Justificativa Basta demonstrar que existe pelo menos uma linguagem recursiva que não é sensível ao contexto. Como instância desse fato, podem-se citar todas as linguagens que incluem a cadeia vazia, uma vez que, conforme foi discutido anteriormente na Seção 5.1, as linguagens sensíveis ao contexto não incluem tais cadeias.

Um exemplo não-trivial, desta vez sem recorrer à cadeia vazia, é a linguagem L_R, apresentada no final da Seção 5.7, quando foi empregada para demonstrar a existência de linguagens que não são sensíveis ao contexto (Teorema 5.8).

Partindo-se (Teorema 5.7) de uma enumeração de todas as gramáticas sensíveis ao contexto $(G_1, G_2, ...)$, e também de uma enumeração de todas as cadeias possíveis de serem geradas sobre o alfabeto $\{a, b\}$ $(\alpha_1, \alpha_2, ...)$, o Teorema 5.8 mostra que a linguagem

$$L_R = \{\alpha_i \mid \alpha_i \notin L(G_i), \forall\, i \geq 1\}$$

não pode ser sensível ao contexto.

Por outro lado, qualquer que seja a cadeia $\beta \in \Sigma^*$, será sempre possível determinar mecanicamente se β pertence ou não a L_R, o que pode ser feito conforme o método descrito no Algoritmo 6.3.

Algoritmo 6.3 ($\beta \in L_R$?) *Determinação da pertinência de $\beta \in \Sigma^*$ a L_R, definida no Teorema 5.8.*

- Entrada: uma cadeia $\beta \in \Sigma^*$;
- Saída: SIM, se $\beta \in L_R$; NÃO, se $\beta \notin L_R$;
- Método:

    1. Compara-se β com cada uma das cadeias α_i, até que haja coincidência entre ambas;
    2. Seleciona-se a gramática G_i correspondente à cadeia α_i (lembrar que foi estabelecida uma função bijetora entre o conjunto das gramáticas e o conjunto das cadeias);
    3. Determina-se se $\alpha_i \in L(G_i)$. Como se trata de linguagens sensíveis ao contexto, tal determinação pode sempre ser efetuada;
    4. Se o resultado for que $\alpha_i \in L(G_i)$, então, por construção, $\alpha_i \notin L_R$ e a resposta é NÃO. Caso o resultado seja que $\alpha_i \notin L(G_i)$, então, por construção, $\alpha_i \in L_R$ e a resposta é SIM.

Logo, conforme o Algoritmo 6.3, é sempre possível determinar se uma cadeia β qualquer pertence a L_R. Em outras palavras, L_R é uma linguagem decidível e, portanto, recursiva, sem ser sensível ao contexto. ■

6.5 Linguagens que não são Recursivas

A classe das linguagens recursivas não é a mais abrangente que se conhece. Ao contrário, é possível demonstrar que existe pelo menos uma linguagem não-recursiva.

Teorema 6.4 (Linguagens não-recursivas) *Existem linguagens que não são recursivas.*

Justificativa É suficiente provar que existe pelo menos uma linguagem não-recursiva. Essa demonstração consiste na definição da linguagem L_U, conforme abaixo, e na prova, por contradição, de que a mesma não pode pertencer à classe das linguagens recursivas. Em outras palavras, que esta linguagem não é decidível.

$$L_U = \{C(M)w \in \Sigma^* \mid w \in L(M)\}$$

onde:

- $C(M)$ é a codificação de uma Máquina de Turing com fita limitada, cujo alfabeto de entrada é Σ, como cadeia sobre o próprio alfabeto Σ;
- $w \in \Sigma^*$ é uma cadeia de entrada qualquer para M.

A linguagem L_U é também conhecida como **Linguagem Universal**, uma vez que o problema de se determinar se uma certa cadeia $w \in L(M)$, para uma certa Máquina de Turing M, pode ser reduzido ao problema de se determinar se $C(M)w \in L_U$. Conforme será discutido no Capítulo 7, é possível provar que L_U, apesar de não ser recursiva, é uma linguagem recursivamente enumerável. Logo, existe uma Máquina de Turing que aceita L_U. Tal máquina, denotada M_U, é denominada **Máquina de Turing Universal**. Em outras palavras, $L_U = L(M_U)$.

L_U presume uma certa forma de codificação das Máquinas de Turing com fita limitada, codificação esta que, em princípio, pode ser feita sobre o mesmo alfabeto de entrada Σ de M. Em outras palavras: se M é uma Máquina de Turing sobre um alfabeto Σ, codifica-se M — denotada $C(M)$ — como uma cadeia sobre o próprio alfabeto Σ.

Essa codificação pode ser feita de várias maneiras. A forma escolhida é irrelevante, uma vez que os resultados obtidos dela independem.

L_U deve ser entendida, portanto, como a linguagem formada pelo conjunto das cadeias $C(M)w$, sendo M uma Máquina de Turing com fita limitada qualquer, com alfabeto de entrada Σ, e w uma cadeia qualquer sobre Σ^*, desde que w pertença à linguagem definida por M.

A forma de codificação $C(M)$ de uma máquina M é, como foi antecipado, irrelevante. É apenas necessário estabelecer uma convenção que permita a correspondência unívoca entre cada máquina distinta e a cadeia correspondente que a representa, e também garantir que $C(M)$ e w sejam cadeias construídas sobre um mesmo alfabeto. Serão consideradas duas possibilidades de codificação:

1. Codificar a máquina M como uma cadeia sobre o próprio alfabeto de entrada Σ de M, deixando a cadeia de entrada w inalterada, resultando na cadeia combinada $C(M)w$;
2. Codificar não apenas M, mas também a própria cadeia de entrada w, como cadeias sobre um segundo alfabeto fixo Δ, resultando em cadeias do tipo $C(M)C(w)$.

Para efeito prático de demonstração de uma forma de codificação que possa ser aplicada a qualquer Máquina de Turing, com qualquer alfabeto de entrada, será adotada a segunda alternativa.

Apresenta-se a seguir, portanto, um esquema genérico de codificação de máquinas e cadeias de entrada quaisquer sobre o alfabeto $\Delta = \{a, b, c\}$, escolhido arbitrariamente. Note-se, na codificação proposta, que, apesar de o alfabeto Δ possuir apenas três símbolos, ela permite a representação de Máquinas de Turing com qualquer quantidade de símbolos em seu alfabeto de entrada Σ, e também da própria cadeia $w \in \Sigma^*$ a ser processada por M.

- Cada estado não-final do conjunto de estados $Q = \{q_0, q_1, q_2...q_n\}$ será representado, respectivamente, pela cadeia $aa, aaaa, aaaaaa...a^{2(n+1)}$ (ou seja, uma quantidade par de símbolos a);

  $C(q_i) = a^{2(i+1)}$, para $q_i \in Q - Q_F$;

- Cada estado final do conjunto de estados $Q = \{q_0, q_1, q_2...q_n\}$ será representado, respectivamente, pela cadeia $a, aaa, aaaaa...a^{2n+1}$ (ou seja, uma quantidade ímpar de símbolos a);

  $C(q_i) = a^{2i+1}$, para $q_i \in Q_F$;

- O símbolo especial "$<$" será representado pela cadeia ba;

  $C(B) = ba$;

- O símbolo especial "B" será representado pela cadeia bba;

  $C(B) = bba$;

- Cada elemento σ_n do alfabeto $\Sigma = \{\sigma_0, \sigma_1, \sigma_2...\sigma_n\}$ será representado, respectivamente, pela cadeia $bbba, bbbba, bbbbba...b^{n+3}a$;

  $C(\sigma_i) = b^{i+3}a,\ \sigma_i \in \Sigma$;

- O sentido de movimentação do cursor de acesso será representado como c (esquerda) ou cc (direita).

  $C(E) = c$;

  $C(D) = cc$;

Assim, cada uma das transições $\delta(q_i, \sigma_m) = (q_j, \sigma_n, E)$ de M poderá ser representada como:

- Se $q_i \notin F, q_j \notin F$, então $a^{2(i+1)}b^{m+3}aa^{2(j+1)}b^{n+3}ac$
- Se $q_i \in F, q_j \notin F$, então $a^{2i+1}b^{m+3}aa^{2(j+1)}b^{n+3}ac$
- Se $q_i \notin F, q_j \in F$, então $a^{2(i+1)}b^{m+3}aa^{2j+1}b^{n+3}ac$
- Se $q_i \in F, q_j \in F$, então $a^{2i+1}b^{m+3}aa^{2j+1}b^{n+3}ac$

De maneira análoga, caso o sentido de movimentação do cursor de acesso seja D e não E, basta substituir a subcadeia c por cc nas formas gerais acima. Por exemplo:

$$\delta(q_i, \sigma_m) = (q_j, \sigma_n, D) \quad \text{e} \quad a^{2(i+1)}b^{m+3}aa^{2(j+1)}b^{n+3}acc, \quad \text{com } q_i \notin F, q_j \notin F$$

A representação de $M = (Q, \Sigma, \Gamma, \delta, q_0, <, B, F)$ será efetivada da seguinte forma:

- Q e Σ podem ser inferidos a partir de δ;
- Se $\Sigma \neq \Gamma$, então codifica-se $\Sigma \cup \Gamma$;
- A função de transição δ será codificada como uma seqüência de cadeias concatenadas conforme a convenção apresentada individualmente para a codificação de cada transição (acima);
- q_0 será convencionado como sendo o primeiro estado a ser referenciado na definição de δ;
- "$<$" e B possuem codificações fixas sobre $\{a, b, c\}$;
- F poderá ser inferido a partir dos estados presentes na definição da função de transição que estejam codificados com uma quantidade ímpar de símbolos a.

Codificações como essa são úteis para a demonstração deste e de diversos outros teoremas, uma vez que permitem que Máquinas de Turing sejam representadas como cadeias de entrada (ou parte de cadeias de entrada) para outras Máquinas de Turing.

Exemplo 6.4 Seja a máquina M definida na Figura 6.5:

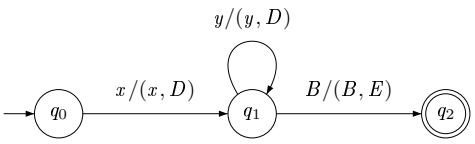

Figura 6.5. Máquina de Turing M que aceita xy^*

A codificação de seus componentes, de acordo com as convenções anteriormente apresentadas, ocorre conforme a Figura 6.6.

Q	q_0	aa
	q_1	$aaaa$
	q_2	$aaaaa$
-	$<$	ba
-	B	bba
Σ	x	$bbba$
	y	$bbbba$
δ	$\delta(q_0,x) = (q_1,x,D)$	$aabbbaaaaabbbacc$
	$\delta(q_1,y) = (q_1,y,D)$	$aaaabbbbaaaaabbbbacc$
	$\delta(q_1,B) = (q_2,B,E)$	$aaaabbaaaaaabbac$

Figura 6.6. Codificação dos elementos de M sobre $\{a, b, c\}$

A representação completa desta máquina é obtida concatenando-se as cadeias que representam as suas transições, conforme mostrado na Figura 6.7:

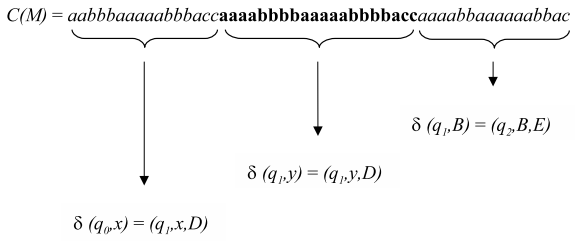

Figura 6.7. $C(M) : M$ como uma cadeia sobre $\{a, b, c\}$

□

A Figura 6.8 ilustra a codificação $C(w)$ para a cadeia de entrada $w = xyyB$.

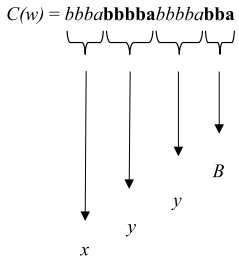

Figura 6.8. $C(w) : w$ como uma cadeia sobre $\{a, b, c\}$

A cadeia da Figura 6.9 representa a codificação de M seguida da codificação de uma cadeia w, ambas construídas sobre o mesmo alfabeto de codificação $\Delta = \{a, b, c\}$, ou seja, $C(M) - C(w)$. O símbolo especial "$-$" é usado para separar a codificação da máquina da codificação da cadeia:

C(M)-C(w) = aabbbaaaaabbbacc**aaaabbbbaaaaabbbbacc**aaaabbaaaaaabbac-bbba**bbbba**bbbba**bba**

Figura 6.9. $C(M) - C(w) : M$ e w como cadeias sobre $\{a, b, c\}$

Retorna-se agora à demonstração de que L_U não é recursiva. Para isso, será adotada a primeira forma de codificação sugerida, ou seja, aquela em que o alfabeto de codificação empregado é o próprio alfabeto de entrada da máquina M considerada. A segunda forma, exemplificada acima, também pode ser usada, sem que isso implique qualquer alteração nos resultados obtidos.

Suponha-se, portanto, que L_U seja uma linguagem recursiva, e portanto decidível. Isso acarreta a existência de uma Máquina de Turing com fita limitada M que decide L_U. Seja $C(N)w$ a cadeia formada pela concatenação da codificação da máquina N (sobre o alfabeto Σ) com a cadeia de entrada $w \in \Sigma^*$. A análise de $C(N)w$ por M é ilustrada na Figura 6.10.

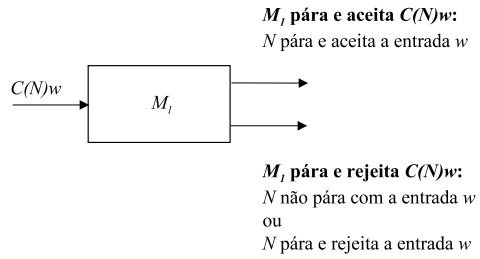

Figura 6.10. L_U não-recursiva: passo 1

Em seguida, constrói-se M_2 a partir de M_1, de modo que, para toda cadeia capaz de fazer M_1 parar, aceitando a entrada, M_2 deverá passar a executar uma seqüência infinita de movimentações. Caso M_1 pare, rejeitando a entrada, M_2 deverá também parar, rejeitando a entrada (figura 6.11).

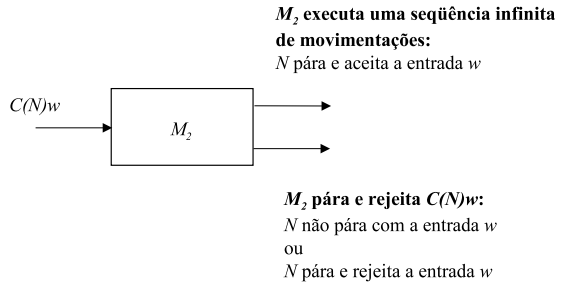

Figura 6.11. L_U não-recursiva: passo 2

Tal modificação é de fácil realização: basta fazer com que M_1, ao atingir uma configuração de aceitação, transite para um novo estado, criado especialmente para essa finalidade, em M_2, e lá permaneça indefinidamente lendo símbolos na fita de entrada e deslocando o cursor de acesso para a direita.

Seja M_3 uma Máquina de Turing com fita limitada que duplica a cadeia fornecida como entrada. Se a cadeia sobre a fita de entrada é γ, ao término do processamento a fita de trabalho conterá $\gamma\gamma$, conforme a Figura 6.12:

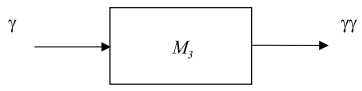

Figura 6.12. L_U não-recursiva: passo 3

Considere-se agora a máquina M_4, obtida pela combinação das máquinas M_3 e M_2, de tal forma que a saída de M_3 seja usada como entrada para M_2 (Figuras 6.13 e 6.14).

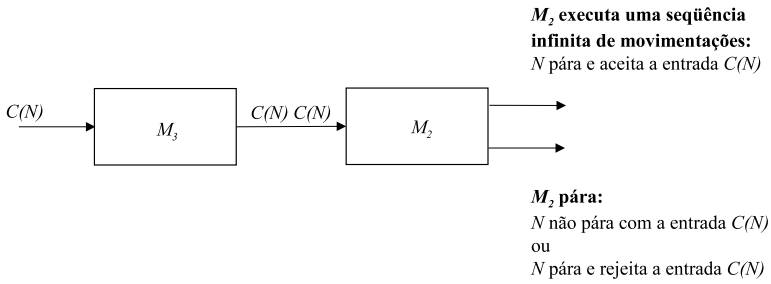

Figura 6.13. L_U não-recursiva: passo 4

ou ainda:

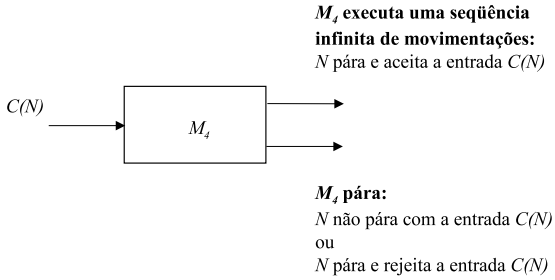

Figura 6.14. L_U não-recursiva: passo 5

Faça-se $N = M_4$, ou seja, submeta-se a M_4 uma codificação da própria máquina M_4. Isso é possível, uma vez que M_4 aceita cadeias sobre um certo alfabeto Σ, e $C(M_4)$ será codificada também como uma cadeia sobre este mesmo Σ. No caso considerado, $\Sigma = \{a, b, c\}$, porém qualquer outro alfabeto produziria os mesmos resultados (ver Figura 6.15).

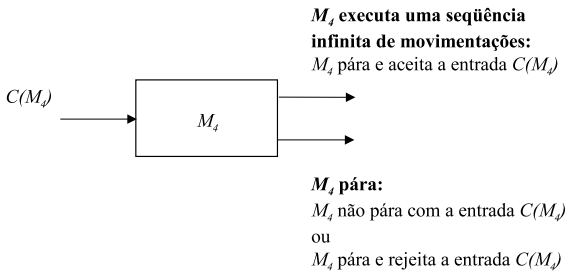

Figura 6.15. L_U não-recursiva: passo 6

A Figura 6.15 remete a uma contradição: 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 nossa hipótese inicial não é válida, ou seja, L_U não pode ser uma linguagem recursiva.

Naturalmente, esses resultados são válidos apenas para os casos de máquinas e entradas arbitrárias, não conhecidas *a priori*. Eventualmente, esses problemas podem ser resolvidos para combinações de máquinas e/ou entradas predeterminadas. ■

Uma outra importante linguagem não-recursiva é a linguagem L_P, que representa um problema fundamental da computação, conhecido como o **Problema da Parada** da Máquina de Turing. Formalmente:

$$L_P = \{C(M)w \in \Sigma^* \mid M \text{ pára com a entrada } w\}$$

Se L_P fosse uma linguagem recursiva, e portanto decidível, estaria demonstrada a existência de um algoritmo capaz de determinar se uma certa máquina (ou programa) pára ao processar uma dada entrada. A prova de que L_P é não-recursiva (e portanto indecidível) demonstra a inexistência de tal algoritmo.

Teorema 6.5 (L_P não-recursiva) *A linguagem L_P não é recursiva.*

Justificativa A mesma seqüência de passos que foi usada para demonstrar que L_U não é recursiva também pode ser usada, com pequenas modificações, para demonstrar que L_P não é recursiva.

Admita-se, inicialmente, que L_P seja recursiva. Considere-se, também, a seqüência das Figuras 6.10 até 6.15, modificadas de forma a não importar se a cadeia de entrada w é aceita ou rejeitada pela Máquina de Turing correspondente, mas apenas se w permite que ela páre ou não.

As Figuras 6.16 até 6.20 incorporam tais modificações. O passo 3 da presente demonstração coincide com o passo 3 ilustrado para o caso da linguagem L_U (Figura 6.12), e não será repetido.

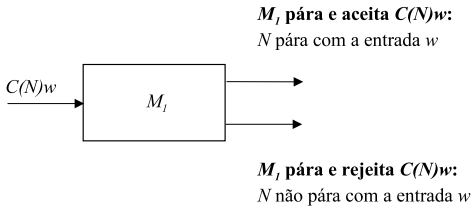

Figura 6.16. L_P não-recursiva: passo 1

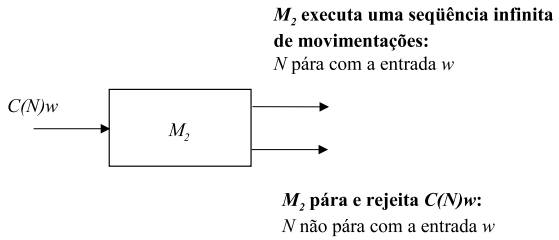

Figura 6.17. L_P não-recursiva: passo 2

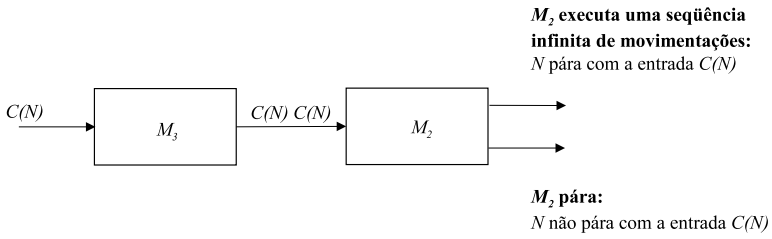

Figura 6.18. L_P não-recursiva: passo 4

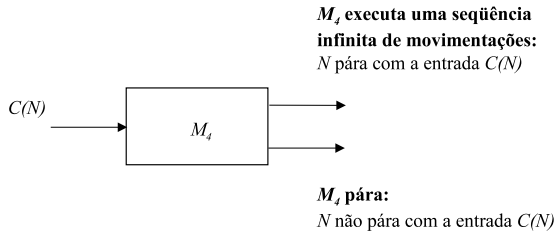

Figura 6.19. L_P não-recursiva: passo 5

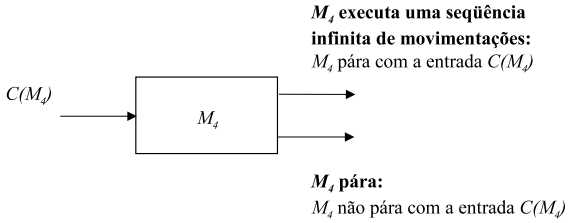

Figura 6.20. L_P não-recursiva: passo 6

Como conclusão, a seqüência de passos acima induz a uma contradição: M_4 pára com a cadeia $C(M_4)$ se e apenas se M_4 não pára com a cadeia $C(M_4)$. Logo, a hipótese inicial é falsa e L_P não é recursiva. ■

A linguagem L_K a seguir apresentada é também um exemplo clássico de linguagem não-recursiva:

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

Esta linguagem pressupõe uma certa ordenação (por exemplo, lexicográfica) das cadeias w_i sobre um certo alfabeto Σ, e também do conjunto das Máquinas de Turing sobre o mesmo alfabeto. Ela compreende as cadeias w_i que são aceitas pelas Máquinas de Turing M_i correspondentes, e pressupõe uma bijeção entre o conjunto das máquinas M_i e o conjunto das cadeias $w_i, i \geq 0$, conforme a Tabela 6.1.

w_0	w_1	w_2	...	w_n	...
$\updownarrow$	$\updownarrow$	$\updownarrow$		$\updownarrow$	
M_0	M_1	M_2	...	M_n	...

Tabela 6.1. Bijeção entre cadeias e Máquinas de Turing

Teorema 6.6 (L_K não-recursiva) *A linguagem L_K não é recursiva.*

Justificativa Conforme o Teorema 7.21 (apresentado mais adiante), uma linguagem L é recursiva se e somente L e seu complemento forem recursivamente enumeráveis. O Teorema 7.15 (também apresentado mais adiante), por sua vez, prova que a linguagem $L_D = \Sigma^* - L_K$ (o complemento de L_K) não é recursivamente enumerável. Logo, L_K não é recursiva. ■

6.6 Propriedades de Fechamento

A seguir serão demonstradas algumas das propriedades de fechamento mais importantes das linguagens recursivas. Esta classe de linguagens é fechada em relação às operações de:

- União
- Concatenação
- Complementação
- Intersecção

As respectivas demonstrações serão apresentadas na seqüência. As linguagens recursivas, no entanto, não são fechadas em relação à operação de:

- Fechamento reflexivo e transitivo (Fecho de Kleene)

resultado este que não será demonstrado neste texto, podendo ser encontrado em [11].

Teorema 6.7 (Fecho na união) *A classe das linguagens recursivas é fechada em relação à operação de união.*

Justificativa Sejam L_1 e L_2 duas linguagens recursivas quaisquer. Então, L_1 é decidida por uma Máquina de Turing M_1 e L_2 é decidida por uma Máquina de Turing M_2. A linguagem $L_3 = L_1 \cup L_2$ é decidida por M_3, construída de acordo com o Algoritmo 6.4.

Algoritmo 6.4 (Fecho na união) *Obtenção de uma Máquina de Turing que decide a união de duas linguagens, a partir das Máquinas de Turing que decidem cada uma das linguagens.*

- Entrada: M_1 e M_2, duas Máquinas de Turing que decidem, respectivamente, as linguagens L_1 e L_2;
- Saída: uma Máquina de Turing M_3 que decide a linguagem $L_1 \cup L_2$;