

Controle de Tráfego Urbano Inteligente

Trabalho Final do Curso de Teoria da Computação

Caio Mello - XXXX
João Roberto - 222217111
Jonas - ZZZZZZZZ
Leandro - ZZZZZZZZ
Vinicius - YYYYYYYYYY

,

Período Letivo
2025.2

Índice

1.	Introdução	3
1.1.	Enunciado do problema	3
1.2.	Contextualização e importância na área de cidades inteligentes.	3
1.3.	Abstrações, notações específicas e corolários significativos	4
2.	Máquinas de Turing	7
2.1.	Máquina de uma única fita (MT_1)	8
2.1.1.	Módulos e submódulos	8
2.1.2.	Submódulo ψ (ou “função de contagem”)	9
2.1.3.	Módulo α	12
2.1.4.	Do custo da contagem no módulo α	14
2.1.4.1.	Análise para $w = y^k * x^m$	15
2.1.4.2.	Análise para $w = x^m y^k$	15
2.1.5.	O custo da limpeza	17
2.1.6.	Conclusão do custo de α	17
2.1.7.	Módulo α na íntegra	18
2.2.	Máquina na íntegra	19
2.3.	Análise de complexidade	19
2.4.	Exemplos	19
3.	Máquina de múltiplas fitas	19
3.1.	Módulos	19
3.2.	Máquina na íntegra	19
3.3.	Análise de complexidade	19
3.4.	Exemplos	19

1. Introdução

1.1. Enunciado do problema

A Companhia Turing Traffic Systems está desenvolvendo um sistema de controle de tráfego urbano baseado em Máquinas de Turing para gerenciar os semáforos de uma cidade inteligente. O objetivo é determinar qual deve ser o próximo estado do semáforo principal de um cruzamento, considerando informações sobre o fluxo de veículos.

O cruzamento possui três avenidas que se encontram no mesmo ponto, e cada avenida possui sensores que contam o número de veículos aguardando em cada direção.

O sistema deve decidir a próxima cor do semáforo principal de acordo com a seguinte política:

I - Se uma avenida tiver mais da metade do total de veículos somados das três avenidas, ela terá prioridade, e o semáforo para essa avenida ficará verde, enquanto os outros ficarão vermelho.

II Caso nenhuma avenida tenha mais da metade dos veículos, o sistema escolhe a avenida com maior número de veículos.

III Em caso de empate, a avenida A tem prioridade sobre B, e B sobre C (ordem A > B > C).

1.2. Contextualização e importância na área de cidades inteligentes.

1.3. Abstrações, notações específicas e corolários significativos

Como parte desse trabalho, temos a missão de modular Máquinas de Turing que solucionem o problema preposto. Para tal, teremos como entrada uma string representando a distribuição de carros dentre as avenidas A, B e C, seguindo o formato:

Seja w uma palavra e $n \in \mathbb{N}$

$$w = \varphi_0 * \dots * \varphi_{n-1} * \varphi_n$$

$$\varphi_k \in \{a, b, c\} \text{ tal que } 0 \leq k \leq n$$

Nesta representação, cada carro na avenida A será uma letra ‘a’, na avenida B, a letra ‘b’, e na C, ‘c’.

Uma outra definição útil é a do conjunto de ocorrências de um conjunto de símbolos ou de um único símbolo (sendo esse o caso do conjunto unário).

Seja w uma palavra, x uma letra e $X = \{x\}$

X_w é o conjunto de ocorrências dos símbolos de X em w

$$X_w = \{j \mid \forall j \in N \text{ tal que } 0 \leq j \leq |w| \text{ e } \varphi_j \in X\}$$

Será útil também a definição de uma função para cálculo da cardinalidade de uma letra, descrito como segue:

Seja w uma palavra e x uma letra

$$\gamma_x(w) : \{a, b, c\} \rightarrow \mathbb{N}$$

$$\gamma_x(w) = |X_w|$$

De forma direta, $\gamma_x(w)$ nos retorna o número de ocorrências da letra x na palavra w .

Retornando ao enunciado da questão, vamos destrinchar os comandos escritos e abstrair duas propriedades relevantes.

1. Determinar a avenida com quantidade de carros superior a metade do total de automóveis em todas avenidas.

1.1 Ou seja, **determinar a letra x cuja cardinalidade é superior ao tamanho total da string w de entrada:**

$$x \in \{a, b, c\} \mid \gamma_x(w) > \frac{|w|}{2}$$

dado que as únicas letras de w são $\{a, b, c\}$, segue que:

$$|w| = \gamma_a(w) + \gamma_b(w) + \gamma_c(w)$$

$$\text{Portanto, } \gamma_x(w) > \frac{|w|}{2} \Leftrightarrow \gamma_x(w) > \frac{\gamma_a(w) + \gamma_b(w) + \gamma_c(w)}{2}$$

1.2 Caso mais de uma avenida se qualifique, devemos respeitar a A sobre B, B sobre C e, por transitividade, A sobre C.

2. Determinar a avenida com a maior quantidade de carros.

2.1 Ou seja, **determinar a letra x de maior cardinalidade na palavra w :**

$$x, y, z \in \{a, b, c\} \wedge y \neq z \wedge z \neq x \wedge x \neq z$$

$$\gamma_x(w) > \gamma_y(w) \wedge \gamma_x(w) > \gamma_z(w)$$

2.2 Em caso de empate, devemos respeitar a ordem de prioridade definida em 1.2.

Também é importante considerar que a computação da [propriedade 2](#) só deve ser avaliada caso não haja parada por estado final na computação da [condição 1](#).

Vamos considerar também algumas equivalências para facilitar a modelagem e validação do automônto.

A primeira é centrada na [propriedade A](#), a qual nos dá que:

$$\gamma_x(w) > \frac{\gamma_a(w) + \gamma_b(w) + \gamma_c(w)}{2}$$

$$2 * \gamma_x(w) > \gamma_a(w) + \gamma_b(w) + \gamma_c(w)$$

Dado que $x \in \{a, b, c\}$,

$$\gamma_x(w) \in \{\gamma_a(w), \gamma_b(w), \gamma_c(w)\}$$

Generalizando,

$$\text{Sejam } x, y, z \in \{a, b, c\} \mid x \neq y \wedge y \neq z$$

$$2 * \gamma_x(w) > \gamma_x(w) + \gamma_y(w) + \gamma_z(w)$$

$$\gamma_x(w) > \gamma_y(w) + \gamma_z(w) \tag{\Delta}$$

Ou seja,

$$\gamma_x(w) > \frac{\gamma_x(w) + \gamma_y(w) + \gamma_z(w)}{2} \Leftrightarrow \gamma_x(w) > \gamma_y(w) + \gamma_z(w)$$

Portanto, basta que nosso automômato compute o [corolário \$\Delta\$](#) :

$$\gamma_x(w) > \gamma_y(w) + \gamma_z(w)$$

para que também seja verificada a [condição 1](#) do enunciado e portanto determinada a avenida prioritária.

2. Máquinas de Turing

Com objetivo de tratar da complexidade do problema de maneira didática, facilitando nossa argumentação, além de simplificar o processo de desenvolvimento da máquina abstrata, vamos segmentar porções de MT's (Máquinas de Turing) em ‘módulos’.

Definida MT como:

$$\text{MT} = (E, \Sigma, \Gamma, \delta, i, F)$$

E : Conjunto finito de estados

Σ : Alfabeto de entrada

Γ : Alfabeto da fita

δ : Funções de transição

i : Estado inicial

F : Conjunto de estados finais

O módulo \mathbb{M} de MT é:

$$\mathbb{M} = (E_\xi, \Sigma, \Gamma, \delta_\xi, e_{\text{in}}, E_{\text{out}})$$

Onde,

E_ξ : Subconjunto de estados em E .

δ_ξ : Subconjunto de transições em δ que atuem somente sobre os estados definidos em E_ξ .

e_{in} : Um estado eleito de E_ξ , deve denotar o ínicio do módulo. Vamos nos referir a ele como estado de entrada.

e_{out} : Um conjunto de estados finais pertencentes a E_ξ , representam o fim da computação do módulo. Chamaremos eles de estados de saída.

Importante notar que não necessariamente $e_{\text{out}} \subset F$. Ou seja, o fim da computação de \mathbb{M} não é necessário e suficiente para parada da MT.

No decorrer desse texto também usaremos o termo “submódulo”. Essa estrutura é análoga à um módulo e sua definição é construída em função de um módulo \mathbb{M} ao invés da MT por inteiro. Essa outra abstração também tem finalidades puramente organizacionais.

2.1. Máquina de uma única fita (MT_1)

Nossa máquina de uma fita só, ou MT_1 , é definida como:

$$MT_1 = (E, \Sigma, \Gamma, \delta, i, F)$$

Tal que,

$$\Sigma = \{a, b, c\}$$

$$\Gamma = \{x_\# \mid \forall x \in \Sigma\} \cup \{\varepsilon, <\} \cup \Sigma, \text{ ou seja, } \{a, b, c, a_\#, b_\#, c_\#, <, \varepsilon\}$$

$$F = \{F_a, F_b, F_c\}$$

$<$ é o símbolo delimitador do início da fita. Tratamos a fita como infinita a direita.

ε representa o vazio que preenche os espaços à direita da palavra inserida.

Cada estado final representa a decisão de conceder prioridade a uma avenida no cruzamento, onde:

Sendo $x \in \Sigma \mid F_x$: Decisão de priorizar a avenida x

As transições δ , estados E , F e i serão definidos em representações gráficas mais a frente.

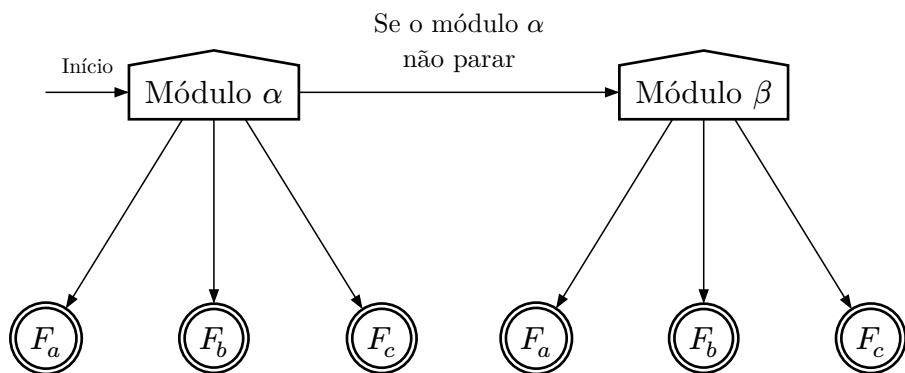
MT_1 tem dois módulos principais: α e β os quais serão estabelecidos e destrinchados a seguir.

2.1.1. Módulos e submódulos

O módulo α será responsável por verificar se a [condição 1](#) do enunciado vale para alguma letra de Σ através do [corolário \$\Delta\$](#) .

Caso nenhum estado final seja alcançado dentro de α , o automônato recorrerá ao módulo β , onde avaliará a regra de desempate definida na [condição 2](#) do problema. Verificando qual símbolo tem a maior cardinalidade na entrada.

A macroarquitetura da máquina assume o formato:



Fluxograma 1: Arquitetura do automônto

2.1.2. Submódulo ψ (ou “função de contagem”)

A peça central de MT1 é o súbmódulo ψ , o qual também chamaremos de função de contagem, responsável por comparar a cardinalidade de diferentes conjuntos de símbolos em uma dada palavra.

Definindo formalmente,

$$L \subset \Gamma$$

X_w, L_w são os conjuntos de ocorrências de L, x em w

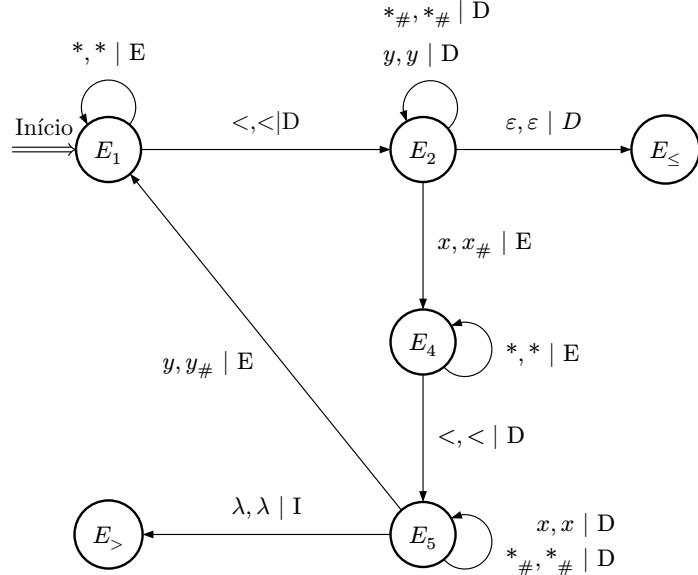
$\psi(x, L) :$ Avalia a natureza do produto cartesiano $X_w \times L_w$

Verificando a ocorrência de bijeção, injecão e/ou sobrejeção abstraímos sentidos de acordo ao definido pela Teoria de Conjuntos, em caso de:

1. bijeção, intuimos que $\gamma_x(w) = \gamma_y(w) + \gamma_z(w)$;
2. injecão, mas não sobrejeção, $\gamma_x(w) < \gamma_y(w) + \gamma_z(w)$;
3. sobrejeção, mas não injecão, $\gamma_x(w) > \gamma_y(w) + \gamma_z(w)$

Como as regras de negócio estipuladas ao sistema de acordo ao enunciado da questão preocupam-se apenas com o que já definimos em Δ , o súbmódulo ψ foca-se no caso 2).

A título de demonstração, tratemos $L = \{y\}$, $\psi(x, L)$ assume o formato:



Submódulo 1: Função ψ

Os dois estados de saída ou parada desse submódulo serão $E_<$ e $E_≥$, os quais semânticamente representam:

- $E_<: X_w < L_w$
- $E_≥: X_w \geq L_w$

Naturalmente, em caso de $|L| > 1$, as transições que contenham símbolos de L na representação acima, seriam múltiplas mas ainda de formato análogo, apenas tendo uma correspondente para cada símbolo de L .

Descrevendo de forma direta, a função de contagem que definimos tem uso de um *loop* (estrutura de repetição) e vai efetivamente **mapeando** cada ocorrência de x com ocorrências dos símbolos em L , nunca repetindo items envolvidos entre pares.

Para isso, é feita uma marcação no formato:

Sendo $x \in \Gamma$ | Marcamos um x qualquer o substituindo pelo respectivo $x_{\#}$

Essa marcação sinaliza que esse símbolo deve ser ignorado nas iterações seguintes do loop.

Vamos então destrinchar o funcionamento de ψ transição a transição para verificarmos formalmente a computação das seguintes propriedades:

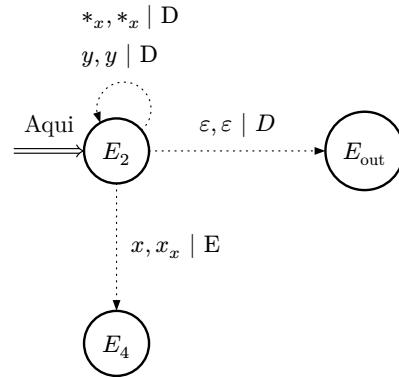
Parar em $E_>$ $\Leftrightarrow \gamma_x(w) > \gamma_y(w) + \gamma_z(w)$

Sair por E_{\leq} $\Leftrightarrow \gamma_x(w) \leq \gamma_y(w) + \gamma_z(w)$

Passo 1 - Rebobina ao início da fita.

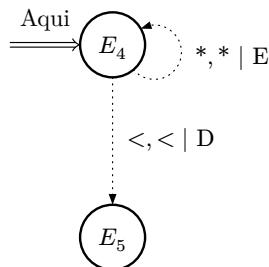


Passo 2 - Avança pela fita passando por todos y e x_x – que são os x 's já pareados – até encontrar um x ou o fim da fita.

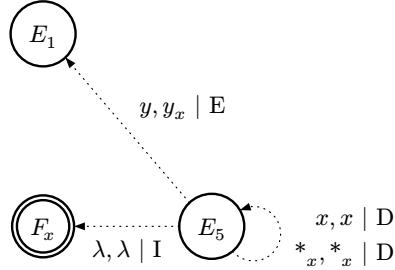


- a) Caso ache um x ou x_y , o sobreescriveremos por x_x e seguimos para o passo 3.
- b) Caso não ache um x , escapamos para E_{\leq} .

Passo 3 - Rebobina pro início da fita;



Passo 4 - Avança pela fita buscando um y tal que $y \notin \{x, x_x\}$;



- a) Caso não encontre, pára em $E_>$.
- b) Caso encontre, sobreescrivemos o y na fita com y_x e retornamos ao passo 1.

Um aspecto relevante é que trata-se um *loop* com apenas duas possíveis interrupções: as etapas 2.b) e 4.a) do passo-a-passo.

Agora faremos uma breve verificação formal de que o algoritmo abstraído na máquina constrói as propriedades almejadas.

Dado que,

$$n, i \in N$$

$$x, k_n \in \{a, b, c\}$$

$$w = k_0 * \dots * k_n$$

Os passos 2.a) e 4.b) marcam os elementos substituindo-os por $x_\#$ e $y_\#$, respectivamente construindo:

$$X_w = \{j \mid \forall j \in N \text{ tal que } 0 \leq j \leq n \text{ e } k_j = x\} \text{ (Ocorrências de X)}$$

$$\bar{X}_w = \{j \mid \forall j \in N \text{ tal que } 0 \leq j \leq n \text{ e } k_j \neq x\} \text{ (Ocorrências de não-X)}$$

Como fazemos uma substituição 2.a), rebobinamos, fazemos uma substituição em 4.b) e reiniciamos o loop. Estamos sempre compondo X e \bar{X} , um por um, mantendo igualdade quanto a cardinalidade dos conjuntos. Incrementando primeiro X e então \bar{X} .

Dessa forma, independente da iteração do loop, no instante precedendo a execução do passo 2.a), temos que $|X_w| = |\bar{X}_w| + 1$. Até que executarmos 4.b), retornando a igualdade $|X_w| = |\bar{X}_w|$.

Ou seja, somente nos pontos de interrupção do loop ocorre disruptura permanente dessas igualdades.

Em 2.b), não encontramos mais x e escapamos para E_{\leq} , como interrompemos o processamento do módulo, não buscamos mais y , mas temos certeza da cardinalidade de x , naquele instante alcançamos que $|X| = |\bar{X}|$. Entretanto, como não buscamos mais y , podemos afirmar de maneira forte que $\gamma(x)_w \leq \gamma(y)_w + \gamma(z)_w$ visto que podem ainda haver mais ocorrências de y .

Já na parada em 4.a), temos do passo 2.a) que $|X_w| = |\bar{X}_w| + 1$ e interrompemos o processamento em $E_>$. Neste caso, $\gamma(x)_w \geq \gamma(y)_w + 1$, o que equivale a $\gamma(x)_w > \gamma(y)_w$.

2.1.3. Módulo α

O módulo α irá analisar se o [corolário \$\Delta\$](#) vale para alguma letra em $\{a, b, c\}$. Caso verifique validade de tal propriedade para a letra-alvo, é eleito o estado final apropriado representativo da decisão apropriada e a computação cessa.

Visto que o automômato encerra processamento assim que é validado Δ para um elemento, extraímos uma capacidade de “curto-circuito”, a qual em conjunto a um processamento ordenado dos elementos de Σ permite zelar pela regra de desempate definida em [1.2](#)

Portanto, efetuamos essa avaliação para as letra na ordem “a”, “b” e, então, “c”, implicitamente zelando pela ordem de desempate.

Caso nenhuma letra do alfabeto cumpra com a propriedade alvo de α , a computação é encaminhada para o módulo β .

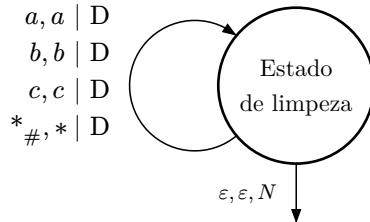
Vamos então quebrar o processamento em 3 instâncias de submódulos ψ , tratando os estados $E_>$ de cada como a decisão de concessão de prioridade para a letra.

Esse submódulos serão intercalados por um estado de “limpeza” com transições que irão remover os símbolos marcados retornando o conteúdo da fita a entrada inicial. Dessa forma símbolos escritos por ψ não afetarão os submódulos seguintes.

Vale notar que existe uma alternativa que reduziria o custo computacional da MT, por meio da remoção desses estados de limpeza, incluindo ao alfabeto da fita símbolos de marcação específicos pra cada ψ , e então incorporando-os nas transições dos submódulos de contagem seguintes.

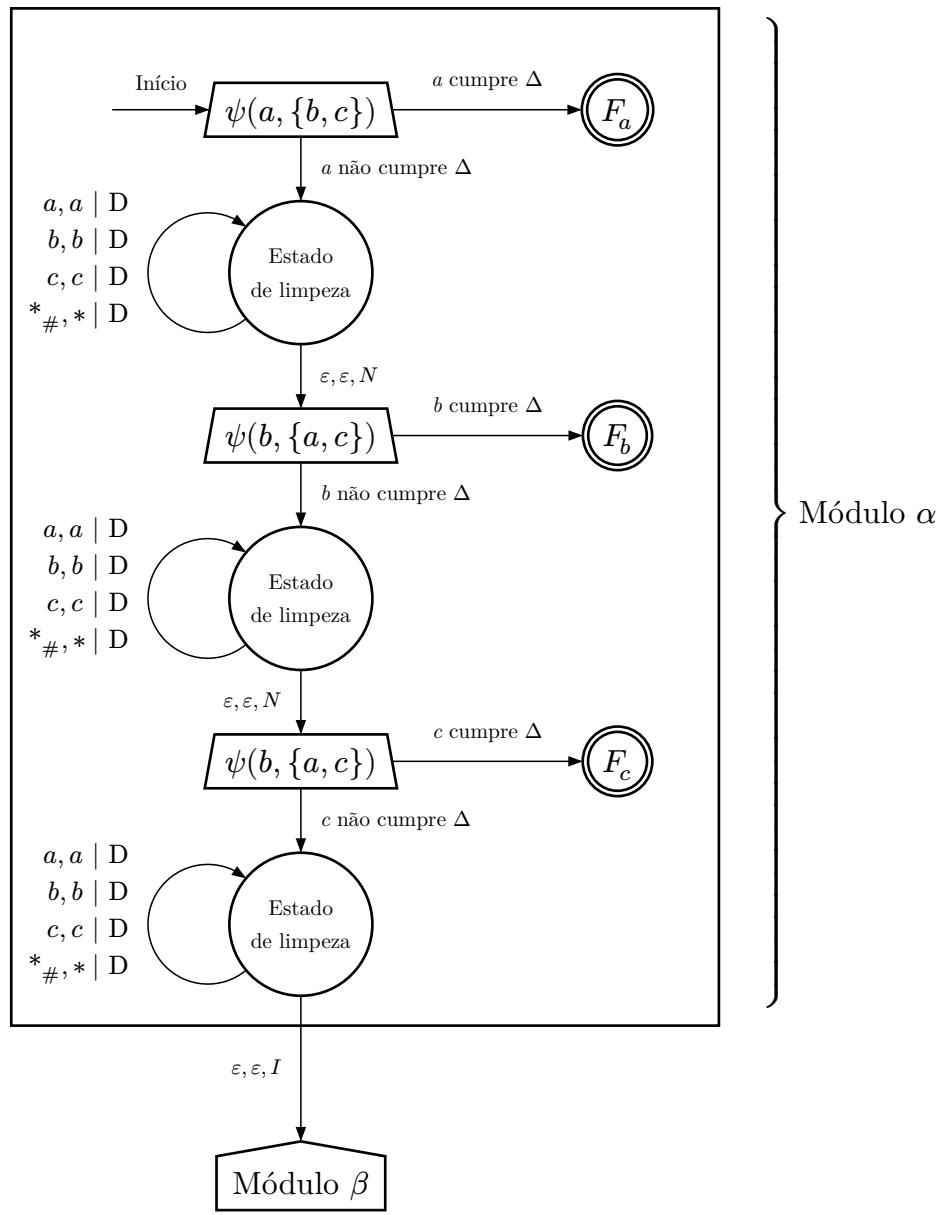
Apesar de mais eficiente, isso acrescentaria um número considerável de transições, então optamos por manter essa estrutura.

Cada estado de limpeza, segue o formato



Máquina de Turing 1: Estado de limpeza

Dando ao módulo α o formato:



Fluxograma 2: Raio-x do módulo α

2.1.4. Do custo da contagem no módulo α

$\psi(x, \{y, z\})$ deve verificar se para um certo x vale:

$$\gamma_x(w) > \gamma_y(w) + \gamma_z(w) \text{ onde } x, y, z \in \{a, b, c\} \text{ e } x \neq y \neq z$$

Para tal, vamos comparar a **cardinalidade** de x com a de y e z . Com esse objetivo, utilizamos nossa função de comparação de cardinalidade $\psi(x, L)$, sendo $L = \{y, z\}$.

Dentro do módulo, substituimos o $E_>$ por F_x simbolizando a parada total da MT1 na decisão de priorizar a avenida x . Uma vez que esse estado implica na condição verdade de Δ para o x alvo.

Enquanto E_{\leq} será:

- E_{in} do submódulo ψ seguinte, caso $x \neq c$;
- o estado de entrada do módulo β , caso $x = c$;

Dado a definição de x e L , vamos agora para o cálculo da complexidade temporal associado a α .

Por convenção, nos preocupamos nessa análise com o pior caso possível, com isso em mente temos de determinar qual seria a palavra de maior custo associado para que α compute.

Com esse fim, vamos destrinchar cada passo de ψ , determinando o pior formato possível da palavra para cada um deles. Depois, comparar essas composições de w , calcular o custo associado e eleger o pior candidato entre eles.

Trateremos por i , a iteração atual do passo-a-passo, visto que ψ opera como *loop*, iniciando nossa contagem em 1.

Passo 1 - Rebobinação da fita

— Pior palavra: $w = x^k * y^j$

A partir de $i > 0$, essa etapa rebobina a partir da posição do cabeçote determinada no passo 4.b). O pior caso então é o y^j como sufixo da palavra, maximizando a quantidade de transições para rebobinar.

Passo 2 - Busca por um x não marcado

— Pior palavra: $w = y^k * x^j$

Como aqui buscamos um x não pareado, iniciando nossa busca do início de w seguindo também seguindo linearmente. O pior caso é de todos x como sufixo de w .

Passo 3 - Rebobinação da fita

— Pior palavra: $w = y^k * x^j$

Análogo ao primeiro passo, só que dessa vez a posição inicial do cabeçote é determinada pelo passo 2), assim invertemos a ordem.

Passo 4 - Buscamos por um y não marcado

— Pior palavra: $w = x^k * y^j$

Análogo ao passo 2), só que dessa vez queremos que o sufixo de w seja y^j .

Temos então dois formatos possíveis de w disputando a posição de pior possível. Vamos considerar cada um e calcular o custo associado.

Considere a seguir que,

$$w \text{ é palavra e, } n = |w| = m + k$$

2.1.4.1. Análise para $w = y^k * x^m$

- **Passo 1:** Da posição i até o início $\rightarrow i$
 - **Passo 2:** Passar por k símbolos y + i símbolos $x_{\#}$ $\rightarrow k + i$
 - **Passo 3:** Da posição $k + i$ até o início $\rightarrow k + i$
 - **Passo 4:** Passar pelos primeiros i símbolos até o próximo y livre $\rightarrow i$
- **Custo por iteração (C_i)**

$$C_i = i + (k + i) + (k + i) + i = 2k + 4i$$

2.1.4.2. Análise para $w = x^m y^k$

- **Passo 1:** Da posição $m + i$ até o início $\rightarrow m + i$
- **Passo 2:** Passar por i símbolos $x_{\#}$ até o próximo x $\rightarrow i$
- **Passo 3:** Da posição i até o início $\rightarrow i$
- **Passo 4:** Passar por m símbolos x (todos já marcados) + i símbolos $y_{\#}$ $\rightarrow m + i$

Custo da iteração i :

$$C_i = (m + i) + i + i + (m + i) = 2m + 4i$$

Para $y^k x^m$ com ℓ iterações:

$$T_1 = \sum_{i=1}^{\ell-1} (2k + 4i) = 2k\ell + 2\ell(\ell - 1)$$

Para $x^m y^k$ com ℓ iterações:

$$T_2 = \sum_{i=1}^{\ell-1} (2m + 4i) = 2m\ell + 2\ell(\ell - 1)$$

Resta determinar para cada caso a pior distribuição de m e k .

Em T_1 , ℓ é o número de iterações do *loop*, sendo esse determinado de acordo aos dois cenários de parada.

- Caso 1: $k \leq m \rightarrow$ Pára em $F_x \Leftrightarrow l = k$
- Caso 2: $k > m \rightarrow$ Escapa para $E_{\text{out}} \Leftrightarrow l = m$

Portanto, $l = \min(k, m)$.

Substituindo em T_1 ,

$$T_1(k) = 2k \cdot \min(k, m) + 2 \min(k, m) \cdot (\min(k, m) - 1)$$

$$T_1(k) = 2k \cdot \min(k, m) \cdot [k + \min(k, m) - 1]$$

- Caso 1: $\min(k, m) = k$

$$T_1(k) = 2k \cdot k + 2k \cdot (k - 1) = 4k^2 - 2k$$

Com a restrição de $k + m = n$ e $k \leq m$:

$$k \leq m \Rightarrow k \leq n - k \Rightarrow 2k \leq n \Rightarrow k < \frac{n}{2}$$

Importante notar que estamos considerando $|w| > 0$ aqui (ou seja, uma palavra não vazia), logo $n \geq 2$.

Com isso temos o domínio $k \in [1, \frac{n}{2}]$.

Vamos otimizar,

$$\frac{dT_1}{dk} = 8k - 2$$

Em nosso domínio obtemos que, seja $k \geq 1$,

$$8k - 2 \geq 8(1) - 2 = 6 > 0$$

Como nossa função é estritamente crescente no intervalo relevante, basta pegarmos o extremo da direita para termos o máximo.

Ou seja, no caso 1), a pior distribuição possível é $k = \frac{n}{2}$, e como $n = k + m$, assim obtemos $k = m = \frac{n}{2}$.

O que nos dá um custo máximo de: $T_1^A = 4\left(\frac{n}{2}\right)^2 - 2\left(\frac{n}{2}\right) = n^2 - n$

- Caso 2: $\min(k, m) = m = n - k$

$$T_1(k) = 2k \cdot (n - k) + 2(n - k) \cdot ((n - k) - 1) = 2(n - k)(k - 1)$$

$$T_1(k) = 2(n - 1)(n - k) = 2n^2 - 2n - 2nk + 2k$$

Aqui o domínio é $m \in (\frac{n}{2}, n)$, visto que $k > m = n - k$.

Derivando,

$$\frac{dT_1}{dk} = -2n + 2 = 2(1 - n)$$

Para $n > 1$, $2(1 - n) < 0$, ou seja, a função é decrescente no domínio inteiro. Logo basta pegarmos o valor a extrema esquerda para $k = \frac{n}{2}$.

O custo desse caso será $T_1^B = 2\left(\frac{n}{2}\right)(n - 1) = n(n - 1) = n^2 - n$. O mesmo do caso A).

Ou seja, o custo computacional é o mesmo independente de qual caso de interrupção, e a pior composição possível para a palavra dado o formato $w = y^k + x^m$ é $k = m = \frac{|w|}{2}$.

Outro fato importante que extraímos da análise da derivada, é de que a função é crescente a medida que os valores de k e m convergem.

Por fim, determinamos o custo de um ψ em α igual à $n^2 - n$.

Em T_2 , por simetria, a análise é idêntica e o mesmo custo obtido.

Determinamos a pior palavra para qualquer ψ em α sendo:

$$x, y, z \in \{a, b, c\}$$

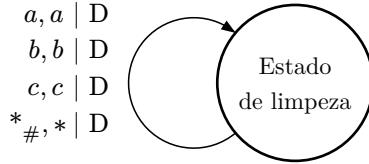
$$w = x^i \cdot y^i \cdot z^i$$

com qualquer permutação de x, y e z .

Obtendo custo total de $n^2 - n$, ou seja, contido no $\Theta(n^2)$.

2.1.5. O custo da limpeza

Vamos fazer uma breve demonstração da complexidade da operação de limpeza, previamente definidas como:



Máquina de Turing 2: Estado de limpeza

O custo aqui é o da distância do ponto de entrada no *estado de limpeza* ao fim da palavra de entrada, visto que iteramos por toda a palavra removendo os símbolos marcados.

Como entramos nesse estado somente no caso de escape de ψ por E_{\leq} que sempre precede o posicionamento do cabeçote no início da fita. Ou seja, sempre será necessário traversar por toda a palavra nesse estado. Dessa forma, o custo aqui é de n , sendo $n = |w|$, o qual obviamente pertence a $\Theta(n)$.

2.1.6. Conclusão do custo de α

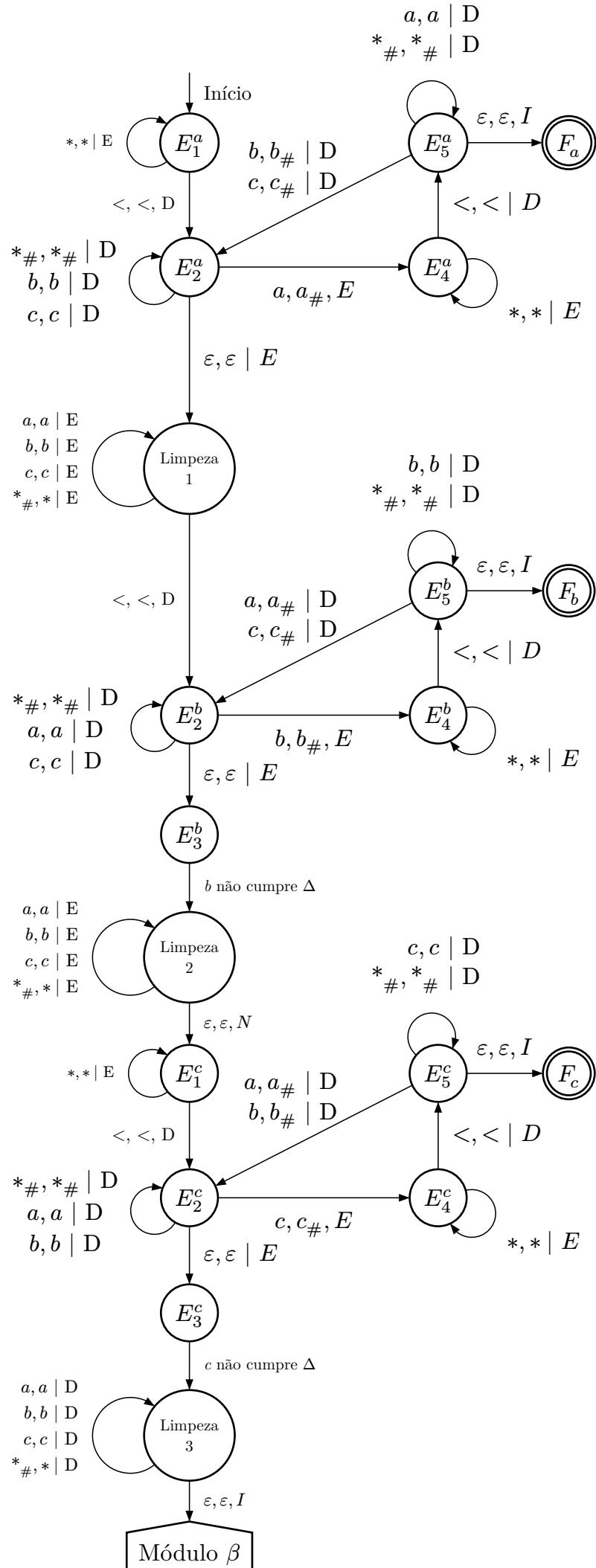
Podemos finalmente estipular o custo acumulado de α . Dado que consideramos o pior caso como o de empate contínuo, teremos:

1. $\psi(a, \{b, c\})$: $n^2 - n$
2. Operação de limpeza: n
3. $\psi(b, \{a, c\})$: $n^2 - n$
4. Operação de limpeza: n
5. $\psi(c, \{a, b\})$: $n^2 - n$
6. Operação de limpeza: n

Totalizando,

$$\text{Custo total de } \alpha = 3((n^2 - n) + n) = 3n^2$$

2.1.7. Módulo α na íntegra



Máquina de Turing 3: Diagrama completo do módulo α com submódulos ψ expandidos

2.2. Máquina na íntegra

ipsi literis ipsi literis ipsi literis ipsi literis ipsi literis ipsi literis

2.3. Análise de complexidade

ipsi literis ipsi literis ipsi literis ipsi literis ipsi literis ipsi literis

2.4. Exemplos

ipsi literis ipsi literis ipsi literis ipsi literis ipsi literis ipsi literis

3. Máquina de múltiplas fitas

3.1. Módulos

ipsi literis ipsi literis ipsi literis ipsi literis ipsi literis ipsi literis

3.2. Máquina na íntegra

ipsi literis ipsi literis ipsi literis ipsi literis ipsi literis ipsi literis

3.3. Análise de complexidade

ipsi literis ipsi literis ipsi literis ipsi literis ipsi literis ipsi literis

3.4. Exemplos

ipsi literis ipsi literis ipsi literis ipsi literis ipsi literis ipsi literis