Controle de Tráfego Urbano Inteligente

Trabalho Final do Curso de Teoria da Computação

João Roberto - 222217111
Caio Mello - XXXX

Vinicius - YYYYYYYYY
Ciclano - ZZZZZZZZ

Período Letivo
2025.2

Índice

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.

lero lero lero lero lero lero lero lero lero lero lero lero lero lero lero lero lero lero lero lero lero lero lero lero lero lero lero lero lero lero lero lero lero lero lero lero lero lero lero lero lero lero lero lero lero lero lero lero

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:

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

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

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

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

Será útil também a definição de uma função para cálculo da cardinalidade de uma letra, construída segundo o descrito abaixo:

$$\text{Seja } w \text{ uma palavra e } x \text{ uma letra}$$

$$\gamma_x(w) : \{a, b, c\}^* \to \mathbb{N}$$

$$\gamma_x(w) = |\{i \in \{1, ..., |w|\} \wedge i = x\}|$$

Onde γ é a função de cardinalidade.

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

A partir do enunciado da questão, abstraímos duas propriedades que nosso automômato deve computar, as chamaremos de A e B.

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**:

$$x \in \{a, b, c\} \mid \gamma_x(w) > \frac{\gamma_a(w) + \gamma_b(w) + \gamma_c(w)}{2}$$

   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 ocorrer caso não haja parada por estado final na computação da condição 1.

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

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

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

Generalizando,

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

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

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

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 Δ para que também valha a condição 1 do enunciado.

2. Máquinas de Turing

Para fins de tratar a complexidade do problema de maneira didática, facilitando a argumentação e desenvolvimento, vamos abstrair porções de MT's (Máquinas de Turing) em módulos.

Pela definição de MT vinda do Newton (2006) temos

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

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

A partir dela, definimos o módulo $\mathbb{M}$ de MT como:

$$\mathbb{M} = (E_{\text{ш}}, \Sigma, \Gamma, \delta_{\text{ш}}, e_{\text{entrada}}, E_{\text{saída}})$$

Onde,
$E_{\text{ш}}$: Subconjunto dos estados em E.
$\delta_{\text{ш}}$: Subconjunto de transições de δ que atuem sobre os estados definidos em $E_{\text{ш}}$.
e_{entrada}: Um estado inicial em $E_{\text{ш}}$, denota o ínicio do módulo. Vamos nos referir a ele como estado de entrada.
$e_{\text{saída}}$: Um conjunto de estados finais pertencentes a $E_{\text{ш}}$, representam o fim da computação do módulo. Chamaremos eles de estados de saída.

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

No decorrer desse texto usaremos a noção de um "submódulo". Tal estrutura será útil a nível didático e deve ter seu conceito compreendido como análogo ao de um módulo, só que ao invês de ser definido em função de toda a MT, ele tem seus elementos delimitados por um $\mathbb{M}$.

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,
$\Sigma = \{a, b, c\}$
$\Gamma = \{a, b, c, a_a, a_b, a_c, b_a, b_b, b_c, c_a, c_b, c_c, <, \varepsilon\}$
$F = \{F_a, F_b, F_c\}$

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

$$x \in \Sigma \mid F_x : \text{Sistema prioriza avenida x}$$

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.

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 Δ.

Caso nenhum estado final seja alcançado dentro de α, o automômato recorrerá ao módulo B, onde avaliará a condição 2 do problema. Verificando qual letra tem a maior cardinalidade na entrada.

A macroarquitetura da máquina tem o formato:

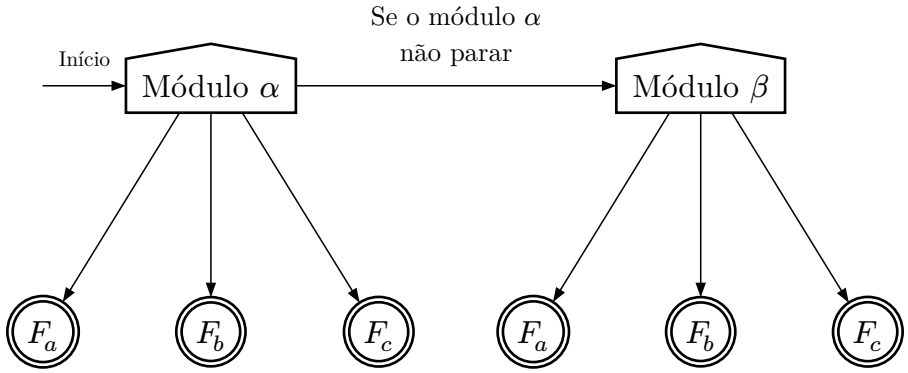


Diagrama 1: Arquitetura do automômato

2.1.2. Módulo α

O módulo α irá analisar se o corolário Δ 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 de uma decisão quanto a avenida prioritária para a sinaleira e a computação cessa.

Visto que o automômato encerra processamento assim que é validado Δ para um elemento alvo, extraímos uma capacidade de "curto-circuito", a qual em conjunto a um processamento sequencial ordenado dos elementos de Σ permite zelar pela regra de desempate definida por 1.2

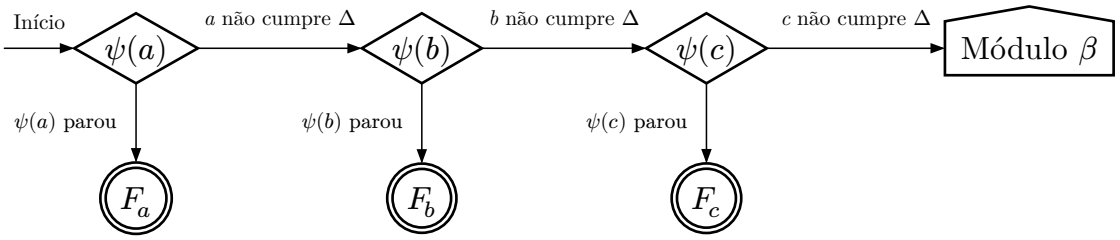
Portanto, efetuamos essa avaliação para as letra na ordem "a", "b" e, então, "c".

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

Dessa forma, abstraímos um submódulo ψ tal que:

$\psi(x)$: Computa se x cumpre a propriedade Δ, caso positivo, o automômato pára em F_x

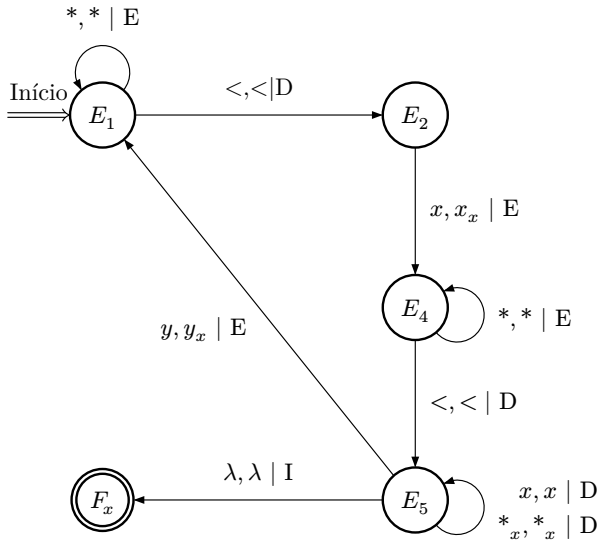
Então avaliaremos, em ordem, $\psi(a)$, $\psi(b)$, $\psi(c)$, de acordo a imagem:



Fluxograma 1: Formato do Módulo A

2.1.3. Submódulo ψ

Definimos de forma genérica o submódulo ψ, dado que $x, y \in \Sigma$ e $x \neq y$, $\psi(x)$ será:



Submódulo 1: Função ψ

Vamos quebrar as transições em algoritmo a partir do qual verificaremos a computação das propriedades desejadas, tal que os passos são:

Vamos quebrar o fluxo das transições num passo-a-passo para facilitar a verificação da computação das propriedades desejadas. As etapas seguem:

Passo 1 - Lê cada letra de w até encontrar uma ocorrência de x;

a) Caso não encontre x:
- x não cumpre a propriedade C, escapamos para E_{out} e cessamos computação.

b) Encontrou um x:
- Sobreescreveremos o x na fita com x_x e seguimos para o passo 2.

Passo 2 - Retornamos ao início da fita;

Passo 3 - Buscamos uma letra y tal que $y \notin \{x, x_x\}$;

a) Caso não encontre:
- x cumpre com a a propriedade C e para no estado final F_x e cessamos computação.

b) Caso encontre:
- Sobreescreveremos o y na fita com y_x e seguimos para o passo 4.

Passo 4 - Retornamos ao início da fita e voltamos ao passo 1.

Vamos para a prova de que o algoritmo descrito verifica C.

De antemão é relevante notar que o *script* é um *loop* que se repete até parar em 1-a) ou 3-a).

Tomamos por certeza a eventual parada, em razão da palavra de entrada ser finita e que para que cesse o processamento é necessário e suficiente que sejam esgotados elementos distintos ou iguais a x.

Uma vez que sempre retornamos ao início da fita após encontrar uma elemento satisfaça uma dessas condições, e que marcamos elementos após encontra-los, temos por óbvio a garantia de que haverá parada.

Vamos agora para demonstração de como esse processo também computa a validade de C para x.

Primeiro estabeleçamos que,

$$n \in \mathbb{N} \mid x, \gamma_n \in \{a, b, c\} \mid w \text{ é uma palavra.} \mid w = \gamma_0 * ... * \gamma_n$$

Por meio do *loop* vamos gradativamente consumindo as letras de w e produzindo dois conjuntos disjuntos.

Os passos 1, 1.2, e 2 e 4 constroem X:

$$X = \{\forall \gamma_i \in w \mid \gamma_i = x\}$$

Enquanto 3, 3.2 e 4 definem $\overline{X}$:

$$\overline{X} = \{\forall \gamma_i \in w \mid \gamma_i \neq x\}$$

Basta que verifiquemos:

$$|X| > |\overline{X}|$$

2.2. Máquina na íntegra

ipsi literis 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 ipsi literis

2.4. Exemplos

ipsi literis 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 ipsi literis

3.2. Máquina na íntegra

ipsi literis 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 ipsi literis

3.4. Exemplos

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