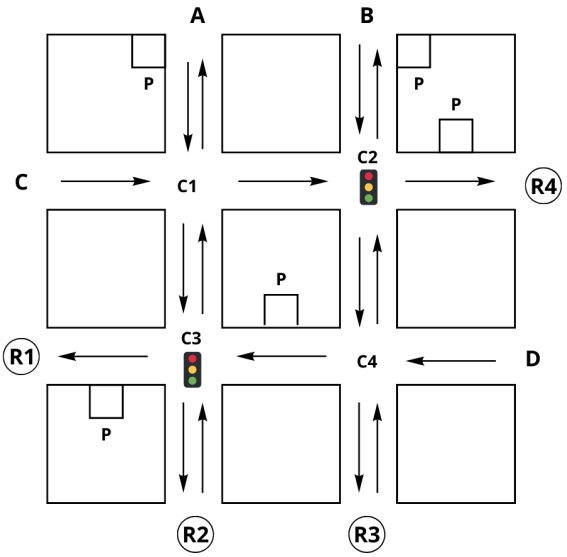
# LFA - Modelagem / Simulação de uma região com trânsito de veículos

Herton da Silveira e Silva
2 de Dezembro de 2024

## 1 Introdução

O trabalho consiste em modelar um sistema de trânsito simplificado, representado por um cenário inicial (cruzamentos, ruas e estacionamentos), utilizando Autômatos Finitos Determinísticos (AFD) e Autômatos com Pilha (AP).

A meta é simular e analisar os comportamentos dos veículos no sistema. O foco está na modelagem de fluxos de veículos e no funcionamento de semáforos em diferentes cenários.




Cenário original baseado no modelo dado pelo professor

## 2 Representação do Movimento dos Veículos

Para a representação do movimento dos veículos no cenário apresentado, foi modelado um AFD geral que ilustra os estados, bem como todas as possibilidades de transição. Partindo desse AFD geral é possível obter então os quatro autômatos que modelam o sistema, em que cada um representa um ponto de partida ou estado inicial diferente no mapa.

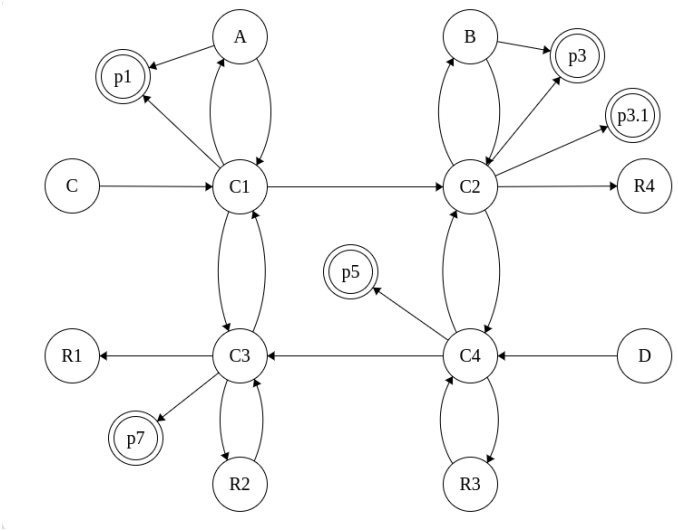


Imagem geral apenas com os estados e transições ilustrativas.

## 2.1 Representação e nomenclatura dos estados

Para modelar tal autômato, considerou-se os possíveis pontos de partida como A, B, C e D, e é a partir da consideração ou não como ponto de partida em cada autômato que servirá como base para distinção deles.
Ademais, os cruzamentos foram representados como C1, C2, C3 e C4, mais para frente veremos que cada um deles pode ter ou não um semáforo.

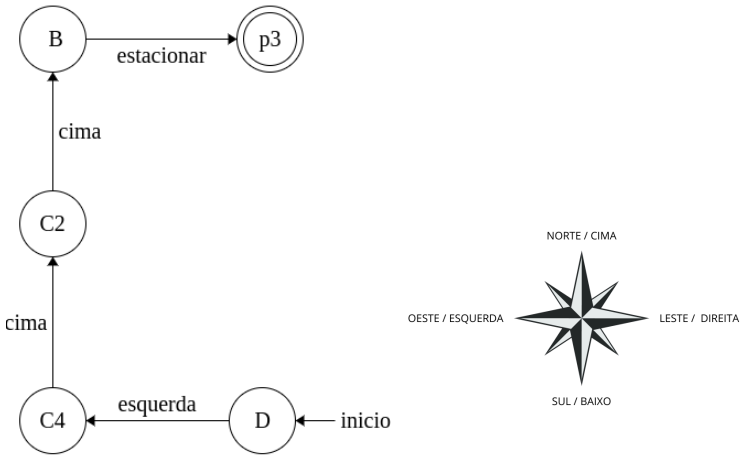
Baseando-se na imagem do cenário original, vemos que terão nove quarteirões e cada um desses pode ter ou não um ou mais estacionamentos que serão os estados de aceitação p1, p3, p3.1, p5 e p7, sendo que suas nomenclaturas são baseadas na posição de cada quarteirão visto de cima.

E os pontos de retorno serão R1, R2, R3 e R4, é neles que será possível mudanças de direção, como 90°, 180°, 270° ou inversão completa.

## 2.2 Lógica de transição e alfabeto

As transições nesse modelo nada mais são que os movimentos dos veículos de um estados para outro, onde os estados representam pontos específicos no mapa, ou seja, no cenário original ao iniciar no estado A é possível realizar duas transições, uma para p1 e já estacionar e outra para C1, para assim prosseguir com sua trajetória.

Foi pensando na visualização do autômato como um mapa visto de cima em que a trajetória ( a fita que representará um percurso específico) terá como alfabeto: cima, baixo, esquerda e diereita que simplificará a visualização do trajeto, assim como em apps de navegação por GPS. Portanto, a fita a ser consumida terá instruções bem claras que representarão qualquer percurso dentro do escopo de aceitação do modelo.

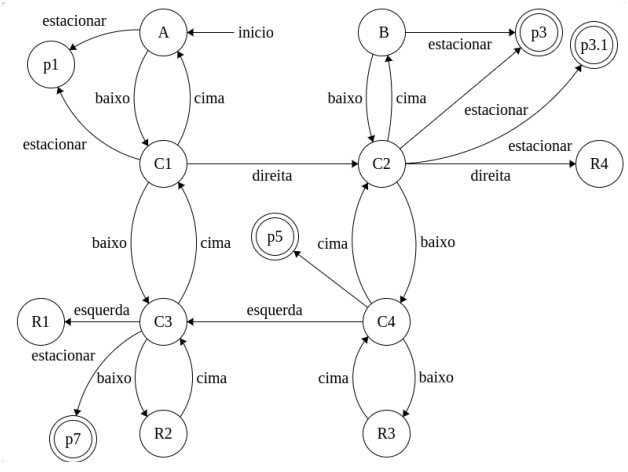



Exemplo de trajeto e rosa dos ventos

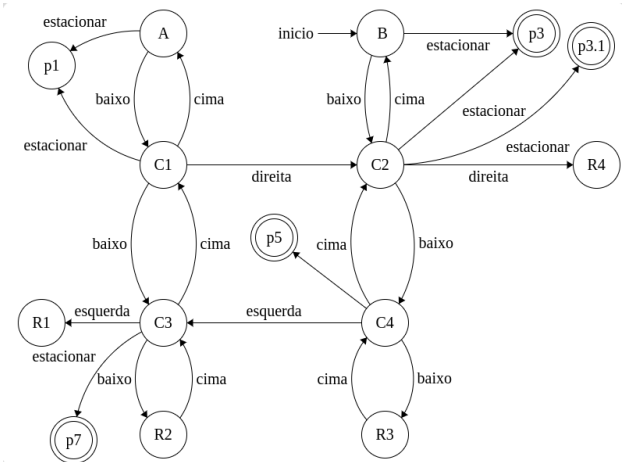
## 2.3 Autômatos obtidos

A partir de cada estado inicial, foi possível obter um autômato diferente por meio da exclusão de estados inacessíveis.

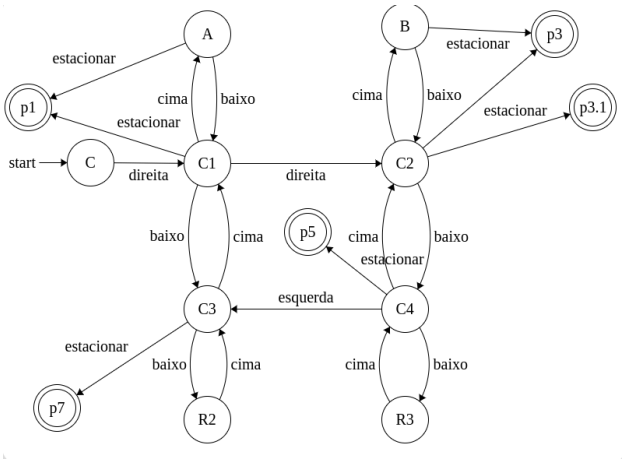
Caso o ponto de partida seja em A, temos:



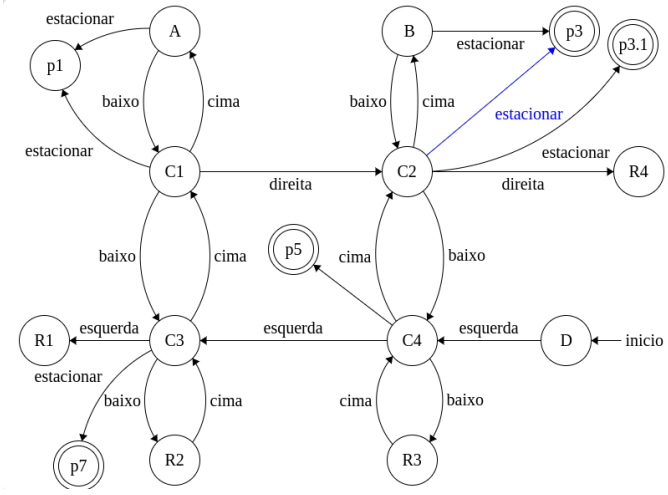
Caso o ponto de partida seja em B,  temos:




Caso o ponto de partida seja em C, temos:



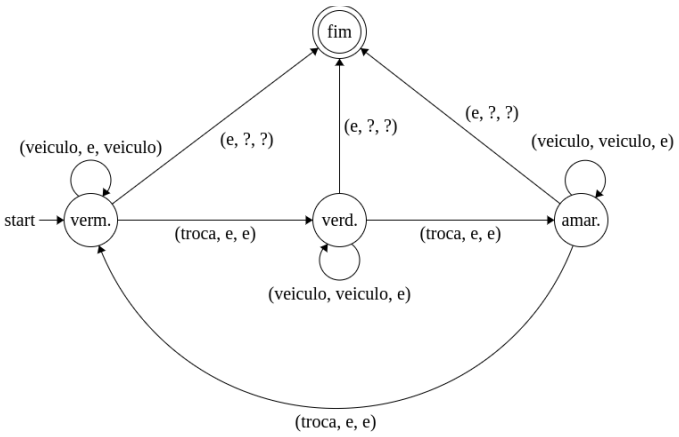

E por último, caso o ponto de partida seja em D:



# 3 Modelagem dos semáforos

Primeiramente, é importante ressaltar que cada um dos cruzamentos C1, C2, C3 e C4 pode possuir ou não um semáforo que servirá para controle do fluxo que passará por esse estado.

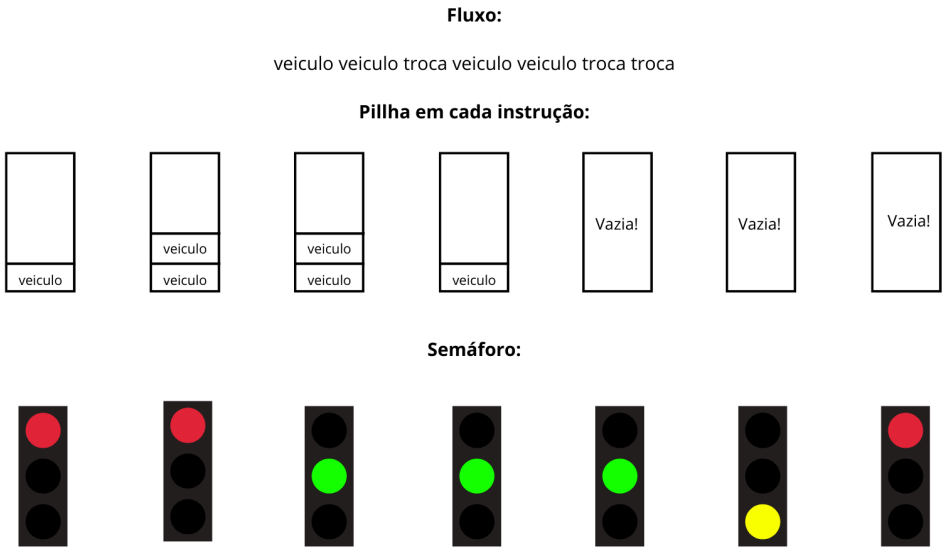
Para modelar tal cenário, foi criado um AP (autômato com pilha) que terá os três estados de um semáforo padrão: vermelho(que será o estado inicial), verde e amarelo, assim a passagem para cada estado irá depender do fluxo estabelecido no dado momento. Logo, foi criado o seguinte AP:




Tal AP, além dos estados já ressaltados, terá alfabeto: veículo e troca e alfabeto da pilha: veículo, que seguirá a seguinte lógica de transição:

- Estando no estado vermelho e recebendo *veículo*, empilha *veículo* e continua no mesmo estado.
- Estando no estado vermelho e recebendo *troca*, troca para o estado verde.
- Estando no estado verde e recebendo *veículo,* desempilha *veículo* e continua no mesmo estado.
- Estando no estado verde e recebendo *troca,* troca para o estado amarelo.
- Estando no estado amarelo e recebendo *veículo,* desempilha *veículo* e continua no mesmo estado.
- E por último, estando no estado amarelo e recebendo *troca,* troca para o estado vermelho.

A seguir, como exemplo pode-se ver a situação da pilha e do semáforo em cada momento do seguinte fluxo aceito pelo autômato: veiculo, veiculo, troca, veiculo, veiculo, troca e troca.




Dessa forma, é possível controlar o fluxo de diversos veículos por meio de um único semáforo e configurar na implementação quantos veículos serão necessários para que o sinal mude de estado (o que veremos mais à frente na hora de discutir sobre a implementação).

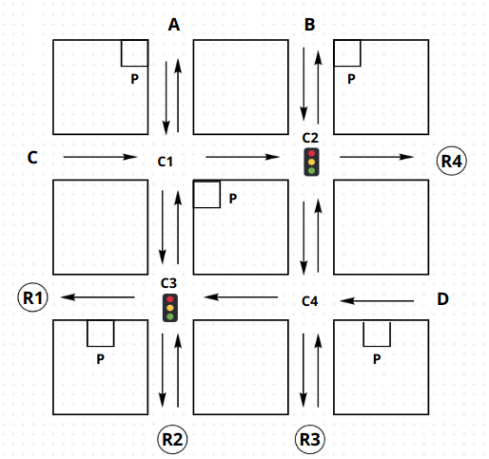
# 4 Análise e avaliação dos estacionamentos

Neste tópico, será estudado a eficiência no trajeto de cada veículo dado as posições dos estados finais ou estacionamentos, ou seja, partindo do mesmo estado inicial, é preferível se ter uma trajetória com menos transições, sem alterar o quarteirão que será o destino de nosso veículo e a quantidade de estacionamentos do cenário original.

Avaliando o cenário original, temos cinco estacionamentos para nove quarteirões, sendo que um quarteirão (o quarteirão três) possui dois estacionamentos. Visando evitar paradas desnecessárias em semáforos tomando diferentes pontos de início, pode-se fazer os seguintes ajustes:

- Alterar um dos estacionamentos do quarteirão três para o nono, assim os pontos finais ficam organizados de maneira mais uniforme no cenário. Além disso, agora é possível que o veículo partindo de qualquer ponto inicial consiga estacionar sem parar em semáforos.
- Modificar o estacionamento do quarteirão central (de número 5) para mais próximo dos pontos iniciais A e C, evitando assim que estes parem em semáforos.

Dessa maneira, o seguinte cenário é criado, pensado para ser mais prático para veículos estacionarem partindo de diferentes estados iniciais:




# 5 Implementação e Validação

Na etapa final deste trabalho, será realizado a implementação utilizando a linguagem de programação Python dos AFDs e APs, para que seja possível a validação das trajetórias e fluxos, respectivamente. Para rodar o programa foi usado a linha de comando:

python3.10 ./main.py

## 5.1 Implementação do AFD e AP

O código geral do AFD e do AP usado no modelo será criado como uma classe que é [1] e em Python, tal classe terá funções e variáveis próprias a cada instanciamento, o que facilitará a criação de vários AFDs e APs. Por exemplo,  essa a função que processa uma cadeia de instrução de trajetos de dado AFD:

```
def processar(self, cadeia):

    estado_atual = self.estado_inicial

    for simbolo in cadeia:
        if simbolo not in self.alfabeto:
            print(f"Erro: símbolo '{simbolo}' não pertence ao alfabeto.")
            return False

        if simbolo in self.transicoes[estado_atual]:
            estado_atual = self.transicoes[estado_atual][simbolo]
        else:
            print(f"Erro: não há transição definida para o estado '{estado_atual}' com a instrução '{simbolo}'.")
            return False

    if estado_atual in self.estados_finais:
```

```
            return True
        else:
            return False
```

O código que processa o fluxo em um dado semáforo, ou seja, a função que aceita ou não uma cadeia dada ao AP é:

```
def processar(self, cadeia):
        estado_atual = self.estado_inicial
        proximo = ''
        pilha = ['#']

        for instrucao in cadeia:
            if instrucao == 'fim' and pilha[-1] == '#':
                return True
            if instrucao not in self.alfabeto:
                print("Instrução inválida")
                return False
            for (estado, simbolo_entrada, topo), transicoes in self.transicoes.items():
                if estado == estado_atual and simbolo_entrada == instrucao and (topo == pilha[-1] or topo == 'ε'):
                    for (proximo_estado, empilha) in transicoes:
                        proximo = proximo_estado

                        if topo != 'ε' and pilha[-1] != '#':
                            pilha.pop()
                        if empilha != 'ε':
                            pilha.append(empilha)
                estado_atual = proximo
        print("Fluxo recusado")
        return False
```

Os códigos acima utilizam transições pré configuradas em formato de dicionário em python, ou seja, dada uma chave que é o estado atual ele vai para um dado estado se receber um dado símbolo. O dicionário que representa as transições do AFD geral fica dessa forma:

```
transicoes_AFD = {
    # Possiveis estados iniciais
    'A': {'baixo': 'C1', 'estacionar': 'p1'},
    'B': {'baixo': 'C2', 'estacionar': 'p3'},
    'C': {'direita': 'C1'},
    'D': {'esquerda': 'C4'},

    # Retornos
    'R1': {'': ''},
    'R2': {'cima': 'C3'},
    'R3': {'cima': 'C4'},
    'R4': {'': ''},

    # Cruzamentos
    'C1': {'cima': 'A', 'estacionar': 'p1', 'direita': 'C2', 'baixo': 'C3'},
    'C2': {'cima': 'B', 'estacionar': 'p3', 'estacionar': 'p3.1', 'direita': 'R4', 'baixo': 'C4'},
    'C3': {'cima': 'C1', 'esquerda': 'R1', 'estacionar': 'p7', 'baixo': 'R2'},
```

```
    'C4': {'cima': 'C2', 'estacionar': 'p5', 'baixo': 'R3', 'esquerda': 'C3'}
}
```

Assim como o código do AP muda em relação ao AFD, o seu dicionário que representa o semáforo também segue uma lógica de configuração diferente:

```
transicoes_AP = {

    # Transições dos estados do semáforo
    ('vermelho', 'veiculo', 'ε'): [('vermelho', 'veiculo')],
    ('vermelho', 'troca', 'ε'): [('verde', 'ε')],
    ('verde', 'veiculo', 'veiculo'): [('verde', 'ε')],
    ('verde', 'troca', 'ε'): [('amarelo', 'ε')],
    ('amarelo', 'veiculo', 'veiculo'): [('amarelo', 'ε')],
    ('amarelo', 'troca', 'ε'): [('vermelho', 'ε')],

    # Transições para o estado final
    ('amarelo', 'ε', '?'): [('vermelho', '?')],
    ('vermelho', 'ε', '?'): [('vermelho', '?')],
    ('verde', 'ε', '?'): [('vermelho', '?')],
}
```

## 5.2 Implementação geral

Como dito anteriormente, será utilizado classes para cada objeto estudado no cenário original, por exemplo, um cruzamento será uma classe Cruzamento em python, que pode ou não receber uma classe Semáforo (este semáforo é o AP) como argumento na sua criação. Além da organização, modelar o sistema dessa forma irá facilitar para que, caso seja necessário, uma implementação mais complexa seja feita, usando o Paralelismo [2]. A seguir todas as classes usadas:

```
class Sistema:
    def __init__(self, entradas, retornos, estacionamentos, cruzamentos, alfabeto, transicoes):

        self.entradas = entradas
        self.retornos = retornos
        self.estacionamentos = estacionamentos
        self.cruzamentos = cruzamentos
        self.transicoes = transicoes
        self.alfabeto = alfabeto
…

class Cruzamento:
    def __init__(self, nome, semaforo = None):

        self.nome = nome
        self.semaforo = semaforo
…
```

```
class Semaforo:
    def __init__(self, estado_inicial, estados, alfabeto, alfabeto_pilha, estados_aceitacao, transicoes):
        self.estado_atual = estado_inicial
        self.estados = estados
        self.alfabeto = alfabeto
        self.alfabeto_pilha = alfabeto_pilha
        self.estados_aceitacao = estados_aceitacao
        self.transicoes = transicoes
```

## 5.3 Validação de trajetórias e fluxos

Agora com os modelos implementados, é de total importância testar diferentes trajetórias e fluxos que, dado a análise do cenário original proposto deveriam ou não ser aceitos. Para isso foi criado um menu para a main do programa que dá opções para esses testes:

```
def menu_principal():
    while True:
        print("\n=== SIMULAÇÃO DE TRÂNSITO ===")
        print("1. Testar trajetória")
        print("2. Testar fluxo")
        print("3 Sair")

        try:
            opcao = int(input("Escolha uma opção: "))
        except ValueError:
            print("Por favor, insira um número válido.")
            continue

        if opcao == 1:
            estado_inicial = input("Digite o estado inicial: ")
            entrada = input("Digite a trajetória: ")

            trajetoria = [item.strip() for item in entrada.split()]
            sistema.valida_trajeto(estado_inicial, trajetoria)

        elif opcao == 2:
            entrada = input("Digite o fluxo: ")

            fluxo = [item.strip() for item in entrada.split()]
            fluxo.append('fim')
            semaforo.valida_fluxo(fluxo)
        elif opcao == 3:
            print("Saindo do sistema. Até logo!")
            break
        else:
            print("Opção inválida. Tente novamente.")
```

### Trajetórias aceitas

Partindo do B: baixo baixo esquerda estacionar, baixo baixo estacionar, baixo direita estacionar
Partindo do A: estacionar, baixo direita cima estacionar, baixo baixo estacionar

Partindo do C: direita direita cima estacionar, direita direita estacionar, direita direita baixo estacionar

Partindo do D: esquerda estacionar, esquerda esquerda cima estacionar

### Trajetórias recusadas

Partindo do B: esquerda estacionar, baixo esquerda estacionar, baixo direita estacionar

Partindo do A: baixo, baixo direita cima, baixo baixo baixo estacionar

Partindo do C: direita direita direita cima estacionar, estacionar, direita direita baixo

Partindo do D: estacionar, esquerda esquerda esquerda estacionar

### Fluxos aceitos

veiculo veiculo troca veiculo veiculo, veiculo troca veiculo, veiculo veiculo troca troca veiculo veiculo troca

### Fluxos recusados

veiculo troca, veiculo, veiculo veiculo troca troca

# Referências

[1] https://docs.python.org/pt-br/3/tutorial/classes.html
[2] https://pt.wikipedia.org/wiki/Computa%C3%A7%C3%A3o_paralela