Indecidibilidade

Problemas Computáveis

- Máquinas de Turing ou Problemas Computáveis ou Linguagens Recursivamente Enumeráveis ***LER***$^{(*)}$ podem ser divididas em 2 classes:
  - as MT que sempre param (**Algoritmos**), quer aceitem ou não suas entradas (Linguagens *Recursivas ou Decidíveis)*
  - as MT que podem funcionar indefinidamente sobre entradas que elas não aceitam *(Linguagens Indecidíveis)*
- Problemas ou Linguagens *Indecidíveis* são aqueles para os quais não existe nenhum *algoritmo*, ou seja, uma MT que sempre pára.

(*) Linguagens que podem ter seus elementos listados (enumerados) em alguma ordem coincidem com as linguagens aceitas por alguma MT.

Objetivo

- Estudar técnicas para mostrar problemas que são *indecidíveis*
- Estratégia: usar como base o fato de que o problema a seguir é indecidível:
  - Esta MT aceita a si própria (seu código) como entrada?
- Isso vai nos levar a uma situação contraditória, análoga à do problema *hello, world*.

- Ou seja, é indecidível a linguagem que consiste de pares (M,w) tais que:
  1. M é uma MT (adequadamente codificada em binário) com o alfabeto de entrada {0,1}
  2. w é uma cadeia de 0´s e 1´s
  3. M aceita a entrada w.
- Se esse problema com entradas restritas ao alfabeto binário é indecidível, então o mais geral, com qualquer alfabeto, também é indecidível.
- Providência: codificar uma MT com 0´s e1´s independentemente de quantos estados ela tenha.
- Dessa forma, podemos tratar qualquer cadeia binária como se fosse uma MT, e vice-versa.
- Se a cadeia não for uma representação bem-formada de alguma MT, podemos imaginá-la como a representação de uma MT sem movimentos. Assim, qualquer cadeia binária pode ser uma MT.

Codificando uma MT

- Cada estado(q_1, q_2,...), símbolo da fita (X_1 ,X_2...) e sentido L (D_1) ou R (D_2)é representado como um inteiro.
- Cada regra de transição $\delta(q_i,X_j) = (q_k,X_l,D_m)$, para alguns inteiros i,j,k,l,m ≥ 1, é codificada como $0^i10^j10^k10^l10^m$ Como i,j,k,l,m são pelo menos 1, não existe ocorrência de dois ou mais 1´s consecutivos.
- Um código para a MT M inteira consiste em todos os códigos para as transições, em alguma ordem, separados por pares de 1´s:

$$\mathbf{C_1 11 C_2 11 ... C_{n-1} 11 C_n}$$

onde cada um dos C´s é o código para uma transição de M.

Exemplo

- Seja a MT M= ({q_1,q_2,q_3,}, {0,1}, {0(X_1),1(X_2),B(X_3)}, δ, q_1, B, {q_2}), onde δ consiste nas regras:

$$\delta(q_1, 1) = (q_3, 0, R)$$

$$\delta(q_3, 0) = (q_1, 1, R)$$

$$\delta(q_3, 1) = (q_2, 0, R)$$

$$\delta(q_3, B) = (q_3, 1, L)$$

Exemplo

- Seja a MT M= ({q_1,q_2,q_3,}, {0,1}, {0(X_1),1(X_2),B(X_3)}, δ, q_1, B, {q_2}), onde δ consiste nas regras:

$$\delta(q_1, 1) = (q_3, 0, R) \qquad \mathbf{\delta(q_1, X_2) = (q_3, X_1, D_2)}$$
$$\delta(q_3, 0) = (q_1, 1, R)$$
$$\delta(q_3, 1) = (q_2, 0, R)$$
$$\delta(q_3, B) = (q_3, 1, L)$$

- Os códigos para cada uma dessas regras, são:

0100100010100 $\qquad \mathbf{0^1 1 0^2 1 0^3 1 0^1 1 0^2}$

0001010100100

00010010010100

000100010010010

- Um código para M é:

0100100010100**11**0001010100100**11**0001001001010**11**0001000100010010

- Existem outros códigos possíveis para M. Em particular, os códigos para as 4 transições podem ser listados em 4! ordens distintas, o que nos dá 24 códigos para M.

Vamos enumerar as cadeias binárias que representam MTs:

-se w é uma cadeia binária, trate ‘1w’ como um inteiro binário i. Então chamaremos w_i a i=ésima cadeia. Isto é:

- ε (1) é a primeira (1-ésima) cadeia.

- 0 (10) é a segunda (2-ésima) cadeia

- 1 (11) é a terceira (3-ésima) cadeia

-10 (110) é a quarta (4-ésima) cadeia, etc.

-Assim, todas as cadeias podem ser enumeradas e w_i se refere à i-ésima cadeia dessa sequência.

Linguagem da diagonalização

- *Def*. L_d, a linguagem da diagonalização, consiste em todas as cadeias w_i tais que w_i não está em $L(M_i)$. Ou seja, a MT representada por w_i não aceita w_i como entrada.
- L_d não tem nenhuma MT que a aceite (veja que isso é mais forte do que mostrar que ela é indecidível). Ela é não-computável.
- Sua existência provoca o paradoxo de discordar dela própria ao receber um código para si mesma como entrada.

Teorema: **L_d não é uma LRE, isto é, não existe nenhuma MT que aceite L_d.**

Prova: Suponha que L_d fosse L(M) para alguma MT M. Tendo em vista que L_d é uma linguagem sobre {0,1}, M teria um código i (w_i); ou seja, M= M_i.

w_i está em L_d?

- Se sim, então M_i aceita w_i. Mas por definição de L_d, w_i não está em L_d, porque L_d contém somente os valores w_j tais que M_j *não* aceita w_j.
- Se w_i não está em L_d, então M_i não aceita w_i. Portanto, pela definição de L_d, w_i *está* em L_d.

Como w_i não pode estar e não estar em L_d, concluímos que há uma contradição na suposição de que M existe. Isto é, L_d não é uma LRE.

Refinando as LRE

- 2 classes:
  - Linguagens cujas MT não apenas reconhecem a linguagem, mas nos dizem quando decide que a cadeia de entrada não pertence a ela. Ou seja, ela sempre pára, independentemente do fato de alcançar ou não um estado final *(Algoritmos)*
  - Linguagens aceitas por MT sem garantia de parada: se a entrada estiver na linguagem, eventualmente saberemos, mas se a entrada não estiver na linguagem, então a MT poderá continuar funcionando para sempre e nunca teremos certeza de que a entrada não será aceita mais tarde.

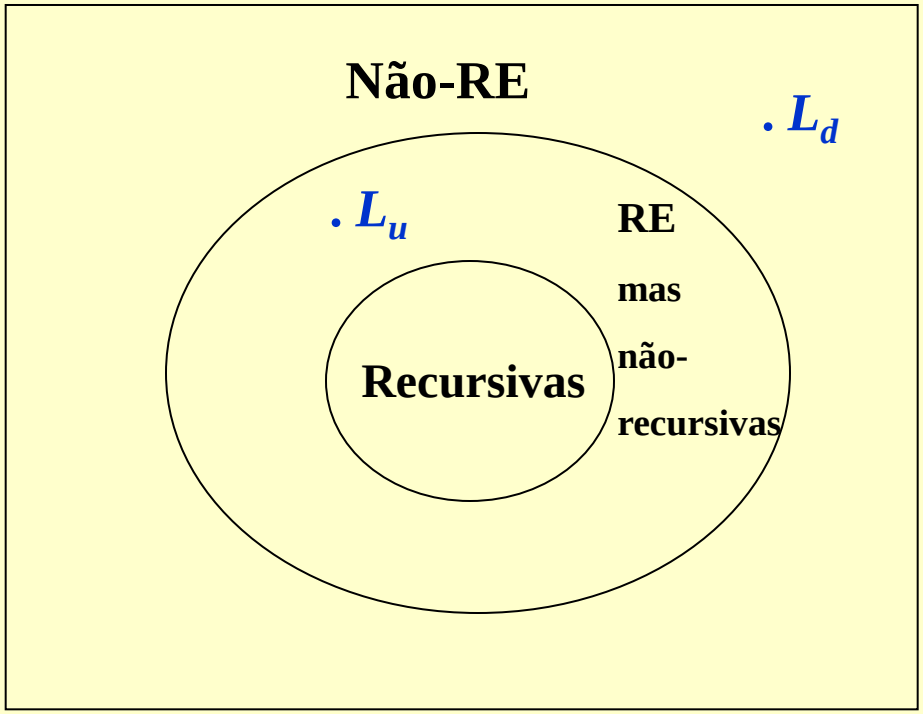
Linguagens Recursivas

Def.: Uma linguagem L é ***recursiva*** se L = L(M) para alguma MT M tal que:

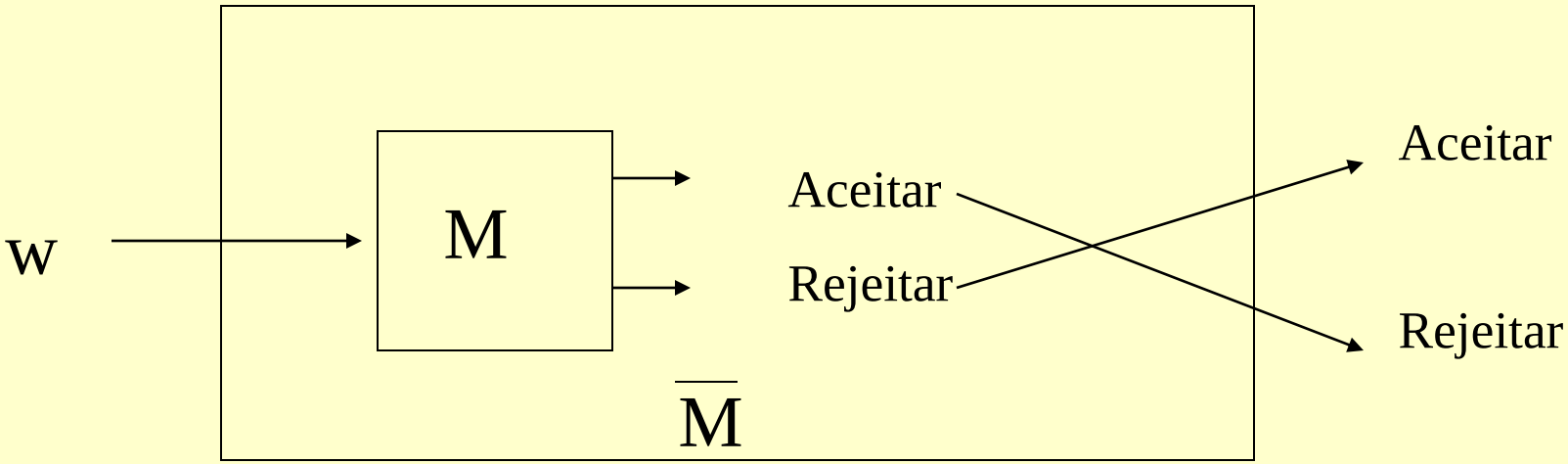
1. Se w está em L, então M aceita (e portanto pára)
2. Se w não está em L, então M pára eventualmente, embora nunca entre em um estado final.

L, vista como um problema, é dita ***decidível*** se for uma linguagem recursiva, ou ***indecidível***, se não for uma linguagem recursiva.

A divisão entre problemas decidíveis e indecidíveis é frequentemente mais relevante do que a divisão entre RE e não-RE, dado que uma MT que não pára sequer nos dá a resposta de que uma entrada não é aceita, ou seja, “não resolve o problema”.



Teorema: **Se L é uma linguagem recursiva, então $\overline{L}$ também o é.**

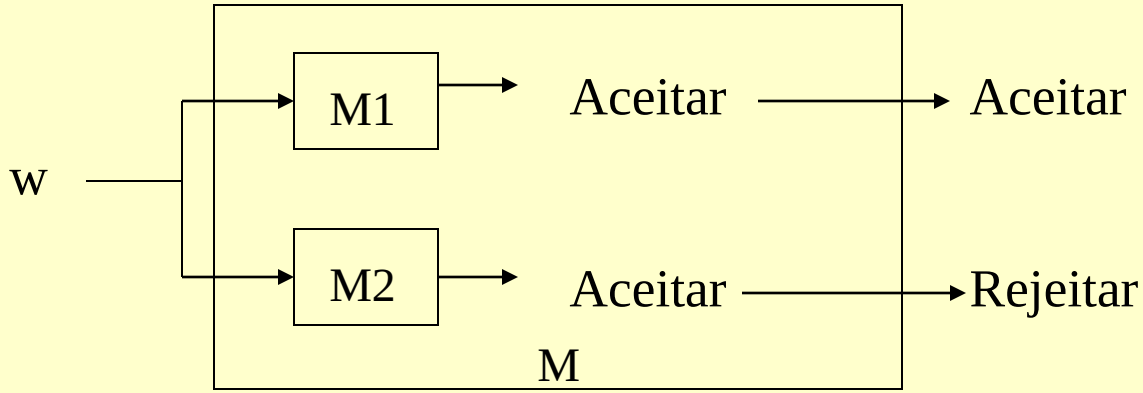


Se L = L(M), então construímos $\overline{M}$ tal que $\overline{L}$ = L($\overline{M}$) como acima. Como M tem a garantia de parar, então $\overline{M}$ também terá. Além disso, $\overline{M}$ aceita exatamente as cadeias que M não aceita. Desse modo, $\overline{M}$ aceita $\overline{L}$, e $\overline{L}$ é RE.

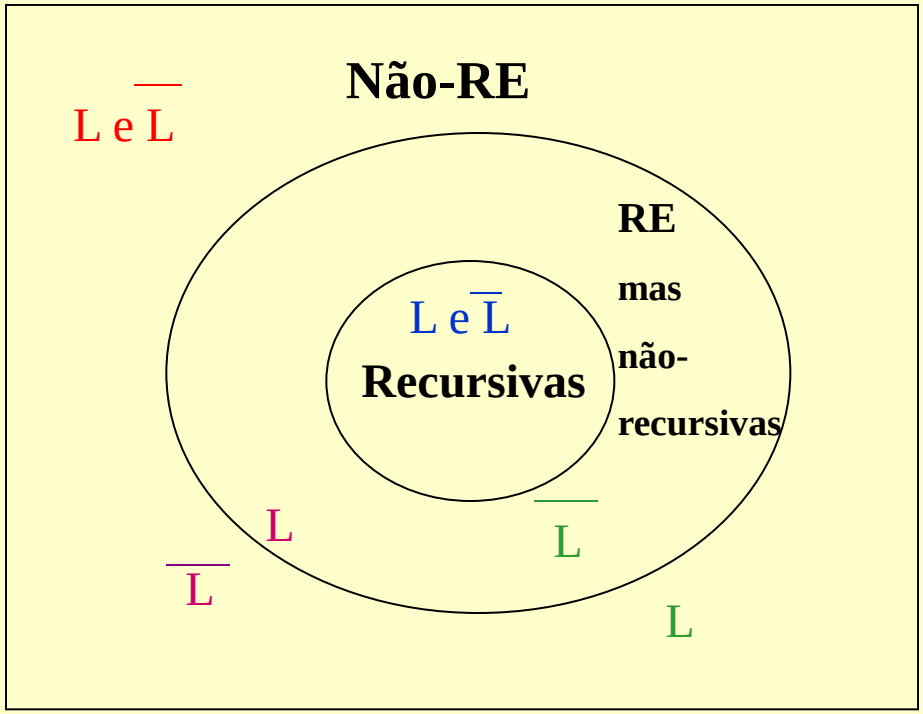
Esse resultado nos permite provar que uma linguagem não é recursiva, se soubermos que seu complemento não é recursivo.

Um outro resultado mais restrito diz que apenas as linguagens recursivas são fechadas sob a operação de complemento:

Teorema: **Se uma linguagem L e seu complemento são ambas RE, então L é recursiva (e, pelo Teorema anterior, $\overline{L}$ também é recursiva).**

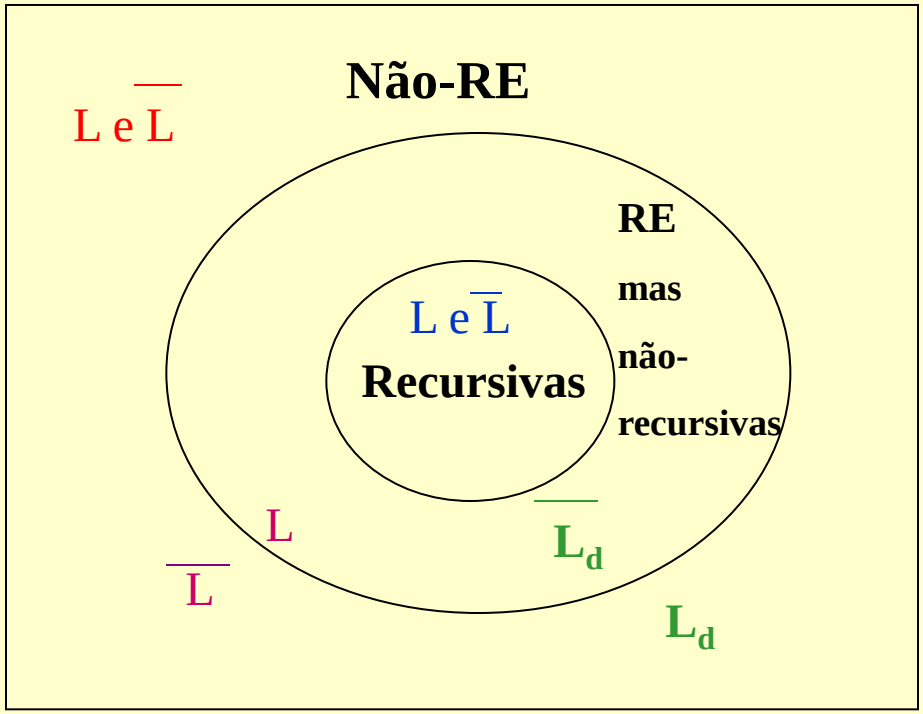


Resumindo.....



Exemplo: Sabemos que L_d não é RE. Assim, $\overline{L_d}$ não pode ser recursiva. Ela seria não-RE ou RE mas não-recursiva? De fato, $\overline{L_d}$ é RE mas não-recursiva: ela consiste das cadeias w_i tal que M_i aceita w_i, ou seja, dos pares (M,w) tais que M aceita w.

Resumindo.....



Exemplo: Sabemos que L_d não é RE. Assim, $\overline{L_d}$ não pode ser recursiva. Ela seria não-RE ou RE mas não-recursiva? De fato, $\overline{L_d}$ é RE mas não-recursiva: ela consiste das cadeias w_i tal que M_i aceita w_i, ou seja, dos pares (M,w) tais que M aceita w.

A linguagem universal

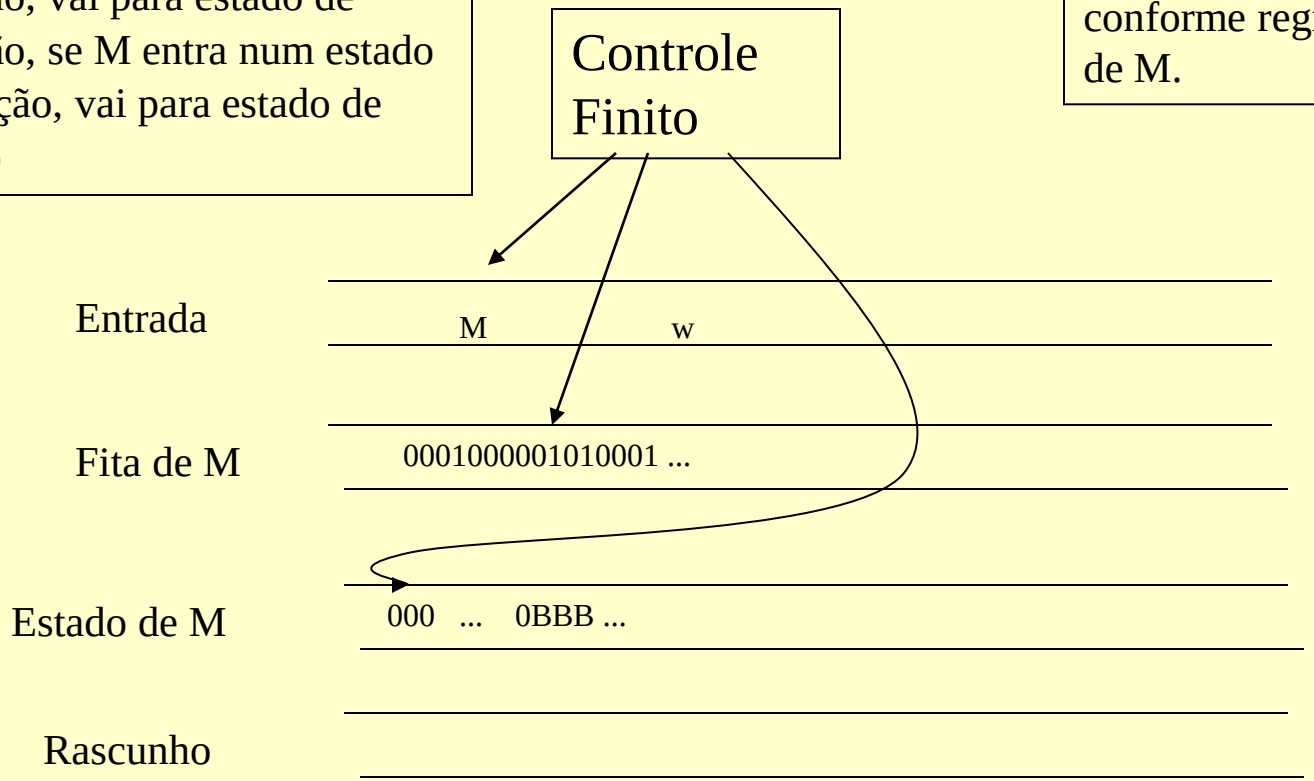
- **Def**.: ***L_u*, a *linguagem universal***, é o conjunto de cadeias binárias que codificam um par (M,w), M é uma MT com alfabeto de entrada binário e w é uma cadeia em {0,1}*, tal que w pertence a L(M).
- Ou seja, L_u é o conjunto de cadeias que representam uma MT e uma entrada aceita por essa MT. Então ela difere da L_d pelo fato de aceitar a cadeia, enquanto L_d não aceita.
- É possível definir uma MT U tal que $L_u = L(U)$ *(veja descrição de U na bibliografia).* Logo, L_u é RE.

Esboço da MT Universal - U

U: MT de N Fitas

1. Simule M com entrada w
2. Se M entra num estado de aceitação, vai para estado de aceitação, se M entra num estado de rejeição, vai para estado de rejeição

Fita e estados de M codificados conforme regra de codificação de M.



Indecidibilidade de L_u

- A indecidibilidade de L_u nos permite mostrar (pelo processo de redução) a indecidibilidade de outros problemas P *(ou seja, que não existe um algoritmo para P)*, independentemente de P ser ou não RE.
- Já a redução de L_d a P só é possível se P não for RE e assim L_d não pode ser usada para mostrar a indecidibilidade dos problemas que são RE mas não-recursivos.
- Se quisermos mostrar que P não é RE *(não existe uma MT)*, então somente L_d poderá ser usada.

Repare que:

- L_d = { w | MT codificada por w não aceita w}
- L_u = { (M,w) | M aceita w}
- $\overline{L_u}$ = { $(\overline{M},\overline{w})$ | $\overline{M}$ aceita $\overline{w}$}, ou seja, w não é aceita por $\overline{M}$
- Podemos então reduzir L_d a $\overline{L_u}$ transformando cada entrada w de L_d em $(w, \overline{w})$. Assim, quando a MT para L_u parar (i.e. M aceita $\overline{w}$), significa que a MT para L_d pararia (i.e. w não aceitaria w).

• **Teorema**: **L_u é RE mas não é recursiva. Isto é, L_u é indecidível.**

Prova: Sabemos que L_u é RE. Vamos supor que L_u seja recursiva. Então, $\overline{L_u}$ também seria recursiva. Porém, se tivermos uma MT M para aceitar $\overline{L_u}$, então poderemos construir uma MT para aceitar L_d *(conjunto de pares (M,w) tal que w não está em L(M))**. Mas já sabemos que L_d não é RE. Então temos uma contradição de que L_u é recursiva.

• Logo, L_u é RE mas não-recursiva; é indecidível.

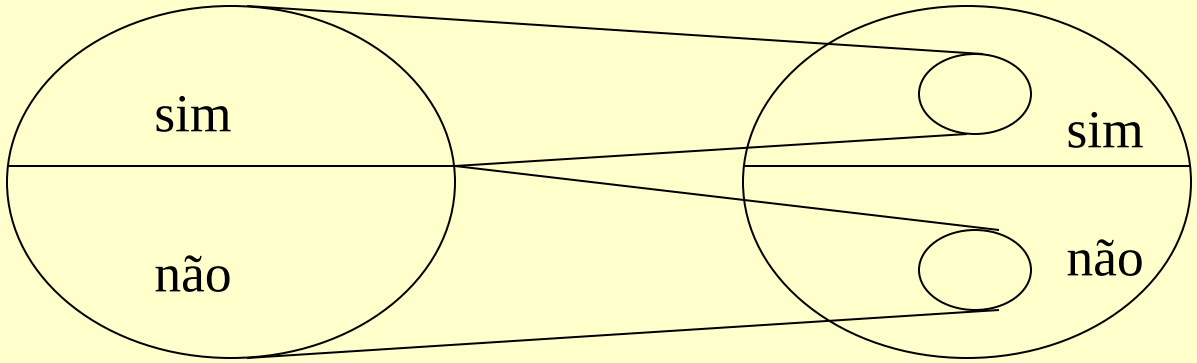
**Reduzindo L_d a $\overline{L_u}$*

O Problema da Parada

- Dado um programa *P* e uma entrada *e*, existe um algoritmo que responda sim ou não, dependendo se *P* pára ou não para a entrada *w*, respectivamente?
- Esse problema equivale à linguagem universal L_u - substituindo o par (M,w) pelo par (*P*,*e*).
- Dessa forma, o Problema da Parada é análogo à L_u, e igualmente **indecidível**.

Redução de Problemas

- P_1 *se reduz a* P_2 quando existe um algoritmo que converte instâncias de P_1 em instâncias de P_2 que têm a mesma resposta.(P_1 é o que se conhece; P_2 é a incógnita – nunca o oposto)



Formalmente, uma redução de P_1 a P_2 é uma MT que toma uma instância de P_1 gravada em sua fita e pára com uma instância de P_2 em sua fita. Ou seja, as reduções são como programas que transformam uma entrada numa saída.

Teorema: **Se existe uma redução de P_1 a P_2, então:**

1. **Se P_1 é indecidível, então P_2 também o é.**
2. **Se P_1 é não-RE, então P_2 também o é.**

Problemas indecidíveis sobre MT

- MT que aceitam a linguagem vazia
  - Sejam:

$L_e = \{M| L(M) = \varnothing\}$ – *conj. das MT (codificadas) cuja linguagem é vazia*

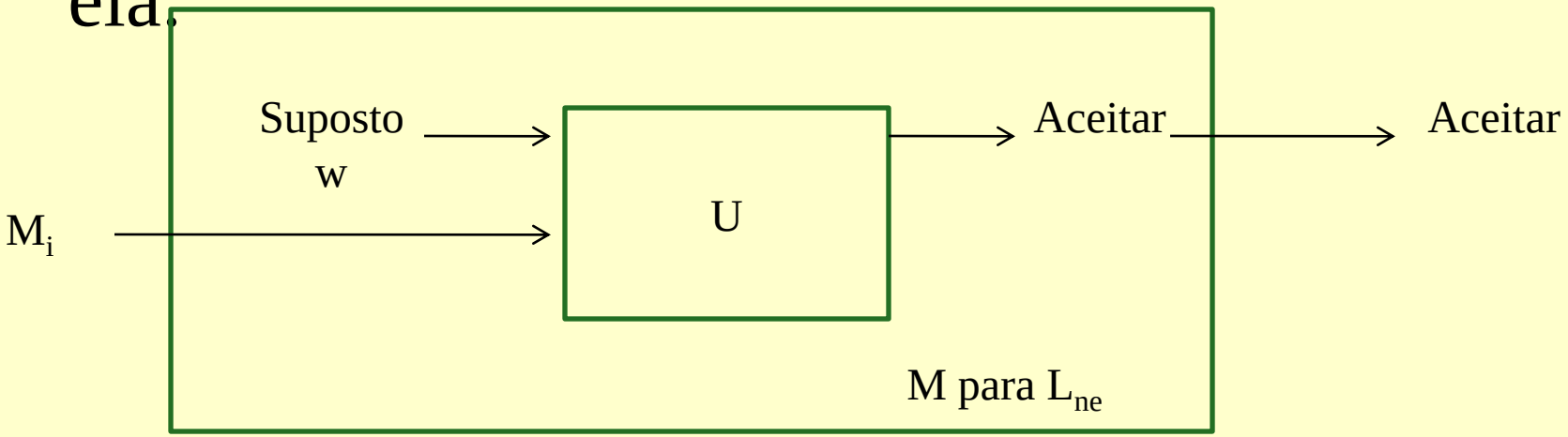
$L_{ne} = \{M| L(M) \neq \varnothing\}$ – *conj. das MT (codificadas) cuja linguagem contém ao menos uma cadeia*

Logo, L_e e L_{ne} são **complementos** uma da outra.

L_{ne} é a "mais fácil" das duas e é RE, mas não recursiva. Por outro lado, L_e é não-RE.

Teorema: L_{ne} é recursivamente enumerável.

Prova: Temos que exibir uma MT (ND) para ela.



A operação de M segue:

-M toma como entrada o código de uma MT M_i

-Usando sua capacidade não-determinística, M supõe uma entrada w que M_i poderia aceitar.

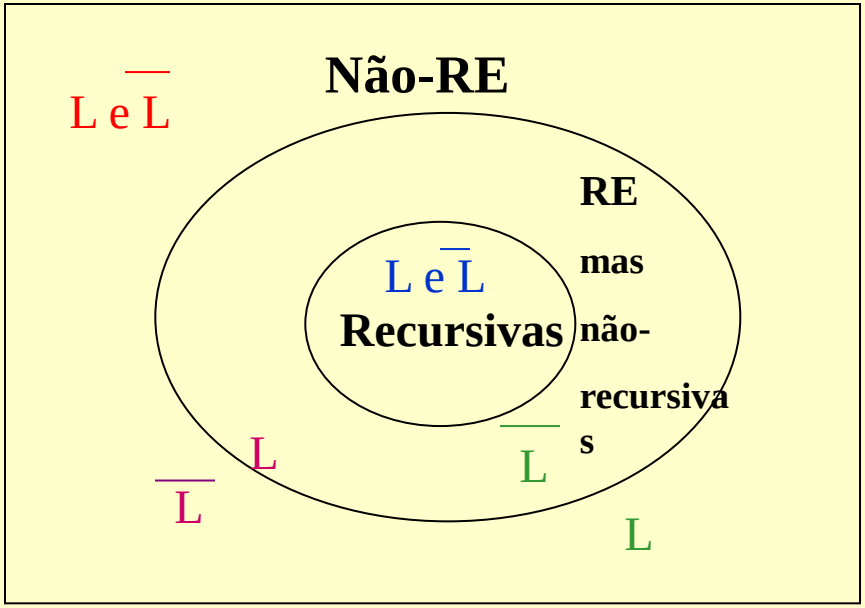
-M testa se M_i aceita w. Para essa parte, M pode simular a MT universal U que aceita L_u.

-Se M_i aceita w, então M aceita sua própria entrada, M_i.

Dessa maneira, se M_i aceita até mesmo uma única cadeia, M irá supor essa cadeia (entre todas as outras) e aceitará M_i. Porém, se $L(M_i) = \varnothing$, então nenhuma suposição de w levará à aceitação por M_i e assim M não aceitará M_i. Desse modo, $L(M) = L_{ne}$.

Teorema: **L_{ne} é não-recursiva.**

$\therefore$ Teorema: **L_e não é RE.**



Teorema de Rice: **Todas as propriedades não-triviais (satisfeitas por nenhuma ou por todas as REs) das linguagens RE são indecidíveis.**

Instâncias:

- Se a linguagem aceita por uma MT é finita
- Se a linguagem aceita por uma MT é uma LR
- Se a linguagem aceita por um MT é uma LLC
- Se a linguagem aceita por uma MT é não vazia

- Atenção: características sobre MT, e não sobre as linguagens aceitas, podem ser decidíveis. Ex. é decidível se uma MT tem cinco estados. Basta examinar o código da MT e contar o número de estados que aparecem em qualquer de suas transições.

Há uma analogia do Teorema de Rice para programas: qualquer propriedade não-trivial que envolva aquilo que o programa faz tem de ser indecidível.

Teorema: **É indecidível se uma GLC é ou não ambígua**

Teorema: **Sejam G1 e G2 gramáticas livres de contexto, e seja R e uma expressão regular. Então, as seguintes questões são indecidíveis:**

a) **$L(G1) \cap L(G2) = \emptyset$?**
b) **$L(G1) = L(G2)$?**
c) **$L(G1) = L(R)$?**
d) **$L(G1) = T^*$ para algum alfabeto T?**
e) **$L(G1) \subseteq L(G2)$?**
f) **$L(R) \subseteq L(G1)$?**

Hierarquia das Classes de Máquinas e Linguagens