3ª edição
- ▶ *Teoria da Computação: Máquinas Universais e Computabilidade (capítulo 9)*  
T. A. Diverio e P B Menezes  
Bookman, 2011, 3ª edição

# Roteiro

- 1 Introdução
- 2 Problemas decidíveis
- 3 Linguagem  $L_d$
- 4 Complemento de linguagens
- 5 Máquina de Turing Universal
- 6 Linguagem  $L_u$
- 7 Redutibilidade
- 8 Problema da parada
- 9 Linguagens  $L_e$  e  $L_{ne}$
- 10 Teorema de Rice
- 11 Autômato Linearmente Limitado
- 12 Problemas indecidíveis e histórias de computação
- 13 PCP
- 14 Problemas relacionados com GLCs e LLCs

# Questões

- ▶ Existe um algoritmo que resolve um certo problema?
- ▶ Como demonstrar que existe — ou que não existe — tal algoritmo?

# Definições

- ▶ Decidibilidade é o estudo dos problemas codificados como linguagens;
- ▶ Máquinas de Turing são usadas como representação formal da noção de algoritmo;
- ▶ A prova da existência (ou não) de um algoritmo que resolve um certo problema é equivalente à demonstração da existência (ou não) de uma Máquina de Turing que resolve o mesmo problema.

# Problema de decisão

## Conceito

- ▶ Um problema é dito um “problema de decisão” quando ele é transformado num problema equivalente, cujas respostas são apenas SIM ou NÃO;
- ▶ A coleção das instâncias de um problema de decisão cujas respostas são apenas afirmativas forma a linguagem que representa o referido problema;
- ▶ Necessidade de se codificar as instâncias do problema de forma unívoca.

# Problema de decisão

## Essência

- ▶ Determinar se a linguagem que representa um problema de decisão é recursiva.
  - ▶ Em caso afirmativo, existe um algoritmo (melhor caso);
  - ▶ Em caso negativo, investigar se a linguagem é recursivamente enumerável.
- ▶ Determinar se a linguagem que representa um problema de decisão é recursivamente enumerável.
  - ▶ Em caso afirmativo, é possível determinar as instâncias afirmativas do problema, mas haverá sempre pelo menos uma entrada (cuja resposta é negativa) que nunca produzirá resposta;
  - ▶ Em caso negativo, haverá sempre pelo menos uma entrada (cuja resposta é positiva) que nunca produzirá resposta (pior caso);

# Problema de decisão

## Exemplo

- ▶ Problema  $P$ : determinar se um número binário é par.
- ▶ Problema de decisão equivalente  $P'$ : agrupar os números binários que são pares (resposta afirmativa ao problema) e formar uma linguagem  $L$  com eles.
- ▶  $L = \{0, 10, 100, 110, 1000, 1010, 1100, 1110, \dots\}$ . Note que os números ímpares (1, 01, 11 etc) não pertencem à  $L$ ;



# Problema de decisão

## Exemplo

- ▶ A resposta ao problema  $P$  — determinar se um número binário é par — é transformada na resposta à pergunta: “o número binário fornecido pertence à linguagem  $L$ ?”
- ▶ Genericamente, pretende-se determinar se existe uma Máquina de Turing  $M$  que sempre pára e é capaz de decidir se uma cadeia qualquer de zeros e uns pertence à linguagem  $L$ ;
- ▶ Caso exista tal máquina, isso implica a existência de um algoritmo que resolve  $P$  e diz-se que  $M$  “decide”  $P$ . Caso contrário, não existe tal algoritmo.

# Problema de decisão

## Exemplos

Suponha que  $c(X)$  representa uma codificação de  $X$  sobre um certo alfabeto  $\Sigma$ .

- ▶ Dadas duas gramáticas livres de contexto  $G_1$  e  $G_2$ , é possível determinar se  $L(G_1) = L(G_2)$ ?
  - ▶ Codificar  $G_1$  e  $G_2$  de forma adequada;
  - ▶ Considerar a linguagem  $\{c(G_1)c(G_2) \mid L(G_1) = L(G_2)\}$
  - ▶ Determinar se essa linguagem é recursiva.
- ▶ Dadas uma Máquina de Turing  $M$  e uma entrada  $w$ , é possível determinar se  $M$  aceita  $w$ ?
  - ▶ Codificar  $M$  e  $w$  de forma adequada;
  - ▶ Considerar a linguagem  $\{c(M)c(w) \mid M \text{ aceita } w\}$
  - ▶ Determinar se essa linguagem é recursiva.

# Linguagens Recursivas e Decidibilidade

- ▶ Um problema de decisão é dito “decidível” se a linguagem que representa as instâncias **afirmativas** do problema forma uma linguagem recursiva. Caso contrário o problema é dito “não-decidível” (ou “indecidível”).
- ▶ Como linguagens recursivas são reconhecidas por Máquinas de Turing que sempre param, qualquer que seja a entrada, a existência de um algoritmo que resolve um problema de decisão implica a existência de pelo menos uma Máquina de Turing que **sempre pára**, qualquer que seja a entrada fornecida (afirmativa ou negativa).

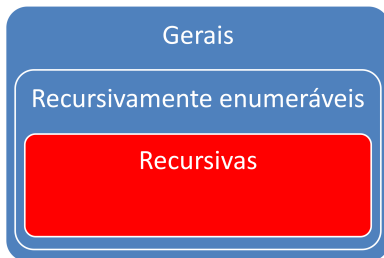
# Outras Classes de Linguagens

- ▶ Problemas de decisão que formam linguagens recursivamente enumeráveis e não-recursivas são aceitos **apenas** por Máquinas de Turing que entram em loop para pelo menos uma instância do problema de decisão cuja resposta é negativa;
- ▶ Problemas de decisão que formam linguagens não-recursivamente enumeráveis não são aceitos por nenhuma Máquina de Turing que pare sempre que as instâncias são afirmativas (ou seja, **toda e qualquer** Máquina de Turing construída para este problema entra em loop com pelo menos uma instância do problema cuja resposta é afirmativa).

# Definições

Solucionável × Não-solucionável

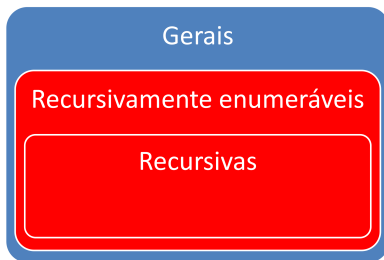
- ▶ Problema solucionável  $\Leftrightarrow$  Linguagem recursiva
- ▶ Problema não-solucionável  $\Leftrightarrow$  Linguagem não-recursiva



# Definições

Parcialmente solucionável  $\times$  Totalmente insolúvel

- ▶ Problema parcialmente solucionável (ou computável)  $\Leftrightarrow$  Linguagem recursivamente enumerável
- ▶ Problema totalmente insolúvel (ou não-computável)  $\Leftrightarrow$  Linguagem não-recursivamente enumerável



# Definições

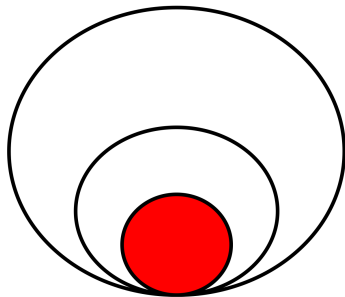
## Resumo

- ▶ Todo problema é solucionável ou não-solucionável;
- ▶ Todo problema solucionável é parcialmente solucionável;
- ▶ Todo problema é parcialmente solucionável ou totalmente insolúvel;
- ▶ Um problema não-solucionável pode ser parcialmente solucionável ou totalmente insolúvel;
- ▶ Um problema parcialmente solucionável pode ser solucionável ou não.

# Definições

## Problema Solucionável

- ▶ Problema **solucionável**  $\Leftrightarrow$  Linguagem **recursiva** ;
- ▶ Existe pelo menos uma MT que aceita a linguagem e pára com toda entrada.

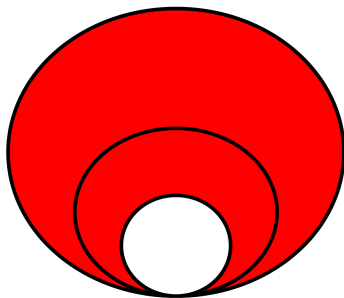




# Definições

## Problema Não-Solucionável

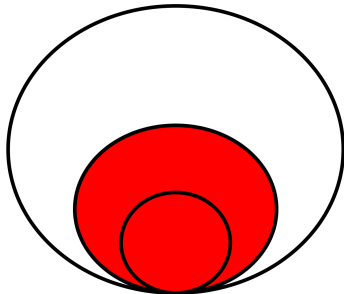
- ▶ Problema não-solucionável  $\Leftrightarrow$  Linguagem não-recursiva ;
- ▶ Se existir uma MT que aceite a linguagem, ela necessariamente entra em loop com alguma instância negativa. Mas também pode ser que não exista nenhuma MT que a aceite.



# Definições

## Problema Parcialmente Solucionável

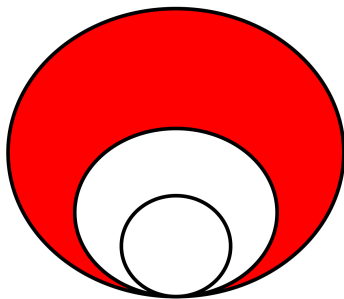
- ▶ Problema **parcialmente solucionável**  $\Leftrightarrow$  Linguagem **recursivamente enumerável** ;
- ▶ Existe pelo menos uma MT que aceita esta linguagem. Pode ser que ela pare com todas as instâncias negativas, ou não.



# Definições

## Problema Totalmente Insolúvel

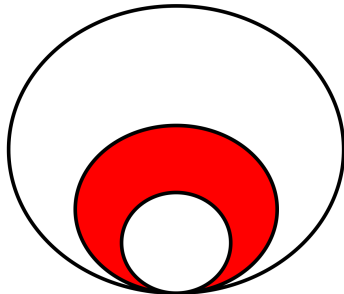
- ▶ Problema totalmente insolúvel  $\Leftrightarrow$   
Linguagem não-recursivamente enumerável ;
- ▶ Não existe nenhuma MT que aceite esta linguagem.



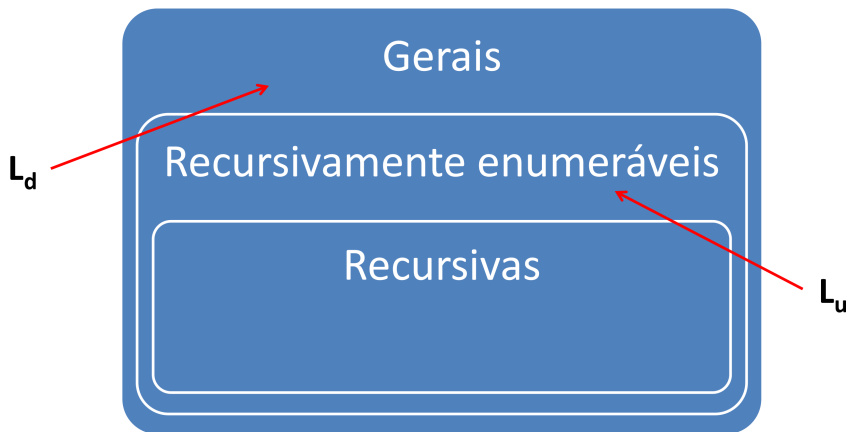
# Definições

## Problema Não-Solucionável e Parcialmente Solucionável

- ▶ Problema não-solucionável e parcialmente solucionável  $\Leftrightarrow$   
Linguagem recursivamente enumerável e não-recursiva;
- ▶ Existe pelo menos uma MT que aceita esta linguagem. Porém, todas elas entram em loop com alguma instância negativa.



## Conceitos



# Motivação

Por que estudar decidibilidade?

- ▶ Ajuda a identificar problemas insolúveis;
- ▶ Evita desperdício de tempo e esforço com a tentativa de resolução de problemas insolúveis;
- ▶ Aponta para possibilidades de simplificações e/ou alterações do problema original, a fim de que ele se torne solúvel;
- ▶ Amplia a sua compreensão sobre a natureza, as possibilidades e os limites da computação.

# Seqüência

- 1 Problemas decidíveis;
- 2 Problemas indecidíveis;
- 3 Técnicas para classificar problemas de natureza originalmente desconhecida como sendo decidíveis ou indecidíveis.

# Problema $A_{AFD}$

Aceitação em autômatos finitos determinísticos sem transições em vazio:

$$A_{AFD} = \{\langle B, w \rangle \mid B \text{ é um AFD que aceita a cadeia } w\}$$

Teorema:  $A_{AFD}$  é uma linguagem decidível.

Prova:

- 1 Construir uma MT  $M$  que analisa  $\langle B \rangle$ ;
- 2 Se  $\langle B \rangle$  não representa um AFD válido,  $M$  pára e rejeita a entrada;
- 3 Se  $\langle B \rangle$  representa um AFD válido,  $M$  simula  $B$  com a entrada  $w$ ;
- 4 Se  $B$  pára numa configuração final, então  $M$  pára e aceita;
- 5 Se  $B$  pára numa configuração não-final, então  $M$  pára e rejeita.



# Problema $A_{AFN}$

Aceitação em autômatos finitos  
não-determinísticos com transições em vazio:

$$A_{AFN} = \{\langle B, w \rangle \mid B \text{ é um AFN que aceita a cadeia } w\}$$

Teorema:  $A_{AFN}$  é uma linguagem decidível.

Prova:

- 1 Construir uma MT  $M$  que analisa  $\langle B \rangle$ ;
- 2 Se  $\langle B \rangle$  não representa um AFN válido,  $M$  pára e rejeita a entrada;
- 3 Se  $\langle B \rangle$  representa um AFN válido,  $M$  converte o AFN  $B$  para um AFD  $B'$  equivalente;
- 4  $M$  simula  $B'$  com a entrada  $w$ ;
- 5 Se  $B'$  pára numa configuração final, então  $M$  pára e aceita;
- 6 Se  $B'$  pára numa configuração não-final, então  $M$  pára e rejeita.

# Problema $A_{AFN}$

O algoritmo usado para demonstrar a decidibilidade de  $A_{AFD}$  não se aplica. A existência de um ciclo formado por transições em vazio no autômato finito pode fazer com que a simulação entre em loop infinito e, portanto, não consiga produzir uma resposta.

# Problema $A_{EXR}$

Geração de cadeia por expressão regular:

$$A_{EXR} = \{ \langle R, w \rangle \mid R \text{ é uma expressão regular que gera a cadeia } w \}$$

Teorema:  $A_{EXR}$  é uma linguagem decidível.

Prova:

- 1 Construir uma MT  $M$  que analisa  $\langle R \rangle$ ;
- 2 Se  $\langle R \rangle$  não representa uma expressão regular válida,  $M$  pára e rejeita;
- 3 Se  $\langle R \rangle$  representa uma expressão regular válida,  $M$  converte  $R$  para um AFN  $B$  que reconhece a mesma linguagem;
- 4  $M$  converte o AFN  $B$  para um AFD  $B'$  equivalente;
- 5  $M$  simula  $B'$  com a entrada  $w$ ;
- 6 Se  $B'$  pára numa configuração final, então  $M$  pára e aceita;
- 7 Se  $B'$  pára numa configuração não-final, então  $M$  pára e rejeita.

# Problema $V_{AFD}$

Vacuidade da linguagem reconhecida por autômato finito determinístico:

$$V_{AFD} = \{\langle B \rangle \mid B \text{ é um AFD e } L(B) = \emptyset\}$$

Teorema:  $V_{AFD}$  é uma linguagem decidível.

Prova:

- 1 Marcar o estado inicial de  $B$ ;
- 2 Repetir até que nenhum novo estado venha a ser marcado:
  - ▶ Marque todos os estados de destino para os quais existam transições partindo de um estado já marcado;
- 3 Se nenhum estado final estiver marcado, páre e aceite; caso contrário, páre e rejeite.

# Problema $EQ_{AFD}$

Igualdade das linguagens reconhecidas por dois autômatos finitos determinísticos:

$$EQ_{AFD} = \{\langle A, B \rangle \mid A, B \text{ são AFDs e } L(A) = L(B)\}$$

Teorema:  $EQ_{AFD}$  é uma linguagem decidível.

Prova:

- 1 Construir o AFD  $C$  que reconhece a linguagem:  
 $L(A) \cap \overline{L(B)} \cup (\overline{L(A)} \cap L(B))$   
 Notar que  $L(A) = L(B) \Leftrightarrow L(C) = \emptyset$ ;
- 2 Determinar se  $L(C) = \emptyset$ ;
- 3 Em caso afirmativo, páre e aceite a entrada;
- 4 Caso contrário, páre e rejeite a entrada;
- 5 Pode ser feita também com minimizações.

# Problema $A_{GLC}$

Geração de cadeia por gramática livre de contexto:

$$A_{GLC} = \{ \langle G, w \rangle \mid G \text{ é uma GLC que gera } w \}$$

Teorema:  $A_{GLC}$  é uma linguagem decidível.

Prova:

- 1 Construir uma MT que obtém  $G'$  na Forma Normal de Chomsky ( $A \rightarrow BC \mid a$ ) tal que  $L(G) = L(G')$ ;
- 2 Considerar  $n = |w|$ ;
- 3 Se  $n > 0$ , então fazer todas as derivações com  $2 * n - 1$  passos;
- 4 Se  $n = 0$ , então fazer todas as derivações com 1 passo;
- 5 Se alguma dessas derivações gera  $w$ , páre e aceite;
- 6 Caso contrário, páre e rejeite.

# Problema $A_{GLC}$

- ▶ A geração de uma cadeia de comprimento  $n$  numa GLC na Forma Normal de Chomsky demanda, necessariamente, a aplicação de  $n - 1$  regras do tipo  $A \rightarrow BC$  (para gerar uma forma sentencial com  $n$  símbolos não-terminais) e também a aplicação de  $n$  regras do tipo  $A \rightarrow a$  para transformar a forma sentencial numa sentença.
- ▶ Portanto, a geração de uma cadeia de  $n$  símbolos requer a aplicação de  $2 * n - 1$  regras ou passos de derivação;
- ▶ Existe um conjunto finito de seqüências de derivação com qualquer quantidade de passos;
- ▶ Basta obter todas elas e verificar se alguma produz a cadeia informada na entrada;
- ▶ Em caso afirmativo, a cadeia pertence à linguagem;
- ▶ Em caso negativo, ela não pertence.

# Problema $A_{GLC}$

## Exemplo

Seja:

- ▶  $S \rightarrow AB$
- ▶  $S \rightarrow c$
- ▶  $A \rightarrow AA$
- ▶  $B \rightarrow BB$
- ▶  $A \rightarrow a$
- ▶  $B \rightarrow b$

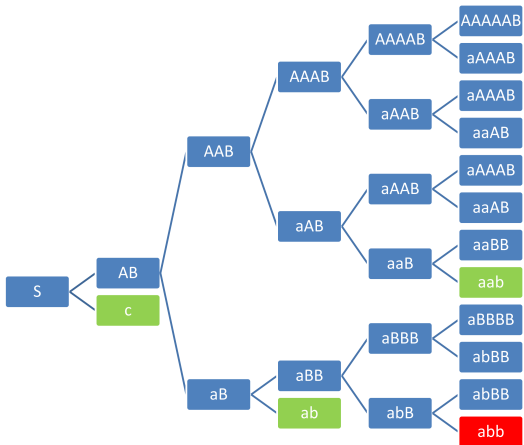
Considere a cadeia  $abb$ :

- ▶  $|abb| = 3$ ;
- ▶ Deve-se pesquisar todas as seqüências de derivação com até  $2 * 3 - 1 = 5$  passos;
- ▶ Para simplificar, serão consideradas apenas derivações mais à esquerda.



# Problema $AGLC$

## Exemplo



# Problema $AGLC$

Construir uma MT que simula  $G$  diretamente pode não funcionar, pois pode haver seqüências infinitas de derivações em  $G$ . Por exemplo, se  $G$  possuir as regras unitárias  $X \rightarrow Y$  e  $Y \rightarrow X$ , é possível que a seqüência de derivações torne-se infinita:

$$\dots \Rightarrow \alpha X \beta \Rightarrow \alpha Y \beta \Rightarrow \alpha X \beta \Rightarrow \alpha Y \beta \Rightarrow \dots$$

# Problema $V_{GLC}$

Vacuidade da linguagem gerada por uma gramática livre de contexto:

$$V_{GLC} = \{\langle G \rangle \mid G \text{ é uma GLC e } L(G) = \emptyset\}$$

Teorema:  $V_{GLC}$  é uma linguagem decidível.

Prova:

- 1 Marcar todos os símbolos terminais de  $G$ ;
- 2 Repetir até que nenhum novo símbolo não-terminal venha a ser marcado:
  - ▶ Marque todos os símbolos não-terminais  $X$  para os quais existam regras  $X \rightarrow Y_1 Y_2 \dots Y_n$  e cada  $Y_i$  já esteja marcado;
- 3 Se a raiz da gramática não estiver marcada, páre e aceite; caso contrário, páre e rejeite.

# Problema $V_{GLC}$

Testar todas as cadeias  $w$  em  $G$  diretamente pode não funcionar, pois pode haver uma quantidade infinita de cadeias a serem testadas.

Problema  $EQ_{GLC}$ 

Igualdade das linguagens geradas por duas gramáticas livres de contexto:

$$EQ_{GLC} = \{\langle G, H \rangle \mid G, H \text{ são GLCs e } L(G) = L(H)\}$$

Teorema:  $EQ_{AFD}$  é uma linguagem indecidível.

Prova:

- ▶ Será vista mais adiante;
- ▶ A classe das linguagens livres de contexto não é fechada em relação às operações de complementação e intersecção (logo, a estratégia usada na prova de  $EQ_{AFD}$  não pode ser usada aqui).

# Problema $LLC$

Determinar se uma cadeia pertence à uma determinada linguagem livre de contexto  $L$  (análise sintática):

$$LLC = \{\langle w \rangle \mid w \in L(G)\}$$

Teorema:  $LLC$  é uma linguagem decidível.

Prova:

- ▶ Seja  $G$  uma GLC tal que  $L = L(G)$ ;
- ▶ Determinar se  $\langle G, w \rangle$  é aceita pela MT que decide  $A_{GLC}$ ;
- ▶ Em caso afirmativo, páre e aceite;
- ▶ Caso contrário, páre e rejeite.

# Problema $LLC$

Construir uma MT que simula diretamente um autômato de pilha  $P$  que reconhece  $L$  pode não funcionar, pois podem haver seqüências de movimentações infinitas em  $P$ .

# Ordenação de cadeias binárias

Seja  $\Sigma = \{0, 1\}$ . Então o conjunto  $\Sigma^*$  é enumerável.

- ▶ Basta considerar as cadeias  $w \in \Sigma^*$  em ordem crescente de comprimento;
- ▶ Para cada comprimento, considerar as cadeias ordenadas lexicograficamente;
- ▶  $\epsilon, 0, 1, 00, 01, 10, 11, 000, 001, 010, \dots$
- ▶ A  $i$ -ésima cadeia será denotada  $w_i$ ;
- ▶  $w_1 = \epsilon, w_2 = 0, w_3 = 1, w_4 = 00, w_5 = 01, w_6 = 10, w_7 = 11, \dots$



# Codificação de Máquinas de Turing

## Convenções

Seja  $M$  com alfabeto de entrada  $\Sigma = \{0, 1\}$ . Uma codificação de  $M$  sobre o próprio alfabeto  $\Sigma$  é a seguinte:

- ▶  $Q = \{q_1, q_2, \dots, q_r\}$ ;
- ▶ Suponha que o estado inicial é  $q_1$ ;
- ▶ Suponha critério de aceitação “Entrada” (a máquina pára quando entra num estado final);
- ▶ Suponha que há um único estado final, e ele é  $q_2$ ;
- ▶  $\Gamma = \{X_1, X_2, \dots, X_s\}$ ;
- ▶ Suponha  $X_1 = 0, X_2 = 1, X_3 = B$ . Os demais símbolos são auxiliares;
- ▶ Suponha que  $D_1$  representa movimento para a esquerda,  $D_2$  para a direita.

# Codificação de Máquinas de Turing

## Convenções

Considere  $\delta(q_i, X_j) = (q_k, X_l, D_m)$ . Uma codificação para essa transição é:

$$0^i 10^j 10^k 10^l 10^m$$

onde:

- ▶  $0^i$  representa o estado  $q_i$ ;
- ▶  $0^j$  representa o símbolo  $X_j$ ;
- ▶  $0^k$  representa o estado  $q_k$ ;
- ▶  $0^l$  representa o símbolo  $X_l$ ;
- ▶  $0^m$  representa o movimento  $D_m$ .

Como  $i, j, k, l, m$  são maiores que zero, a cadeia 11 não é subcadeia de  $0^i 10^j 10^k 10^l 10^m$ . 11 será usada para separar transições.

# Codificação de Máquinas de Turing

## Convenções

Considere  $|\delta| = n$ . Uma codificação para  $\delta$  (e conseqüentemente para a Máquina de Turing  $M$ ) é:

$$C_111C_211\dots C_{n-1}11C_n$$

onde  $C_i$  representa a codificação da transição  $i$ .

Como cada  $C_i$  começa e termina com pelo menos um símbolo 0, a cadeia 111 não é subcadeia de  $C_111C_211\dots C_{n-1}11C_n$ . 111 será usada para separar a MT de outros elementos, se for o caso.

## Codificação de Máquinas de Turing

## Exemplo

Seja:

$$M = (\{q_1, q_2, q_3\}, \{0, 1\}, \{0, 1, B\}, \delta, q_1, B, \{q_2\})$$

com:

$$\begin{array}{l} \delta(q_1, 1) = (q_3, 0, R) \quad \underbrace{0}_{q_1} \ 1 \ \underbrace{00}_1 \ 1 \ \underbrace{000}_{q_3} \ 1 \ \underbrace{0}_0 \ 1 \ \underbrace{00}_R \\ \delta(q_3, 0) = (q_1, 1, R) \quad 0001010100100 \\ \delta(q_3, 1) = (q_2, 0, R) \quad 00010010010100 \\ \delta(q_3, B) = (q_3, 1, L) \quad 0001000100010010 \end{array}$$

Portanto, a cadeia que representa  $M$  é:

$$\underbrace{0100100010100}_{\delta(q_1, 1) = (q_3, 0, R)} \ 11 \ \underbrace{0001010100100}_{\delta(q_3, 0) = (q_1, 1, R)} \ 11 \ \underbrace{00010010010100}_{\delta(q_3, 1) = (q_2, 0, R)} \ 11 \ \underbrace{0001000100010010}_{\delta(q_3, B) = (q_3, 1, L)}$$

# Cadeias binárias e Máquinas de Turing

Com a ressalva abaixo, é possível considerar a  $i$ -ésima cadeia binária  $w_i$  como sendo a representação de uma Máquina de Turing, denotada  $M_i$ .

- ▶ Se  $w_i$  não respeita as regras de formação enunciadas anteriormente, então considerar  $M_i$  como a Máquina de Turing formada por um único estado (não-final), sem transições, e que pára para qualquer entrada; portanto,  $L(M_i) = \{\}$ ;
- ▶ Caso contrário,  $w_i$  denota a Máquina de Turing  $M_i$  codificada conforme as regras expostas.

# Método diagonal de Cantor

- ▶ Publicado em 1891;
- ▶ Mostra como obter um conjunto diferente de todos os conjuntos de uma dada coleção de conjuntos, seja ela finita ou infinita;
- ▶ Cada um dos conjuntos dessa coleção, por sua vez, pode conter um número finito ou infinito de elementos;
- ▶ O número de elementos usados para caracterizar tais conjuntos também pode ser finito ou infinito;
- ▶ Bastante usado até os dias de hoje.

# Método diagonal de Cantor

## Características:

- ▶ Matriz com linhas e colunas;
- ▶ Cada coluna representa um certo elemento (podem existir infinitas colunas);
- ▶ Cada linha representa um conjunto criado com esses elementos (se a quantidade de colunas for infinita, podem existir infinitos conjuntos);
- ▶ O cruzamento de uma linha com uma coluna é marcado para indicar se aquele elemento pertence (1) ou não pertence (0) ao respectivo conjunto;
- ▶ Considere a diagonal e troque 0s por 1s e vice-versa;
- ▶ O conjunto assim obtido é diferente de todos os conjuntos representados na matriz.

# Método diagonal de Cantor

## Exemplo

- ▶ Suponha que os elementos são números naturais;
- ▶ Cada coluna representa um número natural;
- ▶ Cada linha representa um subconjunto dos números naturais.
- ▶ Quaisquer que sejam os conjuntos considerados nas linhas, a complementação da diagonal principal produz um novo subconjunto desses mesmos elementos que difere de todos os considerados nas linhas da matriz.



# Método diagonal de Cantor

## Exemplo

	0	1	2	3	4	...
$S_1$	1	1	1	0	1	...
$S_2$	1	1	0	0	1	...
$S_3$	1	0	0	1	0	...
$S_4$	0	1	1	1	0	...
$S_5$	0	0	1	0	0	...
...	...	...	...	...		...

# Método diagonal de Cantor

## Exemplo

Na figura anterior, temos:

- ▶  $S_1 = \{0, 1, 2, 4, \dots\}$ ;
- ▶  $S_2 = \{0, 1, 4, \dots\}$ ;
- ▶  $S_3 = \{0, 3, \dots\}$ ;
- ▶  $S_4 = \{1, 2, 3, \dots\}$ ;
- ▶  $S_5 = \{2, \dots\}$ ;
- ▶ Diagonal: 11010...;
- ▶ Diagonal complementada: 00101...;
- ▶ Conjunto obtido:  $X = \{2, 4, \dots\}$ ;
- ▶  $X \neq S_i, i \geq 0$ .

# Método diagonal de Cantor

## Aplicações

Serve, por exemplo, para demonstrar que  $|\mathbb{N}| < |2^{\mathbb{N}}|$ :

- ▶ Suponha que  $|\mathbb{N}| = |2^{\mathbb{N}}|$ ;
- ▶ Então, existe uma bijeção entre  $\mathbb{N}$  e  $2^{\mathbb{N}}$ ;
- ▶ As colunas representam os números naturais;
- ▶ Cada linha representa um subconjunto dos números naturais dessa bijeção; suponha que eles sejam rotulados por números naturais, a partir de zero;
- ▶ Sempre é possível obter um novo subconjunto que não foi considerado pela bijeção;
- ▶ A hipótese é falsa e não existe tal bijeção;
- ▶  $|\mathbb{N}| < |2^{\mathbb{N}}|$ .

# Método diagonal de Cantor

## Aplicações

Bijeção hipotética representada pela matriz:

	0	1	2	3	4	...
0	$p_{00}$	$p_{01}$	$p_{02}$	$p_{03}$	$p_{04}$	...
1	$p_{10}$	$p_{11}$	$p_{12}$	$p_{13}$	$p_{14}$	...
2	$p_{20}$	$p_{21}$	$p_{22}$	$p_{23}$	$p_{24}$	...
3	$p_{30}$	$p_{31}$	$p_{32}$	$p_{33}$	$p_{34}$	...
4	$p_{40}$	$p_{41}$	$p_{42}$	$p_{43}$	$p_{44}$	...
...	...	...	...	...	...	...

Basta considerar o subconjunto  $\{i \in \mathbb{N} \mid p_{ii} = 0, \forall i \geq 0\}$ .

Linguagem  $L_d$ 

$$L_d = \{w_i \in \{0, 1\}^* \mid w_i \notin L(M_i)\}$$

- ▶ Contém as cadeias que, quando consideradas como codificações de Máquinas de Turing, são tais que elas não são aceitas pelas respectivas Máquinas de Turing que elas representam;
- ▶ Linguagem da “diagonalização”.

# Diagonalização e a linguagem $L_d$

Para cada par linha/coluna  $(i, j)$ , a tabela indica se  $M_i$  aceita  $w_j$ :

	$w_1$	$w_2$	$w_3$	$w_4$	...
$\langle M_1 \rangle = w_1$	0	1	1	0	...
$\langle M_2 \rangle = w_2$	1	1	0	0	...
$\langle M_3 \rangle = w_3$	0	0	1	1	...
$\langle M_4 \rangle = w_4$	0	1	0	1	...
...	...	...	...	...	...

1 indica aceitação, 0 indica rejeição ou loop (os valores apresentados são hipotéticos).

# Diagonalização e a linguagem $L_d$

- ▶ Vetor característico:  $0, 1, 1, 1, \dots$ ;
- ▶ Complemento do vetor característico:  $1, 0, 0, 0, \dots$ ;
- ▶  $w_1 \in L_d, w_2 \notin L_d, w_3 \notin L_d, w_4 \notin L_d$  etc;
- ▶ Portanto,  $L_d = \{w_1, \dots\}$ ;
- ▶  $L_d = \{w_i \mid w_i \notin L(M_i)\}$ ;

# Diagonalização e a linguagem $L_d$

- ▶  $L_d$  não é aceita por nenhuma Máquina de Turing, pois o vetor característico dela difere em pelo menos uma posição do vetor característico de todas as linguagens aceitas por todas as Máquinas de Turing que existem;
- ▶ Em outras palavras, existe pelo menos uma cadeia que difere  $L_d$  de  $L(M_i), \forall i \geq 1$ ;
- ▶  $L_d$  não é uma linguagem recursivamente enumerável;
- ▶ Não existe nenhuma Máquina de Turing que aceite  $L_d$ .



# Teorema 1

$L_d$  não é recursivamente enumerável

## Teorema:

A linguagem  $L_d$  não é recursivamente enumerável.

## Prova:

- ▶ Suponha que  $L_d$  seja recursivamente enumerável. Então deve existir uma Máquina de Turing  $M$  que aceita  $L_d$ . Logo,  $M = M_i$  para algum valor de  $i$ . Considere, portanto, que  $M_i$  aceita  $L_d$  e considere a cadeia  $w_i$ :
  - ▶ Se  $w_i \in L_d$ , então  $M_i$  aceita  $w_i$ . Mas, por definição, se  $M_i$  aceita  $w_i$  então  $w_i$  não pode pertencer à  $L_d$ ;
  - ▶ Se  $w_i \notin L_d$ , então  $M_i$  não aceita  $w_i$ . Mas, por definição, se  $M_i$  não aceita  $w_i$  então  $w_i$  deve pertencer à  $L_d$ .
- ▶ Qualquer que seja o caso, há uma contradição;
- ▶ Logo, a hipótese é falsa e não existe  $M_i$  que aceite  $L_d$ .

# Teorema 2

Se  $L$  é recursiva, então  $\overline{L}$  também é recursiva

Ideia geral:



## Teorema 2

Se  $L$  é recursiva, então  $\overline{L}$  também é recursiva

### Teorema:

Se  $L$  é recursiva, então  $\overline{L}$  também é recursiva.

### Prova:

Seja  $L = L(M)$ , onde  $M$  é uma Máquina de Turing que sempre pára. O seguinte método mostra como obter  $M'$  a partir de  $M$  de tal forma que  $L(M') = \overline{L(M)}$ . Inicialmente,  $M' = M$ .

- 1 Os estados finais de  $M$  tornam-se não-finais em  $M'$ ;
- 2 As transições que partiam dos estados finais de  $M$  (agora não finais em  $M'$ ) são removidas em  $M'$  (critério "entrada" apenas);
- 3  $M'$  tem um novo e único estado final, não existente em  $M$ , denotado  $r$ ;
- 4 Para cada combinação de estado não-final de  $M$  e símbolo de entrada não aceito nesse estado, adicionar, em  $M'$ , uma transição do mesmo estado com esse símbolo para  $r$ .

## Teorema 2

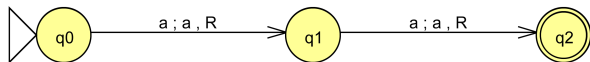
Se  $L$  é recursiva, então  $\overline{L}$  também é recursiva

- ▶ (1) e (2) garantem que todas as cadeias aceitas por  $M$  são rejeitadas por  $M'$ ;
- ▶ (3) e (4) garantem que todas as cadeias rejeitadas por  $M$  são aceitas por  $M'$ ;
- ▶ Como  $M$  sempre pára, então  $M'$  sempre pára também;
- ▶ Portanto,  $M'$  aceita  $\overline{L}$  e  $\overline{L}$  é recursiva.

## Teorema 2

## Exemplo

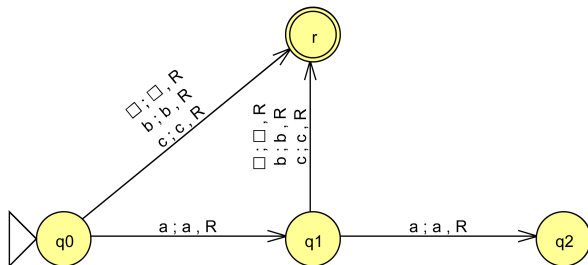
A Máquina de Turing  $M$  abaixo aceita a linguagem  $aa(a|b|c)^*$  (cadeias que possuem  $aa$  como prefixo):



## Teorema 2

## Exemplo

A Máquina de Turing  $M'$  abaixo aceita a linguagem  $(a|b|c)^* - aa(a|b|c)^*$  (cadeias que não possuem  $aa$  como prefixo):



## Teorema 3

$L$  e  $\overline{L}$  são recursivamente enumeráveis se e somente se  $L$  é recursiva

### Teorema:

$L$  e  $\overline{L}$  são recursivamente enumeráveis se e somente se  $L$  é recursiva.

### Prova:

( $\Leftarrow$ ) Se  $L$  é recursiva, então, pelo Teorema 1,  $\overline{L}$  também é recursiva.

Como, pela definição, toda linguagem recursiva é também recursivamente enumerável, isso prova que  $L$  e  $\overline{L}$  são recursivamente enumeráveis.

( $\Rightarrow$ ) Sejam  $M_1$  e  $M_2$  as Máquinas de Turing que aceitam, respectivamente,  $L$  e  $\overline{L}$ . Os métodos apresentados a seguir mostram como obter  $M$  a partir de  $M_1$  e  $M_2$  de tal forma que  $L(M) = L$  e  $M$  sempre pára. Ou seja, eles provam que  $L$  é recursiva.

# Teorema 3

## Método

Idéia geral: Simular  $M_1$  e  $M_2$  de forma intercalada, até que uma das duas pare e aceite a entrada:

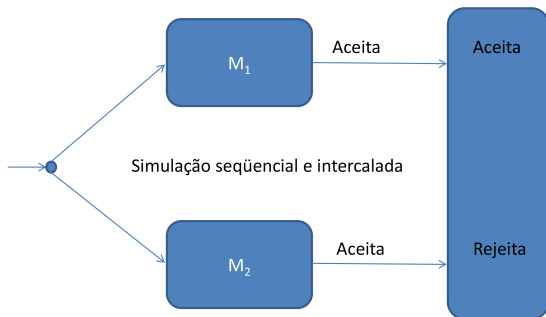
- 1 Executar, alternadamente, movimentos em  $M_1$  e  $M_2$ ;
- 2 Como toda cadeia  $w$  pertence à  $L(M_1)$  ou  $L(M_2)$ , é fato que  $M_1$  ou  $M_2$  aceita  $w$ ;
- 3 Se  $M_1$  aceita, então  $M$  pára e aceita;
- 4 Se  $M_2$  aceita, então  $M$  pára e rejeita;
- 5 Assim,  $L(M) = L$ ,  $M$  sempre pára, e portanto  $L$  é recursiva.



# Teorema 3

## Método

### Idéia geral:



# Teorema 3

## Método

Descrição: Construir  $M$  com duas fitas para simular, de forma intercalada, a operação de  $M_1$  na primeira fita e de  $M_2$  na segunda fita:

- 1 Ambas as fitas são inicializadas com a cadeia de entrada  $w$  a ser analisada;
- 2 Os estados de  $M$  são construídos para representar pares de estados de  $M_1$  e  $M_2$ , e também a máquina (1 ou 2) que irá se movimentar em seguida;
- 3 Em cada estado de  $M$ , são considerados alternadamente os símbolos presentes na primeira e na segunda fita;
- 4 Todos os estados de  $M$  que representam algum estado final de  $M_1$  são finais; os demais estados de  $M$  são todos não-finais;
- 5 Se  $M_1$  ( $M_2$ ) parar sem aceitar, continuar com  $M_2$  ( $M_1$ ).

# Teorema 3

## Método

### Detalhamento:

1.  $M$  copia a entrada  $w$  da fita 1 para a fita 2;
2.  $M$  seleciona  $M_1$ ;
3.  $M$  tentar executar um movimento de  $M_1$ ;
4. Se  $M_1$  não tem movimento possível,  $M$  seleciona  $M_2$  e vá para 6;
5. Senão,  $M$  simula o movimento de  $M_1$  na fita 1 e seleciona  $M_2$ ;
6.  $M$  tentar executar um movimento de  $M_2$ ;
7. Se  $M_2$  não tem movimento possível, vá para 2;
8. Senão,  $M$  simula o movimento de  $M_2$  na fita 2 e vá para 2.

## Teorema 3

## Método

Algoritmo:Entrada:

- ▶ MT  $M_1 = (Q_1, \Sigma, \Gamma_1, \delta_1, q_{01}, B, F_1)$  determinística que aceita  $L$  e tem “entrada” como critério de aceitação;
- ▶ MT  $M_2 = (Q_2, \Sigma, \Gamma_2, \delta_2, q_{02}, B, F_2)$  determinística que aceita  $\bar{L}$  e tem “entrada” como critério de aceitação;

Saída:

- ▶ MT  $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$  que aceita  $L$  e sempre pára;
- ▶  $M$  possui duas fitas, é determinística e tem “entrada” como critério de aceitação.

# Teorema 3

## Método

Método:

- ▶  $\Gamma = \Gamma_1 \cup \Gamma_2$
- ▶  $Q = Q_1 \times Q_2 \times \{1, 2\}$
- ▶  $q_0 = (q_{01}, q_{02}, 1)$
- ▶  $F = \{(q_1, q_2, f) \in Q_1 \times Q_2 \times \{1, 2\} \mid q_1 \in F_1\}$
- ▶  $G = \{(q_1, q_2, f) \in Q_1 \times Q_2 \times \{1, 2\} \mid q_2 \in F_2\}$

## Teorema 3

## Método

- ▶  $\forall q \in (Q - (F \cup G)), \forall x \in \Gamma$ , suponha  $q = (q_1, q_2, f)$  e faça:
- ▶ Se  $f = 1$  então:
  - ①  $\forall \delta_1(q_1, x) = (q_3, y, D)$ , faça:  
 $\delta((q_1, q_2, 1), x, \epsilon) = ((q_3, q_2, 2), (y, D), (\epsilon, S))$
  - ②  $\forall \delta_1(q_1, x)$  não definida, faça:  
 $\delta((q_1, q_2, 1), x, \epsilon) = ((q_1, q_2, 2), (x, S), (\epsilon, S))$
- ▶ Se  $f = 2$  então:
  - ①  $\forall \delta_2(q_2, x) = (q_3, y, D)$ , faça:  
 $\delta((q_1, q_2, 2), \epsilon, x) = ((q_1, q_3, 1), (\epsilon, S)(y, D))$
  - ②  $\forall \delta_2(q_2, x)$  não definida, faça:  
 $\delta((q_1, q_2, 2), \epsilon, x) = ((q_1, q_2, 1), (\epsilon, S)(x, S))$

# Teorema 3

## Método

### Observações importantes:

- ▶ Se  $M_1$  entra num estado final, então  $M$  também entra num estado final. Logo, se  $M_1$  aceita então  $M$  também aceita;
- ▶ Se  $M_2$  entra num estado final, então  $M$  pára e rejeita; isto é garantido pela ausência de transições nos estados do conjunto  $G$ ;
- ▶ Se  $M_2$  pára e rejeita, não é necessário continuar com a simulação de  $M_1$ . Neste caso,  $M$  pode parar e aceitar imediatamente, uma vez que a cadeia de entrada não pertence à  $\bar{L}$  e por hipótese pertence à  $L$ ;
- ▶ Da mesma forma, se  $M_1$  pára e rejeita, não é necessário continuar com a simulação de  $M_2$ . Neste caso,  $M$  pode parar e rejeitar imediatamente, uma vez que a cadeia de entrada não pertence à  $L$  e por hipótese pertence à  $\bar{L}$ .

## Teorema 3

## Exemplo

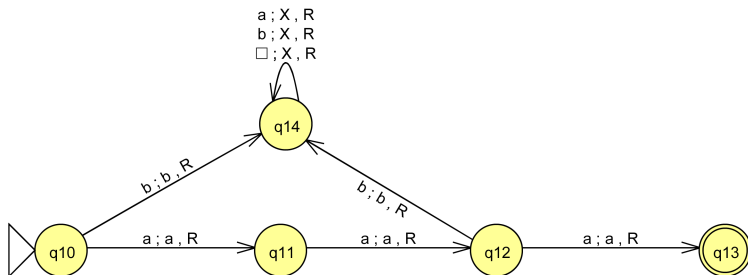
Suponha  $M_1$  tal que:

- ▶  $M_1$  é determinística;
- ▶  $L_1(M_1) = ACEITA(M_1) = aaa(a|b)^*$
- ▶  $REJEITA(M_1) = a|aa|ab(a|b)^*$
- ▶  $LOOP(M_1) = (aab|b)(a|b)^*$
- ▶  $ACEITA(M_1) \cup REJEITA(M_1) \cup LOOP(M_1) = \{a, b\}^*$
- ▶  $ACEITA(M_1) \cap REJEITA(M_1) \cap LOOP(M_1) = \emptyset$



## Teorema 3

## Exemplo



## Teorema 3

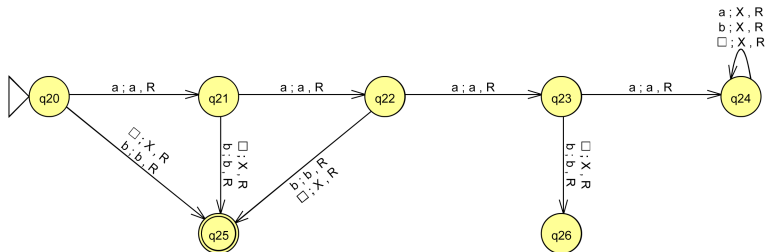
## Exemplo

Suponha  $M_2$  tal que:

- ▶  $M_2$  é determinística;
- ▶  $L_2(M_2) = \overline{L(M_1)} = ACEITA(M_2) = \epsilon|a|aa|(b|ab|aab)(a|b)^*$
- ▶  $REJEITA(M_2) = aaa|aaab(a|b)^*$
- ▶  $LOOP(M_2) = aaaa(a|b)^*$
- ▶  $ACEITA(M_2) \cup REJEITA(M_2) \cup LOOP(M_2) = \{a, b\}^*$
- ▶  $ACEITA(M_2) \cap REJEITA(M_2) \cap LOOP(M_2) = \emptyset$

## Teorema 3

## Exemplo



## Teorema 3

## Exemplo

		$M_2$		
		A	R	L
$M_1$	A	×	aaab aaa	aaaa
	R	abaa aa	×	×
	L	aabb	×	×

## Teorema 3

## Exemplo

aaab		aaa	
M <sub>1</sub>	M <sub>2</sub>	M <sub>1</sub>	M <sub>2</sub>
A	R	A	R
( $\epsilon, q_{10}, aaab$ )	( $\epsilon, q_{20}, aaab$ )	( $\epsilon, q_{10}, aaa$ )	( $\epsilon, q_{20}, aaa$ )
( $a, q_{11}, aab$ )	( $a, q_{21}, aab$ )	( $a, q_{11}, aa$ )	( $a, q_{21}, aa$ )
( $aa, q_{12}, ab$ )	( $aa, q_{22}, ab$ )	( $aa, q_{12}, aba$ )	( $aa, q_{22}, a$ )
( $aaa, q_{13}, b$ )	( $aaa, q_{23}, b$ )	( $aaa, q_{13}, \epsilon$ )	( $aaa, q_{23}, \epsilon$ )
✓	( $aaab, q_{26}, \epsilon$ )	✓	×
	×		

## Teorema 3

## Exemplo

abaa		aa	
$M_1$	$M_2$	$M_1$	$M_2$
R	A	R	A
$(\epsilon, q_{10}, abaa)$	$(\epsilon, q_{20}, abaa)$	$(\epsilon, q_{10}, aa)$	$(\epsilon, q_{20}, aa)$
$(a, q_{11}, baa)$	$(a, q_{22}, baa)$	$(a, q_{11}, a)$	$(a, q_{21}, a)$
×	$(ab, q_{23}, aa)$	$(aa, q_{12}, \epsilon)$	$(aa, q_{22}, \epsilon)$
	✓	×	$(aaX, q_{25}, \epsilon)$
			✓

## Teorema 3

## Exemplo

aaaa		aabb	
M <sub>1</sub>	M <sub>2</sub>	M <sub>1</sub>	M <sub>2</sub>
A	L	L	A
( $\epsilon, q_{10}, aaaa$ )	( $\epsilon, q_{20}, aaaa$ )	( $\epsilon, q_{10}, aabb$ )	( $\epsilon, q_{20}, aabb$ )
( $a, q_{11}, aaa$ )	( $a, q_{21}, aaa$ )	( $a, q_{11}, abb$ )	( $a, q_{21}, abb$ )
( $aa, q_{12}, aa$ )	( $aa, q_{22}, aa$ )	( $aa, q_{12}, bb$ )	( $aa, q_{22}, bb$ )
( $aaa, q_{13}, a$ )	( $aaa, q_{23}, a$ )	( $aab, q_{14}, b$ )	( $aab, q_{25}, b$ )
✓	( $aaaaa, q_{24}, \epsilon$ )	( $aabX, q_{14}, \epsilon$ )	✓
	( $aaaaaX, q_{24}, \epsilon$ )	( $aabXX, q_{14}, \epsilon$ )	
	( $aaaaaXX, q_{24}, \epsilon$ )	( $aabXXX, q_{14}, \epsilon$ )	
	...	...	

# Teorema 3

## Exemplo

Composição de  $M_1$  e  $M_2$ :

- ▶  $Q_1 = \{q_{10}, q_{11}, q_{12}, q_{13}, q_{14}\}$ ;
- ▶  $Q_2 = \{q_{20}, q_{21}, q_{22}, q_{23}, q_{24}, q_{25}, q_{26}\}$ ;
- ▶  $|Q| = 5 * 7 * 2 = 70$ ;
- ▶  $|F| = 1(q_{13}) * 7 * 2 = 14$ ;
- ▶  $|G| = 5 * 1(q_{25}) * 2 = 10$ ;
- ▶ Estado inicial  $q_0 = (q_{10}, q_{20}, 1)$ ;
- ▶ Próximo passo: definir as transições de  $70 - 14 - 10 = 46$  estados.



## Teorema 3

## Exemplo

Composição de  $M_1$  e  $M_2$ :

- ▶ Estado inicial  $q_0 = (q_{10}, q_{20}, 1)$ ;
- ▶ Como  $f = 1$  então:
  - ▶  $\delta((q_{10}, q_{20}, 1), a, \epsilon) = ((q_{11}, q_{20}, 2), (a, R), (\epsilon, S))$   
pois  $\delta_1(q_{10}, a) = (q_{11}, a, R)$ ;
  - ▶  $\delta((q_{10}, q_{20}, 1), b, \epsilon) = ((q_{14}, q_{20}, 2), (b, R), (\epsilon, S))$   
pois  $\delta_1(q_{10}, b) = (q_{14}, b, R)$ ;
  - ▶  $\delta((q_{10}, q_{20}, 1), \square, \epsilon) = ((q_{10}, q_{20}, 2), (\square, S), (\epsilon, S))$   
pois  $\delta_1(q_{10}, \square)$  não é definida;
  - ▶  $\delta((q_{10}, q_{20}, 1), X, \epsilon) = ((q_{10}, q_{20}, 2), (X, S), (\epsilon, S))$   
pois  $\delta_1(q_{10}, X)$  não é definida.

## Teorema 3

## Exemplo

Composição de  $M_1$  e  $M_2$ :

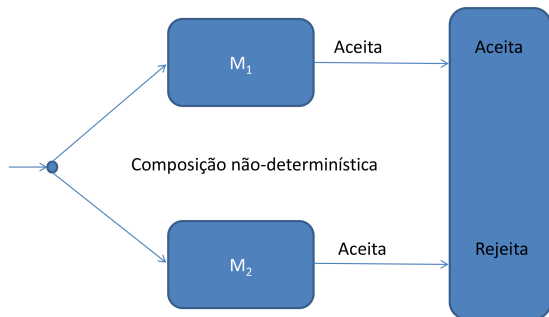
- ▶ Estado  $(q_{11}, q_{20}, 2)$ ;
- ▶ Como  $f = 2$  então:
  - ▶  $\delta((q_{11}, q_{20}, 2), \epsilon, a) = ((q_{11}, q_{21}, 1), (\epsilon, S), (a, R))$   
pois  $\delta_2(q_{20}, a) = (q_{21}, a, R)$ ;
  - ▶  $\delta((q_{11}, q_{20}, 2), \epsilon, b) = ((q_{11}, q_{25}, 1), (\epsilon, S), (b, R))$   
pois  $\delta_2(q_{20}, b) = (q_{25}, b, R)$ ;
  - ▶  $\delta((q_{11}, q_{20}, 2), \epsilon, \square) = ((q_{11}, q_{25}, 1), (\epsilon, S), (X, R))$   
pois  $\delta_2(q_{20}, \square) = (q_{25}, X, R)$ ;
  - ▶  $\delta((q_{11}, q_{20}, 2), \epsilon, X) = ((q_{11}, q_{20}, 1), (\epsilon, S), (X, S))$   
pois  $\delta_2(q_{20}, X)$  não é definida.

As transições dos demais estados são obtidas de forma similar.

# Teorema 3

## Método 2?

Construir  $M$  com uma única fita, a partir da composição não-determinística de  $M_1$  e de  $M_2$ :



# Teorema 3

## Método 2?

Suponha  $w \notin L$  tal que  $w \in LOOP(M_1)$  e  $w \in ACEITA(M_2)$ . Nesse caso,  $w$  possui duas seqüências distintas de movimentação em  $M$ :

- ▶ Na primeira,  $w$  faz  $M$  entrar em loop infinito;
- ▶ Na segunda,  $w$  é rejeitada por  $M$  (pois ela é aceita por  $M_2$ );
- ▶ Logo,  $w \in LOOP(M)$  e  $M$  não pára com qualquer entrada;
- ▶  $M$  não constitui prova de que  $L$  seja recursiva;
- ▶ Método 2 não funciona!
- ▶ Para ilustrar, considere a cadeia  $aab$  no exemplo anterior.

# Teorema 3

## Conclusões

- ▶ Toda cadeia  $w$  está em  $L$  ou  $\overline{L}$ ;
- ▶ Portanto, pelo menos uma das duas máquinas  $M_1$  e  $M_2$  sempre pára com  $w$  ( $M_1$  aceitando ou  $M_2$  aceitando);
- ▶ Como  $M$  pára sempre quando  $M_1$  ou  $M_2$  aceitam, então  $M$  sempre pára;
- ▶  $M$  aceita todas as cadeias de  $L$ ;
- ▶  $M$  rejeita todas as cadeias de  $\overline{L}$ .
- ▶  $L$  é recursiva.

$L \times \bar{L}$ 

## Possibilidades

Considere que as linhas representam  $L$  e as colunas representam  $\bar{L}$ . As seguintes combinações, e apenas essas, são possíveis:

	Recursiva	RE não-recursiva	Não-RE
Recursiva	✓	-	-
RE não-recursiva	-	-	✓
Não-RE	-	✓	✓

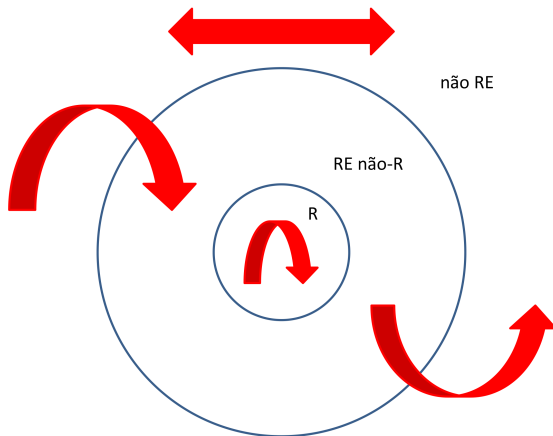
- ▶ O Teorema 2 exclui as possibilidades Recursiva/RE não-recursiva, Recursiva/Não-RE, RE não-recursiva/Recursiva e Não-RE/Recursiva (pois o complemento de uma linguagem recursiva é também uma linguagem recursiva);
- ▶ O Teorema 3 exclui a possibilidade RE não-recursiva/RE não-recursiva (pois se ambas são RE então o complemento é uma linguagem recursiva).

$L \times \bar{L}$ 

## Problemas e seus complementos:

- ▶ O complemento de um problema solucionável é sempre um problema solucionável;
  - ▶ Não há loop com nenhuma cadeia de  $\Sigma^*$ ;
- ▶ O complemento de um problema estritamente parcialmente solucionável é totalmente insolúvel:
  - ▶ Como existe pelo menos uma cadeia  $w \in \Sigma^* - L$  que provoca loop, em  $L' = \Sigma^* - L$  ela não será aceita;
- ▶ O complemento de um problema totalmente insolúvel pode ser estritamente parcialmente solucionável ou totalmente insolúvel:
  - ▶ Como existe pelo menos uma cadeia  $w \in L$  que provoca loop, em  $L' = \Sigma^* - L$  ela provoca loop também;
  - ▶ Se existe uma cadeia  $w \in \Sigma^* - L$  que provoca loop, em  $L' = \Sigma^* - L$  ela provoca loop também;

$$L \times \bar{L}$$





$L \times \bar{L}$ 

## Exemplo

Considere a linguagem  $L_d$ :

- ▶ Conforme o Teorema 1,  $L_d$  é não-RE;
- ▶ Conseqüentemente,  $\bar{L}_d$  deve ser RE não-recursiva ou não-RE;
- ▶ Certamente  $\bar{L}_d$  não é recursiva;
- ▶  $L_d = \{w_i \mid w_i \notin L(M_i)\}$ ;
- ▶  $\bar{L}_d = \{w_i \mid w_i \in L(M_i)\}$ ;
- ▶ Conforme será provado mais adiante,  $\bar{L}_d$  é RE não-recursiva.

# Conceito

- ▶ Máquinas de Turing incorporam os programas que elas executam na sua definição;
- ▶ Como transformar uma Máquina de Turing em dados para outra Máquina de Turing processar?
- ▶ Resposta: Máquina de Turing Universal ( $U$ );
- ▶ Aceita como entrada a descrição de uma outra Máquina de Turing e a entrada que essa outra máquina deve processar;
- ▶ Podemos considerar, sem perda de generalidade, que a Máquina de Turing a ser simulada é determinística;
- ▶ Simula a máquina descrita e produz como resultado o mesmo resultado que a máquina simulada produziria;
- ▶ É universal pois é capaz de executar qualquer algoritmo.

# Convenções

$U$  possui quatro fitas:

- ▶ A primeira fita contém a descrição da máquina a ser simulada ( $\langle M \rangle$ ) e a sua correspondente entrada ( $w$ );
- ▶ A segunda fita é usada para simular a fita da máquina a ser simulada ( $M$ ); símbolos  $X_i, i \geq 1$ , são denotados  $0^i$  e são separados na fita pelo símbolo 1; 0 representa 0, 00 representa 1 e 000 representa  $B$ ;
- ▶ A terceira fita é usada para representar o estado de  $M$ ; estados  $q_i, i \geq 1$ , são denotados  $0^i$ ;
- ▶ A quarta fita é usada para rascunho.

# Convenções

Suponha  $\langle M \rangle = C_111C_211\dots11C_{n-1}11C_n$  e  $w = 01011\dots$

Então:

Fita 1 <M>w	C <sub>1</sub> 11C <sub>2</sub> 11... 11C <sub>n-1</sub> 11C <sub>n</sub> 11101011 ...												
	↑												
Fita 2 <w>	0	1	0	0	1	0	1	0	0	1	0	0	1 ...
	↑												
Fita 3 q <sub>i</sub>	0												
	↑												
Fita 4 aux													
	↑												

# Inicialização

- 1)  $U$  verifica se  $\langle M \rangle$  corresponde à descrição de uma Máquina de Turing válida; em caso negativo,  $U$  pára e rejeita a entrada (descrições inválidas representam máquinas que aceitam a linguagem vazia, portanto toda entrada deve ser rejeitada);
- 2)  $U$  copia a cadeia  $w$  da primeira para a segunda fita, codificando os seus símbolos da maneira apropriada (seqüências de 0 separadas pelo símbolo 1);
- 3)  $U$  posiciona a cabeça de leitura no primeiro símbolo da segunda fita;
- 4) Como, por convenção, o estado inicial de  $M$  é  $q_1$ ,  $U$  grava o símbolo  $0^1 = 0$  na terceira fita.

# Operação

- 5) Se o símbolo gravado na posição corrente da segunda fita é  $0^j$  (símbolo corrente de  $M$ ) e a cadeia contida na terceira fita é  $0^i$  (estado corrente de  $M$ ), então  $U$  procura, na primeira fita, pela cadeia  $0^i 10^j 10^k 10^l 10^m$ , a qual representa a transição que seria executada por  $M$  nessa configuração (lembre-se que  $M$  é determinístico);
- 6) Caso não exista tal transição, então  $M$  pára e portanto  $U$  deve parar também;
- 7) Caso exista tal transição, então  $U$ :
  - ▶ Modifica o símbolo corrente de  $M$  na segunda fita (de  $0^j$  para  $0^l$ )
  - ▶ Modifica o estado corrente de  $M$  na terceira fita (de  $0^i$  para  $0^k$ );
  - ▶ Desloca a cabeça de leitura na segunda fita para o próximo símbolo da esquerda (se  $m = 1$ ) ou da direita (se  $m = 2$ ); lembre-se que os símbolos são cadeias de 0 separadas por 1;
  - ▶ Se o novo estado for  $00$  (que representa  $q_2$ , o estado final de  $M$ ), então  $U$  pára e aceita a entrada.

# Conclusão

- ▶  $U$  simula  $M$  com a entrada  $w$ ;
- ▶  $U$  pára e aceita  $\langle M \rangle w \Leftrightarrow M$  pára e aceita  $w$ ;
- ▶  $U$  pára e rejeita  $\langle M \rangle w \Leftrightarrow M$  pára e rejeita  $w$ ;
- ▶  $U$  entra em loop infinito com  $\langle M \rangle w \Leftrightarrow M$  entra em loop infinito com  $w$ ;

# Conceito

Suponha que  $\langle M \rangle$  representa uma codificação de uma MT  $M$  sobre o alfabeto  $\{0, 1\}$ . Suponha que  $w$  é uma cadeia sobre esse mesmo alfabeto. A “linguagem universal”:

$$L_u = \{\langle M \rangle w \mid M \text{ é uma MT que aceita } w\}$$

é aceita por  $U$ .

- ▶ O problema de determinar se uma Máquina de Turing  $M$  aceita a cadeia  $w$  pode ser traduzido...
- ▶ Pelo problema de determinar se  $\langle M \rangle w \in L_u \dots$
- ▶ Ou seja, determinar se  $\langle M \rangle w \in L(U)$ ;
- ▶  $L_u = L(U)$  é recursiva, RE não-recursiva ou não-RE?



# Teorema 4

$L_u$  é RE não-recursiva

$L_u$  é RE:

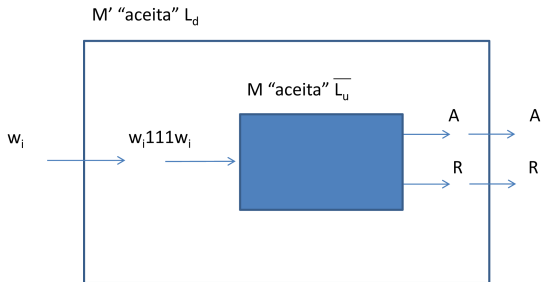
- ▶  $U$  é uma Máquina de Turing que aceita  $L_u$ .

## Teorema 4

 $L_u$  é RE não-recursiva $L_u$  não é recursiva (Hopcroft):

- ▶ Suponha que  $L_u$  seja recursiva;
- ▶ Então,  $\overline{L_u}$  também é recursiva;
- ▶ Considere que  $M$  é tal que  $L(M) = \overline{L_u}$ ;
- ▶ Seja  $M'$  tal que, com a entrada  $w$ :
  - ▶  $M'$  transforma  $w$  em  $w111w$ ;
  - ▶  $M'$  executa  $M$  com a entrada  $w111w$ ;
  - ▶ Considere  $w = w_i = \langle M_i \rangle$ ;
  - ▶  $M$  aceita  $w_i111w_i$  se e somente se  $w_i \notin L(M_i)$ , ou seja, se  $w_i \in L_d$ ; caso contrário  $M$  rejeita  $w_i111w_i$ ;
  - ▶ Suponha que  $M'$  aceita quando  $M$  aceita e rejeita quando  $M$  rejeita;
  - ▶ Logo,  $M'$  decide  $L_d$ ;
  - ▶ Como  $L_d$  é não-RE, a hipótese é falsa e  $L_u$  não pode ser recursiva.

## Teorema 4

 $L_u$  é RE não-recursiva

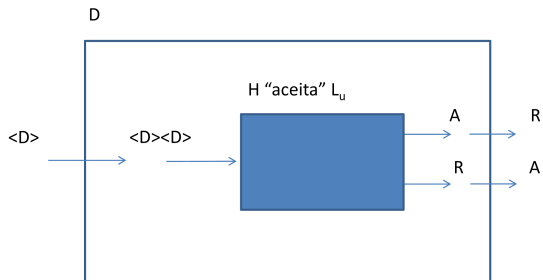
# Teorema 4

$L_u$  é RE não-recursiva

$L_u$  não é recursiva (Sipser):

- ▶ Suponha que  $L_u$  seja recursiva e que  $H(\langle M \rangle w)$  decida  $L_u$ ;
- ▶ Considere a máquina  $D$ :
  - ▶  $D$  aceita como entrada  $\langle M \rangle$ ;
  - ▶  $D$  executa  $H$  com a entrada  $\langle M \rangle \langle M \rangle$ ;
  - ▶  $D$  aceita se  $H$  rejeita e rejeita se  $H$  aceita.
- ▶ Considere que  $D$  receba como entrada  $\langle D \rangle$ ;
- ▶ Se  $D$  aceita  $\langle D \rangle$  (pela execução de  $H$ ) então  $D$  rejeita  $\langle D \rangle$ ;
- ▶ Se  $D$  rejeita  $\langle D \rangle$  (pela execução de  $H$ ) então  $D$  aceita  $\langle D \rangle$ ;
- ▶ Em qualquer caso, uma contradição; logo, a hipótese é falsa e  $L_u$  não é recursiva.

## Teorema 4

 $L_u$  é RE não-recursiva

## Teorema 4

## Diagonalização

Para cada par linha/coluna  $(i, j)$ , a tabela indica se  $M_i$  aceita  $w_j$ :

	1	2	3	4	...
1	✓			✓	...
2	✓	✓	✓	✓	...
3					...
4	✓	✓			...
...	...	...	...	...	...

## Teorema 4

## Diagonalização

Para cada par linha/coluna  $(i, j)$ , a tabela indica o resultado produzido por  $H$ :

	1	2	3	4	...
1	✓	×	×	✓	...
2	✓	✓	✓	✓	...
3	×	×	×	×	...
4	✓	✓	×	×	...
...	...	...	...	...	...

## Teorema 4

## Diagonalização

Se existisse a Máquina de Turing  $D$ , a contradição aconteceria na posição  $(\langle D \rangle, D)$ :

	1	2	3	4	...	D	...
1	✓	×	×	✓	...	✓	...
2	✓	✓	✓	✓	...	✓	...
3	×	×	×	×	...	×	...
4	✓	✓	×	×	...	✓	...
...	...	...	...	...	...	...	...
$\langle D \rangle$	×	×	✓	✓	...	?	...
...	...	...	...	...	...	...	...



# Teorema 4

## Conclusão

Se houvesse solução para o problema de determinar se uma Máquina de Turing não aceita uma cadeia qualquer ( $\overline{L_u}$ ), haveria solução para o problema, mais simples, de determinar se uma Máquina de Turing não aceita uma cadeia específica ( $L_d$ ).

## Teorema 5

 $\overline{L_d}$  é RE não-recursiva

$$\overline{L_d} = \{w_i \mid w_i \in L(M_i)\}$$

$\overline{L_d}$  é RE pois é aceita pela seguinte Máquina de Turing  $M$ :

- ▶  $M$  aceita  $w_i$  como entrada;
- ▶  $M$  transforma  $w_i$  em  $w_i111w_i$  e simula a Máquina de Turing Universal  $U$  com essa cadeia;
- ▶ Se  $U$  pára e aceita, então  $M$  pára e aceita;
- ▶ Se  $U$  pára e rejeita, então  $M$  pára e rejeita;
- ▶ Se  $U$  entra em loop, então  $M$  entra em loop.

Conclusão:  $M$  aceita  $\overline{L_d}$  e portanto  $\overline{L_d}$  é RE.

## Teorema 5

 $\overline{L_d}$  é RE não-recursiva

$$\overline{L_d} = \{w_i \mid w_i \in L(M_i)\}$$

$\overline{L_d}$  é não-recursiva pois:

- ▶ Se  $\overline{L_d}$  fosse recursiva, então  $L_d$  também deveria ser (pelo Teorema 2);
- ▶ Mas sabemos que  $L_d$  é não-RE (e conseqüente não-recursiva).
- ▶ Logo, a hipótese é falsa e  $\overline{L_d}$  não é recursiva.

Conclusão:  $\overline{L_d}$  é RE não-recursiva.

# Linguagens e complementos

## Resumo até este ponto

- ▶  $L_d = \{w_i \mid w_i \notin L(M_i)\}$  é não-RE;
- ▶  $\overline{L_d} = \{w_i \mid w_i \in L(M_i)\}$  é RE não-recursiva;
- ▶  $L_u = \{\langle M \rangle w \mid M \text{ é uma MT que aceita } w\}$  é RE não-recursiva;
- ▶  $\overline{L_u} = \{\langle M \rangle w \mid M \text{ é uma MT que não aceita } w\}$  é não-RE.

# Conceito

- ▶ Técnica para determinar a decidibilidade de um problema a partir de outro cuja natureza é conhecida;
- ▶ Uma redução é uma maneira de converter um problema em outro de tal forma que uma solução para o segundo problema possa ser usada para resolver o primeiro problema;

# Exemplos

Uma solução para  $P_2$  é uma solução para  $P_1$ :

- ▶  $P_1$ : orientar-se numa nova cidade;  
 $P_2$ : obter um mapa;
- ▶  $P_1$ : viajar de São Paulo para New York;  
 $P_2$ : comprar uma passagem de avião;
- ▶  $P_1$ : comprar uma passagem de avião;  
 $P_2$ : dispor do dinheiro necessário;
- ▶  $P_1$ : dispor do dinheiro necessário;  
 $P_2$ : conseguir um trabalho.

# Exemplos

Uma solução para  $P_2$  é uma solução para  $P_1$ :

- ▶  $P_1$ : medir a área de um retângulo;  
 $P_2$ : medir o seu comprimento e largura;
- ▶  $P_1$ : resolver um sistema de equações lineares;  
 $P_2$ : inverter uma matriz;
- ▶  $P_1$ : provar que uma linguagem  $L$  não é regular;  
 $P_2$ : encontrar  $w = xyz \in L$  tal que  $|w| > n$ ,  $|y| \geq 1$  e, para algum  $i \geq 0$ ,  $xy^iz \notin L$ ;
- ▶  $P_1$ : construir um analisador sintático determinístico para uma linguagem  $L$ ;  
 $P_2$ : obter uma gramática  $LR(k)$  que gera  $L$ .

# Conceito

Se existe uma redução de  $P_1$  para  $P_2$ , então diz-se que:

- ▶  $P_1$  “não é mais difícil do que”  $P_2$ ;
- ▶  $P_2$  “é no mínimo tão difícil quanto”  $P_1$ .

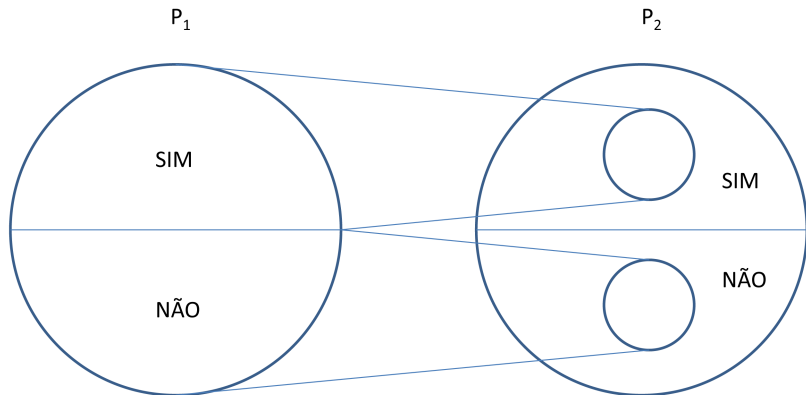


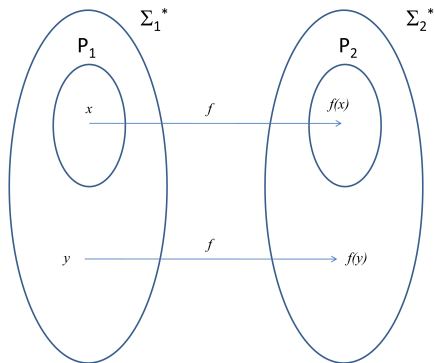
# Definição

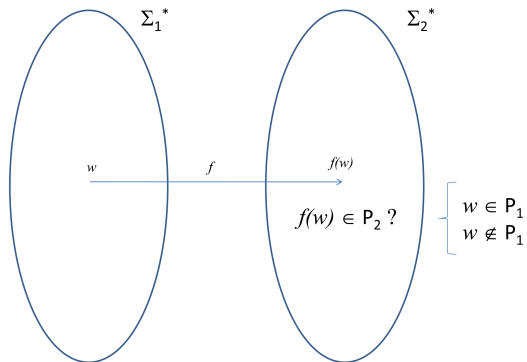
- ▶ Uma redução de  $P_1$  para  $P_2$  é uma função **total**  $f$  que mapeia sentenças de  $P_1$  para sentenças de  $P_2$ :

$$w \in P_1 \Leftrightarrow f(w) \in P_2$$

- ▶ Uma redução também pode ser vista como uma MT (algoritmo) que mapeia sentenças de  $P_1$  em sentenças de  $P_2$ ;
- ▶ A função de mapeamento  $f$  **não necessita ser sobrejetora nem injetora**.

Redução de  $P_1$  para  $P_2$ 

Redução de  $P_1$  para  $P_2$ 

Redução de  $P_1$  para  $P_2$ 

# Teorema 6

## Enunciados

Se  $f$  é uma redução de  $P_1$  para  $P_2$ , então:

- 1 Se  $P_1$  é indecidível, então  $P_2$  também é indecidível;
- 2 Se  $P_1$  é não-RE, então  $P_2$  também é não-RE.

# Teorema 6

$P_1$  indecidível  $\Rightarrow P_2$  indecidível

Suponha que  $P_2$  seja decidível. Então é possível combinar o algoritmo que decide  $P_2$  com a redução  $f$  para obter um algoritmo que decide  $P_1$ .

- ▶ Seja  $w \in \Sigma_1^*$  ( $\Sigma_1$  é o alfabeto de  $P_1$ );
- ▶ Obter  $f(w)$ ;
- ▶ Como  $P_2$  é decidível, por hipótese, é possível determinar se  $f(w) \in P_2$ ;
- ▶ Em caso afirmativo, e como  $f$  é uma redução, é certo que  $w \in P_1$ ;
- ▶ Em caso negativo, e como  $f$  é uma redução, é certo que  $w \notin P_1$ ;
- ▶ Em qualquer caso é possível determinar se  $w \in P_1$ ;
- ▶ Logo,  $P_1$  seria decidível;
- ▶ Mas isso contrária a hipótese, portanto  $P_2$  não pode ser decidível.

# Teorema 6

$P_1$  não-RE  $\Rightarrow P_2$  não-RE

Suponha que  $P_2$  seja RE. Então é possível combinar a MT  $M_2$  que aceita  $P_2$  com a redução  $f$  para obter uma MT  $M_1$  que aceita  $P_1$ .

- ▶ Seja  $w \in \Sigma_1^*$ ;
- ▶ Obter  $f(w)$ ;
- ▶ Executar  $M_2$  com a entrada  $f(w)$ ;
- ▶ Se  $M_2$  aceita  $f(w)$ , então  $w \in P_1$ ;
- ▶ Se  $M_2$  não aceita  $f(w)$  ( $M_2$  pára e rejeita ou entra em loop), então  $w \notin P_1$ ;
- ▶ Logo, é possível construir  $M_1$  que aceita  $P_1$ ;
- ▶ Mas isso contrária a hipótese, portanto  $P_2$  não pode ser RE.

# Teorema 6

Enunciados com corolários

Se  $f$  é uma redução de  $P_1$  para  $P_2$ , então:

- 1 Se  $P_1$  é indecidível, então  $P_2$  também é indecidível;  
Se  $P_2$  é decidível, então  $P_1$  também é decidível;
- 2 Se  $P_1$  é não-RE, então  $P_2$  também é não-RE;  
Se  $P_2$  é RE, então  $P_1$  também é RE.



# Teorema 6

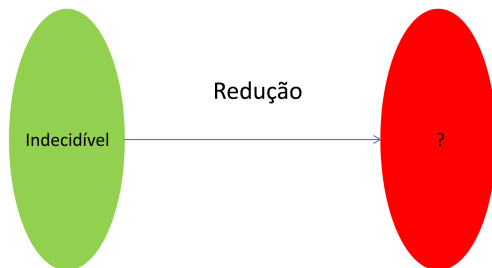
## Estratégias

Aplicação do teorema (parte 1):

- ▶ Para demonstrar que um problema  $P_2$  de natureza desconhecida é indecidível:
  - ▶ Obter uma redução de um problema  $P_1$ , reconhecidamente indecidível, para  $P_2$ ;
- ▶ Para demonstrar que um problema  $P_1$  de natureza desconhecida é decidível:
  - ▶ Obter uma redução de  $P_1$  para um problema  $P_2$ , reconhecidamente decidível;

# Teorema 6

## Estratégias



# Teorema 6

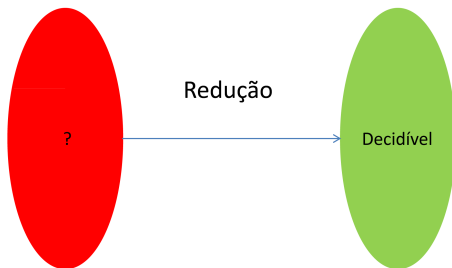
## Estratégias

E o contrário? Haveria interesse em reduzir um problema  $P_1$  de natureza desconhecida para um problema  $P_2$  reconhecidamente indecidível?

- ▶ Qual o interesse em fazer isso?
- ▶ Como  $P_2$  é indecidível, tal fato não permite obter nenhuma conclusão em relação à  $P_1$ ;
- ▶ Não adianta para nada, portanto.

# Teorema 6

## Estratégias



# Teorema 6

## Estratégias

E o contrário? Haveria interesse em reduzir um problema  $P_1$  reconhecidamente decidível para um problema  $P_2$  de natureza desconhecida?

- ▶ Só seria útil se fosse possível obter  $f^{-1}$  (a função de redução inversa);
- ▶ Nesse caso, dada uma instância  $w \in P_2$  seria possível combinar a aplicação de  $f^{-1}$  com a decisão de  $P_1$  para determinar se  $w \in P_1$ ;
- ▶ Vale lembrar que  $f$  não é necessariamente injetora e nem, principalmente, sobrejetora, o que dificulta a obtenção de  $f^{-1}$  com as características necessárias;
- ▶ Mas essa estratégia recai exatamente no caso direto;
- ▶ Não adianta nada, portanto.

# Teorema 6

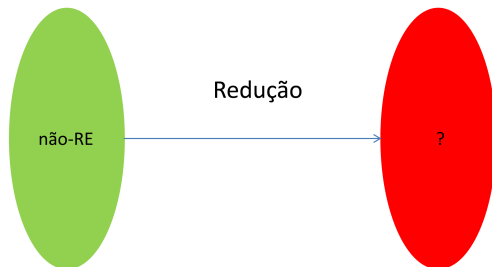
## Estratégias

Aplicação do teorema (parte 2):

- ▶ Para demonstrar que um problema  $P_2$  de natureza desconhecida é não-RE:
  - ▶ Obter uma redução de um problema  $P_1$ , reconhecidamente não-RE, para  $P_2$ ;
- ▶ Para demonstrar que um problema  $P_1$  de natureza desconhecida é RE:
  - ▶ Obter uma redução de  $P_1$  para um problema  $P_2$ , reconhecidamente RE;

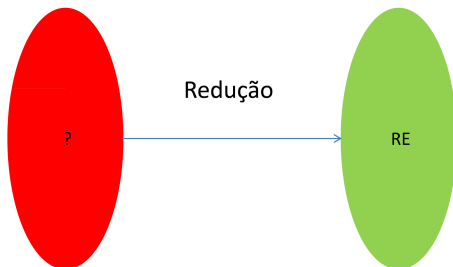
# Teorema 6

## Estratégias



# Teorema 6

## Estratégias





Reduções com  $L_u$  e  $L_d$ 

- 1  $L_u$  é indecidível (RE não-recursiva);
- 2  $L_d$  é não-RE;
- 3  $L_u$  pode ser usada para demonstrar que um problema  $P$  qualquer (RE ou não-RE) é indecidível:
  - ▶ Basta obter uma redução de  $L_u$  para  $P$ ;
- 4  $L_d$  pode ser usada para demonstrar que um problema  $P$  é não-RE:
  - ▶ Basta obter uma redução de  $L_d$  para  $P$ ;
- 5  $L_d$  não pode ser usada para demonstrar a indecidibilidade de um problema que é RE porém é não-recursivo (pois  $L_d$  é não-RE e só reduz para  $P$  não-RE); para esses casos deve-se usar  $L_u$ ;
- 6  $L_u$  não pode ser usada para demonstrar que um problema é não-RE (pois  $L_u$  é RE não-recursivo e só reduz para  $P$  não-recursivo, sem discriminar se  $P$  é RE ou não-RE); para esses casos deve-se usar  $L_d$ .

# Conceito

Suponha que  $\langle M \rangle$  representa uma codificação de  $M$  sobre o alfabeto  $\{0, 1\}$ . Suponha que  $w$  é uma cadeia sobre esse mesmo alfabeto. A “linguagem da parada” é definida como:

$$PARA_{MT} = \{ \langle M, w \rangle \mid M \text{ pára com a entrada } w \}$$

- ▶ Corresponde ao problema fundamental de determinar se um programa qualquer pára com uma entrada qualquer;
- ▶  $PARA_{MT}$  é decidível ou indecidível?

## Teorema 7

$PARA_{MT}$  é indecidível através de redução

Função  $f$  que reduz  $L_u$  para  $PARA_{MT}$ :

- ▶  $L_u = \{\langle M, w \rangle \mid M \text{ aceita a entrada } w\}$
- ▶  $PARA_{MT} = \{\langle M', w \rangle \mid M' \text{ pára com a entrada } w\}$
- ▶ A redução  $f$  é computada pela seguinte MT:
  - ▶ A partir da entrada  $\langle M, w \rangle$ , construir  $M'$  de tal forma que  $M'$  simula  $M$  com a entrada  $w$ ;
  - ▶ Se  $M$  aceita  $w$ , então  $M'$  aceita  $w$ ;
  - ▶ Se  $M$  rejeita  $w$ , então  $M'$  entra em loop (e, naturalmente, se  $M$  entra em loop, então  $M'$  também entra em loop);
- ▶  $\langle M, w \rangle \in L_u \Leftrightarrow \langle M', w \rangle \in PARA_{MT}$ ;
- ▶ Como  $L_u$  é indecidível,  $PARA_{MT}$  também é indecidível.

## Teorema 7

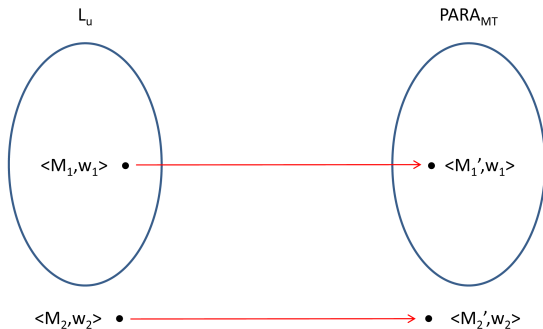
$PARA_{MT}$  é indecidível através de redução

Obtenção de  $\langle M', w \rangle$  a partir de  $\langle M, w \rangle$ :

- ▶  $M'$  simula  $M$  com a entrada  $w$ ;
- ▶ Se  $M$  aceita  $w$ ,  $\langle M, w \rangle \in L_u$  e  $M'$  deve aceitar  $w$ , pois dessa forma  $\langle M', w \rangle \in PARA_{MT}$ ;
- ▶ Se  $M$  rejeita  $w$ ,  $\langle M, w \rangle \notin L_u$  e  $M'$  deve entrar em loop infinito, pois dessa forma  $\langle M', w \rangle \notin PARA_{MT}$ ;
- ▶ Se  $M$  entra em loop infinito com  $w$ ,  $\langle M, w \rangle \notin L_u$  e  $M'$  entra automaticamente em loop infinito também. Portanto,  $\langle M', w \rangle \notin PARA_{MT}$ ;

Logo,  $\langle M, w \rangle \in L_u \Leftrightarrow \langle M', w \rangle \in PARA_{MT}$

## Teorema 7

 $PARA_{MT}$  é indecidível através de redução

## Teorema 7

 $PARA_{MT}$  é RE

Basta simular  $M$  com a entrada  $w$  e gerar, na saída, o mesmo resultado da simulação.

- ▶  $PARA_{MT} = \{\langle M, w \rangle \mid M \text{ pára com a entrada } w\}$  é RE não-recursiva, portanto o problema é parcialmente solucionável;
- ▶  $\overline{PARA_{MT}} = \{\langle M, w \rangle \mid M \text{ entra em loop com a entrada } w\}$ , no entanto, é não-RE, e portanto completamente insolúvel.

# Teorema 7

$PARA_{MT}$  é indecidível através de contradição

Suponha que  $PARA_{MT}$  é decidível. Então, a partir da MT  $R$  que decide  $PARA_{MT}$ , é possível obter uma outra MT  $S$  que decide  $L_u$ :

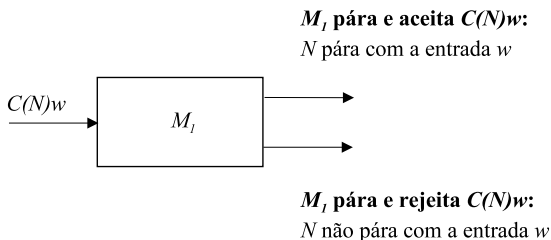
- ▶ Executar  $R$  sobre a entrada  $\langle M, w \rangle$ ;
- ▶ Se  $R$  rejeita,  $S$  também rejeita;
- ▶ Se  $R$  aceita, simular  $M$  com a entrada  $w$  até  $M$  parar;
- ▶ Se  $M$  aceita,  $S$  também aceita;
- ▶ Se  $M$  rejeita,  $S$  também rejeita.

Se  $R$  decide  $PARA_{MT}$ , então  $S$  decide  $L_u$ . Como é sabido que  $L_u$  é indecidível, a hipótese de que  $R$  existe é falsa e  $PARA_{MT}$  é indecidível.

## Teorema 7

$PARA_{MT}$  é indecidível através de diagramas

Supor que  $PARA_{MT}$  é decidível. Então existe  $M_1$ :





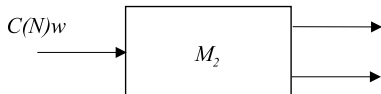
## Teorema 7

$PARA_{MT}$  é indecidível através de diagramas

Construir  $M_2$  a partir de  $M_1$ :

**$M_2$  executa uma seqüência infinita  
de movimentações:**

$N$  pára com a entrada  $w$



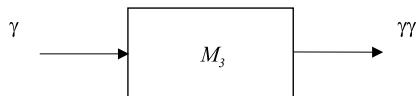
**$M_2$  pára e rejeita  $C(N)w$ :**

$N$  não pára com a entrada  $w$

# Teorema 7

$PARA_{MT}$  é indecidível através de diagramas

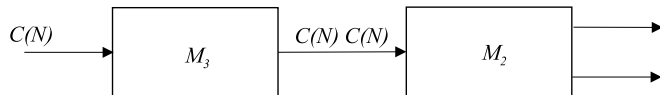
Construir  $M_3$ :



## Teorema 7

$PARA_{MT}$  é indecidível através de diagramas

Combinar  $M_3$  e  $M_2$ :



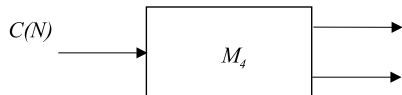
**$M_2$  executa uma seqüência infinita de movimentações:**  
 $N$  pára com a entrada  $C(N)$

**$M_2$  pára:**  
 $N$  não pára com a entrada  $C(N)$

## Teorema 7

$PARA_{MT}$  é indecidível através de diagramas

Renomear para  $M_4$ :



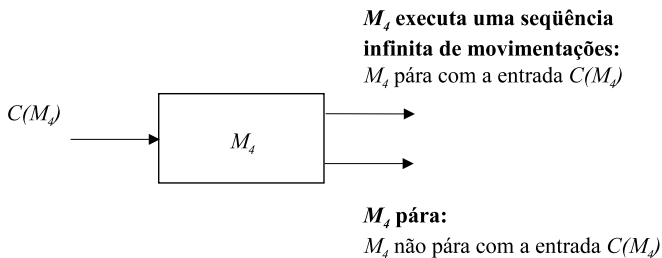
**$M_4$  executa uma seqüência  
infinita de movimentações:**  
 $N$  pára com a entrada  $C(N)$

**$M_4$  pára:**  
 $N$  não pára com a entrada  $C(N)$

## Teorema 7

$PARA_{MT}$  é indecidível através de diagramas

Fornecer para  $M_4$  a sua própria descrição:



# Teorema 7

$PARA_{MT}$  é indecidível através de diagramas

Conclusão:

- ▶ Por um lado, temos a informação de que, ao analisar a cadeia  $C(M_4)$ , se a máquina  $M_4$  parar, então  $M_4$  executa uma seqüência infinita de movimentações;
- ▶ Por outro, que ao analisar a cadeia  $C(M_4)$ , se  $M_4$  não parar, então  $M_4$  pára. Tem-se, portanto, uma contradição;
- ▶ Logo, a hipótese inicial não é válida, ou seja, não existe  $M_1$  que decida  $PARA_{MT}$ ;
- ▶  $PARA_{MT}$  é indecidível.

# Linguagens e complementos

Resumo até este ponto

- ▶  $L_d = \{w_i \mid w_i \notin L(M_i)\}$  é não-RE;
- ▶  $\overline{L_d} = \{w_i \mid w_i \in L(M_i)\}$  é RE não-recursiva;
- ▶  $L_u = \{\langle M \rangle w \mid M \text{ é uma MT que aceita } w\}$  é RE não-recursiva;
- ▶  $\overline{L_u} = \{\langle M \rangle w \mid M \text{ é uma MT que não aceita } w\}$  é não-RE;
- ▶  $PARA_{MT} = \{\langle M', w \rangle \mid M' \text{ pára com a entrada } w\}$  é RE não-recursiva;
- ▶  $\overline{PARA_{MT}} = \{\langle M', w \rangle \mid M' \text{ entra em loop com a entrada } w\}$  é não-RE.

# Definições

Considere  $\langle M \rangle$  como a codificação de uma MT  $M$  sobre o alfabeto  $\{0, 1\}$ .  
Então:

- ▶  $L_e = \{\langle M \rangle \mid L(M) = \emptyset\}$
- ▶  $L_{ne} = \{\langle M \rangle \mid L(M) \neq \emptyset\}$
- ▶  $L_e = \overline{L_{ne}}$



$L_{ne}$  é RE

Teorema: A linguagem  $L_{ne}$  é recursivamente enumerável.

Prova:

1. Construir uma MT  $M$  que aceita como entrada a codificação de uma outra MT  $M'$ ;
2.  $M$  opera de forma não-determinística, fazendo escolhas de cadeias arbitrárias para serem testadas em  $M'$ ;
3. Em cada ramo da sua execução não-determinística,  $M$  gera uma cadeia e testa se  $M'$  aceita a mesma;
4. Para isso,  $M$  simula a máquina  $U$  que aceita a linguagem  $L_u$ ;
5. Se algum caminho de  $M'$  for de aceitação, então  $M$  pára e aceita a sua entrada ( $M'$ );

$L_{ne}$  é RE

Exemplo: Geração não-determinística de cadeias arbitrárias sobre o alfabeto  $\{a, b, c\}$  para serem posteriormente testadas:



$L_{ne}$  é RE

Em resumo:

- ▶ Se  $M'$  aceita alguma cadeia,  $M$  “adivinha” essa cadeia e aceita  $M'$ ;
- ▶ Se  $M'$  não aceita nenhuma cadeia, então não há cadeia que conduza à aceitação em  $M'$  e  $M$  não aceita  $M'$  (nesse caso,  $M$  pode rejeitar  $M'$  ou entrar em loop);
- ▶ Portanto,  $L(M) = L_{ne}$ .

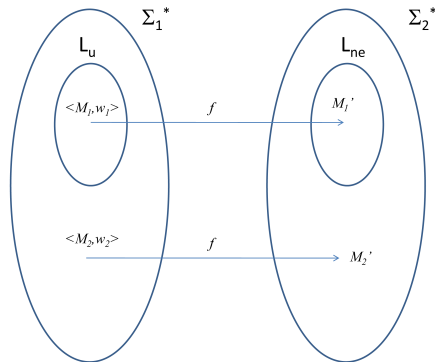
# $L_{ne}$ não é recursiva

Idéia geral:

- ▶ Fazer uma redução de  $L_u$  para  $L_{ne}$ ;
- ▶ Construir  $M'$  a partir de  $\langle M, w \rangle$  tal que:
  - ▶ Se  $w \in L(M)$ , então  $L(M') \neq \emptyset$ ;
  - ▶ Se  $w \notin L(M)$ , então  $L(M') = \emptyset$ ;
- ▶  $M'$  ignora a sua entrada e simula  $M$  com a entrada  $w$ ;
- ▶ Se  $M$  aceita  $w$ ,  $M'$  também aceita a sua entrada, qualquer que seja ela.

$L_{ne}$  não é recursiva

- ▶  $M_1$  aceita  $w_1 \Rightarrow \langle M_1, w_1 \rangle \in L_u \Rightarrow L(M'_1) \neq \emptyset \Rightarrow \langle M'_1 \rangle \in L_{ne}$ ;
- ▶  $M_2$  não aceita  $w_2 \Rightarrow \langle M_2, w_2 \rangle \notin L_u \Rightarrow L(M'_2) = \emptyset \Rightarrow \langle M'_2 \rangle \notin L_{ne}$ ;



# $L_{ne}$ não é recursiva

Teorema: A linguagem  $L_{ne}$  não é recursiva.

Prova:

1. É suficiente provar a existência de um algoritmo que efetua a redução de  $L_u$  para  $L_{ne}$ ;
2. O algoritmo deve mapear  $\langle M, w \rangle$  em  $M'$  de tal forma que  $w \in L(M) \Leftrightarrow L(M') \neq \emptyset$ ;
3. A construção de  $M'$  a partir de  $\langle M, w \rangle$  é detalhada a seguir;

$L_{ne}$  não é recursiva

4.  $M'$  ignora a sua entrada  $x$ , qualquer que seja ela.  $M'$  substitui  $x$  por  $\langle M, w \rangle$ , tomando o cuidado de trocar os símbolos finais de  $x$  por brancos, caso  $|x| > |\langle M, w \rangle|$ ;
5.  $M'$  posiciona a cabeça de leitura/escrita sobre o primeiro símbolo da cadeia  $\langle M, w \rangle$ ;
6.  $M'$  simula a Máquina Universal  $U$  com a entrada  $\langle M, w \rangle$ ;
7. Se  $U$  aceita  $\langle M, w \rangle$ , então  $M'$  pára e aceita a sua entrada, qualquer que seja ela e  $L(M') \neq \emptyset$  (e se  $U$  não aceita  $\langle M, w \rangle$ , então  $M'$  não aceita nenhuma entrada e  $L(M') = \emptyset$ ).

# $L_{ne}$ não é recursiva

Em resumo:

- ▶ Existe um algoritmo que reduz  $L_u$  para  $L_{ne}$ ;
  - ▶  $M'$  aceita qualquer cadeia de entrada (e portanto  $\langle M' \rangle \in L_{ne}$ ) sse  $w \in L(M)$  (ou seja, se  $\langle M, w \rangle \in L_u$ );
  - ▶  $M'$  não aceita nenhuma cadeia de entrada (e portanto  $\langle M' \rangle \notin L_{ne}$ ) sse  $w \notin L(M)$  (ou seja, se  $\langle M, w \rangle \notin L_u$ );
- ▶ Como  $L_u$  é indecidível, então  $L_{ne}$  é indecidível.



$L_{ne}$  não é recursiva

Suponha que  $L_{ne}$  fosse decidível. Então seria possível decidir  $L_u$ , da seguinte forma:

- ▶ Fazer a redução de  $\langle M, w \rangle$  para  $M'$ ;
- ▶ Decidir se  $L(M') \neq \emptyset$ , ou seja, se  $\langle M' \rangle \in L_{ne}$ ;
- ▶ Em caso afirmativo,  $\langle M, w \rangle \in L_u$ , ou seja,  $w \in L(M)$ ;
- ▶ Em caso negativo,  $\langle M, w \rangle \notin L_u$ , ou seja,  $w \notin L(M)$ ;

Mas como é sabido que  $L_u$  não é recursiva, então a suposição de que  $L_{ne}$  é recursiva é falsa.

# $L_e$ é não-RE

Teorema:  $L_e$  não é recursivamente enumerável.

Prova:

1. Suponha que  $L_e$  seja recursivamente enumerável;
2. Portanto, de acordo com um teorema anterior, tanto  $L_e$  quanto  $\overline{L_e}$  devem ser recursivas;
3. Mas  $\overline{L_e} = L_{ne}$ ;
4. Além disso, foi demonstrado que  $L_{ne}$  não é recursiva;
5. Logo,  $L_e$  não é recursivamente enumerável.

# Linguagens e complementos

Resumo até este ponto

- ▶  $L_d = \{w_i \mid w_i \notin L(M_i)\}$  é não-RE;
- ▶  $\overline{L_d} = \{w_i \mid w_i \in L(M_i)\}$  é RE não-recursiva;
- ▶  $L_u = \{\langle M \rangle w \mid M \text{ é uma MT que aceita } w\}$  é RE não-recursiva;
- ▶  $\overline{L_u} = \{\langle M \rangle w \mid M \text{ é uma MT que não aceita } w\}$  é não-RE;
- ▶  $PARA_{MT} = \{\langle M', w \rangle \mid M' \text{ pára com a entrada } w\}$  é RE não-recursiva;
- ▶  $\overline{PARA_{MT}} = \{\langle M', w \rangle \mid M' \text{ entra em loop com a entrada } w\}$  é não-RE;
- ▶  $L_e = \{\langle M \rangle \mid L(M) = \emptyset\}$  é não-RE;
- ▶  $L_{ne} = \overline{L_e} = \{\langle M \rangle \mid L(M) \neq \emptyset\}$  é RE não-recursiva.

# Enunciado

Teorema: Qualquer propriedade não-trivial das linguagens recursivamente enumeráveis é indecidível.

- ▶ Propriedade?
- ▶ Não-trivial?

# Propriedade não-trivial

Propriedade:

- ▶ Condição que deva ser satisfeita por um grupo de linguagens;
- ▶ Um conjunto de linguagens que satisfazem uma certa condição.

Não-trivial:

- ▶ Condição que seja satisfeita por pelo menos uma linguagem e que não seja satisfeita por pelo menos uma linguagem;
- ▶ Caso contrário, ou seja, se a propriedade é satisfeita por todas as linguagens ou então não é satisfeita por nenhuma linguagem, então ela é dita “trivial”;
- ▶ Propriedade não-trivial exclui todas as propriedades triviais.

As linguagens RE serão representadas pelas MT que as aceitam, pois essas máquinas são descrições finitas de tais linguagens.

# Exemplos

Dada uma MT  $M$  qualquer:

- ▶  $L(M) = \emptyset$ ?  $L(M) \neq \emptyset$ ?
- ▶  $\epsilon \in L(M)$ ?
- ▶  $w \in L(M)$ ?
- ▶  $L(M)$  é finita?  $L(M)$  é infinita?
- ▶  $L(M)$  contém pelo menos duas cadeias?
- ▶  $L(M)$  é regular?
- ▶  $L(M)$  é livre de contexto?
- ▶  $L(M) = \Sigma^*$ ?
- ▶  $L(M) = L(M)^R$ ?
- ▶ etc.

## Exemplos

- ▶  $L(M) = \emptyset$ ?

Demonstrada indecidível anteriormente através do problema de decisão  $L_e$  ( $\langle M \rangle \in L_e$ ?)

- ▶  $L(M) \neq \emptyset$ ?

Demonstrada indecidível anteriormente através do problema de decisão  $L_{ne}$  ( $\langle M \rangle \in L_{ne}$ ?)

- ▶ Demais propriedades:

Considerar  $\mathcal{P}$  como o conjunto de todas as linguagens que satisfazem a propriedade;

Considerar a linguagem  $M_{\mathcal{P}} = \{\langle M \rangle \mid L(M) \in \mathcal{P}\}$ ;

$M_{\mathcal{P}}$  é o conjunto de todas as codificações de Máquinas de Turing que aceitam as linguagens pertencentes à  $\mathcal{P}$ ;

Determinar se  $L(M) \in \mathcal{P}$  é o mesmo que determinar se  $\langle M \rangle \in M_{\mathcal{P}}$ .

# Demonstração

Teorema: Qualquer propriedade não-trivial das linguagens recursivamente enumeráveis é indecidível.

Prova:

1. Seja  $\mathcal{P}$  uma propriedade não-trivial das linguagens RE;
2. Suponha que a linguagem vazia ( $\emptyset$ ) não pertence a  $\mathcal{P}$ ;
3. Como  $\mathcal{P}$  é não-trivial, então existe pelo menos uma linguagem  $L \in \mathcal{P}$ ;
4. Considere essa linguagem  $L$  e  $M_L$  tal que  $L = L(M_L)$ ;
5. Fazer uma redução de  $L_u$  para  $M_{\mathcal{P}}$  (conforme explicado a seguir):
  - ▶  $M$  aceita  $w \Rightarrow M'$  aceita  $L$ , portanto  $M' \in M_{\mathcal{P}}$ ;
  - ▶  $M$  não aceita  $w \Rightarrow M'$  aceita  $\emptyset$ , portanto  $M' \notin M_{\mathcal{P}}$ .
6. Como  $L_u$  é indecidível, conclui-se que  $M_{\mathcal{P}}$  também é indecidível.



## Redução de $L_u$ para $L_{\mathcal{P}}$

Obtenção de  $M'$  a partir de  $\langle M, w \rangle$  tal que  $\langle M, w \rangle \in L_u \Leftrightarrow \langle M' \rangle \in M_{\mathcal{P}}$ :  
(lembrar que  $L = L(M_L) \in \mathcal{P}$ )

1.  $M'$  simula a Máquina Universal  $U$  com a entrada  $\langle M, w \rangle$ ;
2. Se  $M$  não aceita  $w$  (ou seja, se  $\langle M, w \rangle \notin L_u$ ), então  $M'$  não faz nada. Portanto,  $M'$  não aceita a sua entrada, qualquer que seja ela; logo,  $L(M') = \emptyset$ ; como  $\emptyset \notin \mathcal{P}$ , então  $\langle M' \rangle \notin M_{\mathcal{P}}$ ;
3. Se  $M$  aceita  $w$ , então  $M'$  simula  $M_L$  com a sua entrada original, qualquer que seja ela; logo,  $L(M') = L$ ; como  $L \in \mathcal{P}$ , então  $\langle M' \rangle \in M_{\mathcal{P}}$ ;

Observar que todas as sentenças de  $L_u$  reduzem para uma mesma sentença  $\langle M' \rangle$  (que aceita  $M_L$ ) de  $M_{\mathcal{P}}$ , e também que todas as cadeias que não pertencem à  $L_u$  reduzem para a mesma cadeia  $\langle M' \rangle$  (que aceita  $\emptyset$ ) que não pertence à  $M_{\mathcal{P}}$ .

# Conclusão

A demonstração do teorema considerou que a linguagem vazia ( $\emptyset$ ) não pertence a  $\mathcal{P}$ ;

E se a linguagem vazia ( $\emptyset$ ) pertencer a  $\mathcal{P}$ ?

- ▶ Considerar  $\overline{\mathcal{P}}$ ;
- ▶ Dessa maneira,  $\emptyset \notin \overline{\mathcal{P}}$ ;
- ▶ Considerar  $M_{\overline{\mathcal{P}}}$ ;
- ▶ Aplicar os mesmos passos da demonstração do teorema para  $M_{\overline{\mathcal{P}}}$ ;
- ▶ Conclui-se que  $M_{\overline{\mathcal{P}}}$  não é recursiva;
- ▶ Observar que  $M_{\overline{\mathcal{P}}} = \overline{M_{\mathcal{P}}}$ ;
- ▶ Se  $\overline{M_{\mathcal{P}}}$  não é recursiva, então  $M_{\mathcal{P}}$  não é recursiva, pois o complemento de uma linguagem recursiva é também uma linguagem recursiva;
- ▶ Portanto  $M_{\mathcal{P}}$  não é recursiva da mesma forma.

# Exemplo

O problema de determinar se a linguagem aceita por uma Máquina de Turing é livre de contexto é indecidível.

- ▶ Pelo Teorema de Rice, é suficiente provar que “ser livre de contexto” é uma propriedade não-trivial das linguagens recursivamente enumeráveis;
- ▶ Ou seja, basta apresentar duas linguagens RE, uma que seja livre de contexto e outra que não seja;
- ▶ A linguagem  $\{a^i b^i c^i \mid i \geq 0\}$  é RE mas não é livre de contexto;
- ▶ A linguagem  $\{a^i b^i \mid i \geq 0\}$  é RE e livre de contexto.

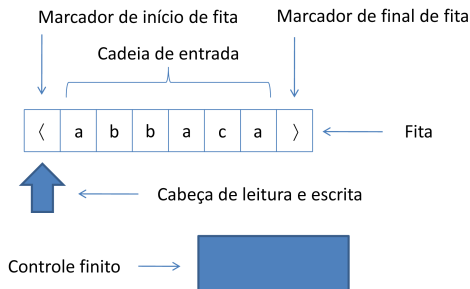
E se  $\mathcal{P}$  for trivial?

Então  $\mathcal{P}$  é decidida por uma MT que sempre aceita (se  $\mathcal{P}$  contém todas as linguagens) ou sempre rejeita (se  $\mathcal{P} = \emptyset$ ). O teorema, nesses casos, não pode ser aplicado:

- ▶  $\mathcal{P}$  contém todas as linguagens  $\Rightarrow$  o passo 2 da prova do teorema não é verificado;
- ▶  $\mathcal{P} = \emptyset \Rightarrow$  o passo 3 da prova do teorema não é verificado;

# Conceito

Um Autômato Linearmente Limitado (ALL), também conhecido como Máquina de Turing com Fita Limitada, é uma Máquina de Turing na qual o tamanho da fita de entrada é limitada ao comprimento da cadeia a ser analisada.



# Formalização

Um Autômato Linearmente Limitado (ALL) é uma 8-upla:

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

onde:

- ▶  $Q$  é o conjunto de estados;
- ▶  $\Sigma$  é o alfabeto de entrada;
- ▶  $\Gamma$  é o alfabeto de símbolos que podem ser lidos e/ou escritos na fita,  $\Sigma \subseteq \Gamma$ ;
- ▶  $\delta$  é a função de transição;
- ▶  $\langle$  e  $\rangle$  são os símbolos que delimitam a cadeia de entrada na fita,  $\langle \notin \Gamma, \rangle \notin \Gamma$ ;
- ▶  $F$  é o conjunto de estados finais.

O ALL não pode se movimentar para à direita do símbolo  $\rangle$  nem para a esquerda do símbolo  $\langle$  e nem pode substituí-los por outros símbolos.

# Observações

- ▶ Um ALL é um caso particular de MT em que a movimentação da cabeça de leitura/escrita é limitada ao trecho da fita que contém a cadeia de entrada a ser analisada;
- ▶ A quantidade de memória de trabalho disponível depende do alfabeto  $\Gamma$  e cresce linearmente com o comprimento da cadeia de entrada (por isso o nome “Linearmente Limitado”);
- ▶ Um ALL pode ser determinístico ou não-determinístico;
- ▶ Demonstra-se que a classe das linguagens reconhecidas pelos ALL não-determinísticos coincide com a classe das linguagens geradas pelas gramáticas sensíveis ao contexto (a menos da cadeia vazia);
- ▶ Não se sabe se ALLs não-determinísticos reconhecem a mesma classe de linguagens que os ALL determinísticos;
- ▶ Em qualquer caso, a quantidade de configurações distintas de um ALL para uma certa entrada é sempre finita.

# Aplicações

Apesar da limitação da fita de entrada (e conseqüentemente da sua memória), ALLs são bastante poderosos:

- ▶ Eles são modelos bastante próximos dos computadores reais modernos;
- ▶ Eles são capazes de decidir  $A_{AFD}$ ,  $A_{GLC}$ ,  $V_{AFD}$  e  $V_{GLC}$ ;
- ▶ Eles reconhecem uma classe de linguagens mais ampla do que os autômatos de pilha (que por sua vez necessitam de uma memória infinita na forma de pilha) - a classe das linguagens sensíveis ao contexto ou tipo 1;
- ▶ Ainda assim, existem problemas que não podem ser decididos por ALLs (qualquer problema que seja representado por uma linguagem que não seja do tipo 1);
- ▶ Além disso, também existem problemas **sobre** ALLs que são indecidíveis. Um exemplo é  $V_{ALL}$ , que será apresentado mais adiante.



# Quantidade máxima de configurações

Lema: Seja  $M$  um  $ALL$  (determinístico ou não-determinístico) com  $|Q|$  estados,  $|\Gamma|$  símbolos no seu alfabeto de fita e uma cadeia de entrada  $w$ ,  $|w| = n$ . Então existem exatamente  $|Q| * (n + 2) * |\Gamma|^n$  configurações distintas para  $M$ .

Prova:

1. A configuração é uma tripla composta por estado, posição da cabeça de leitura/escrita na fita e conteúdo da fita;
2.  $M$  possui  $|Q|$  estados distintos;
3. A cabeça de leitura/escrita pode se encontrar em  $n + 2$  posições distintas;
4. Existem  $|\Gamma|^n$  combinações diferentes de conteúdo para a fita de entrada;
5. Portanto, existem  $|Q| * (n + 2) * |\Gamma|^n$  configurações distintas para  $M$ .

# Quantidade máxima de configurações

## Exemplo

Seja  $M$  um  $ALL$  com a cadeia de entrada  $aabbcc$  e:

- ▶  $Q = \{q_0, q_1, q_2\}$ ;
- ▶  $\Gamma = \{a, b, c, X\}$ ;

Então existem exatamente  $3 * (6 + 2) * 4^6$  configurações distintas para  $M$ :

- ▶ 3 estados possíveis;
- ▶  $6 + 2$  posições distintas para a cabeça de leitura/escrita;
- ▶  $4^6$  cadeias distintas sobre a fita de entrada.

# Repetição de configurações

Lema: Seja  $M$  um  $ALL$  determinístico. Se  $M$  assumir a mesma configuração mais de uma vez durante a análise de uma cadeia, então ele está em loop.

Prova:

1. Suponha que a sequência de movimentações é  $\dots C_i C_{i+1} C_{i+2} \dots C_k C_{k+1} C_{k+2} \dots$  e que  $C_i = C_k$ ;
2. Então, o conteúdo da fita de entrada, a posição do cursor de leitura/escrita e o estado corrente são idênticos;
3. Como o  $ALL$  é determinístico, e como ele se movimenta da configuração  $C_i$  para a configuração  $C_{i+1}$ , então é certo que o mesmo acontecerá com  $C_k$ , e portanto  $C_{i+1} = C_{k+1}$ ;
4. Além disso, como  $C_i$  leva até  $C_k$ , então é fato que  $C_k$  conduzirá o  $ALL$  até uma outra configuração, que também será idêntica à  $C_i$ ;
5. Este ciclo se repete por um número indeterminado de vezes e portanto o  $ALL$  está em loop.

# Repetição de configurações

Corolário: Seja  $M$  um  $ALL$  determinístico. Se  $M$  assume, durante o reconhecimento de uma determinada cadeia de entrada, uma quantidade de configurações que é idêntica à quantidade total de configurações distintas que ele possui para esta cadeia, então ele está em loop.

Prova:

1. Suponha que a quantidade de configurações distintas que ele possui para uma certa cadeia de entrada seja  $n$ ;
2. Suponha que, durante o reconhecimento desta cadeia, ele já passou por  $n$  configurações;
3. Portanto, a próxima configuração que ele irá assumir será a de número  $n + 1$ ;
4. Como o  $ALL$  possui apenas  $n$  configurações distintas, é certo que, dentre estas  $n + 1$  configurações, duas deverão ser repetidas;
5. Logo, pelo lema anterior, o  $ALL$  está em loop com a cadeia de entrada.

# Problema $A_{ALL}$

Aceitação em Autômatos Linearmente Limitados Determinísticos:

$$A_{ALL} = \{ \langle M, w \rangle \mid M \text{ é um } ALL \text{ determinístico que aceita a cadeia } w \}$$

Teorema:  $A_{ALL}$  é uma linguagem decidível.

Prova:

1. Suponha que o ALL seja  $M = (Q, \Sigma, \Gamma, \delta, q_0, \langle, \rangle, F)$ ;
2. Suponha  $|w| = n$ ;
3. Construir  $M'$  que simula  $M$  com a entrada  $w$ :
  - ▶ Simular até que  $M$  pare ou até que tenham sido executadas  $|Q| * (n + 2) * |\Gamma|^n - 1$  movimentações;
  - ▶ Se  $M$  pára e aceita, então  $M'$  pára e aceita;
  - ▶ Se  $M$  pára e rejeita, então  $M'$  pára e rejeita;
  - ▶ Se  $M$  não parou, então  $M'$  rejeita (veja corolário anterior).

# Problema $A_{ALL}$

- ▶ A quantidade máxima de configurações distintas que o  $ALL$  determinístico pode assumir é conhecida e essa informação é usada para detectar loops.

	MT	ALL
Fita de entrada	Ilimitada	Limitada
Quantidade de configurações para uma mesma entrada	Infinita	Finita
Problema da aceitação	$L_u$	$A_{ALL}$
Problema da aceitação	Indecidível	Decidível

# História de computação

Uma “história de computação” de uma MT  $M$  sobre uma cadeia de entrada  $w$  é a seqüência de configurações  $C_1C_2\dots C_n$  que  $M$  assume durante a análise de  $w$ .

- ▶ Se  $w \in L(M) = ACEITA(M)$  então  $C_1C_2\dots C_n$  é uma “história de computação de aceitação” onde  $C_1$  é a configuração inicial,  $C_n$  é configuração final de aceitação e  $C_i$  segue de forma legítima  $C_{i-1}$ , para  $1 < i \leq n$ ;
- ▶ Se  $w \in REJEITA(M)$  então  $C_1C_2\dots C_n$  é uma “história de computação de rejeição” onde  $C_1$  é a configuração inicial,  $C_n$  é configuração final de rejeição e  $C_i$  segue de forma legítima  $C_{i-1}$ , para  $1 < i \leq n$ ;
- ▶ Se  $w \in LOOP(M)$  então  $C_1C_2\dots C_n\dots$  é uma seqüência infinita de configurações.

# História de computação

Sejam  $M$  e  $w$ :

- ▶ Se  $M$  é determinística, então existe uma única história de computação (de aceitação, de rejeição ou de loop) para  $w$ ;
- ▶ Se  $M$  é não-determinística, então podem existir várias histórias de computação para  $w$  (finitas ou infinitas).



# Problemas indecidíveis

Provaremos a seguir que os dois problemas abaixo são indecidíveis:

- ▶  $V_{ALL} = \{\langle M \rangle \mid M \text{ é um } ALL \text{ e } L(M) = \emptyset\}$ ;
- ▶  $TODAS_{GLC} = \{\langle G \rangle \mid G \text{ é uma } GLC \text{ e } L(G) = \Sigma^*\}$ .

A prova, em ambos os casos, será por redução a partir de  $L_u$  e com o emprego de histórias de computação.

# Problema $V_{ALL}$

Vacuidade da linguagem aceita por um Autômato Linearmente Limitado:

$$V_{ALL} = \{\langle M \rangle \mid M \text{ é um } ALL \text{ e } L(M) = \emptyset\}$$

Teorema:  $V_{ALL}$  é uma linguagem indecidível.

Prova:

- ▶ Suponha que  $V_{ALL}$  é decidível;
- ▶ Logo,  $\overline{V_{ALL}}$  também é decidível;
- ▶ Fazer uma redução de  $L_u$  para  $\overline{V_{ALL}}$  usando histórias de computação;
- ▶ Se  $\overline{V_{ALL}}$  fosse decidível, então  $L_u$  também seria;
- ▶ Como  $L_u$  não é decidível, segue que a hipótese é falsa,  $\overline{V_{ALL}}$  não é decidível e  $V_{ALL}$  não é decidível.

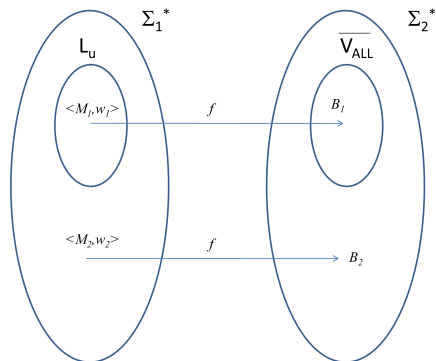
# Problema $V_{ALL}$

Redução de  $L_u$  para  $\overline{V_{ALL}}$ :

- ▶ Construir um ALL  $B$  a partir de  $\langle M, w \rangle$  tal que:  
 $\langle M, w \rangle \in L_u \Leftrightarrow \langle B \rangle \in \overline{V_{ALL}}$
- ▶ O ALL  $B$  é construído de forma que  $L(B)$  compreende todas as histórias de computação de aceitação de  $M$  para  $w$ ;
- ▶ Se  $M$  rejeita  $w$ , ou seja, se  $\langle M, w \rangle \notin L_u$ , então  $L(B) = \emptyset$  e  $\langle B \rangle \notin \overline{V_{ALL}}$ ;
- ▶ Se  $M$  aceita  $w$ , ou seja, se  $\langle M, w \rangle \in L_u$ , então  $L(B) \neq \emptyset$  e  $\langle B \rangle \in \overline{V_{ALL}}$ .

# Problema $V_{ALL}$

- ▶  $M_1$  aceita  $w_1 \Rightarrow \langle M_1, w_1 \rangle \in L_u \Rightarrow L(B_1) \neq \emptyset \Rightarrow \langle B_1 \rangle \in \overline{V_{ALL}}$ ;
- ▶  $M_2$  não aceita  $w_2 \Rightarrow \langle M_2, w_2 \rangle \notin L_u \Rightarrow L(B_2) = \emptyset \Rightarrow \langle B_2 \rangle \notin \overline{V_{ALL}}$ ;



# Problema $V_{ALL}$

Construção de  $B$  a partir de  $\langle M, w \rangle$ :

1. Suponha que a entrada para  $B$  é  $C_1 \# C_2 \# \dots \# C_n$ ;
2. As três condições seguintes devem ser válidas;
3.  $B$  verifica se  $C_1$  é uma configuração inicial válida para  $M$  com a cadeia  $w$ :  
 $C_1 = q_0 w$  pode ser verificado conhecendo-se  $M$  e  $w$ ;
4.  $B$  verifica se  $C_i$  segue de forma legítima  $C_{i-1}$ , para  $1 < i \leq n$ :  
 $C_i$  deve corresponder à combinação da configuração  $C_{i-1}$  com a aplicação de uma transição de  $M$ ;
5.  $B$  verifica se  $C_n$  é uma configuração de aceitação para  $M$ :  
 $C_n = \alpha q_f \beta$  pode ser verificado conhecendo-se  $M$ .

# Problema $V_{ALL}$

Examinando por outro ângulo:

- ▶ Deseja-se determinar se  $\langle M, w \rangle \in L_u$ ;
- ▶ Suponha que  $V_{ALL}$  é decidível por uma MT  $R$ ;
- ▶ A partir de  $\langle M, w \rangle$  obter o ALL  $B$  conforme descrito;
- ▶ Executar  $R$  com a entrada  $\langle B \rangle$ ;
- ▶ Se  $R$  aceita, isso significa que  $L(B)$  é vazia e portanto que  $w \notin L(M)$ , ou seja,  $\langle M, w \rangle \notin L_u$ ;
- ▶ Se  $R$  rejeita, isso significa que  $L(B)$  é não-vazia e portanto que  $w \in L(M)$ , ou seja,  $\langle M, w \rangle \in L_u$ ;
- ▶ Logo, seria possível decidir  $L_u$ ;
- ▶ Mas isso é uma contradição e portanto  $V_{ALL}$  não é decidível;

Observar que  $L_e$  está para as MTs assim como  $V_{ALL}$  está para os ALLs, e que **ambos são indecidíveis** (ao contrário de  $L_u$  e  $V_{ALL}$ ).

# Problema $TODAS_{GLC}$

Totalidade da linguagem gerada por uma gramática livre de contexto:

$$TODAS_{GLC} = \{\langle G \rangle \mid G \text{ é uma } GLC \text{ e } L(G) = \Sigma^*\}$$

Teorema:  $TODAS_{GLC}$  é uma linguagem indecidível.

Prova:

- ▶ Suponha que  $TODAS_{GLC}$  é decidível;
- ▶ Logo,  $\overline{TODAS_{GLC}}$  também é decidível;
- ▶ Fazer uma redução de  $L_u$  para  $\overline{TODAS_{GLC}}$  usando histórias de computação;
- ▶ Se  $\overline{TODAS_{GLC}}$  fosse decidível, então  $L_u$  também seria;
- ▶ Como  $L_u$  não é decidível, segue que a hipótese é falsa,  $\overline{TODAS_{GLC}}$  não é decidível e  $TODAS_{GLC}$  não é decidível.

# Problema $TODAS_{GLC}$

Redução de  $L_u$  para  $\overline{TODAS_{GLC}}$

- ▶ Construir uma GLC  $G$  a partir de  $\langle M, w \rangle$  tal que:
  - $\langle M, w \rangle \in L_u \Leftrightarrow \langle G \rangle \in \overline{TODAS_{GLC}}$ ;
- ▶  $G$  gera cadeias sobre um alfabeto  $\Sigma$ ;
- ▶  $G$  gera todas as cadeias sobre  $\Sigma$  que não representam histórias de computação de aceitação para  $M$  com  $w$ ;
  - ▶ Isto inclui todas as cadeias que **não** representam histórias de computação para  $M$  com  $w$ ;
  - ▶ Inclui, também, todas as cadeias que representam histórias de computação que **não são de aceitação** para  $M$  com  $w$ ;
  - ▶  $G$  gera todas as cadeias sobre  $\Sigma$  se e apenas se  $w \notin M$ ;
  - ▶  $G$  não gera todas as cadeias sobre  $\Sigma$  se e apenas se  $w \in M$ ; nesse caso,  $G$  deve falhar em gerar justamente em gerar a cadeia que representa a história de computação de aceitação para  $M$  com  $w$ ;



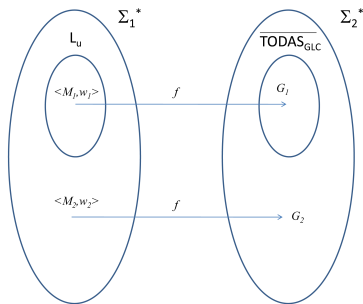
# Problema $TODAS_{GLC}$

Redução de  $L_u$  para  $\overline{TODAS_{GLC}}$  (resumo):

- ▶ Se  $M$  não aceita  $w$ , ou seja, se  $\langle M, w \rangle \notin L_u$ , então  $G$  não falha em gerar nenhuma cadeia sobre  $\Sigma$ ,  $L(G) = \Sigma^*$  e  $\langle G \rangle \notin \overline{TODAS_{GLC}}$ ;
- ▶ Se  $M$  aceita  $w$ , ou seja, se  $\langle M, w \rangle \in L_u$ , então  $G$  falha em gerar a cadeia que representa a história de computação de aceitação de  $w$  em  $M$ ,  $L(G) \neq \Sigma^*$  e  $\langle G \rangle \in \overline{TODAS_{GLC}}$ .

# Problema $TODAS_{GLC}$

- ▶  $M_1$  aceita  $w_1 \Rightarrow \langle M_1, w_1 \rangle \in L_u \Rightarrow L(G_1) \neq \Sigma^* \Rightarrow \langle G_1 \rangle \in \overline{TODAS_{GLC}}$ ;
- ▶  $M_2$  não aceita  $w_2 \Rightarrow \langle M_2, w_2 \rangle \notin L_u \Rightarrow L(G_2) = \Sigma^* \Rightarrow \langle G_2 \rangle \notin \overline{TODAS_{GLC}}$ ;



# Problema $TODAS_{GLC}$

Construção de  $G$  a partir de  $\langle M, w \rangle$ :

1.  $G$  gera todas as cadeias sobre  $\Sigma$  que não representam histórias de computação (seqüências de configurações) para  $M$  com  $w$  (por exemplo, onde uma configuração possui 0 ou 2 ou mais estados, entre outras possibilidades);
2.  $G$  gera todas as cadeias que representam histórias de computação (seqüências de configurações) que não são de aceitação para  $M$  com  $w$ ;
3. Histórias de computação de  $M$  com  $w$  tem o formato:  
 $C_1\#C_2\#\dots\#C_n$ ;
4. Para gerar todas as cadeias que representam histórias de computação que não são de aceitação para  $M$  com  $w$ , pelo menos uma das seguintes condições devem ser verificadas;

Problema  $TODAS_{GLC}$ 

- $G$  gera todas as histórias de computação tais que  $C_1$  não é uma configuração inicial válida para  $M$  com a cadeia  $w$ :  
Pode ser feito conhecendo-se  $M$  e  $w$ ;
- $G$  gera todas as histórias de computação tais que  $C_i$  não segue de forma legítima  $C_{i-1}$ , para  $2 \leq i \leq n$ :  
Pode ser feito conhecendo-se  $M$  e  $w$ ;
- $G$  gera todas as histórias de computação tais que  $C_n$  não é uma configuração de aceitação para  $M$ :  
Pode ser feito conhecendo-se  $M$ .

# Problema $TODAS_{GLC}$

Construção de  $G$  a partir de  $\langle M, w \rangle$ :

- ▶ Para a linguagem especificada anteriormente, projetar um autômato de pilha não-determinístico (APN) é mais fácil do que projetar a gramática diretamente;
- ▶ Para obter  $G$ , iremos inicialmente obter um APN  $D$  que aceita  $L(G)$ ;
- ▶ Finalmente, o APN  $D$  pode ser convertido para uma GLC  $G$ .

# Problema $TODAS_{GLC}$

Construção do APN  $D$  a partir de  $\langle M, w \rangle$ :

1. A cadeia de entrada para  $D$  é uma cadeia sobre o alfabeto  $\Sigma$ ;
2. Se a cadeia de entrada não representa uma história de computação, então  $D$  pára e aceita;
3. Senão,  $D$  seleciona, de forma não-determinística, qual das três condições ele irá testar;
4. No primeiro ramo,  $D$  aceita se  $C_1$  não é uma configuração inicial válida para  $M$  com a cadeia  $w$ ; caso contrário, rejeita;
5. No segundo ramo,  $D$  seleciona não-deterministicamente um par de configurações  $C_i$  e  $C_{i-1}$  (com  $2 \leq i \leq n$ ) para analisar:
  - ▶  $D$  aceita se  $C_i$  não segue de forma legítima  $C_{i-1}$ ; caso contrário, rejeita;
6. No terceiro ramo,  $D$  aceita se  $C_n$  não é uma configuração de aceitação para  $M$ ; caso contrário, rejeita.

# Problema $TODAS_{GLC}$

Observações:

- ▶ No segundo ramo,  $D$  empilha a configuração  $C_{i-1}$  e depois compara com a configuração  $C_i$ ;
- ▶ Para que isso seja possível, será necessário que as configurações de ordem par sejam escritas na cadeia de entrada de forma revertida;
- ▶  $C_1 \# C_2^R \# C_3 \# C_4^R \# \dots$

# Problema $TODAS_{GLC}$

## Conclusões:

- ▶  $D$  rejeita a cadeia que representa a história de computação de aceitação para  $M$  com  $w$  (se esta existir); corresponde ao caso em que todos os ramos da execução não-determinística rejeitam a entrada;
- ▶  $D$  aceita todas as cadeias que não representam histórias de computação para  $M$  com  $w$ ;
- ▶  $D$  aceita todas as história de computação que não são de aceitação para  $M$  com  $w$ ;
- ▶ Se  $w \notin L(M)$ , então  $L(D) = L(G) = \Sigma^*$ , ou seja,  $G \notin \overline{TODAS_{GLC}}$ ;
- ▶ Se  $w \in L(M)$ , então  $L(D) = L(G) \neq \Sigma^*$ , ou seja,  $G \in \overline{TODAS_{GLC}}$ ;
- ▶ A existência de  $D$  prova a existência de  $G$ , e, conseqüentemente, a existência de uma redução de  $L_u$  para  $\overline{TODAS_{GLC}}$ . Logo,  $\overline{TODAS_{GLC}}$  e  $TODAS_{GLC}$  são indecidíveis.



# Origem e natureza

*“Post’s Correspondence Problem”* (Problema da Correspondência de Post)

- ▶ Problema que não está relacionado com Máquinas de Turing ou as linguagens por elas aceitas;
- ▶ Problema combinatorial que envolve a manipulação (emparelhamento) de cadeias de caracteres;
- ▶ Demonstra-se ser indecidível;
- ▶ A indecidibilidade do PCP foi provada por Post em 1946;
- ▶ É usado para demonstrar a indecidibilidade de vários outros problemas.

# Definição

Uma “instância” PCP consiste de duas listas  $A$  e  $B$  de cadeias formadas sobre um mesmo alfabeto  $\Sigma$ . As duas listas devem ter o mesmo comprimento.

- ▶  $A = w_1, w_2, \dots, w_k$ ;
- ▶  $B = x_1, x_2, \dots, x_k$ ;
- ▶ Para um certo valor de  $i$ , diz-se que o par  $(w_i, x_i)$  é um par que está em correspondência;
- ▶ Pares em correspondência podem ser considerados como peças de um dominó:

$$\left[ \begin{array}{c} w_1 \\ x_1 \end{array} \right], \left[ \begin{array}{c} w_2 \\ x_2 \end{array} \right], \dots, \left[ \begin{array}{c} w_k \\ x_k \end{array} \right]$$

# Solução

Diz-se que uma instância PCP tem uma solução se existir uma seqüência de um ou mais números inteiros (repetições permitidas)  $i_1, i_2, \dots, i_m$ , os quais, quando interpretados como índices de cadeias nas listas  $A$  e  $B$ , produzem como resultado a mesma cadeia.

- ▶  $A = w_1, w_2, \dots, w_k$ ;
- ▶  $B = x_1, x_2, \dots, x_k$ ;
- ▶ Diz-se que  $i_1, i_2, \dots, i_m, m \geq 1$ , é uma solução para esta instância PCP se  $w_{i_1}w_{i_2}\dots w_{i_m} = x_{i_1}x_{i_2}\dots x_{i_m}$

# Solução

PCP como um tipo de jogo de dominó:

- ▶ Composto por uma quantidade finita de peças:

$$\left[ \begin{array}{c} w_1 \\ x_1 \end{array} \right], \left[ \begin{array}{c} w_2 \\ x_2 \end{array} \right], \dots, \left[ \begin{array}{c} w_k \\ x_k \end{array} \right]$$

- ▶ Peças são combinadas para formar cadeias idênticas na parte de cima e na parte de baixo;
- ▶ Peças podem ser duplicadas para formar cadeias:

$$\left[ \begin{array}{c} w_{i_1} \\ x_{i_1} \end{array} \right] \left[ \begin{array}{c} w_{i_2} \\ x_{i_2} \end{array} \right] \dots \left[ \begin{array}{c} w_{i_m} \\ x_{i_m} \end{array} \right]$$

## Exemplo

Seja  $\Sigma = \{0, 1\}$  e suponha que as listas  $A$  e  $B$  sejam as seguintes:

	Lista $A$	Lista $B$
$i$	$w_i$	$x_i$
1	1	111
2	10111	10
3	10	0

- Uma solução para essa instância é a seqüência:

$i_1 = 2, i_2 = 1, i_3 = 1, i_4 = 3$ , ou simplesmente 2, 1, 1, 3, pois

$$w_2 w_1 w_1 w_3 = \underbrace{10111}_{w_2} \underbrace{1}_{w_1} \underbrace{1}_{w_1} \underbrace{10}_{w_3} =$$

$$x_2 x_1 x_1 x_3 = \underbrace{10}_{x_2} \underbrace{111}_{x_1} \underbrace{111}_{x_1} \underbrace{0}_{x_3} = 101111110;$$

- Entre outras, 2, 1, 1, 3, 2, 1, 1, 3 também é solução.

## Exemplo

## Continuação

- ▶ Representação da solução 2, 1, 1, 3 na forma de dominós:

$$\begin{array}{cccc}
 w_2 & w_1 & w_1 & w_3 \\
 \left[ \begin{array}{c} 10111 \\ \hline 10 \end{array} \right] & \left[ \begin{array}{c} 1 \\ \hline 111 \end{array} \right] & \left[ \begin{array}{c} 1 \\ \hline 111 \end{array} \right] & \left[ \begin{array}{c} 10 \\ \hline 0 \end{array} \right] \\
 x_2 & x_1 & x_1 & x_3
 \end{array}$$

## Exemplo

Seja  $\Sigma = \{a, b, c\}$  e suponha que as listas  $A$  e  $B$  sejam as seguintes:

	Lista $A$	Lista $B$
$i$	$w_i$	$x_i$
1	abc	ab
2	ca	a
3	acc	ba

- ▶ Essa instância não possui solução, pois  $|w_i| > |x_i|, \forall i$ .

## Exemplo

Seja  $\Sigma = \{0, 1\}$  e suponha que as listas  $A$  e  $B$  sejam as seguintes:

	Lista $A$	Lista $B$
$i$	$w_i$	$x_i$
1	10	101
2	011	11
3	101	011

- ▶ Essa instância também não possui solução:
- ▶ Se  $i_1 = 2$ , então  $A = 011\dots$ ,  $B = 11\dots$  e não é possível gerar uma solução;
- ▶ Se  $i_1 = 3$ , então  $A = 101\dots$ ,  $B = 011\dots$  e não é possível gerar uma solução;



# Exemplo

## Continuação

- ▶ Com  $i_1 = 1$ , então  $A = 10\dots$ ,  $B = 101\dots$  talvez seja possível obter uma solução;
- ▶ Se  $i_2 = 1$ , então  $A = 1010\dots$ ,  $B = 101101\dots$  e não é possível gerar uma solução;
- ▶ Se  $i_2 = 2$ , então  $A = 10011\dots$ ,  $B = 10111\dots$  e não é possível gerar uma solução;
- ▶ Com  $i_2 = 3$ , então  $A = 10101\dots$ ,  $B = 101011\dots$  talvez seja possível obter uma solução;
- ▶ No entanto, o mesmo raciocínio leva à escolha de  $i_3 = 3$ , e assim por diante, e não é possível nunca gerar uma solução.

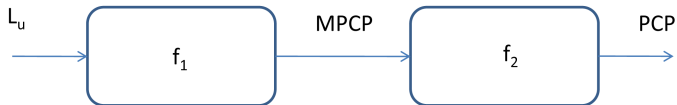
# Problema

Dada uma instância PCP sobre um certo alfabeto  $\Sigma$ , determinar se ela possui uma solução.

- ▶  $PCP = \{\langle P \rangle \mid P \text{ é uma instância PCP com uma solução}\}$ ;
- ▶  $PCP$  é indecidível.

# Estratégia da demonstração

- 1 Reduzir  $L_u$  para uma versão modificada do PCP (MPCP);
- 2 Reduzir MPCP para PCP;
- 3 Como  $L_u$  é indecidível, MPCP e PCP são também indecidíveis.



# Definição

“Modified Post Correspondence Problem” (Problema da Correspondência de Posto Modificado):

- ▶ Uma instância MPCP é definida da mesma forma que uma instância PCP;
- ▶ A solução, no entanto, deve obrigatoriamente iniciar com o par 1;
- ▶  $A = w_1, w_2, \dots, w_k$ ;
- ▶  $B = x_1, x_2, \dots, x_k$ ;
- ▶ Diz-se que  $i_1, i_2, \dots, i_m, m \geq 0$ , é uma solução para esta instância MPCP se  $w_1 w_{i_1} w_{i_2} \dots w_{i_m} = x_1 x_{i_1} x_{i_2} \dots x_{i_m}$

## Exemplo

Seja  $\Sigma = \{0, 1\}$  e suponha que as listas  $A$  e  $B$  sejam as seguintes:

	Lista $A$	Lista $B$
$i$	$w_i$	$x_i$
1	1	111
2	10111	10
3	10	0

- ▶ Considerada como instância PCP, há solução;
- ▶ Considerada como instância MPCP, não há solução.

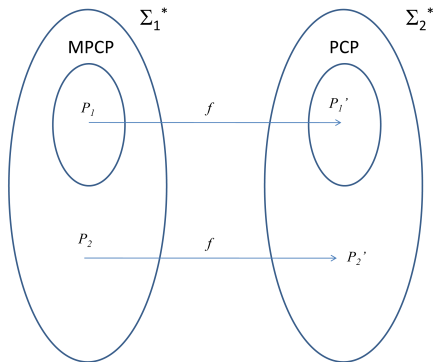
# Exemplo

## Continuação

- ▶  $A = w_1\dots = 1\dots$ ,  $B = x_1\dots = 111\dots$
- ▶ Se  $i_1 = 2$ , então  $A = 110111\dots$ ,  $B = 11110\dots$  e não há solução possível;
- ▶ Se  $i_1 = 3$ , então  $A = 110\dots$ ,  $B = 1110\dots$  e não há solução possível;
- ▶ Se  $i_1 = 1$ , então  $A = 11\dots$ ,  $B = 111111\dots$  e não há solução possível, pois as cadeias nunca terão o mesmo tamanho.

# Redução de MPCP para PCP

$P_1$  é uma instância MPCP com solução  $\Leftrightarrow P'_1$  é uma instância PCP com solução.



# Redução de MPCP para PCP

A obtenção de  $P'_1(P'_2)$  (PCP) a partir de  $P_1(P_2)$  (MPCP) pode ser feita da seguinte forma:

- ▶ MPCP= $(A, B)$  sobre  $\Sigma$ ;
- ▶ Suponha  $A = w_1, w_2, \dots, w_k$ ;
- ▶ Suponha  $B = x_1, x_2, \dots, x_k$ ;
- ▶ Suponha que  $*$   $\notin \Sigma$ ,  $\$$   $\notin \Sigma$ ;
- ▶ PCP= $(C, D)$  sobre  $\Sigma \cup \{*, \$\}$ ;
- ▶  $C = y_0, y_1, y_2, \dots, y_k, y_{k+1}$ ;
- ▶  $D = z_0, z_1, z_2, \dots, z_k, z_{k+1}$ ;



# Redução de MPCP para PCP

- ▶  $\forall i, 1 \leq i \leq k$ ,  $y_i$  é obtido a partir de  $w_i$  pela inserção do símbolo \* após cada símbolo de  $w_i$
- ▶  $\forall i, 1 \leq i \leq k$ ,  $z_i$  é obtido a partir de  $x_i$  pela inserção do símbolo \* antes cada símbolo de  $x_i$
- ▶  $y_0 = *y_1$
- ▶  $z_0 = z_1$
- ▶  $y_{k+1} = \$$
- ▶  $z_{k+1} = *\$$

## Exemplo

Suponha a instância MPCP:

	Lista $A$	Lista $B$
$i$	$w_i$	$x_i$
1	1	111
2	10111	10
3	10	0

A aplicação da construção anterior resulta na instância PCP:

	Lista $C$	Lista $D$
$i$	$y_i$	$z_i$
0	*1*	*1*1*1
1	1*	*1*1*1
2	1*0*1*1*1*	*1*0
3	1*0*	*0
4	\$	*\$

# Redução de MPCP para PCP

Para provar que a construção proposta é uma redução, é necessário (suponha que  $P_1$  reduz para  $P'_1$ ):

- 1 Provar que se  $P_1$  é uma instância MPCP com solução, então  $P'_1$  é uma instância PCP com solução;
- 2 Provar que se  $P'_1$  é uma instância PCP com solução, então  $P_1$  é uma instância MPCP com solução.

# Estratégias de prova

- ▶ Método alternativo para provar que se trata de uma redução:

$$(A \leftrightarrow B) \leftrightarrow ((A \rightarrow B) \wedge (B \rightarrow A))$$

- ▶ Ele contrasta com o método usado até então:

$$(A \leftrightarrow B) \leftrightarrow ((A \rightarrow B) \wedge (\sim A \rightarrow \sim B))$$

- ▶ No entanto, é possível provar a equivalência dos dois métodos:

$$((A \rightarrow B) \wedge (\sim A \rightarrow \sim B)) \leftrightarrow ((A \rightarrow B) \wedge (B \rightarrow A))$$

onde:

- ▶ Na redução  $f$  de  $P_1$  para  $P'_1$ ;
- ▶  $A \equiv w \in P_1$ ;
- ▶  $B \equiv f(w) \in P'_1$

# Redução de MPCP para PCP

Se  $P_1$  é uma instância MPCP com solução, então  $P'_1$  é uma instância PCP com solução:

- ▶ Suponha que a solução de  $P_1$  seja  $i_1, i_2, \dots, i_m$ ;
- ▶ Portanto,  $w_1 w_{i_1} w_{i_2} \dots w_{i_m} = x_1 x_{i_1} x_{i_2} \dots x_{i_m}$ ;
- ▶ Considerar  $y_1 y_{i_1} y_{i_2} \dots y_{i_m}$  e  $z_1 z_{i_1} z_{i_2} \dots z_{i_m}$ ;
- ▶ As duas cadeias são idênticas, exceto pelo primeiro símbolo da primeira cadeia e pelo último símbolo da segunda cadeia;
- ▶ Ou seja,  $*y_1 y_{i_1} y_{i_2} \dots y_{i_m} = z_1 z_{i_1} z_{i_2} \dots z_{i_m} *$ ;
- ▶ Mas esse resultado pode ser obtido substituindo-se o primeiro par (de 1 por 0) e acrescentando-se um novo par no final ( $k + 1$ );
- ▶ Ou seja,  $y_0 y_{i_1} y_{i_2} \dots y_{i_m} y_{k+1} = z_0 z_{i_1} z_{i_2} \dots z_{i_m} z_{k+1}$ ;
- ▶ Logo,  $0, i_1, i_2, \dots, i_m, k + 1$  é uma solução de  $P'_1$ .

# Redução de MPCP para PCP

Se  $P'_1$  é uma instância PCP com solução, então  $P_1$  é uma instância MPCP com solução:

- ▶ A solução de  $P'_1$  deve começar com o par 0 e terminar com o par  $k + 1$ , pois apenas o par 0 inicia com o mesmo símbolo (\*) e apenas o par  $k + 1$  termina com o mesmo símbolo (\$);
- ▶ Portanto, a solução de  $P'_1$  é  $0, i_1, i_2, \dots, i_m, k + 1$ ;
- ▶ Logo,  $y_0 y_{i_1} y_{i_2} \dots y_{i_m} y_{k+1} = z_0 z_{i_1} z_{i_2} \dots z_{i_m} z_{k+1}$ ;
- ▶ Se forem removidos todos os símbolos \* e \$ de ambas as cadeias, resulta:
  - ▶  $w_1 w_{i_1} w_{i_2} \dots w_{i_m} = x_1 x_{i_1} x_{i_2} \dots x_{i_m}$ ;
  - ▶ Ou seja,  $1, i_1, i_2, \dots, i_m$  é solução para  $P_1$ .

# Redução de MPCP para PCP

Conclusões até o momento:

- ▶ Se PCP for decidível, então MPCP também será decidível;
- ▶ Se MPCP for indecidível, então PCP também será indecidível.

## Exemplo

Suponha a instância MPCP:

	Lista $A$	Lista $B$
$i$	$w_i$	$x_i$
1	ab	abb
2	bab	ba
3	ba	a
4	a	ba



## Exemplo

## Continuação

A aplicação da construção anterior resulta na instância PCP:

	Lista $C$	Lista $D$
$i$	$y_i$	$z_i$
0	$*a*b*$	$*a*b*b$
1	$a*b*$	$*a*b*b$
2	$b*a*b*$	$*b*a$
3	$b*a*$	$*a$
4	$a*$	$*b*a$
5	$\$$	$*\$$

## Exemplo

## Continuação

- ▶ Uma solução para a instância MPCP é 3, 2, 4;

$$y_1 y_3 y_2 y_4 = \underbrace{a * b * b}_{y_1} * \underbrace{a * b}_{y_3} * \underbrace{a * b}_{y_2} * \underbrace{a}_{y_4}$$

$$z_1 z_3 z_2 z_4 = \underbrace{* a * b * b}_{z_1} * \underbrace{* a}_{z_3} * \underbrace{* b * a}_{z_2} * \underbrace{* b * a}_{z_4}$$

- ▶ Substituir o par 1 pelo par 0 no início e acrescentar o par 5 no final;

$$y_0 y_3 y_2 y_4 y_5 = \underbrace{* a * b * b}_{y_0} * \underbrace{a * b}_{y_3} * \underbrace{a * b}_{y_2} * \underbrace{a}_{y_4} * \underbrace{\$}_{y_5}$$

$$z_0 z_3 z_2 z_4 z_5 = \underbrace{* a * b * b}_{z_0} * \underbrace{* a}_{z_3} * \underbrace{* b * a}_{z_2} * \underbrace{* b * a}_{z_4} * \underbrace{* \$}_{z_5}$$

- ▶ Portanto, 0, 3, 2, 4, 5 é uma solução para o PCP correspondente.

# Exemplo

## Continuação

- ▶ Uma solução para a instância PCP é 0, 3, 5;

$$\text{▶ } y_0 y_3 y_5 = \underbrace{*a * b*}_{y_0} \underbrace{b * a*}_{y_3} \underbrace{\$}_{y_5} = z_0 z_3 z_5 = \underbrace{*a * b * b}_{z_0} \underbrace{*a}_{z_3} \underbrace{*\$}_{z_5}$$

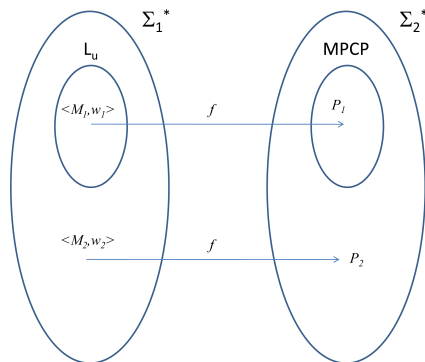
- ▶ Remover todos os símbolos \* e \$ de ambas as cadeias;

$$\text{▶ O resultado é } w_1 w_3 = \underbrace{ab}_{w_1} \underbrace{ba}_{w_3} = x_1 x_3 = \underbrace{abb}_{x_1} \underbrace{a}_{x_3}$$

- ▶ Logo, 1, 3 é uma solução para o MPCP correspondente.

# Redução de $L_u$ para MPCP

$\langle M_1, w_1 \rangle \in L_u \Leftrightarrow P_1$  é uma instância MPCP com solução.



# Redução de $L_u$ para MPCP

A obtenção de  $P$  a partir de  $\langle M, w \rangle$  pode ser feita da seguinte forma:

- ▶ As listas  $A$  e  $B$  representam a história de computação de  $M$  com  $w$ ;
- ▶ Soluções parciais para  $P$  representam histórias de computação incompletas para  $w$  em  $M$ ;
- ▶ Se  $w \in L(M)$ , ou seja, se  $\langle M, w \rangle \in L_u$ , então é possível gerar uma solução para  $P$ ;
- ▶ Se  $w \notin L(M)$ , ou seja, se  $\langle M, w \rangle \notin L_u$ , então não há solução possível para  $P$ ;
- ▶ A construção da lista  $A$  está sempre uma configuração “atrasada” em relação à construção da lista  $B$ ;
- ▶ As listas coincidem se e apenas se  $M$  entra num estado final.

# Redução de $L_u$ para MPCP

## Premissa

Seja  $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$  e suponha que:

- 1  $M$  não grava brancos na fita;
- 2  $M$  não se desloca para à esquerda da posição inicial da fita.

Nesse caso, é possível afirmar que:

- ▶ As configurações de  $M$  tem o formato geral  $\alpha q \beta$ , com  $q \in Q$ ,  $\alpha \in \Gamma^*$  e  $\beta \in \Gamma^*$ , ou seja,  $\alpha$  e  $\beta$  são compostos apenas por símbolos diferentes de  $B$ ;
- ▶ As cadeias  $\alpha$  e  $\beta$  representam as posições da fita inicialmente ocupadas pela cadeia de entrada  $w$ , além de eventuais posições visitadas à direita da mesma.

# Redução de $L_u$ para MPCP

Passo 1:

- ▶ O primeiro par da instância MPCP é:

	Lista $A$	Lista $B$
1	#	# $q_0w$ #

- ▶ Ele será usado para iniciar a solução, caso exista;
- ▶ Notar que a lista  $B$  está uma configuração adiantada em relação à lista  $A$ .

# Redução de $L_u$ para MPCP

Passo 2:

- ▶ Novos pares são criados a partir de  $\delta$ , com o objetivo de reproduzir a história de computação de  $w$  em  $M$ ;
- ▶  $\forall q_i \in Q - F, q_j \in Q, x, y, z \in \Gamma$ , acrescentar os pares:

Lista A	Lista B	
$q_i x$	$y q_j$	se $\delta(q_i, x) = (q_j, y, R)$
$z q_i x$	$q_j z y$	se $\delta(q_i, x) = (q_j, y, L)$
$q_i \#$	$y q_j \#$	se $\delta(q_i, B) = (q_j, y, R)$
$z q_i \#$	$q_j z y \#$	se $\delta(q_i, B) = (q_j, y, L)$

- ▶ Para cada transição possível de ser aplicada numa certa configuração de  $M$ , há um par correspondente em  $P$ ;
- ▶ Um par para cada transição com deslocamento à direita;  $|\Gamma|$  pares para cada transição com deslocamento à esquerda;
- ▶ A lista  $B$  está uma configuração adiantada em relação à lista  $A$ .



# Redução de $L_u$ para MPCP

Passo 3:

- ▶  $\forall x \in \Gamma$ , acrescentar os pares:

Lista $A$	Lista $B$
$x$	$x$
$\#$	$\#$

- ▶ Permitem a cópia de símbolos que não envolvam o estado corrente;
- ▶ Serão usados para permitir o avanço da solução até chegar numa nova configuração.

# Redução de $L_u$ para MPCP

Passo 4:

- ▶ Se um estado final foi alcançado, deve-se permitir que as cadeias se tornem idênticas;
- ▶  $\forall q_f \in F, x \in \Gamma, y \in \Gamma$ , acrescentar os pares:

Lista A	Lista B
$xq_fy$	$q_f$
$xq_f$	$q_f$
$q_fy$	$q_f$

- ▶ São geradas novas cadeias que não representam configurações;
- ▶ O uso recorrente desses pares permite o “consumo” dos símbolos que se encontram à esquerda e à direita do estado  $q_f$  na última configuração.

# Redução de $L_u$ para MPCP

Passo 5:

- ▶ Todos os símbolos, a menos de  $q_f$ , foram removidos da última configuração;

- ▶  $w_1 \dots w_k = \# \mu \#$

- ▶  $x_1 \dots x_k = \# \mu \# q_f \#$

- ▶ Para torná-las iguais, basta acrescentar o par:

Lista A	Lista B
$q_f \# \#$	$\#$

- ▶  $w_1 \dots w_k = \# \mu \# q_f \# \#$

- ▶  $x_1 \dots x_k = \# \mu \# q_f \# \#$

- ▶  $P$  tem uma solução.

## Exemplo

Construção de  $P$  a partir de

$M = (\{q_1, q_2, q_3\}, \{0, 1\}, \{0, 1, B\}, \delta, q_1, B, \{q_3\})$  e  $w = 01$ , com  $\delta$ :

$q_i$	$\delta(q_i, 0)$	$\delta(q_i, 1)$	$\delta(q_i, B)$
$q_1$	$(q_2, 1, R)$	$(q_2, 0, L)$	$(q_2, 1, L)$
$q_2$	$(q_3, 0, L)$	$(q_1, 0, R)$	$(q_2, 0, R)$
$q_3$	—	—	—

A história de computação de  $w$  em  $M$  é:

$$q_1 0 1 \vdash 1 q_2 1 \vdash 1 0 q_1 \vdash 1 q_2 0 1 \vdash q_3 1 0 1$$

## Exemplo

## Continuação

Passo	Lista A	Lista B	Origem
(1)	#	# $q_1$ 01#	
(2)	$q_1$ 0 $0q_1$ 1 $1q_1$ 1 $0q_1$ # $1q_1$ # $0q_2$ 0 $1q_2$ 0 $q_2$ 1 $q_2$ #	$1q_2$ $q_2$ 00 $q_2$ 10 $q_2$ 01# $q_2$ 11# $q_3$ 00 $q_3$ 10 $0q_1$ $0q_2$ #	$\delta(q_1, 0) = (q_2, 1, R)$ $\delta(q_1, 1) = (q_2, 0, L)$ $\delta(q_1, 1) = (q_2, 0, L)$ $\delta(q_1, B) = (q_2, 1, L)$ $\delta(q_1, B) = (q_2, 1, L)$ $\delta(q_2, 0) = (q_3, 0, L)$ $\delta(q_2, 0) = (q_3, 0, L)$ $\delta(q_2, 1) = (q_1, 0, R)$ $\delta(q_2, B) = (q_2, 0, R)$
(3)	0 1 #	0 1 #	

## Exemplo

## Continuação

Passo	Lista $A$	Lista $B$	Origem
(4)	$0q_30$	$q_3$	
	$0q_31$	$q_3$	
	$1q_30$	$q_3$	
	$1q_31$	$q_3$	
	$0q_3$	$q_3$	
	$1q_3$	$q_3$	
	$q_30$	$q_3$	
	$q_31$	$q_3$	
(5)	$q_3\#\#$	$\#$	

# Exemplo

## Continuação

- ▶ A solução para essa instância MPCP começa com o primeiro par (passo 1):

$$A : \#$$
$$B : \#q_101\#$$

# Exemplo

## Continuação

- ▶ Para continuar, é necessário que o próximo par da lista  $A$  seja prefixo da cadeia  $q_101\#$ ;
- ▶ O par  $(q_10, 1q_2)$  é selecionado (passo 2):

$A$  :  $\#q_10$

$B$  :  $\#q_101\#1q_2$



# Exemplo

## Continuação

- ▶ Para continuar, é necessário copiar o restante da configuração até alcançar o estado  $q_2$ ;
- ▶ Os pares  $(1, 1)$ ,  $(\#, \#)$  e  $(1, 1)$  são selecionados (passo 3):

$$A : \#q_101$$

$$B : \#q_101\#1q_21$$

$$A : \#q_101\#$$

$$B : \#q_101\#1q_21\#$$

$$A : \#q_101\#1$$

$$B : \#q_101\#1q_21\#1$$

# Exemplo

## Continuação

- ▶ Para continuar, é necessário que o próximo par da lista  $A$  seja prefixo da cadeia  $q_21\#1$ ;
- ▶ O par  $(q_21, 0q_1)$  é selecionado (passo 2):

$$A : \#q_101\#1q_21$$
$$B : \#q_101\#1q_21\#10q_1$$

# Exemplo

## Continuação

- ▶ Em seguida, pode-se copiar 2 (#1) ou 3 (#10) símbolos antes de aplicar uma nova transição;
- ▶ No entanto, a inserção de 3 símbolos impede o desenvolvimento das cadeias, pois não existem pares na lista  $A$  que sejam prefixo de  $q_1\#10$ :

$$A : \#q_101\#1q_21\#10$$

$$B : \#q_101\#1q_21\#10q_1\#10$$

# Exemplo

## Continuação

- ▶ Isso acontece porque a próxima movimentação de  $M$  é para a esquerda e, portanto, o símbolo à esquerda de  $q_1$  é necessário para fazer a escolha do par correto nesse caso;
- ▶ Deve-se copiar apenas 2 símbolos ( $\#$  e 1), resultando em:

$$A : \#q_101\#1q_21\#1$$

$$B : \#q_101\#1q_21\#10q_1\#1$$

# Exemplo

## Continuação

- ▶ Para continuar, é necessário que o próximo par da lista  $A$  seja prefixo da cadeia  $0q_1\#1$ ;
- ▶ O par  $(0q_1\#, q_201\#)$  é selecionado (passo 2):

$$A : \#q_101\#1q_21\#10q_1\#$$

$$B : \#q_101\#1q_21\#10q_1\#1q_201\#$$

# Exemplo

## Continuação

- ▶ Para continuar, pode-se seleccionar o par  $(1, 1)$  (passo 3) ou então seleccionar o par  $(1q_20, q_310)$  (passo 2);
- ▶ Como a primeira escolha impede o desenvolvimento futuro das cadeias, deve-se optar pela segunda alternativa e o resultado é:

$$A : \#q_101\#1q_21\#10q_1\#1q_20$$

$$B : \#q_101\#1q_21\#10q_1\#1q_201\#q_310$$

# Exemplo

## Continuação

- ▶ Esse ponto corresponde à entrada de  $M$  num estado de aceitação ( $q_3$ );
- ▶ Portanto, são iniciados os procedimentos para tornar as cadeias idênticas;
- ▶ Antes, porém, são selecionados os pares  $(1, 1)$  e  $(\#, \#)$ :

$$A : \#q_101\#1q_21\#10q_1\#1q_201$$

$$B : \#q_101\#1q_21\#10q_1\#1q_201\#q_3101$$

$$A : \#q_101\#1q_21\#10q_1\#1q_201\#$$

$$B : \#q_101\#1q_21\#10q_1\#1q_201\#q_3101\#$$

# Exemplo

## Continuação

- ▶ Para continuar, é selecionado o par  $(q_31, q_3)$  (passo 4):

$$A : \#q_101\#1q_21\#10q_1\#1q_201\#q_31$$

$$B : \#q_101\#1q_21\#10q_1\#1q_201\#q_3101\#q_3$$

- ▶ Copiando os símbolos 0, 1 e # (passo 3):

$$A : \#q_101\#1q_21\#10q_1\#1q_201\#q_3101\#$$

$$B : \#q_101\#1q_21\#10q_1\#1q_201\#q_3101\#q_301\#$$



## Exemplo

## Continuação

- ▶ Para continuar, é selecionado o par  $(q_30, q_3)$  (passo 4):

$$A : \#q_101\#1q_21\#10q_1\#1q_201\#q_3101\#q_30$$

$$B : \#q_101\#1q_21\#10q_1\#1q_201\#q_3101\#q_301\#q_3$$

- ▶ Copiando os símbolos 1 e # (passo 3):

$$A : \#q_101\#1q_21\#10q_1\#1q_201\#q_3101\#q_301\#$$

$$B : \#q_101\#1q_21\#10q_1\#1q_201\#q_3101\#q_301\#q_31\#$$

# Exemplo

## Continuação

- ▶ Para continuar, é selecionado novamente o par  $(q_31, q_3)$  (passo 4):

$A$  :  $\#q_101\#1q_21\#10q_1\#1q_201\#q_3101\#q_301\#q_31$

$B$  :  $\#q_101\#1q_21\#10q_1\#1q_201\#q_3101\#q_301\#q_31\#q_3$

- ▶ Copiando o símbolo  $\#$  (passo 3):

$A$  :  $\#q_101\#1q_21\#10q_1\#1q_201\#q_3101\#q_301\#q_31\#$

$B$  :  $\#q_101\#1q_21\#10q_1\#1q_201\#q_3101\#q_301\#q_31\#q_3\#$

# Exemplo

## Continuação

- ▶ Para terminar, é selecionado o par  $(q_3\#\#, \#)$  (passo 5):

$A : \#q_101\#1q_21\#10q_1\#1q_201\#q_3101\#q_301\#q_31\#q_3\#\#$

$B : \#q_101\#1q_21\#10q_1\#1q_201\#q_3101\#q_301\#q_31\#q_3\#\#$

- ▶ As cadeias são idênticas e  $P$  tem solução.

# Redução de $L_u$ para MPCP

Para provar que a construção proposta é uma redução, é necessário (considerar  $P$  obtido a partir de  $M, w$  conforme visto anteriormente):

- 1 Provar que se  $\langle M, w \rangle \in L_u$ , então  $P$  é uma instância MPCP com solução;
- 2 Provar que se  $P$  é uma instância MPCP com solução, então  $\langle M, w \rangle \in L_u$ .

# Redução de $L_u$ para MPCP

Se  $\langle M, w \rangle \in L_u$ , então  $P$  é uma instância MPCP com solução:

- ▶ Iniciar a simulação com o par 1;
- ▶ Usar os pares do passo 2 para representar movimentações de  $M$  e pares do passo 3 para copiar símbolos da fita e  $\#$  conforme necessário;
- ▶ Se  $M$  entrar num estado de aceitação, usar os pares do passo 4 e depois o par do passo 5 para permitir que as cadeias fiquem idênticas;
- ▶ Logo, se  $\langle M, w \rangle \in L_u$ , então  $P$  tem solução.

# Redução de $L_u$ para MPCP

Se  $P$  é uma instância MPCP com solução, então  $\langle M, w \rangle \in L_u$ :

- ▶ Por se tratar de MPCP, a solução parcial começa com:

$$A : \#$$
$$B : \#q_0w\#$$

# Redução de $L_u$ para MPCP

- ▶ Enquanto  $M$  não entra em um estado de aceitação, apenas os pares dos passos 2 e 3 pode ser usados, e as cadeias possuem o formato geral (observar que  $|xy| > |x|$ ):

$$A : x$$

$$B : xy$$

- ▶ Se existir uma solução, então isso significa que, em algum momento, os pares do passo 4 terão sido usados;
- ▶ Logo, se  $P$  tem solução, então  $M$  entra em um estado de aceitação, ou seja,  $M$  aceita  $w$  e  $\langle M, w \rangle \in L_u$ .

# Problema $AMB_{GLC}$

Determinar se uma gramática livre de contexto  $G$  qualquer é ambígua:

$$AMB_{GLC} = \{ \langle G \rangle \mid G \text{ é uma GLC ambígua} \}$$

Teorema:  $AMB_{GLC}$  é indecidível.

Prova:

- ▶ Por redução a partir de PCP.



# Problema $AMB_{GLC}$

Construção de uma GLC  $G$  a partir de uma instância PCP  $P$ , tal que  $P$  tem solução  $\Leftrightarrow G$  é ambígua:

- ▶ Seja  $P = (A, B)$  sobre  $\Sigma$ ;
- ▶  $A = w_1, w_2, \dots, w_k$ ;
- ▶  $B = x_1, x_2, \dots, x_k$ ;
- ▶ Seja  $G_A$  uma GLC que gera uma linguagem  $L_A$  sobre  $\Sigma' = \Sigma \cup \{a_1, a_2, \dots, a_k\}$ :

$$A \rightarrow w_1 A a_1 \mid w_2 A a_2 \mid \dots \mid w_k A a_k$$

$$A \rightarrow w_1 a_1 \mid w_2 a_2 \mid \dots \mid w_k a_k$$

- ▶  $a_i$  representa o índice  $i$  usado para selecionar o par correspondente.

# Problema $AMB_{GLC}$

A linguagem  $L_A$ :

- ▶ Suas sentenças tem a forma geral:

$$w_{i_1} w_{i_2} \dots w_{i_m} a_{i_m} \dots a_{i_2} a_{i_1}$$

com  $m \geq 1$  e  $1 \leq i_1, i_2, \dots, i_m \leq k$ .

- ▶  $L_A$  é não-ambígua (todas as suas sentenças possuem uma única seqüência de derivações mais à esquerda).

# Problema $AMB_{GLC}$

Seja  $G_B$  uma GLC que gera uma linguagem  $L_B$  sobre  $\Sigma' = \Sigma \cup \{a_1, a_2, \dots, a_k\}$ :

- ▶  $A = w_1, w_2, \dots, w_k$ ;
- ▶  $B = x_1, x_2, \dots, x_k$ ;
- ▶  $G_B$ :

$$B \rightarrow x_1 B a_1 \mid x_2 B a_2 \mid \dots \mid x_k B a_k$$

$$B \rightarrow x_1 a_1 \mid x_2 a_2 \mid \dots \mid x_k a_k$$

- ▶  $a_i$  representa o índice  $i$  usado para selecionar o par correspondente;
- ▶  $L_B$  é não-ambígua.

# Problema $AMB_{GLC}$

Construção de uma GLC  $G_{AB}$  a partir de  $G_A$  e  $G_B$ , que por sua vez foram construídas a partir de  $P$ :

- ▶ Seja  $G_A = (\{A\} \cup \Sigma', \Sigma', P_A, A)$ ;
- ▶ Seja  $G_B = (\{B\} \cup \Sigma', \Sigma', P_B, B)$ ;
- ▶ Construir  $G_{AB} =$

$$(\{S, A, B\} \cup \Sigma', \Sigma', P_A \cup P_B \cup \{S \rightarrow A, S \rightarrow B\}, S)$$

- ▶  $L(G_{AB}) = L_A \cup L_B$ .

# Problema $AMB_{GLC}$

Para provar que a construção proposta é uma redução, basta provar que:

- 1 Se a instância PCP  $P$  tem solução, então  $G$  é ambígua;
- 2 Se  $G$  é ambígua, então a instância PCP  $P$  tem solução.

# Problema $AMB_{GLC}$

Se  $G$  é ambígua, então  $P$  tem solução:

1. Considere  $G = G_{AB}$ ;
2. Se  $G$  é ambígua, então existe pelo menos uma cadeia  $\alpha$  com duas ou mais derivações mais à esquerda em  $L(G)$ ;
3. Como, por construção,  $G_A$  e  $G_B$  são não-ambíguas, então as duas derivações para  $\alpha$  devem ser:

$$S \Rightarrow A \Rightarrow \dots \Rightarrow \alpha$$

$$S \Rightarrow B \Rightarrow \dots \Rightarrow \alpha$$

4. No entanto,  $\alpha = w_{i_1} w_{i_2} \dots w_{i_m} a_{i_m} \dots a_{i_2} a_{i_1} = x_{i_1} x_{i_2} \dots x_{i_m} a_{i_m} \dots a_{i_2} a_{i_1}$
5. Portanto,  $w_{i_1} w_{i_2} \dots w_{i_m} = x_{i_1} x_{i_2} \dots x_{i_m}$
6. Logo,  $P$  tem uma solução  $i_1, i_2, \dots, i_m$ .

# Problema $AMB_{GLC}$

Se  $P$  tem solução, então  $G$  é ambígua:

1. Considere  $G = G_{AB}$ ;
2. Suponha que  $i_1, i_2, \dots, i_m$  seja uma solução para  $P$ ;
3. Considere as seguintes derivações em  $G$ :

$$\begin{aligned}
 S &\Rightarrow A \Rightarrow w_{i_1} A a_{i_1} \Rightarrow w_{i_1} w_{i_2} A a_{i_2} a_{i_1} \Rightarrow \dots \\
 &\Rightarrow w_{i_1} w_{i_2} \dots w_{i_{m-1}} A a_{i_{m-1}} \dots a_{i_2} a_{i_1} \\
 &\Rightarrow w_{i_1} w_{i_2} \dots w_{i_{m-1}} w_{i_m} a_{i_m} a_{i_{m-1}} \dots a_{i_2} a_{i_1} \\
 S &\Rightarrow B \Rightarrow x_{i_1} B a_{i_1} \Rightarrow x_{i_1} x_{i_2} B a_{i_2} a_{i_1} \Rightarrow \dots \\
 &\Rightarrow x_{i_1} x_{i_2} \dots x_{i_{m-1}} B a_{i_{m-1}} \dots a_{i_2} a_{i_1} \\
 &\Rightarrow x_{i_1} x_{i_2} \dots x_{i_{m-1}} x_{i_m} a_{i_m} a_{i_{m-1}} \dots a_{i_2} a_{i_1}
 \end{aligned}$$

# Problema $AMB_{GLC}$

4. Como  $i_1, i_2, \dots, i_m$  é uma solução, então  $w_{i_1}w_{i_2}\dots w_{i_m} = x_{i_1}x_{i_2}\dots x_{i_m}$
5. Como as cadeias são idênticas, e como elas foram geradas de formas distintas, usando apenas derivações mais à esquerda, então  $G$  é ambígua.

Como  $PCP$  reduz para  $AMB_{GLC}$  e  $PCP$  é indecidível, então  $AMB_{GLC}$  é também indecidível.



## Exemplo

- ▶ Seja  $P$  a seguinte instância PCP sobre  $\{a, b\}$ :

	Lista $A$	Lista $B$
	$w_i$	$x_i$
1	$aaa$	$aa$
2	$baa$	$abaaa$

- ▶ Considerar  $\Sigma' = \{a, b, a_1, a_2\}$  e  $G_A$ :

$$A \rightarrow aaaAa_1 \mid baaAa_2 \mid aaaa_1 \mid baaa_2$$

- ▶ Considerar  $\Sigma' = \{a, b, a_1, a_2\}$  e  $G_B$ :

$$B \rightarrow aaBa_1 \mid abaaaBa_2 \mid aaa_1 \mid abaaaa_2$$

## Exemplo

- ▶ Considerar  $\Sigma' = \{a, b, a_1, a_2\}$  e  $G_{AB}$ :

$$S \rightarrow A | B$$

$$A \rightarrow aaaAa_1 | baaAa_2 | aaaa_1 | baaa_2$$

$$B \rightarrow aaBa_1 | abaaaBa_2 | aaa_1 | abaaaa_2$$

## Exemplo

## Continuação

$P$  tem solução  $\Rightarrow G_{AB}$  é ambígua:

- ▶ A seqüência 121 é uma solução para  $P$ ;
- ▶ Considerar  $w_1w_2w_1a_1a_2a_1 \in L_A$  e  $x_1x_2x_1a_1a_2a_1 \in L_B$ ;
- ▶ Como 121 é solução, então  $w_1w_2w_1 = x_1x_2x_1$  e, portanto,  
 $w_1w_2w_1a_1a_2a_1 = x_1x_2x_1a_1a_2a_1 = aaabaaaaa_1a_2a_1$ ;
- ▶ No entanto, existem duas derivações mais à esquerda distintas para essa cadeia em  $G_{AB}$ :

$$S \Rightarrow A \Rightarrow aaaAa_1 \Rightarrow aaabaaAa_2a_1 \Rightarrow aaabaaaaa_1a_2a_1$$

$$S \Rightarrow B \Rightarrow aaBa_1 \Rightarrow aaabaaaBa_2a_1 \Rightarrow aaabaaaaa_1a_2a_1$$

- ▶ Portanto,  $G_{AB}$  é ambígua.

## Exemplo

## Continuação

$G_{AB}$  é ambígua  $\Rightarrow P$  tem solução:

- ▶ Seja cadeia  $aaabaaaaa_1a_2a_1 \in L_{AB}$ ;
- ▶ Essa cadeia tem duas derivações mais à esquerda distintas:

$$S \Rightarrow A \Rightarrow aaaAa_1 \Rightarrow aaabaaAa_2a_1 \Rightarrow aaabaaaaa_1a_2a_1$$

$$S \Rightarrow B \Rightarrow aaBa_1 \Rightarrow aaabaaaBa_2a_1 \Rightarrow aaabaaaaa_1a_2a_1$$

- ▶ Da primeira derivação, pode-se concluir que  $aaabaaaaa = w_1w_2w_1$ ;
- ▶ Da segunda derivação, pode-se concluir que  $aaabaaaaa = x_1x_2x_1$ ;
- ▶ Portanto,  $P$  tem uma solução (121).

Significado de  $L_A \cap L_B$ 

A quantidade de sentenças em  $L_A \cap L_B$  é igual à quantidade de soluções distintas da instância  $P$  do PCP que deu origem às gramáticas  $G_A$  e  $G_B$ :

- ▶ Cada elemento de  $L_A \cap L_B$  corresponde à uma solução distinta;
- ▶  $L_A \cap L_B = \emptyset \Rightarrow P$  não tem solução;
- ▶  $L_A \cap L_B \neq \emptyset \Rightarrow P$  tem (pelo menos uma) solução.

No exemplo anterior:

- ▶  $aaabaaaaa_1a_2a_1 \in (L_A \cap L_B) \Rightarrow P$  tem solução (121).

# Complemento de uma linguagem de lista

- ▶  $L_A$  e  $L_B$  são linguagens livres de contexto;
- ▶ Deseja-se provar que  $\overline{L_A}$  e  $\overline{L_B}$  são também livres de contexto;
- ▶ Linguagens livres de contexto **não são** fechadas em relação à operação de complementação;
- ▶ Esses resultados permitirão a demonstração de que outros problemas acerca das linguagens livres de contexto são também indecidíveis.

$\overline{L_A}$  é LLC

Teorema: Seja  $L_A$  uma linguagem para a lista  $A$  de uma instância PCP  $P$  sobre  $\Sigma \cup \{a_1, a_2, \dots, a_k\}$ . Então  $\overline{L_A}$  é também livre de contexto.

Prova:

Será apresentado um autômato de pilha determinístico  $M$ , com critério de aceitação estado final, que reconhece  $\overline{L_A}$ .

$\overline{L}_A$  é LLC

1. Enquanto  $M$  encontrar apenas símbolos de  $\Sigma$  na entrada, ele os insere na pilha. Se a cadeia de entrada esgotar,  $M$  aceita pois todas as cadeias de  $\Sigma^* \in \overline{L}_A$ ;
2. Verificar se o próximo símbolo da cadeia de entrada é  $a_i$ ; se não é, aceitar;
3. Desempilhar  $|w_i|$  símbolos do topo da pilha; se não houverem  $|w_i|$  símbolos na pilha, aceitar; se houverem, verificar se eles correspondem à  $w_i^R$ :
  - (a) Em caso negativo, então a cadeia de entrada certamente não pertence à  $L_A$ . Nesse caso,  $M$  deve esgotar a leitura dos símbolos da cadeia de entrada e ir para um estado de aceitação;
  - (b) Em caso afirmativo, e se a pilha ainda não está vazia, ir para 2;
  - (c) Em caso afirmativo, e se a pilha está vazia, a cadeia analisada até o momento pertence à  $L_A$ . A aceitação ou rejeição de  $M$  estará condicionada à presença de novos símbolos no final da cadeia.
4. Se houverem outros símbolos de  $\Sigma$  na cadeia entrada, aceitar. Caso contrário, rejeitar.



## Exemplo

Seja  $\Sigma = \{0, 1\}$  e suponha que as listas  $A$  e  $B$  sejam as seguintes:

	Lista $A$	Lista $B$
$i$	$w_i$	$x_i$
1	1	111
2	10111	10
3	10	0

$$A \rightarrow 1Aa_1 \mid 10111Aa_2 \mid 10Aa_3 \mid 1a_1 \mid 10111a_2 \mid 10a_3$$

## Exemplo

► A cadeia  $\underbrace{10111}_{w_2} \underbrace{10}_{w_3} a_3 a_2 \notin \overline{L_A}$ , pois  $M$ :

- ① Empilha  $1011110$ ;
- ② Quando encontra  $a_3$ , desempilha  $|w_3| = 2$  símbolos,  $\sigma_1 \sigma_2$ ;
- ③ Verifica que  $\sigma_1 \sigma_2 = 01 = w_3^R$ ;
- ④ Quando encontra  $a_2$ , desempilha  $|w_2| = 5$  símbolos,  $\sigma_1 \sigma_2 \sigma_3 \sigma_4 \sigma_5$ ;
- ⑤ Verifica que  $\sigma_1 \sigma_2 \sigma_3 \sigma_4 \sigma_5 = 11101 = w_2^R$ ;
- ⑥ Como não há outros símbolos na cadeia de entrada, a cadeia pertence à  $L_A$  e portanto  $M$  a rejeita.

## Exemplo

▶ A cadeia  $\underbrace{10111}_{w_2} \underbrace{10}_{w_3} a_2 a_3 \in \overline{L_A}$ , pois:

- 1 Empilha 1011110;
- 2 Quando encontra  $a_2$ , desempilha  $|w_2| = 5$  símbolos,  $\sigma_1 \sigma_2 \sigma_3 \sigma_4 \sigma_5$ ;
- 3 Verifica que  $\sigma_1 \sigma_2 \sigma_3 \sigma_4 \sigma_5 = 01111 \neq w_2^R$  e aceita a entrada.

▶ A cadeia  $a_3 a_2 \underbrace{10111}_{w_2} \underbrace{10}_{w_3} \in \overline{L_A}$ , pois:

- 1 Não existem símbolos na pilha para verificar depois de encontrado  $a_3$ .

## Exemplo

▶ A cadeia  $\underbrace{10111}_{w_2} \underbrace{10}_{w_3} a_3 a_2 a_1 \in \overline{L_A}$ :

- ① Empilha 1011110;
- ② Quando encontra  $a_3$ , desempilha  $|w_3| = 2$  símbolos,  $\sigma_1 \sigma_2$ ;
- ③ Verifica que  $\sigma_1 \sigma_2 = 01 = w_3^R$ ;
- ④ Quando encontra  $a_2$ , desempilha  $|w_2| = 5$  símbolos,  $\sigma_1 \sigma_2 \sigma_3 \sigma_4 \sigma_5$ ;
- ⑤ Verifica que  $\sigma_1 \sigma_2 \sigma_3 \sigma_4 \sigma_5 = 11101 = w_2^R$ ;
- ⑥ Quando encontra  $a_1$ ,  $M$  aceita a entrada.

▶ A cadeia  $\underbrace{10111}_{w_2} \underbrace{10}_{w_3} \in \overline{L_A}$ :

- ① Empilha 1011110;
- ② Como não encontra nenhum  $a_i$ , a entrada é aceita.

## Exemplo

- ▶ A cadeia  $\underbrace{11111}_? a_2 \in \overline{L_A}$ :
- ① Empilha 11111;
  - ② Quando encontra  $a_2$ , desempilha  $|w_2| = 5$  símbolos,  $\sigma_1\sigma_2\sigma_3\sigma_4\sigma_5$ ;
  - ③ Verifica que  $\sigma_1\sigma_2\sigma_3\sigma_4\sigma_5 = 11111 \neq w_2^R$  e aceita a entrada.
- ▶ A cadeia  $\underbrace{1111}_? \underbrace{1}_{w_1} a_1 a_2 \in \overline{L_A}$ :
- ① Empilha 11111;
  - ② Quando encontra  $a_1$ , desempilha  $|w_1| = 1$  símbolos,  $\sigma_1$ ;
  - ③ Verifica que  $\sigma_1 = 1 = w_1^R$ ;
  - ④ Quando encontra  $a_2$ , tenta desempilhar  $|w_2| = 5$  símbolos, mas existem apenas 4 deles na pilha;
  - ⑤  $M$  aceita a entrada.

# Problemas

Sejam  $G_1, G_2$  gramáticas livres de contexto quaisquer e  $R$  uma expressão regular qualquer. Os seguintes problemas são indecidíveis:

- 1  $L(G_1) \cap L(G_2) = \emptyset?$
- 2  $L(G_1) = L(G_2)?$
- 3  $L(G_1) = L(R)?$
- 4  $L(G_1) = T^*$  para algum alfabeto  $T$ ?
- 5  $L(G_1) \subseteq L(G_2)?$
- 6  $L(R) \subseteq L(G_1)?$

# Problemas

Serão feitas reduções de PCP para cada um desses problemas:

- ▶ Seja  $\Sigma$  o alfabeto da instância PCP  $P$  considerada;
- ▶ Seja  $I = \{a_1, a_2, \dots, a_k\}$ ;
- ▶  $L_A, L_B, \overline{L_A}$  e  $\overline{L_B}$  são linguagens livres de contexto construídas sobre  $P$ ;

# Observações

Considere  $P$  uma instância PCP sobre  $\Sigma$  com listas  $A$  e  $B$ . Então:

- ▶ Se  $P$  tem solução, então  $L_A \cap L_B \neq \emptyset$ ;
- ▶  $L_A \cap L_B$ , sobre  $\Sigma \cup I$ , contém todas as cadeias que **representam** solução para  $P$ ;
- ▶  $L_A \cap L_B$  contém tantas cadeias quantas sejam as soluções distintas para  $P$ ;
- ▶ Se  $P$  não tem solução, então  $L_A \cap L_B = \emptyset$ ;
- ▶  $\overline{L_A} \cup \overline{L_B} = \overline{L_A \cap L_B}$ ;
- ▶  $\overline{L_A \cap L_B}$ , sobre  $\Sigma \cup I$ , contém todas as cadeias que **não representam** solução para  $P$ ;
- ▶  $(\Sigma \cup I)^*$  contém todas as cadeias sobre  $(\Sigma \cup I)$ .



$$L(G_1) \cap L(G_2) = \emptyset?$$

$$INT_{GLC} = \{\langle G_1, G_2 \rangle \mid G_1 \text{ e } G_2 \text{ são GLCs e } L(G_1) \cap L(G_2) = \emptyset\}$$

- ▶ Seja  $G_1 = G_A$ ;
- ▶ Seja  $G_2 = G_B$ ;
- ▶ Se existe  $w \in L(G_1) \cap L(G_2)$ , então existe solução para  $P$ ;
- ▶ O conjunto  $L(G_1) \cap L(G_2)$  contém todas as cadeias que representam soluções de  $P$ ;
- ▶ Logo, se  $P$  tem solução  $\Rightarrow L(G_1) \cap L(G_2) \neq \emptyset$ ;
- ▶ Além disso, se  $P$  não tem solução  $\Rightarrow L(G_1) \cap L(G_2) = \emptyset$ ;
- ▶ Temos uma redução de  $P$  para  $\overline{INT_{GLC}}$ ;
- ▶ Portanto,  $\overline{INT_{GLC}}$  e também  $INT_{GLC}$  são indecidíveis.

$$L(G_1) = L(G_2)?$$

$$EQ_{GLC} = \{\langle G_1, G_2 \rangle \mid G_1 \text{ e } G_2 \text{ são GLCs e } L(G_1) = L(G_2)\}$$

- ▶ Seja  $G_1$  tal que  $L(G_1) = \overline{L_A} \cup \overline{L_B}$  (LLCs são fechadas em relação à união);
- ▶ Seja  $G_2$  tal que  $L(G_2) = (\Sigma \cup I)^*$  (a linguagem é regular);
- ▶ Notar que  $L(G_1) = \overline{L_A} \cup \overline{L_B} = \overline{L_A \cap L_B}$ ; portanto,  $L(G_1)$  contém todas as cadeias que **não representam** solução para  $P$ ;
- ▶  $L(G_2)$  contém todas as cadeias sobre o alfabeto  $\Sigma \cup I$ ;
- ▶ Logo,  $P$  tem solução  $\Leftrightarrow L(G_1) \neq L(G_2)$  e temos uma redução de  $P$  para  $\overline{EQ_{GLC}}$ ;
- ▶ Portanto,  $\overline{EQ_{GLC}}$  e também  $EQ_{GLC}$  são indecidíveis.

$$L(G_1) = L(R)?$$

$$EQ_{GLC/R} = \{ \langle G_1, R \rangle \mid G_1 \text{ é uma GLC,} \\ R \text{ é uma expressão regular e } L(G_1) = L(R) \}$$

- ▶ Idêntico ao caso anterior;
- ▶ Basta substituir  $G_2$  por  $R$  e usar  $R$  tal que  $L(R) = (\Sigma \cup I)^*$ .

$$L(G_1) = T^*?$$

$$TOT_{GLC} = \{\langle G_1 \rangle \mid G_1 \text{ é uma GLC e } L(G_1) = T^* \text{ para algum alfabeto } T\}$$

- ▶ Seja  $G_1$  tal que  $L(G_1) = \overline{L_A} \cup \overline{L_B}$  (LLCs são fechadas em relação à união);
- ▶ Notar que  $L(G_1) = \overline{L_A} \cup \overline{L_B} = \overline{L_A \cap L_B}$ ; portanto,  $L(G_1)$  contém todas as cadeias que **não representam** solução para  $P$ ;
- ▶ Logo,  $P$  tem solução  $\Leftrightarrow L(G_1) \neq T^*$  e temos uma redução de  $P$  para  $\overline{TOT_{GLC}}$ ;
- ▶ Notar que  $T = \Sigma \cup \{a_1, a_2, \dots, a_k\}$  é o único alfabeto sobre o qual  $\overline{L_A} \cup \overline{L_B}$  pode corresponder a um fechamento;
- ▶ Portanto,  $\overline{TOT_{GLC}}$  e também  $TOT_{GLC}$  são indecidíveis.

$L(G_1) \subseteq L(G_2)?$ 

$$SUB_{GLC} = \{\langle G_1, G_2 \rangle \mid G_1 \text{ e } G_2 \text{ são GLCs e } L(G_1) \subseteq L(G_2)\}$$

- ▶ Seja  $G_1$  tal que  $L(G_1) = (\Sigma \cup I)^*$ ;  $L(G_1)$  contém todas as cadeias;
- ▶ Seja  $G_2$  tal que  $L(G_2) = \overline{L_A} \cup \overline{L_B}$ ;  $L(G_2)$  contém apenas as cadeias que **não representam** solução para  $P$ ;
- ▶  $L(G_1) \subseteq L(G_2) \Leftrightarrow L(G_1) = L(G_2)$ ;
- ▶  $L(G_1) \not\subseteq L(G_2) \Leftrightarrow L(G_1) \neq L(G_2)$ ;

$L(G_1) \subseteq L(G_2)?$ 

$$SUB_{GLC} = \{\langle G_1, G_2 \rangle \mid G_1 \text{ e } G_2 \text{ são GLCs e } L(G_1) \subseteq L(G_2)\}$$

- ▶ Logo,  $P$  tem solução  $\Rightarrow L(G_1)$  contém pelo menos uma cadeia que não pertence à  $L(G_2) \Rightarrow L(G_1) \neq L(G_2) \Rightarrow L(G_1) \not\subseteq L(G_2)$ ;
- ▶ Além disso,  $P$  não tem solução  $\Rightarrow L(G_2)$  contém todas as cadeias  $\Rightarrow L(G_1) = L(G_2) \Rightarrow L(G_1) \subseteq L(G_2)$ ;
- ▶ Temos uma redução de  $P$  para  $\overline{SUB_{GLC}}$ ;
- ▶ Portanto,  $\overline{SUB_{GLC}}$  e também  $SUB_{GLC}$  são indecidíveis.

$L(R) \subseteq L(G_1)?$ 

$$SUB_{R/GLC} = \{ \langle R, G_1 \rangle \mid R \text{ é uma expressão regular,} \\ G_1 \text{ é uma GLC e } L(R) \subseteq L(G_1) \}$$

- ▶ Idêntico ao caso anterior;
- ▶ Basta substituir  $G_1$  por  $R$  e usar  $R$  tal que  $L(R) = (\Sigma \cup I)^*$ .
- ▶ Substituir  $G_2$  por  $G_1$ .